

Machine Learning in Geoscience

Applications of Deep Neural Networks in 4D Seismic Data Analysis

Jesper Sören Dramsch

Kongens Lyngby 2021



DTU Physics
Department of Physics
Technical University of Denmark

Fysikvej
Building 311
2800 Kgs. Lyngby, Lyngby
info@fysik.dtu.dk
Tel.: +45 4525 3344
<https://www.fysik.dtu.dk>

Abstract

Machine Learning provides a tool for the modelling and analysis of geoscientific data. I have placed recent developments in deep learning into the greater context of machine learning by reviewing the approaches and challenges of the use of machine learning in geoscience. The thesis consists of six peer-reviewed publications and one submitted journal paper. Furthermore, five peer-reviewed publications are placed in the appendix.

The aim of this thesis is to apply recent developments in computer vision systems, neural networks, and machine learning to geoscientific data, particularly 4D seismic analysis. Neural networks are a type of machine learning that has made significant contributions to modern artificial intelligence and automation. The applicability of neural networks for their capability of being a universal function approximator was recognized within geophysics from an early stage. Following the recent interest in deep learning, neural networks have experienced a renaissance in geoscience applications, particularly in automatic seismic interpretation, inversion processes and sequence modelling.

This is followed by an exploration of unsupervised machine learning to segment chalk sediments in back-scatter scanning electron microscopy data. The next chapter shows that using neural networks pre-trained on natural images can reduce the data necessary for transfer learning to geoscience problems. This is followed by a chapter showing that complex-valued convolutions can stabilize training and data compression on non-stationary physical data. Subsequently, pressure-saturation data is extracted from 4D seismic amplitude difference maps using a novel deep dense sample-based encoder-decoder network. The network contains a low-assumption physical basis (Amplitude Versus Offset) as explicit features and learns the residual for the regression of the "inversion" data. This work shows that transfer from simulation data to field data is possible.

Finally, an unsupervised method is devised to extract 3D time-shifts from two 4D seismic cubes. The network extracts these 3D time-shifts including uncertainty measures. Commonly, time-shifts are extracted in 1D, due to processing speed, computational cost and poor performance of 3D methods. Within the training loop, the stationary velocity field is numerically integrated to obtain 3D time shifts that are constrained by the topology in a geologically consistent manner. The unsupervised implementation of the network structure ensures that biases from other time-shift extraction methods are not implicitly included in the network. This application utilizes unsupervised learning by devising a way of behaviour for the network to follow instead of supplying ground truth labels. Moreover, this results in a way to increase trust in the system, by limiting the extraction process to the deep learning system and performing well-defined operations within the network to automate the unsupervised training.

Dansk Resumé

Maskinlæring ('machine learning') er et redskab til modellering og analyse af geovidenskabelige data. Jeg har sat den seneste udvikling inden for dyb læring ('deep learning') ind i en større sammenhæng indenfor maskinlæring ved at gennemlæse de tilgange og udfordringer som maskinlæring har inden for geovidenskab. Afhandlingen består af seks peer-reviewed udgivelser og en indsendt journalartikel. Yderligere er der fem peer-reviewed udgivelser i appendix.

Formålet med denne afhandling er, at anvende den seneste udvikling inden for systemer for computer vision, neurale netværk og maskinlæring for geovidenskabelige data, især 4D seismisk analyse. Neurale netværk er en type maskinlæring, der har bidraget betydeligt til moderne kunstig intelligens og automatisering. Det blev på et tidligt tidspunkt anerkendt inden for geofysik, at neurale netværk var anvendelige. Brugen af neurale netværk for deres evne til at være universelle funktions-approksimatorer blev tidligt anderkendt inden for geofysik. Grundet den nylige interesse for dyb læring, har neurale netværk oplevet en renæssance inden for geovidenskabelige anvendelser, særligt automatisk seismisk fortolkning, inverteringsprocessor og sekvensmodellering.

Dette efterfølges af en udforskning af uovervåget læring til segmentring af kalksedimenter i tilbagesprednings-elektronmikroskopi "back-scatter scanning electron microscopy" data. Det næste kapitel viser, at brugen af neurale netværk prætrænede på billeder, kan reducere den nødvendige mængde data, der er nødvendige for at overføre læring til geovidenskabelige problemer. Kapitlet derefter viser, at foldninger med komplekse tal kan stabilisere træningen og datakompressionen af ikke-stationære fysiske data. Derpå beregnes tryk og mætningsdata med brug af 4D seismiske data ved hjælp af et nyt dybt tæt prøvebaseret indkoder-dekoder netværk. Netværket indeholder et fysisk grundlag, for selv at lære resten af inversionsprocessen. Arbejdet viser overførsel fra simulerede til rigtige data er muligt.

Endelig blev der udviklet en uovervåget 'unsupervised' metode, til at udregne 3D-tidsforskydninger fra to 4D seismiske kuber. Netværket beregner disse 3D tidsskift inklusiv usikkerhedsmålinger på dem. På grund af de beregningsmæssige omkostninger og dårlig kvalitet, bliver disse normalt kun beregnet i 1D. Inden for træningsløkken integreres det stationære hastighedsfelt numerisk for at få 3D tidsskift, som er begrænset af topologien på en geologisk konsistent måde. Den uovervågende implementation af netværksstrukturen sikrer at bias fra andre tidsforskydnings ekstraktionsmetoder ikke implicit indgår i netværket. Den uovervågende metode lærer netværket at følge en bestemt opførsel uden brug af sande "ground truth" eksempler. Yderligere, styrker dette tilliden til systemet, da ekstraktionsmetoden begrænses til det dybe læringssystem og veldefinerede operationer inden for dette som automatiserer den uovervågede træning.

Preface

This dissertation is presented by
Jesper Sören Dramsch
to the
Department of Physics
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy (Ph.D.)
at the
Technical University of Denmark

Kongens Lyngby, October 31st, 2020



Jesper Sören Dramsch

Ph.D. Thesis

University: Technical University of Denmark (DTU)

Department: Department of Physics / Centre for Oil and Gas – DTU

Author: Jesper Søren Dramsch

Title: Machine Learning in Geoscience

Applications of Deep Neural Networks in 4D Seismic Data Analysis

Principal Advisor: Mikael Luthje

Co-Advisor: Anders Nymark Christensen (DTU Compute)

External Advisor: Colin MacBeth (Heriot-Watt University, UK)

Submitted: 2019-11-14

Revised Submission: 2020-10-31

Acknowledgements

This thesis would not exist, without the support of my peers.

My sincere gratitude to my supervisor Mikael L  thje for his continuous support, supervision and guidance through the perilous cliffs of pursuing a Ph.D. in a new and ever-evolving academic center. The open discussions and trust in my ability allowed me to thrive during this period. My appreciation extends to Colin MacBeth, who in the position as my external co-supervisor welcomed me to Edinburgh and provided further guidance and insight into the practical workings of 4D seismics. I am deeply indebted to my internal co-supervisor Anders Nymark Christensen, who provided valuable insight into the statistical working of machine learning and kept my weights and biases in check. Thank you for inspiring me to achieve more than I ever thought possible.

To the friends we had and made along the way! Kirstie Wright and Anna Clark you kept me sane from day to day and made Scotland feel home. Thank you. Robert Leckenby, Tim Albrecht, Bettina Schmidt, Matthias Schneider, Manuela K  llner, Clara Dabrock, Brian Burnham and Florian Smit, you were always there and I appreciate you for it. Marie-Daphne Mangriotis thank you for welcoming me to ETLP and the great conversations. Furthermore, I would like to thank Antony Hallam and Gustavo Corte for great discussion and peership.

My thanks go out to the Software Underground community, for keeping the spirit of sharing and collaboration. Especially, Matt Hall for the leadership and trust. To Lukas Mosser, for always encouraging and inspiring me to strive for more.

I would like to thank my colleagues at the DHRTC, particularly Tala Maria Aab   for being a fantastic co-conspirator in the early days and Florian Smit for welcoming me into this new environment. Furthermore, I'd like to thank Fr  d  ric Amour, Charlotte Lind Laurentzius, Anne Lysgaard and Helle Baumann for being a pleasure to work with.

I want to thank part of the Open Source community and in particular the scientific Python community, without these tools this thesis would be much less substantial.

*Difficulties strengthen the mind,
as labor does the body.*

SENECA THE YOUNGER

Disclaimer

This thesis reprints several published scientific articles and converts them to the format of this thesis. It was attempted to reprint these in their original type-setting, but deemed unacceptable by the evaluation of the examining committee.

The copyright of these articles belongs to their respective owners and are reprinted under the Fair Use policy in fulfillment of this academic thesis. The publications are outlined as clearly as possible within the means of this thesis.

No copyright infringement intended.

Publication List

Book Chapters

Dramsch, J. S. (Sept. 2020a). “70 years of machine learning in geoscience in review”. In: *Advances in Geophysics*. Ed. by B. Moseley and L. Krischer. Published, Peer-Reviewed, Section 2.2. Academic Press. Chap. 4. ISBN: 9780128216699.

Journal Articles

Dramsch, J. S., A. N. Christensen, C. MacBeth, and M. Luthje (2019b). “Deep Unsupervised 4D Seismic 3D Time-Shift Estimation with Convolutional Neural Networks”. In: *IEEE Transactions in Geoscience and Remote Sensing*. In Review, Chapter 7.

Dramsch, J. S., M. Luthje, and A. N. Christensen (2019h). “Complex-valued neural networks for machine learning on non-stationary physical data”. In: *Computers & Geoscience*. Accepted, Chapter 5.

Peer-Reviewed Conference Proceedings

Dramsch, J. S. and M. Luthje (2018d). “Deep-learning seismic facies on state-of-the-art CNN architectures”. In: *SEG Technical Program Expanded Abstracts 2018*. Published, Chapter 4. Society of Exploration Geophysicists, pp. 2036–2040. DOI: 10.1190/segam2018-2996783.1. URL: <https://doi.org/10.1190/segam2018-2996783.1>.

Peer-Reviewed Workshop Proceedings

Dramsch, J. S., G. Corte, H. Amini, M. Luthje, and C. MacBeth (2019d). “Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data”. In: *Second EAGE Workshop Practical Reservoir Monitoring 2019*. Published, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201900028.

- Dramsch, J. S.**, G. Corte, H. Amini, C. MacBeth, and M. L  thje (2019g). “Including Physics in Deep Learning – An Example from 4D Seismic Pressure Saturation Inversion”. In: *81st EAGE Conference and Exhibition 2019 Workshop Programme. Published*, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201901967. URL: <https://doi.org/10.3997/2214-4609.201901967>.
- Dramsch, J. S.**, F. Amour, and M. L  thje (2018a). “Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk”. In: *First EAGE/PESGB Workshop Machine Learning. Published*, Chapter 3. EAGE. DOI: 10.3997/2214-4609.201803014. URL: <https://doi.org/10.3997/2214-4609.201803014>.

Publications Not Included

Journal Articles

- Aabø, T. M., **J. S. Dramsch**, C. L. Würtzen, S. Seyum, F. Amour, M. Welch, and M. Lühje (2020). “An integrated workflow for fracture characterization in chalk reservoirs, applied to the Kraka Field”. In: *Marine and Petroleum Geology* 112. Published, Appendix B. ISSN: 0264-8172. DOI: <https://doi.org/10.1016/j.marpetgeo.2019.104065>. URL: <http://www.sciencedirect.com/science/article/pii/S026481721930501X>.
- Côrte, G., **J. S. Dramsch**, H. Amini, and C. MacBeth (2020). “Deep neural network application for 4D seismic inversion to changes in pressure and saturation: Optimizing the use of synthetic training datasets”. In: *Geophysical Prospecting* 68.7. Published, Appendix B.2, pp. 2164–2185.

Peer-Reviewed Conference Proceedings

- Mosser, L., W. Kimman, **J. S. Dramsch**, S. Purves, A. De la Fuente Briceño, and G. Ganssle (June 2018a). “Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks”. In: *80th EAGE Conference and Exhibition 2018*. Published, Appendix C. EAGE. DOI: 10.3997/2214-4609.201800734. URL: <https://doi.org/10.3997/2214-4609.201800734>.
- Aabø, T. M., **J. S. Dramsch**, M. Welch, and M. Lühje (June 2017a). “Correlation of Fractures From Core, Borehole Images and Seismic Data in a Chalk Reservoir in the Danish North Sea”. In: *79th EAGE Conference and Exhibition 2017*. Published, Appendix C.2. EAGE. DOI: 10.3997/2214-4609.201701283. URL: <https://doi.org/10.3997/2214-4609.201701283>.

Peer-Reviewed Workshop Proceedings

- Dramsch, J. S.** and M. Lühje (2018e). “Information Theory Considerations In Patch-Based Training Of Deep Neural Networks On Seismic Time-Series”. In: *First EAGE/PESGB Workshop Machine Learning*. Published, Appendix D. EAGE. DOI: 10.3997/2214-4609.201803020. URL: <https://doi.org/10.3997/2214-4609.201803020>.

Open Source Software List

Open Source Packages

Dramsch, J. S. and Contributors (2019c). *Complex-Valued Neural Networks in Keras with Tensorflow*. Open-Source Software. DOI: 10.6084/m9.figshare.9783773. URL: <https://github.com/JesperDramsch/keras-complex>.

Reproducible Code

Dramsch, J. S. (2019c). *Reproducible Code: Complex-valued neural networks for machine learning on non-stationary physical data*. URL: <https://github.com/JesperDramsch/Complex-CNN-Seismic>.

Dramsch, J. S. (2019d). *Reproducible Code: Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data*. URL: <https://github.com/JesperDramsch/4D-seismic-neural-inversion>.

Dramsch, J. S. (2019e). *Reproducible Code: Deep Unsupervised 4D Seismic 3D Time-Shift Estimation with Convolutional Neural Networks*. URL: <https://github.com/JesperDramsch/voxelmorph-seismic>.

Dramsch, J. S. (Dec. 15, 2018e). *Reproducible Code: Deep-learning seismic facies on state-of-the-art CNN architectures*. DOI: 10.6084/m9.figshare.7227545. URL: <https://github.com/JesperDramsch/seismic-transfer-learning>.

Dramsch, J. S. (2018f). *Reproducible Code: Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk*. URL: <https://github.com/JesperDramsch/backscatter-sem-segmentation>.

Dramsch, J. S. (2018g). *Reproducible Code: Information Theory Considerations in Patch-based Training of Deep Neural Networks on Seismic Time-Series*. URL: <https://github.com/JesperDramsch/windowing-seismic-for-deep-learning>.

Commit Contributions to Free Open Source Software

Open Geoscience Awesome List

Contribution: Creator and Maintainer of List. Implemented guidelines and automated testing. [Link]

Full Citation: J. S. Dramschi and Contributors (2018b). *Awesome Open Geoscience*. Maintainer. URL: <https://github.com/softwareunderground/awesome-open-geoscience>

Bruges

Contribution: Created documentation, fixed docstrings and exposed functional API. [Link]

Full Citation: *Bruges: Bag of really useful geophysical equations and stuff* (2016). URL: <https://github.com/agile-geoscience/bruges>

Welly

Contribution: Created documentation, exposed functional API. [Link]

Full Citation: *Welly: Manage subsurface well data* (2015). URL: <https://github.com/agile-geoscience/welly>

Tensorflow

Contribution: Contribution to documentation of `tf.keras.Conv1D` and `tf.keras.Lambda` [Link].

Full Citation: Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine*

Learning on Heterogeneous Systems. Software available from tensorflow.org. URL: <http://tensorflow.org/>

Scikit-Learn

Contribution: Created examples for documentation of `sklearn.model_selection.GroupShuffleSplit`, and `sklearn.ensemble.BaggingClassifier`. [Link]

Full Citation: F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830

Pandas

Contribution: Wrote documentation and examples for `pd.str.slice`, `pd.str.pad`, and `pd.str.repeat`. [Link]

Full Citation: W. McKinney et al. (2010). “Data structures for statistical computing in python”. In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX, pp. 51–56

Presentation List

Conference Presentation

- Dramsch, J. S.** and M. L  thje (2018d). “Deep-learning seismic facies on state-of-the-art CNN architectures”. In: *SEG Technical Program Expanded Abstracts 2018*. Published, Chapter 4. Society of Exploration Geophysicists, pp. 2036–2040. DOI: 10.1190/segam2018-2996783.1. URL: <https://doi.org/10.1190/segam2018-2996783.1>.
- Mosser, L., W. Kimman, **J. S. Dramsch**, S. Purves, A. De la Fuente Brice  o, and G. Ganssle (June 2018a). “Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks”. In: *80th EAGE Conference and Exhibition 2018*. Published, Appendix C. EAGE. DOI: 10.3997/2214-4609.201800734. URL: <https://doi.org/10.3997/2214-4609.201800734>.

Workshop Presentation

- Dramsch, J. S.**, G. Corte, H. Amini, M. L  thje, and C. MacBeth (2019d). “Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data”. In: *Second EAGE Workshop Practical Reservoir Monitoring 2019*. Published, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201900028.
- Dramsch, J. S.**, G. Corte, H. Amini, C. MacBeth, and M. L  thje (2019g). “Including Physics in Deep Learning – An Example from 4D Seismic Pressure Saturation Inversion”. In: *81st EAGE Conference and Exhibition 2019 Workshop Programme*. Published, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201901967. URL: <https://doi.org/10.3997/2214-4609.201901967>.
- Dramsch, J. S.**, F. Amour, and M. L  thje (2018a). “Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk”. In: *First EAGE/PESGB Workshop Machine Learning*. Published, Chapter 3. EAGE. DOI: 10.3997/2214-4609.201803014. URL: <https://doi.org/10.3997/2214-4609.201803014>.

Workshop Poster

- Dramsch, J. S.** and M. L  thje (2018e). “Information Theory Considerations In Patch-Based Training Of Deep Neural Networks On Seismic Time-Series”. In: *First EAGE/PESGB*

Workshop Machine Learning. Published, Appendix D. EAGE. DOI: 10.3997/2214-4609.201803020. URL: <https://doi.org/10.3997/2214-4609.201803020>.

Other Presentations

Dramsch, J. S., G. Corte, H. Amini, M. L  thje, and C. MacBeth (May 2019f). *Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data*. ETLP Sponsor Meeting 2019.

Dramsch, J. S. (2018c). *KFold in Deep Learning*. Lightning Talk – EuroScipy 2018. DOI: 10.6084/m9.figshare.7035908.

Other Posters

Dramsch, J. S., A. N. Christensen, and M. L  thje (June 2019a). *Physics and Deep Learning - Incorporating prior knowledge in deep neural networks*. DOI: 10.6084/m9.figshare.8217518.v1.

Dramsch, J. S. and M. L  thje (2018c). *Deep Learning: From Cats to 4D Seismic - Reducing cycle time and model training cost in asset management*. Tech. rep. Danish Hydrocarbon Research and Technology Centre. DOI: 10.6084/m9.figshare.7422629.

Dramsch, J. S. (2017b). *Edge detection in 3D seismic data*. DHRTC PhD Day.

Invited Presentation

Dramsch, J. S. (Nov. 2019a). *Cracking Open the Black Box – Making sense of Machine Learning and Neural Networks*. EAGE E-Lecture. URL: <https://www.youtube.com/watch?v=5o0XTfUZQm0>.

Dramsch, J. S. (June 2019b). *Making sense of AI for a career in a changing industry*. EAGE Annual Meeting 2019.

Dramsch, J. S. (Sept. 2018a). *4D Seismics in Fracture Characterization – A machine learning perspective*. Company Talk - ConocoPhillips.

Dramsch, J. S. (Oct. 2018b). *A practitioner’s guide to deep learning in geophysical imaging*. FORCE Velocity Modeling Meeting. DOI: 10.6084/m9.figshare.7170299.

Dramsch, J. S. (July 2018d). *Machine Learning Workshop*. Heriot-Watt University, ETLP.

MacBeth, C., R. Chassagne, and **J. S. Dramsch** (Nov. 2018). *A guided discussion on machine learning for 4D QI*. ETLP Sponsor Meeting 2018.

Dramsch, J. S. (May 2017a). *Edge detection in 3D seismic*. DTU Vision Day.

Contents

Abstract	i
Dansk Resumé	ii
Preface	iii
Acknowledgements	v
Disclaimer	vii
Publication List	ix
Book Chapters	ix
Journal Articles	ix
Peer-Reviewed Conference Proceedings	ix
Peer-Reviewed Workshop Proceedings	ix
Publications Not Included	xi
Journal Articles	xi
Peer-Reviewed Conference Proceedings	xi
Peer-Reviewed Workshop Proceedings	xi
Open Source Software List	xii
Open Source Packages	xii
Reproducible Code	xii
Presentation List	xv
Conference Presentation	xv
Workshop Presentation	xv
Workshop Poster	xv
Other Presentations	xvi
Other Posters	xvi
Invited Presentation	xvi
Contents	xvii
1 Introduction	1

2	Methods & Theory	5
2.1	4D seismic	5
2.2	Machine Learning in Geoscience	8
2.3	Contributions of this Study	44
3	Unsupervised Geological Image Segmentation	45
3.1	Unsupervised Image Segmentation	45
3.2	Workshop Paper: Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk .	49
3.3	Computational Granulometry	52
3.4	Contributions of this Study	53
4	Transfer learning in Automatic Seismic Interpretation	55
4.1	Training and Fine-Tuning	58
4.2	Conference Paper: Deep learning seismic facies on state of the art CNN architectures	61
4.3	Applications of Transfer Learning for Automatic Seismic Interpretation .	69
4.4	Contributions of this Study	70
5	Complex-valued neural networks	71
5.1	Journal Paper: Complex-valued neural networks for machine learning on non-stationary physical data	72
5.2	Contributions of this Study	91
6	Machine Learning in 4D Seismic Inversion	93
6.1	Data	93
6.2	Machine Learning Model	94
6.3	Training the Deep Neural Network for 4D Seismic Inversion	95
6.4	Workshop Paper: Including Physics in Deep Learning – An example from 4D seismic pressure saturation inversion	97
6.5	Workshop Paper: Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data	102
6.6	Discussion of 4D Inversion	108
6.7	Contribution of this study	108
7	3D Time Warping for 4D Data	109
7.1	Diffeomorphisms	110
7.2	Image Matching Algorithms	110
7.3	Dynamic Time and Image Warping	112
7.4	Journal Paper: Deep Unsupervised 4D Seismic 3D Time-Shift Estimation with Convolutional Neural Networks	115
7.5	Contributions of This Study	137
8	Conclusion of this Thesis	139

A ImageNet Results	141
B Journal Papers	143
B.1 An Integrated Approach to Fracture Characterization of the Kraka Field	143
B.2 Deep neural network application for 4D seismic inversion to changes in pressure and saturation: Optimizing the use of synthetic training datasets	156
C Conference Papers	179
C.1 Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks	179
C.2 Correlation of Fractures From Core, Borehole Images and Seismic Data in a Chalk Reservoir in the Danish North Sea	184
D Workshop Papers	189
D.1 Information Theory Considerations in Patch-based Training of Deep Neural Networks on Seismic Time-Series	189
E Reproducible Code	193
E.1 Unsupervised Geological Image Segmentation	193
E.2 Transfer learning in Automatic Seismic Interpretation	198
E.3 Complex-valued neural networks	214
E.4 Machine Learning in 4D Seismic Inversion	223
E.5 Software Manual: Keras Complex	250
Acronyms	269
Bibliography	271

CHAPTER 1

Introduction

This thesis explores machine learning in geoscience with a special focus on deep learning in 4D seismics. Recently, machine learning and neural networks in particular have made essential impacts in many scientific disciplines, with geoscience exploring these new approaches as well. This study contributes to this body of emerging work in deep neural networks and computer vision systems for the modelling and analysis of geoscientific data. The main contribution being a physics-based neural architecture for pressure-saturation inversion and a novel algorithm for 3D timeshift extraction in 4D seismic.

The growing interest in machine learning sometimes overlooks the fact that the underlying idea of Machine Learning was introduced in 1950. Section 2.2 reviews the history of Machine Learning with a special focus on geoscience. Geoscience and in particular geophysics has followed the innovation in artificial intelligence and especially neural networks closely. Early applications of neural networks include seismic processing and seismic inversion. Moreover, Gaussian Processes were early introduced in geostatistics as kriging, which have gained interest in a wider Machine Learning context as Gaussian Process. Recently, Deep Learning becoming popular and particularly breakthroughs in computer vision have sparked interest in applying machine learning computer vision to Automatic Seismic Interpretation in the hopes for increased accuracy, reproducibility and automation.

In recent years, 4D seismic itself has made an impact in geophysical reservoir analysis and other geophysical areas. The method enables imaging of changes in the subsurface. This is essential in hydrocarbon production, enabling extended production, reducing the direct environmental footprint and ensuring resource safety. Moreover, it enables CO₂ sequestration monitoring for reservoir and seal integrity and has applications in nuclear test treaty compliance, waste storage, and deep geothermal monitoring. 4D seismic matching has exposed deficits in 3D seismic processing, therefore furthered our understanding of amplitude-preserving and surface-consistent processing steps. Additionally, furthering our understanding of in-situ validation of geomechanical concepts and update of heterogeneous subsurface models.

The structure of this study is composed of topical groupings of five peer-reviewed and two submitted publications into chapters. Each chapter will provide an individual introduction to the topic and outline relevant theoretical and methodological aspects, where the publication falls short. This is particularly relevant for the shorter workshop and conference papers.

Chapter 2 provides a theoretical introduction into 4D seismic principles, followed by a thorough overview of the development of machine learning with a special focus on geoscience. This chapter focuses particularly on the development of Machine Learning

applications in geoscience through history. The main contribution in this chapter is a peer-reviewed book chapter “70 years of machine learning in geoscience in review” published in *Advances in Geophysics* (Dramsch, 2020d).

Chapter 3 contains a workshop paper (Dramsch et al., 2018a), which explores the application of unsupervised learning to the segmentation of chalk grains in Backscatter Scanning-Electron Microscopy images. The chapter expands on the method and provides a theoretical treatment of the methods applied in the short paper. The method is also compared to classical image processing techniques. Then an overview of additional computational granulometry based on the segmentation maps is presented to apply the work and close out the chapter.

Chapter 4 discusses a conference paper contribution to Automatic Seismic Interpretation using Deep Learning (Dramsch et al., 2018c). The paper uses transfer learning of Neural Networks pre-trained on natural image data sets to fine-tune the network to perform Automatic Seismic Interpretation on seismic data. The chapter expands on the data and training of the Neural Network. The chapter then expands on the applications that resulted from the paper, using the composition of Neural Networks into more adequate architectures for a task that is called semantic segmentation, which more closely resembles Automatic Seismic Interpretation.

Chapter 5 covers a journal paper on the application of complex-valued Convolutional Neural Networks to seismic data (Dramsch et al., 2019f). These networks perform a complex convolution in the Neural Network layers. The paper tests the hypothesis that providing phase information explicitly can improve the capacity of the Convolutional Neural Network, which is tested on an AutoEncoder architecture, which lossily compresses the data at different rates and measures the reconstruction error. The phase information is derived directly from the seismic data via a Hilbert transform, hence a Deep Neural Network could, in theory, extract this information automatically. For this chapter, networks at varying compression were trained for both real-valued and complex-valued networks to perform an adequate comparison.

Chapter 6 consists of two workshop papers which introduce a Deep Neural Network architecture for 4D quantitative pressure-saturation inversion (Dramsch et al., 2019e; Dramsch et al., 2019d). The Deep Neural Network (DNN) regression model implements a layer that computes basic physical knowledge within the network architecture to stabilize the network. The physical knowledge encoded in the layer is the Amplitude versus Offset (AVO) gradient between the input seismic data. This data is passed into a Variational AutoEncoder architecture. In this work, we show that this network can be trained on simulation data and transferred to field data by applying Gaussian noise to the noise-free simulation input data to condition the network to accept noisy inputs from field data.

Chapter 7 is comprised of a re-submitted journal paper and introduces a robust method for 3D time shift extraction in 4D data (Dramsch et al., 2019b). Time shifts in 4D data are commonly extracted in 1D due to computational cost and often poor performance of 3D methods. This method uses a self-supervised deep learning system to extract the timeshift mapping of two seismic volumes without supplying a-priori timeshift data. Moreover, the method limits the neural network to the extraction of

the stationary timeshift but leaves the matching to a non-learning 3D interpolation to increase the transparency of the method. Additionally, the method supplies uncertainty values for the warp velocity. Constraining the possible 3D time shifts is vital to ensure sensible results for the time shifts, as well as, the aligned monitor seismic. This is ensured by implementing a geologically intuitive constraint on the 3D timeshifts, which prohibits crossing or looping of reflectors after mapping the seismic volumes. This learning-based method can be trained in advance, providing fast 3D results on previously unseen data, which is essential in 4D seismic analysis.

Finally, Chapter 8 is the conclusion of this thesis recapitulating the contributions and findings of the papers and scientific work. The contributions span multiple geoscientific disciplines with a focus in geophysics and particularly 4D seismic unified by Machine Learning.

CHAPTER 2

Methods & Theory

This thesis applies Machine Learning methods to 4D seismic data. In this chapter I introduce 4D seismic concepts and the motivation to acquire and analyze 4D seismic data. I go on to introduce machine learning and review the development of machine learning in itself and in the field of geoscience. The focus on this thesis is on Neural Networks, particularly Deep Learning to geophysical problems. Considering recent developments in computer vision, a focus on Convolutional Neural Networks, the developments and break-throughs of this type of Neural Network (NN) and the innovations that lead to the recent adoption of Machine Learning in geoscience are explored in a published book chapter, reprinted here.

2.1 4D seismic

4D seismic is the analysis of seismic data that was acquired over the same location after some calendar time has passed. The repeated imaging of the same subsurface location highlights changes in the subsurface that can lead to improved understanding of subsurface processes and fluid movement. Exploration & Production companies, in particular, have an interest in imaging hydrocarbon reservoirs (Johnston, 2013b). However, 4D seismic imaging has broad applications for subsurface characterization, such as observing volcanic activity (Londoño et al., 2018) or CO₂ sequestration monitoring (Arts et al., 2004).

The main applications of 4D seismic analysis, according to Yilmaz (2003) and Johnston (2013a) include:

- Tracking fluid movement (steam, gas, and water)
- Monitoring pressure depletion and validating depletion plans
- Fault property estimation, i.e. sealing or leaking faults
- Locating bypassed oil in heterogeneous reservoirs
- Validating and updating geological and reservoir-simulation models

4D seismic data analysis suffers from the superposition of multiple effects on seismic imaging. These effects include changes in the acquisition equipment due to technological advances, changes in acquisition geometry (source-receiver mismatch), as well as physical

changes in the subsurface (Yilmaz, 2003; Johnston, 2013b). These physical changes are in part due to fluid movement in the subsurface (Lumley, 1995), as well as, geological changes due to compaction and expansion (Hatchell et al., 2005a). These geomechanical effects change the position of the reflectors, the thickness of stratigraphy, and the physical properties such as density and wave velocity (Herwanger, 2015).

Successful 4D applications rely on careful acquisition planning, closely matching the mismatch of the source (ΔS) and receiver (ΔR). This awareness has generally improved the repeatability of seismic acquisition; however, the Normalized Root Mean Squared Error (NRMS) remains to be an essential measure of noise sources that deteriorate the 4D seismic analysis. Moreover, 4D seismic analysis has brought to light that some 3D seismic processing workflows are not as repeatable and amplitude-preserving as they were thought to be (Lumley, 2001). Modern processing flows include co-processing of the base and monitor seismic volumes with specialized tools to reduce differences from processing (Johnston, 2013a).

The standard analysis tool in 4D seismic interpretation is amplitude differences (Johnston, 2013b). Differences can among others stem from fluid movement or replacement, i.e. oil, gas, or brine, as well as, changes in the rock matrix due to compaction, temperature changes, and movement of injected CO₂ plumes. Usually, a simple difference of the 3D seismic volumes will not yield satisfactory results due to small-scale fluctuations in both arrival times and amplitudes, making time-shift analysis a vital process to match the reflection events. These time-shift values are a valuable source of information themselves (Hall et al., 2002; Hatchell et al., 2005c), considering their sole dependence on wavefield kinematics, time shifts tend to be a more robust measurement than amplitude differences (Johnston, 2013b).

Considering normal incidence on a horizontal layer of thickness z and a P-wave velocity v with a traveltime t , we can express the changes in traveltime as:

$$\frac{\Delta t}{t} = \frac{\Delta z}{z} - \frac{\Delta v}{v}, \quad (2.1)$$

for homogeneous isotropic v and small changes in z and v . Originally developed in Hatchell et al. (2005c), with a rigorous integral derivation presented in MacBeth et al. (2019).

The vertical strain $\frac{\Delta z}{z}$ directly relates to the geomechanical strain ξ_{zz} , describing the vertical strain on the vertical surface of an infinitesimal element (Herwanger, 2015). Independently Hatchell et al. (2005c) and R  ste et al. (2006) developed a single-parameter solution to relate velocity changes and vertical strain

$$\frac{\Delta v}{v} = -R\xi_{zz} \quad (2.2)$$

with R being the single parameter Hatchell-Bourne-R  ste (HBR)-factor (Hatchell et al., 2005a; MacBeth et al., 2019). The HBR being a lithological constant, we can relate (2.2) and (2.1) and obtain a direct relationship between the vertical strain ξ_{zz} and the time shift Δt for a given lithology with property R

$$\Delta t = t \cdot (1 + R) \cdot \xi_{zz}. \quad (2.3)$$

Contingent on the assumption of zero-offset incidence, homogeneous velocity and isotropy, time-shift extraction is mostly performed in the z-direction by comparing traces directly. Prominently, the 1D windowed cross-correlation is used due to its computational speed and general lack of limiting underlying assumptions (Rickett et al., 2001). The main drawback of this method is, however, that the result is highly dependent on the window-size and susceptible to noise. Other methods for post-stack seismic time shift extraction include Dynamic Time Warping (DTW) (Hale, 2013b) and inversion-based approaches (Rickett et al., 2007b).

More recently, research into pre-stack time shift extraction and 3D-based methods has been conducted. These methods relax the constraints of some assumptions of 1D applications (Ghaderi et al., 2005; Hall et al., 2002). 3D time shifts can capture the subsurface movement of reflectors and account for 3D effects of the $\Delta R/\Delta S$ acquisition mismatch, which effect seismic illumination.

Quantitative Interpretation (QI) extends the interpretation of 4D changes to estimate fluid saturation and pressure changes within the reservoir. The subsurface changes recorded by the seismic data can be related numerically to subsurface changes. Taran-tola (2005) defines the scientific procedure for the study of a physical system loosely as a three-step process involving parameterization of the system, forward modelling, and inverse modelling, additionally defining the inversion as the only deductive method. The inversion process is usually non-unique, where multiple causal processes could explain an observation. Therefore prior information can often be beneficial in applications such as Bayesian inversion. In 4D seismic data particularly, decoupling of pressure and saturation changes is non-trivial and relies on pre-stack or angle-stack information (Landrø, 2001). This process is, however, highly desirable with the benefit of quantifying the subsurface changes from observed seismic data directly.

Active areas of research in 4D seismic are the use of 4D seismic data to estimate saturation and pressure changes quantitatively, however, these approaches often depend on reliable rock-physics models. Moreover, volumetric time-shift estimation as opposed to trace-wise time-shift extraction, is particularly beneficial to quantitative pre-stack seismic analysis. Additional research in extractive data-based methods and model-based approaches investigate how much information is available directly from the data and what information is available from the modelling feedback-loop.

Lately there has been an increased focus on using machine learning in 4D seismic and the wider field of geoscience. Many 4D seismic approaches depend on statistical methods, one example being time-shift extraction by trace-wise windowed cross-correlation, which lends itself to machine learning applications. The next chapter will give a detailed review of the use of machine learning in geoscience, while introducing important theoretical concepts for this thesis.

2.2 Machine Learning in Geoscience

In Section 2.2 a published peer-reviewed book chapter “70 years of machine learning in geoscience in review” provides a treatment of the history and recent advancements and developments of machine learning in geoscience (Dramsch, 2020d). The book chapter covers the historical development of machine learning with a focus on co-developments with geoscience and provides the theoretical background for this thesis. Section 2.2.1.2 specifically gives a treatment of Neural Networks, the main driver of modern Machine Learning applications. Section 2.2.2.4 goes on to discuss the development of Deep Learning, with Section 2.2.2.6 going into detail about Convolutional Neural Network architectures that are particularly relevant to both this thesis and the wider field of Machine Learning in geoscience.

Essential machine learning concepts that are used throughout this thesis will be introduced. This includes DNNs and Convolutional Neural Networks (CNNs), as well as, common natural image benchmarks, i.e. ImageNet. Moreover, the Convolutional Neural Networks architectures VGG-16 and ResNet are discussed, which are used in Chapter 4. This chapter goes on to discuss the U-Net architecture, which is at the core of the Voxelmorph algorithm discussed in Chapter 7. Moreover, Section 2.2 discusses composition of Neural Networks as applied to geoscience.

In addition Support Vector Machines, kriging and Gaussian Processes, and Random Forests are discussed as they are important Machine Learning models used in geoscience detailed in their respective sections. Gaussian Processes in particular have a rich history in geoscience, originating in geostatistics, having reached the wider Machine Learning community. These methods are particularly suitable for problems on smaller datasets, where Neural Networks would overfit on the dataset and not generalize to unseen data.

The review shows the use of modern Machine Learning software applications and discusses the necessity of thorough model validation. The machine learning applications in this thesis split the labelled data into subsets that are used for training and validation. This serves as a basic test of generalization of the individual Machine Learning model to unseen data.

Book Chapter: 70 years of machine learning in geoscience in review

Published as a peer-reviewed book chapter in *Advances in Geophysics*, Vol. 61, Machine Learning and Artificial Intelligence in Geosciences

Abstract: This review gives an overview of the development of machine learning in geoscience. A thorough analysis of the co-developments of machine learning applications throughout the last 70 years relates the recent enthusiasm for machine learning to developments in geoscience. I explore the shift of kriging towards a mainstream machine learning method and the historic application of neural networks in geoscience, following the general trend of machine learning enthusiasm through the decades. Furthermore, this chapter explores the shift from mathematical fundamentals and knowledge in software development towards skills in model validation, applied statistics, and integrated subject matter expertise. The review is interspersed with code examples to complement the theoretical foundations and illustrate model validation and machine learning explainability for science. The scope of this review includes various shallow machine learning methods, e.g. Decision Trees, Random Forests, Support-Vector Machines, and Gaussian Processes, as well as, deep neural networks, including feed-forward neural networks, convolutional neural networks, recurrent neural networks and generative adversarial networks. Regarding geoscience, the review has a bias towards geophysics but aims to strike a balance with geochemistry, geostatistics, and geology, however excludes remote sensing, as this would exceed the scope. In general, I aim to provide context for the recent enthusiasm surrounding deep learning with respect to research, hardware, and software developments that enable successful application of shallow and deep machine learning in all disciplines of Earth science.

J. S. Dramsch (Sept. 2020d). “70 years of machine learning in geoscience in review”. In: *Advances in Geophysics*. Ed. by B. Moseley and L. Krischer. Published, Peer-Reviewed, Section 2.2. Academic Press. Chap. 4. ISBN: 9780128216699

In recent years machine learning has become an increasingly important interdisciplinary tool that has advanced several fields of science, such as biology (Ching et al., 2018), chemistry (Schütt et al., 2017a), medicine (Shen et al., 2017) and pharmacology (Kadurin et al., 2017). Specifically, the method of deep neural networks has found wide application. While geoscience was slower in the adoption, bibliometrics show the adoption of deep learning in all aspects of geoscience. Most subdisciplines of geoscience have been treated to a review of machine learning. Remote sensing has been an early adopter (Lary et al., 2016), with geomorphology (Valentine et al., 2016), solid Earth geoscience (Bergen et al., 2019), hydrogeophysics (Shen, 2018), seismology (Kong et al., 2019), seismic interpretation (Wang et al., 2018) and geochemistry (Zuo et al., 2019) following

suite. Climate change, in particular, has received a thorough treatment of the potential impact of varying machine learning methods for modelling, engineering and mitigation to address the problem (Rolnick et al., 2019). This review addresses the development of applied statistics and machine learning in the wider discipline of geoscience in the past 70 years and aims to provide context for the recent increase in interest and successes in machine learning and its challenges¹.

Machine learning (ML) is deeply rooted in applied statistics, building computational models that use inference and pattern recognition instead of explicit sets of rules. Machine learning is generally regarded as a sub-field of artificial intelligence (AI), with the notion of AI first being introduced by Turing (1950). Samuel (1959) coined the term machine learning itself, with Mitchell et al. (1997) providing a commonly quoted definition:

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E .

Mitchell et al. (1997)

This means that a machine learning model is defined by a combination of requirements. A task such as, classification, regression, or clustering is improved by conditioning of the model on a training data set. The performance of the model is measured with regard to a loss, also called metric, which quantifies the performance of a machine learning model on the provided data. In regression, this would be measuring the misfit of the data from the expected values. Commonly, the model improves with exposure to additional samples of data. Eventually, a good model generalizes to unseen data, which was not part of the training set, on the same task the model was trained to perform.

Accordingly, many mathematical and statistical methods and concepts, including Bayes' rule (Bayes, 1763), least-squares (Legendre, 1805), and Markov models (Markov, 1906; Markov, 1971), are applied in machine learning. Gaussian processes stand out as they originate in time series applications (Kolmogorov, 1939) and geostatistics (Kriging, 1951), which roots this machine learning application in geoscience (Rasmussen, 2003). "Kriging" originally applied two-dimensional Gaussian processes to the prediction of gold mine valuation and has since found wide application in geostatistics. Generally, Matheron (1963) is credited with formalizing the mathematics of kriging and developing it further in the following decades.

Between 1950 and 2020 much has changed. Computational resources are now widely available both as hardware and software, with high-performance compute being affordable to anyone from cloud computing vendors. High-quality software for machine learning is widely available through the free and open-source software movement, with major companies (Google, Facebook, Microsoft) competing for the usage of their open-source

¹The author of this manuscript has a background in geophysics, exploration geoscience, and active source 4D seismic. While this skews the expertise, they attempt to give a full overview over developments in all of geoscience with the minimum amount of bias possible.

machine learning frameworks (Tensorflow, Pytorch, CNTK²) and independent developments reaching wide applications such as scikit-learn (Pedregosa et al., 2011) and xgboost (Chen et al., 2016).

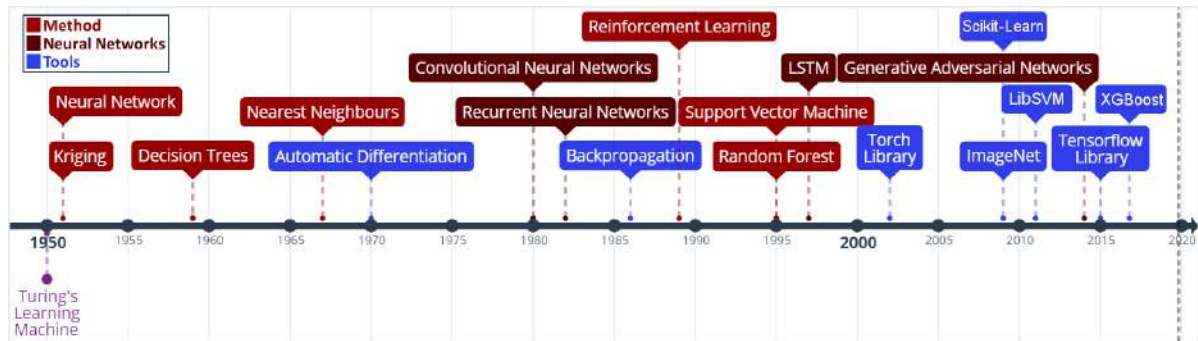


Figure 2.1: Machine Learning timeline from (Dramsch, 2019a). Neural Networks: (Russell et al., 2010); Kriging: (Krige, 1951); Decision Trees: (Belson, 1959); Nearest Neighbours: (Cover et al., 1967); Automatic Differentiation: (Linnainmaa, 1970); Convolutional Neural Networks: (Fukushima, 1980; LeCun et al., 2015); Recurrent Neural Networks: (Hopfield, 1982); Backpropagation: (Kelley, 1960; Bryson, 1961; Dreyfus, 1962; Rumelhart et al., 1988b); Reinforcement Learning: (Watkins, 1989); Support Vector Machines: (Cortes et al., 1995); Random Forests: (Ho, 1995); LSTM: (Hochreiter et al., 1997); Torch Library: (Collobert et al., 2002); ImageNet: (Deng et al., 2009b); Scikit-Learn: (Pedregosa et al., 2011); LibSVM: (Chang et al., 2011); Generative Adversarial Networks: (Goodfellow et al., 2014b); Tensorflow: (Martín Abadi et al., 2015); XGBoost: (Chen et al., 2016)

Nevertheless, investigations of machine learning in geoscience are not a novel development. The research into machine learning follows interest in artificial intelligence closely. Since its inception, artificial intelligence has experienced two periods of a decline in interest and trust, which has impacted negatively upon its funding. Developments in geoscience follow this wide-spread cycle of enthusiasm and loss of interest with a time lag of a few years. This may be the result of a variety of factors, including research funding availability and a change in willingness to publish results.

2.2.1 Historic Machine Learning in Geoscience

The 1950s and 1960s were decades of machine learning optimism, with machines learning to play simple games and perform tasks like route mapping. Intuitive methods like k-means, Markov models, and decision trees have been used as early as the 1960s in geoscience. K-means was used to describe the cyclicity of sediment deposits (Preston et al., 1964). Krumbein et al. (1969) give a thorough treatment of the mathematical foundations of Markov chains and embedded Markov chains in a geological context through

²Deprecated 2019

application to sedimentological processes, which also provides a comprehensive bibliography of Markov processes in geology. Some selected examples of early applications of Markov chains are found in sedimentology (Schwarzacher, 1972), well log analysis (Agterberg, 1966), hydrology (Matalas, 1967), and volcanology (Wickman, 1968). Decision tree-based methods found early applications in economic geology and prospectivity mapping (Newendorp, 1976; Reddy et al., 1991).

The 1970s were left with few developments in both the methods of machine learning, as well as, applications and adoption in geoscience (cf. Figure 2.1), due to the "first AI winter" after initial expectations were not met. Nevertheless, as kriging was not considered an AI technology, it was unaffected by this cultural shift and found applications in mining (Huijbregts et al., 1970), oceanography (Chiles et al., 1975), and hydrology (Delhomme, 1978). This was in part due to superior results over other interpolation techniques, but also the provision of uncertainty measures.

2.2.1.1 Expert Systems to Knowledge-Driven AI

The 1980s marked uptake in interest in machine learning and artificial intelligence through so-called "expert systems" and corresponding specialized hardware. While neural networks were introduced in 1950, the tools of automatic differentiation and back-propagation for error-correcting machine learning were necessary to spark their adoption in geophysics in the late 1980s. Zhao et al. (1988) performed seismic deconvolution with a recurrent neural network (Hopfield network). Dowla et al. (1990) discriminated between natural earthquakes and underground nuclear explosions using feed-forward neural networks. An ensemble of networks was able to achieve 97 % accuracy for nuclear monitoring. Moreover, the researchers inspected the network to gain the insight that the ratio of particular input spectra was beneficial to the discrimination of seismological events to the network. However, in practice the neural networks underperformed on uncured data, which is often the case in comparison to published results. Huang et al. (1990) presented work on self-organizing maps (also Kohonen networks), a special type of unsupervised neural network applied to pick seismic horizons. The field of geostatistics saw a formalization of theory and an uptake in interest with Matheron et al. (1981) formalizing the relationship of spline-interpolation and kriging and Dubrule (1984) further develop the theory and apply it to well data. At this point, kriging is well-established in the mining industry as well as other disciplines that rely on spatial data, including the successful analysis and construction of the Channel tunnel (Chilès et al., 2018). The late 1980s then marked the second AI winter, where expensive machines tuned to run "expert systems" were outperformed by desktop hardware from non-specialist vendors, causing the collapse of a half-billion-dollar hardware industry. Moreover, government agencies cut funding in AI specifically.

The 1990s are generally regarded as the shift from a knowledge-driven to a data-driven approach in machine learning. The term AI and especially expert systems were almost exclusively used in computer gaming and regarded with cynicism and as a failure in the scientific world. In the background, however, with research into applied

statistics and machine learning, this decade marked the inception of Support-Vector Machines (SVM) (Cortes et al., 1995), the tree-based method Random Forests (RF) (Ho, 1995), and a specific type of recurrent neural network (RNN) Long Short-Term Memories (LSTM) (Hochreiter et al., 1997). SVMs were utilized for land usage classification in remote sensing early on (Hermes et al., 1999). Geophysics applied SVMs a few years later to approximate the Zoeppritz equations for AVO inversion, outperforming linearized inversion (Kuzma, 2003). Random Forests, however, were delayed in broader adoption, due to the term "random forests" only being coined in 2001 (Breiman, 2001) and the statistical basis initially being less rigorous and implementation being more complicated. LSTMs necessitate large amounts of data for training and can be expensive to train, after further development in 2011 (Ciresan et al., 2011) it gained popularity in commercial time series applications particularly speech and audio analysis.

2.2.1.2 Neural Networks

McCormack (1991) marks the first review of the emerging tool of neural networks in geophysics. The paper goes into the mathematical details and explores pattern recognition. The author summarizes neural network applications over the 30 years prior to the review and presents worked examples in automated well-log analysis and seismic trace editing. The review comes to the conclusion that neural networks are, in fact, good function approximators, taking over tasks that were previously reserved for human work. He criticizes slow training, the cost of retraining networks upon new knowledge, imprecision of outputs, non-optimal training results, and the black box property of neural networks. The main conclusion sees the implementation of neural networks in conventional computation and expert systems to leverage the pattern recognition of networks with the advantages of conventional computer systems.

Neural networks are the primary subject of the modern day machine learning interest, however, significant developments leading up to these successes were made prior to the 1990s. The first neural network machine was constructed by Minsky [described in Russell et al. (2010)] and soon followed by the "Perceptron", a binary decision boundary learner (Rosenblatt, 1958). This decision was calculated as follows:

$$\begin{aligned}
 o_j &= \sigma \left(\sum_i w_{ij} x_i + b \right) \\
 &= \sigma(a_j) \\
 &= \begin{cases} 1 & a_j > 0 \\ 0 & \text{otherwise} \end{cases}
 \end{aligned} \tag{2.4}$$

It describes a linear system with the output o , the linear activation a of the input data x , the index of the source i and target node j , the trainable weights w , the trainable bias b and a binary activation function σ . The activation function σ in particular has received ample attention since its inception. During this period, a binary σ became uncommon and was replaced by non-linear mathematical functions. Neural networks are commonly trained by gradient descent, therefore, differentiable functions like sigmoid or tanh, allowing for the activation o of each neuron in a neural network to be continuous.

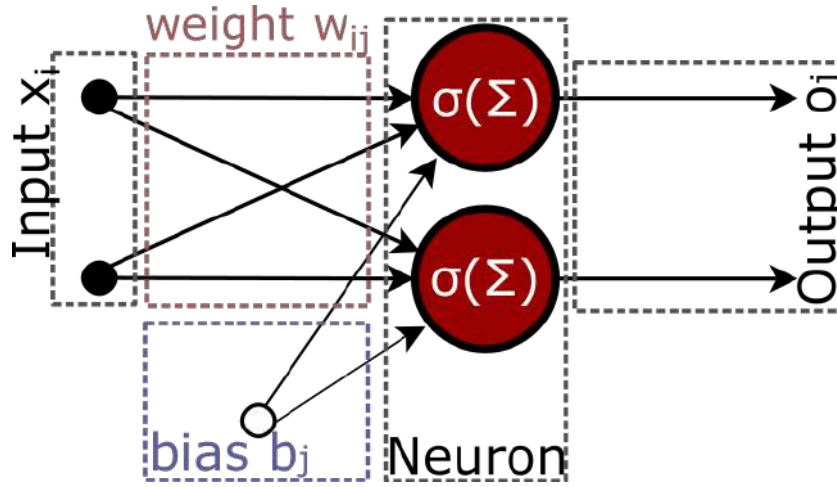


Figure 2.2: Single layer neural network as described in equation 2.4. Two inputs x_i are multiplied by the weights w_{ij} and summed with the biases b_j . Subsequently an activation function σ is applied to obtain out outputs o_j .

Deep learning (Dechter, 1986) expands on this concept. It is the combination of multiple layers of neurons in a neural network. These deep networks learn representations with multiple levels of abstraction and can be expressed using equation 2.4 as **input neurons** to the next layer

$$\begin{aligned} o_k &= \sigma \left(\sum_k w_{jk} \cdot o_j + b \right) \\ &= \sigma \left(\sum_k w_{jk} \cdot \sigma \left(\sum_j w_{ij} x_i + b \right) + b \right) \end{aligned} \quad (2.5)$$

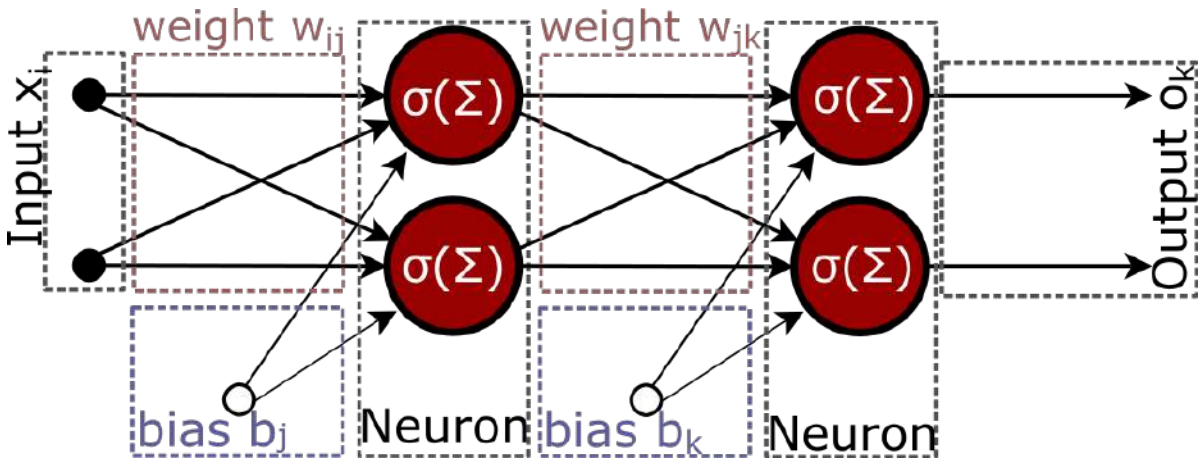


Figure 2.3: Deep multi-layer neural network as described in equation 2.5.

Röth et al. (1994) apply these building blocks of multi-layered neural networks with sigmoid activation to perform seismic inversion. They successfully invert low-noise and noise-free data on small training data. The authors note that the approach is susceptible to errors at low signal-to-noise ratios and coherent noise sources. Further applications

include electromagnetic subsurface localization (Poulton et al., 1992), magnetotelluric inversion via Hopfield neural networks (Zhang et al., 1997), and geomechanical microfractures modelling in triaxial compression tests (Feng et al., 1998).

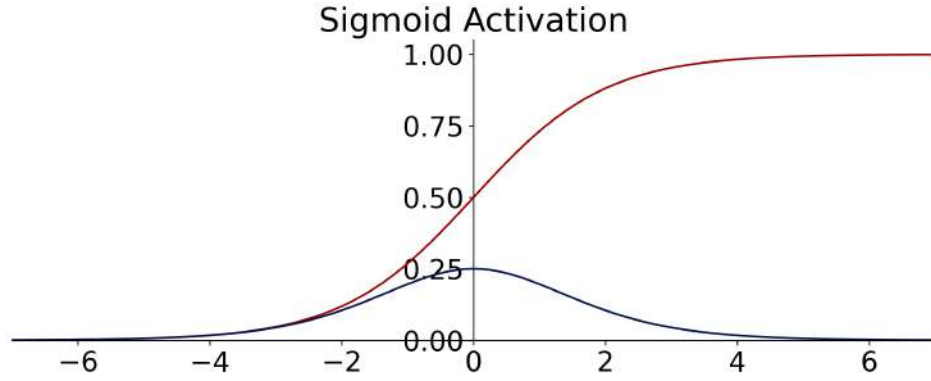


Figure 2.4: Sigmoid activation function (red) and derivative (blue) to train multi-layer Neural Network described in equation 2.5.

2.2.1.3 Kriging and Gaussian Processes

Cressie (1990) review the history of kriging, prompted by the uptake of interest in geostatistics. The author defines kriging as Best Linear Unbiased Prediction and reviews the historical co-development of disciplines. Similar concepts were developed with mining, meteorology, physics, plant and animal breeding, and geodesy that relied on optimal spatial prediction. Later, Williams (1998) provide a thorough treatment of Gaussian Processes, in the light of recent successes of neural networks.

An alternative method of putting a prior over functions is to use a Gaussian process (GP) prior over functions. This idea has been used for a long time in the spatial statistics community under the name of "kriging", although it seems to have been largely ignored as a general-purpose regression method.

Williams (1998)

Overall, Gaussian Processes benefit from the fact that a Gaussian distribution will stay Gaussian under conditioning. That means that we can use Gaussian distributions in this machine learning process and they will produce a smooth Gaussian result after conditioning on the training data. To become a universal machine learning model, Gaussian Processes have to be able to describe infinitely many dimensions. Instead of storing infinite values to describe this random process, Gaussian Processes go the path of describing a distribution over functions that can produce each value when required.

$$p(x) \approx \mathcal{GP}(\mu(x), k(x, x')), \quad (2.6)$$

The multivariate distribution over functions $p(x)$ is described by the Gaussian Process depends on mean a function $\mu(x)$ and a covariance function $k(x, x')$. It follows that choosing an appropriate mean and covariance function, also known as kernel, is essential. Very commonly, the mean function is chosen to be zero, as this simplifies some of the math. Therefore, data with a non-zero mean is commonly centered to comply with this assumption (Görtler et al., 2019). Choosing an appropriate kernel for the machine learning task is one of the benefits of the Gaussian Process. The kernel is where expert knowledge can be incorporated into data, e.g. seasonality metereological data can be described by a periodic covariance function.

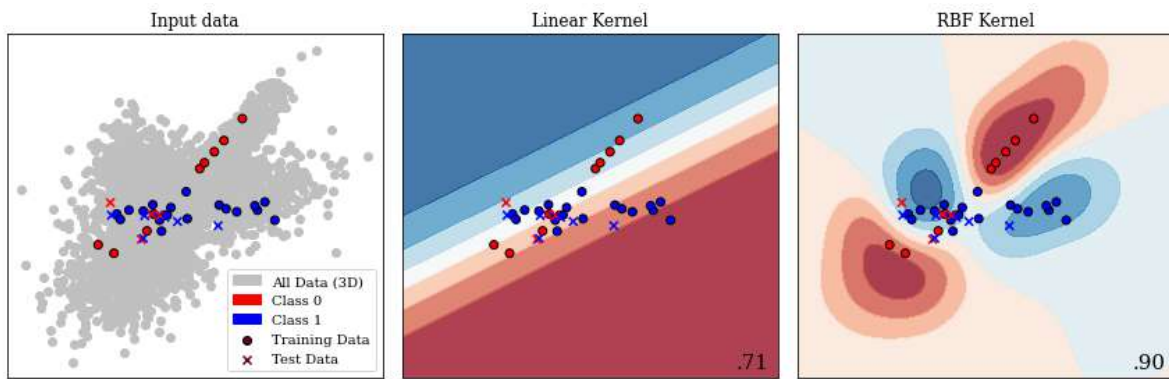


Figure 2.5: Gaussian Process separating two classes with different kernels. This image presents a 2D slice out of a 3D decision space. The decision boundary learnt from the data is visible, as well as the prediction in every location of the 2D slice. The two kernels presented are a linear kernel and a radial basis function (RBF) kernel, which show a significant discrepancy in performance. The bottom right number shows the accuracy on unseen test data. The linear kernel achieves 71 % accuracy, while the RBF kernel achieves 90 %.

Figure 2.5 present a 2D slice of 3D data with two classes. This binary problem can be approached by applying a Gaussian Process to it. In the second panel, a linear kernel is shown, which predicts the data relatively poorly with an accuracy of 71 %. A radial basis function (RBF) kernel, shown in the third panel generalizes to unseen test data with an accuracy of 90 %. This figure shows how a trained Gaussian Process would predict any new data point presented to the model. The linear kernel would predict any data in the top part to be blue (Class 0) and any data in the bottom part to be red (Class 1). The RBF kernel, which we explore further in the subsection introducing support-vector machines, separates the prediction into four uneven quadrants. The choice of kernel is very important in Gaussian Processes and research into extracting specific kernels is ongoing (Duvenaud, 2014).

In a more practical sense, Gaussian processes are computationally expensive, as an $n \times n$ matrix must be inverted, with n being the number of samples. This results in a space complexity of $\mathcal{O}(n^2)$ and a time complexity $\mathcal{O}(n^3)$ (Williams et al., 2006). This makes Gaussian Processes most feasible for smaller data problems, which is one explana-

tion for their rapid uptake in geoscience. An approximate computation of the inverted matrix is possible using the Conjugate Gradient (CG) optimization method, which can be stopped early with a maximum time cost of $\mathcal{O}(n^3)$ (Williams et al., 2006). For problems with larger data sets, neural networks become feasible due to being computationally cheaper than Gaussian Processes, regularization on large data sets being viable, as well as, their flexibility to model a wide variety of functions and objectives. Regularization being essential as neural networks tend to not "overfit" and simply memorize the training data, instead of learning a generalizable relationship of the data. Interestingly, Hornik et al. (1989) showed that neural networks are a universal function approximator as the number of weights tend to infinity, and Neal (1996) were able to show that the infinitely wide stochastic neural network converges to a Gaussian Process. Oftentimes Gaussian Processes are trained on a subset of a large data set to avoid the computational cost. Gaussian Processes have seen successful application on a wide variety of problems and domains that benefit from expert knowledge.

The 2000s were opened with a review by Baan et al. (2000) recapitulating the most recent geophysical applications in neural networks. They went into much detail on the neural networks theory and the difficulties in building and training these models. The authors identify the following subsurface geoscience applications through history: First-break picking, electromagnetics, magnetotellurics, seismic inversion, shear-wave splitting, well log analysis, trace editing, seismic deconvolution, and event classification. They reveal a strong focus on exploration geophysics. The authors evaluated the application of neural networks as subpar to physics-based approaches and concluded that neural networks are too expensive and complex to be of real value in geoscience. This sentiment is consistent with the broader perception of artificial intelligence during this decade. Artificial intelligence and expert systems over-promised human-like performance, causing a shift in focus on research into specialized sub-fields, e.g. machine learning, fuzzy logic, and cognitive systems.

2.2.2 Contemporary Machine Learning in Geoscience

Mjolsness et al. (2001) review machine learning in a broader context outside of exploration geoscience. The authors discuss recent successes in applications of remote sensing and robotic geology using machine learning models. They review graphical models, (hidden) Markov models, and SVMs and go on to disseminate the limitations of applications to vector data and poor performance when applied to rich data, such as graphs and text data. Moreover, the authors from NASA JPL go into detail on pattern recognition in automated rovers to identify geological prospects on Mars. They state:

The scientific need for geological feature catalogs has led to multiyear human surveys of Mars orbital imagery yielding tens of thousands of cataloged, characterized features including impact craters, faults, and ridges.

Mjolsness et al. (2001)

The review points out the profound impact SVMs have on identifying geomorphological features without modelling the underlying processes.

2.2.2.1 Modern Machine Learning Tools

This decade of the 2000s introduces a shift in tooling, which is a direct contributor to the recent increase in adoption and research of both shallow and deep machine learning research.

Machine Learning software has been primarily comprised of proprietary software like MatlabTM with the Neural Networks Toolbox and Wolfram MathematicaTM or independent university projects like the Stuttgart Neural Network Simulator (SNNS). These tools were generally closed source and hard or impossible to extend and could be difficult to operate due to limited accompanying documentation. Early open-source projects include WEKA (Witten et al., 2005), a graphical user interface to build machine learning and data mining projects. Shortly after that, LibSVM was released as free open-source software (FOSS) (Chang et al., 2011), which implements support vector machines efficiently. It is still used in many other libraries to this day, including WEKA (Chang et al., 2011). Torch was then released in 2002, which is a machine learning library with a focus on neural networks. While it has been discontinued in its original implementation in the programming language Lua (Collobert et al., 2002), PyTorch, the reimplementation in the programming language Python, is one of the leading deep learning frameworks at the time of writing (Paszke et al., 2017). In 2007, the libraries Theano and scikit-learn were released openly licensed in Python (Team, 2016; Pedregosa et al., 2011). Theano is a neural network library that was a tool developed at the Montreal Institute for Learning Algorithms (MILA) and ceased development in 2017 after strong industrial developers had released openly licensed deep learning frameworks. Scikit-learn implements many different machine learning algorithms, including SVMs, Random Forests and single-layer neural networks, as well as utility functions including cross-validation, stratification, metrics and train-test splitting, necessary for robust machine learning model building and evaluation.

2.2.2.2 Support-Vector Machines

The impact of scikit-learn has shaped the current machine learning software package by implementing a unified application programming interface (API) (Buitinck et al., 2013). This API is explored by example in the following code snippets, the code can be obtained at Dramsch (2020c). First, we generate a classification dataset using a utility function. The `make_classification` function takes different arguments to adjust the desired arguments, we are generating 5000 samples (`n_samples`) for two classes, with five features (`n_features`), of which three features are actually relevant to the classification (`n_informative`). The data is stored in X , whereas the labels are contained in y .

```
1 # Generate random classification dataset for example
2 from sklearn.datasets import make_classification
```



```

3 X, y = make_classification(n_samples=5000, n_features=5,
4                           n_informative=3, n_redundant=0,
5                           random_state=0, shuffle=False)

```

It is good practice to divide the available labeled data into a training data set and a validation or test data set. This split ensures that models can be evaluated on unseen data to test the generalization to unseen samples. The utility function `train_test_split` takes an arbitrary amount of input arrays and separates them according to specified arguments. In this case 25% of the data are kept for the hold-out validation set and not used in training. The `random_state` is fixed to make these examples reproducible.

```

1 # Split data into train and validation set
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X, y,
4                                                    test_size=.25,
5                                                    random_state=0)

```

Then we need to define a machine learning model, considering the previous discussion of high impact machine learning models, the first example is an SVM classifier. This example uses the default values for hyperparameters of the SVM classifier, for best results on real-world problems these have to be adjusted. The machine learning training is always done by calling `classifier.fit(X, y)` on the classifier object, which in this case is the SVM object. In more detail, the `.fit()` method implements an optimization loop that will condition the model to the training data by minimizing the defined loss function. In the case of the SVM classification the parameters are adjusted to optimize a hinge loss, outlined in equation 2.8. The trained model scikit-learn model contains information about all its hyperparameters in addition to the trained model, shown below. The exact meaning of all these hyperparameters is laid out in the scikit-learn documentation (Buitinck et al., 2013).

```

1 # Define and train a Support Vector Machine Classifier
2 from sklearn.svm import SVC
3 svm = SVC(random_state=0)
4 svm.fit(X_train, y_train)
5
6 >>> SVC(C=1.0, break_ties=False, cache_size=200,
7        class_weight=None, coef0=0.0, degree=3,
8        decision_function_shape='ovr', gamma='scale',
9        kernel='rbf', max_iter=-1, probability=False,
10       random_state=0, shrinking=True, tol=0.001,
11       verbose=False)

```

The trained SVM can be used to predict on new data, by calling `classifier.predict(data)` on the trained classifier object. The new data has to contain four features like the training data did. Generally, machine learning models always need to be trained on the same

set of input features as the data available for prediction. The `.predict()` method outputs the most likely estimate on the new data to generate predictions. In the following code snippet, three predictions on three input vectors are performed on the previously trained model.

```
1 # Predict on new data with trained SVM
2 print(svm.predict([[0, 0, 0, 0, 0],
3                   [-1, -1, -1, -1, -1],
4                   [1, 1, 1, 1, 1]]))
5 >>> [1 0 1]
```

The blackbox model should be evaluated with the `classifier.score()` function. Evaluating the performance on the training data set gives an indication how well the model is performing, but this is generally not enough to gauge the performance of machine learning models. In addition, the trained model has to be evaluated on the hold-out set, a dataset the model has not been exposed to during training. This avoids that the model only performs well on the training data by "memorization" instead of extracting meaningful generalizable relationships, an effect called overfitting. In this example the hyperparameters are left to the default values, in real-life applications hyperparameters are usually adjusted to build better models. This can lead to an additional meta-level of overfitting on the hold-out set, which necessitates an additional third hold-out set to test the generalizability of the trained model with optimized hyperparameters. The default score uses the class accuracy, which suggests our model is approximately 90% correct. Similar train and test scores indicate that the model learned a generalizable model, enabling prediction on unseen data without a performance loss. Large differences between the training score and test score indicate either overfitting, in the case of a better training score. A higher test score than training score can be an indication of a deeper problem with the data split, scoring, class imbalances, and needs to be investigated by means of external cross-validation, building standard "dummy" models, independence tests, and further manual investigations.

```
1 # Score SVM on train and test data
2 print(svm.score(X_train, y_train))
3 print(svm.score(X_test, y_test))
4 >>> 0.9098666666666667
5 >>> 0.9032
```

Support-vector machines can be employed for each class of machine learning problem, i.e. classification, regression, and clustering. In a two-class problem, the algorithm considers the n -dimensional input and attempts to find a $(n-1)$ -dimensional hyperplane that separates these input data points. The problem is trivial if the two classes are linearly separable, also called a hard margin. The plane can pass the two classes of data without ambiguity. For data with an overlap, which is usually the case, the problem becomes an optimization problem to fit the ideal hyperplane. The hinge loss provides

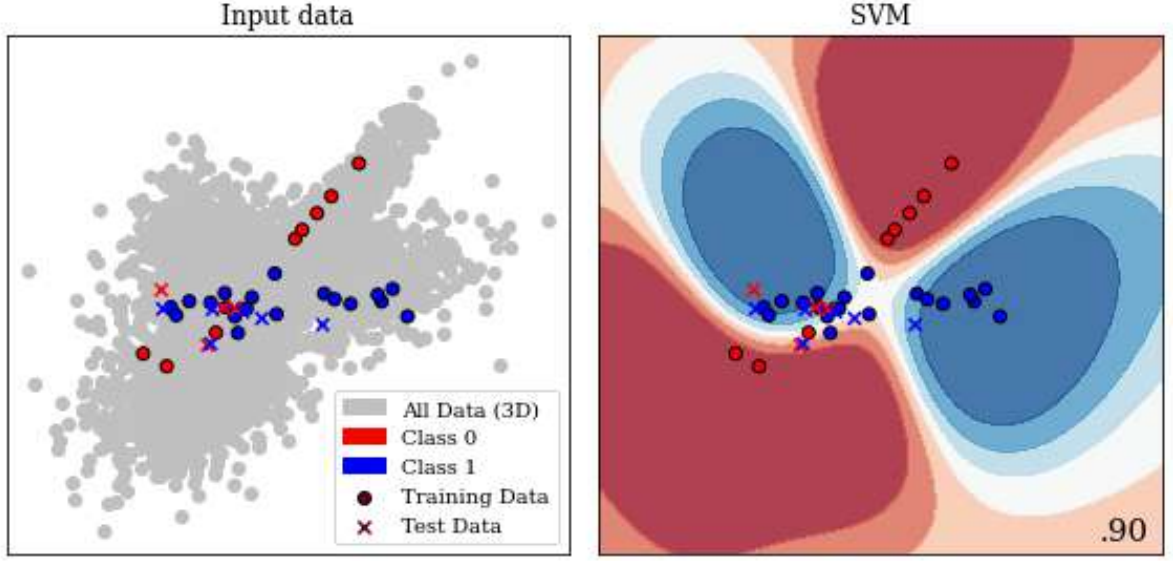


Figure 2.6: Example of Support Vector Machine separating two classes, showing the decision boundary learnt from the data. The data contains three informative features, the decision boundary is therefore three dimensional, shown is a central slice of data points in 2D. (A video is available at (Dramsch, 2020a))

the ideal loss function for this problem, yielding 0 if none of the data overlap, but a linear residual for overlapping points that can be minimized:

$$\max(0, (1 - y_i(\vec{w} \cdot \vec{x}_i - b))), \quad (2.7)$$

with y_i being the current target label and $\vec{w} \cdot \vec{x}_i - b$ being the hyperplane under consideration. The hyperplane consists of w the normal vector and point x , with the offset b . This leads the algorithm to optimize

$$\left[\frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i(w \cdot x_i - b)) \right] + \lambda \|w\|^2, \quad (2.8)$$

with λ being a scaling factor. For small λ the loss becomes the hard margin classifier for linearly separable problems. The nature of the algorithm dictates that only values for \vec{x} close to the hyperplane define the hyperplane itself; these values are called the support vectors.

The SVM algorithm would not be as successful if it were simply a linear classifier. Some data can become linearly separable in higher dimensions. This, however, poses the question of how many dimensions should be searched, because of the exponential cost in computation that follows due to the increase of dimensionality (also known as the curse of dimensionality). Instead, the "kernel trick" was proposed (Aizerman, 1964), which defines a set of values that are applied to the input data simply via the dot product. A common kernel is the radial basis function (RBF), which is also the kernel we applied in the example. The kernel is defined as:

$$k(\vec{x}_i, \vec{x}_j) \rightarrow \exp\left(-\gamma\|\vec{x}_i - \vec{x}_j\|^2\right) \quad (2.9)$$

This specifically defines the Gaussian Radial Basis Function of every input data point with regard to a central point. This transformation can be performed with other functions (or kernels), such as, polynomials or the sigmoid function. The RBF will transform the data according to the distance between x_i and x_j , this can be seen in Figure 2.7. This results in the decision surface in Figure 2.6 consisting of various Gaussian areas. The RBF is generally regarded as a good default, in part, due to being translation invariant (i.e. stationary) and smoothly varying.

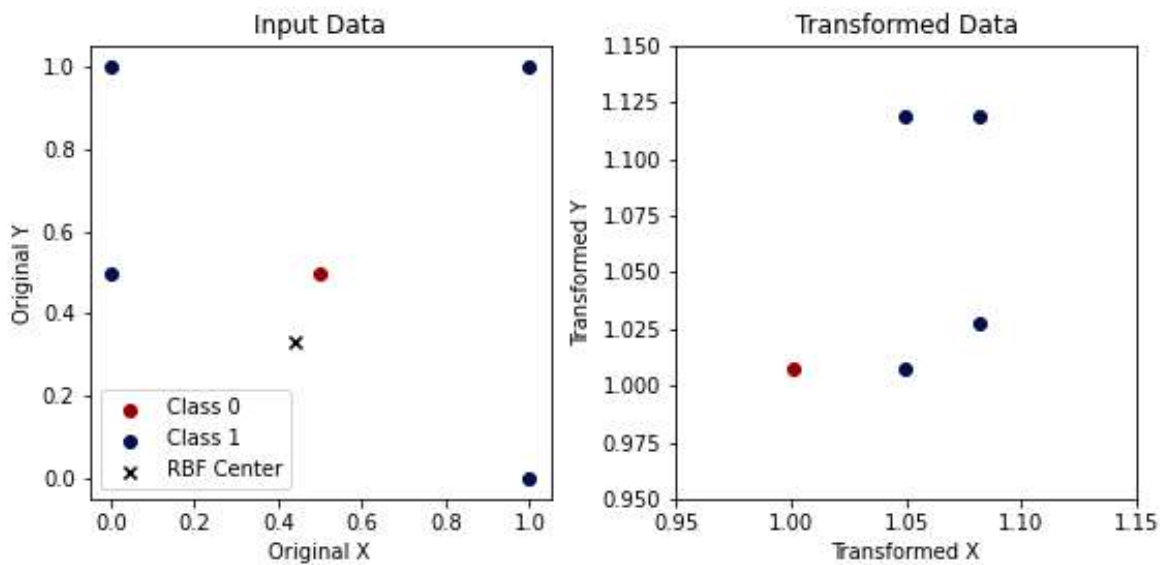


Figure 2.7: Samples from two classes that are not linearly separable input data (left). Applying a Gaussian Radial Basis Function centered around (0.4, 0.33) with $\lambda = .5$ results in the two classes being linearly separable.

An important topic in machine learning is explainability, which inspects the influence of input variables on the prediction. We can employ the utility function `permutation_importance` to inspect any model and how they perform with regard to their input features (Breiman, 2001). The permutation importance evaluates how well the blackbox model performs, when a feature is not available. Practically, a feature is replaced with random noise. Subsequently, the score is calculated, which provides a representation how informative a feature is compared to noise. The data we generated in the first example contains three informative features and two random data columns. The mean values of the calculated importances show that three features are estimated to be three magnitudes more important, with the second feature containing the maximum amount of information to predict the labels.

```
1 # Calculate permutation importance of SVM model
2 from sklearn.inspection import permutation_importance
```

```

3 importances = permutation_importance(svm, X_train, y_train,
4                                     n_repeats=10, random_state=0)
5
6 # Show mean value of importances and the ranking
7 print(importances.importances_mean)
8 print(importances.importances_mean.argsort())
9 >>> [ 2.1787e-01  2.8712e-01  1.2293e-01 -1.8667e-04  7.7333e-04]
10 >>> [3 4 2 0 1]

```

Support-vector machines were applied to seismic data analysis (Li et al., 2004) and the automatic seismic interpretation (Liu et al., 2015; Di et al., 2017b; Mardan et al., 2017). Compared to convolutional neural networks, these approaches usually do not perform as well, when the CNN can gain information from adjacent samples. Seismological volcanic tremor classification (Masotti et al., 2006; Masotti et al., 2008) and analysis of ground-penetrating radar (Pasolli et al., 2009; Xie et al., 2013) were other notable applications of SVM in Geoscience. The 2016 Society of Exploration Geophysicists (SEG) machine learning challenge was held using a SVM baseline (Hall, 2016). Several other authors investigated well log analysis (Anifowose et al., 2017; Caté et al., 2018; Gupta et al., 2018; Saporetto et al., 2018), as well as seismology for event classification (Malfante et al., 2018) and magnitude determination (Ochoa et al., 2018). These rely on SVMs being capable of regression on time-series data. Generally, many applications in geoscience have been enabled by the strong mathematical foundation of SVMs, such as microseismic event classification (Zhao et al., 2017b), seismic well ties (Chaki et al., 2018), landslide susceptibility (Marjanović et al., 2011; Ballabio et al., 2012), digital rock models (Ma et al., 2012), and lithology mapping (Cracknell et al., 2013).

2.2.2.3 Random Forests

The following example shows the application of Random Forests, to illustrate the similarity of the API for different machine learning algorithms in the scikit-learn library. The Random Forest classifier is instantiated with a maximum depth of seven, and the random state is fixed to zero again. Limiting the depth of the forest forces the random forest to conform to a simpler model. Random forests have the capability to become highly complex models that are very powerful predictive models. This is not conducive to this small example dataset, but easy to modify for the inclined reader. The classifier is then trained using the same API of all classifiers in scikit-learn. The example shows a very high number of hyperparameters, however, Random Forests work well without further optimization of these.

```

1 # Define and train a Random Forest Classifier
2 from sklearn.ensemble import RandomForestClassifier
3 rf = RandomForestClassifier(max_depth=7, random_state=0)
4 rf.fit(X_train, y_train)
5
6 >>> RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,

```

```

7         class_weight=None, criterion='gini', max_depth=7,
8         max_features='auto', max_leaf_nodes=None,
9         max_samples=None, min_impurity_decrease=0.0,
10        min_impurity_split=None, min_samples_leaf=1,
11        min_samples_split=2, min_weight_fraction_leaf=0.0,
12        n_estimators=100, n_jobs=None, oob_score=False,
13        random_state=0, verbose=0, warm_start=False)

```

The prediction of the random forest is performed in the same API call again, also consistent with all classifiers available. The values are slightly different from the prediction of the SVM.

```

1 # Predict on new data with trained Random Forest
2 print(rf.predict([[0, 0, 0, 0, 0],
3                  [-1, -1, -1, -1, -1],
4                  [1, 1, 1, 1, 1]]))
5 >>> [1 0 1]

```

The training score of the random forest model is 2.5 % better than the SVM in this instance, this score however not informative. Comparing the test scores shows only a 0.88 % difference, which is the relevant value to evaluate, as it shows the performance of a model on data it has not seen during the training stage. The random forest performed slightly better on the training set than the test data set. This slight discrepancy is usually not an indicator of an overfit model. Overfit models "memorize" the training data and do not generalize well, which results in poor performance on unseen data. Generally, overfitting is to be avoided in real application, but can be seen in competitions, on benchmarks, and show-cases of new algorithms and architectures to oversell the improvement over state-of-the-art methods (Recht et al., 2019).

```

1 # Score Random Forest on train and test data
2 print(rf.score(X_train, y_train))
3 print(rf.score(X_test, y_test))
4 >>> 0.9306
5 >>> 0.912

```

Random forests have specialized methods available for introspection, which can be used to calculate feature importance. These are based on the decision process the random forest used to build the machine learning model. The feature importance in Random Forests uses the same method as permutation importance, which is dropping out features to estimate their importance on the model performance. Random Forests use a measure to determine the split between classes at each node of the trees called Gini impurity. While the permutation importance uses the accuracy score of the prediction, in Random Forests this Gini impurity can be used to measure how informative a feature is in a model. It is important to note that this impurity-based process can be susceptible to noise and overestimate high number of classes in features. Using the permutation importance

instead is a valid choice. In this instance as opposed to the permutation importance, the random forest estimates the two non-informative features to be one magnitude less useful than the informative features, instead of two magnitudes.

```

1 # Inspect random forest for feature importance
2 print(rf.feature_importances_)
3 print(rf.feature_importances_.argsort())
4 >>> [0.2324 0.4877 0.2527 0.0141 0.0129]
5 >>> [4 3 0 2 1]

```

Random forests and other tree-based methods, including gradient boosting, a specialized version of random forests, have generally found wider application with the implementation into scikit-learn and packages for the statistical languages R and SPSS. Similar to neural networks, this method is applied to ASI (Guillen et al., 2015) with limited success, which is due to the independent treatment of samples, like SVMs. Random forests have the ability to approximate regression problems and time series, which made them suitable for seismological applications including localization (Dodge et al., 2016), event classification in volcanic tremors (Maggi et al., 2017) and slow slip analysis (Hulbert et al., 2018). They have also been applied to geomechanical applications in fracture modelling (Valera et al., 2017) and fault failure prediction (Rouet-Leduc et al., 2017; Rouet-Leduc et al., 2018), as well as, detection of reservoir property changes from 4D seismic data (Cao et al., 2017). Gradient Boosted Trees were the winning models in the 2016 SEG machine learning challenge (Hall et al., 2017) for well-log analysis, propelling a variety of publications in facies prediction (Bestagini et al., 2017; Blouin et al., 2017; Caté et al., 2018; Saporette et al., 2018).

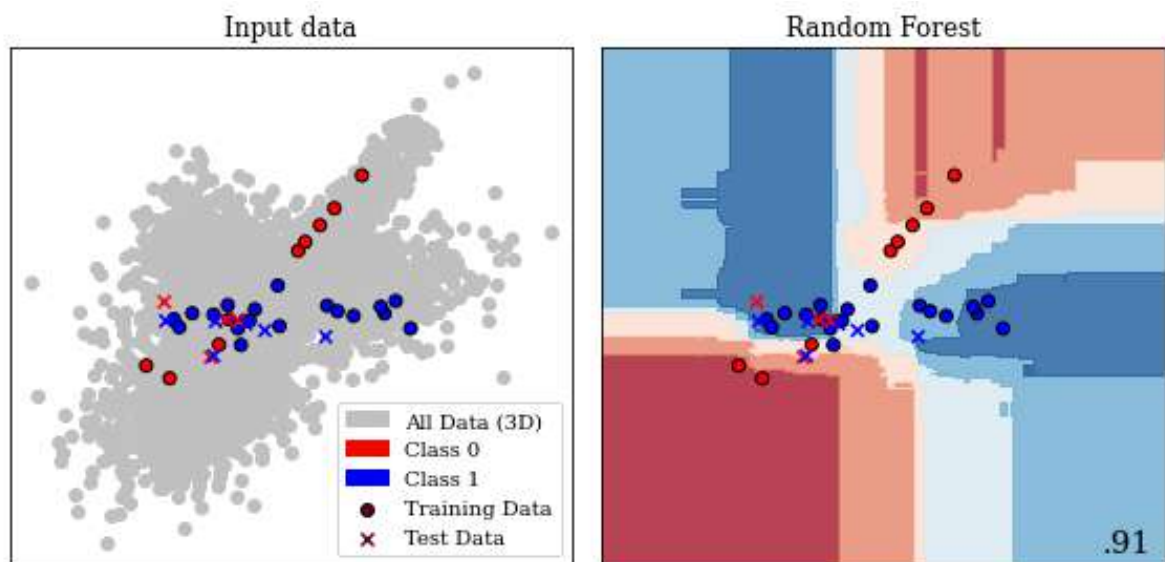


Figure 2.8: Binary Decision Boundary for Random Forest in 2D. This is the same central slice of the 3D decision volume used in Figure 2.6.

Furthermore, various methods that have been introduced into scikit-learn have been applied to a multitude of geoscience problems. Hidden Markov models were used on seismological event classification (Ohrnberger, 2001; Beyreuther et al., 2008; Bicego et al., 2013), well-log classification (Jeong et al., 2014; Wang et al., 2017a), and landslide detection from seismic monitoring (Dammeier et al., 2016). These hidden Markov models are highly performant on time series and spatially coherent problems. The "hidden" part of Markov models enables the model to assume influences on the predictions that are not directly represented in the input data. The K-nearest neighbours method has been used for well-log analysis (Caté et al., 2017; Saporetti et al., 2018), seismic well ties (Wang et al., 2017b) combined with dynamic time warping and fault extraction in seismic interpretation (Hale, 2013a), which is highly dependent on choosing the right hyperparameter k . The unsupervised k -NN equivalent, k -means has been applied to seismic interpretation (Di et al., 2017a), ground motion model validation (Khoshnevis et al., 2018), and seismic velocity picking (Wei et al., 2018). These are very simple machine learning models that are useful for baseline models. Graphical modelling in the form of Bayesian networks has been applied to seismology in modelling earthquake parameters (Kuehn et al., 2011), basin modelling (Martinelli et al., 2013), seismic interpretation (Ferreira et al., 2018) and flow modelling in discrete fracture networks (Karra et al., 2018). These graphical models are effective in causal modelling and gained popularity in modern applications of machine learning explainability, interpretability, and generalization in combination with do-calculus (Pearl, 2012).

2.2.2.4 Modern Deep Learning

The 2010s marked a renaissance of deep learning and particularly convolutional neural networks. The convolutional neural network (CNN) architecture AlexNet (Krizhevsky et al., 2012b) was the first CNN to enter the ImageNet challenge (Deng et al., 2009b). The ImageNet challenge is considered a benchmark competition and database of natural images established in the field of computer vision. This improved the classification error rate from 25.8 % to 16.4 % (top-5 accuracy). This has propelled research in CNNs, resulting in error rates on ImageNet of 2.25 % on top-5 accuracy in 2017 (Russakovsky et al., 2015). The Tensorflow library (Martín Abadi et al., 2015) was introduced for open source deep learning models, with some different software design compared to the Theano and Torch libraries.

The following example shows an application of deep learning to the data presented in the previous examples. The classification data set we use has independent samples, which leads to the use of simple densely connected feed-forward networks. Image data or spatially correlated datasets would ideally be fed to a convolutional neural network (CNN), whereas time series are often best approached with recurrent neural networks (RNN). This example is written using the Tensorflow library. PyTorch would be an equally good library to use.

All modern deep learning libraries take a modular approach to building deep neural networks that abstract operations into layers. These layers can be combined into input

and output configurations in highly versatile and customizable ways. The simplest architecture, which is the one we implement below, is a sequential model, which consists of one input and one output layer, with a "stack" of layers. It is possible to define more complex models with multiple inputs and outputs, as well as the branching of layers to build very sophisticated neural network pipelines. These models are called functional API and subclassing API, but would not be conducive to this example.

The example model consists of Dense layers and a Dropout layer, which are arranged in sequence. Densely connected layers contain a specified number of neurons with an appropriate activation function, shown in the example below. Each neuron performs the calculation outlined in equation 2.4, with σ defining the activation. Modern neural networks rarely implement `sigmoid` and `tanh` activations anymore. Their activation characteristic leads them to lose information for large positive and negative values of the input, commonly called saturation (Hochreiter et al., 2001). This saturation of neurons prevented good deep neural network performance until new non-linear activation functions took their place (Xu et al., 2015). The activation function Rectified linear unit (ReLU) is generally credited with facilitating the development of very deep neural networks, due to their non-saturating properties (Hahnloser et al., 2000). It sets all negative values to zero and provides a linear response for positive values, as seen in equation 2.10. Since its inception, many more rectifiers with different properties have been introduced.

$$\sigma(a) = \max(0, a) \quad (2.10)$$

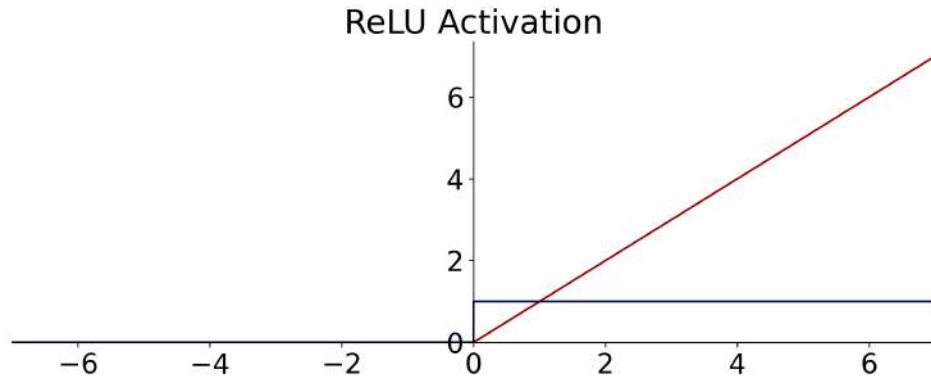


Figure 2.9: ReLU activation (red) and derivative (blue) for efficient gradient computation.

The other activation function used in the example is the "softmax" function on the output layer. This activation is commonly used for classification tasks, as it normalizes all activations at all outputs to one. It achieves this by applying the exponential function to each of the outputs in \vec{a} for class C and dividing that value by the sum of all exponentials:

$$\sigma(\vec{a}) = \frac{e^{a_j}}{\sum_p e^{a_p}} \quad (2.11)$$

The example additionally uses a Dropout layer, which is a common layer used for regularization of the network by randomly setting a specified percentage of nodes to zero for each iteration. Neural networks are particularly prone to overfitting, which is counteracted by various regularization techniques that also include input-data augmentation, noise injection, \mathcal{L}_1 and \mathcal{L}_2 constraints, or early-stopping of the training loop (Goodfellow et al., 2016). Modern deep learning systems may even leverage noisy student-teacher networks for regularization (Xie et al., 2019b).

```

1 import tensorflow as tf
2 model = tf.keras.models.Sequential([
3     tf.keras.layers.Dense(32, activation='relu'),
4     tf.keras.layers.Dropout(.3),
5     tf.keras.layers.Dense(16, activation='relu'),
6     tf.keras.layers.Dense(2, activation='softmax')])

```

These sequential models are also used for simple image classification models using CNNs. Instead of Dense layers, these are built up with convolutional layers, which are readily available in 1D, 2D, and 3D as Conv1D, Conv2D and Conv3D respectively. A two-dimensional CNN learns a so-called filter f for the $n \times m$ -dimensional image G , expressed as:

$$G^*(x, y) = \sum_{i=1}^n \sum_{j=1}^m f(i, j) \cdot G(x - i + c, y - j + c), \quad (2.12)$$

resulting in the central result G^* around the central coordinate c . In CNNs each layer learns several of these filters f , usually following by a down-sampling operation in n and m to compress the spatial information. This serves as a forcing function to learn increasingly abstract representations in subsequent convolutional layers.

This sequential example model of densely connected layers with a single input, 32, 16, and two neurons contains a total of 754 trainable weights. Initially, each of these weights is set to a pseudo-random value, which is often drawn from a distribution beneficial to fast training. Consequently, the data is passed through the network, and the result is numerically compared to the expected values. This form of training is defined as supervised training and error-correcting learning, which is a form of Hebbian learning. Other forms of learning exist and are employed in machine learning, e.g. competitive learning in self-organizing maps.

$$MAE = |y_j - o_j| \quad (2.13)$$

$$MSE = (y_j - o_j)^2 \quad (2.14)$$

In regression problems the error is often calculated using the Mean Absolute Error (MAE) or Mean Squared Error (MSE), the \mathcal{L}_1 shown in equation 2.13 and the \mathcal{L}_2 norm

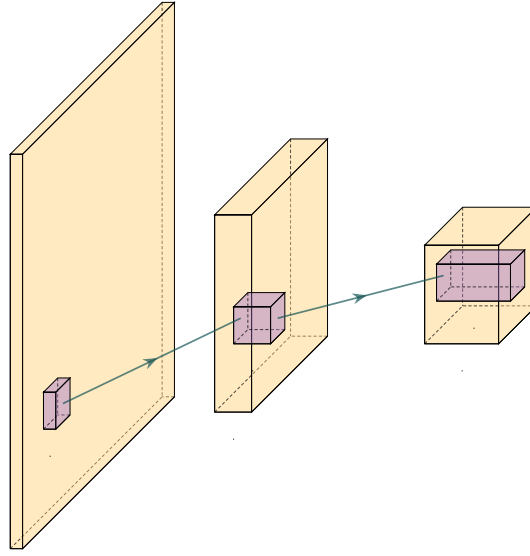


Figure 2.10: Three layer convolutional network. The input image (yellow) is convolved with several filters or kernel matrices (purple). Commonly, the convolution is used to downsample an image in the spatial dimension, while expanding the dimension of the filter response, hence expanding in "thickness" in the schematic. The filters are learned in the machine learning optimization loop. The shared weights within a filter improve efficiency of the network over classic dense networks.

shown in equation 2.14 respectively. Classification problems form a special type of problem that can leverage a different kind of loss called cross-entropy (CE). The cross-entropy is dependent on the true label y and the prediction in the output layer.

$$CE = - \sum_j^C y_j \log(o_j) \quad (2.15)$$

Many machine learning data sets have one true label $y_{true} = 1$ for class $C_{j=true}$, leaving all other $y_j = 0$. This makes the sum over all labels obsolete. It is debatable how much binary labels reflects reality, but it simplifies equation 2.15 to minimizing the (negative) logarithm of the neural network output o_j , also known as negative log-likelihood:

$$CE = - \log(o_j) \quad (2.16)$$

Technically, the data we generated is a binary classification problem, and this means we could use the sigmoid activation function in the last layer and optimize a binary CE. This can speed up computation, but in this example, an approach is shown that works for many other problems and can therefore be applied to the readers data.

```
1 model.compile(optimizer='adam', # Often 'adam' or 'sgd' are good
2               loss='sparse_categorical_crossentropy',
```

```
3 | metrics=['accuracy']) # Monitor other metrics
```

Large neural networks can be extremely costly to train with significant developments in 2019/2020 reporting multi-billion parameter language models (Google, OpenAI) trained on massive hardware infrastructure for weeks with a single epoch taking several hours. This calls for validation on unseen data after every epoch of the training run. Therefore, neural networks, like all machine learning models, are commonly trained with two hold-out sets, a validation and a final test set. The validation set can be provided or be defined as a percentage of the training data, as shown below. In the example, 10% of the training data are held out for validation after every epoch, reducing the training data set from 3750 to 3375 individual samples.

```
1 model.fit(X_train,
2           y_train,
3           validation_split=.1,
4           epochs=100)
5 >>> [...]
6 Epoch 100/100
7 3375/3375 [=====] - 0s 66us/sample
8 loss: 0.1567 - accuracy: 0.9401 -
9 val_loss: 0.1731 - val_accuracy: 0.9359
```

Neural networks are trained with variations of stochastic gradient descent (SGD), an incremental version of the classic steepest descent algorithm. We use the Adam optimizer, a variation of SGD that converges fast, but a full explanation would go beyond the scope of this chapter. The gist of the Adam optimizer is that it maintains a per-parameter learning rate of the first statistical moment (mean). This is beneficial for sparse problems and the second moment (uncentered variance), which is beneficial for noisy and non-stationary problems (Kingma et al., 2014). The main alternative to Adam is SGD with Nesterov momentum (Sutskever et al., 2013), an optimization method that models conjugate gradient methods (CG) without the heavy computation that comes with the search in CG. SGD anecdotally finds a better optimal point for neural networks than Adam but converges much slower.

In addition to the loss value, we display the accuracy metric. While accuracy should not be the sole arbiter of model performance, it gives a reasonable initial estimate, how many samples are predicted correctly with a percentage between zero and one. As opposed to scikit-learn, deep learning models are compiled after their definition to make them fit for optimization on the available hardware. Then the neural network can be fit like the SVM and Random Forest models before, using the `X_train` and `y_train` data. In addition, a number of epochs can be provided to run, as well as other parameters that are left on default for the example. The amount of epochs defines how many cycles of optimization on the full training data set are performed. Conventional wisdom for neural network training is that it should always learn for more epochs than machine learning researchers estimate initially.

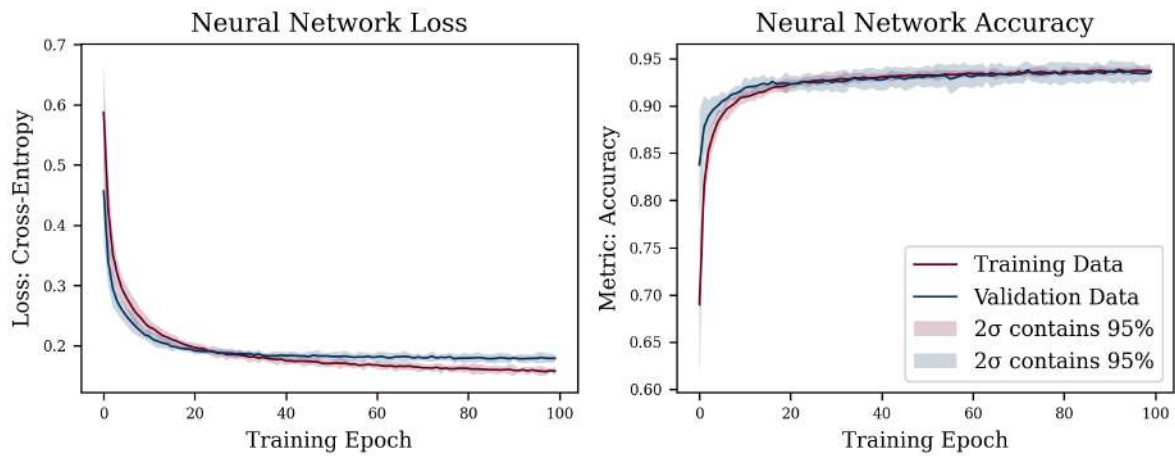


Figure 2.11: Loss and Accuracy of example neural network on ten random initializations. Training for 100 epochs with the shaded area showing the 95% confidence intervals of the loss and metric. Analyzing loss curves is important to evaluate overfitting. The training loss decreasing, while validation loss is close to plateauing is a sign of overfitting. Generally, it can be seen that the model converged and is only making marginal gains with the risk of overfitting.

It can be difficult to fix all sources of randomness and stochasticity in neural networks, to make both research and examples reproducible. This example does not fix these so-called random seeds as it would detract from the example. That implies that the results for loss and accuracy will differ from the printed examples. In research fixing the seed is very important to ensure reproducibility of claims. Moreover, to avoid bad practices or so-called "lucky seeds", a statistical analysis of multiple fixed seeds is good practice to report results in any machine learning model.

```

1 model.evaluate(X_test, y_test)
2 >>> 1250/1250 [=====] - 0s 93us/sample
3     loss: 0.1998 - accuracy: 0.9360
4     [0.19976349686831235, 0.936]

```

In the example before, the SVM and Random Forest classifier were scored on unseen data. This is equally important for neural networks. Neural networks are prone to overfit, which we try to circumvent by regularizing the weights and by evaluating the final network on an unseen test set. The prediction on the test set is very close to the last epoch in the training loop, which is a good indicator that this neural network generalizes to unseen data. Moreover, the loss curves in figure 2.11 do not converge too fast, while converging. However, it appears that the network would overfit if we let training continue. The exemplary decision boundary in figure 2.12 very closely models the local distribution of the data, which is true for the entire decision volume (Dramsch, 2020a).

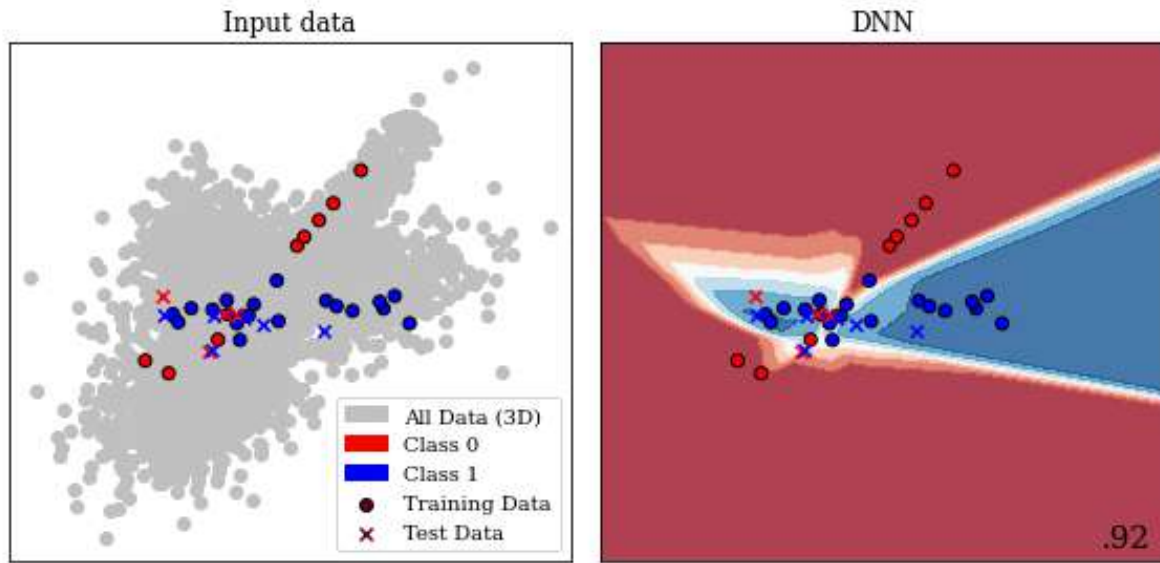


Figure 2.12: Central 2D slice of decision Boundary of deep neural network in trained on data with 3 informative features. The 3D volume is available in (Dramsch, 2020a).

These examples illustrate the open source revolution in machine learning software. The consolidated API and utility functions make it seem trivial to apply various machine learning algorithms to scientific data. This can be seen in the recent explosion of publications of applied machine learning in geoscience. The need to be able to implement algorithms has been replaced by merely installing a package and calling `model.fit(X, y)`. These developments call for strong validation requirements of models to ensure valid, reproducible, and scientific results. Without this careful validation these modern day tools can be severely misused to oversell results and even come to incorrect conclusions.

In aggregate, modern-day neural networks benefit from the development of non-saturating non-linear activation functions. The advancements of stochastic gradient descent with Nesterov momentum and the Adam optimizer (following AdaGrad and RMSProp) was essential faster training of deep neural networks. The leverage of graphics hardware available in most high-end desktop computers that is specialized for linear algebra computation, further reduced training times. Finally, open-source software that is well-maintained, tested, and documented with a consistent API made both shallow and deep machine learning accessible to non-experts.

2.2.2.5 Neural Network Architectures

In deep learning, implementation of models is commonly more complicated than understanding the underlying algorithm. Modern deep learning makes use of various recent developments that can be beneficial to the data set it is applied to, without specific implementation details results are often not reproducible. However, the machine learning community has a firm grounding in openness and sharing, which is seen in both publica-

tions and code. New developments are commonly published alongside their open-source code, and frequently with the trained networks on standard benchmark data sets. This facilitates thorough inspection and transferring the new insights to applied tasks such as geoscience. In the following, some relevant neural network architectures and their application are explored.

2.2.2.6 Convolutional Neural Network Architectures

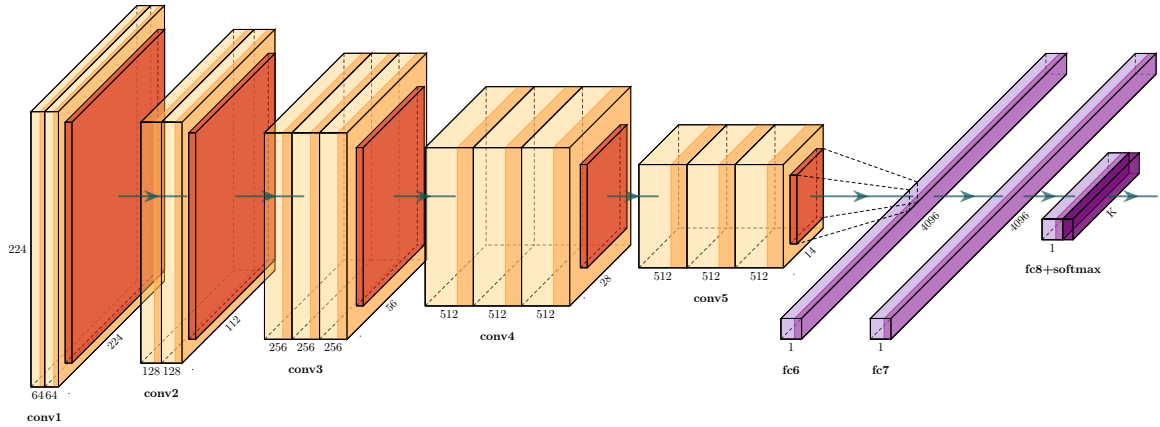


Figure 2.13: Schematic of a VGG16 network for ImageNet. The input data is convolved and down-sampled repeatedly. The final image classification is performed by flattening the image and feeding it to a classic feed-forward densely connected neural network. The 1000 output nodes for the 1000 ImageNet classes are normalized by a final softmax layer (cf. equation 2.11). Visualization library (Iqbal, 2018)

The first model to discuss is the VGG-16 model, a 16-layer deep convolutional neural network (Simonyan et al., 2014a) represented in figure 2.13. This network was an attempt at building even deeper networks and uses small 3×3 convolutional filters in the network, called f in equation 2.12. This small filter-size was sufficient to build powerful models that abstract the information from layer to deeper layer, which is easy to visualize and generalize well. The trained model on natural images also transfers well to other domains like seismic interpretation (Dramschi et al., 2018d). Later, the concept of Network-in-Network was introduced, which suggested defined sub-networks or blocks in the larger network structure (Lin et al., 2013). The ResNet architecture uses this concept of blocks to define residual blocks. These use a shortcut around a convolutional block (He et al., 2016) to achieve neural networks with up to 152 layers that still generalize well. ResNets and residual blocks, in particular, are very popular in modern architectures including the shortcuts or skip connections they popularized, to address the following problem:

When deeper networks start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which

might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error.

He et al. (2016)

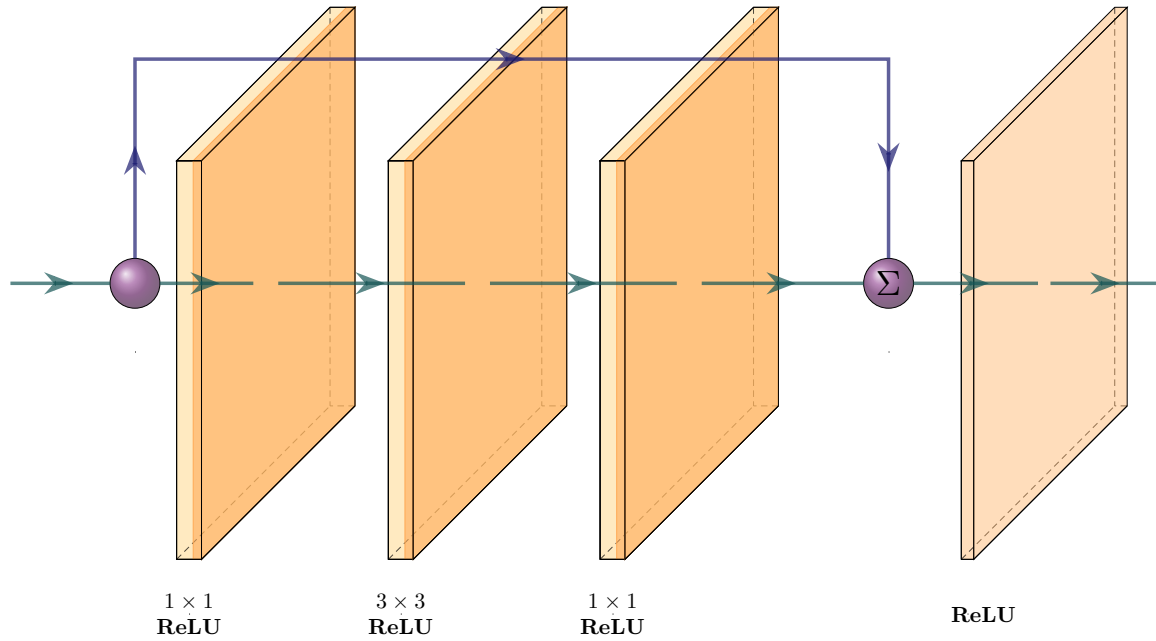


Figure 2.14: Schematic of a ResNet block. The block contains a 1×1 , 3×3 , and 1×1 convolution with ReLU activation. The output is concatenated with the input and passed through another ReLU activation function.

The developments and successes in image classification on benchmark competitions like ImageNet and Pascal-VOC inspired applications in automatic seismic interpretation. These networks are usually single image classifiers using convolutional neural networks (CNNs). The first application of a convolutional neural network to seismic data used a relatively small deep CNN for salt identification (Waldeland et al., 2017). The open source software "MaLenoV" implemented a single image classification network, which was the earliest freely available implementation of deep learning for seismic interpretation (Ildstad et al., 2017). Dramsch et al. (2018d) applied pre-trained VGG-16 and ResNet50 single image seismic interpretation. Recent succesful applications build upon pre-trained pre-built architectures to implement into more sophisticated deep learning systems, e.g. semantic segmentation. Semantic segmentation is important in seismic interpretation. This is already a narrow field of application of machine learning and it can be observed that many early applications focus on sub-subsections of seismic interpretation utilizing these pre-built architectures such as salt detection (Waldeland et al., 2018; Di et al., 2018; Gramstad et al., 2018), fault interpretation (Araya-Polo et al., 2017; Guitton,

2018; Purves et al., 2018), facies classification (Chevitarese et al., 2018; Dramsch et al., 2018d), and horizon picking (Wu et al., 2018). In comparison, this is however, already a broader application than prior machine learning approaches for seismic interpretation that utilized very specific seismic attributes as input to self-organizing maps (SOM) for e.g. sweet spot identification (Guo et al., 2017; Zhao et al., 2017a; Roden et al., 2017).

In geoscience single image classification, as presented in the ImageNet challenge, is less relevant than other applications like image segmentation and time series classification. The developments and insights resulting from the ImageNet challenge were, however, transferred to network architectures that have relevance in machine learning for science. Fully convolutional networks are a way to better achieve image segmentation. A particularly useful implementation, the U-net, was first introduced in biomedical image segmentation, a discipline notorious for small datasets (Ronneberger et al., 2015a). The U-net architecture shown in Figure 2.15 utilizes several shortcuts in an encoder-decoder architecture to achieve stable segmentation results. Shortcuts (or skip connections) are a way in neural networks to combine the original information and the processed information, usually through concatenation or addition. In ResNet blocks this concept is extended to an extreme, where every block in the architecture contains a shortcut between the input and output, as seen in Figure 2.14. These blocks are universally used in many architectures to implement deeper networks, i.e. ResNet-152 with 60 million parameters, with fewer parameters than previous architectures like VGG-16 with 138 million parameters. Essentially, enabling models that are ten times as deep with less than half the parameters, and significantly better accuracy on image benchmark problems.

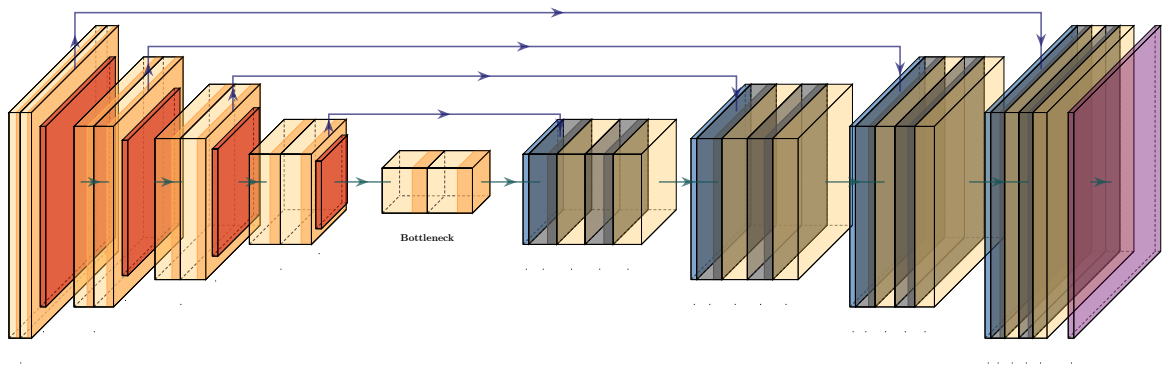


Figure 2.15: Schematic of U-net architecture. Convolutional layers are followed by a downsampling operation in the encoder. The central bottleneck contains a compressed representation of the input data. The decoder contains upsampling operations followed by convolutions. The last layer is commonly a softmax layer to provide classes. Equally sized layers are connected via shortcut connections.

In 2018 the seismic contractor TGS made a seismic interpretation challenge available on the data science competition platform Kaggle. Successful participants in the competition combined ResNet architectures with the U-net architecture as their base architecture and modified these with state-of-the-art image segmentation applications

(Babakhin et al., 2019a). Moreover, Drams et al. (2018d) showed that transferring networks trained on large bodies of natural images to seismic data yields good results on small datasets, which was further confirmed in this competition. The learnings from the TGS Salt Identification challenge have been incorporated in production scale models that perform human-like salt interpretation (Sen et al., 2020). In broader geoscience, U-nets have been used to model global water storage using GRAVE satellite data (Sun et al., 2019), landslide prediction (Hajimoradlou et al., 2019), and earthquake arrival time picking (Zhu et al., 2018). A more classical approach identifies subsea scale worms in hydrothermal vents (Shashidhara et al., 2020), whereas Drams et al. (2019b) includes a U-net in a larger system for unsupervised 3D timeshift extraction from 4D seismic.

This modularity of neural networks can be seen all throughout the research and application of deep learning. New insights can be incorporated into existing architectures to enhance their predictive power. This can be in the form of swapping out the activation function σ or including new layers for improvements e.g. regularization with batch normalization (Ioffe et al., 2015). The U-net architecture originally is relatively shallow, but was modified to contain a modified ResNet for the Kaggle salt identification challenge instead (Babakhin et al., 2019a). Overall, serving as examples for the flexibility of neural networks.

2.2.2.7 Generative Adversarial Networks

Generative adversarial networks (GAN) take composition of neural network to another level, where two networks are trained in aggregate to get a desired result. In GANs, a generator network G and a discriminator network D work against each other in the training loop (Goodfellow et al., 2014b). The generator G is set up to generate samples from an input, these were often natural images in early GANs, but has now progressed to anything from time series (Engel et al., 2019) to high-energy physics simulation (Paganini et al., 2018). The discriminator network D attempts to distinguish whether the sample is generated from G i.e. fake or a real image from the training data. Mathematically, this defines a min max game for the value function V of G and D

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))], \quad (2.17)$$

with x representing the data, z is the latent space G draws samples from, and p represents the respective probability distributions. Eventually reaching a Nash equilibrium (Nash, 1951), where neither the generator network G can produce better outputs, nor the discriminator network D can improve its capability to discern between fake and real samples.

Despite how versatile U-nets are, they still need an appropriate defined loss function and labels to build a discriminative model. GANs however, build a generative model that approximates the training sample distribution in the Generator and a discriminative model of the Discriminator modeled dynamically through adversarial training. The Discriminator effectively providing an adversarial loss in a GAN. In addition to providing two models that serve different purposes, learning the training sample distribution with

an adversarial loss makes GANs one of the most versatile models currently discovered. Mosser et al. (2017) were applied GANs early on to geoscience, modeling 3D porous media at the pore scale with a deep convolutional GAN. The authors extended this approach to conditional simulations of oolitic digital rock (Mosser et al., 2018a). Early applications of GANs also included approximating the problem of velocity inversion of seismic data (Mosser et al., 2018c) and generating seismograms (Krischer et al., 2017). Richardson (2018) integrate the Generator of the GAN into full waveform inversion of the scalar wavefield. Alternatively, a Bayesian inversion using the Generator as prior for velocity inversion was introduced in Mosser et al. (2018b). In geomodeling, generation of geological channel models was presented (Chan et al., 2017), which was subsequently extended with the capability to be conditioned on physical measurements (Dupont et al., 2018). Naturally, GANs were applied to the growing field of automatic seismic interpretation (Lu et al., 2018).

2.2.2.8 Recurrent Neural Network Architectures

The final type of architecture applied in geoscience is recurrent neural networks (RNN). In contrast to all previous architectures, recurrent neural networks feed back into themselves. There are many types of RNNs, Hopfield networks being one that were applied to seismic source wavelet prediction (Wang et al., 1992) early on. However, LSTMs (Hochreiter et al., 1997) are the main application in geoscience and wider machine learning. This type of network achieves state-of-the-art performance on sequential data like language tasks and time series applications. LSTMs solve some common problems of RNNs by implementing specific gates that regulate information flow in an LSTM cell, namely, input gate, forget gate, and output gate, visualized in Figure 2.16. The input gate feeds input values to the internal cell. The forget gate overwrites the previous state. Finally, the output gate regulates the direct contribution of the input value to the output value combined with the internal state of the cell. Additionally, a peephole functionality helps with the training that serves as a shortcut between inputs and gates.

A classic application of LSTMs is text analysis and natural language understanding, which has been applied to geological relation extraction from unstructured text documents (Luo et al., 2017; Blondelle et al., 2017). Due to the nature of LSTMs being suited for time series data, it has been applied to seismological event classification of volcanic activity Titos et al., 2018, multi-factor landslide displacement prediction (Xie et al., 2019a), and hydrological modelling (Kratzert et al., 2019). Talarico et al. (2019) applied LSTM to model sedimentological sequences and compared the model to baseline Hidden Markov Model (HMM), concluding that RNNs outperform HMMs based on first-order Markov chains, while higher order Markov chains were too complex to calibrate satisfactorily. Gated Recurrent Unit (GRU) (Cho et al., 2014) is another RNN developed based on the insights into LSTM, which was applied to predict petrophysical properties from seismic data (Alfarraj et al., 2019).

The scope of this review only allowed for a broad overview of types of networks, that were successfully applied to geoscience. Many more specific architectures exist and are

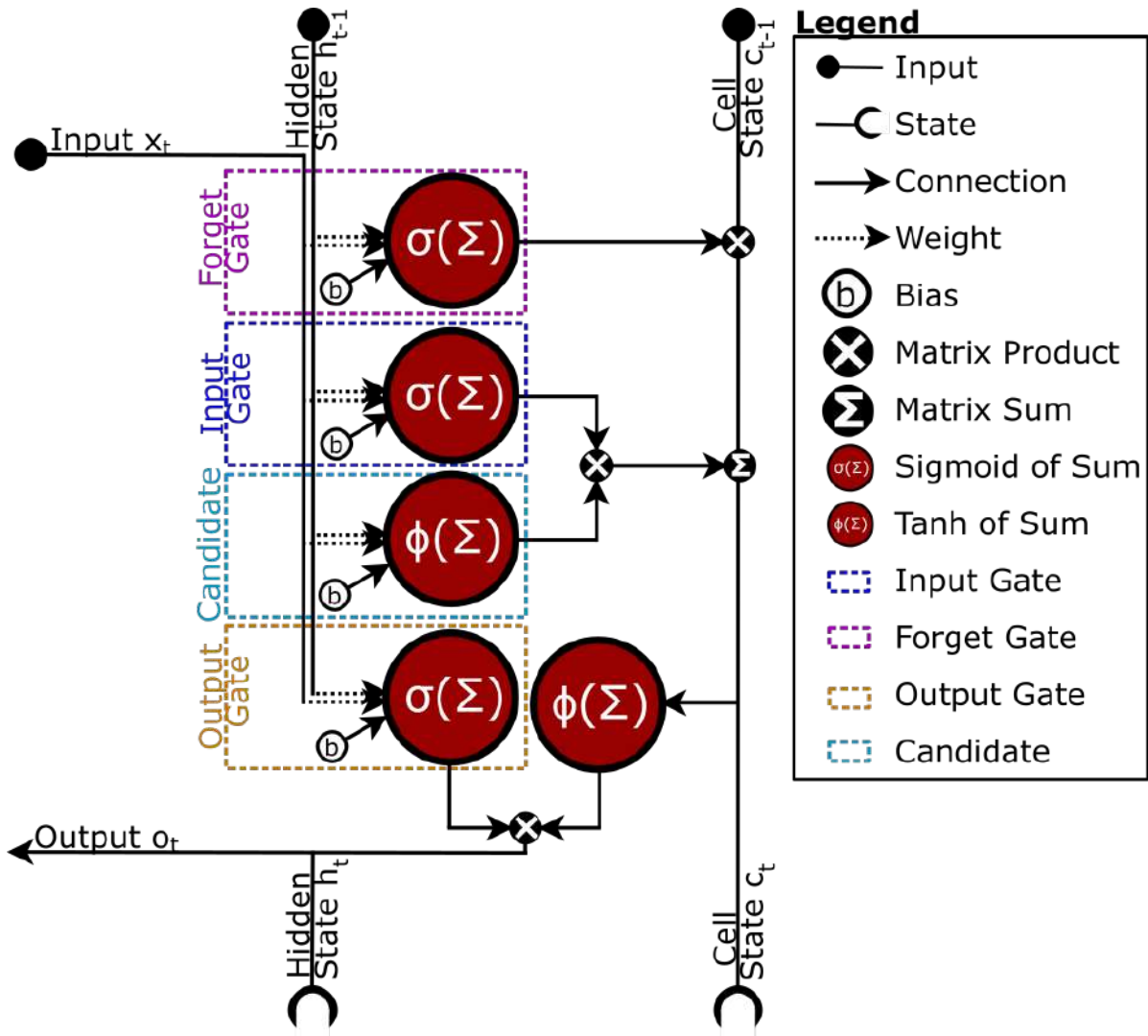


Figure 2.16: Schematic of LSTM architecture. The input data is processed together with the hidden state and cell state. The LSTM avoid the exploding gradient problem by implemented a input, forget, and output gate.

in development that provide different advantages. Siamese networks for one-shot image analysis (Koch et al., 2015), transformer networks that largely replaced LSTM and GRU in language modelling (Vaswani et al., 2017), or attention as a general mechanism in deep neural networks (Zheng et al., 2017).

Neural network architectures have been modified and applied to diverse problems in geoscience. Every architecture type is particularly suited to certain data types that are present in each field of geoscience. However, fields with data present in machine-readable format experienced accelerated adoption of machine learning tools and applications. For example, Ross et al. (2018a) were able to successfully apply CNNs to seismological phase detection, relying on an extensive catalogue of hand-picked data (Ross et al., 2018a) and consequently generalize this work (Ross et al., 2018b). It has to be noted that synthetic or specifically sampled data can introduce an implicit bias into the network (Wirgin, 2004; Kim et al., 2019). Nevertheless, particularly this blackbox property of machine learning model makes them versatile and powerful tools that were leveraged in every subdiscipline of the Earth sciences.

2.2.2.9 The State of ML on Geoscience

Overall, geoscience and especially geophysics has followed developments in machine learning closely. Across disciplines, machine learning methods have been applied to various problems that can generally be categorized into three subsections:

1. Build a surrogate ML model of a well-understood process. This model usually provides an advantage in computational cost.
2. Build an ML model of a task previously only possible with human interaction, interpretation, or knowledge and experience.
3. Build a novel ML model that performs a task that was previously not possible.

Granulometry on SEM images is an example of an application in category I, where previously sediments were hand-measured in images (Dramsche et al., 2018a). Applying large deformation diffeomorphic mapping of seismic data was computationally infeasible for matching 4D seismic data, however, made feasible by applying a U-net architecture to the problem of category II (Dramsche et al., 2019b). The problem of earthquake magnitude prediction falls into category III due to the complexity of the system but was nevertheless approached with neural networks (Panakkat et al., 2007).

The accessibility of tools, knowledge, and compute make this cycle of machine learning enthusiasm unique, with regard to previous decades. This unprecedented access to tools makes the application of machine learning algorithms to any problem possible, where data is available. The bibliometrics of machine learning in geoscience, shown in figure 2.17 serve as a proxy for increased access. These papers include varying degrees of depth in application and model validation. One of the primary influences for the current increase in publications are new fields such as automatic seismic interpretation, as well as, publications soliciting and encouraging machine learning publications. Computer

vision models were relatively straight forward to transfer to seismic interpretation tasks, with papers in this sub-sub-field ranging from single 2D line salt identification models with limited validation to 3D multi-facies interpretation with validation on a separate geographic area.

Geoscientific publishing can be challenging to navigate with respect to machine learning. While papers investigating the theoretical fundamentals of machine learning in geoscience exist, it is clear that the overwhelming majority of papers present applications of ML to geoscientific problems. It is complex to evaluate whether a paper is a case study or a methodological paper with an exemplary application to a specific data set. Despite the difficulty of most thorough applications of ML, "idea papers" exist that simply present an established algorithm to a problem in geoscience without a specific implementation or addressing the possible caveats. On the flip-side, some papers apply machine learning algorithms as pure regression models without the aim to generalize the model to other data. Unfortunately, this makes meta-analysis articles difficult to impossible. This kind of meta-analysis article, is commonly done in medicine and considered a gold-standard study, and would greatly benefit the geoscientific community to determine the efficacy of algorithms on sets of similar problems.

Machine Learning Papers in Geoscience

A sample of 242 papers published before 2018

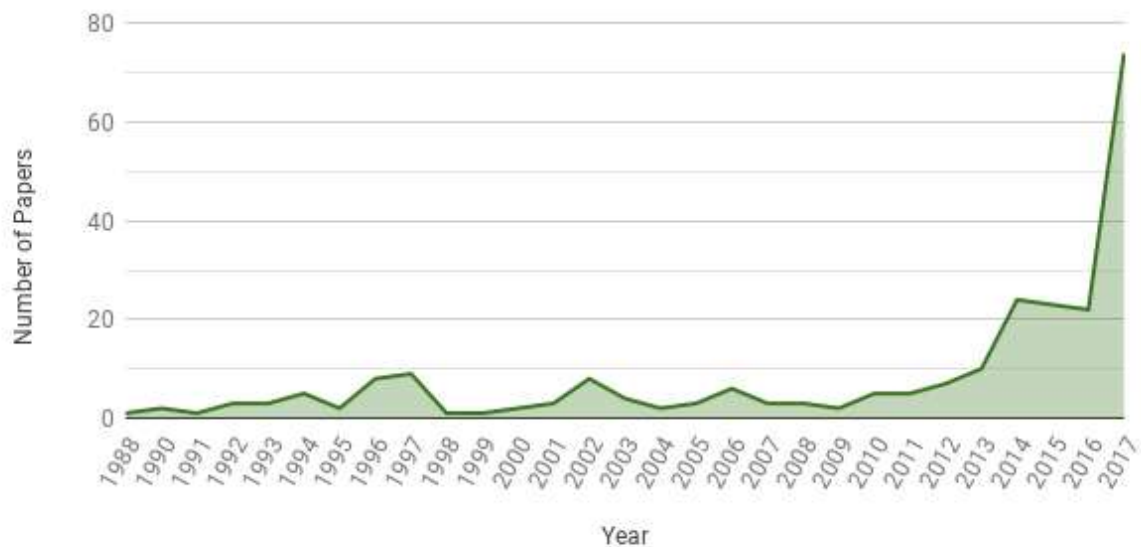


Figure 2.17: Bibliometry of 242 papers in Machine Learning for Geoscience per year. Search terms include variations of machine learning terms and geoscientific subdisciplines but exclude remote sensing and kriging.

Analogous to the medical field, obtaining accurate ground truth data, is often impossible and usually expensive. Geological ground truth data for seismic data is usually

obtained through expert interpreters. Quantifying the uncertainty of these interpretations is an active field of research, which suggest a broader set of experiences and a diverse set of sources of information for interpretation facilitate correct geological interpretation between interpreters (Bond et al., 2007). Radiologists tasked to interpret x-ray images showed similar decreases in both inter- and intra-interpreter error rate with more diverse data sources Jewett et al., 1992. These uncertainties in the training labels are commonly known as "label noise" and can be a detriment to building accurate and generalizable machine learning models. A significant portion of data in geoscience, however, is not machine learning ready. Actual ground truth data from drilling reports is often locked away in running text reports, sometimes in scanned PDFs. Data is often siloed and most likely proprietary. Sometimes the amount of samples to process is so large that many insights are yet to be made from samples in core stores or the storage rooms of museums. Benchmark models are either non-existent or made by consortia that only provide access to their members. Academic data is usually only available within academic groups for competitive advantage, respect for the amount of work, and fear of being exposed to legal and public repercussions. These problems are currently addressed by a culture change. Nevertheless, liberating data will be a significant investment, regardless of who will work on it and a slow culture change can be observed already.

Generally, machine learning has seen the fastest successes in domains where decisions are cheap (e.g. click advertising), data is readily available (e.g. online shops), and the environment is simple (e.g. games) or unconstrained (e.g. image generation). Geoscience generally is at the opposite of this spectrum. Decisions are expensive, be it drilling new wells or assessing geohazards. Data is expensive, sparse, and noisy. The environment is heterogeneous and constrained by physical limitations. Therefore, solving problems like automatic seismic interpretation see a surge of activity having fewer constraints initially. Problems like inversion have solutions that are verifiably wrong due to physics. These constraints do not prohibit machine learning applications in geoscience. However, most successes are seen in close collaboration with subject matter experts. Moreover, model explainability becomes essential in the geoscience domain. While not being a strict equivalency, simpler models are usually easier to interpret, especially regarding failure modes.

A prominent example of "excessive" (Mignan et al., 2019a) model complexity was presented in DeVries et al. (2018) applying deep learning to aftershock prediction. Independent data scientists identified methodological errors, including data leakage from the train set to the test set used to present results (Shah et al., 2019). Moreover, Mignan et al. (2019b) showed that using the central physical interpretation of the deep learning model, using the von Mises yield criterion, could be used to build a surrogate logistic regression. The resulting surrogate or baseline model outperforms the deep network and overfits less. Moreover, replacing the ~13,000 parameter model with the two-parameter baseline model increases calculation speed, which is essential in aftershock forecasting and disaster response³. More generally, this is an example where data science practices

³All authors point out the potential in deep and machine learning research in geoscience regardless

such as model validation, baseline models, and preventing data leakage and overfitting become increasingly important when the tools of applying machine learning become readily available.

Despite potential setbacks and the field of deep learning and data science being relatively young, they can rely on mathematical and statistical foundations and make significant contributions to science and society. Machine learning systems have contributed to modelling the protein structure of the current pandemic virus COVID-19 (Jumper et al., 2020). A deep learning computer vision system was built to stabilize food safety by identifying Cassava plant disease on offline mobile devices (Ramcharan et al., 2017; Ramcharan et al., 2019). Self-driving cars have become a possibility (Bojarski et al., 2016) and natural language understanding has progressed significantly (Devlin et al., 2018).

Geoscience is slower in the adoption of machine learning, compared to other disciplines. To be able to adapt the progress in machine learning research, many valuable data sources have to be made machine-readable. There has already been a change in making computer code open source, which has lead to collaborations and accelerating scientific progress. While specific open benchmark data sets have been tantamount to the progress in machine learning, it is questionable whether these would be beneficial to machine learning in geoscience. The problems are often very complex with non-unique explanations and solutions, which historically has lead to disagreements over geophysical benchmark data sets. Open data and open-source software, however, have and will play a significant role in advancing the field. Examples of this include basic utility function to load geoscientific data (Kvalsvik et al., 2019) or more specifically cross-validation functions tailored to geoscience (Uieda, 2018).

Moreover, machine learning is fundamentally conservative, training on available data. This bias of data collection will influence the ability to generate new insights in all areas of geoscience. Machine learning in geoscience may be able to generate insights and establish relationships in existing data. Entirely new insights from previously unseen or analysis of particularly complex models will still be a task performed by trained geoscientists. Transfer learning is an active field of machine learning research, that geoscience can significantly benefit from. However, no significant headway has been made to transfer trained machine learning models to out-of-distribution data, i.e. data that is conceptually similar but explicitly different from the training data set. The fields of self-supervised learning, including reinforcement learning that can learn by exploration, may be able to approach some of these problems. They are, however, notoriously hard to set up and train, necessitating significant expertise in machine learning.

Large portions of publications are concerned with weakly or unconstrained predictions such as seismic interpretation and other applications that perform image recognition on SEM or core photography. These methods will continue to improve by implementing algorithmic improvements from machine learning research, specialized data augmentation strategies, and more diverse training data being available. New techniques such as multi-task learning (Kendall et al., 2018) which improved computer vision and

and do not wish to stifle such research.(Shah et al., 2019; Mignan et al., 2019b)

computer linguistic models, deep bayesian networks (Mosser et al., 2019) to obtain uncertainties, noisy teacher-student networks (Xie et al., 2019b) to improve training, and transformer networks (Graves, 2012) for time series processing, will significantly improve applications in geoscience. For example, automated seismic interpretation may advance to provide reliable outputs for relatively difficult geological regimes beyond existing solutions. Success will be reliant on interdisciplinary teams that can discern why geologically specific faults are important to interpret, while others would be ignored in manual interpretations, to encode geological understanding in automatic interpretation systems.

Currently, the most successful applications of machine learning and deep learning, tie into existing workflows to automate sub-tasks in a grander system. These models are highly specific, and their predictive capability does not resemble an artificial intelligence or attempt to do so. Mathematical constraints and existing theory in other applied fields, especially neuroscience, were able to generate insights into deep learning and geoscience has the opportunity to develop significant contributions to the area of machine learning, considering their unique problem set of heterogeneity, varying scales and non-unique solutions. This has already taken place with the wider adoption of "kriging" or more generally Gaussian processes into machine learning. Moreover, known applications of signal theory and information theory employed in geophysics are equally applicable in machine learning, with examples utilizing complex-valued neural networks (Trabelsi et al., 2017), deep Kalman filters (Krishnan et al., 2015), and Fourier analysis (Tancik et al., 2020). Therefore, possibly enabling additional insights, particularly when integrated with deep learning, due to its modularity and versatility.

Previous reservations about neural networks included the difficulty of implementation and susceptibility to noise in addition to computational costs. Research into updating trained models and saving the optimizer state with the model has in part alleviated the cost of re-training existing models. Moreover, fine-tuning pre-trained large complex models to specific problems has proven successful in several domains. Regularization techniques and noise modelling, as well as data cleaning pipelines, can be implemented to lessen the impact of noise on machine learning models. Specific types of noise can be attenuated or even used as an additional source of information. The aforementioned concerns have mainly transitioned into a critique about overly complex models that overfit the training data and are not interpretable. Modern software makes very sophisticated machine learning models, and data pipelines available to researchers, which has, in turn, increased the importance to control for data leakage and perform thorough model validation.

Currently, machine learning for science primarily relies on the emerging field of explainability (Lundberg et al., 2018). These provide primarily post-hoc explanations for predictions from models. This field is particularly important to evaluate which inputs from the data have the strongest influence on the prediction result. The major point of critique regarding post-hoc explanations is that these methods attempt to explain how the algorithm reached a wrong prediction with equal confidence. Bayesian neural networks intend to address this issue by providing confidence intervals for the prediction based on prior beliefs. These neural networks intend to incorporate prior expert

knowledge into neural networks, which can be beneficial in geoscientific applications, where strong priors can be necessary. Machine learning interpretability attempts to impose constraints on the machine learning models to make the model itself explainable. Closely related to these topics is the statistics field of causal inference. Causal inference attempts to model the cause of variable, instead of correlative prediction. Some methods exist that can perform causal machine learning, i.e. causal trees (Athey et al., 2016). These three fields will be necessary to glean verifiable scientific insights from machine learning in geoscience. They are active fields of research and more involved to correctly apply, which often makes cooperation with a statistician necessary.

In conclusion, machine learning has had a long history in geoscience. Kriging has progressed into more general machine learning methods, and geoscience has made significant progress applying deep learning. Applying deep convolutional networks to automatic seismic interpretation has progressed these methods beyond what was possible, albeit still being an active field of research. Using modern tools, composing custom neural networks, and conventional machine learning pipelines has become increasingly trivial, enabling wide-spread applications in every sub-field of geoscience. Nevertheless, it is important to acknowledge the limitations of machine learning in geoscience. Machine learning methods are often cutting-edge technology, yet properly validated models take time to develop, which is often perceived as inconvenient when working in a hot scientific field. Despite being cutting edge, it is important to acknowledge that none of these applications are fully automated, as would be suggested by the lure of artificial intelligence. Nevertheless, within applied geoscience, significant new insights have been presented. Applications in geoscience are using machine learning as a utility for data pre-processing, implementing previous insights beyond the theory and synthetic cases, or the model itself enabling unprecedented applications in geoscience. Overall, applied machine learning has matured into an established tool in computational geoscience and has the potential to provide further insights into the theory of geoscience itself.

2.3 Contributions of this Study

This chapter provides the basic principles in 4D seismic and an overview of Machine Learning in geoscience in the last 70 years. This lays the foundation for the applications outlined in the following chapters that use Convolutional Neural Networks on seismic data, as well as Deep Neural Networks on seismic maps. Specialized theory and methods are introduced in their respective chapters. The work in this chapter resulted in a review book chapter (Dramsch, 2020d) with the code available in (Dramsch, 2020e).

CHAPTER 3

Unsupervised Geological Image Segmentation

Analysis of chalk samples can be done by Backscatter Scanning-Electron Microscopy. The resulting images contain black-white images of the sample that can be measured and analysed. The back-scatter measurements add information regarding the chemical composition from the back-scattered energy from the electron beam. Research into the porosity and sedimentology of the chalk reservoirs conducted using electron microscopes. Identifying the grain size and orientation of the oolites is usually a manual work-intensive task, ideal for computer vision tasks, considering the good contrast of light-grey to white oolites and the black background. Figure 3.1 shows a chalk sample from the analysis. The chalk grains are of varying size, with inter- and intra-grain porosity. The intra-grain porosity is best seen in the grain located at (1000, 200) in Figure 3.1. Backscatter Scanning-Electron Microscopy (BSEM) data is noisy, which can distort the image. Additionally, grain boundaries tend to be jagged, which is aggravated by the noise.

3.1 Unsupervised Image Segmentation

Labelled training data was not available to apply Convolutional Neural Networks to this problem. Instead of hand-labelling the data, unsupervised clustering was appropriate to find the optimal boundary of the grains from the background. Gaussian mixture models (GMMs) learned a two-fold representation that separated the background well from the rock. Clustering the pixel intensity into two clusters with the GMM and classical histogram boundary are displayed in Figure 3.2. These

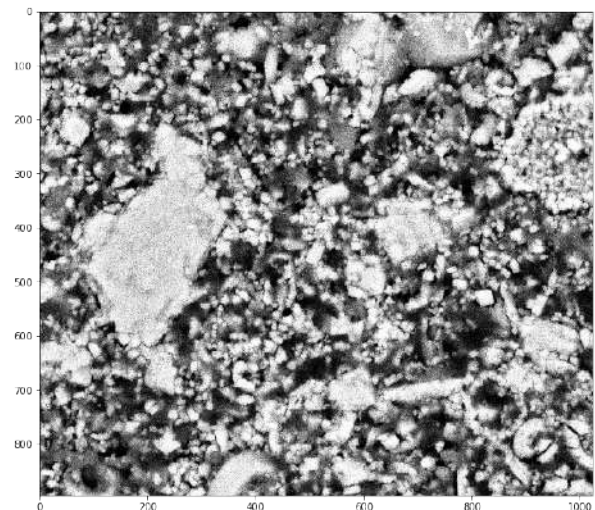


Figure 3.1: Backscatter Scanning-Electron Microscopy of chalk sample.

boundaries do not coincide, and while they are close, the two-pixel difference changes the prediction significantly.

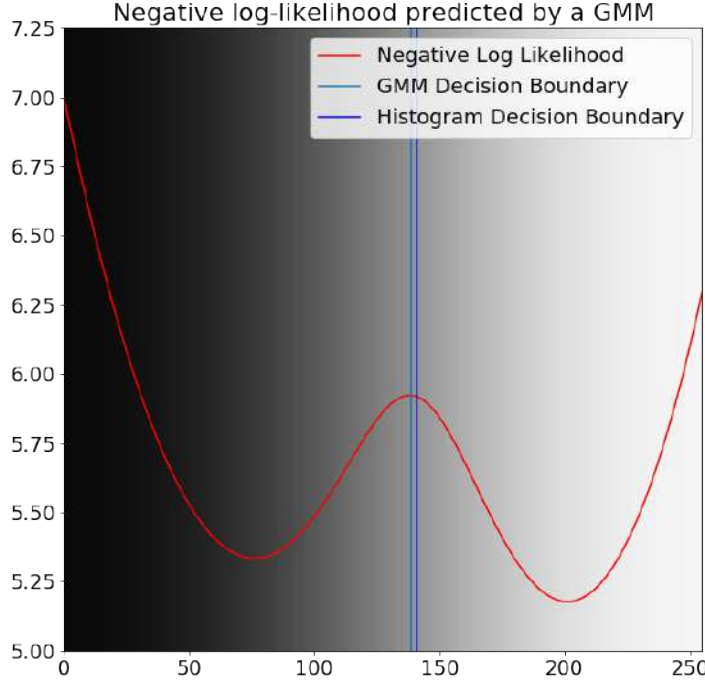


Figure 3.2: Greyscale values overlaid with negative log likelihood predicted by Gaussian mixture model (red) with decision boundaries from GMM (green) and histogram decision boundary (blue).

3.1.1 Gaussian Mixture Models

In detail, the Gaussian mixture model can model normally distributed sub-populations that constitute a larger population. The Gaussian distributions are usually estimated by using the maximum likelihood method. Because differentiating the log-likelihood is computationally infeasible, commonly, the expectation maximisation (EM) is utilised to approximate the maximum log-likelihood. EM is an iterative approach that calculates the expectation of a sample to be part of a component and maximisation, which updates the model parameters.

We define $\mathbf{x} = (x_1, x_2, \dots, x_n)$ as the data points with n independent observations. Then $\mathbf{z} = (z_1, z_2, \dots, z_n)$ is the latent vector and $X_i | (Z_i = k) \propto \mathcal{N}_d(\mu_k, \Sigma_k)$ for k components and a d -dimensional Gaussian. Then the distribution

$$p(\mathbf{x}) = \sum_{i=1}^k \phi_i \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i),$$

where ϕ_i are the model weights and normalize to

$$\sum_{i=1}^K \phi_i = 1.$$

The Expectation step calculates the membership probabilities for all samples i and components k :

$$\hat{\gamma}_{ik} = \frac{\hat{\phi}_k \mathcal{N}(x_i | \hat{\mu}_k, \hat{\sigma}_k)}{\sum_{j=1}^K \hat{\phi}_j \mathcal{N}(x_i | \hat{\mu}_j, \hat{\sigma}_j)} \quad (3.1)$$

then the estimated $\hat{\gamma}_{ik}$ describes the probability that a sample x_i is generated by the k -th component C_k , leading to the conditional probability $\hat{\gamma}_{ik} = p(C_k | x_i, \hat{\phi}, \hat{\mu}, \hat{\sigma})$.

The Maximization step then updates the parameter set $(\hat{\phi}, \hat{\mu}, \hat{\sigma})$ for all k :

$$\hat{\phi}_k = \sum_{i=1}^N \frac{\hat{\gamma}_{ik}}{N} \quad (3.2)$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^N \hat{\gamma}_{ik} x_i}{\sum_{i=1}^N \hat{\gamma}_{ik}} \quad (3.3)$$

$$\hat{\sigma}_k^2 = \frac{\sum_{i=1}^N \hat{\gamma}_{ik} (x_i - \hat{\mu}_k)^2}{\sum_{i=1}^N \hat{\gamma}_{ik}} \quad (3.4)$$

The parameters are initialized by a naive strategy, so that all samples in \mathbf{x} get randomly assigned to a component and the weights are uniformly distributed as $\phi_1, \phi_2, \dots, \phi_K = \frac{1}{K}$. The iteration is stopped when the model converges, that is the expectation of subsequent steps changes less than a pre-set ϵ .

3.1.2 Morphological Filtering

This single-valued intensity value causes the boundary to be non-smooth around the grains. Therefore morphological filtering was applied to smooth out the boundaries programmatically. Smooth boundaries are essential for chalk grains, as the perimeter of the oolites can be used to calculate the specific surface of chalk. The optimal boundary of chalk grains could then be used to generate training data for more sophisticated machine learning systems.

Mathematical morphology is the theory of analysing geometrical structures. Morphological filtering implements multiple shape-based filters that perform non-linear transformations. In this case, we apply morphological dilation and morphological erosion. Morphological erosion set the central pixel to the minimum of a neighbourhood. More formally A is a binary image and B is a structuring element on the Euclidean space E ,

e.g. a 3×3 square, or a disk with defined radius, then $A \ominus B = \{z \in E | B_z \subseteq A\}$, where B_z is the translation of B by the vector z . More concisely morphological erosion is:

$$A \ominus B = \bigcap_{b \in B} A_{-b} \quad (3.5)$$

The morphological dilation, in turn, sets a central pixel to the maximum of the neighbourhood of pixels in an image. Then

$$A \oplus B = \bigcup_{b \in B} A_b \quad (3.6)$$

These operations can then be combined to perform morphological opening, which is commonly used to "clean up" edges in binary images. Morphological closing can be written as

$$A \circ B = (A \ominus B) \oplus B. \quad (3.7)$$

Repeating erosion and dilation alternatingly smoothes out the boundary we obtain from Gaussian mixture model.

3.2 Workshop Paper: Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk

Published in the First EAGE/PESGB Workshop on Machine Learning.

Abstract: Scanning-Electron images from North Sea Chalk are studied for important rock properties. To relieve this manual labor, we investigated several standard image processing methods that underperformed on complicated chalk. Due to the lack of manually labeled data, deep neural networks could not be adequately applied. Gaussian Mixture Models learnt a two-fold representation that separated the background well from the rock. Subsequent morphological filtering cleans up the prediction and enables automatic analysis.

J. S. Dramsch, F. Amour, and M. L  thje (2018a). “Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk”. In: *First EAGE/PESGB Workshop Machine Learning. Published, Chapter 3.* EAGE. DOI: 10.3997/2214-4609.201803014. URL: <https://doi.org/10.3997/2214-4609.201803014>

3.2.1 Introduction

In the oil and gas industry, assessment and prediction of the hydrocarbon reserves and flow properties throughout a chalk reservoir lifetime relies, among others, on conventional and special core analysis (CCAL and SCAL) and computed tomography (CT) imaging in order to characterise the petrophysical properties and 3-D pore network geometry of chalk.

The latter laboratory experiments are technically challenging, costly, and time-consuming and require a large amount of core material. Various image analysis techniques, studying the 2-D distribution of grains, pores, and pore throats on thin-sections, have been extensively tested over more than 50yrs for workflow optimization.

Nevertheless, such techniques have not yet been integrated by reservoir engineers and geoscientists as a routine task during reservoir characterization, especially, due to a limited number of samples tested or a spatially-restricted study area that do not allow the results to be statistically representative of the chalk heterogeneity across a reservoir and between oil and gas fields.

Back-scattered electron microscopy (BSEM) analysis historically has been very manual work. Separating grains from the background, measuring perimeter and area of the grains. Recently, publications showed automatic segmentation of BSEM images using

computational methods. The present study represents a robust method in the application of machine learning on thin-section images collected by BSEM. This cheap and relatively rapid technique allows to quickly analyse a large number of pictures that do not need to be manually labeled.

3.2.2 Method

3.2.2.1 SEM Analysis as Image Segmentation

Scanning Electron Microscopy (SEM) is an imaging method that allows the visualisation of the grains and pores of chalk deposits (Figure 3.3). Grayscale images of the rock fabric can be collected at various scales of observation, from the micro-scale, typically single pore and grain, to few tens of microns where the network of pores can be studied, to the millimetres-scale. This provides a complete insight of the heterogeneity of each sample. Nanotube SEM and many applications separate very well the grains from the background in the SEM images. Therefore, these images can be segmented by histogram methods. Carbonates and specifically chalk vary on grayscale, and grains are not illuminated homogeneously. However, image segmentation has made many improvements in recent years, which extends the toolkit beyond histogram segmentation.

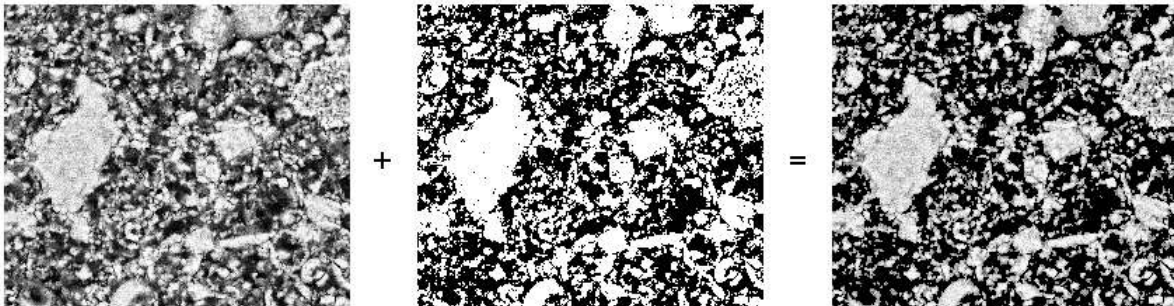


Figure 3.3: Original SEM image, binary mask obtained by GMM, and resulting grain image.

Modern Neural Networks (NN) can segment images exceptionally well (Ronneberger et al., 2015b). Modarres et al. (2017) investigated the application of NNs to SEM images. However, as with most applications in Geoscience and supervised learning, we would have to label a significant amount of images by hand to assure quality or automatically with subpar methods to train the network adequately. This defeats the point for this application, therefore, this study investigates unsupervised methods, which will be assessed in order to select the one that performs the best across all scales of observation. Several BSEM images of the rock fabric at the same scale are also collected to validate the results.

Gaussian Mixture Model (GMM) learns a number of joint distributions approximated by Gaussians in the search space (Lindsay, 1995). The number of Gaussians has to be specified, similar to many clustering methods, like k-means. In this application, we aim

at segmenting the background from the chalk, which lends itself to specify two Gaussian distributions as learning parameter to obtain a binary mask, presented in Figure 3.3.

3.2.2.2 Morphological Filtering

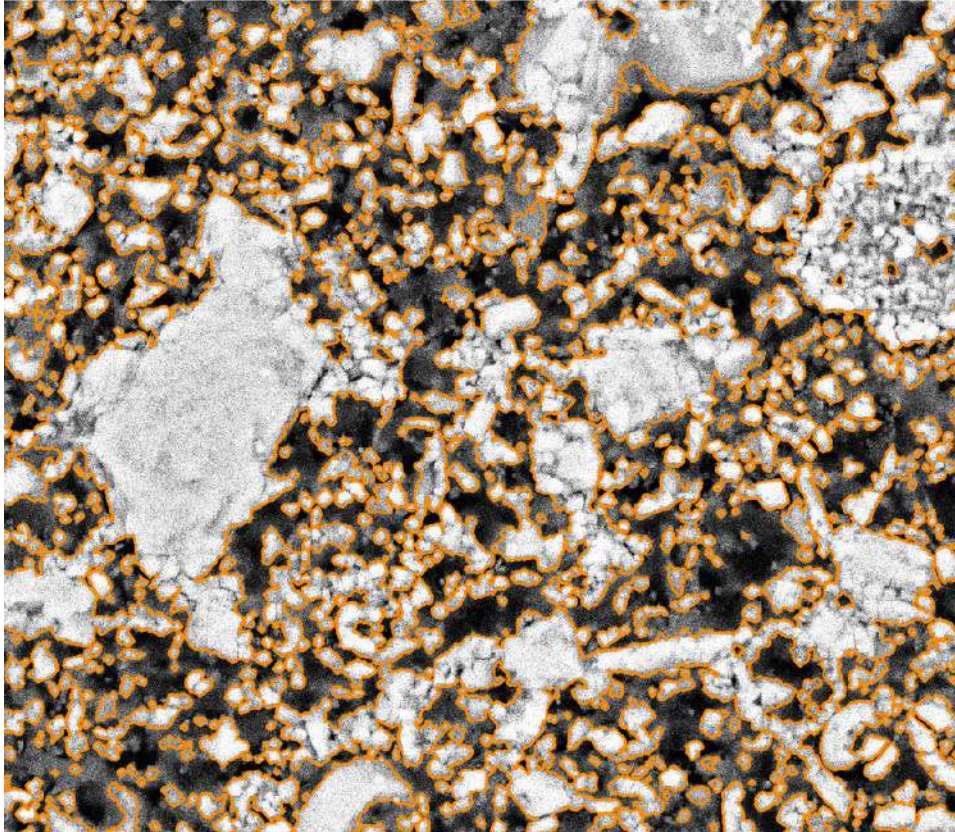


Figure 3.4: Filtered segmentation of BSEM

We apply morphological filtering to clean up the segmentation (Serra et al., 1992). Due to the noisy images of BSEM, the edges of grains appear fuzzy. For the automatic analysis of the perimeter for instance, seen in Figure 3.4.

Subsequently, we can programmatically analyse the result using scikit-learn and scikit-image (Pedregosa et al., 2011). This provides area, perimeter and rotation of grains in the image among other geometrical factors of the grains. These can be very valuable in digital rock physics and pore analysis.

3.2.3 Conclusions

We present an effective segmentation method for BSEM image data. Gaussian Mixture Models learn a good representation of the grayscale data and morphological filtering further improves the results.

3.2.4 Acknowledgements

The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding program.

3.3 Computational Granulometry

Identifying the grains in an Backscatter Scanning-Electron Microscopy image enables us to perform computational granulometry on the images. The segmented images can be analysed with standard image processing algorithms that can return standard measures such as perimeter, area, and eccentricity of grains. Moreover, depending on the preparation of the chalk sample, the angle of orientation can be extracted for each grain.

Figure 3.5a shows the distribution of grain sizes over the image. The data shows an even distribution toward smaller chalk grains, with three very large samples, that can be clearly identified in Figure 3.6. Figure 3.5b shows the shape distribution of the grains, which is distributed toward less circular grains due to compaction. Nevertheless, there are two strong spikes toward circular grains which is in accordance with our expectation for chalk oolites.

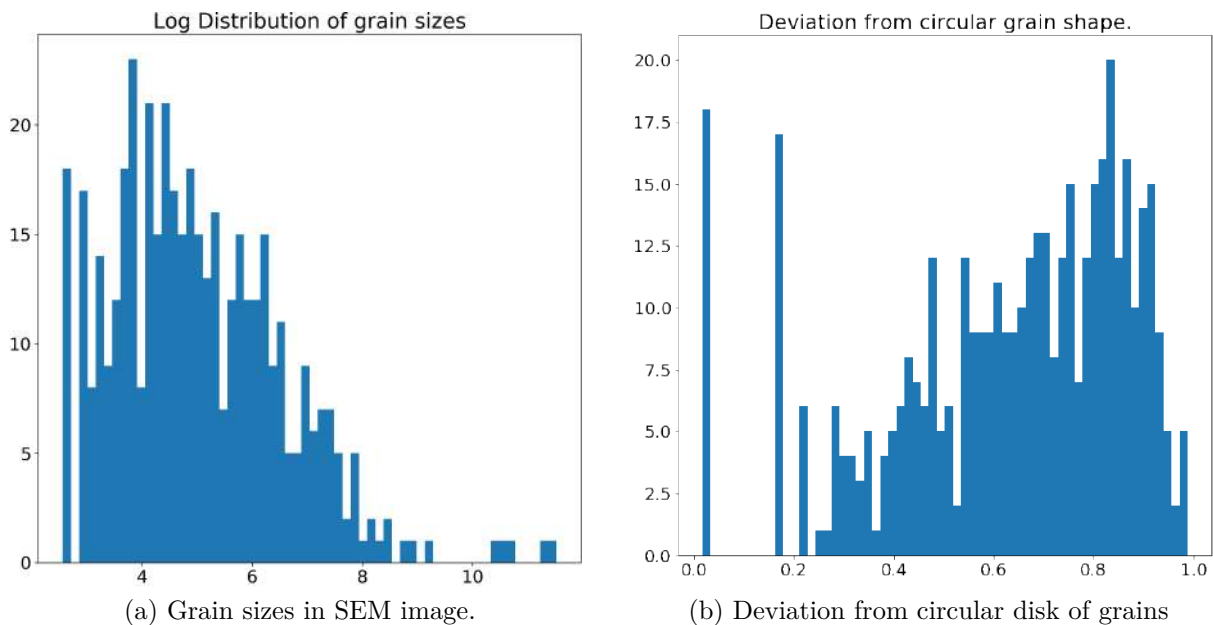


Figure 3.5: Granulometry of chalk in SEM images.

Moreover, this analysis enables us to calculate the approximate porosity from the image. The porosity calculated is 44.25%. The measured porosity of the chalk sample is 42%, which is close for a 2D image of the 3D pore space.

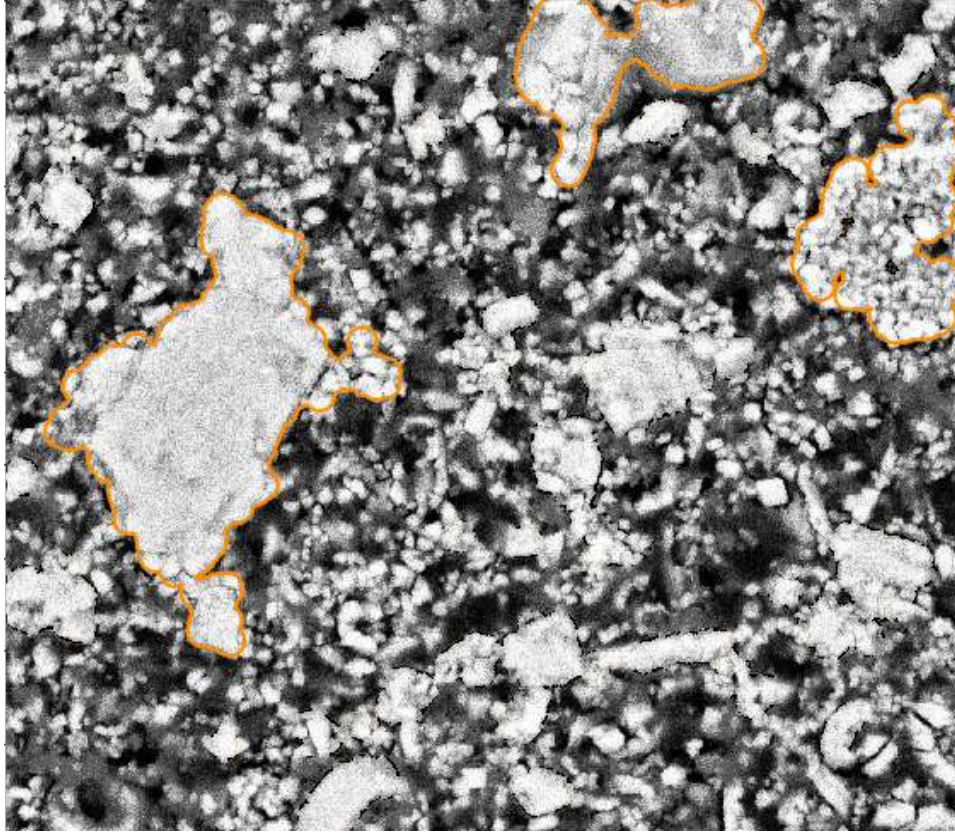


Figure 3.6: Backscatter Scanning-Electron Microscopy data with three large chalk grains outlined from the Gaussian mixture model process (orange). The image shows mostly brecciated chalk grains with some interspersed circular oolites.

3.4 Contributions of this Study

This study introduced unsupervised Gaussian mixture model clustering for chalk grain Backscatter Scanning-Electron Microscopy image segmentation. Overall, the method shows a very good separation of the grains from the background in the image. The method performs well on images with varying lightness, due to the unsupervised nature of the GMM algorithm. This model, however, benefits from the contrast between the light chalk grains and the dark background. Nevertheless, it does outperform classical methods, i.e. a histogram-based analysis.

Morphological filtering improves the segmentation of the image. The morphological filtering application is computationally efficient and reliable in removing small scale variations in the data. The morphological opening smooths the boundaries between the grain and the background and remove small grains and possible noise from the binary labels.

These binary labels enable computational granulometry on the grain data. This data has good accordance with the image data, as well as measured porosity on the rock sample. Finally, this method can be used to generate labels for more complex machine

learning models, i.e. Convolutional Neural Networks.

The code of this analysis is published under J. S. Dramschi (2018b). *Reproducible Code: Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk*. URL: <https://github.com/JesperDramschi/backscatter-sem-segmentation>.

CHAPTER 4

Transfer learning in Automatic Seismic Interpretation

This chapter discusses transfer learning in Automatic Seismic Interpretation. Transfer learning is a technique that uses a Deep Neural Networks pre-trained on a different data set that is usually larger and more diverse, which is then fine-tuned to the target data. Deep Neural Networks are notorious for needing large numbers of diverse annotated samples. That is often prohibitive to geoscience applications of Machine Learning, where data is expensive and difficult to acquire, labelling by experts is complicated and prone to bias (Bond et al., 2007), and often only available within commercial environments. In “Deep-learning seismic facies on state-of-the-art CNN architectures” (Dramsch et al., 2018d) we show that state-of-the-arts Convolutional Neural Networks pre-trained on a natural image data set (ImageNet, cf. Section 2.2.2.4) can be transferred to perform Automatic Seismic Interpretation. This paper forms the central contribution of this chapter.

In the computer vision community, hand-labelled data sets like ImageNet, CIFAR, and PASCAL-VOC are openly available, which catalyzed the development of new architectures and approaches in deep learning. Geoscientific data is often expensive to acquire, and companies are reluctant to make data available, even less so for processed or interpreted data. Early machine learning workshops often showed results on the open Dutch F3 dataset; however, national data repositories have started to change this approach to foster innovation. With data becoming more available recently, the next problem is the lack of ground truth. Obtaining accurate labels for seismic data is impossible, as any inversion process is non-unique and digging is not practical. In other imaging-based fields (e.g. radiology) that rely on the interpretation of imaging results, studies investigate both inter-interpreter variations, by making several interpretations available and intra-interpreter variability by re-interpreting the dataset after a set time interval (McErlean et al., 2013; Alikhassi et al., 2018; Al-Khawari et al., 2010). Additionally, simulations provide ground truth, but can implicitly include modelling assumptions in the data or commit the inverse crime (Wirgin, 2004). The inverse crime presents the problem of modelling and inverting data with the same theoretical ingredients.

In geophysics itself, seismic data presents a unique challenge to computer vision

problems. Displays of seismic data usually clip amplitudes in the 3rd to 5th percentile to make most of the seismic amplitude content visible. These particularly strong amplitudes make up a very small number of the distribution of amplitudes. However, they have to be contained within the constant dynamic range of the data, while adding minimal information gain (Forel et al., 2005). Moreover, limiting these outlier amplitudes decompresses the main distribution of amplitudes over the full dynamic range. This becomes particularly important when compressing data to lower bitrates, i.e. from 32-bit floats to 16-bit floats. Clipping amplitudes has also proven to be a viable preprocessing step before feeding seismic data to computer vision systems, such as convolutional neural networks. Machine learning systems have been known to be vulnerable to noise. This noise can be physical noise (e.g. low Signal-to-Noise Ratio (SNR)) for simpler models or adversarial attacks that reverse engineer more complex models. These adversarial attacks on machine learning models attempt to find vulnerabilities in the trained models intentionally. Frequently, these adversarial attacks can provide insights into edge-behaviours and susceptibility to noise. Adversarial attacks include a one-pixel attack on ImageNet classifiers, which changes a single value in an image to cause a misclassification (Su et al., 2019). Humanly imperceptible noise changes the digital image so slightly that the human eye cannot see a change, but the classifier is led to misclassify the image (Goodfellow et al., 2014a), which is particularly interesting to physical applications of machine learning, that can have significant amounts of noise in their data. Alternatively, even physical printed stickers are used to fool a Convolutional Neural Network in real-world applications (Brown et al., 2017). Besides, geological data contains regions of geological interest and regions that are inconsequential to geological interpretation. This selective interpretation of geological features, which has been common in seismic interpretation, as well as, well-log interpretation is challenging to represent in metrics adequately (Purves et al., 2019).

Realistically, the limited availability of labelled ground truth data can be addressed in different ways. In the case when labels are available but not abundant, transfer learning of highly generalizable models like VGG-16 can be fine-tuned to seismic data. The VGG-16 architecture can also be included in U-Nets as a decoder to leverage the benefits of transfer learning in semantic segmentation tasks (Dramsche et al., 2018d). Moreover, weakly-supervised training can perform label propagation of labelled subsections of the full data set to unlabeled sets. Unsupervised or self-supervised training can be applicable, where no reliable ground truth is available. Unsupervised training is applicable, when a desired operation on the data is known, or an internal structure of the data can be exploited (Dramsche et al., 2019b). Additionally, multi-task learning has been shown to be able to stabilize network performance in Natural Language Processing (Liu et al., 2019b) and Reinforcement Learning (Yu et al., 2019).

Research into deep convolutional networks showed that the data in the network would lose signal with increasing depth, named vanishing gradient problem (Hochreiter, 1998). This vanishing gradient problem led to the limitation of VGG at 19 layers; this is detailed further in Section 2.2.2.6. Residual blocks introduced a solution to this problem by implementing a shortcut between the original data and the output from the block. Figure 2.14 presents the original ResNet block architecture, which was used in ResNet-50

and ResNet-101 in Figure 4.1 (He et al., 2016). Details on ResNet blocks differ, the main take-away being the sum or concatenation of the original data with the block output. DenseNets (Huang et al., 2017) and Inception-style networks (Szegedy et al., 2015) are other approaches to build deeper NNs.

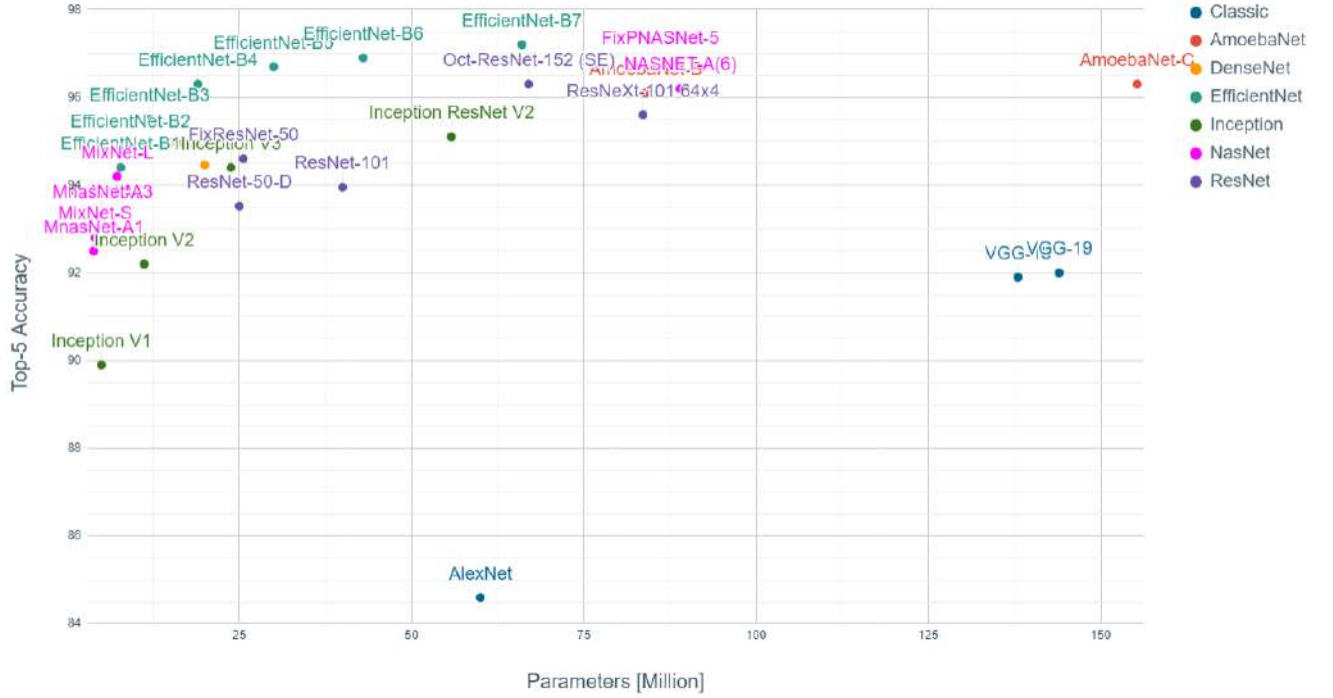


Figure 4.1: Top-5 Accuracies of Neural Architectures on ImageNet plotted against Million Parameters, color-coded to similar network type. Data and references shown in Table A.1

Figure 4.1 additionally contains several classes of Neural Network architectures, namely AmoebaNet, NASNet, and EfficientNet. These categories are a more recent development in neural architecture research, based on Neural Architecture Search (NAS), which automates the search for novel architectures instead of completely hand-tuning new developments. This optimization scheme to search for neural architectures has been developed to include different optimization objectives. The AmoebaNet is based on Evolutionary Computing (EC), a numeric optimization technique mimicking biological evolution, and subsequent fine-tuning of the solution to search for an ideal neural architecture to perform image classification (Real et al., 2019). The NASNet goes on with fixed overall architecture, but uses a controller Recurrent Neural Network (RNN) to modify the blocks within the architecture (Zoph et al., 2018). The EfficientNet architecture was also acquired by NAS, by optimizing for both accuracy and Floating-Point Operations (FLOPs). Optimizing for FLOPs reduces the computational cost of the final architecture (Tan et al., 2019b). Moreover, Tan et al. (2019b) derives a method of simultaneously scaling multiple dimensions in deep neural networks named compound scaling. The standard ResNet-50 and ResNet-101 differ only in-depth, whereas com-

pound scaling establishes a relationship between depth, width and resolution-scaling of deep neural networks using a single scaling parameter.

VGG-16 and ResNet-52 are two network architectures that are used in the paper in this chapter. These can be identified in Figure 4.1. The performance of both models in the Top-5 accuracy on ImageNet is comparable, while the number of parameters vastly differ. VGG-16 contains 138 million parameters, while ResNet-52 contains 23 million parameters, the VGG-16 network is, however, 16 layers deep, while Resnet-52 contains 52 layers. These networks are compared to the end-to-end trained CNN built by Waldeland et al. (2016).

4.1 Training and Fine-Tuning

The training of the three networks in this chapter, namely Waldeland CNN, VGG-16, and Resnet-52, requires different strategies to obtain optimal results. The Waldeland CNN is end-to-end trained on the training data. The VGG-16 and ResNet-52 are fine-tuned with pre-trained weights, which require a lower learning rate and fixing the weights in parts of the network. The networks are trained with the categorical cross-entropy loss discussed in (2.15). The categorical cross-entropy enables training on multi-class labels by optimizing the multi-variate negative log-likelihood. It is reprinted here for convenience:

$$CE = - \sum_j^C y_j \log(o_j)$$

The VGG-16 model has the first seven layers frozen. The ResNet-52 has the first 44 layers frozen. This ensures that the most general features are preserved, while higher abstraction features in layers can be adjusted to the training data. Moreover, the last layer that outputs the classification has to be replaced by an appropriate layer, which instead of predicting 1000 classes for ImageNet, predicts the number of classes in our training set 9.

The training relies on the custom loader presented in Appendix E.2.4. This loader extracts patches from the 2D seismic image and the according label and provides a convenient generator. This generator can perform the data preparation on CPU while the training is performed on GPU. Additionally, the training is monitored to implement an early-stopping procedure. This enables us to stop the training when the validation loss and validation accuracy deteriorate. This avoids overfitting of the network, which is particularly essential when fine-tuning an over-parametrized network to smaller-scale data.

4.1.1 End-to-End CNN training

The training of the Waldeland CNN is trained end-to-end. The optimizer for the Waldeland CNN is the Adam optimizer (Kingma et al., 2014) with a learning rate of 0.001,

the decay of first-order moments of $\beta_1 = 0.9$, and second-order moments of $\beta_2 = 0.999$.

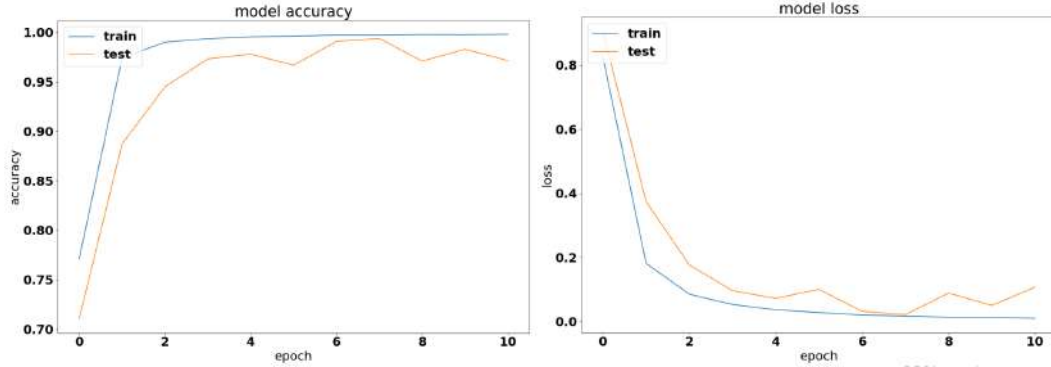


Figure 4.2: Accuracy and Categorical Cross Entropy for Waldeland CNN

Figure 4.2 shows the training loss of end-to-end training. The accuracy shows that the network very quickly reaches 100% accuracy on the training data while performing close to perfect on the test set. The training is stopped after ten epochs. The loss shows that the model starts overfitting at epoch 7. A dataset with more diverse labels and samples would improve this situation.

4.1.2 Fine-Tuning Pre-Trained Networks

Pre-trained networks were trained on a dataset and made available by the researchers and companies, including weights and biases. These are often trained on large corpuses of data. In computer vision, classically pre-trained networks were trained on ImageNet, CIFAR, and PASCAL-VOC. The state-of-the-art (SOTA) networks are pre-trained on up to a billion images with 17,000 labels and subsequently fine-tuned on the ImageNet-1K dataset (Mahajan et al., 2018). This strategy is applied across deep learning, including computational linguistics with 175 billion parameters pre-trained on 0.499 trillion words in GPT-3 (Brown et al., 2020). The pre-trained networks in this chapter were trained on the ImageNet corpus and transferred to the MaleNov seismic dataset (Ildstad et al., 2017).

The VGG-16 and ResNet-52 are finetuned using Stochastic Gradient Descent (SGD) with Nesterov momentum. The learning rate for the SGD is set to 0.0001, with a momentum of 0.9. Additionally, a learning rate schedule is implemented that updates the learning rate (lr) according to $lr(t) = 0.0001 \cdot (1 + 10^{-6} \cdot t)^{-1}$.

The VGG-16 network quickly converges to 100% accuracy, the loss, however smoothly converges towards a cross-entropy of 0.1. The network does not show signs of overfitting and trains the full 20 epochs. With the available hardware at the time of writing the paper and the good results despite possibly increasing the convergence.

The ResNet-52 network immediately reports a training accuracy of close to 100% while the test data report 11% accuracy, which is a performance equivalent to random chance on this dataset containing nine classes. The loss in Figure 4.4 shows the same

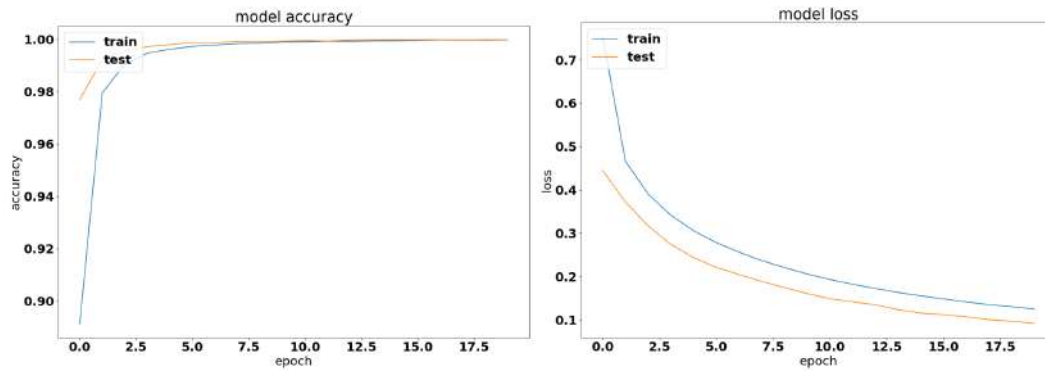


Figure 4.3: Accuracy and Categorical Cross Entropy for VGG16 CNN

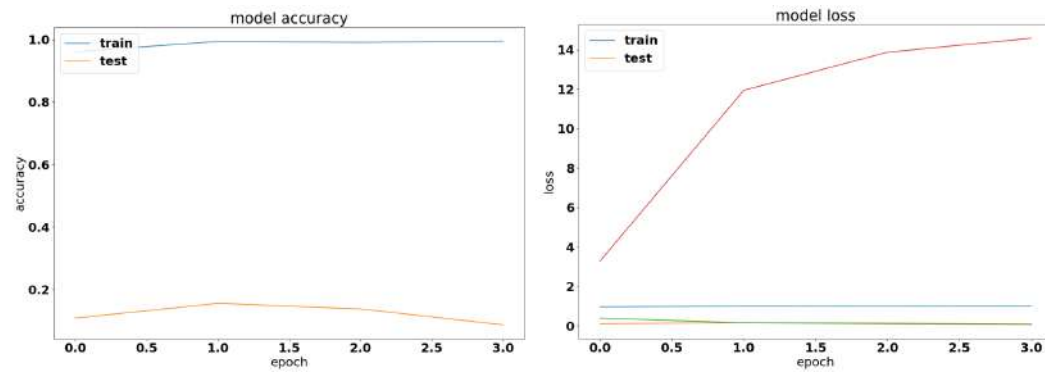


Figure 4.4: Accuracy and Categorical Cross Entropy for ResNet52 CNN

problem of a massively overfit network. For this reason, the network predictions were not displayed in the paper in this chapter.

4.2 Conference Paper: Deep learning seismic facies on state of the art CNN architectures

Abstract: We explore propagation of seismic interpretation by deep learning in stacked 2D sections. We show the application of state-of-the-art image classification algorithms on seismic data. These algorithms were trained on big labeled photograph databases. We use transfer learning to benefit from pre-trained networks and evaluate their performance on seismic data.

J. S. Dramsch and M. L  thje (2018d). “Deep-learning seismic facies on state-of-the-art CNN architectures”. In: *SEG Technical Program Expanded Abstracts 2018. Published*, Chapter 4. Society of Exploration Geophysicists, pp. 2036–2040. DOI: 10.1190/segam2018-2996783.1. URL: <https://doi.org/10.1190/segam2018-2996783.1>

4.2.1 Introduction

Seismic interpretation is often dependent on the interpreters experience and knowledge. While deep learning cannot replace expert knowledge, we explore the accuracy of convolutional networks in interpreting seismic data to support human interpretation.

In the 1950s neural networks started as a simple direct connection of several nodes in an input layer to several nodes in an output layer (Widrow et al., 1990). In geophysics this puts us to the introduction of seismic trace stacking (Yilmaz, 2001a). In 1989 the first idea of a convolutional neural network was born (Lecun, 1989) and back-propagation was formalized as an error-propagation mechanism (Rumelhart et al., 1988a). In 2012 the paper (Krizhevsky et al., 2012a) propelled the field of deep learning forward implementing essential components, namely GPU training, ReLu activation functions (Dahl et al., 2013) and dropout (Srivastava et al., 2014). They outperformed previous models in the ImageNet challenge (Deng et al., 2009a) by almost halving the prediction error. Waldeland et al. (2016) showed that neural networks can be used to classify salt diapirs in 3D seismic data. Rutherford Ildstad et al. (2017) generalized this work to nD and beyond two classes of salt and ”else”.

The task of automatic seismic interpretation can be equated to dense object detection (Lin et al., 2017) or semantic segmentation. These tasks are currently best solved by Mask R-CNN architectures (Long et al., 2015). Statoil has used U-Nets for automatic seismic interpretation. Yet, classification networks can be used for semantic segmentation, but are significantly slower. The benefit is a testable example of generalization of pre-trained networks from photographic data to seismic images. As well as, a testable framework for choosing hyper-parameters for neural networks on seismic data.

Deep learning relies heavily on vast amounts of labeled data to train on initially. However, the features learned from these networks can often be transferred to adjacent problem spaces (Baxter, 1998). Often these transfer learning tasks are tested on photographs rather than seismic or medical imaging tasks. The aim of this study is to evaluate state-of-the-art pre-trained networks in the task of automatic seismic interpretation. We compare three convolutional neural networks of increasing complexity in the task of supervised automatic seismic interpretation. We evaluate these tasks qualitatively and quantitatively.

4.2.2 Methods

The neural networks in this study learn supervised. The features were published alongside the open source framework MalenoV and describe nine seismic facies in the open F3 data set. The classes describe steep dipping reflectors, salt intrusions, low coherency regions, low amplitude dipping reflectors, high amplitude regions continuous high amplitude regions and grizzly amplitude patterns presented in figure 4.7. Additionally, a catch-all “else” region are picked. In this approach we chose Keras (Chollet et al., 2015a) with a Tensorflow (Martín Abadi et al., 2015) backend on a K5200 GPU at DHRTC. Keras is a high level abstraction of tensor arithmetics. Tensorflow is an open source numerical computation library on static graphs. We train 2D convolutional neural networks (CNN) of varying depth on seismic slices to propagate single slice interpretations to a volume. CNNs are highly flexible models for computer vision tasks.

Network one depicted in figure 4.5 was developed by Waldeland et al., 2016 to identify salt bodies in 3D seismic data. Three layers are fully connected for classification. The network uses a kernel of 5 by 5 pixels for convolution and a stride of 2 for down-sampling. We use the Adam optimizer and cross-categorical entropy as a loss function. The Adam optimizer is an extension to stochastic gradient descent (SGD) that implements adaptive learning rates and bias correction (Ruder, 2016). We add dropout and batch normalization to the network. These methods improve regularization and prevent overfitting. Furthermore, we use early-stopping to prevent overfitting the model by over-training. We chose two metrics to monitor in the training and validation sets, namely mean absolute error and accuracy. The Waldeland CNN is relatively shallow compared to modern deep learning networks with 95,735 parameters to optimize for.

Network two is the VGG16 network (Simonyan et al., 2014b) by the Visual Geometry Group. It contains 16 layers and 1,524,2605 parameters. 13 of these layers are convolutional layers with a 3x3 kernel. Convolutional blocks are interspersed with max-pooling layers for down-sampling. The last three layers are fully connected layers for classification. The VGG16 architecture was proposed for the ImageNet challenge in 2013. It is widely used for its simplicity in teaching and its generalizability in transfer learning tasks.

Network three is the ResNet50 architecture by Microsoft. The network consists of 50 layers with 2,361,6569 parameters. It implements a recent development, called residual blocks. These residual blocks add a skip- or identity-connection around a stack

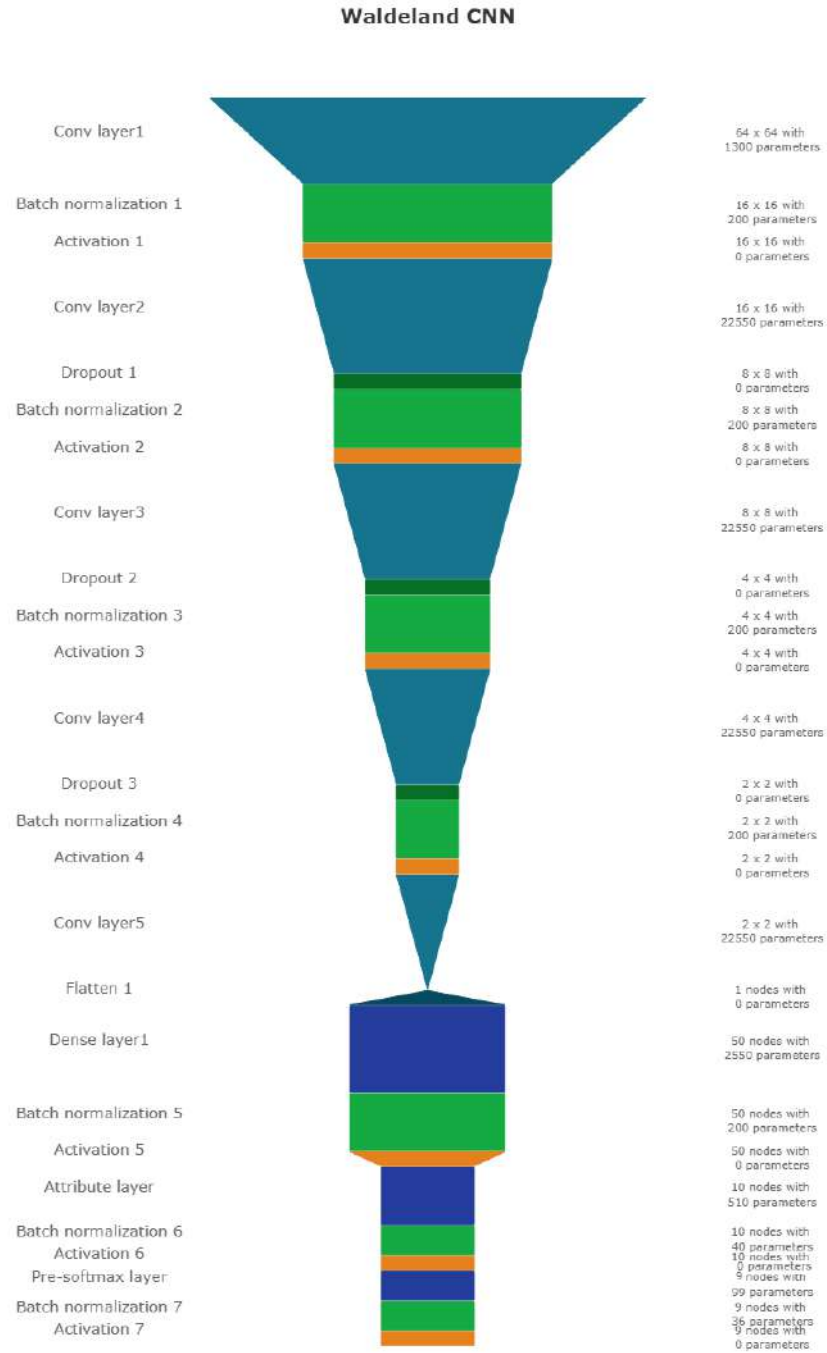


Figure 4.5: Waldeland CNN architecture. Input at the Top. Softmax Classification Layer on bottom. Width of objects shows \log of spatial extent of layer. Height shows \log of complexity of layer. The layers are color coded to show similar purpose.

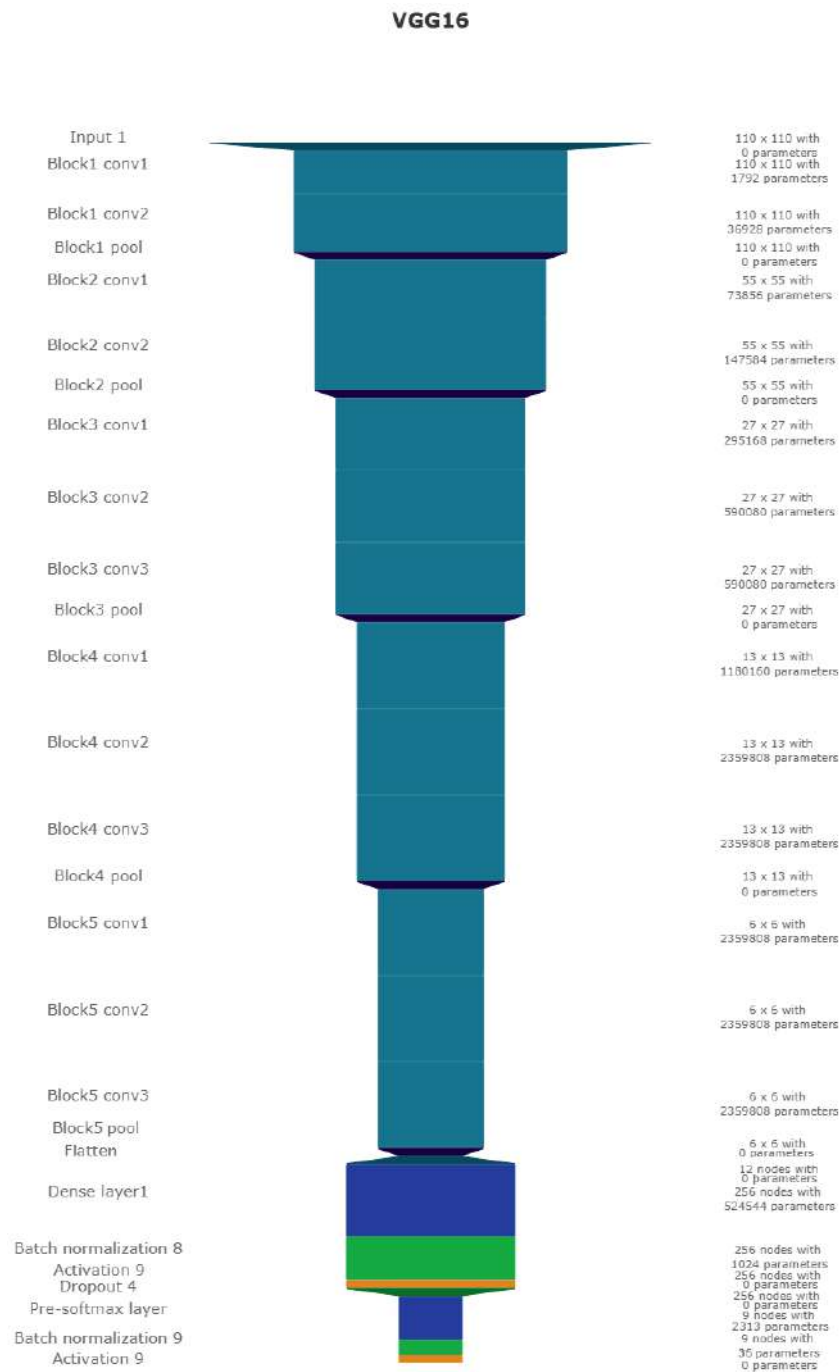


Figure 4.6: VGG16 architecture. Same visualization as figure 4.5

of 1x1, 3x3, 1x1 convolutional layers (He et al., 2016). The 1x1 are identity convolutions, used for down- and subsequent up-sampling to decrease the computational cost of very deep CNNs. The convolutional layers are followed by one fully connected layer for classification.

All networks use rectified linear units (ReLU) as neural activation. The last layer uses Softmax as activation to output a probability for each class. Training both VGG16 and the ResNet50 end to end would be very expensive. These models have been trained on big labeled data that are not available in geoscience. However, transfer learning enables us to use pre-trained networks on very different tasks. In transfer learning, we use the learned weights of the networks and replace the fully connected layers. These untrained layers are specific to our task and have to be fine-tuned to the data. This process is very fast and requires little data. We fine-tune an entire network on one sparsely interpreted 2D seismic slice. For the fine-tuning process, we replace the Adam optimizer by a classic SGD optimizer with lower learning rate, very low weight decay and Nesterov momentum. We still use early-stopping on validation loss and cross-categorical entropy.

We added the same fully connected layer architecture to VGG16 and ResNet50 that Waldeland added to their architecture. Therefore, we test if pre-trained convolution kernels are fit to recognize texture features in seismic data. We set up a validation set to quantify the accuracy of our networks on previously unseen data. Additionally, we set up a prediction pipeline to populate each one 2D inline and crossline of the seismic data to qualitatively visualize the prediction capability of the networks. The labels for the supervised interpretation are taken from the MalenoV interpretation by ConocoPhillips, shown in figure 4.7.

Network	Run	Loss	MAE	Acc
Waldeland CNN	Training	0.001	0.000	100.0%
	Test	0.003	0.000	99.9%
VGG16	Training	0.010	0.005	99.8%
	Test	0.127	0.026	100.0%
ResNet50	Training	0.011	0.001	100.0%
	Test	14.166	0.195	12.1%

Table 4.1: Training and Test scores on Networks. Test scores are prediction results on a labeled hold-out data set. Mismatch of test and training scores indicates over-fitting.

4.2.3 Results

We use the open Dutch F3 data set to calibrate our predictions. Crossline 339 has been interpreted by ConocoPhillips and made available freely. We show results of crossline slice 500. We have used the same plotting parameters for both either results, both have been generated programatically, without human intervention. Figure 4.8a shows the prediction of the Waldeland CNN at every location of the 2D slice based on a 65 x 65 patch of the data. Border patches were zero padded. We see clear patches for the low

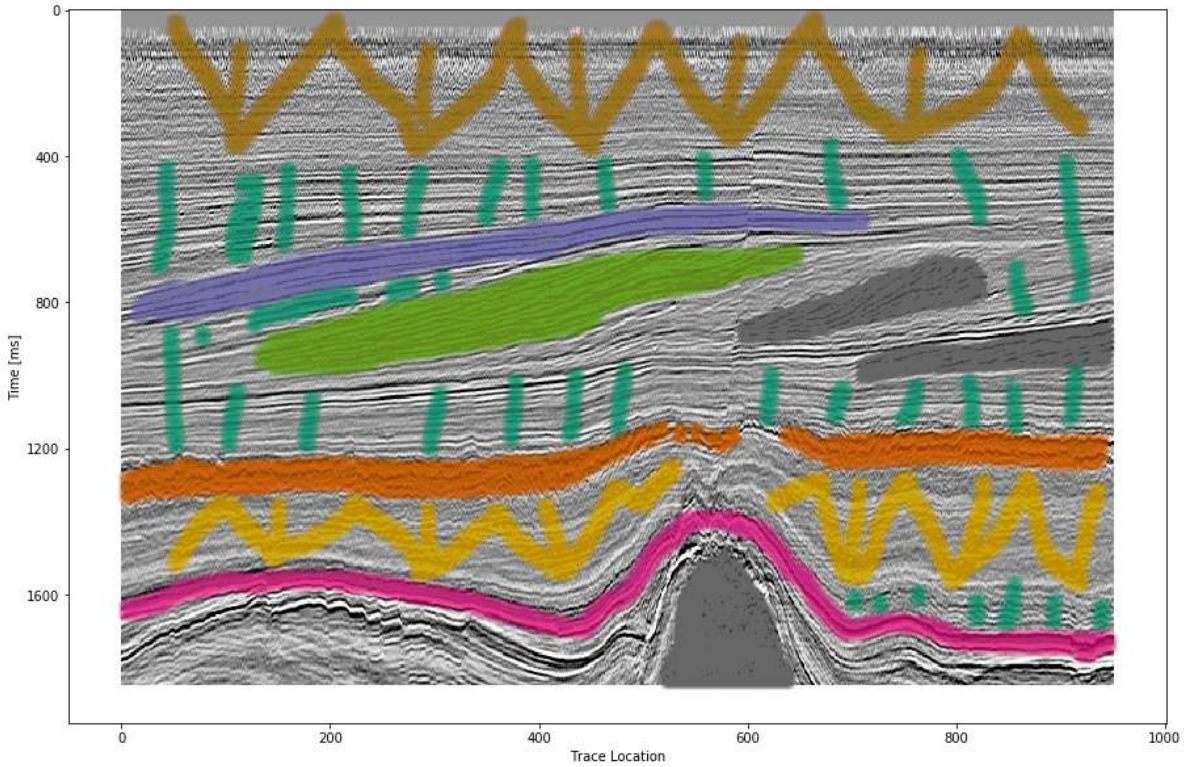


Figure 4.7: Labeled data set on one 2D inline slice. Color interpretation: Low coherency (brown), Steep dipping reflectors (gray), low amplitude dipping reflectors (grass green), continuous high amplitude regions (blue), grizzly (orange), low amplitude (yellow), high amplitude (magenta), salt intrusions (gray), else (turquoise).

coherency region in brown. The low amplitude dipping (grass green) region has been reproduced well, however some regions at $t \approx 1080$ ms have been marked incorrectly, where two seismic packages meet. This faulty region also contains patches that were interpreted as low amplitude region (yellow). While this may be a low amplitude region, we expect the packages to be largely continuous, which leaves this interpretation as questionable at best. The gray area was reproduced well, however it was marked as salt body in the original manuscript, this would be incorrect here. We see the grizzly amplitude pattern (orange) and the low amplitude (yellow) regions are well-defined and separated. The underlying package of high amplitudes has been identified well. However, between location 600 - 800 the top part was marked as "else" (turquoise), which undesirable but correct, judging from the texture. Here, retraining would be possible by feeding this relabeled region to the network. Below this region, the networks predictions become erratic. The classification is blocky between grizzly and salt with "else" interspersed. However, the edges will often give problems due to the padding. Around location 800 high amplitudes (orange) have been mislabeled as grizzly amplitudes.

The VGG16 network classification is shown in figure 4.8b. The network performs similar to the Waldeland CNN in figure 4.8a, however some key differences will be

pointed out. The separation of low coherency and the "else" region around $t \approx 400$ ms is less defined and, therefore, worse. The coherency of low amplitude dipping (grass green) and high amplitude continuous (blue) is worse in the region around location 280, $t \approx 800$ ms. This might be due to higher sensitivity to declines in seismic quality. Below $t \approx 1000$ ms the "else" region is free from differing patches, in contrast, the Waldeland CNN interspersed two other classes in this region. VGG16 also classifies some "else" regions in the high amplitude (magenta) region between location 600-800. The area around location 200 below the high amplitude (magenta) region is also blocky, although less so. The misclassification of the bottom high amplitude (magenta) region as grizzly (orange) is less pronounced in the VGG16 interpretation. It is present toward the bottom left corner.

The results of the ResNet50 are not shown. The network classifies all seismic facies as "else". This indicates that the network is overfitting the data. This is supported by the numeric results presented in table 4.1. The network training error indicates a perfect fit to the data, whereas the test score is unseen data with labels to evaluate the performance of networks on unseen data. While both the Waldeland CNN and VGG16 perform well, the ResNet50 performs very poorly.

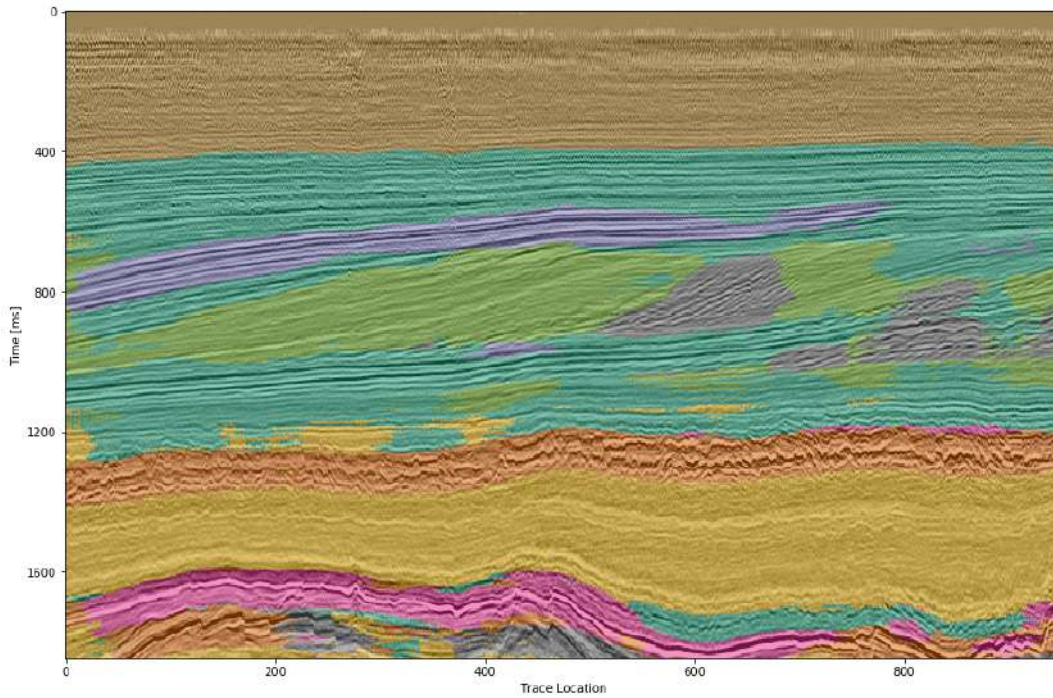
4.2.4 Conclusion

Convolutional neural networks show good results for propagating interpretations through seismic cubes. The pre-trained VGG16 CNN has shown very good results in adapting to seismic texture identification. Transfer learning was fast and the results are similar to the shallower Waldeland CNN. Both networks have trade-offs in the misclassification and can be improved upon.

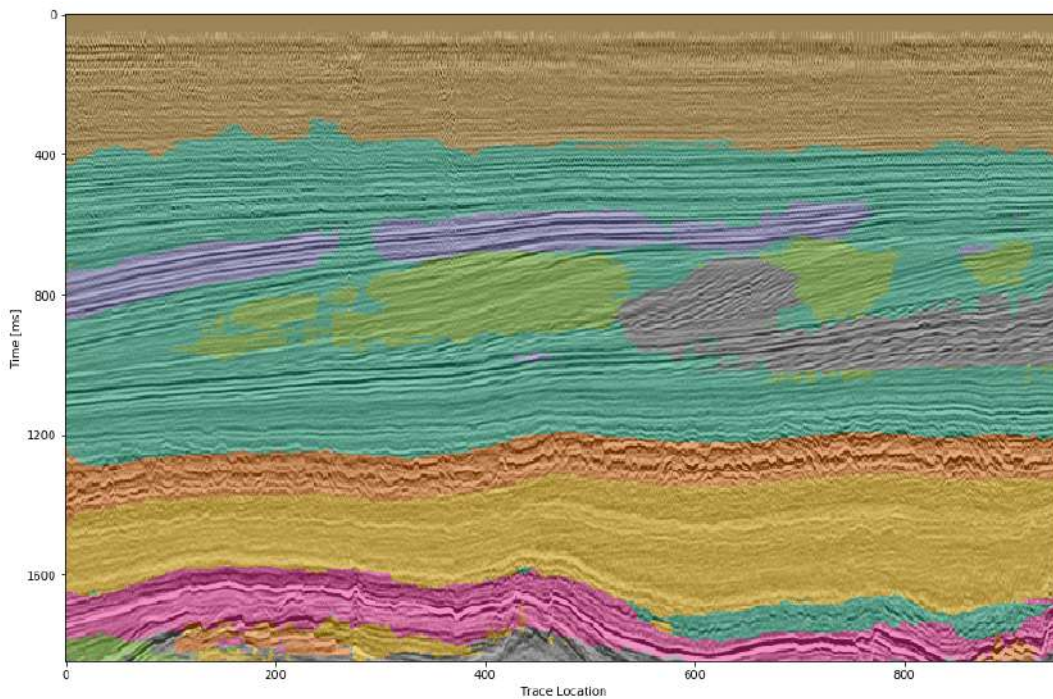
The ResNet50 was shown to be ineffective on transfer learning seismic data with pre-trained weights. This is in accordance with results from other attempts at transfer learning. The ResNet filters are more specific to photography and transfer poorly to other data sources, where the VGG learned features prove to be more general to computer vision tasks. More complicated architectures may perform well, trained directly with the according data, but they learn specific features fit for the problem space that do not transfer well.

4.2.5 Acknowledgments

The authors would like to thank the DHRTC and DUC for their continued support. We thank Colin MacBeth, Peter Bormann, Sebastian Tølbøll Glavind, Lukas Mosser and the "Software Underground" community for great discussion and support with MalenoV and ConocoPhillips for making the data and software freely available. We also thank Agile Scientific for great tutorials at the intersection of Python and geoscience. We thank dgb for providing the F3 data set.



(a) Waldeland CNN automatic interpretation of crossline 500.



(b) VGG16 automatic interpretation of crossline 500.

Figure 4.8: Automatic seismic interpretation with CNNs. Color interpretation: Low coherency (brown), Steep dipping reflectors (gray), low amplitude dipping reflectors (grass green), continuous high amplitude regions (blue), grizzly (orange), low amplitude (yellow), high amplitude (magenta), salt intrusions (gray), else (turquoise).

4.3 Applications of Transfer Learning for Automatic Seismic Interpretation

Figure 4.8a shows the results of a fully trained network compared to a pre-trained network. The pre-trained network decreases both training time and data requirements significantly, while not compromising accuracy. A pre-trained network with diverse generalizable learned filters seems to alleviate some limitations of smaller non-diverse data sets used in the fine-tuning process. These pre-trained networks themselves are of little use to most applications in geoscience. Nevertheless, they can be integrated into more task-appropriate Neural Network architectures that leverage the pre-training.

Apart from building deeper networks for image classification, the neural architectures can serve as a forcing function to the task the network is built for. Encoder-Decoder networks will compress the data with a combination of downsampling layers, which in the case of a computer vision could either be strided convolutions or pooling layers after convolutional layers. During these operations, the number of filters increases, while the spatial extent is diminished significantly. This encoding operation is equivalent to lossy compression, with the low-dimensional layer called "code" or "bottleneck". The bottleneck is then upsampled by either strided transpose Convolutions or upsampling layers that perform a specified interpolation. This is the decoder of the Encoder-Decoder pair. These networks can be used for data compression in AutoEncoders (AEs), where the decoder restores the original data as good as possible (Hinton et al., 2006). Alternatively, the decoder can learn a dense classification task like semantic segmentation or seismic interpretation.

U-Nets present a special type of encoder-decoder networks that learn semantic segmentation on from small datasets (Ronneberger et al., 2015a). They form a special kind of Fully Convolutional Network (FCN) shown in Figure 2.15. Originally developed on biomedical images, the network found wide acceptance in label-sparse disciplines. The U-Net implements shortcut connections between convolutional layers of equal extent in the Encoder and Decoder networks. This alleviates the pressure of the network learning and reconstructing the output data from the bottleneck in isolation.

The data set in this training is very small and non-diverse as shown in Figure 4.7 and this only made training on a classification network possible. Image segmentation would need a dense labelling of the training data and more than one 2D section available. This has been approached by Alaudah et al. (2019) by labelling the full Dutch F3 dataset, which cites the paper presented here. Modern applications of transfer learning were able to leverage ResNet architectures as an encoder in U-nets on seismic data (Babakhin et al., 2019a).

4.4 Contributions of this Study

This study introduced transfer learning for deep learning tasks in Automatic Seismic Interpretation and has found an application across geophysics (see e.g. Babakhin et al., 2019b; Li et al., 2019; Liu et al., 2019a). The transfer learning enables utilizing neural networks that were trained on a diverse dataset and then fine-tuning them with data that contains far fewer samples. This outperforms smaller networks that can be trained end-to-end on these small datasets. The code is available at J. S. Dramschi (Dec. 15, 2018a). *Reproducible Code: Deep-learning seismic facies on state-of-the-art CNN architectures.* DOI: 10.6084/m9.figshare.7227545. URL: <https://github.com/JesperDramschi/seismic-transfer-learning>.

CHAPTER 5

Complex-valued neural networks

In the paper “Complex-valued neural networks for machine learning on non-stationary physical data” (Dramsch et al., 2019f) I explore complex-valued deep convolutional networks to show that phase content in non-stationary data improves generalization of CNNs. This work implements self-supervised AutoEncoders (AEs) that compress the data and measure the reconstruction of the seismic data.

Four different deep convolutional AutoEncoders are constructed. Two AutoEncoders are real-valued and two AutoEncoders are complex-valued. The complex-valued Convolutional Neural Network is implemented as two real-valued feature maps, one for the real component a and one for the complex component b each, which are combined into a complex-valued number with $a + bi$. The complex convolution is then implemented explicitly in the calculation to avoid some drawbacks of using complex numbers by a computer. However, matching the networks proved to be a complicated task with regard to the number of parameters. This led to building four different architectures that get progressively bigger and compare the results.

This study implements AutoEncoders to increase the validity of this experiment. While Variational AutoEncoders have shown better performance on reconstruction tasks, it would also introduce more variability in the network to control for. Considering that Automatic Seismic Interpretation is a fairly new discipline, it is difficult to disambiguate effects on misclassification. These effects include erroneous labels, the difficulty of the task of Automatic Seismic Interpretation, as well as the choice of architecture.

Therefore, this leads us to the decision to inspect the reconstructed seismic data numerically. Signal analysis is well-explored in the seismic data processing. Moreover, this enables analysing the result in the Frequency-Wavenumber (FK)-domain providing additional insight to the denoising effect of the AutoEncoders. Overall, the complex-valued networks result in smaller networks compared to a larger real-valued network achieving comparable reconstruction error.

5.1 Journal Paper: Complex-valued neural networks for machine learning on non-stationary physical data

Abstract: Deep learning has become an area of interest in most scientific areas, including physical sciences. Modern networks apply real-valued transformations on the data. Particularly, convolutions in convolutional neural networks discard phase information entirely. Many deterministic signals, such as seismic data or electrical signals, contain significant information in the phase of the signal. We explore complex-valued deep convolutional networks to leverage non-linear feature maps. Seismic data commonly has a lowcut filter applied, to attenuate noise from ocean waves and similar long wavelength contributions. In non-stationary data, the phase content can stabilize training and improve the generalizability of neural networks. While it has been shown that phase content can be restored in deep neural networks, we show how including phase information in feature maps improves both training and inference from deterministic physical data. Furthermore, we show that smaller complex networks outperform larger real-valued networks.

J. S. Drams, M. Lüthje, and A. N. Christensen (2019f). “Complex-valued neural networks for machine learning on non-stationary physical data”. In: *Computers & Geoscience*. Accepted, Chapter 5

5.1.1 Introduction

Seismic data has its caveats due to the complicated nature of bandwidth-limited wave-based imaging. Common problems are cycle-skipping of wavelets and nullspaces in inversion problems (Yilmaz, 2001b). Automatic seismic interpretation is complicated, as the modelling of seismic data is computationally expensive and often proprietary. Seismic field data is often not available and their interpretation is highly subjective and ground truth is not available. The lack of training data has been delaying the adoption of existing methods and hindering the development of specific geophysical deep learning methods. Incorporating domain knowledge into general deep learning models has been successful in other fields (Paganini et al., 2017).

The state-of-the-art method has been an iterative windowed Fourier transform for phase reconstruction (Griffin et al., 1984). Modern neural audio synthesis focuses on methods that do not require explicit reconstruction of the phase (Mehri et al., 2016; Oord et al., 2016; Oord et al., 2017; Prenger et al., 2018). Mehri et al. (2016) introduced a recurrent neural network formulation, where Oord et al. (2016) reformulated the network for audio synthesis in a strided convolutional network. The original WaveNet formulation in Oord et al. (2016) is slow due to the autoregressive filter, warranting the parallel

formulation in Oord et al. (2017).

We explicitly incorporate phase information in a deep convolutional neural network. These have been heavily explored in the digital signal processing community, before the recent renaissance of neural networks and deep learning. Relevant examples to seismic data processing include source separation (Scarpiniti et al., 2008), adaptive noise reduction (Suksmono et al., 2002), and optical flow (Miyauchi et al., 1993) with complex-valued neural networks. Sarroff (2018) gives a comprehensive overview of applications of complex-valued neural networks in signal and image processing.

In this work, we evaluate the reconstruction error after compression in an autoencoder to test how reliable information can be contained within a network with and without explicit phase information. This insight can be transferred to the aforementioned applications that benefit from an increase in information recovery. We calculate the complex-valued seismic trace by applying the Hilbert transform to each trace. Phase information has been shown to be valuable in the processing (Liner, 2002) and interpretation of seismic data (Roden et al., 1999; Mavko et al., 2003). Purves (2014) provides a tutorial that shows the implementation details of Hilbert transforms.

In this paper we give a brief overview of convolutional neural networks and then introduce the extension to complex neural networks and seismic data. We show that including explicit phase information provides superior results to real-valued convolutional neural networks for seismic data. Difficult areas that contain seismic discontinuities due to geologic faulting are resolved better without leakage of seismic horizons. We train and evaluate several complex-valued and real-valued autoencoders to show and compare these properties. These results can be directly extended to automatic seismic interpretation problems.

5.1.2 Complex Convolutional Neural Networks

5.1.2.1 Basic principles

Convolutional neural networks (LeCun et al., 1999) use multiple layers of convolution and subsampling to extract relevant information from the data (see Figure 5.1)

The input image is repeatedly convolved with filters and subsampled. This creates many, but smaller and smaller images. For a classification task, the final step is then a weighting of these very small images leading to a decision about what was in the original image. The filters are learned as part of the training process by exposing the network to training images. The salient point is, that the convolution kernels are learned based on the training. If the goal is - for example - to classify geological facies, the convolutional kernels will learn to extract information from the input, that helps with that task. It is thus a very strong methodology, that can be adapted to many tasks.

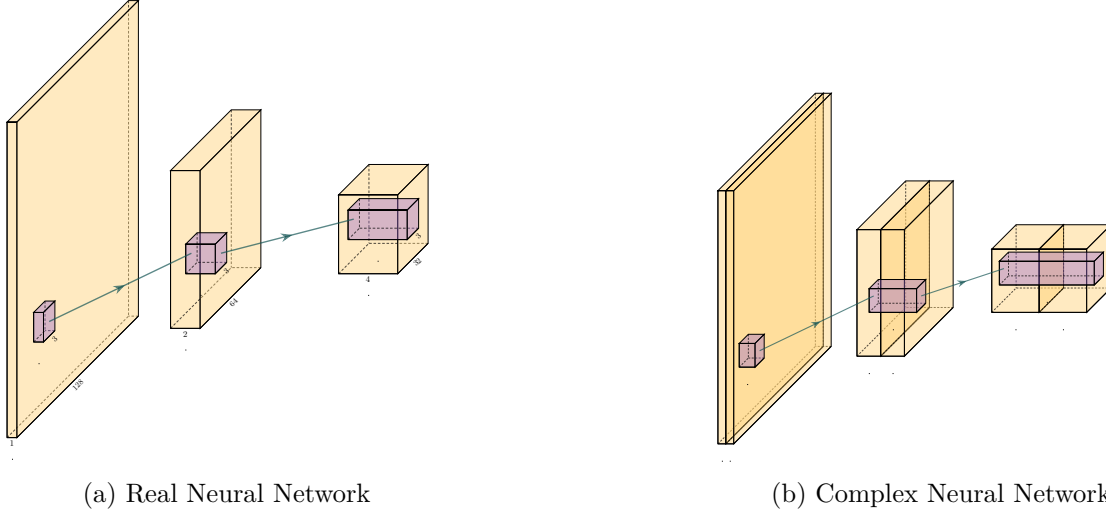


Figure 5.1: Schematic of equivalent real- and complex-valued convolutional neural networks. Yellow is the input image data and purple shows the 3×3 convolutional filters. In (a) the input image is convolved with filters. This results in several smaller outputs. The process is repeated, resulting in more outputs even further reduced in size. In (b) we present the complex-valued network. We start with a complex valued input represented by two layers, namely the real-valued a and complex-valued b complement in $a + ib$. The complex information is propagated through the network by keeping - in each step - the real and complex information in different layers after convolution with complex-valued filters.

5.1.2.2 Real- and Complex-valued Convolution

Convolution is an operation on two signals f and g or a signal and a filter that produce a third signal, containing information from both of the inputs. An example is the moving average filter, which smoothes the input, acting as a low-pass filter. Convolution is defined as

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau, \quad (5.1)$$

at the location τ . While often applied to real value signals, convolution can be used on complex signals. For the integral to exist both f and g must decay when approaching infinity. Convolution is directly generalizable to N-dimensions by multiple integrations and maintains commutativity, distributivity, and associativity. In digital signals this extends to discrete values by replacing the integration with summation.

5.1.2.3 Complex Convolutional Neural Networks

Complex convolutional networks provide the benefit of explicitly modelling the phase space of physical systems (Trabelsi et al., 2017). Unfortunately it is not possible to feed complex numbers directly to a CNN, as they are not supported by any of the standard implementations (PyTorch or Tensorflow). Instead, we can represent them in

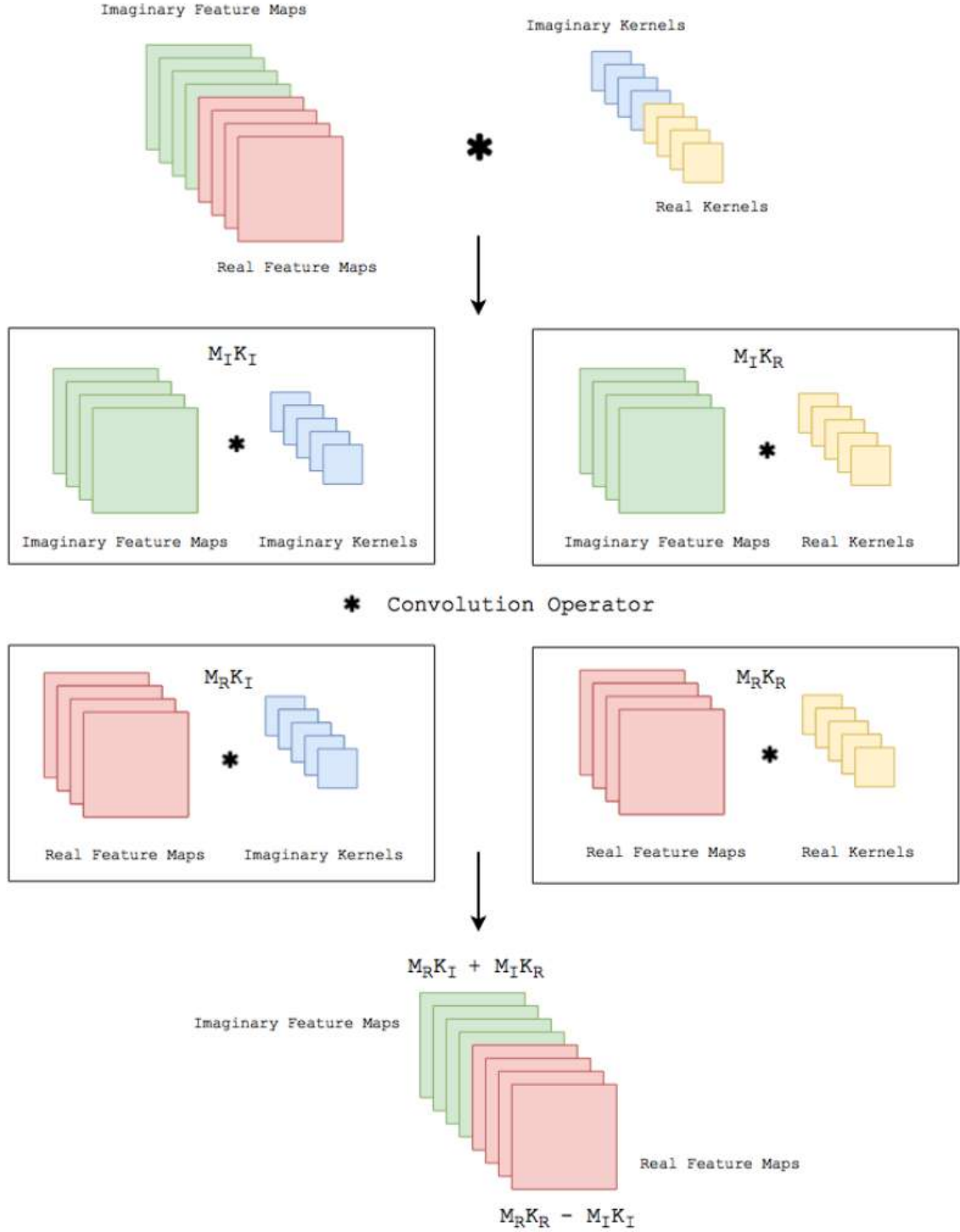


Figure 5.2: Implementation details of Complex Convolution (Courtesy Trabelsi et al. (2017))

another form. The complex convolution introduced in Section 5.1.2.2, can be explicitly implemented as convolutions of the real and complex components of both kernels and the data. A complex-valued data matrix in cartesian notation is defined as $\mathbf{M} = M_{\Re} + iM_{\Im}$ and equally, the complex-valued convolutional kernel is defined as $\mathbf{K} = K_{\Re} + iK_{\Im}$. The individual coefficients $(M_{\Re}, M_{\Im}, K_{\Re}, K_{\Im})$ are real-valued matrices, considering vectors are special cases of matrices with one of two dimensions being one.

Solving the convolution of

$$M' = K * M = (M_{\Re} + iM_{\Im}) * (K_{\Re} + iK_{\Im}), \quad (5.2)$$

we can apply the distributivity of convolutions (cf. section 5.1.2.2) to obtain

$$M' = \{M_{\Re} * K_{\Re} - M_{\Im} * K_{\Im}\} + i\{M_{\Re} * K_{\Im} + M_{\Im} * K_{\Re}\}, \quad (5.3)$$

where K is the Kernel and M is a data vector (see Figure 5.2).

We can reformulate this in algebraic notation

$$\begin{bmatrix} \Re\{M * K\} \\ \Im\{M * K\} \end{bmatrix} = \begin{bmatrix} K_{\Re} & -K_{\Im} \\ K_{\Im} & K_{\Re} \end{bmatrix} * \begin{bmatrix} M_{\Re} \\ M_{\Im} \end{bmatrix} \quad (5.4)$$

Complex convolutional neural networks learn by back-propagation. Sarroff et al. (2015) state that the activation functions, as well as the loss function must be complex differentiable (holomorphic). Trabelsi et al. (2017) suggest that employing complex losses and activation functions is valid for speed, however, refers that Hirose et al. (2012) show that complex-valued networks can be optimized individually with real-valued loss functions and contain piecewise real-valued activations. We reimplement the code Trabelsi et al. (2017) provides in keras (Chollet et al., 2015a) with tensorflow (Martín Abadi et al., 2015), which provides convenience functions implementing a multitude of real-valued loss functions and activations.

While common up- and downsampling functions like MaxPooling, UpSampling, or striding do not suffer from complex-valued neural networks, batch normalization (BN) (Ioffe et al., 2015) does. Real-valued batch normalization normalizes the data to zero mean and a standard deviation of 1. This does not guarantee normalization in complex values. Trabelsi et al. (2017) suggest implementing a 2D whitening operation as normalization in the following way.

$$\tilde{x} = V^{-\frac{1}{2}}(x - \mathbb{E}[x]), \quad (5.5)$$

where x is the data and V is the 2x2 covariance matrix, with the covariance matrix being

$$V = \begin{bmatrix} V_{\Re\Re} & V_{\Re\Im} \\ V_{\Im\Re} & V_{\Im\Im} \end{bmatrix} \quad (5.6)$$

Effectively, this multiplies the inverse of the square root of the covariance matrix with the zero-centred data. This scales the covariance of the components instead of the variance of the data (Trabelsi et al., 2017).

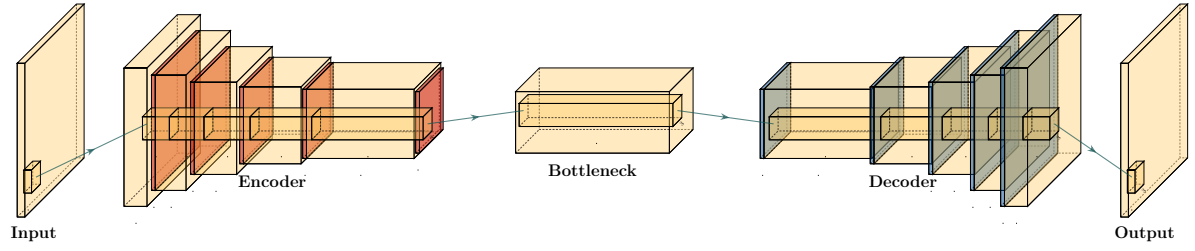


Figure 5.3: Typical autoencoder architecture. The data is compressed to a low dimensional bottleneck, and then reconstructed. In the encoder convolutional layers (yellow) are followed by a down-sampling operation (red) to reduce the spatial extent of the input image. The bottleneck contains a lower-dimensional compressed representation of the input. The decoder contains upsampling operations (blue) followed by convolutional layers symmetrical to the encoder. Alternatively, the encoder is sometimes made up of transpose convolutions.

5.1.2.4 Autoencoders

Autoencoders (Hinton et al., 2006) are a special configuration of the encoder-decoder network that map data to a low-level representation and back to the original data. This low-level representation - the latent space - is often called bottleneck or code layer. Autoencoder networks map $f(x) = x$, where x is the data and f is an arbitrary network. The architecture of autoencoders is an example of lossy compression and recovery from the lossy representation. Commonly, recovered data is blurred by this process.

The principle is illustrated in figure 5.3. The input is transformed to a low-dimensional representation - called a code or latent space - and then reconstructed again from this low dimensional representation. The intuition is, that the network has to extract the most salient parts from the data, to be able to perform a reconstruction. As opposed to other methods for dimensionality reduction - e.g. principal component analysis - an autoencoder can find a non-linear representation of the data. The low-dimensional representation can then be used for anomaly detection, or classification.

5.1.3 Aliasing in Patch-based training

5.1.3.1 Mean-Shift in Neural Networks

A single neuron in a neural network can be described by $\sigma(w \cdot x + b)$, where w is the network weights, x is the input data, b is the network bias, and σ is a non-linear activation function. During training, the network weights w and biases b are adjusted to a value that represents the training minimum. Learning on a mean-shift of q of an arbitrary distribution over x leads to $\sigma(w \cdot (x + q) + b)$, which increases the neuron response by q , weighted by w . During inference, both w and b are fixed, by extension the mean-shift q is fixed as well. The mean-shift over larger inference data disappears, introducing

an additional bias of $w \cdot q$ before non-linear activation. This training bias may lead to prediction errors of the neuron and consequently the full neural network.

5.1.3.2 Windowed Aliasing

Non-stationary data such as seismic data can contain sections within the data that contain spurious offsets from the mean. Figure 5.4 shows varying sizes of cutouts, with 101 and 256 samples respectively. In the middle, the full normalised amplitude spectra are presented. On the right, the corresponding phase spectra are presented. On the left, we focus on the frequency content of the amplitude spectra around 0 Hz. The cutouts were Hanning tapered, however, a mean shift appears for any patch size.

These concepts of mean-shift corresponds to a DC offset in spectral data, which can be audio, seismic or electrical data. In images this corresponds to a non-zero alpha channel. While batch normalization can correct the mean shift in individual mini-batches (Ioffe et al., 2015), this may shift the entire spectrum by the aliased offset. Additionally, batch normalization may not be feasible in some physical applications pertaining to regression tasks.

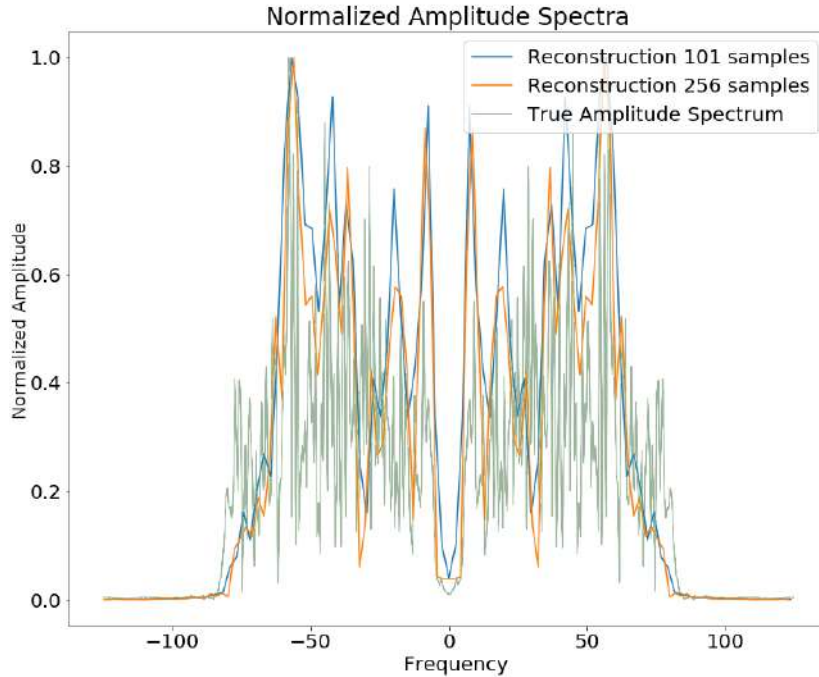


Figure 5.4: Spectral aliasing dependent on window-size (modified from Dramsch et al. (2018e)). The true amplitude spectrum (green) is 0 at a frequency of 0 Hz, whereas windows of the data experience low-frequency aliasing that introduce a non-zero offset at 0 Hz analogous to the Nyquist-Shannon theorem for high frequencies.

5.1.4 Complex Seismic Data

Complex seismic traces are calculated by applying the Hilbert transform to the real-valued signal. The Hilbert transform applies a convolution with to the signal, which is equivalent to a -90-degree phase rotation. It is essential that the signal does not contain a DC component, as this would not have a phase rotation.

The Hilbert transform is defined as

$$H(u)(t) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{u(\tau)}{t - \tau} d\tau, \quad (5.7)$$

of a real-valued time series $u(t)$, where the improper integral has to be interpreted as the Cauchy principal value. In the Fourier domain, the Hilbert transform has a convenient formulation, where frequencies are set zero and the remaining frequencies are multiplied by 2. This can be written as

$$x_a = F^{-1}(F(x)2U) = x + iy \quad (5.8)$$

where x_a is the analytical signal, x is the real signal, F is the Fourier transform, and U is the step function. The imaginary component y is simultaneously the quadrature of the real-valued trace. This provides locality to explicit phase information, where the Fourier transform itself does not lend itself to the resolution of the phase in the time domain. In conventional seismic trace analysis, the complex data is used to calculate the instantaneous amplitude and instantaneous frequency. These are beneficial seismic attributes for interpretation (Barnes, 2007).

5.1.5 Experiments

5.1.5.1 Data

The data is the F3 seismic data, acquired in the Dutch North Sea in 1987 over an area of 375.31 km². The sampling-rates are 4 ms in time and inline/crossline bins of 25 m. The extent being 650 inline traces and 950 crossline traces with a total length of 1.848 s. The data contains faulted reflector packets, of which the lowest one overlays a salt diapir. The data contains some noise that masks lower-amplitude events.

We generate 2D patches of size 64x64 in the inline and crossline direction from the 3D volume to train our network. We obtain inline and crossline 64x64 patches that are taken overlapping with a stride of 8 samples. The total amount of data is 188736 patches with 141552 for training and 47184 for validation in a 75/25 train-validation split. The test data is the holdout Alaudah et al. (2019) stored in test_once. The seismic data is normalized to values in the range of [-1, 1]. To obtain complex-valued seismic data we apply a Hilbert transform to every trace of the data and subtract the real-valued seismic from the real component as laid out in Taner et al. (1979).

Layer (Size)	Spatial X	Y	Complex Small	Real Small	Complex Large	Real Large
Input	64	64	2	1	2	1
(\mathbb{C})-Conv2D	64	64	8	8	16	16
(\mathbb{C})-Conv2D + BN	64	64	8	8	16	16
Pool + (\mathbb{C})-Conv2D + BN	32	32	16	16	32	32
Pool + (\mathbb{C})-Conv2D + BN	16	16	32	32	64	64
Pool + (\mathbb{C})-Conv2D + BN	8	8	64	64	128	128
Pool + (\mathbb{C})-Conv2D	4	4	128	128	256	256
Up + (\mathbb{C})-Conv2D + BN	8	8	64	64	128	128
Up + (\mathbb{C})-Conv2D + BN	16	16	32	32	64	64
Up + (\mathbb{C})-Conv2D + BN	32	32	16	16	32	32
Up + (\mathbb{C})-Conv2D	64	64	8	8	16	16
(\mathbb{C})-Conv2D + BN	64	64	8	8	16	16
(\mathbb{C})-Conv2D	64	64	2	1	2	1
Parameters on Graph			100,226	198,001	397,442	790,945
Compression Ratio			4:1	2:1	2:1	1:1
Size on Disk [MB]			1.4	2.5	4.8	9.2

Table 5.1: Layers used in the four autoencoders and according parameter count on the computational graph for complex-valued convolutions and real-valued convolutions respectively. The spatial extents in X and Y per layer are kept constant across all networks, varying the amount of filters. The compression is calculated by number comparing the total input parameters to the bottleneck parameters.

5.1.5.2 Architecture

The autoencoder architecture compresses the input data to a lower dimensional representation, i.e. bottleneck (cf. Figure 5.3), with an encoder network and reconstruct the input data from the bottleneck with a decoder network. It is common that the encoder and decoder networks are formulated symmetrically, as we have done in this paper. We reduce a 64x64 input 4 times by a factor of two spatially to encode a 4x4 encoding layer. We define varying amounts of filters during the downsampling steps and in the code layer to achieve varying amounts of compression shown in Table 5.1. The architecture for the complex convolutional network is identical to the real network, except for replacing the real-valued 2D convolutions with complex-valued convolutions represented by two feature maps instead of one. The layers for each network are shown in Table 5.1 with additional values, including learnable parameters counted on the computational graph, compression ratio, and size on disk. In total four network architectures are presented, two real-valued and complex-valued networks each matched in the number of feature maps, resulting in different amounts of parameters and compression ratio. The parameters are counted on the computational graph compiled by Tensorflow.

The neural networks specifically use 2D convolutions with 3x3 kernels. We employ batch normalization to regularize and speed up training (Ioffe et al., 2015). The down and up sampling is achieved by MaxPooling and the UpSampling operation respectively.

Complex-valued neural networks contain two feature maps for every feature map contained in a real-valued network. Conceptually, this is equivalent to $a + ib$, with $b = 0$ for the real-valued network. The information in the complex complement for these two feature maps is derived from the input data using the Hilbert transform. Following the argument of deep learning, this input could be derived from a neural network directly and should not provide an improvement to the networks reconstruction error. We define a complex-valued network that has the same number of filters as the real-valued network in both the "small" and "large" formulation in Table 5.1. This network effectively has half the available feature maps for the real-valued seismic input, as the other half is used for the complex-valued information. That means the smaller real-valued network contains as many feature maps for the real-valued seismic as the large complex network, the large real-valued network contains an additional feature map for every real-valued input for the complex component.

5.1.5.3 Training

We train the networks with an Adam optimizer (Kingma et al., 2014) and a learning rate of 10^{-3} without decay, for 100 epochs. The loss function is mean squared error, as the seismic data contains values in the range of $[-1,1]$. All networks reach stable convergence without overfitting, shown in Figure 5.5.

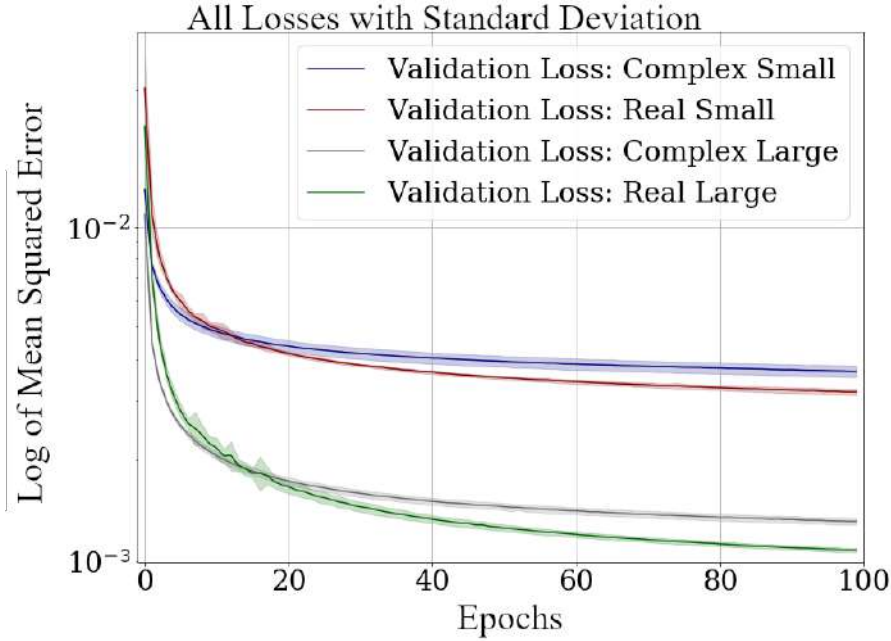


Figure 5.5: Validation Loss (MSE) on 7 random seeds per network. (Real-valued loss on real-valued seismic and combined complex-valued loss on complex-valued seismic, as the network "sees" it.)

5.1.5.4 Evaluation

We compare the complex autoencoders with the real-valued autoencoders, through the reconstruction error on unseen test data on 7 individual realizations of the respective four networks and qualitative analysis of reconstructed images. We focus on evaluating the real-valued reconstruction of the seismic data for both networks.

5.1.6 Results

We trained four neural network autoencoders with seven random initializations for each network, to allow for error bars on the estimates in Figure 5.5. The mean squared error and the mean absolute error for each parameter configuration during training is given in Table 5.2. There is a clear correspondence of the reconstruction error of the autoencoder to the size of network. The real-valued networks outperform the complex-valued networks in both the mean squared error and mean absolute error, however, we see that a real-valued network needs around twice as many parameters as a complex-valued network to attain the same reconstruction errors.

The seismic sections in Figure 5.6 show the unseen test seismic section. We perform a closer inspection of the regions "top" and "bottom" to focus on geologically relevant sections in the reconstruction process. The noisy segment without strong reflectors is a good baseline to evaluate the noise reduction of the autoencoder and the behaviour of the different networks on low amplitude data. Overall, all networks denoise the original

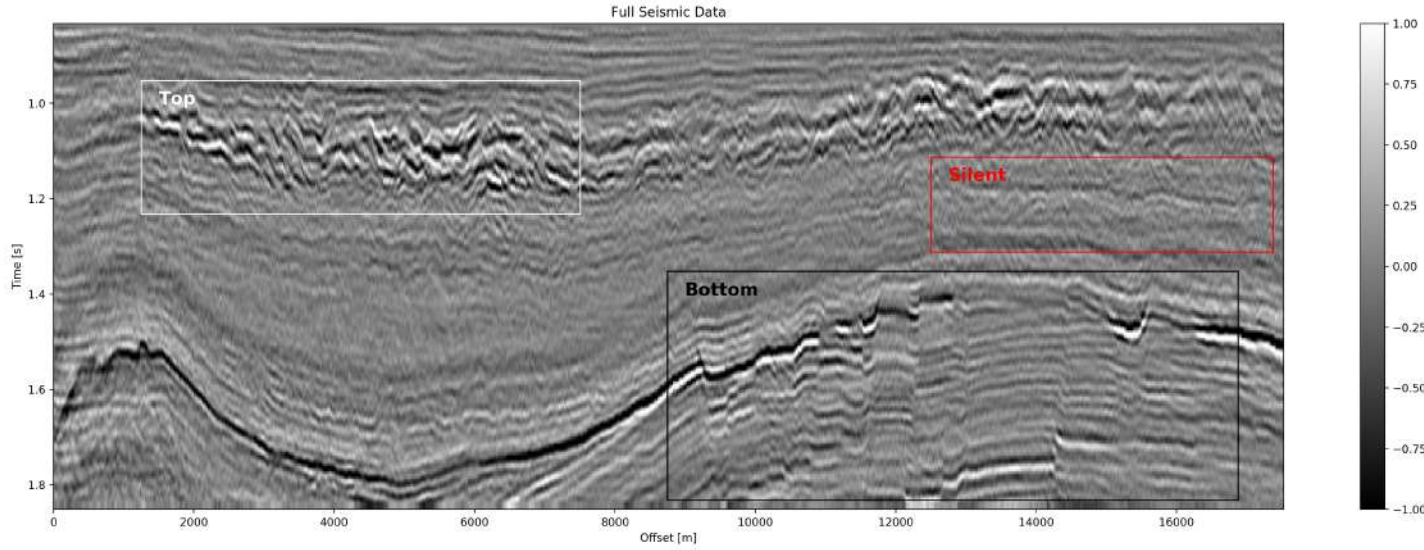


Figure 5.6: Seismic Test Data with marked section for closer inspection. We chose the "top" section for it's faulted chaotic texture, "bottom" for the faulted blocks, and "silent" for a noisy but geologically uninteresting section.

Network	Compression	Parameters	MSE [$\times 10^{-2}$]	MAE [$\times 10^{-2}$]
1) $\mathbb{C}_{\text{small}}$	4:1	100,226	0.484 ± 0.013	4.695 ± 0.058
2) $\mathbb{R}_{\text{small}}$	2:1	198,001	0.436 ± 0.006	4.500 ± 0.028
3) $\mathbb{C}_{\text{large}}$	2:1	397,442	0.227 ± 0.003	3.247 ± 0.025
4) $\mathbb{R}_{\text{large}}$	1:1	790,945	0.196 ± 0.002	3.050 ± 0.013

Table 5.2: Compression, parameters and errors for networks (lower is better). Losses on network validation. The complex-valued networks achieve similar reconstruction errors at twice the compression values.

seismic, with the lowest reconstruction errors being root-mean-squared (RMS) of 0.1187 and MAE of 0.0947 (cf. Table 5.3). Figure 5.7 shows the frequency-wavenumber (FK) of the ground truth (5.7 (a)) and the large complex network reconstruction (5.7 (b)). These show a decrease in the 0 - 60 Hz band for larger absolute wavenumbers.

Network	Full		Silent		Top		Bottom	
	RMS	MAE	RMS	MAE	RMS	MAE	RMS	MAE
1) $\mathbb{C}_{\text{small}}$	0.1549	0.1145	0.1265	0.1010	0.2315	0.1759	0.1588	0.1200
2) $\mathbb{R}_{\text{small}}$	0.1581	0.1153	0.1247	0.0994	0.2395	0.1810	0.1612	0.1205
3) $\mathbb{C}_{\text{large}}$	0.1508	0.1101	0.1187	0.0947	0.2301	0.1747	0.1514	0.1135
4) $\mathbb{R}_{\text{large}}$	0.1469	0.1072	0.1214	0.0967	0.2222	0.1679	0.1459	0.1088

Table 5.3: RMS and MAE on real component of Data Patches.

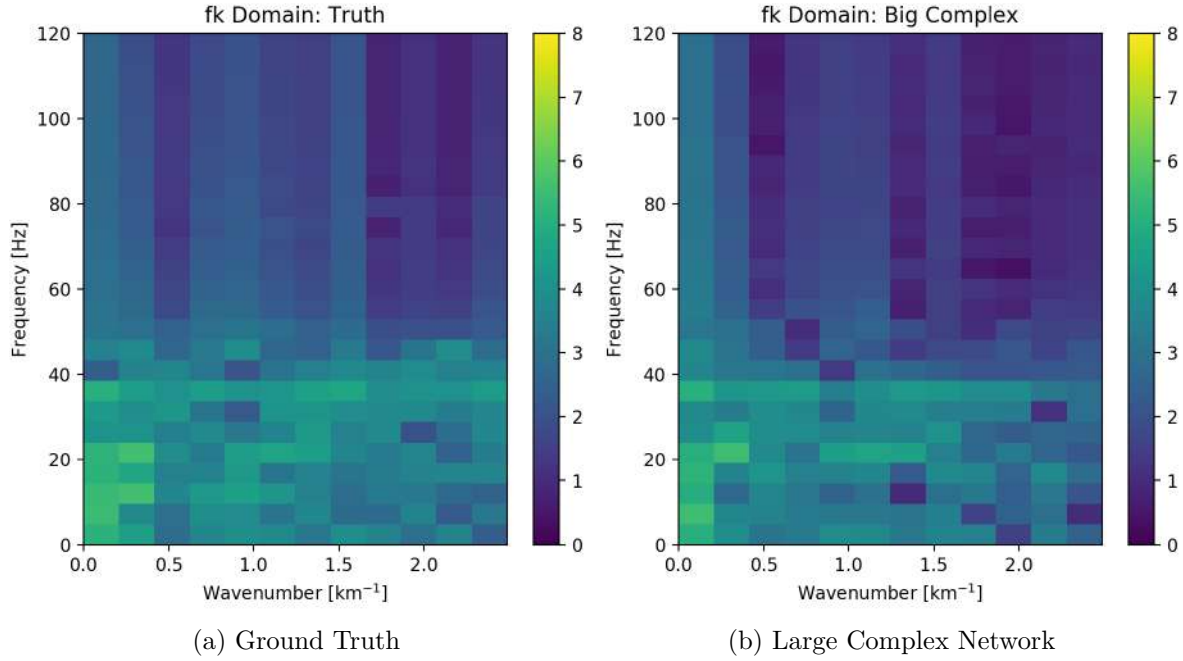


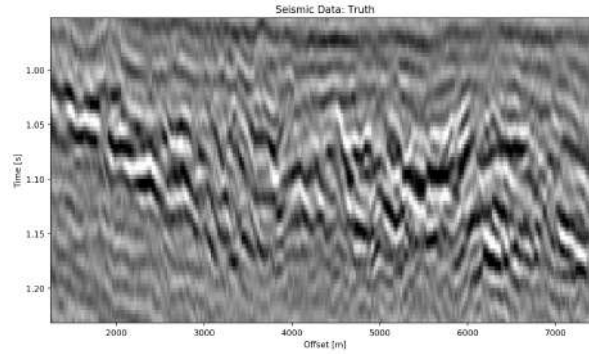
Figure 5.7: Evaluation on Silent Noise Patch in FK Domain. Noise reduction of frequencies below 50 Hz apparent, while reconstruction does not introduce visible aliasing.

5.1.6.1 “Top” seismic section

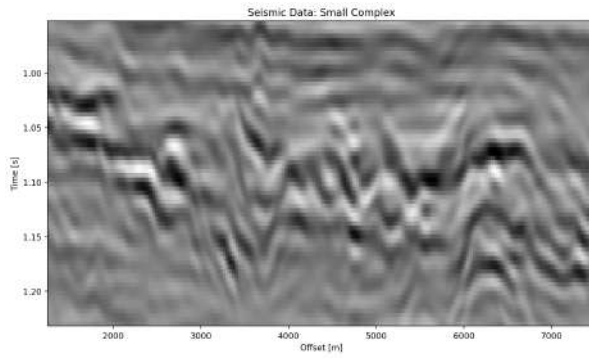
The “top” segment contains strong reflections that are very faulted with strong reflectors. Figure 5.8 shows the top segment and the reconstructions of the four networks. All networks display various amounts of smoothing. The quantitative results show that the complex networks perform very similar regardless of size. The large real-valued network outperforms the complex networks by 2.5 % on RMS, while the small real-valued network underperforms by 2.5 % on RMS. The panel in Figure 5.8c shows a very smooth result. Despite the close score of the complex networks, it appears that the complex-valued network restores more high-frequency content. We can also see less smearing of discontinuities in the larger complex network, particularly visible in the lower part (1.2 s) at 6000 m offset, which is smeared to appear like a diffraction in the smaller network. The large real-valued network shows good reconstruction with minor smearing with higher amplitude fidelity in areas like 1.1 s at 2000 m, however, some of the steeply dipping artifacts are visible below the reflector packet between 0 m and 2000 m offset.

5.1.6.2 “Bottom” seismic section

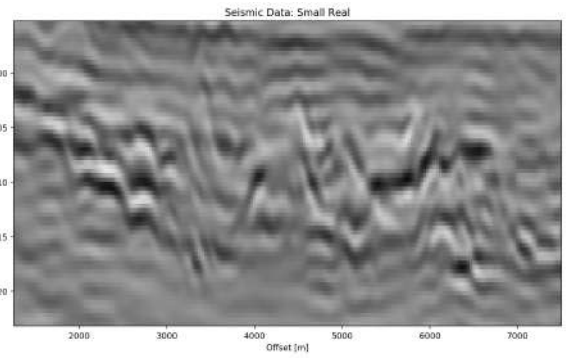
The data marked as “bottom” in Figure 5.6 contains a faulted anticline and relatively strong noise levels. The small complex network in Figure 5.9b reconstructs a denoised image with good reconstruction of the visible discontinuities. Some leakage of the reflector



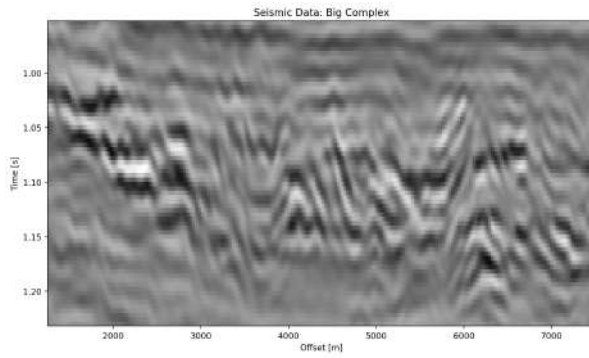
(a) Ground Truth



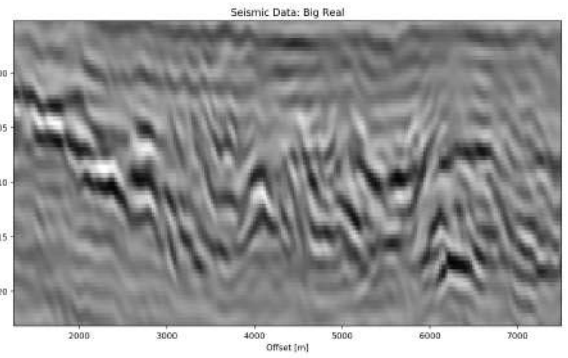
(b) Small Complex Network top Patch



(c) Small Real Network top Patch



(d) Large Complex Network top Patch



(e) Large Real Network top Patch

Figure 5.8: Evaluation on top Noise Patch. Data is normalized between -1 and 1.

starting at 1.5 s across discontinuities is visible. The real small network in Figure 5.9c reconstructs a strongly smoothed image, with some ringing below the main reflector, which is not visible in the other reconstructions. The dipping reflector at an offset of 16000 m is well reconstructed, however, it seems like the reconstruction introduced ringing noise over the vertical image. The large real-valued network in Figure 5.9e performs best quantitatively (cf. Table 5.3). The complex-valued large network in Figure 5.9d does a fairly good job at reconstructing the image, similar to the large real-valued network. However, the amplitude reconstruction of high-amplitude events particularly in the main reflector around 1.5 s is showing.

5.1.6.3 Full seismic test data

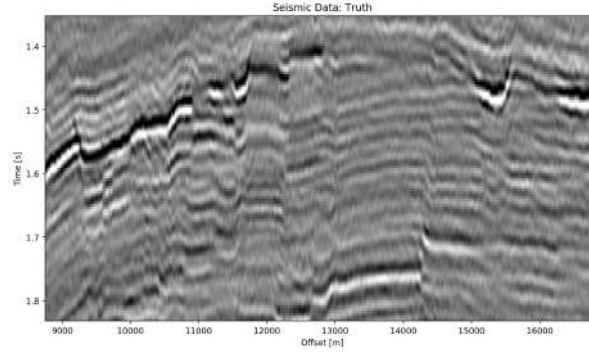
It is evident, that the small real-valued network does not match the performance of the smaller complex-valued network, even less so when compared to the large complex-valued network. We therefore compare the large networks on the full seismic data.

Overall, both networks return a smoothed image. The findings for the strongly faulted sections in the "top" panel hold across the entire faulted area around 1.1 s in Figure 5.11. The complex-valued network does a better job at reconstructing faults and discontinuities. The real-valued network is better at reconstructing high-amplitude regions that appear dimmer in the complex-valued region. The reconstruction of both networks seems adequately close to the ground truth, with differences in the details. Quantitatively, the real-valued network does the better reconstruction in Table 5.3 with an improvement of 2.5 % over the large complex-valued network. The FK domain shows a very similar reduction in noise in the sub 50 Hz band in Figure 5.10. All networks introduce an increase of energy across all frequencies at wave-number $k = 0 \text{ km}^{-1}$. Additionally, a dimming of the frequencies around $k = 2.5 \text{ km}^{-1}$ appears in all reconstructions, but is more prominent in the large complex-valued network. The ground truth seismic contains some scattered energy in the high-frequency mid-wavenumber region, visible as "diagonal stripes". These were attenuated in the complex-valued network in Figure 5.10b, but are partially present in the real-valued reconstruction in Figure 5.10c.

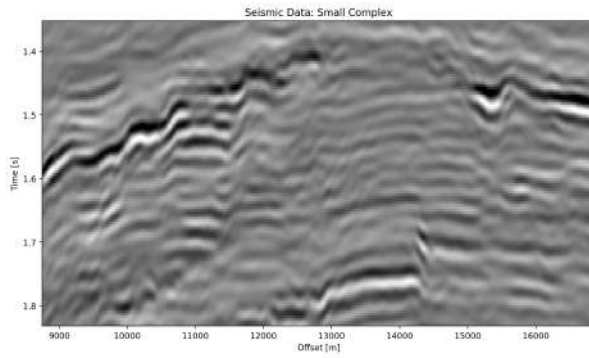
5.1.7 Discussion

We evaluated the outputs of the real-valued and complex-valued neural networks. All autoencoder outputs are blurred to different degrees and denoised. The denoising effect of the seismic was most visible in the frequency band below 50 Hz. Additionally, some scattered high-frequency energy was attenuated by the networks.

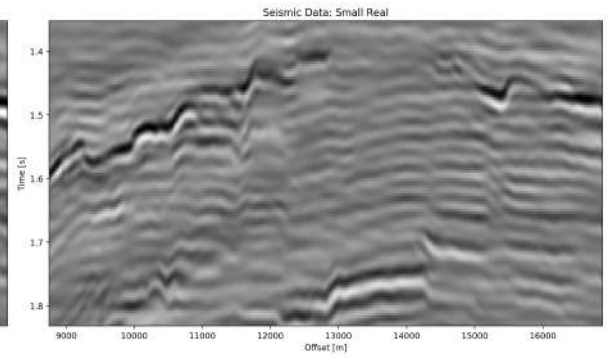
The largest differences of the outputs in real-valued and complex-valued networks can be observed in discontinuous areas. Particularly, the faulted blocks in the top quarter and in the bottom center of the seismic section show inconsistencies. The real-valued network smooths over discontinuities and steep reflectors. Fault lines are imaged better in the complex-valued network output.



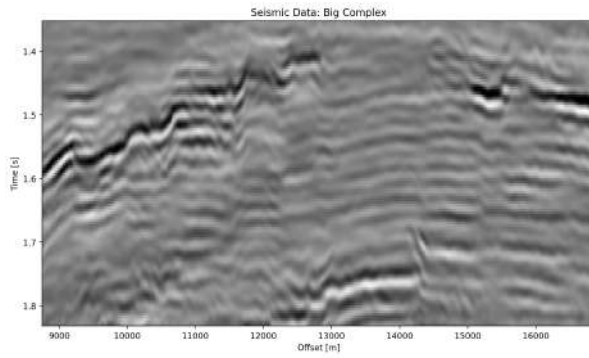
(a) Ground Truth



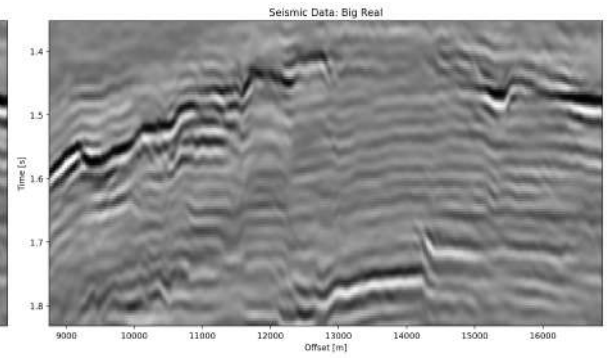
(b) Small Complex Network bottom Patch



(c) Small Real Network bottom Patch



(d) Large Complex Network bottom Patch



(e) Large Real Network bottom Patch

Figure 5.9: Evaluation on bottom Noise Patch. Data is normalized between -1 and 1.

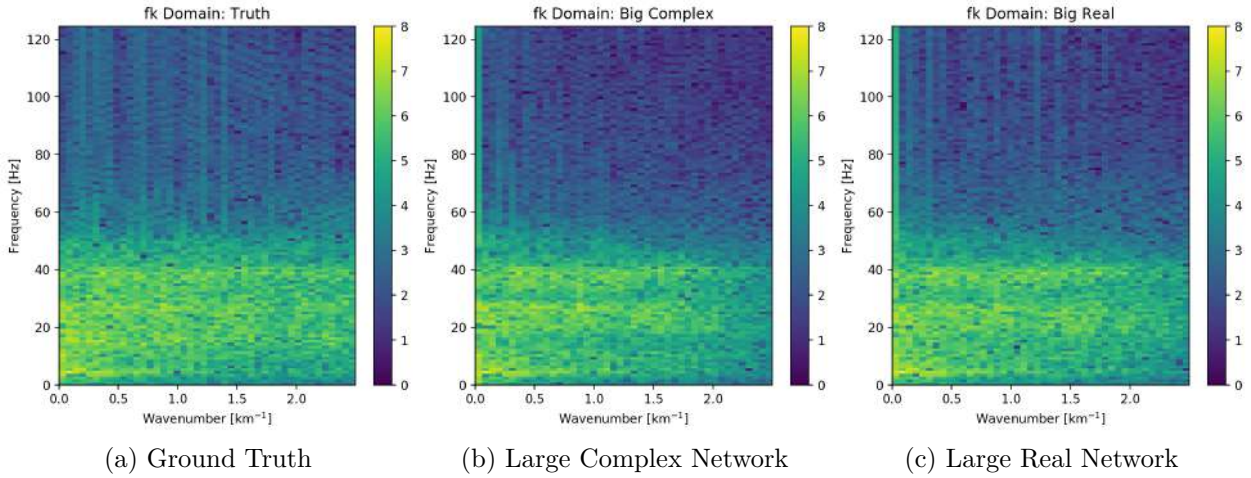


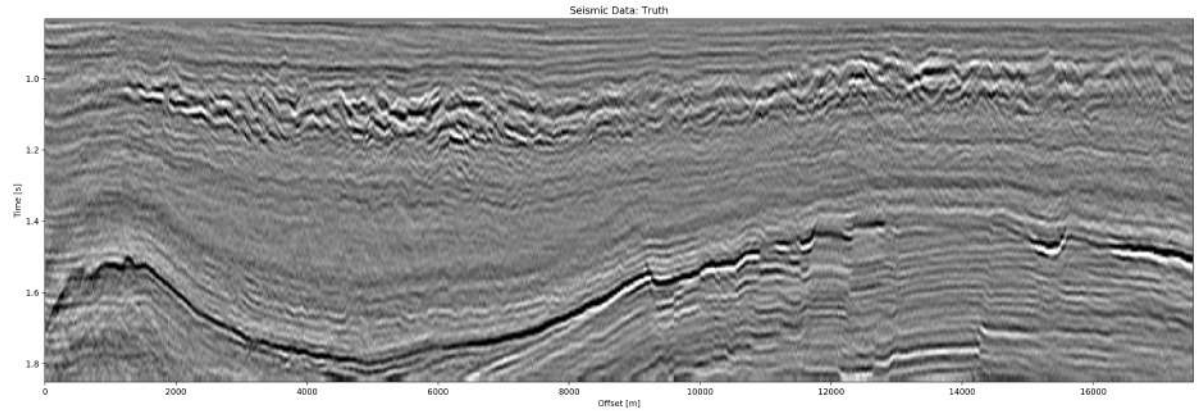
Figure 5.10: FK domain of full seismic data.

In seismic data processing, including phase information stabilizes discontinuities and disambiguates cycle-skipping in horizons. This could be observed in the network performance and reconstruction. The increase in performance of the real-valued networks was significant (7.0 % RMS), while the complex-valued networks already had an acceptable performance on the smaller network architecture (2.6 % RMS). We provide the complex-valued networks with a bias towards learning phase information, by providing the Hilbert transformed analytical trace, while the real-valued network needs to learn this information implicitly from the data itself. Considering, that during the training, the complex network evaluates both the real-valued seismic, which we primarily care about in addition to the complex-valued component, we can see how the losses in Figure 5.5 differ from the real-valued networks.

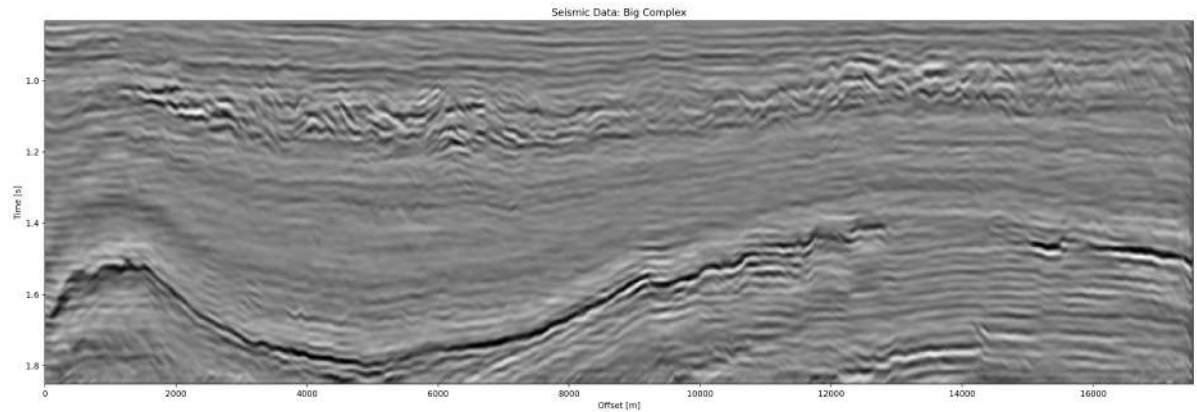
The largest network with 790,945 trainable parameters quantitatively performed the best on the reconstruction of the data. However, analysis of the reconstructed seismic shows, that while the high-amplitude regions are reconstructed to higher fidelity, discontinuous sections may be smeared by the real-valued network. The real-valued network that was matched to contain as many filters for the real-valued component of the seismic as the large complex-valued network, did not perform well. Furthermore, the smaller complex-valued network with 100,226 parameters that contains as many filter maps as the real-valued network in total, and half the trainable parameters, outperformed the smaller real-valued network across all test cases.

5.1.8 Conclusion

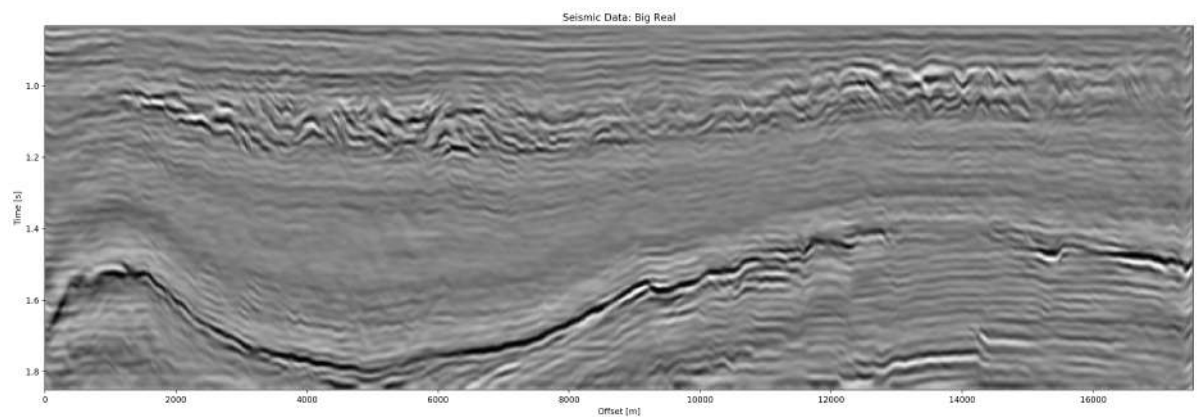
The inclusion of phase-information leads to a better representation of seismic data in convolutional neural networks. Complex-valued networks perform consistently, where real-valued networks have to learn phase-representations through implicit correlation, which requires larger networks. We show that complex trace information in deep neural



(a) Ground Truth



(b) Large Complex Network



(c) Large Real Network

Figure 5.11: Evaluation on full seismic data. Data is normalized between -1 and 1.

networks improves the imaging of discontinuities as well as steep reflectors, particularly in chaotic seismic textures that are smoothed by real-valued neural networks of the same size and level of compression.

We show that convolutional neural networks can perform lossy compression on seismic data, where the reconstruction error is dependent on both network architecture and implementation details, like providing explicit phase information. During this compression, noise and scattered energy get attenuated. The real-valued network is prone to introduce steeply dipping artifacts in the reconstruction and is matched by complex-valued networks half the size with twice the amount of compression. This is particularly interesting in the light of the complex complement of the data being derived from the real-valued data through a Hilbert transform, which should have been possible to approximate by a neural network.

The stabilization of the reconstruction can be useful in other seismic applications. While automatic seismic interpretation may benefit from the inclusion of information on discontinuities, we see the main application to be lossy seismic compression. The open source tool developed to make this research possible, enables further research and development of complex-valued solutions to non-stationary physics problems that benefit from explicit phase information.

This research also shows that a change as small as 2.5 % in RMS can change the reconstruction from being acceptable to very smeared to a geoscientist. This touches on the fact that better metrics to evaluate computer vision tasks in geoscience are necessary. Additionally, these tasks have to be noise-robust and, while amplitude-preserving, be robust against outliers. Moreover, more research in the frequency dimming of bands in the network reconstruction is necessary.

Overall, the computational memory footprint of the complex convolution is higher than real-valued convolutional neural networks comparing singular convolutional operations. A significant increase in depth and width of networks to obtain an acceptable result in real-valued neural network to implicitly learn the phase information is necessary. The complex-valued networks an 8th of the size already performs well, suggesting that domains where a significant part of the information is in the phase of signals, could benefit from applying complex convolutional networks.

5.1.9 Acknowledgments

We thank Andrew Ferlitsch for his valuable insights. The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding program. We thank DTU Compute for access to the GPU Cluster.

5.2 Contributions of this Study

This chapter and Drams et al. (2019f) investigate the application of complex trace analysis to Convolutional Neural Networks. It uses lossy compression to measure the reconstruction error and therefore, the informational content in complex-valued NNs. We were able to show that networks containing phase information in the complex complement of data reduce the error as compared to real-valued networks. Moreover, the code to reproduce the findings in this paper (Drams, 2019b), as well as, a standalone Python library for complex-valued Convolutional Neural Networks in tensorflow has been made available as Free Open Source Software (Drams et al., 2019c).

CHAPTER 6

Machine Learning in 4D Seismic Inversion

This chapter discusses a neural network application to approximate the pressure-saturation inversion of 4D seismic data. It contains two workshop papers that discuss two different aspects of the construction of the neural network architecture. Traditionally, 4D seismic Quantitative Interpretation (QI) often relies on priors to reduce variance in the face of uncertainty. The inversion problem in this chapter is a pressure-saturation inversion from seismic amplitude difference maps in the Schiehallion field. The first paper presents an ablation study of the components in the architecture. The second paper discusses the neural network result and presents a comparison to a classical Bayesian inversion.

6.1 Data

The Schiehallion field is a stacked turbidite reservoir in the UK North Sea, which makes it very heterogeneous and compartmentalized. The T31 sandstone reservoir has the most lateral extent with the thickness ranging from 5 m to 30 m. The small thickness of the reservoir layer results in the entire reservoir being contained in a single trough of a seismic wavelet ($\approx \frac{1}{2}\lambda$), which has historically lead to applications using a 2D map view of the data. In order to make the results comparable, we treat the network as a 2D map instead of a 3D problem.

The data available consists of simulation and field data with several years of collected seismic data. The baseline acquisition is from 1996 with additional time steps acquired in 1999, 2000, 2002, 2004, 2006, 2008, and 2010. There are simulation results and measured amplitude difference maps. The simulated seismic data is based on pore volumes from previous pore volume inversions, pressure changes and saturation changes for water and gas. The ground truth pressure and saturation changes are not available for validation of the field data directly, which would be the ideal validation case.

Specifically, the seismic data consists of angle stacks in near, mid, and far. The reflectivity of seismic data can be angle-dependent, especially in the presence of fluids contained in the rock matrix. (Castagna et al., 1993). Angle stacks are constructed by selecting subsets of the full dataset to average data within defined bands of incidence angles. Commonly, angle stacks are constructed by stacking over the offset hence Amplitude versus Offset (AVO). The process of stacking the data, despite being partial stack

increases the Signal-to-Noise Ratio and is often necessary to obtain reliable results in 4D Quantitative Interpretation.

The simulation results are noise-free calculations with only a single simulation per year available. The recorded field seismic contains significant levels of noise. The seismic field data can therefore diverge from the theoretical prediction based on the pressure and saturation data. These fluctuations are not smooth across individual cells of the map, which can be seen in Figure 6.7.

The validation strategy in this problem setting is using one time step as a hold-out set that is not used during the training of the neural network. The time step used was recorded in the year 2004 and is presented in Figure 6.7. The remaining time steps are used during the training. Results in the paper are presented on the hold-out data.

6.2 Machine Learning Model

A primary application of Machine Learning is building regression models. The data available is not particularly abundant, which restricts the choice of model or training strategy. Following a premise of simplicity, a dense neural network was implemented, which treats each cell of a map independently. It is possible that a Convolutional Neural Network increases the performance, but due to the nature of deep Convolutional Neural Networks more training data needs to be generated.

In Dramsch et al. (2019e) we present a novel network structure that explicitly includes Amplitude versus Offset (AVO) gradient calculation within the network as physical knowledge, shown in Figure 6.6.

The network architecture was chosen to follow an encoder-decoder architecture as a forcing function for information distillation. The encoder decreases in size with each layer, gradually compressing the input data, while the decoder decompresses the data to the designated output (Dony et al., 1995). Conventionally, the middle layer is called "bottleneck" or "code layer" as it contains the compressed representation of the input data. Encoder-decoder architectures have found wide application in neural network applications that necessitate data transformation to a different representation (Worrall et al., 2017).

Additionally, the bottleneck layer is implemented as a variational encoding layer to be less susceptible to noisy input. The specific implementation is based on variational auto-encoders (Kingma et al., 2013). These replace the singular bottleneck layer with a number of layers that represent the parameters in a parametric probability distribution, most commonly the mean and variance of a Gaussian distribution $\mathcal{N}(\mu, \sigma)$. The encoder then informs the Gaussian distribution at the bottleneck and the decoder samples from the distribution during training. At inference, these networks commonly return the mean of the distribution. Neural networks are conventionally trained using stochastic gradient descent, which is not well-behaved calculating the derivative of a random node. Kingma et al. (2013) popularized the "reparameterization trick", which reformulates

$$z \sim P_{\phi}(z|x), \quad (6.1)$$

with z being the bottleneck, P being the probability of the distribution ϕ to approximate, and x being the data sample to

$$z = g(\phi, x, \epsilon) \quad (6.2)$$

where $g()$ is the functional representation of ϕ parameterized by μ and σ for a Gaussian distribution, and ϵ being a random sample from $\mathcal{N}(0, 1)$ that is the source of randomness in the bottleneck layer computing as $z = \mu + \sigma \cdot \epsilon$.

The pore volume is passed as-is to the network. The estimated pore volume helps the network to decouple the rock matrix from the fluid effects, which is further explored in Section 6.4. A schematic of the network is shown in Figure 6.6, which shows the connections of the individual operations.

The network explicitly includes AVO gradient calculation in the network architecture, considering it is physical knowledge we know will stabilize pressure and saturation change separation. Including basic physics knowledge leads to the network learning residual information, essentially defining another forcing function for the networks learning process. The AVO gradient can be calculated explicitly as input to the network. However, performing the AVO gradient calculation within the network enables programmatic augmentation of the input data during training. This implies that instead of learning one pre-computed AVO relation, we can perform data augmentation of the input data and train on a significantly higher amount of correctly calculated AVO gradients. This strategy can significantly improve the training strategy.

6.3 Training the Deep Neural Network for 4D Seismic Inversion

The model training is carried out in multiple phases. The first phase solely trains on un-augmented simulation data to determine an ideal network structure. The second phase trains on the fixed architecture with data augmentation to transfer the network to noisy field data. The network is optimized on standard Mean Squared Error while monitoring the R²-score.

The initial phase was carried out on simulation data with the data split into one part for training and a separate data set for validation. The seismic data from 2004 was held out as a test set. Neural Architecture Search was applied to the network to determine depth and width of the architecture, using a Tree of Parzen Estimator (TPE) hyperparameter search (Bergstra et al., 2015). This ensures an architecture in a controlled test environment on simulation data that is optimized for the complexity of the data.

In the second phase, to transfer the network to field data, the input of the network was combined with additive Gaussian noise (Bishop, 1995) to train the network for noisy field data input. The noise level was estimated in a manual process. Therefore, including the AVO calculation within the network forces the network to learn noisy AVO gradients that correspond to the augmented input. This process reduces the R²-Score and Mean

Squared Error, which is an expected effect of noisy regression data (Hastie et al., 2009). Nevertheless, this produces consistent results on field data upon visual inspection.

The paper “Including Physics in Deep Learning – An Example from 4D Seismic Pressure Saturation Inversion” in Section 6.4 provides an ablation study, where parts of the neural network architecture are systematically switched off. Ablation studies are commonly used to explore and evaluate the effect of the individual components on the regression result. The paper “Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data” in Section 6.5 shows the results of the deep neural network compared to a Bayesian inversion.

6.4 Workshop Paper: Including Physics in Deep Learning – An example from 4D seismic pressure saturation inversion

Abstract: In this work we present a deep neural network inversion on map-based 4D seismic data for pressure and saturation. We present a novel neural network architecture that trains on synthetic data and provides insights into observed field seismic. The network explicitly includes AVO gradient calculation within the network as physical knowledge to stabilize pressure and saturation changes separation. We apply the method to Schiehallion field data and go on to compare the results to Bayesian inversion results. Despite not using convolutional neural networks for spatial information, we produce maps with good signal to noise ratio and coherency.

J. S. Dramsch, G. Corte, H. Amini, M. Luthje, and C. MacBeth (2019d). “Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data”. In: *Second EAGE Workshop Practical Reservoir Monitoring 2019*. Published, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201900028

6.4.1 Introduction

Physics in machine learning often relies on transformations of data to beneficial domains and simulating additional data. Karpatne et al. (2017) show a physics-guided approach to model lake temperatures with neural networks. Schütt et al. (2017b) use deep neural networks to model molecule energies and Oliveira et al. (2017) employ a special architecture to capture scatter patterns in high-energy physics. When building deep learning pipelines, we can make informed choices in data modeling, but also build neural networks to maximize information gain on the available data. Ulyanov et al. (2018) has shown that the network architecture itself can be used as prior in machine learning. These approaches translate well to geoscience, where strong priors are often necessary to inform decisions.

Deep learning has revolutionized machine learning by replacing the feature generation and augmentation step by learned internal representations of features that maximize information gain. On image data analysis of these neural network filters have shown close relations to edge filters and color separators (Grün et al., 2016). Dramsch et al. (2018d) have shown that these filters translate well to seismic data. However, classic feed-forward neural networks do not have the benefit of learning filters. However, these neural networks benefit from recent improvements for regularization (Ioffe et al., 2015), non-saturating and non-vanishing gradients (He et al., 2015), and training on GPUs.

Neural networks for inversion of seismic data have a long history (Roeth et al., 1994). In Dramsch et al., 2019a we show the application of a deep multi-layer perceptron for

map-based 4D seismic pressure saturation inversion. In this work we show the information gain of feed-forward multi-layer perceptron neural networks by including an explicit calculation of the AVO gradient within the network architecture. It's exemplary for including domain knowledge as a prior in machine learning.

6.4.2 Method

We build a deep feed-forward network to invert seismic amplitude maps for pressure and saturation changes. We use the high-level Python framework `keras` with a `tensorflow` backend. The neural network was trained on synthetic data, to subsequently predict field data. The network takes the seismic input samplewise with near, mid, and far stacks, and pore volume. We inject 20% Gaussian noise to model the noisier field data directly after the input layer. This is fed to a custom layer that calculates the PP AVO gradient between far-mid, mid-near, and far-near. The main components are as follows:

6.4.2.1 Gaussian noise injection

The synthetic model is noise-free. While we get good results on the training data and the modelled test data, the network does not transfer well to noisy field data. Although the 4D NRMS is very low in the data set, the sample-wise fluctuations in the field seismic differ significantly from the synthetic data. We apply additive Gaussian noise with $\sigma = .02$ to the seismic inputs separately to simulate independent fluctuations of the seismic maps. This significantly decreases the training and validation performance on noise free synthetic data. On field data, however, this enables good transfer of the neural network.

```
1 noisy_input = GaussianNoise(0.02)(input_data)
```

6.4.2.2 Explicit AVO gradient calculation

The Schiehallion field is a good example of imbalanced learning. We have many samples of pressure changes ΔP , a good selection of water saturation changes ΔS_w , and very few gas saturation changes ΔS_g . Yet, the changes in gas saturation ΔS_g produce the strongest changes in seismic P wave amplitudes. Statistically, these can easily be regarded as outliers, and therefore, possibly disregarded by the neural network. From decades of seismic analysis, we know that the AVO gradient is very good for pressure saturation separation. We implement an explicit calculation of AVO gradients in the network.

$$G = \frac{A_{\Theta_1} - A_{\Theta_0}}{x_{\Theta_1} - x_{\Theta_0}}, \quad (6.3)$$

where G is the PP AVO gradient, A is the seismic P wave amplitude, x is the offset, and Θ is the angle.


```

1 mid_near = Lambda(
2     lambda inputs: (inputs[0] - inputs[1]) / (10)
3 )([noisy_mid, noisy_near])
4
5 far_mid = Lambda(
6     lambda inputs: (inputs[0] - inputs[1]) / (10)
7 )([noisy_far, noisy_mid])
8
9 far_near = Lambda(
10    lambda inputs: (inputs[0] - inputs[1]) / (20)
11 )([noisy_far, noisy_near])

```

6.4.2.3 Encoder-decoder architecture

Subsequently, the four input maps and the three gradient maps are concatenated and fed to an encoder architecture that condenses the information to an embedding layer z . This layer learns a collection of Gaussian distributions to represent the noisy input data. The decoder samples this variational embedding layer to calculate the pressure change ΔP , change in water saturation ΔS_w , and gas saturation ΔS_g .

The full architecture is of the encoder-decoder class. The encoder reduces the number of parameters with each subsequent layer. This forces the network to learn a lossy compression of the input data as z -vector. The decoder increases the number of nodes per layer toward the output. The network therefore learns to correlate the low resolution representation with the desired output.

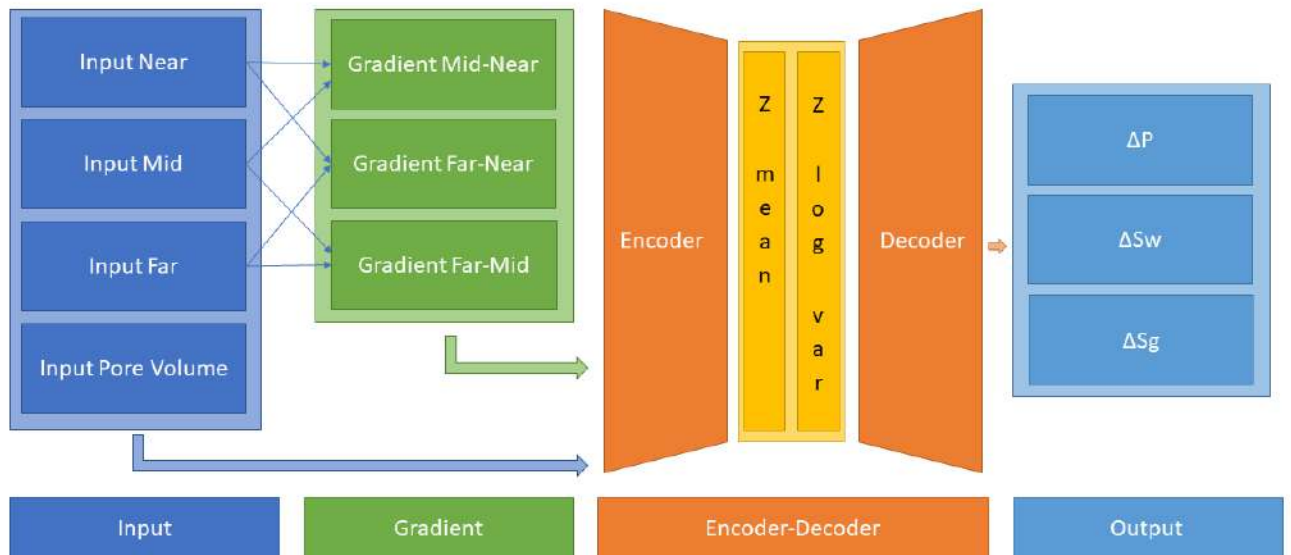


Figure 6.1: Full Architecture from Dramsch et al. (2019a).

6.4.2.4 Variational Z Vector

The inversion of noisy input benefits from a variational representation of compressed z-vector. The network learns Gaussian distributions in the embedding layer. Therefore, we have to apply the reparametrization trick outlined in Kingma et al. (2013) to circumvent the sampling process cannot be learned by gradient descent. We use the implementation in Chollet et al. (2015b) for variational autoencoders.

6.4.3 Results

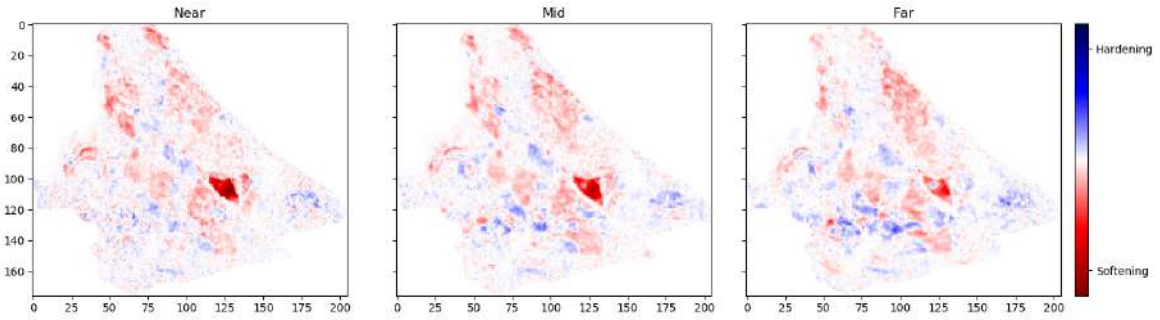


Figure 6.2: Schiehallion 2004 Timestep Seismic data, pore volume and sim2seis results.

In figure 6.2 we show the 2004 time step of the Schiehallion 4D. Figure 6.3 contains the inversion result using the variational encoder decoder architecture. Some coherency in the maps can be seen, but each map is very noisy and the gas saturation map contains many data points that indicate gas desaturation, which cannot be confirmed by production data.

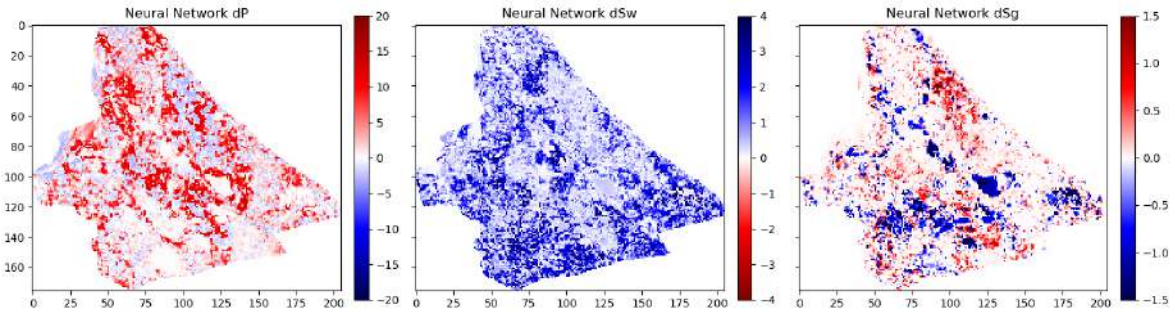


Figure 6.3: Variational Encoder Decoder Architecture Inversion

When we add the gradient, we can clean up some of the misfit in the gas saturation maps ΔS_g . Particularly, the event with the strongest softening in the amplitude maps, is partially reassigned to the pressure map ΔP . However, the inversion process is still very prone to noise. In figure 6.5, we show the inversion results of a AVO-gradient neural network with a noise injection at training of $\sigma = .02$. The inversion maps are very coherent. Noise injection without gradient calculation does not give adequate results.

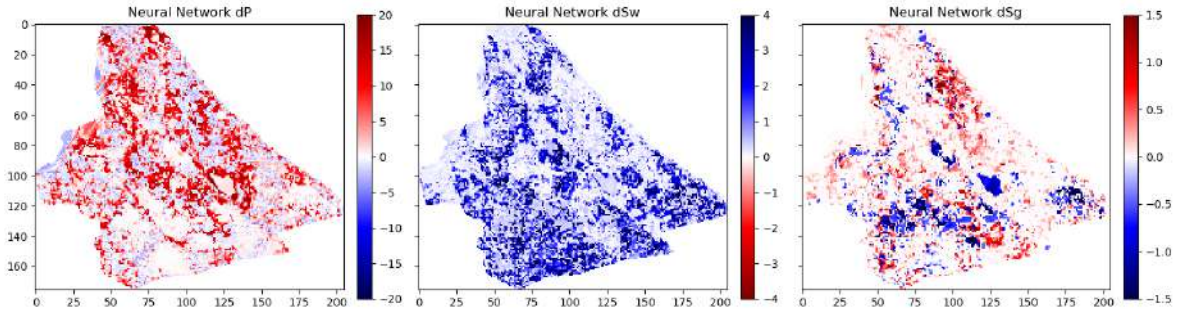


Figure 6.4: AVO-Gradient Variational Encoder Decoder Architecture Inversion

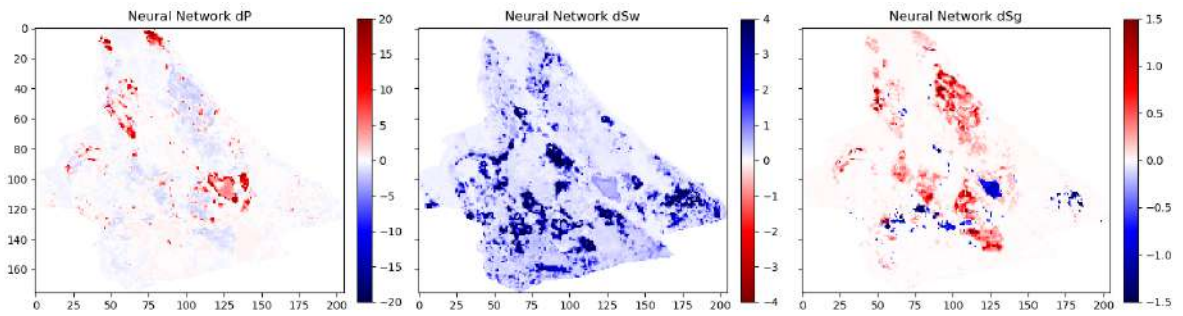


Figure 6.5: Noiseinjected AVO-Gradient Variational Encoder Decoder Architecture Inversion

6.4.4 Conclusions

We have shown a neural network architecture that incorporates physical domain knowledge to enable transfer from synthetic to field data. The final inversion result has very good coherency, despite the network not having any spatial context. While further investigation is necessary, this indicates that useful information has been learned. This is one example, where bias can be intentionally introduced into the network architecture to include physics into machine learning.

6.4.5 Acknowledgements

The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding program. We thank the sponsors of the Edinburgh Time-Lapse Project, Phase VII (AkerBP, BP, CGG, Chevron, ConocoPhillips, ENI, Equinor, ExxonMobil, Halliburton, Nexen, Norsar, OMV, Petrobras, Shell, Taqa, and Woodside) for supporting this research. The Brazilian governmental research-funding agency CNPq. We are also grateful to Linda Hodgson and Ross Walder for important discussions on the field and dataset.

6.5 Workshop Paper: Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data

Abstract: Geoscience data often have to rely on strong priors in the face of uncertainty. Additionally, we often try to detect or model anomalous sparse data that can appear as an outlier in machine learning models. These are classic examples of imbalanced learning. Approaching these problems can benefit from including prior information from physics models or transforming data to a beneficial domain. We show an example of including physical information in the architecture of a neural network as prior information. We go on to present noise injection at training time to successfully transfer the network from synthetic data to field data.

J. S. Dramsch, G. Corte, H. Amini, C. MacBeth, and M. L  thje (2019e). “Including Physics in Deep Learning – An Example from 4D Seismic Pressure Saturation Inversion”. In: *81st EAGE Conference and Exhibition 2019 Workshop Programme*. Published, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201901967. URL: <https://doi.org/10.3997/2214-4609.201901967>

6.5.1 Introduction

Estimating reservoir property change during a period of production from 4D seismic data has been a concentrated challenge and ambition for geoscientists in the oil and gas industry. These estimates can contribute to a better history matching of the reservoir simulation and for more comprehensive reservoir monitoring.

With the advance of machine learning techniques on all fronts in the geosciences we can address what roles machine learning can take in the established pressure and saturation inversion workflows and what other new workflows can be constructed using this tool. Machine learning is such a broad concept that it can be incorporated at different levels on all the current well established workflows to diminish their weaknesses, bringing more value to the pressure and saturation estimations from seismic inversion. Not only that, with this tool we can create completely new workflows that we are only beginning to grasp.

Here we will present results for two separate methodologies of seismic inversion to changes in pressure and saturation. The first method is a well established model-based Bayesian inversion method using a calibrated petro-elastic model and convolution workflow as the forward seismic modeling operator. In the second method we use a deep neural network to model the inversion process, we use synthetic seismic data to train

the network, then apply the inversion to observed data. The methods are applied to the same field data giving a nice platform to compare the neural network inversion results to a more conventional approach.

6.5.2 Schiehallion Data

The inversions are applied to maps of Schiehallion's upper T31 sandstone. It is a fairly thin reservoir (5-30m), which is well defined in the seismic data by one single trough. For this reason, a map-based approach is appropriate. Schiehallion is a highly compartmentalized field with initial pressure close to bubble point pressure. Production in this complex structure led to areas with strong pressurization due to water injection into closed compartments, while other areas lack the pressure support and experience gas release due to pressure depletion. We face the challenge of inverting 4D seismic data to changes in pressure, water saturation and gas saturation (ΔP , ΔS_w and ΔS_g), so the methods need to deal properly with the non-linearities due to each of these effects. The seismic data analysed is a set of eight vintages (from 1996 to 2010). These were reprocessed by CGG in 2014, following a 4D driven multi-vintage workflow. The processing workflow was carefully optimized to maintain 4D AVO amplitudes intact. Synthetic feasibility studies showed that the 4D AVO attributes are in line with the theoretical expectations. The seismic data used for inversion is the 4D difference of the sum of negative amplitudes (ΔSNA) map attribute, extracted from three angle-stacks, along the reservoir time window (see figure 6.7).

6.5.3 Method 1 - Model-based Bayesian inversion

The Bayesian inversion workflow is explained in detail in Corte et al. (submitted 2019). Essentially the workflow uses a petro-elastic model calibrated to the seismic data by Amini (2018) and a convolutional step to model the seismic data. The ΔSNA attribute is then extracted from the synthetic seismic and compared to the real seismic ΔSNA map. Since this is a map-based inversion, all realizations are sampled in map form and then go through a conversion into the vertical reservoir simulation grid in order to run the forward modelling process. We use a monte carlo sampling algorithm to generate thousands of realizations of the full map and from these extract best estimations and uncertainties. This inversion is constructed in a Bayesian model-based form, with the objective of bringing together information from the history matched reservoir simulation and seismic data. Reservoir simulation results for ΔP , ΔS_w and ΔS_g are incorporated as prior knowledge, to settle ambiguities and lack of seismic information. Where the seismic data lacks information about a certain property the method will bring this information from the simulation model. The inversion results will deviate from the simulation in areas where the seismic data contains enough consistent information to indicate an update is necessary.

6.5.4 Method 2 - Neural network inversion

We use a deep neural network to model the inversion process, based on the synthetic convolution seismic data. Although convolutional neural networks are considered the state of the art in spatially correlated data, we show that a sample-wise feed forward neural network trained on noise-free convolutional seismic can invert observed seismic data. We aim to build a regression model that can invert physical seismic angle stack data to pressure and saturation data.

Distinguishing pressure and saturation changes in 4D seismic data is a hard to solve problem. In neural networks, this is no different. The variation of data showing different pressure and saturation change scenarios is sparse, which complicates training and may possibly be disregarded as noise. This increases the need for training data immensely. However, we can include prior physical insights into neural networks to reduce the cost of training and uncertainty. As neural networks are at its basis very large mathematical functions, we can explicitly calculate the P-wave AVO gradient within the network to use as additional information source, without the need of feeding it into the network as input data. This has the added benefit of the network learning on noisy gradients. The design choice for the neural networks can be arbitrary, however, encoder-decoder networks have proven to force neural networks to find meaningful relationships within the data and reduce to these in the bottleneck or embedding layer. For the final architecture we used **hyperopt** (Bergstra et al., 2013) and **keras** (Chollet et al., 2015b). This allows us to use a Tree of Parzen (TPE) estimator for hyperparameter estimation. The estimator models $P(x|y)$ and $P(y)$, where y the quality of fit and x is the hyperparameter set drawn from a non-parametric density (Bergstra et al., 2011).

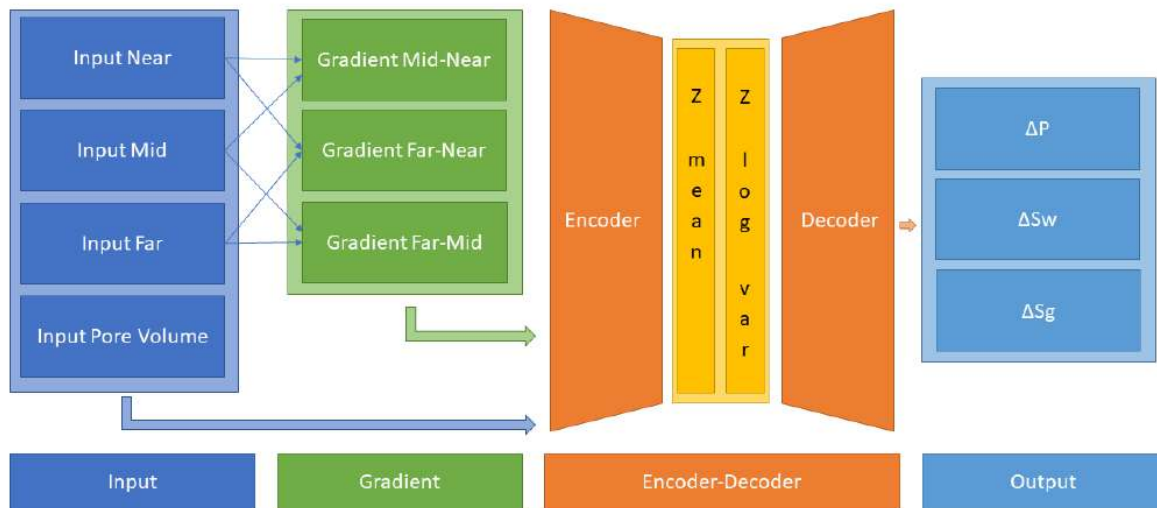


Figure 6.6: Architecture for sample-based seismic inversion with explicit gradient calculation.

The architecture is shown in figure 6.6. Inputs are Near, Mid, Far seismic, and Pore volume. These Input Layers are passed on to calculate the mid-near, far-mid, and far-

near gradients. These four inputs and three gradients are concatenated and fed to the encoder. z_mean and z_log_var build the variational embedding with z_Lambda being the sampler fed to the decoder network. The decoder splits into three output layers ΔP , ΔSw , and ΔSg .

The network is trained using sim2seis results calculated for the seven time-steps at seismic monitor acquisition times, it is then used to invert each seismic monitor individually. The inversion results for the synthetic data gave a consistent R^2 -score of over 0.6 for all simultaneous inversion targets ΔP , ΔSw and ΔSg with an encoder-decoder architecture and a deterministic embedding layer. While we kept the main architecture constant, we replaced the embedding layer with a variational formulation to allow for noise in the input to output mapping added noise injection to the input layer, to apply Gaussian Noise during the training phase. This significantly improved the inference on observed seismic data. The total training time for the network was 3 hours on a K5200 GPU, prediction speed takes $5.11\ s \pm 22.1\ ms$.

6.5.5 Schiehallion Field Data Example

The field data differs significantly from the synthetic data in that it is noisier, assuming the same ground truth. This is a true challenge for a sample-wise process to produce consistent results. We have trained the network with Gaussian noise on the input data with zero mean and a standard deviation of $\sigma = .02$, therefore, approximately 95 % of the noise may distort up to a maximum 40 % of the clean signal.

Figure 6.7 shows the observed 4D seismic maps (ΔSNA) for the 2004 monitor acquisition using the 1996 acquisition as baseline. Figure 6.8 shows, in the first row, the simulation model results (used in the Bayesian method as prior information), in the second row, the inversion results for the Bayesian method, and in the third row, the inversion results for the neural network method.

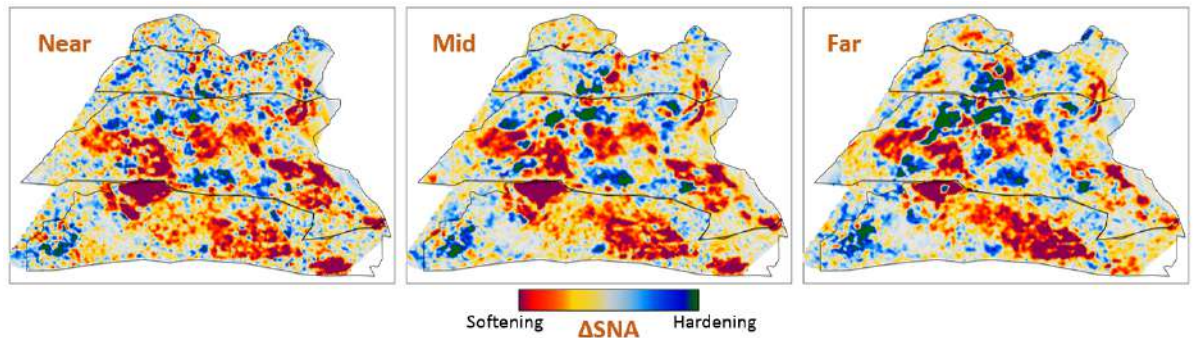


Figure 6.7: Schiehallion 2004 Timestep Seismic data, pore volume and sim2seis results.

From figure 6.8 we can see clearly the influence of the prior simulation model in the Bayesian results. The neural network does not use a prior, so the results are not influenced by the simulation model and can be seen as a direct interpretation of the seismic data. Comparing both we can see what bits of information the Bayesian method

is bringing from the prior. The seismic data is most sensitive to gas saturation changes, so the Bayesian method is able to capture this consistent information from seismic data and deviate ΔS_g results from the initial prior. The results for gas saturation are the most in agreement in both methods precisely because all this information is coming from the seismic data. We see some leakage of hardening effects into the ΔS_g results in method 2 due to the fact that we cannot set constraints to that inversion process. Since there is no initial gas saturation in those areas the saturation change cannot be negative, these comprehensive constraints are imbedded into the Bayesian workflow but not in the neural network.

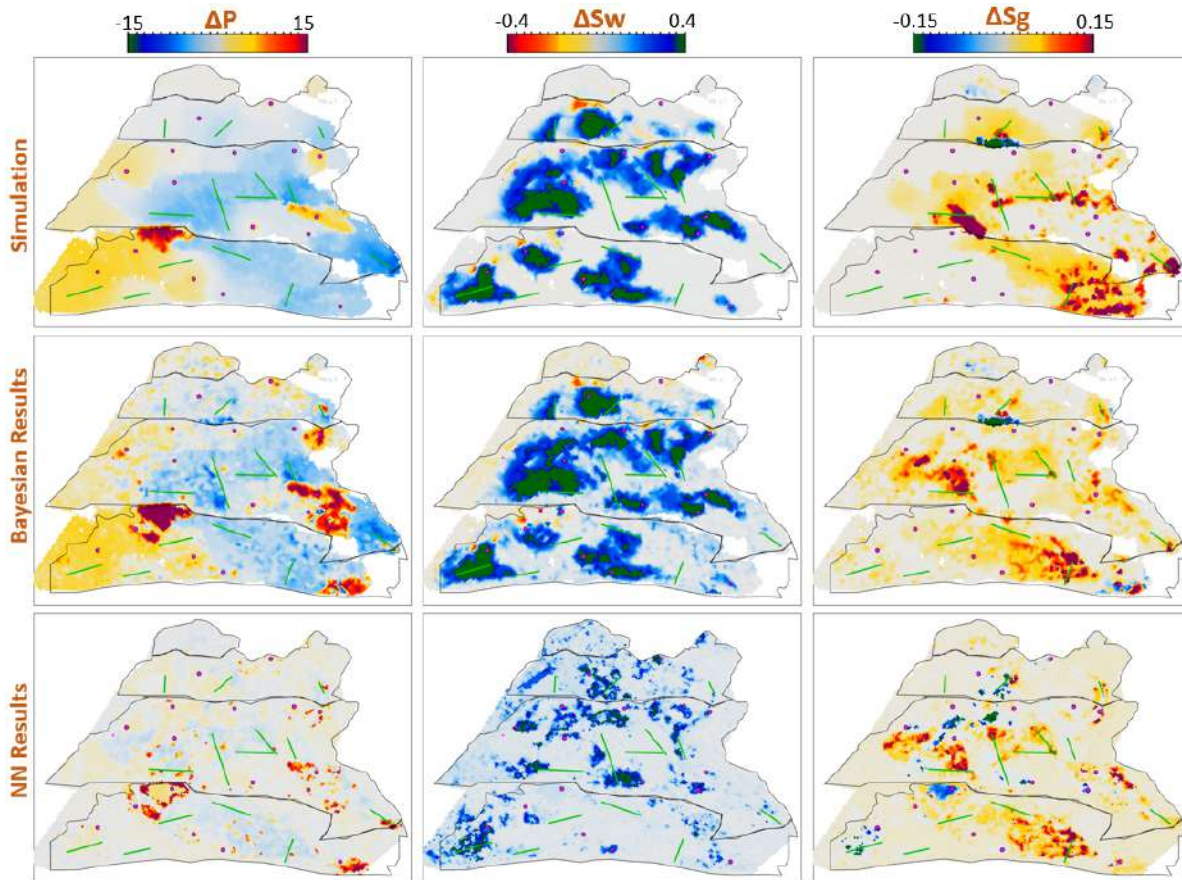


Figure 6.8: Schiehallion 2004 Timestep Bayesian Inversion and Neural Inversion

Water saturation has a distinctive hardening effect on seismic data, but in this map it is highly obscured by stronger overlying softening effects due to pressure increase and gas breakout. The neural network interprets all the hardening anomalies correctly as water saturation increase, while controlling for noise in areas of softening amplitudes. In those areas the seismic data does not contain useful information on the water saturation so the Bayesian result relies on a strong prior to compensate. All of the water saturation inverted by method 2 is in agreement with method 1, but since method 1 has this additional information from the prior, the map seems more coherent.

The pressure effect on seismic is highly non-linear. While high increases in pressure show a very strong softening effect, milder pressure variations (up to ± 7 MPa) have very little influence on the seismic data and are easily obscured by overlying effects. For this reason, the neural network pressure inversion in regions of mild change is low and often correlated with saturation. The Bayesian inversion benefits from the prior to fill those pressure values. This method does deviate from the prior in areas of strong softening signals due to pressure increase, and those areas are also correctly interpreted by the neural network inversion.

When we relax the prior of the Bayesian inversion, these results are very noisy in the pressure and water saturation estimates. In these areas the neural network inversion is robust to noise. During the neural network training the pore volume has shown to be important in guiding the inversion from the seismic data. Adding pore volume data adds a structural component to the neural inversion process, which improves the overall results from the sample-based method significantly.

6.5.6 Conclusions

This work presents Deep Neural Inversion of 4D seismic data. We compare the results with a Bayesian Inversion approach. We show that Deep Neural Networks can model seismic inversion trained on synthetic data. Explicit calculation of the P-wave AVO gradient within the network stabilizes the pressure-saturation separation within the network and Noise Injection enables the transfer to unseen seismic field data. Neural networks can be an important tool to investigate nascent information in 4D seismic data to improve inversion workflows and reduce uncertainty in seismic analysis.

The Neural Inversion can be used as a valuable tool to explore purely data-based inversion results in the presence of noise. It is able to translate the ambiguous seismic amplitudes into much more easily interpreted property maps. The value of the Bayesian inversion results presented is in combining all knowledge about the reservoir to create a general view of the reservoir dynamics. These results show the current understanding of reservoir dynamics updated by imprinting seismic information on top of the history matched simulation results.

6.5.7 Acknowledgements

The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding program. We thank the sponsors of the Edinburgh Time-Lapse Project, Phase VII (AkerBP, BP, CGG, Chevron, ConocoPhillips, ENI, Equinor, ExxonMobil, Halliburton, Nexen, Norsar, OMV, Petrobras, Shell, Taqa, and Woodside) for supporting this research. The Brazilian governmental research-funding agency CNPq. We are also grateful to Linda Hodgson and Ross Walder for important discussions on the field and dataset. We thank Mikael L  thje for valuable feedback.

6.6 Discussion of 4D Inversion

The workshop paper Dramsch et al. (2019d) contains the Neural Network results compared to the simulation and Bayesian inversion results, shown in Figure 6.8. This network does not calculate the inversion solution; it merely approximates the inverse problem. These initial results on limited training data show that a Neural Network can estimate pressure saturation information from field data, after training on simulation data.

The results presented in Figure 6.8 contain three indicators that the network learned a regression for the Schiehallion field. The network returns the overall trend in increase and decrease of pressure and saturation correctly. Additionally, the range of output values for the network is unconstrained, but the network calculates values in the ranges that are expected from the simulation and Bayesian inversion results. However, and more interestingly, the networks do not contain spatial information, being a feed-forward DNN not a CNN, yet returns continuous albeit noisy outputs when re-assembled into maps.

While the overall result is promising, regions of strong gas saturation changes present a problem. This could be due to problems in the modelling, as well as the fact, that they generate strong amplitude differences and are far in between, essentially behaving like outliers.

6.7 Contribution of this study

This study introduced a Deep Neural Network to approximate a 4D Quantitative Interpretation pressure-saturation inversion problem with a regression model. The contribution of this study is threefold in that it approximated the pressure-saturation inversion, included physical information in the network, and trained on simulation data and transferred to field data. The work included in this thesis are two workshop papers (Dramsch et al., 2019e; Dramsch et al., 2019d); however, a journal paper (Côte et al., 2020) and conference paper (Corte et al., 2020) have been published, resulting directly from this work.

CHAPTER 7

3D Time Warping for 4D Data

This chapter consists of the submitted journal paper “Deep Unsupervised 4D Seismic 3D Time-Shift Estimation with Convolutional Neural Networks” (Dramsch et al., 2019b). This paper presents a novel 3D warping technique for the estimation of 4D seismic time-shifts. The algorithm is unsupervised and provides 3D time shifts with uncertainty measures. The unsupervised nature of this algorithm avoids biasing the Machine Learning model with ground truth data from existing time-shift extraction algorithms.

4D seismic time shift extraction is often performed in 1D due to time constraints and often sub-par performance of 3D algorithms (Hatchell et al., 2005b). In geologically complex systems and pre-stack time-shifts, these simple approaches often break down and obtaining 3D time-shifts is beneficial. This chapter explores and summarizes conventional 3D warping methods and machine learning approaches. The paper Dramsch et al. (2019b) in this chapter adapts the medical Voxelmorph algorithm to match 4D seismic data volumes in 3D.

Common 1D approaches include local 1D cross-correlation, dynamic time warping (Hale, 2013c), optical flow methods and methods based on Taylor expansion (Zabihi Naeini et al., 2009). 3D methods include Dynamic Image Warping (DIW) (Hale, 2013a), which expands dynamic time warping to two and three dimensions respectively. DIW is, however, at its core a depth-wise method that then gets smoothed across trace-wise matches. 3D local cross-correlation defines a multi-dimensional cross-correlation in a fixed Gaussian window to make the problem computationally feasible. The method requires processing of the seismic images to perform reasonably, usually smoothing and spectral whitening. Rickett et al. (2007a) introduce a non-linear inversion-based time-shift extraction in 3D. Cherrett et al. (2011) further develop a geostatistical inversion combining data constraints with geostatistical information in a Bayesian inversion scheme.

Zitova et al. (2003) review the rich history of medical registration methods that partially overlap with 4D seismic methods. These methods include affine transformations, piece-wise linear transformations (Goshtasby, 1988), radial basis function-based methods (Zitova et al., 2003), and elastic deformations (Bajcsy et al., 1989). The method most relevant to this paper is Large Deformation Diffeomorphic Metric Mapping (LDDMM) (Beg et al., 2005), which has not found application in 4D seismic, due to being computationally expensive. The method finds a combination of diffeomorphisms, which will be introduced in Section 7.1, through the deformation field of two images. LDDMM then

finds the shortest path of these diffeomorphisms iteratively.

7.1 Diffeomorphisms

In simple terms a diffeomorphism is a smooth transformation of an image, i.e. no discontinuities or holes are introduced. In the following we will constrain ourselves to \mathbb{R}^3 for brevity's sake. We define two images B, M and assume B is a random deformation of M , then $B \in \mathcal{B} := \{B = M \circ \varphi, \varphi \in Diff_V\}$, with φ being a diffeomorphic flow from $Diff_V$. Diffeomorphisms in \mathbb{R}^3 are a group of bijective, smooth transformations of local areas in dense images generated as smooth flows $\phi_t, t \in [0, 1]$, with $\varphi := \phi_1, \phi_0 := \text{id}$ (Beg et al., 2005). They satisfy the Lagrangian and Eulerian specification of the flow field for diffeomorphisms associated with the Ordinary Differential Equation (ODE)

$$\frac{d\phi_t}{dt} = v_t \circ \phi_t, \phi_0 = \text{id}, t \in [0, 1], \quad (7.1)$$

with ϕ being the smooth flow, where $\dot{\phi}_t \in \mathbb{R}^3$ are the Lagrangian vector fields and v the Euclidean velocities of the system. ϕ_0 is determined to be the identity transformation. Beg et al. (2005) approached this problem as a variational problem, whereas Miller et al. (2015) reformulated as a Hamiltonian optimal control problem on the variational objective. The variational objective for densely matched images B and M as is the case in seismic data following can then be defined as minimizing the Cost C of a given vector field v

$$\min_v C(v) := \frac{1}{2} \int_0^1 (Av_t | v_t) dt + \frac{1}{2\sigma^2} \|B \circ \phi_1^{-1} - M\|^2 \quad (7.2)$$

for images $B, M: \mathbb{R}^3 \rightarrow \mathbb{R}^+$, $\phi \cdot B := B \circ \phi^{-1}$. Here A is the one-to-one matrix linear differential operator such that $A: V \rightarrow V^*$, which enforces the smoothness constraint by modelling the norm $(V, \|\cdot\|_V)$. σ represents vector elements in the dual space V^* , which in this case are generalized functions which represent the conjugate momentum representations of the system. They act on smooth vector functions $f \in V$ further following Miller et al. (2015) to provide energy with $(\sigma | f) := \int_X \vec{f}(x) \cdot \vec{\sigma}(dx)$. It follows that Av can be interpreted as the Eulerian momentum. Allowing Av to be singular implies that coordinates can be displaced homogeneously by a singular momentum. Then (7.2) can be interpreted as minimizing two objectives, namely the action integral of kinetic energy and the endpoint matching. This is equivalent to finding the aforementioned shortest path of diffeomorphisms, while matching the resulting image as closely as possible.

7.2 Image Matching Algorithms

Machine learning-based methods within computer vision are mostly applied in image- and video-processing applications. Supervised methods largely work off the assumptions in Optical Flow (Dosovitskiy et al., 2015; Ranjan et al., 2017). FlowNet (Dosovitskiy

et al., 2015) implements an Encoder-Decoder CNN architecture. It has reached wide reception in the field, and several modifications were implemented; namely, FlowNet 2.0 (Ilg et al., 2017) improving accuracy, and LiteFlowNet (Hui et al., 2018) reducing the computational cost. SpyNet Ranjan et al., 2017 and PWC-Net (Sun et al., 2018) implement stacked coarse-to-fine networks for residual flow correction. PatchBatch (Gadot et al., 2016) and deep discrete flow (Güney et al., 2016) implement Siamese Networks (Chopra et al., 2005) to estimate the optical flow. Alternatively, DeepFlow (Weinzaepfel et al., 2013) attempts to extract large displacements optical flow using pyramids of Scale-Invariant Feature Transform (SIFT) features. These methods are prone to the same problems classic optical flow algorithms exhibit. Moreover, supervised methods necessitate ground truth time shifts. This leads to two problems; Either the model needs to be trained on synthetic data, where shifts are known and transfer model to field data, or we need to train the network on time shifts extracted by a different method. The implication of training a deep neural network on data extracted by a different method trains the network to include all assumptions the extraction methods make. Training on time shifts extracted by a 1D method would, therefore bias the network to return pseudo-trace-wise predictions.

Unsupervised methods include different approaches to the problem of image- and volume-matching. Meister et al. (2018) modifies the FlowNet architecture to an unsupervised optical flow estimator with bidirectional census loss called UnFlow. UnFlow makes several changes to the original optical flow formulation, which relax the illumination constraint (Stein, 2004). Bansal et al. (2018) implements a cycle-consistent generative adversarial network (Cycle-GAN) to interpolate video frames. This method is potentially promising but falls short due to training data constraints in seismic data. Video data contains at least 24 frames per second of video, which provides training data. One second of video, therefore, already contains more time steps than the best-covered field in 4D seismic data. Voxelmorph (Balakrishnan et al., 2019) implements a U-net (Ronneberger et al., 2015b) within an architecture that extracts a static velocity field, which is integrated to obtain a diffeomorphic warp field and performs a 3D interpolation to match the fields and trains unsupervised. This method significantly reduces the underlying assumptions necessary to make the network perform well on seismic data. The Voxelmorph algorithm is based on the diffeomorphic assumption, which constrains the solution space of the mapping. The main benefit of applying the diffeomorphic mapping to geoscience data comes in the fact that all diffeomorphisms are homeomorphic. The homeomorphic assumption transfers well to the geological reality that the mathematical topology stays constant, resulting in reflectors neither crossing nor generating loops.

The paper in (Dramschi et al., 2019b) applies the Voxelmorph architecture in Dalca et al. (2018b) to 4D seismic data. I make the network work on seismic data and train it on the Dan 1988-2005 seismic volumes in 3D. Seismic data is significantly larger than most brain scan data, which necessitates patch-based training of the network. I compare the obtained warp field to the best match, I could obtain using classic methods on the available data. The DIW match is sufficiently similar to the Voxelmorph warping field to warrant further investigation. The Voxelmorph architecture implements a subsampled flow field, which I replaced by a full U-Net that provides full-scale 3D flow fields

with uncertainties. The paper includes an investigation of the differences between the subsampled and full-scale flow fields. Moreover, I validate the unsupervised model on the same field with different seismic data, collected at different times, with differing seismic acquisition equipment, including different azimuths. Moreover, I test the model on a seismic data set from a different field, with different geology, acquisition, and year. Finally, the Machine Learning approach is compared to a time-shift field obtained with Dynamic Image Warping.

7.3 Dynamic Time and Image Warping

The paper in Dramsch et al. (2019b) uses Dynamic Time Warping (DTW) but does not expand on the method; hence an introduction to the algorithm is presented here. DTW is a signal processing tool for time series with the capability to match arbitrary time-series. Within geophysics it is applicable to 4D time shifts, seismic-well ties, well-to-well ties, and seismic pre- and post-stack migration (Hale, 2013b; Luo et al., 2014). Dynamic Time Warping itself is a dynamic programming problem described in Algorithm 1.

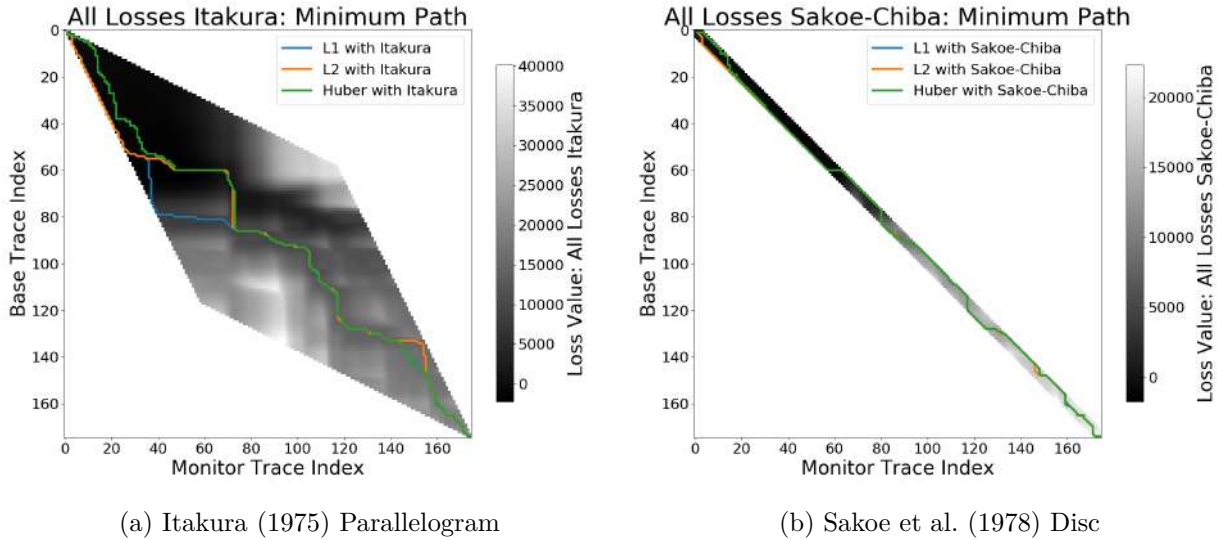


Figure 7.1: Minimum path for constraint masks for cumulative cost in DTW. Images show the optimum path for different loss functions, including L_1 , L_2 , and the Huber loss.

The DTW algorithm, represented in Algorithm 1, relies on calculating a distance matrix sample-wise between two traces a and b . Commonly, the L_1 norm is used to calculate the distance with $|b - a|$. Alternatively, the euclidean distance or L_2 norm can be used, which modifies the calculation to $(b - a)^2$. The difference between L_1 and L_2 is significant in the sense that the L_1 norm is not differentiable or convex; however, it scales linearly for outliers. The L_2 norm converges fast close to zero; however, the

error "explodes" for outliers. The Huber loss from convex optimization combines the advantages of the L_1 norm and L_2 norm

$$L_\delta(a, b) = \begin{cases} \frac{1}{2}(b - a)^2 & \text{for } |b - a| \leq \delta, \\ \delta(|b - a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases} \quad (7.3)$$

which is convex for small values, scales linearly for outliers and is differentiable for all values of \mathbb{R} , with δ being a scaling factor.

procedure DTW(a, b)

Given: Trace a and Trace b of lengths n .

function CALCULATE DISTANCE MATRIX $D(a, b)$

$D \leftarrow \text{dist}(a, b)$

end function

function CALCULATE CUMULATIVE COST $C(D)$

$C[0, 0] \leftarrow 0$

for $i = 1$ to n **do**

▷ Populate Edge

$C[0, i] \leftarrow D[0, i] + C[0, i - 1]$

$C[i, 0] \leftarrow D[i, 0] + C[i - 1, 0]$

end for

for $i = 1$ to n **do**

▷ Fill Cumulative Cost Matrix

for $j = 1$ to n **do**

$C_{min} \leftarrow \min\{C[i, j - 1], C[i - 1, j - 1], C[i - 1, j]\}$

$C[i, j] \leftarrow D[i, j] + C_{min}$

end for

end for

end function

function BACKTRACK MINIMUM COST PATH $P(C)$

$P \leftarrow C[n, n]$

while $i > 0 \mid j > 0$ **do**

$i, j \leftarrow \text{index}\{P[\text{last}]\}$

$C_{min} \leftarrow \min\{C[i, j - 1], C[i - 1, j - 1], C[i - 1, j]\}$

$P.\text{append} \leftarrow \text{index}\{C_{min}\}$

end while

end function

return P

end procedure

Algorithm 1: Dynamic Time Warping algorithm consists of calculating the element-wise distance matrix, cumulative cost and then find the optimal path in the cumulative cost matrix

Additionally, the search space on the cumulative distance matrix can be constrained to both increase performance and avoid non-optimal solutions. The different global constraint strategies are presented in Figure 7.1. The Itakura parallelogram (Itakura,

1975) in Figure 7.1a describes a parallelogram that has the largest width across the diagonal of the matrix, providing the highest degree of flexibility for the DTW algorithm in the centre parts of the seismic traces. The Sakoe-Chiba disc (Sakoe et al., 1978) follows a different strategy, which provides a constant maximum warp path. This strategy in Figure 7.1b introduces a global maximum time shift. Other constraints on the warp path in Dynamic Time Warping are local rate changes that limit the local changes, also called step patterns (Sakoe et al., 1978; Giorgino et al., 2009).

Dynamic Image Warping (DIW) is the extension of DTW to 2D and 3D datasets. Hale (2013b) introduced DIW for seismic data by applying the DTW algorithm in z-direction along the time-series and smoothing adjacent time-shifts to obtain consistent results. This process can be done iteratively with progressively smaller smoothing windows to obtain x-y consistent DIW results. It is important to note that DIW does not increase the computational cost of the DTW algorithm itself. Contrary to the intuition, the distance matrixes and cumulative cost presented in the are calculated in the same way resulting in a 2D cost matrix for each pair of 1D time series. However, the amount of comparisons of traces increases in 2D and 3D, scaling up the computational cost.

7.4 Journal Paper: Deep Unsupervised 4D Seismic 3D Time-Shift Estimation with Convolutional Neural Networks

Abstract: We present a novel 3D warping technique for the estimation of 4D seismic time-shift. This unsupervised method provides a diffeomorphic 3D time shift field that includes uncertainties, therefore it does not need prior time-shift data to be trained. This results in a widely applicable method in time-lapse seismic data analysis. We explore the generalization of the method to unseen data both in the same geological setting and in a different field, where the generalization error stays constant and within an acceptable range across test cases. We further explore upsampling of the warp field from a smaller network to decrease computational cost and see some deterioration of the warp field quality as a result.

J. S. Dramschi, A. N. Christensen, C. MacBeth, and M. L  thje (2019b). “Deep Unsupervised 4D Seismic 3D Time-Shift Estimation with Convolutional Neural Networks”. In: *IEEE Transactions in Geoscience and Remote Sensing*. In Review, Chapter 7

7.4.1 Introduction

Seismic time-lapse data consists of two 3D reflection amplitude cubes that represent the subsurface they were collected from. These cubes are acquired years apart with expected changes in the subsurface due to e.g. hydrocarbon production. The differences in the subsurface cause changes in both amplitudes and velocities, which introduces misalignment of seismic reflectors. Measuring the misalignment and aligning these surfaces to obtain a reliable difference cube is one of the main disciplines in 4D seismic processing.

These time shifts are most commonly obtained by windowed cross-correlation and other statistical or signal processing approaches (MacBeth et al., 2019). Considering the recent advances of machine learning in imaging and domain transfer, we explore possibilities of alignment with convolutional neural networks. Machine learning approaches, however, most commonly require labeled data to find a mapping $f(x) = y$, with x being the input data, f being the blackbox algorithm like a neural network, and y being the labels or target.

A common problem in machine learning for subsurface science is determining the ground truth. Obtaining information from the subsurface is often prohibited by cost, and e.g. core samples are highly localised data that is often altered by the extraction method as well as the sheer act of unearthing the sample. Additionally, synthetic data may introduce the inverse crime (Wirgin, 2004) of using the same theory to generate and invert data. Luckily, the physics of medical imaging and inversion is very similar to geophysics, where methods can be validated and fine-tuned. The main method discussed

in this paper is adapted from the medical imaging literature.

The lack of ground truths leads to another problem that deep learning address but do not solve. For classic neural networks, we need to know a target label dataset, i.e. knowing a prior warp velocity. In 4D seismic this would mean employing an established method to obtain time shifts. This would effectively result in abstracting that method in a neural network, or modelling the warp, which would lead to committing the inverse crime. Logically, this lead us to explore unsupervised methods.

We discuss several options for architectures for mapping the monitor seismic cube to the base seismic cube directly within the network. This is possible in unsupervised configurations but depending on the architecture of the network this problem can be ill-constrained and generate non-physical mappings. One warranted criticism of deep learning and neural networks is the lack of explainability and limited interpretability. However, we employ a deep neural network to obtain warp velocity vectors, a 3D equivalent of time shifts, for dense deterministic warping instead of directly obtaining the warped result from a neural network. This enables us to interpret the warping vectors and constrain the warp path in addition to the warp result.

Moreover, we present the first 4D seismic 3D time shift estimator with uncertainty measures. We achieve this by implementing a variational layer that samples from a Gaussian with the reparametrization trick (Kingma et al., 2015). Therefore, we can counteract some of the influence of noise on the performance of the network.

7.4.2 Theory

Extracting time shifts from 4D seismic data is most commonly done trace-wise (1D), which limits the problem to depth. This provides sufficient results for simple problems. However, geologically complex systems and pre-stack time shifts benefit from obtaining 3D time-shifts. We discuss classical 3D time-shift extraction methods, we then go on to discuss relevant deep learning methods. These methods extract time-shifts with different constraints which we explore. For brevity we present the results of the best method to date, developed for the medical domain: VoxelMorph (Balakrishnan et al., 2019).

The goal of both conventional and machine learning methods is to obtain a warp velocity field $\mathbf{u}(x, y, z)$ that ideally aligns two 3D cubes B and M within given constraints. That means a sample $m[x, y, z]$ will be aligned by adjusting $m[x + u_x, y + u_y, z + u_z]$. In image processing this is considered "dense alignment" or "dense warping", hence we need a dense vector field to align each sample in the base and the monitor cube. Generally, $\mathbf{u}(x, y, z) \in \mathbb{R}^3$, which implies interpolation to obtain the warped result.

7.4.2.1 Conventional Methods

Most conventional methods in 4D seismic warping focus on 1D methods (Hatchell et al., 2005b), which include local 1D cross-correlation, dynamic time warping (Hale, 2013c), optical flow methods and methods based on Taylor expansion (Zabihi Naeini et al., 2009).

We do not cover these methods in detail, but focus on the limited applications of 3D methods in 4D seismic warping.

Local 3D Cross Correlation Hall et al (Hall et al., 2005) introduced local 3D cross-correlation as a method for surface-based image alignment. The horizon-based nodal cross-correlation results were then linearly interpolated to full cubes. Hale et al (Hale, 2006) extended this method to full seismic cubes by calculating the multi-dimensional cross-correlation windowed by a Gaussian with a specified radius. The correlation results are normalized to avoid spurious correlations by amplitude fluctuations and high-amplitude events. Subsequently the cross-correlation result is searched for peaks using the following triple sum:

$$c[u_x, u_y, u_z] = \sum_{x,y,z=-\infty}^{\infty} b[x, y, z] \cdot m[x + u_x, y + u_y, z + u_z], \quad (7.4)$$

with c being the cross-correlation lag. The computational complexity of this method is $\mathcal{O}(N_s \times N_l)$ with N_s being the total number of samples and N_l being the total number of lags.

Stabilization of the results of 3D cross-correlation is obtained by applying spectral whitening of the signals and smoothing the images with a Gaussian filter without increasing the computational complexity despite the windowing function (Hale, 2006).

Inversion-based methods Rickett et al (Rickett et al., 2007a) describe a non-linear inversion approach, with the objective function being

$$\mathbb{E} = |\mathbf{d} - f(\mathbf{m})|^2 + |\nabla_x(\mathbf{m})|^2 + |\nabla_y(\mathbf{m})|^2 + |\nabla_z^2(\mathbf{m})|^2 \quad (7.5)$$

with \mathbf{m} being the model vector, \mathbf{d} being the data vector. The non-linear inversion is constrained by applying the first-derivative to the spatial dimensions z , y and Laplacian in z to obtain a smooth solution. Cherrett et al (Cherrett et al., 2011) implement a geostatistical joint inversion that uses the geostatistical information combined with data constraints as a prior in a Bayesian inversion scheme.

$$P(x|geostats, data) \propto \exp\left(-(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{C}^{-1}(\mathbf{x} - \boldsymbol{\mu})/2\right) \quad (7.6)$$

with \mathbf{C} being the posterior covariance matrix, \mathbf{x} the sample mean vector and $\boldsymbol{\mu}$ being the posterior mean vector.

Medical Imaging According to (Zitova et al., 2003), the rich history of medical image registration consists of four main steps, being feature detection, feature matching, transform model estimation, and image resampling and transformation. Within the scope of this paper, transform model estimation is the main interest, which defines a mapping function from the base image to the moving image. The transformation models fall into several general categories. Global Mapping Models define a global transformation of the entire image, which is unsuitable to this application of 4D seismic. Local

mapping models have been shown to outperform global methods (Zitova et al., 2003) and include piecewise mappings and weighted least squares (Goshtasby, 1988). Alternatively, transforming the moving image through radial basis functions and matching a globally linear model matches images with significant local distortion (Zitova et al., 2003). Finally, elastic matching presents a non-rigid registration method (Bajcsy et al., 1989) that finds an optimal matching between images according to intensity values and boundary conditions such as smoothness and stiffness of the matching vectors (Klein et al., 2009). Diffeomorphic mapping is not explicitly outlined in (Zitova et al., 2003), but particularly relevant to this paper. In (Christensen et al., 1994) large deformation flows were put forth that greedily find a path through diffeomorphic transformations. Diffeomorphisms have gained great attention in the medical field, particularly with large deformation diffeomorphic metric mapping (LDDMM) (Beg et al., 2005). This method iteratively finds the shortest path through small diffeomorphisms and is computationally expensive, which is a possible explanation that they have not found greater use in geophysics, due to larger datasets.

7.4.2.2 Machine Learning Methods

The machine learning methods discussed in this section are imaging based, and therefore rely on recent advances of convolutional neural networks (CNN) in deep learning. We discuss different approaches that include supervised and unsupervised / self-supervised methods. These methods are all based on convolutional neural networks (CNNs).

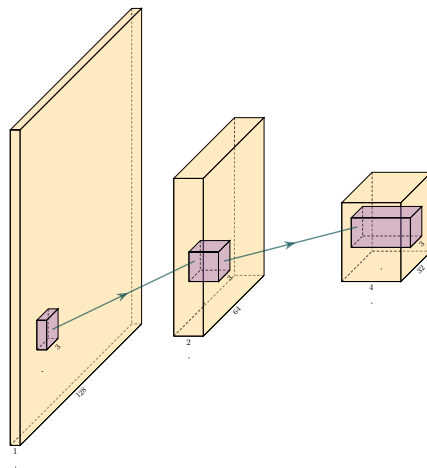


Figure 7.2: Schematic convolutional neural network. The input layer (yellow) is convolved with a 3×3 filter that results in a spatially subsampled subsequent layer that contains the filter responses. This second layer is again convolved with a 3×3 filter to obtain the next layer. Subsampling is achieved by strided convolutions or pooling.

CNNs are a type of neural network that is particularly suited to imaging approaches. They learn arbitrary data-dependent filters that are optimized based on the chosen objective via gradient descent. These filters can operate on real images, medical images,

or seismic data alike. The convolutional filter benefits from weight sharing, making the operation efficient and particularly suited to GPUs or specialized hardware. In Figure 7.2 we show a schematic image, that is convolved with moving 3x3 filters repeatedly to obtain a spatially downsampled representation. These convolutional layers in neural networks can be arranged in different architectures that we explore in the following analysis of prior methods in image alignment.

Supervised CNNs Supervised end-to-end CNNs rely on reliable ground truth, including the time shifts being available. Training a supervised machine learning system requires both a data vector x and a target vector y to train the blackbox system $f(x) \Rightarrow y$. This means that we have to provide extracted time-shifts from other methods, which implicitly introduce assumptions from that method into the supervised model. Alternatively, expensive synthetic models would be required.

The supervised methods are largely based on Optical Flow methods (Dosovitskiy et al., 2015; Ranjan et al., 2017). The FlowNet (Dosovitskiy et al., 2015) architecture is based on an Encoder-Decoder CNN architecture. Particularly, FlowNet has reached wide reception and several modifications were implemented, namely FlowNet 2.0 (Ilg et al., 2017) improving accuracy, and LiteFlowNet (Hui et al., 2018) reducing computational cost. SpyNet (Ranjan et al., 2017) and PWC-Net (Sun et al., 2018) implement stacked coarse-to-fine networks for residual flow correction. PatchBatch (Gadot et al., 2016) and deep discrete flow (Güney et al., 2016) implement Siamese Networks (Chopra et al., 2005) to estimate optical flow. Alternatively, DeepFlow (Weinzaepfel et al., 2013) attempts to extract large displacements optical flow using pyramids of SIFT features. These methods introduce varying types of network architectures, optimizations, and losses that attempt to solve the optical flow problem in computer vision.

Unsupervised CNNs Unsupervised or self-supervised CNNs only rely on the data, relaxing the necessity for ground truth time shifts. In (Meister et al., 2018) the FlowNet architecture is reformulated into an unsupervised optical flow estimator with bidirectional census loss called UnFlow. The UnFlow network relies on the smooth estimation of the forward and backward loss, then adds a consistency loss between the forward and backward loss and finally warps the monitor to the base image to obtain the final data loss. Optical flow has historically underperformed on seismic data, due to both smoothness and illumination constraints. However, UnFlow replaces the commonly used illumination loss by a ternary census loss (Zabih et al., 1994) with the ϵ -modification by (Stein, 2004). While this bears possible promise for seismic data, UnFlow implements 2D losses as opposed to a 3D implementation that we focus on.

Cycle-consistent Generative Adversarial Networks Cycle-GANs are a unsupervised implementation of Generative Adversarial Networks that are known for domain adaptation (Zhu et al., 2017). These implement two GAN networks that perform a forward and backward operation that implements a cycle-consistent loss in addition to the GAN loss. The warping problem can be reformulated as a domain adaptation problem.

This implements two Generator networks F and G and the according discriminators D_X and D_Y . These perform a mapping $G : X \rightarrow Y$ and $F : Y \rightarrow X$, trained via the GAN discrimination. The cycle-consistency implements $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ with the backwards cycle-consistency being $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$.

Cycle-GANs such as pix2pix (Isola et al., 2017) separate image data into a content vector and a texture vector, which could bear promise in the seismic domain, adapting a wavelet vector and an interval vector (Mosser et al., 2018c). However, the confounding of imaging effects, changing underlying geology, changing acquisition, etc makes the separation non-unique. Moreover, extracting the time shift information and conditioning in the GAN is a very complex problem. The Recycle-GAN (Bansal et al., 2018) addresses temporal continuity in videos, this is however hard to transfer to seismic data, considering the low number of time-steps in a 4D seismic survey as opposed to videos. Furthermore, the lack of interpretability of GANs at the point of writing, prohibits GANs from replacing many physics-based approaches, like the extraction of time-shifts.

7.4.3 Method

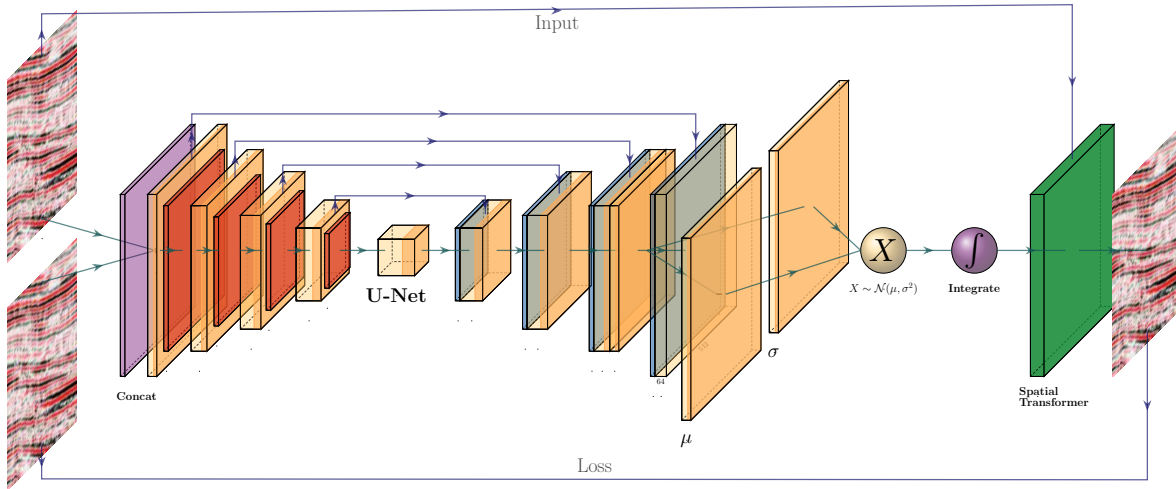


Figure 7.3: 2D representation of Modified 3D Voxelmorph architecture to obtain full scale warp velocity field. The Encoder side of the U-Net architecture consists of four consecutive Convolutional (orange) and Pooling (red) layers, followed by a convolutional Bottleneck layer. The decoder of the U-Net architecture consists of four Upsampling (blue) and Convolutional layers are connected to the respective same size layers in the Encoder. The output is passed to two convolutional layers that are sampled by the reparametrization trick, to provide the static velocity field. The field is integrated via scaling and squaring and passed to the Spatial Transformer layer (green), which transforms the monitor to optimally match the base image, which is enforced by minimizing the mean squared error (MSE) of the images.

The Voxelmorph (Balakrishnan et al., 2019) implements a U-net (Ronneberger et al., 2015b) architecture to obtain a dense warp velocity field and subsequently warps the monitor volume to match the base volume. This minimizes assumptions that have to be satisfied for applying optical flow-based methods. Additionally, the Voxelmorph architecture was specifically developed on medical data. Here we use an advancement of Voxelmorph that includes a variational layer, which introduced uncertainty to the static velocity estimation, developed in (Dalca et al., 2018b). Medical data often has fewer samples, like seismic data, as opposed to popular video datasets, which FlowNet and derivative architectures are geared towards application of popular video datasets. A U-net architecture is particularly suited for segmentation tasks and transformations with smaller than usual amounts of data, considering it was introduced on a small biomedical dataset. The short-cut concatenation between the input and output layers stabilizes training and avoids the vanishing gradient problem. It is particularly suited to stable training in this image matching architecture. In Figure 7.3 the U-Net is the left-most stack of layers, arranged in an hourglass architecture with shortcuts. These feed into a variational layer $\mathcal{N}(\mu, \sigma)$, the variational layer is sampled with the reparametrization trick, due to the sampler not being differentiable (Kingma et al., 2015). The resulting differential flow is integrated using the VecInt layer, which uses Scaling and Squaring (Higham, 2005). Subsequently, the data is passed into a spatial transformation layer. This layer transforms the monitor cube according to the warp velocity field obtained from the integrated sampler. The result is used to calculate the data loss between the warped image and the base cube.

More formally, we define two 3D images \mathbf{b}, \mathbf{m} being the base and monitor seismic respectively. We try to find a deformation field ϕ parameterized by the latent variable z such that $\phi_z : \mathbb{R}^3 \rightarrow \mathbb{R}^3$. The deformation field itself is defined by this ordinary differential equation (ODE) according to (Balakrishnan et al., 2019):

$$\frac{\partial \phi^{(t)}}{\partial t} = v(\phi^{(t)}), \quad (7.7)$$

where t is time, v is the stationary velocity and the following holds true $\phi^{(0)} = \mathbf{I}$. The integration of v over $t = [0, 1]$ provides $\phi^{(1)}$. This integration represents and implements the one-parameter diffeomorphism in this network architecture. The variational Voxelmorph formulation assumes an approximate posterior probability $q_\psi(z|\mathbf{b}; \mathbf{m})$, with ψ representing the parameterization. This posterior is modeled as a multivariate normal distribution with the covariance $\Sigma_{z|m,b}$ being diagonal:

$$q_\psi(z|\mathbf{b}; \mathbf{m}) = \mathcal{N}(z, \boldsymbol{\mu}_{z|m,b}, \Sigma_{z|m,b}), \quad (7.8)$$

the effects of this assumption are explored in (Dalca et al., 2018b).

The approximate posterior probability q_ψ is used to obtain the variational lower bound of the model evidence by minimizing the Kullback-Leibler (KL) divergence with $p(z|\mathbf{b}; \mathbf{m})$ being the intractable posterior probability. Following the full derivation in (Dalca et al., 2018b), considering the sampling of $z_k \sim q_\psi(z|\mathbf{b}, \mathbf{m})$ for each image pair

(\mathbf{b}, \mathbf{m}) , we compute $\mathbf{m} \circ \phi_{z_k}$ the warped image we obtain the loss:

$$\begin{aligned}
\mathcal{L}(\psi; \mathbf{b}, \mathbf{m}) &= -\mathbf{E}_q[\log p(\mathbf{b}|z; \mathbf{m})] \\
&\quad + \mathbf{KL}[q_\psi(z|\mathbf{b}; \mathbf{m})||p_\psi(z|\mathbf{b}; \mathbf{m})] \\
&\quad + \text{const} \\
&= \frac{1}{2\sigma^2 K} \sum_k \|\mathbf{b} - \mathbf{m} \circ \phi_{z_k}\|^2 \\
&\quad + \frac{1}{2} [\text{tr}(\lambda \mathbf{D} \Sigma_{z|x;y}) - \log \Sigma_{z|x;y}] \\
&\quad + \boldsymbol{\mu}_{z|m,b}^T \boldsymbol{\Lambda}_z \boldsymbol{\mu}_{z|m,b} + \text{const},
\end{aligned} \tag{7.9}$$

where Λ_z is a precision matrix, enforcing smoothness by the relationship $\Sigma_z^{-1} = \Lambda_z = \lambda \mathbf{L}$, λ controlling the scale of the velocity field. Furthermore, following (Dalca et al., 2018b) $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the Laplacian of a neighbourhood graph over the voxel grid, where \mathbf{D} is the graph degree matrix, and \mathbf{A} defining the voxel neighbourhood. K signifies the number of samples. We can express $\boldsymbol{\mu}_{z|m,b}$ and $\Sigma_{z|m,b}$ as variational layers in a neural network and sample from the distributions of these layers. Given the diagonal constraint on Σ , we define the variational layer as the according standard deviation σ of the corresponding dimension. Therefore, we sample $\mathcal{X} \sim \mathcal{N}(\mu, \sigma^2)$ using the reparameterization trick first implemented in variational auto-encoders (Kingma et al., 2013). The reparameterization trick defines a differentiable estimator for the variational lower bound, replacing the stochastic, non-differentiable and therefore untrainable, sampler.

Defining the architecture and losses as presented in (Dalca et al., 2018b), ensures several benefits. The registration of two images is domain-agnostic, which enables us to apply the medical algorithm to seismic data. The warp field is diffeomorphic, which ensures physically viable, topology-preserving warp velocity fields. Moreover, this method implements a variational formulation based on the covariance of the flow field. 3D warping with uncertainty measure has not been used in seismic data before.

The network is implemented using Tensorflow (Martín Abadi et al., 2015) and Keras (Chollet et al., 2015a). Our implementation is based on the original code in the Voxelmorph package (Dalca et al., 2018a).

7.4.4 Experimental Results and Discussion

7.4.4.1 Experimental Setup

The experimental setup for this paper is based on a variation of the modified Voxelmorph (Balakrishnan et al., 2019) formulation. We extended the network to accept patches of data, because our seismic cubes are generally larger than the medical brain scans and therefore exceed the memory limits of our GPUs. Moreover, Voxelmorph in its original formulation provides sub-sampled flow fields, this is due to computational constraints. We decided to modify the network to provide full-scale flow fields, despite the computational cost. This enables direct interpretation of the warp field, which is

common in 4D seismic analysis. However, we do provide an analysis in Section 7.4.4.2 of the sub-sampled flow-field interpolated to full scale, in the way it would be passed to the Spatial Transformer layer.

The code is made available in (Dramsch, 2020b). The model is trained with the Adam optimizer (Kingma et al., 2014) with a learning rate of 0.001 and weight decays $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We train the model for 350 epochs to account for experimentation and time. We set the regularization parameter $\lambda = 10$ and the image noise parameter $\sigma = 0.02$ in accordance with the authors of (Dalca et al., 2018b). We adjust the batch-size to the maximum on our architecture, which was 16 and purely manually tuned to the maximum possible. The KL divergence and MSE loss are unweighted in the total loss.

The network definition for the subsampled flow field differs from the definition in Figure 7.3 that the last upsampling and convolution layer in the Unet, including the skip connection, right before the variational layers (μ, σ) is omitted. That leaves the flow field at a subsampled map by a factor of two. Computationally, this lowers the cost on the Integration operation before resampling for the Spatial Transformer.

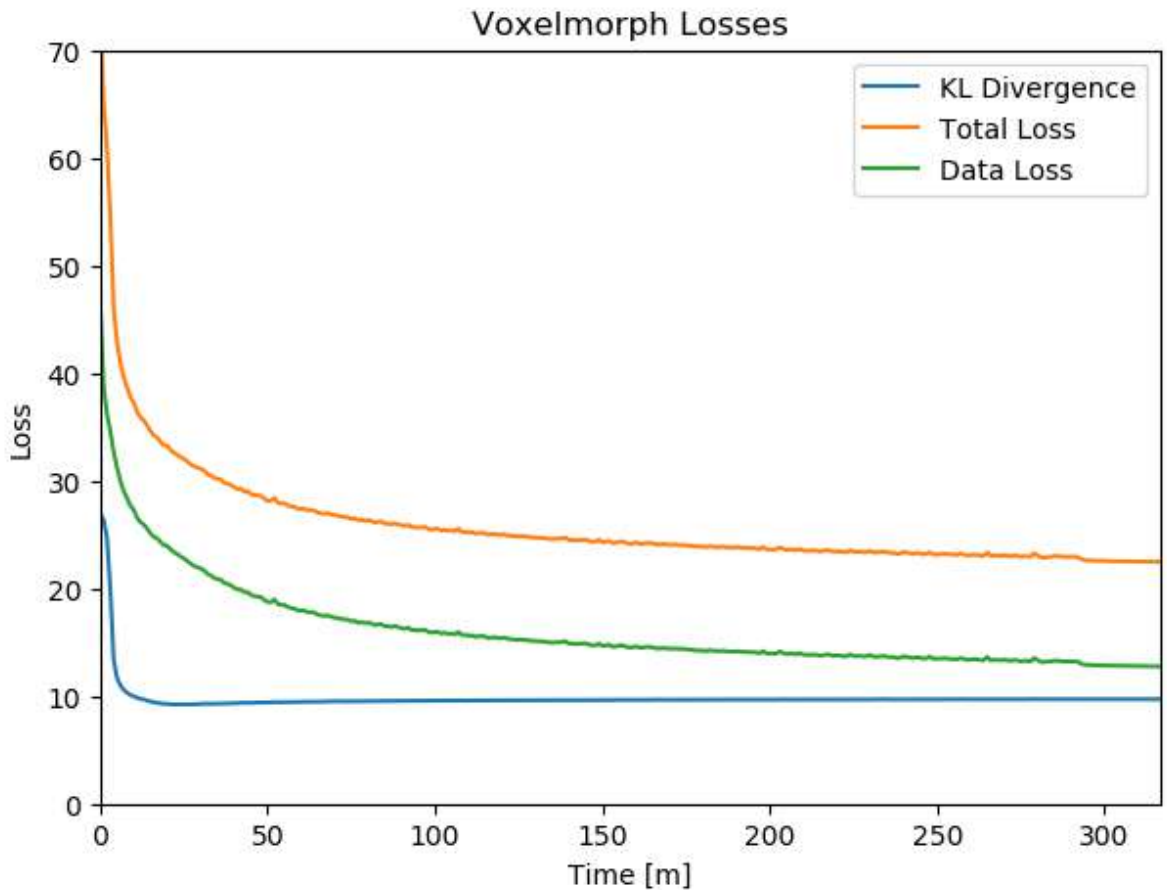


Figure 7.4: Training Losses over time with the KL-divergence at the sampling layer, the data loss calculated by MSE, and the combined total loss.

The data situation for this experiment is special in the sense that the method is self-supervised. We therefore do not provide a validation dataset during training. The data are 6 surveys from the North Sea. Main field from years 1988, 2005 A, 2005 B, and 2012. Further we compare to a different field 1993 and 2005 with different geology, acquisition geometry and acquisition parameters. While we would be content with the method working on the field data (years 1988 and 2005 Survey A) by itself, we do validate the results on separate data from the same field which was acquired with different acquisition parameters and at different times (years 2005 Survey B and 2012). Moreover, we test the data on seismic data from an adjacent field that was acquired independently (years 1993 and 2005). All data is presented with a relative coordinate system due to confidentiality, where 0 s on the y-axis does not represent the actual onset of the recording. The field geology and therefore seismic responses are very different. Due to lack of availability we do not test the trained network on land data or data from different parts of the world. Considering, that the training set is one 4D seismic monitor-base pair, a more robust network would emerge from training on a variety of different seismic volumes.

Figure 7.4 shows the training losses of the batch training. Within a few epochs the network converges strongly, however within 10 epochs the KL divergence increases slightly over the training. The data loss, optimizing the warping result decreases over the training period. An increase of the KL divergence is acceptable as long as the total loss decreases, which indicates better matching of the volumes. In case the KL divergence would increase vastly, it would violate the base assumption that the static velocity can be approximated by Gaussians and requires re-evaluation.

7.4.4.2 Results and Discussion

The network presented generates warp fields in three dimensions as well as uncertainty measures. We present results for three cases in Figure 7.5, 7.10, and 7.12 with the corresponding warp fields and uncertainties in Figure 7.6, 7.11, and 7.13. In Figure 7.5 we show the results on the data, which the unsupervised method was trained on. Obtaining a warp field on the data itself is a good result, however, we additionally explore the generalizability of the method. Considering the network is trained to find an optimum warp field for the data it was originally trained on, we go on to test the network on data from the same field, that was recorded with significantly different acquisition parameters in Figure 7.10. These results test the networks generalizability on co-located data, therefore not expecting vastly differing seismic responses from the subsurface itself. There are imaging differences and differences in equipment in addition to the 4D difference however. In Figure 7.12 we use the network on unseen data from a different field. The geometry of the field, as well as the acquisition parameters are different, making generalization a challenge.

In Figure 7.5 we collect six 2D panels from the 3D warping operation. In Figure 7.5(a) and Figure 7.5(b) we show the unaltered base and monitor respectively. The difference between the unaltered cubes is shown in Figure 7.5(e). In Figure 7.5(c) we show the warped result by applying the z-warp field in Figure 7.5(d), as well as the warp fields in

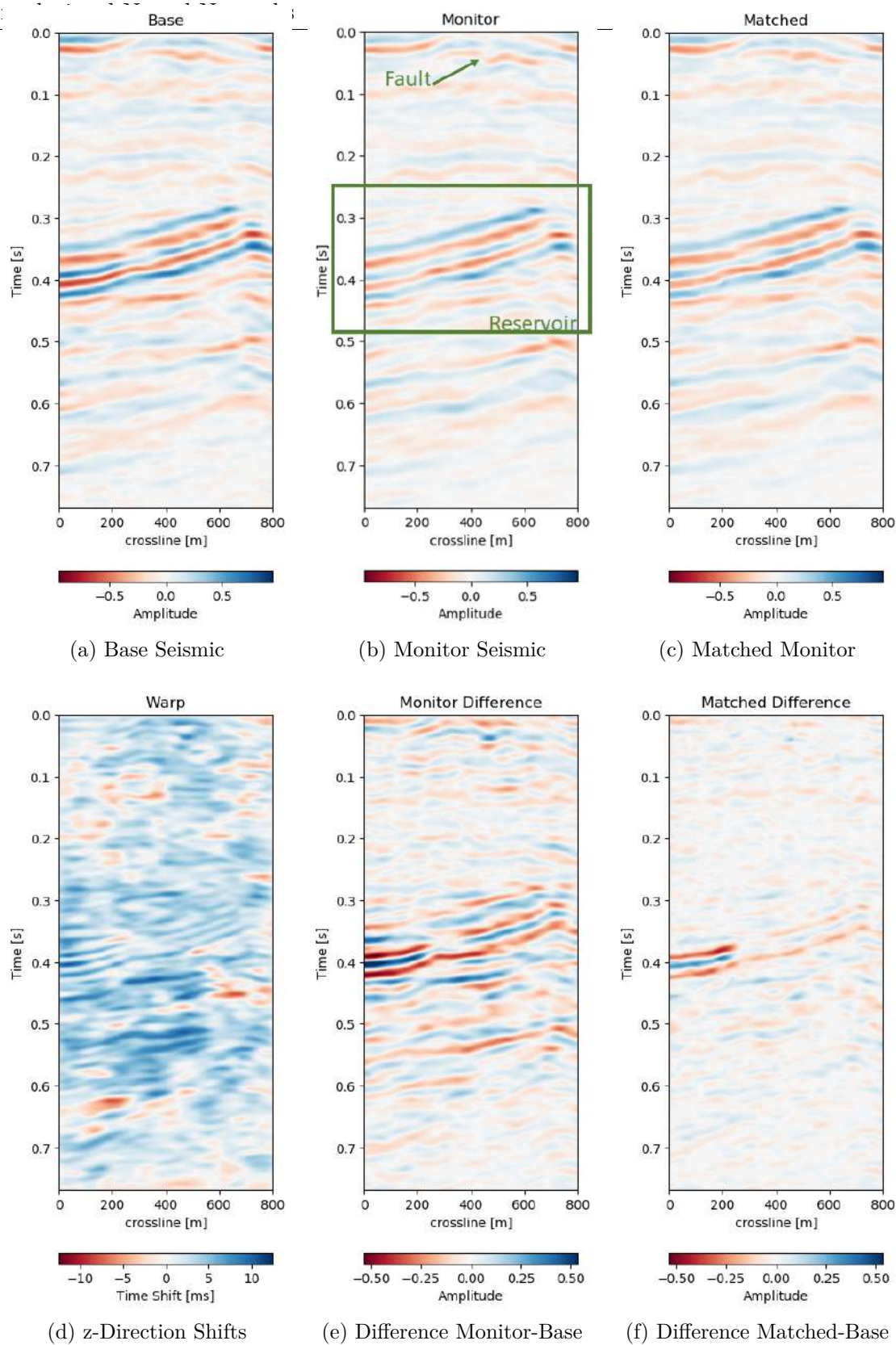


Figure 7.5: Warp results and change in difference on training recall of 1988 to 2005a data. Axes are relative to comply with confidentiality.

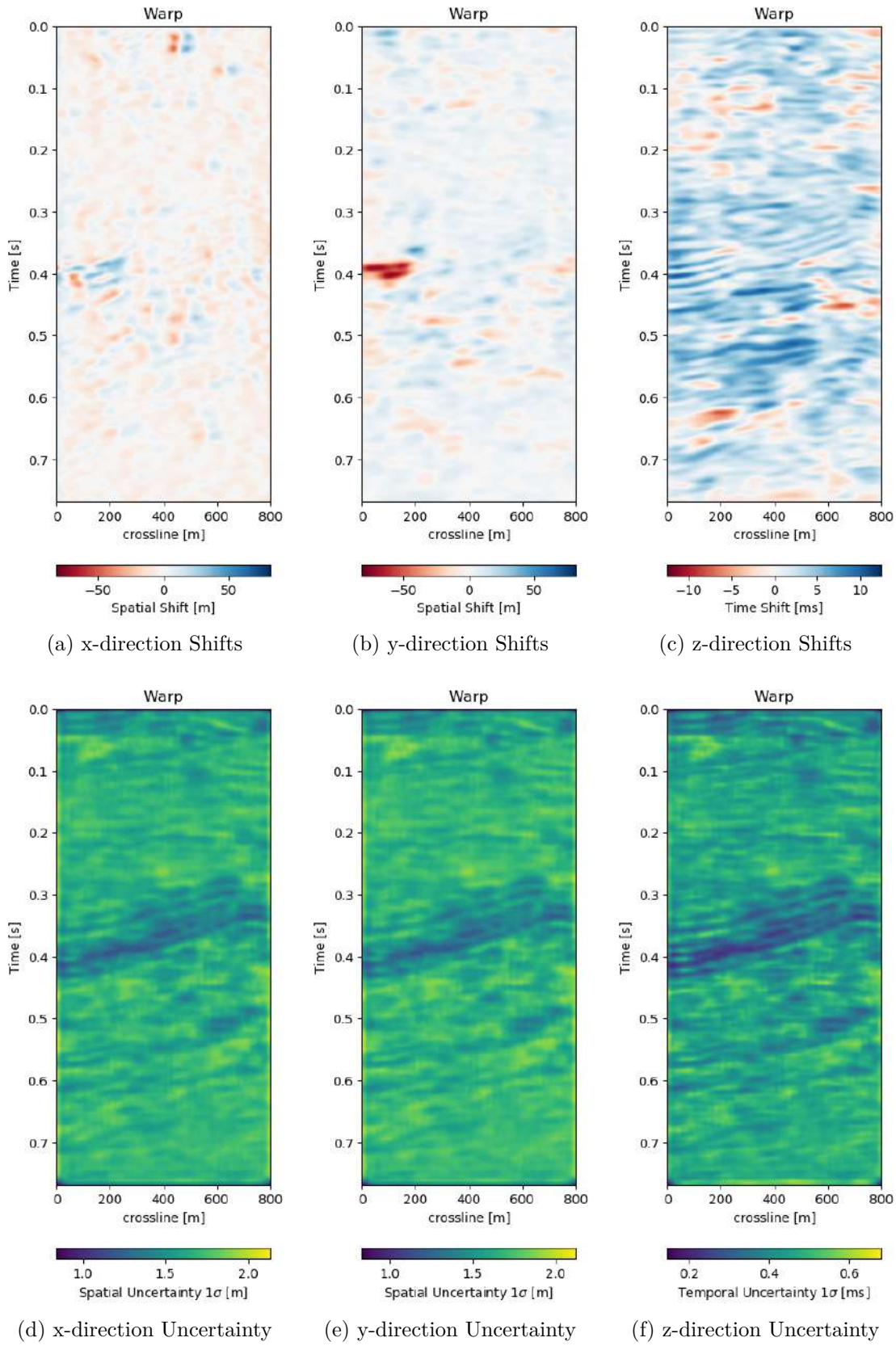


Figure 7.6: Warp fields (top) with uncertainties (bottom) that accompanies training recall in Figure 7.5

(x,y) direction fully displayed in Figure 7.6 including their respective uncertainties. The difference of the warped result in Figure 7.5(f) is calculated from the matched monitor in Figure 7.5(c) and the base in Figure 7.5(a).

It is apparent that the matched monitor significantly reduced noise by mis-aligned reflections. In Table 7.1 we present the numeric results. These were computed on the 3D cube for an accurate representation. We present the root mean square (RMS) and mean absolute error (MAE) and the according difference between Monitor and Matched Difference results. We present RMS and MAE to make the values comparable in magnitude as opposed the mean squared error (MSE). We present both values, because the RMS value is more sensitive to large values, while MAE scales the error linearly therefore not masking low amplitude mis-alignments. Both measurements show a reduction on the train data to 50% or below. The test on both the validation data on the same field and the test data on another field show a similar reduction, while the absolute error differs in a stable manner.

In Figure 7.6 we present the three dimensional warp field to accompany the results in Figure 7.5. Figure 7.6(a), 7.6(b), and 7.6(c) show the warp field in x, y, and z-direction. The z-direction is generally referred to as time shifts in 4D seismic. Figure 7.6(d), 7.6(e), and 7.6(f) contain the corresponding uncertainties in x, y, and z-direction obtained from the network.

Recall to Training Data In Figure 7.5 we evaluate the results of the self-supervised method on the training data itself. The main focus is on the main reflector in the center of the panels. The difference in Figure 7.5(e) shows that the packet of reflectors marked reservoir in the monitor is out of alignment, causing a large difference, which is corrected for in Figure 7.5(f). The topmost section in the panel of Figure 7.5(c) shows the alignment of a faulted segment, marked fault in the monitor, to an unfaulted segment in the base. The fault appearing is most likely due to vastly improved acquisition technology for the monitor.

The warp fields in Figure 7.6 are an integral part in QC-ing the validity of the results. Physically, we expect the strongest changes in the z-direction in Figure 7.6(c). The changes in Figure 7.6(a) and Figure 7.6(b) show mostly sub-sampling magnitude shifts, except for the x-direction shifts around the fault in the top-most panel present in the monitor in Figure 7.5(b). Figure 7.6(a) and Figure 7.6(b) show strong shifts at 0.4s on the left of the panel which corresponds to the strong amplitude changes in the base and monitor. On the one side these correspond to the strongest difference section, additionally these are geological hinges, which are under large geomechanical strain. However, these are very close to the sides of the warp, which may cause artifacts. Figure 7.6(d), Figure 7.6(e), and Figure 7.6(f) show the uncertainty of the network. These uncertainties are across the bank within the 10% range of the sampling rate ($\Delta t = 4$ ms, $\Delta x, y = 25$ m). The certainty within the bulk package in the center of the panels is the lowest in x-, y-, and z-direction. While being relatively lower in the problematic regions discussed before.

The warp field in Figure 7.6(d) contains some reflector shaped warp vectors around

Run	Monitor RMS	Matched RMS	Ratio %	Monitor MAE	Matched MAE	Ratio %
Baseline	0.1047	0.0718	68.6	0.0744	0.0512	68.7
Train	0.1047	0.0525	50.1	0.0744	0.0348	46.7
Test A	0.0381	0.0237	62.2	0.0291	0.0172	59.1
Test B	0.0583	0.0361	62.0	0.0451	0.0254	56.4

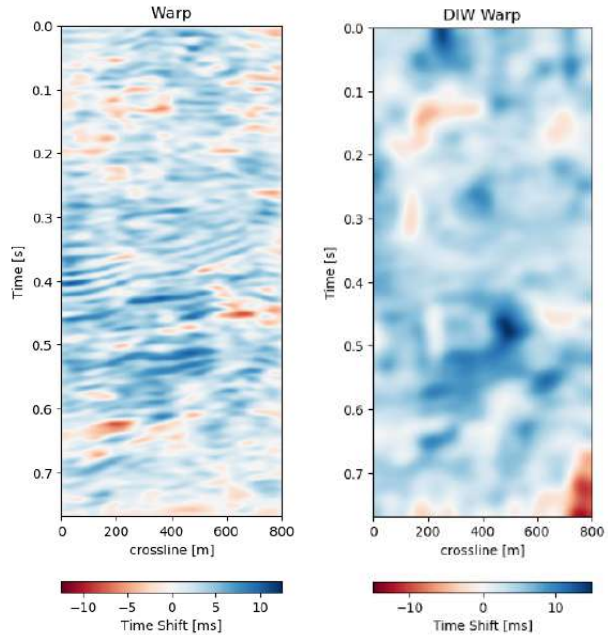
Table 7.1: Quantitative Evaluation of Results. RMS and MAE calculated against respective base data. Training recall, Test A - Same field, different acquisition, Test B - different field, different acquisition

0.4 s, which is due to the wavelet mismatch of the 1988 base to the 2005 monitor. The diffeomorphic nature of the network aligns the reflectors in the image, which causes some reflector artifacts in the z-direction maps.

Comparison to Baseline Method

We use the Dynamic Image Warping method (Hale, 2013c) to align the images in Figure 7.5. This method extends the Dynamic Time Warping method to 2D and provides a much improved result in 2D compared to standard cross-correlation and DTW methods. Inversion methods need pre-stack seismic data, which is not available. We chose this baseline to provide a fair comparison with the available data. Figure 7.7 shows the timeshifts or warp fields generated by the Voxelmorph network and by the DIW algorithm. The DIW algorithm shows a smoothed image. Overall, the Subfigure 7.7b shows the general trends of Subfigure 7.7a. The Voxelmorph algorithm is more detailed than the DIW image, however the general magnitude of the time shifts matches well in the correct areas.

Figure 7.8 shows the matched monitors from Voxelmorph and DIW. The matched monitors align quite well without any significant discrepancies. The matched difference shows that the Voxelmorph algorithm performs similarly to the baseline method, while removing more 4D noise from the image. It keeps the 4D signal intact, albeit slightly varying. The DIW algorithm seems to struggle to align the topmost part of the image, while Voxelmorph aligns these well, removing additional 4D noise. Table 7.1 confirms this quantitatively, where the overall



(a) Full-Scale Warp Field (b) DIW Timeshifts

Figure 7.7: Comparison of Voxelmorph warp field (left) and Dynamic Image Warping (right) warp fields.

RMSE and MAE are reduced proportionally.

Generalization of the Network While the performance of the method on a data set by itself is good, obtaining a trained model that can be applied on other similar data sets is essential even for self-supervised methods. We test the network on two test sets, Test A is conducted on the same geology with unseen data from a different acquisition, while Test B is on a different field and a different acquisition. The network was trained on a single acquisition relation (2005a - 1988). In Figure 7.10 we present the crossline data from the same field the network was trained on. The data sets was however acquired at a different calendar times (2005b - 2012), with different acquisition parameters. It follows that although the geology and therefore the reflection geometry is similar, the wavelet and hence the seismic response are vastly different. This becomes apparent when comparing the base Figure 7.10a to Figure 7.5(b), which were acquired in the same year.

Test A evaluates the network performance on unseen data in the same field (Train: 1988-2005a, Test A: 2005b - 2012). The quantitative results in Table 7.1 for Test A generally show lower absolute errors compared to the training results in Section 7.4.4.2. The reduction of the overall amplitudes in the difference maps is reduce by 40%. The unaligned monitor difference in Figure 7.10(e) shows a strong coherent difference around below the main packet of reflectors around 0.3 s to 0.4 s. This would suggest a velocity draw-down in this packet. While the top half of the unaligned difference contains some misalignment, we would expect the warp field to display a shift around 0.35 s, which can be observed in Figure 7.10(d). The aligned difference in Figure 7.10(f) contains less coherent differences. The difference does still show some overall noise in the maps. This could be improved upon by a more diverse training set. The higher resolution data from 2005 and 2012 possibly has an influence on the result too. Regardless, we can see some persisting amplitude difference around 0.4 s which appears to be signal as opposed to some misalignment noise above. The warp fields in Figure 7.11 show relatively smooth warp fields in x- and y-direction. The warp field in Figure 7.11(f) shows overall good coherence, including the change around 0.4 s we would expect. The uncertainty values are in sub-sampling range, with the strongest certainty within the strong reflector packet at 0.35 s.

Test B evaluates the network performance on a different field, with different geology, with unrelated acquisition geometry and equipment and at different times. The test shows a very similar reduction of overall errors in Table 7.1. The RMS is reduced by 38% and the MAE is reduced more slightly more in comparison to Test A. In Figure 7.12 we present the seismic panels to accompany Test B. The data in Figure 7.12(a) and Figure 7.12(b) is well resolved and shows good coherence. However, the unaligned difference in Figure 7.12(e) shows very strong variations in the difference maps. Figure 7.12(f) reduces these errors significantly, bringing out coherent differences in the main reflector at 0.27 s. We can see strong chaotic differences in Figure 7.12(e), due to the faulted nature of the geology. The network aligns these faulted blocks relatively well, however, some artifacts persist. This is consistent with the warp fields in Figure 7.13. The x- and y-

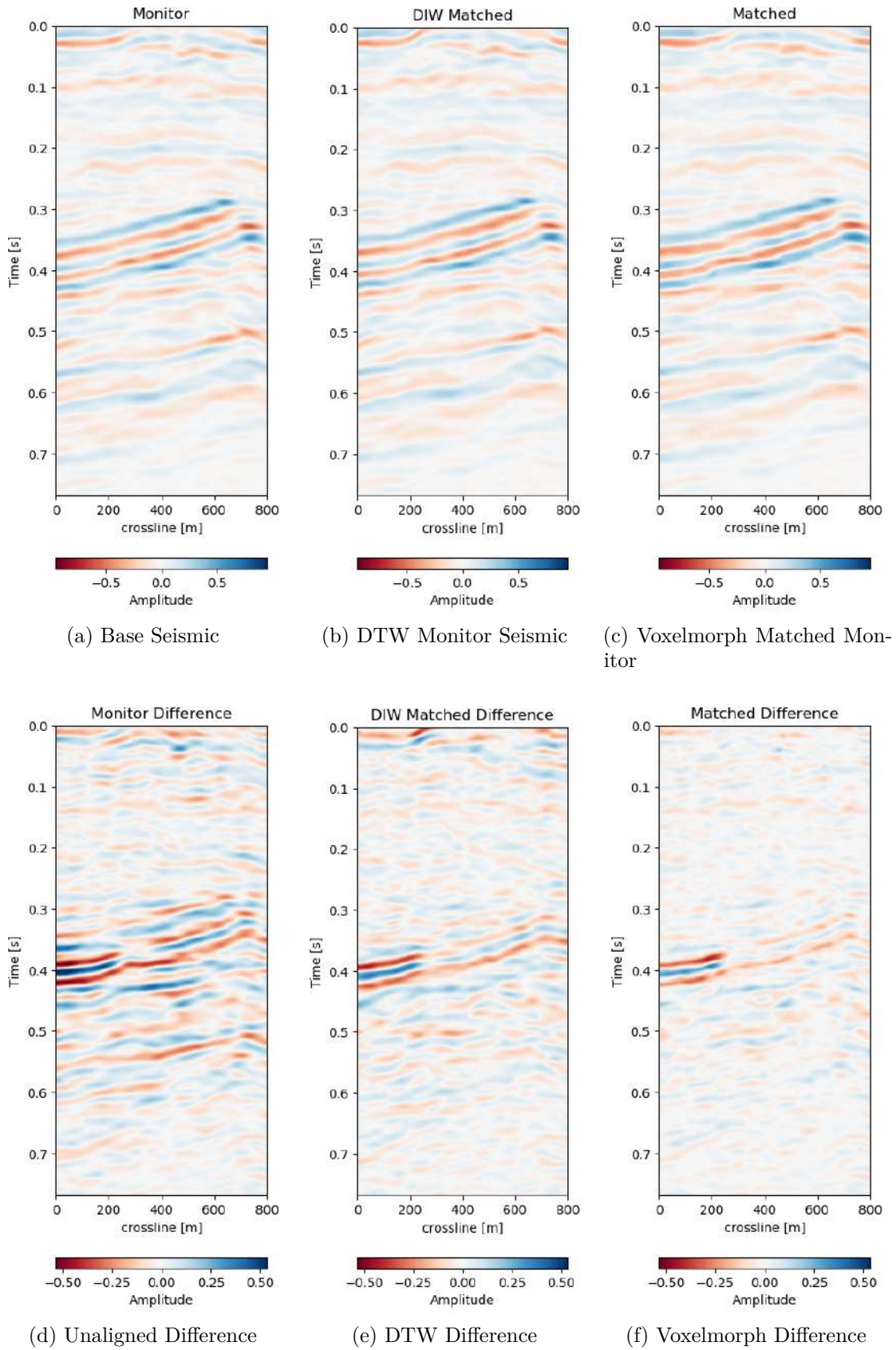


Figure 7.8: Results of Voxelmorph warping compared to baseline dynamic image warping algorithm. Top row shows the aligned monitor, bottom row shows the difference to the base volume.

direction in Figure 7.13(d) and Figure 7.13(e) respectively show overall smooth changes, around faults, these changes are stronger. The z-direction changes are consistent with the Training validation and Test A, where the changes are overall stronger. This is also consistent with our geological intuition.

Subsampled Flow The original Voxel-morph implementation uses a subsampled warp field. The authors claim two benefits, namely a smoother warp velocity field and reduced computational cost. The aforementioned results were obtained using our full-scale network. In Figure 7.9 we present the full scale and upsampled results on the training set. The matched difference in Figure 7.9b contains more overall noise compared to Figure 7.9a. This is congruent with the warp fields in the figure. The upsampled z-direction warp field in Figure 7.9d seems to have some aliasing on the diagonal reflector around 0.4 s. This explains some of the artifacts in the difference in Figure 7.9b. The overall warp velocity in Figure 7.9d is smoother compared to the full-scale field. However, the general structure of coherent negative and positive areas matches in both warp fields, while the details differ. The main persistent difference of the reflector packet at 0.4 s seems similar, nevertheless, the differences further up slope to the right are smoother in the full scale network result and have stronger residual amplitudes in the upsampled network. Overall, the full-scale network results are better for seismic data at a slightly increased computational cost. The subsampled field introduced artifacts in our observations.

7.4.5 Conclusion

We introduce a deep learning based self-supervised 4D seismic warping method. Currently, time shifts are most commonly

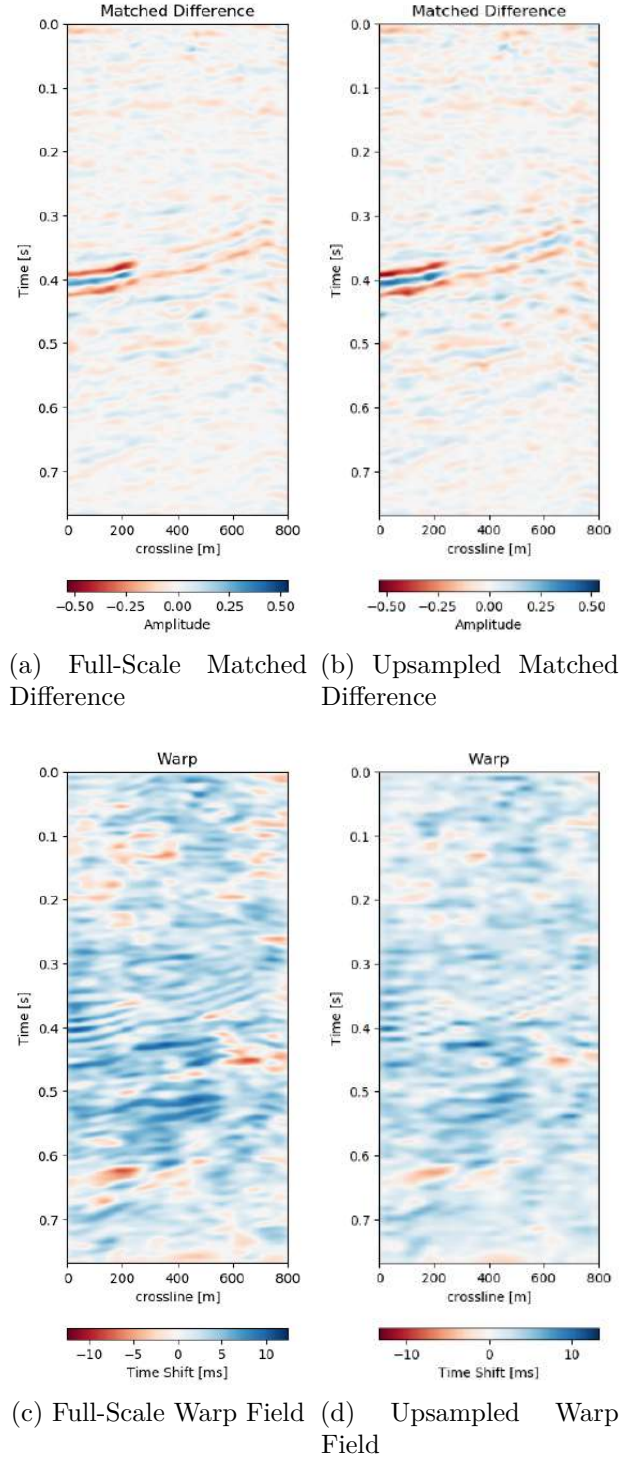


Figure 7.9: Comparison of matched differences (top) and z-direction warp field (bottom) of full-scale neural architecture (left) and subsampled neural architecture (right).

estimated in 1D due to computational constraints. We explore 3D time-shift estimation as a viable alternative, which decouples imaging and acquisition effects, geomechanical movement and changes in physical properties like velocity and porosity from confounding into a single dimension. Existing 3D methods are computationally expensive, where this learnt model can generalize to unseen data without re-training, with calculation times within minutes on consumer hardware. Moreover, this method supplies invertible, reproducible, dense 3D alignment while providing warp fields with uncertainty measures, while leveraging recent advancements in neural networks and deep learning.

We evaluate our network on the training data and two different independent test sets. We do not expect the aligned difference to be exactly zero, due to actual physical changes in the imaged subsurface. Although the network is unsupervised, a transfer to unseen data is desirable and despite some increase in the overall error possible. The warping on the training data is very good and the warp fields are coherent and reflect the physical reality one would expect. The transfer to unseen data works well, although the misalignment error increases. The decrease in both RMS and MAE is consistent across test sets.

Furthermore, we implement a variational scheme which provides uncertainty measures for the time shifts. On the data presented, we obtain subsample scale uncertainties across all directions. The main assumption of the network is a diffeomorphic deformation, which is topology preserving. We show that the network handles faults well in both training recall and test data, that in theory could violate the diffeomorphic assumption.

We go on to compare a full-scale network to an upsampled network. The full-scale network yields better results and is preferable on seismic data in comparison to the upsampled network presented in the original medical Voxelmorph.

We do expect the network to improve upon training on a more diverse variety of data sets and seismic responses. While the initial training is time-consuming (25 h on a Nvidia Titan X with Pascal chipset), inference is near instantaneous. Moreover, transfer of the trained network to a new data set is possible without training, while accepting some error. Alternatively fine-tuning to new data is possible within few epochs (<1 h).

Acknowledgment

The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding program. We thank DTU Compute for access to the GPU Cluster. We thank Total E&P Denmark for permission to use the data and publish examples.

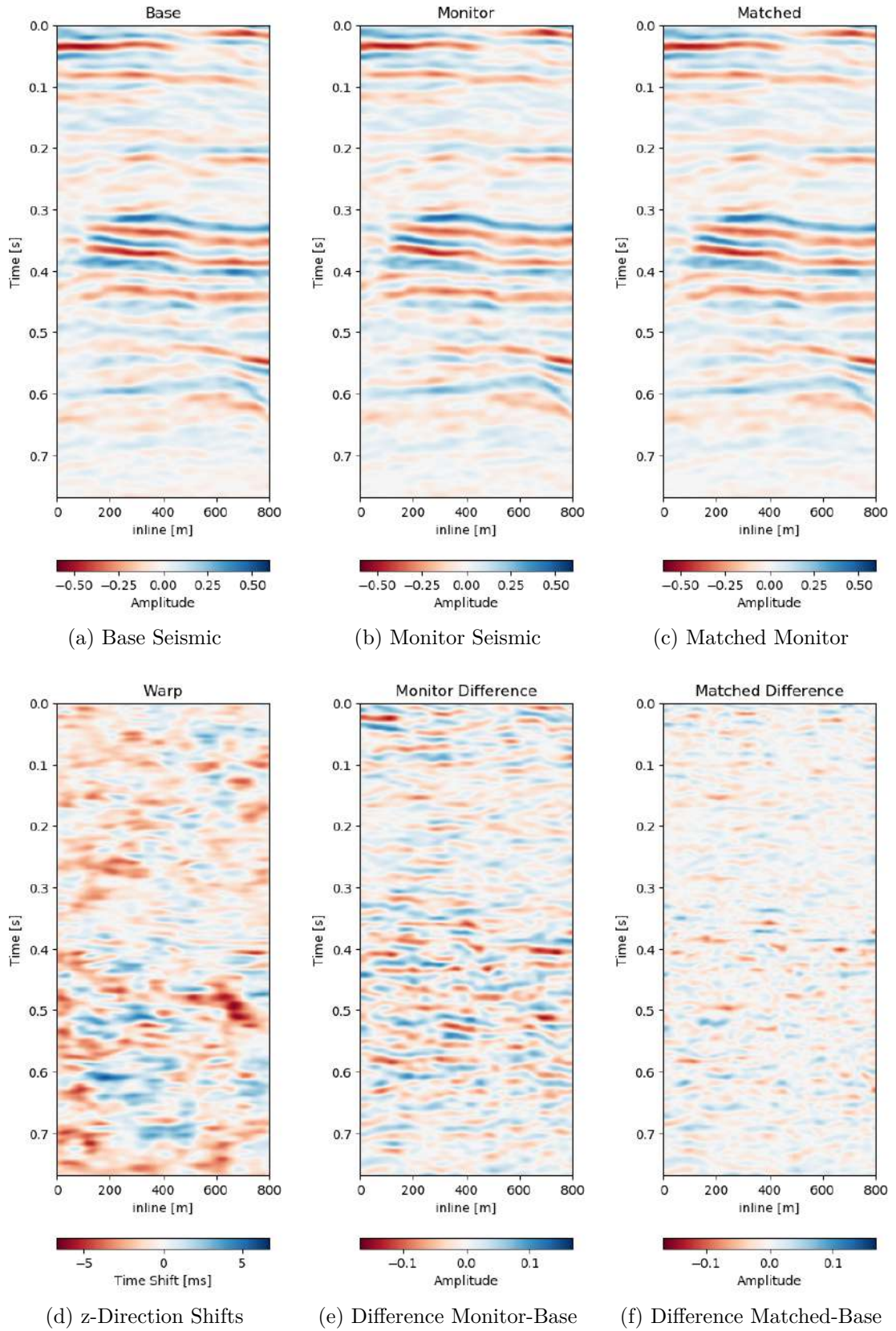


Figure 7.10: Matched difference and warp field for generalization of network to same field with different data (2005b and 2012).

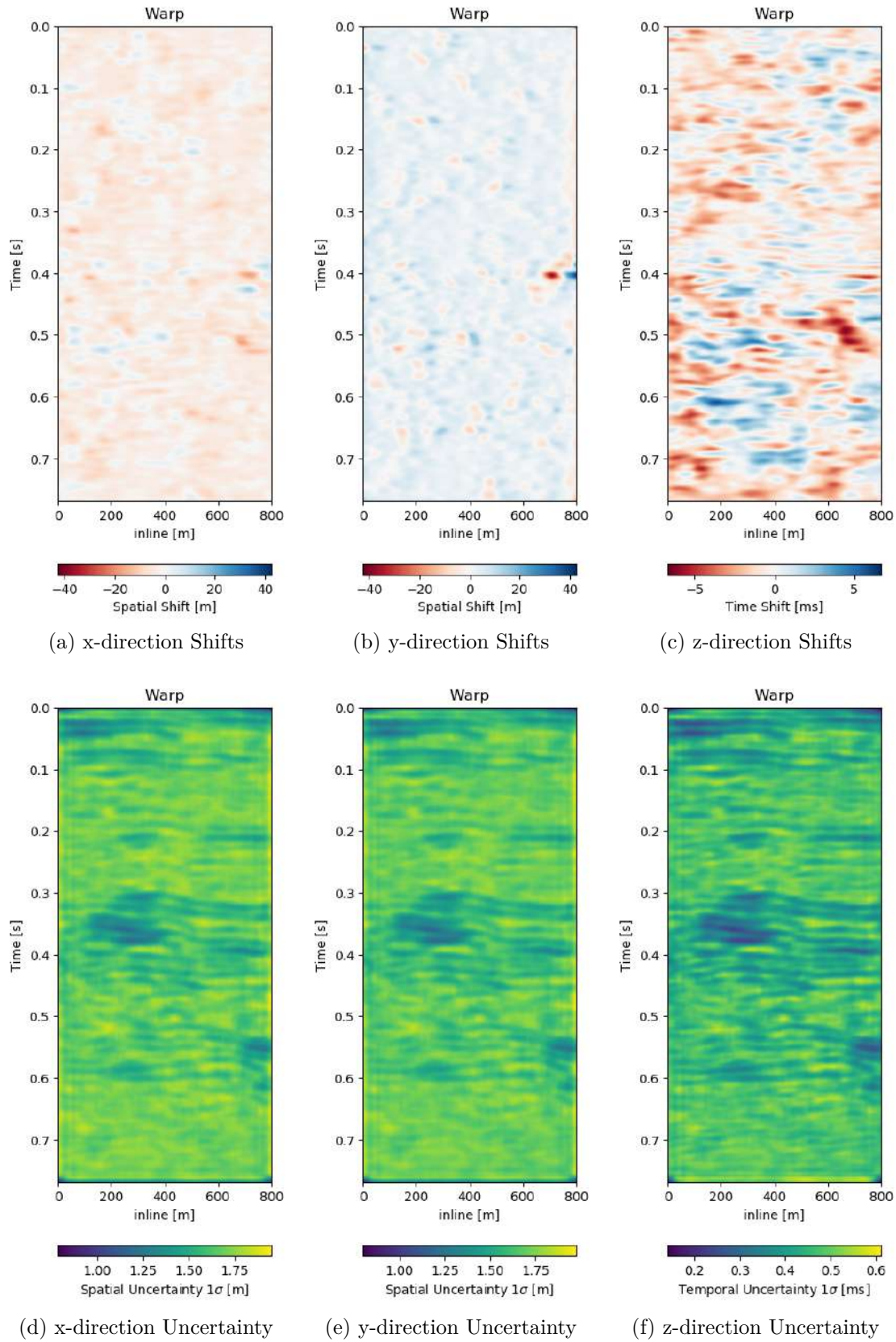


Figure 7.11: Warp fields (top) with uncertainties (bottom) that accompanies same field generalization in Figure 7.10

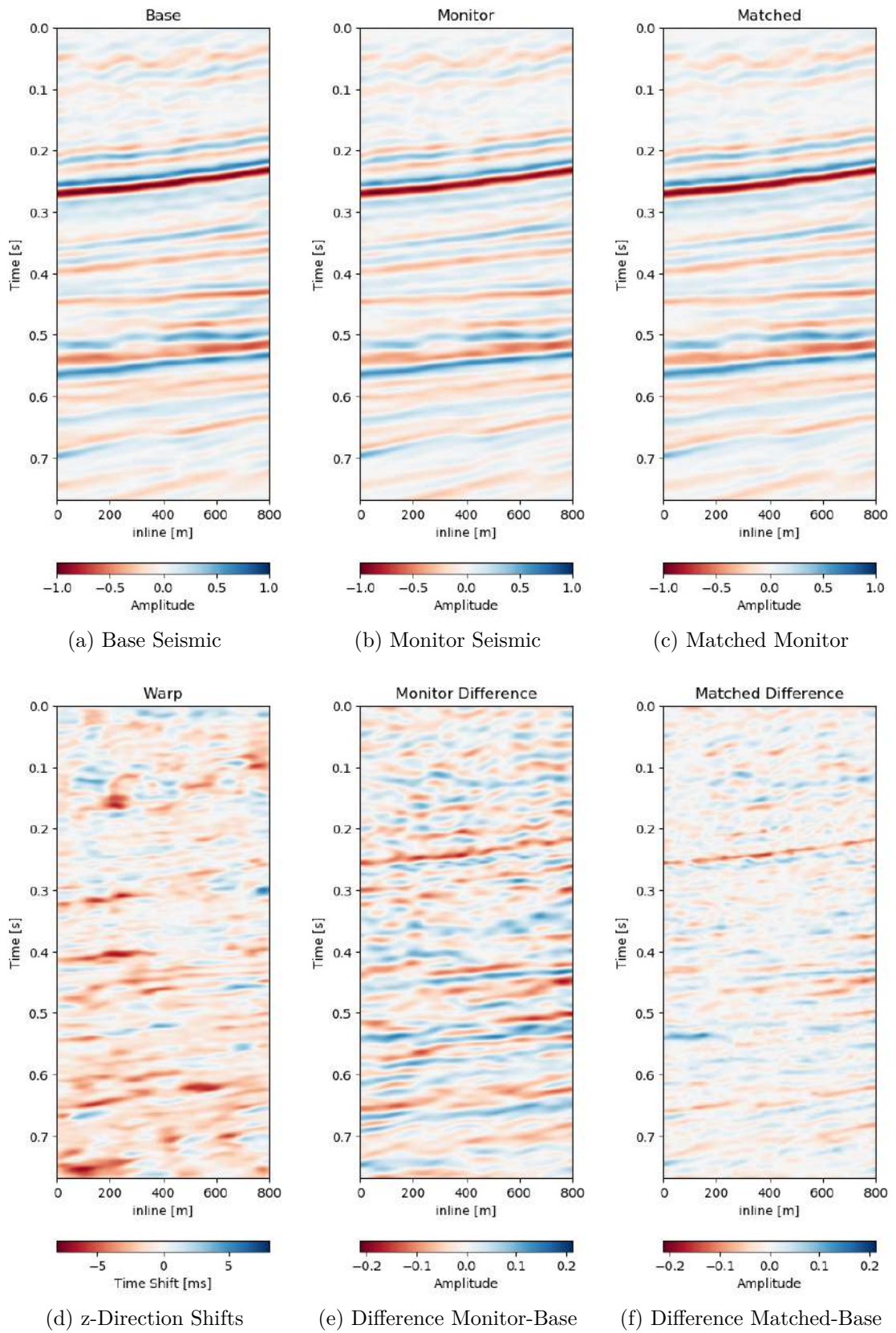


Figure 7.12: Matched difference and warp field for generalization of network to a different field (1993 and 2005).

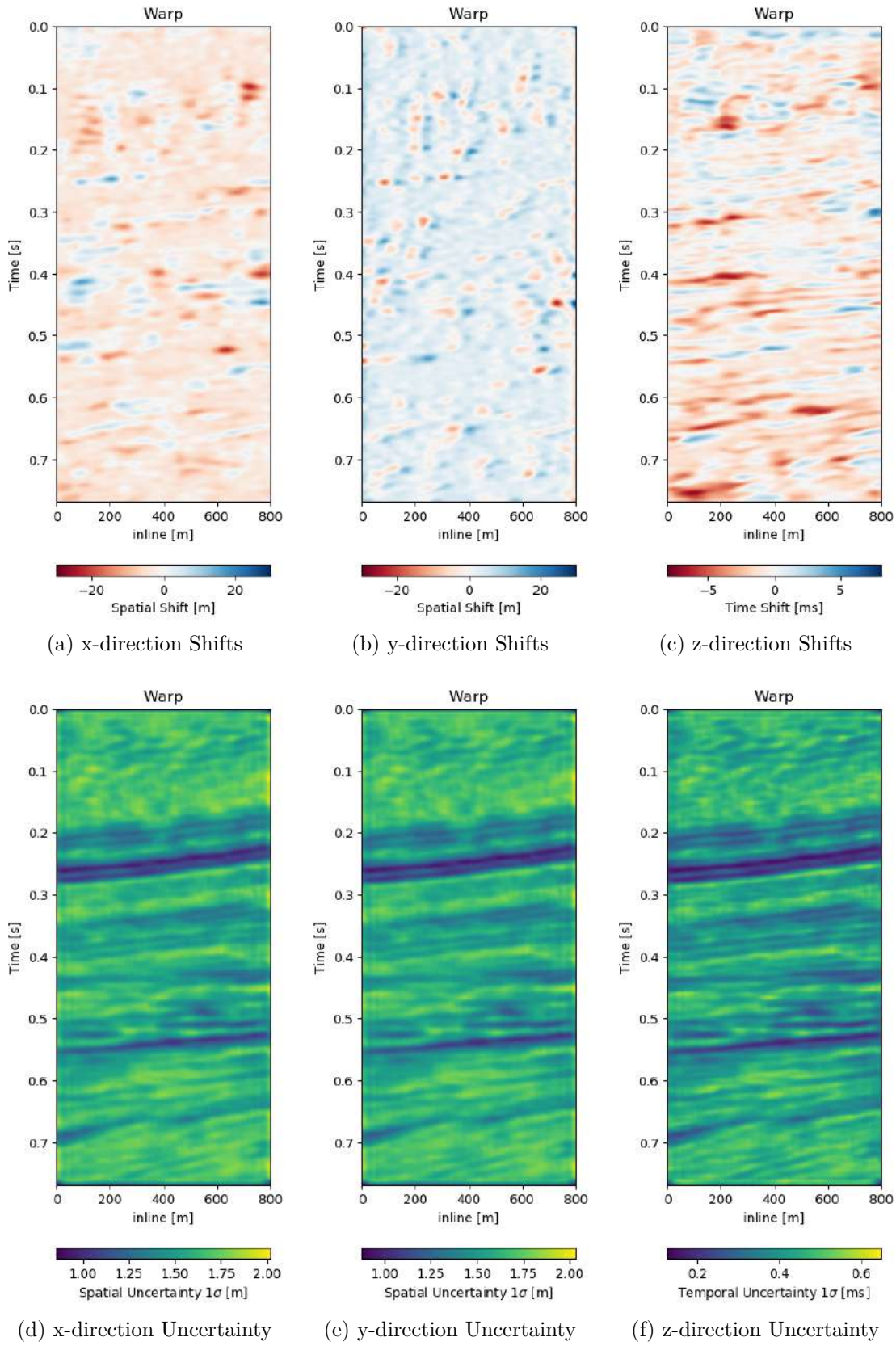


Figure 7.13: Warp fields (top) with uncertainties (bottom) that accompanies generalization to different field in Figure 7.12

7.5 Contributions of This Study

In the paper, we present the modified self-supervised Neural Network system and test the results on the training data itself and two generalization test sets. The first test set is on the same field but recorded at different times to the training set, ensuring similar underlying geology, whereas, the second test set is taken from an adjacent field, recorded at different times, with different geology, testing the full transfer of the trained network. We go on to test the original Voxelmorph architecture, which uses upsampled velocity fields and evaluate the results against our modified architecture, which uses the full flow field. Overall, this technique introduces a generalizable Deep Learning approach to extract 3D time-shifts with uncertainty measures from raw stacked 4D seismic data.

The Voxelmorph network performs very well on seismic data with patch-based seismic data. It is essential to implement the full-scale architecture to obtain reliable 3D time-shifts on 4D seismic data. The network exhibits stable error on the unseen data on the same field and differing test field, which indicates that the networks learn relevant generalizable information. Despite being a 3D method, the primary shifts are estimated in the z-direction, which is consistent with the expectation we have for seismic data. The diffeomorphic assumption performs well on the seismic data even on faulted data, preserving the topology. Additionally, unsupervised training reduces further implicit assumptions from extracted time-shifts or synthetic models. The model would improve from data augmentation methods and including multiple fields in the training data.

CHAPTER 8

Conclusion of this Thesis

This thesis contributes Machine Learning applications in geoscience with a focus on field data applications in 4D seismic, Backscatter Scanning-Electron Microscopy, and Automatic Seismic Interpretation. Additionally, the introduction contains a published review of the history of Machine Learning in geoscience with insights into the recent interest around the topic.

The book chapter in Section 2.2 discusses the historic development of Machine Learning in geoscience. It highlights key papers and developments through the decades, relating the developments to larger developments in the field of Artificial Intelligence and Machine Learning. In the book key algorithms are detailed including Support Vector Machine, Random Forest, Gaussian Process and the development from kriging, as well as, key Neural Network developments and Deep Learning architectures that enable modern applications throughout many scientific disciplines including geoscience as a whole.

The exploration of Backscatter Scanning-Electron Microscopy (BSEM) data in Chapter 3 introduced a novel unsupervised method to extract chalk grain boundaries from image data and shows the improvement of subsequent morphological filtering (Dramsch et al., 2018a). These methods reduce labour-intensive manual tasks, introducing varying degrees of automation in geoscience workflows. Following the extraction of the boundaries in the BSEM images, computational granulometry can be performed. This includes statistics about grain size and circularity of the grains and the orientation of grains. Commonly this data had to be obtained by manual measurement of every grain. The unsupervised nature of this application means that no training data is necessary; in turn, it can be used to obtain high-quality training data for subsequent supervised machine learning tasks.

The research in Chapter 4 showed that transfer learning could alleviate the necessity for large amounts of labelled data, by re-using a Neural Network trained on natural images. This study showed that Neural Networks can be transferred to seismic data and outperform smaller networks trained from scratch. The smaller network size was necessary to avoid overfitting. The source code for this research was made available and has been of use to multiple researchers (Dramsch, 2018a). This has broad applications in industry and research settings performing Automatic Seismic Interpretation (ASI). The limited availability of labelled data and wide availability of pre-trained network architectures makes this a viable option to obtain improved results and more robust models. Moreover, this insight is applicable to pre-training geoscientific Neural Networks.

Dramsch et al. (2019f) shows that explicitly using phase information as input in a complex-valued neural network can stabilize the reconstruction of compressed seismic

data. The smaller complex-valued network in Chapter 5 outperforms larger real-valued networks; however, a very large real-valued network that does not compress the seismic data can implicitly learn partial phase information. The paper touches on deficits of current metrics applied to geoscience and exposes a periodic dimming effect of frequencies from neural networks that should be further investigated, particularly in the context of aliasing. This paper led to the creation of the open-source software package **keras complex** to enable complex-valued deep learning in **Tensorflow** (Manual in E.5). Considering the modularity of neural networks, this insight can be transferred to other deep learning tasks on physical data like seismic data. Additionally, this research could lead to further investigation of including known physical information in neural networks not limited to explicitly using the phase information as input.

Chapter 6 introduces a novel method to perform pressure-saturation inversion on amplitude difference maps (Dramsch et al., 2019d). This work incorporates basic physical relationships directly as features into the neural network architecture, which was shown to stabilize the training result. Moreover, this work shows the possibility of training Deep Neural Networks on simulation data and subsequently transferring the network to field data. This particularly was enabled by applying Gaussian noise within the network. The Deep Neural Network results were compared to results from the Bayesian inversion showing a promising application of Deep Neural Networks in 4D Quantitative Interpretation (Dramsch et al., 2019d). While this work has attracted interest in a sponsors meeting and the workshop presentations (Dramsch et al., 2019d; Dramsch et al., 2019e), further investigation into model explainability and lower complexity baseline models is necessary (Côte et al., 2020; Corte et al., 2020).

In Chapter 7 a novel method for time-shift extraction is presented. This method combines recent advancements in diffeomorphic mapping, Deep Learning and unsupervised learning to introduce a 3D time shift extraction method including uncertainty values, where 1D extraction is the standard (Dramsch et al., 2019b). The method is shown to work on 3D seismic post-stack data with strongly differing acquisition parameters, without supplying any time shift information. After applying the method, the 3D seismic volumes are well aligned, with the diffeomorphic constraint performing well on seismic data. This work tests the trained network on two other 3D seismic volume pairs to test the generalization of the Convolutional Neural Network after training. The two test sets show that the trained model on a single 3D seismic volume pair transfers well to the same field with different acquisition parameters and even a different field with a vastly different geological setting.

Overall, this thesis shows Deep Learning applications in seismic geophysics and resulted in multiple workshop, conference, journal papers, and a book chapter, including reproducible Python code for all publications. The publications, developed through interdepartmental and international collaboration, have been disseminated at international workshops and conferences. Two novel methods for 4D seismic analysis were introduced and compared to conventional methods. Moreover, transfer learning as a viable application in Automatic Seismic Interpretation was shown and has found wide application. The Python code in this thesis has been open-sourced for all published papers for reproducibility including the open-source package "keras complex".

APPENDIX A

ImageNet Results

Table A.1: ImageNet results of different neural network architectures (Partial resource from Papers With Code)

Name	Citation	Top-1 [%]	Top-5 [%]	Param [M]
AlexNet	Krizhevsky et al. (2012b)	63,3	84,6	60
VGG-16	Simonyan et al. (2014a)	74,4	91,9	138
VGG-19	Simonyan et al. (2014a)	74,5	92,0	144
AmoebaNet-B	Real et al. (2019)	82,3	96,1	84
AmoebaNet-C	Real et al. (2019)	83,1	96,3	155,3
DenseNet-201	Huang et al. (2017)	78,5	94,4	20
EfficientNet-B1	Tan et al. (2019b)	78,8	94,4	7,8
EfficientNet-B2	Tan et al. (2019b)	79,8	94,9	9,2
EfficientNet-B3	Tan et al. (2019b)	81,1	95,5	12
EfficientNet-B4	Tan et al. (2019b)	82,6	96,3	19
EfficientNet-B5	Tan et al. (2019b)	83,3	96,7	30
EfficientNet-B6	Tan et al. (2019b)	84,0	96,9	43
EfficientNet-B7	Tan et al. (2019b)	85,0	97,2	66
Inception V1	Szegedy et al. (2015)	69,8	89,9	5
Inception V2	Ioffe et al. (2015)	74,8	92,2	11,2
Inception V3	Szegedy et al. (2016)	78,8	94,4	23,8
InceptResNet V2	Szegedy et al. (2017)	80,1	95,1	55,8
MixNet-S	Tan et al. (2019c)	75,8	92,8	4,1
MixNet-M	Tan et al. (2019c)	77,0	93,3	5
MixNet-L	Tan et al. (2019c)	78,9	94,2	7,3
NasNet-A6	Zoph et al. (2018)	82,7	96,2	89
MNasNet-A1	Tan et al. (2019a)	76,7	93,3	5,2
MNasNet-A3	Tan et al. (2019a)	75,2	92,5	3,9
FixPNasNet-5	Touvron et al. (2019)	83,7	96,8	86,1
ResNet-50-D	He et al. (2019)	77,1	93,5	25
FixResNet-50	Touvron et al. (2019)	79,1	94,6	25,6
ResNet-101	He et al. (2016)	78,2	93,9	40
ResNeXt-101	Xie et al. (2017)	80,9	95,6	83,6
Oct-ResNet-152	Chen et al. (2019)	82,9	96,3	67

APPENDIX B

Journal Papers

B.1 An Integrated Approach to Fracture Characterization of the Kraka Field

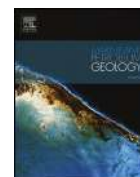
Abstract: Oil and gas production of tight chalk reservoirs frequently rely on the presence of natural fractures, which increases the effective permeability of the reservoirs. Knowledge of these fracture systems can therefore be used strategically in well planning as well as in IOR and EOR efforts. Here we present an integrated workflow for fracture characterization in chalk, developed in the Kraka Field, located in the Danish sector of the North Sea. The workflow is based on data from borehole images, cores and seismic. By introducing two ant-tracked attribute volumes, which display structural trends below the resolution of amplitude seismic, we are able to correlate features at different scales. In Kraka, this approach has revealed that the fracture pattern is more complex than previously suggested. We propose that fracture generation and propagation in the field is in part controlled by the regional maximum horizontal stress and in part formed in response to salt movements.

T. M. Aabø, J. S. Dramsch, C. L. Würtzen, S. Seyum, F. Amour, M. Welch, and M. Lüthje (2020). “An integrated workflow for fracture characterization in chalk reservoirs, applied to the Kraka Field”. In: *Marine and Petroleum Geology* 112. Published, Appendix B. ISSN: 0264-8172. DOI: <https://doi.org/10.1016/j.marpetgeo.2019.104065>. URL: <http://www.sciencedirect.com/science/article/pii/S026481721930501X>



Contents lists available at ScienceDirect

Marine and Petroleum Geology

journal homepage: www.elsevier.com/locate/marpetgeo

Research paper

An integrated workflow for fracture characterization in chalk reservoirs, applied to the Kraka Field



Tala Maria Aabø*, Jesper Søren Dramsch, Camilla Louise Würtzen, Solomon Seyum, Michael Welch

The Danish Hydrocarbon Research and Technology Centre (DHRTC), Technical University of Denmark, Elektrovej 375, 2800, Kgs. Lyngby, Denmark

ARTICLE INFO

Keywords:

Reservoir characterization
Fractures
Structural correlation
Borehole images
Core analysis
Seismic attribute volumes
Ant-tracking
Chalk

ABSTRACT

Oil and gas production of tight chalk reservoirs frequently rely on the presence of natural fractures, which increases the effective permeability of the reservoirs. Fracture characterization is therefore imperative in optimizing production schemes and obtaining economically viable recovery factors. Subsurface fracture characterization is often deemed challenging as the available data is typically of varying age and quality, and represents different scales. We have developed an integrated workflow for fracture characterization in chalk to address these challenges. The workflow is based on data from borehole images, cores and seismic. These data are typically available for most chalk (and hydrocarbon) fields. The interpreted borehole image dataset contains over 17 000 manual dip picks, ensuring a statistically viable base. A total of 150 m of core is available from 3 wells. The applied 3D seismic cube covers an 8×5 km hydrocarbon chalk field in the Danish North Sea.

In this workflow, the scale-gap between the data sets is bridged by the introduction of two ant-tracked attribute volumes, which display structural trends below the resolution of amplitude seismic. Further insight into the intricacy of subsurface fracture systems is obtained from fracture density logs, which provide an opportunity to study spatial distribution of fractures as well as a qualitative measure of fracture clustering. Cumulative density distribution plots and calculation of the variation coefficient of fracture spacing provide a more quantitative analysis of the fracture distribution.

The workflow, presented here in a step-by-step manner, is a general approach applied to data from the Kraka Field of the Danish North Sea. In the Kraka Field, the usage of this integrated approach shows that the fracture pattern in this region is more complex than previously suggested; probably controlled by the regional maximum horizontal stress and salt movements.

1. Introduction

Chalks typically represent high porosity - low permeability reservoirs, in which natural fractures are essential for hydrocarbon production (Koestler and Reksten, 1992). Knowledge of these fracture systems is often used strategically in well planning and in IOR and EOR efforts. Descriptions and models of natural fracture systems allow for simulation of flow and flow patterns in reservoirs, which in turn helps in understanding the quality and amount of hydrocarbon reserves. Natural fractures define the communication in reservoirs, which is the determining factor for well-placing decisions and setup of production schemes for IOR technologies (e.g. water flooding) and EOR technologies (e.g. smart water/chemical flooding).

Accurate predictions of natural fracture systems require an understanding of the controls on fracture orientations and distributions in an

area (Fernø, 2012). We have developed a workflow to correlate structural features at different scales, based on borehole image-, core- and seismic data. The applied seismic data includes an amplitude volume and two ant-tracked volumes. High resolution lineations mapped on the two ant-tracked cubes (generated through a variance cube and through RGB-image processing of the 3D seismic volume, respectively) enables detection of smaller-scale lineations below the resolution of conventional seismic, thus bridging the scaling-gap between well and seismic data. Spatial distributions and fracture clustering is considered using fracture density data.

The suggested workflow has been applied to the Kraka Field. The Kraka Field, an asymmetric anticlinal structure located in the Danish Central Graben (Fig. 1), was chosen as a test-case as it is a relatively simple structure with a manageable amount of data. The applied data was provided by Maersk for the purposes of this research project. Kraka

* Corresponding author.

E-mail address: talama@dtu.dk (T.M. Aabø).<https://doi.org/10.1016/j.marpetgeo.2019.104065>

Received 9 November 2018; Received in revised form 23 September 2019; Accepted 27 September 2019

Available online 04 October 2019

0264-8172/ © 2019 Elsevier Ltd. All rights reserved.

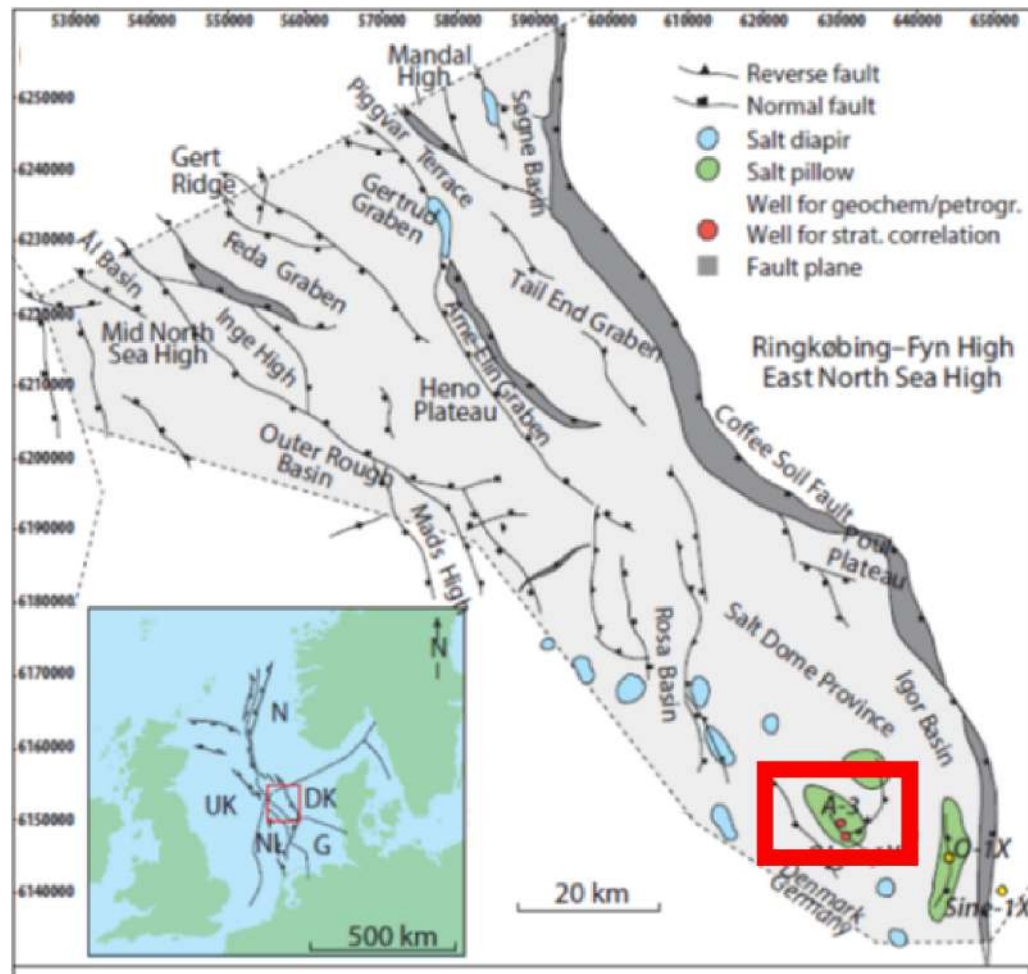


Fig. 1. Location of the Kraka Field, indicated by the red square, on a structural elements map of the Danish Central Graben (modified and edited from Møller and Rasmussen (2003)). (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

is produced mainly from the Danian Ekofisk Formation but also from the Maastrichtian Tor Formation, both of which are naturally fractured chalk reservoirs (Jorgensen and Andersen, 1991).

Combining the three aforementioned data types in an integrated workflow allows for the extrapolation of fractures away from the borehole, increasing our holistic understanding of the natural fracture distributions, in this case of the Kraka Field. Using this integrated approach we are able to evaluate orientations of lineations from well-to seismic scale, allowing for structural modelling based on a geological conceptual model.

Our results will serve as inputs into a discrete fracture network model (DFN) founded on geomechanical principles for fracture propagation, which will improve future well planning and EOR activities in the area.

2. Geological setting

Prior to applying the workflow, the geological setting of the Kraka Field was considered.

The Kraka anticline was induced through halokinesis. Initiation started during the Triassic and continued to move during the remaining Mesozoic (Rank-Friend and Elders, 2004). The structure stretches more than 8 km along its long axis and approximately 5 km along its short

axis (Rasmussen et al., 2005). The lithology in Kraka varies between pure chalk and marly chalk, with varying amounts of chert layers and nodules. Cores from the field show localized staining which is cyclic and often associated with the chert. On well-scale, three main structural features have been identified in Kraka. Large, open fractures with slickensides are abundantly observed throughout the core data. These fractures commonly terminate in clay rich layers. Smaller, chert-associated fractures occur frequently within the Ekofisk Formation and on occasion within the Tor Formation. Stylolite-associated fractures, which is predominantly observed in the Tor formation, are perpendicular to the pressure solution seams and are typically < 25 mm high.

Both reservoir units are characterized as tight. Porosities are in the range of 20–35% and permeabilities range from 3mD to < 1 mD (Klinkby et al., 2005). Effective matrix permeabilities in the field are significantly enhanced due to the presence of these natural fractures. The tectonic fractures (shear and extensional) are the main permeability enhancers. Smaller fractures associated with cherts and stylolites may however be important for local permeability enhancement (Jorgensen and Andersen, 1991).

It has previously been concluded that tectonic fracturing in the Kraka chalk may be understood as simple dome related fractures, possibly dominated by a tangential system (Jorgensen and Andersen, 1991). It was also suggested that Kraka fractures occur in swarms: a

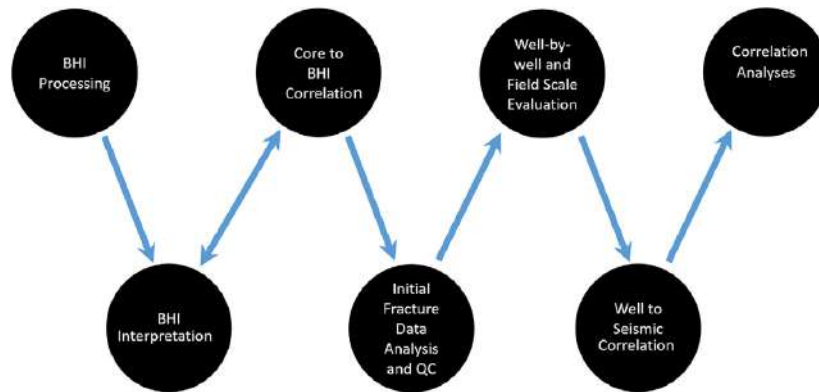


Fig. 2. Integrated workflow.

production logging tool from a horizontal wellbore indicated that only 4 of the 17 perforation intervals in that well contributed 94 % of the fluid production (Jorgensen and Andersen, 1991). The potential existence of fracture swarms is further addressed through our workflow. The results of our data analyses indicate that the Kraka fracture pattern is more complex than previously suggested with a primary set of fractures controlled by the regional maximum horizontal stress as well as a secondary set of dome-related fractures, associated with halokinesis.

3. Workflow: data availability, consistency and correlation

The natural fracture pattern of the Kraka Field was interactively characterized through borehole images (BHIs) and cores, prior to structural correlation with seismic data. The fracture characterization effort was primarily focused on the lateral fracture distribution, as the majority of Kraka wells are horizontal. Main emphasis has been put on the Ekofisk section, as it constitutes the primary target of these wellbores. Seismic was used to map faults and fracture zones away from the borehole and to identify regional structural trends.

The full integrated approach is schematically shown in Fig. 2. Specific details of each step are given in the following subsections.

3.1. BHI processing

The applied borehole image data was acquisitioned in wells drilled in the time period between 1989 and 1997. Microresistivity data in Formation MicroScanner (FMS)- or Formation MicroImager (FMI) quality was available in seven wellbores. Three wellbores surveyed by measurement while drilling technology were excluded from this study due to poor data resolution.

In the first step of the workflow, the microresistivity data were manually processed in Techlog, following the Schlumberger standard outline, illustrated in Fig. 3.

Of the wells surveyed by FMS and FMI tools, one is vertical, one is deviated at approximately 70° at reservoir level and five are horizontal (Table 1).

Compared to newer image data, the BHIs provided from the Kraka Field are of relatively poor image quality. Moreover, internal image quality variations often occur within single well sections. The latter is largely due to tool sticking, artificial signals and key seating, which is observed in most borehole images from the Kraka Field. Chalk sections directly below chert bands have been particularly hard to resolve, as the chert bands do not have planar surfaces and so cause errors in the pad

Table 1
Summary of studied BHI sections.

Well	Orientation	Tool	Length (m)
Well 1	Horizontal	FMS	691
Well 2	Deviated	FMS	730
Well 3	Horizontal	FMS	1987
Well 4	Horizontal	FMI	1704
Well 5	Horizontal	FMI	2391
Well 6	Horizontal	FMI	1681
Well 7	Vertical	FMI	205

alignment stage of processing. The cherts, being highly resistive compared to the chalk, are in turn well resolved. Consequently, so are chert associated fractures.

3.2. BHI interpretation

The reservoir chalk is characterized by internal non-planar resistivity contrasts (that are not an expression of bedding features), which may confuse automatic dip-picking algorithms and lead to incorrect picks. All images have therefore been manually interpreted according to dip-picking principles for horizontal wellbores.

The more than 17 000 interpreted dip picks were subsequently subjected to structural dip removal (with respect to top reservoir). Alonghole fracture densities were calculated, and then corrected for fracture and wellbore orientation using the Terzaghi correction (Terzaghi, 1965; Peacock et al., 2003). This corrects for the sampling bias due when the fractures are near parallel to the wellbores, and allows us to compare the true density of fracture sets with different orientations. This is important in the case of Kraka since all wells are drilled from a single platform located in the centre of the anticline, and hence form a radial pattern (Fig. 4). Moreover four of the seven wells are drilled in a NW-SE direction, perpendicular to the seismically mapped faults. It is therefore essential to compare the true densities of the different fracture sets to determine the main structural controls on fracture orientation.

3.3. Core to BHI correlation

The image interpretation scheme was developed through interactive evaluation of core data in the second- and third step of the workflow. Cores were available for three of the BHI-surveyed well sections (wells 1, 2 and 7). Depth matching between borehole images and cores was



Fig. 3. Processing of BHI data.

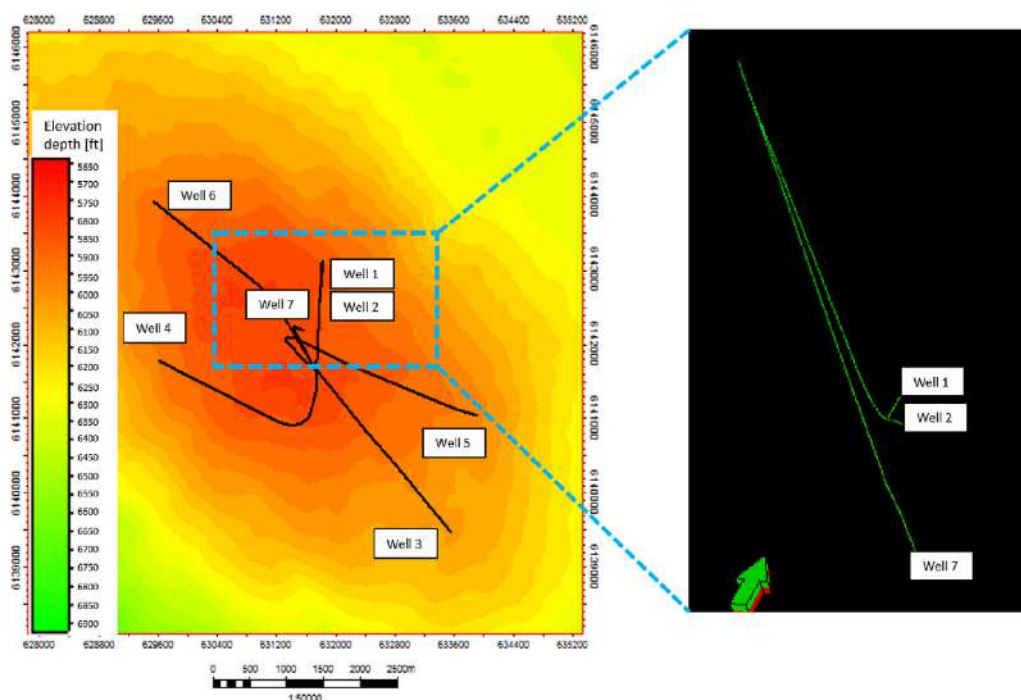


Fig. 4. Well pattern of Kraka wells included in this study on top reservoir depth map.

enabled by chert occurrences. Since the cherts are highly resistive compared to the reservoir chalk, they are easily identifiable on BHIs across the field. In the absence of a chert layer or nodule, core-to-log calibration is based on fractures. Depth shifts along wells vary by up to 8 ft. Relative fracture orientations measured in core have been re-oriented, depth shifted and plotted alongside image fracture-picks in the applied software.

Correlation between borehole images and cores is considered highly advantageous because:

1. BHI and core data are complementary. Borehole images provide true orientations and survey the reservoir in-situ, so we can differentiate open and closed fractures under reservoir conditions. The advantage of core is that we can identify smaller-scale stylolite associated fractures that are not detectable on images because the image resolution is about 1–2 mm.
2. In the BHI data from the Kraka Field, sinusoids representing bedding (chalk and marl) are continuous across borehole images. Most fractures are however only represented by partial sinusoids, either because they are short or because they are only partially open or cemented. Comparison with core data, when available, is imperative in determining which partial signals should be picked. Lessons learned from cored wells are transferable to BHI-surveyed wells without core.
3. In terms of azimuth, the dip-picking tools in the applied software (as in most commercial packages) are highly sensitive to small “tweaks”. This means that the orientation given by tadpoles can change drastically depending on how the partial fracture signal is picked. Where there is ambiguity, we have picked partial sinusoids to be consistent with nearby full sinusoids on borehole images, and calibrated against the fractures in core, where possible.

There is a general good correspondence between orientations of fractures identified in core and fractures picked on borehole images. An example of this is shown in Fig. 5, which shows a logged core interval

(a) with corresponding BHI section (b) in the vertical wellbore.

In this case, the core contains two natural fractures, represented by blue tadpoles in the borehole image. These are recognised as natural fractures in core by the presence of slickensides. In the core section, both fractures are open. However, the logged fractures coincide with one open (conductive) and one closed (resistive) fracture picked on the image section. Here, the image interpretation is considered reliable as BHIs represent the in-situ reservoir conditions. The closed fracture observed on borehole images could possibly have been opened during the coring process itself. The dip and azimuth of both fractures identified in BHI match the orientation of the fractures logged in core within 12°.

In general, dip angles of core- and BHI fracture picks fit to within 9° or less, while dip azimuths are associated with a higher degree of uncertainty. Small discrepancies are to be expected, as core must be reoriented manually to calculate true orientations. Therefore fracture orientations from BHIs are considered the most reliable, while the presence and type of fractures can be identified in the core.

Core-fracture densities are 44% and 36% higher than BHI-fracture densities in wells 1 and 2, respectively (Fig. 6). Stylolite occurrences in the chalk accounts for some of the disparity, as they are not resolved in the images. Moreover, fractures located in bioturbated zones and well-parallel fractures are difficult to distinguish in the BHIs. The remaining discrepancy is linked to the quality and resolution of images. The fracture density percentage for well 7 has not been computed, as there are few data points to compare.

3.4. Initial fracture data analysis and QC

The true orientations of the fracture picks in each well were plotted on upper hemisphere stereonet, and these were used to classify fractures into sets on the basis of orientation. The fracture strikes were then plotted on rose diagrams drawn over the stereonet, after applying the Terzaghi correction, to enable direct comparison of the fracture densities in the different sets, on a well by well basis.

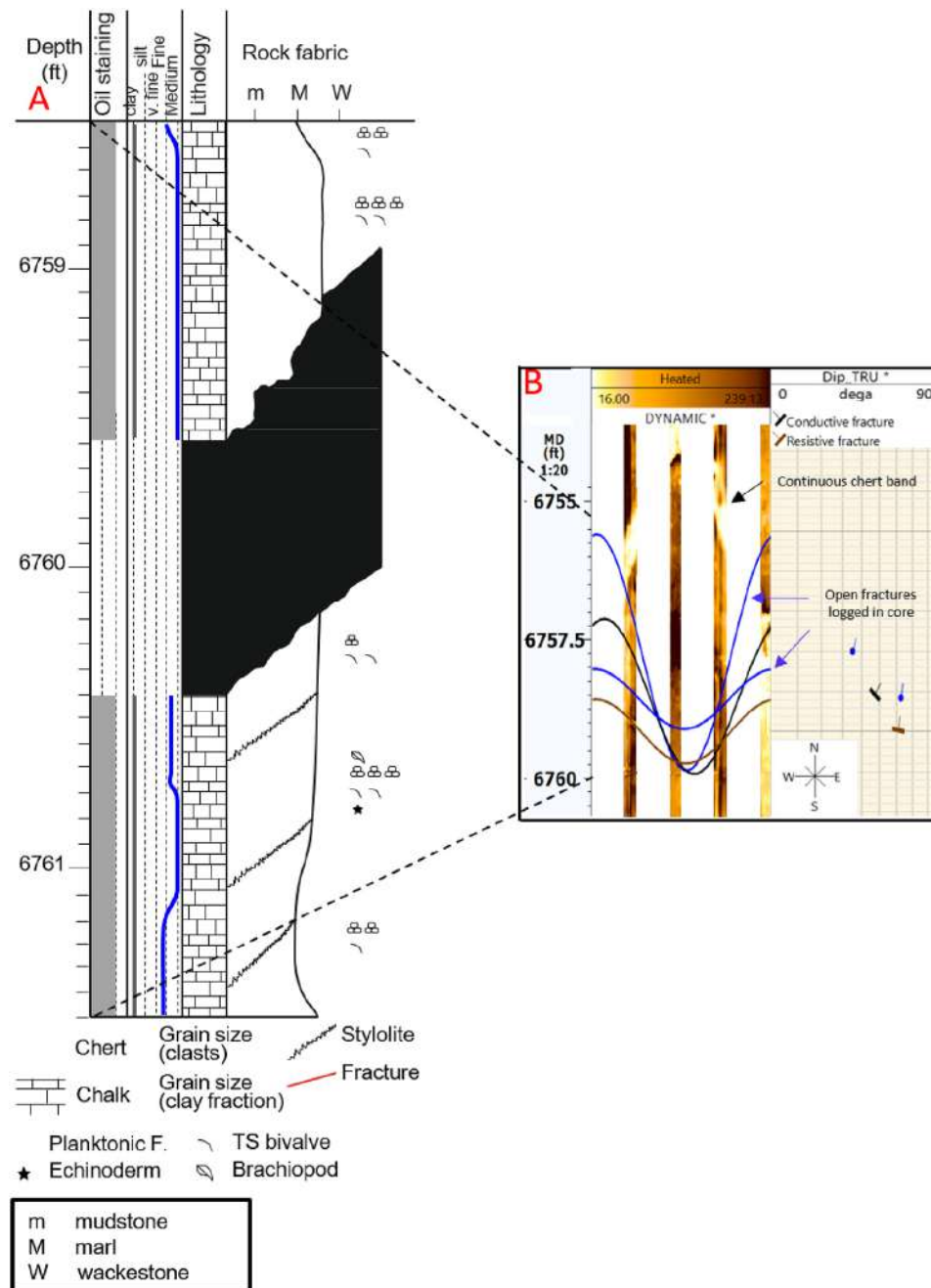


Fig. 5. Core log (a) and Core-to-BHI correlation (b) of fractured interval in the vertical wellbore. The core to log shift is approximately 2.5 ft, based on the chert band. The section of core shown corresponds to the section of borehole image log from 6756.0 to 6759.0 ft (as indicated by the black stippled lines).

3.5. Well-by-well and field scale evaluation

Alonghole fracture density logs were generated, with Terzaghi correction applied to individual fractures, to study the spatial distribution of fractures, and determine qualitatively whether they are clustered or evenly distributed. More quantitative analysis of the fracture distribution was carried out using cumulative density distribution plots, and by calculating the coefficient of variation of fracture spacing.

The cumulative fracture spacing distribution can be plotted by

measuring the distance between each pair of adjacent fractures along a wellbore (without correcting for orientation), ranking them in order, and plotting the results on a cumulative density distribution diagram. The cumulative density distribution with respect to fracture spacing is analogous to the cumulative density distribution with respect to fracture displacement or fracture length (see e.g. Marrett and Allmendinger, 1991, 1992; Westaway, 1994; Marrett, 1996): a straight line on a log-linear plot indicates a random fracture distribution (Olson et al., 2001), while a straight line on a log-log plot indicates a power

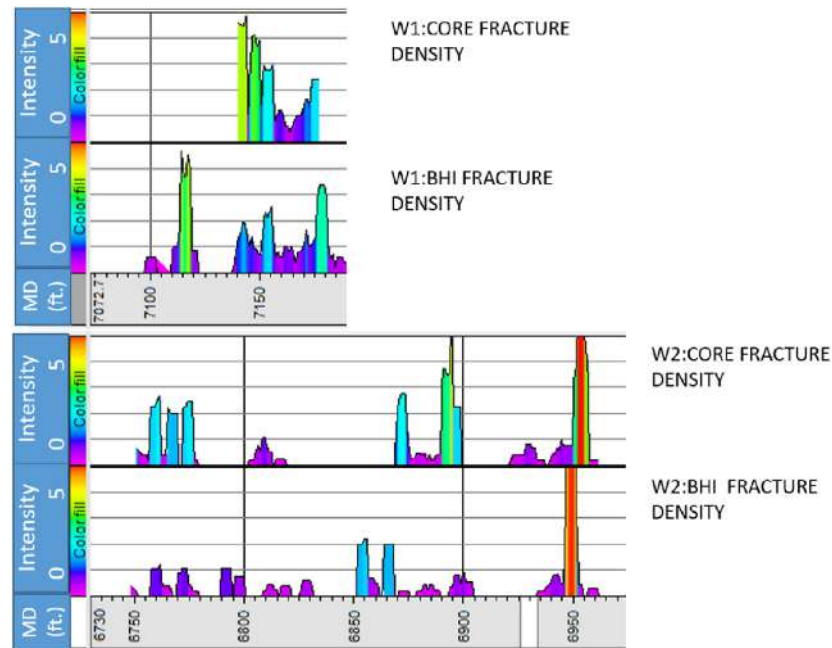


Fig. 6. Corrected fracture intensity logs of core-fractures and BHI-fractures in the cored intervals of wells 1 and 2 (scale 1:700). Fracture densities range from 0 in purple to 5 in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

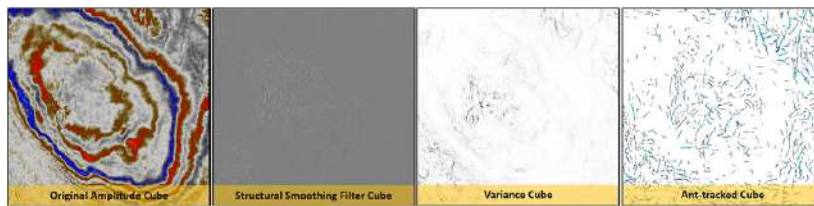


Fig. 7. Workflow for generating ant-tracked volume from the seismic depth cube of the Kraka Field in Petrel.

law distribution with clustering of fractures and may indicate fracture swarms (Gillespie et al., 1993, 2001).

The coefficient of variation of fracture spacing Cox and Lewis (1966) provides a more straightforward method to quantify the degree of clustering. The coefficient of variation R is defined as the standard deviation of fracture spacing divided by the mean fracture spacing:

$$R = \frac{\text{Standard deviation of fracture spacing}}{\text{Mean fracture spacing}} \quad (1)$$

A ratio $R > 1$ implies fracture clustering and the presence of swarms. A ratio of $R = 1$ implies a random fracture distribution. A ratio of $R < 1$ implies regularly spacing fractures.

The fracture distribution can tell us something about the mechanisms of fracture formation. In particular a clustered fracture distribution often develops when the stress anomaly develops around the tip of a propagating fracture, promoting the growth of nearby fractures, in a similar manner to the process zone often observed around igneous dykes (Olson, 2003). A modelling study by Olson (2004) shows that this is often the result of critical fracture propagation in a brittle material.

3.6. Well to seismic correlation

Fracture picks from BHIs and cores were subsequently compared to a structural framework derived from the amplitude seismic volume, provided by Maersk, as well as to two ant-tracked structural models in step 6 of the workflow. The seismic amplitude cube (in depth), acquired

in 2012, has a vertical resolution in the order of 40 m (sampling rate of 4 ms). The ant-tracked volumes enhance subtle faults and fracture zones that are below this vertical resolution. The ant-tracking algorithm systematically analyzes a seismic input cube – mimicking the swarm intelligence of ants (Pedersen et al., 2002). Here, a large number of agents (ants) are distributed in the volume. Each ant propagating through the cube is programmed to detect continuous structural lineations. Confidence levels are assigned depending on the length and width of the path of segments.

The first ant-tracked volume was generated in Petrel according to the following procedure (Fig. 7):

1. Cropped the original amplitude cube to speed up calculation.
2. Generated a structural smoothing/median filter cube to increase horizontal continuity and to pick out the more consistent structural features. The optimal degree of smoothing was achieved through adjusting the attribute parameter and observing its effect on the smoothing cube (in real time) prior to realization.
3. Extracted the variance/chaos cube from the smoothing cube to highlight discontinuities. The variance cube software is based on wavelet analysis. It calculates the direct measurement of dissimilarity rather than the inferred similarity of seismic data, producing sharper, more distinct results than those with traditional coherency techniques (Schlumberger, 2006).
4. Ran ant-tracking algorithm to enhance discontinuities.

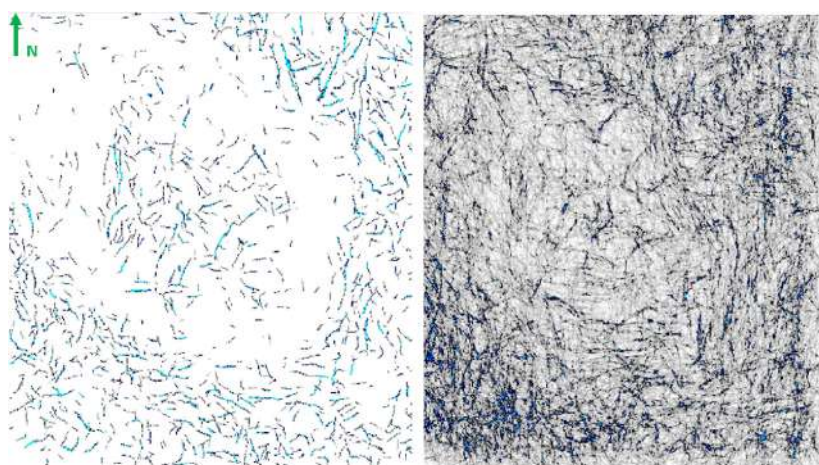


Fig. 8. Comparison of the wavelet-based (left) and the RGB-based (right) ant-tracked volumes.

The second ant-tracked volume was generated in eXchroma and Petrel according to the following procedure:

1. Cropped the original amplitude cube to speed up calculation.
2. Applied the structurally sharpened red-green-blue method, which uses the simultaneous rendering of multiple depth slices in continuous RGB color to highlight geophysical heterogeneities representative of geologic features, to the amplitude cube (Laake, 2015). The result is an image processed photo-style cube.
3. Ran ant-tracking algorithm to enhance discontinuities.

The vertical resolution of the wavelet based ant-tracked volume is 24 ms (7 times the sampling rate). In the RGB ant-tracked cube, the vertical resolution is 12 ms (3 times the sampling rate). The latter cube has therefore been preferentially used in this study (Fig. 8).

3.7. Correlation analyses

The interpretation of structural features along the well bore on seismic scale was carried out on an opacity mix of the ant-tracked volume and the seismic amplitudes (Fig. 9). This mixed view enables a focused interpretation of localized features in the seismic data. Also, we avoid misclassification of noise or acquisition artifacts. The opacity of the ant-track overlay was adjusted dynamically to enable the best interpretation possible.

The structural interpretation of the seismic data was done independently from the fracture interpretation of the BHIs. This reduces bias in the interpretation and “correlation finding” when looking at mixed displays. “Correlation finding” is a bias in interpretive science, where the interpreter has both displays open and finds feature in one display because they know to expect a feature from the other display.

Dip- and azimuth values were averaged along the fault planes of each interpreted fault to allow for direct comparison with well-scale data in upper hemisphere stereonet projections in the seventh- and final

step of the workflow.

4. Workflow outcomes: well-scale fracture trends

4.1. Fracture densities and fracture swarms

Fig. 10 shows the fracture orientations in the four NW-SE oriented horizontal wells prior to- and after applying the Terzaghi correction for borehole orientation. According to expectations, the rose diagrams for uncorrected fracture strike are dominated by NE-SW striking fractures, as fractures in this orientation will be preferentially intersected by the boreholes. After correcting for orientation, however, the rose diagrams largely remain unchanged, indicating that this reflects a real preferred fracture orientation, and not just bias due to well orientation. However in two of the wells (wells 3 and 4), the Terzaghi correction also reveals another fracture set, striking NW-SE, that cannot be identified on the uncorrected data. This suggests that multiple intersecting fracture sets are present in these locations, as well as demonstrating that the Terzaghi correction is correctly revealing the true preferred fracture orientations.

The uncorrected fracture orientations from the two wells oriented approximately north-south also show a majority of the fractures striking east-west (Fig. 11). In these wells, however, correction for wellbore orientation significantly reduces the relative importance of this fracture set, and reveals that the dominant fracture trend strikes north-south, parallel to the wellbores (although there is still a population of east-west fractures, so this location is likely to be characterized by multiple intersecting fracture sets).

In the Terzaghi-corrected fracture intensity logs from the seven wellbores, the fracture density and distribution varies between wellbores (Fig. 12). Relatively high and uniform fracture densities are observed in wells 4, 5 and 6. Mean fracture densities are lower in wells 1, 2 and 3, but the fractures in these wells appear more clustered, with potential fracture swarms observed. The highest fracture density is found in the vertical well 7 and the lowest fracture density occurs in the horizontal well 3.

The fracture distribution was investigated quantitatively by plotting cumulative density distribution plots for fracture spacing, and calculating the coefficient of variation for the fracture spacing in each well. Results were relatively consistent between all investigated wells. Generally, smaller fracture spacings (0.1–10 ft) follow a straight line on the log-log plot, indicating a close-to Power law distribution, while larger spacings (more than 10 ft) follow a straight line on the log-linear plot, indicating a close-to random distribution (Fig. 13). We can also see a clear distinction between wells 1, 2 and 3, characterized by high

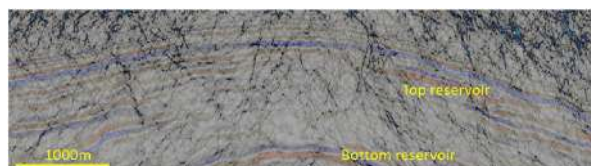


Fig. 9. Amplitude-cube over the Kraka Field with the RGB volume as opaque overlay, used for seismic interpretation.

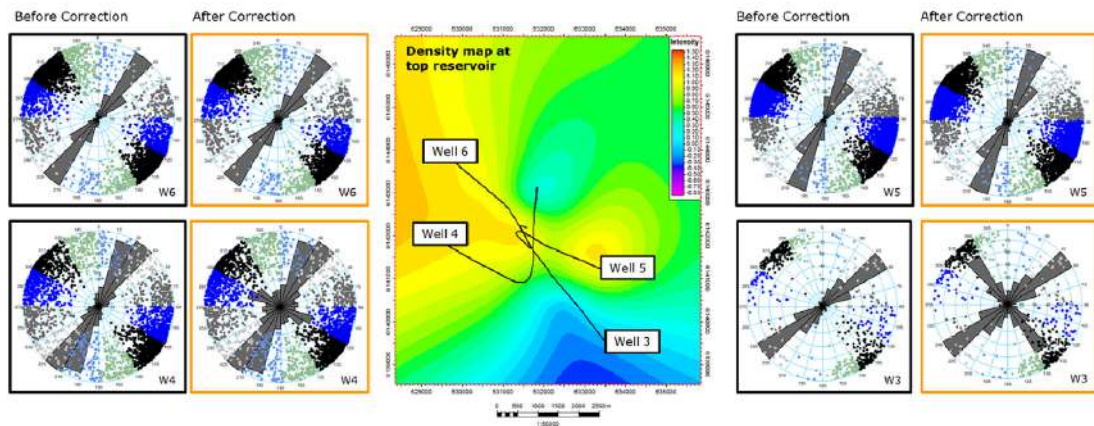


Fig. 10. Uncorrected and Terzaghi-corrected strike-rose diagrams from NW-SE oriented wells. Note that the fracture pole data, shown as dots on the stereonets (upper hemisphere), is uncorrected in both instances.

overall fracture densities (they intersect the y axis at high values) and high power law exponents (steep density distribution curves), and wells 4, 5, and 6, characterized by low overall fracture densities and low power law exponents.

The values of the mean, standard deviation and coefficient of variation R for the fracture spacings in each well are summarized in Table 2. In case of a clustered Power law distribution, the standard deviation should be smaller than the mean, so $R > 1$. In a perfectly random distribution, the standard deviation equals the mean, so $R = 1$.

This data confirms the observations from the fracture intensity plots. The mean fracture spacing is much higher in wells 1, 2 and 3 than in wells 4, 5 and 6, indicating a lower fracture density. As the coefficient of variation $R > 1$ in all wells, some clustering of the fractures occurs in all Kraka wells, but the distribution in wells 1, 2, and 3 is more clustered than in wells 4, 5 and 6. However the coefficient of variation R is still quite low in most wells (and is not much higher in wells 1 and 2 than in wells 4 and 5), suggesting a significant random component in

the distribution of fractures in all wells. The greatest clustering occurs in well 3, which is the well with the lowest fracture density, i.e. the highest mean spacing.

The fracture distribution and the coefficient of variation in vertical well 7 is in good correspondence with that observed in the horizontal wells. This suggests a fairly isotropic fracture distribution in Kraka. However, this inference is based on just one horizontal well, and must therefore be treated with caution.

4.2. Fracture sets

The majority of fractures observed are steeply dipping: 0.24% of the BHI-fractures are shallow dipping ($< 30^\circ$), 24.62% of the fractures have intermediate dip values ($30\text{--}70^\circ$) and 75.14% are steep ($70\text{--}90^\circ$) (although these figures are not corrected for orientation relative to the wellbores).

Fig. 14 shows upper hemisphere stereonets and orientation-

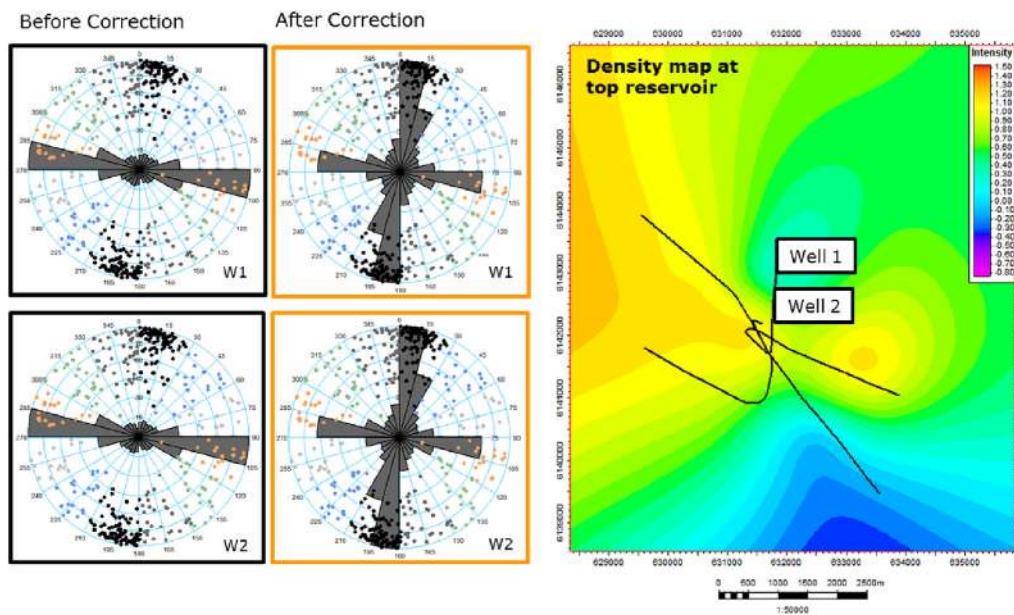


Fig. 11. Uncorrected and Terzaghi-corrected strike-rose diagrams from N-S oriented wells. Note that the fracture point data, shown in the stereonets (upper hemisphere), is uncorrected in both instances.

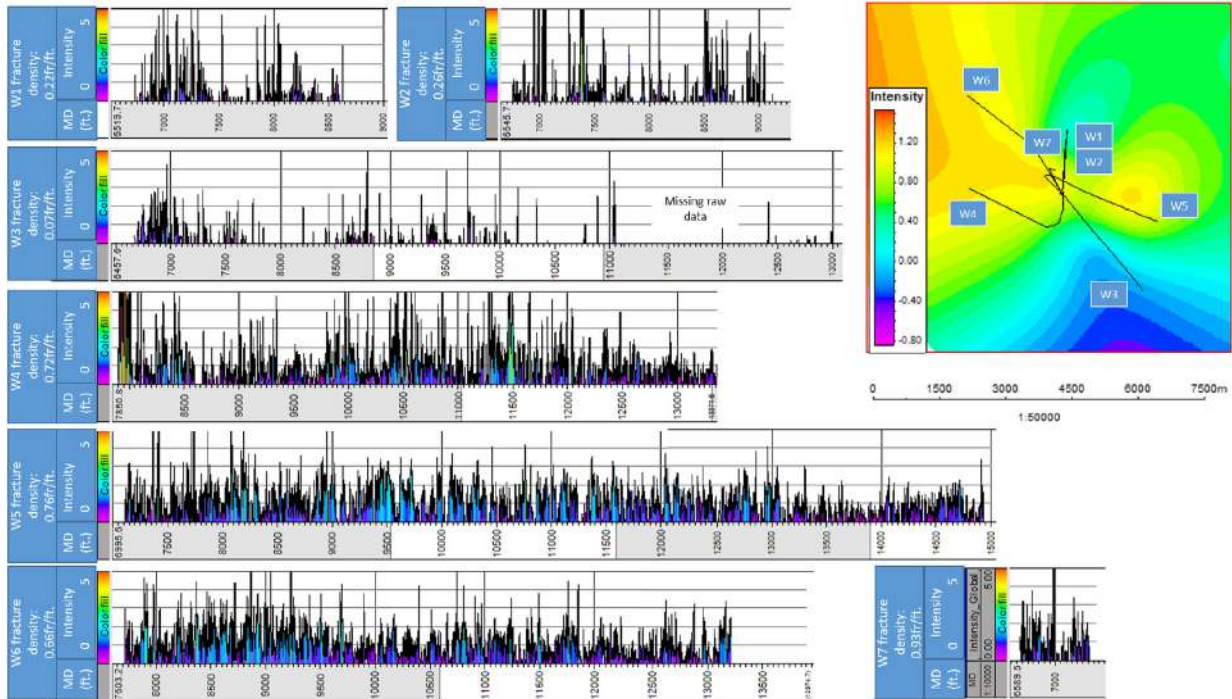


Fig. 12. Terzaghi-corrected fracture intensity logs of wells 1–7 (scale 1:10 000).

corrected rose diagrams for the fracture data from each well, broken down by stratigraphic unit. Data from the Danian Ekofisk formation is highlighted in green, and data from the Maastrichtian Tor formation is highlighted in red.

We have identified two main fracture trends in the Danian Ekofisk section. The first is a dominant NE/NNE trending regional fracture set, which strikes parallel or near-parallel to the maximum horizontal stress in the area. This main fracture trend is present in the Ekofisk intervals of all horizontal/deviated wellbores and has been confirmed by core data (from wells 1 and 2). Due to data constraints, the vertical fracture distribution of the Kraka Field was primarily studied in well 7. Although the data foundation is insufficient, results from this well indicate that the dominant NE/NNE trend of the Ekofisk formation is

Table 2

Standard deviations σ , mean values μ , and coefficient of variation R of the fracture spacing data from the seven Kraka wells.

Well	σ	μ	R
Well 1	8.3	4.2	2.0
Well 2	9.2	4.4	2.0
Well 3	21.4	7.2	3.0
Well 4	2.7	1.4	1.9
Well 5	2.2	1.2	1.9
Well 6	1.7	1.1	1.5
Well 7	4.5	2.9	1.5

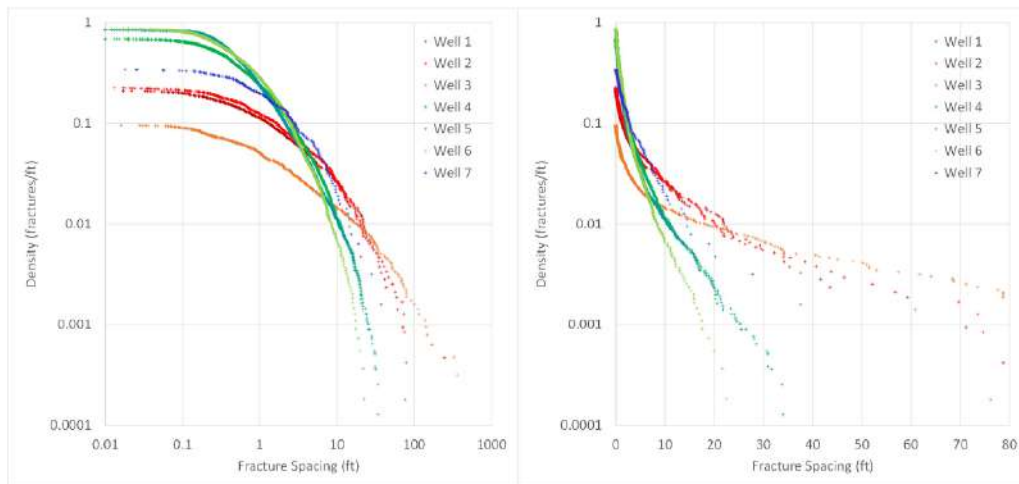


Fig. 13. Fracture spacing distributions for the seven Kraka wells, shown on a log-log plot (left) and log-linear plot (right).

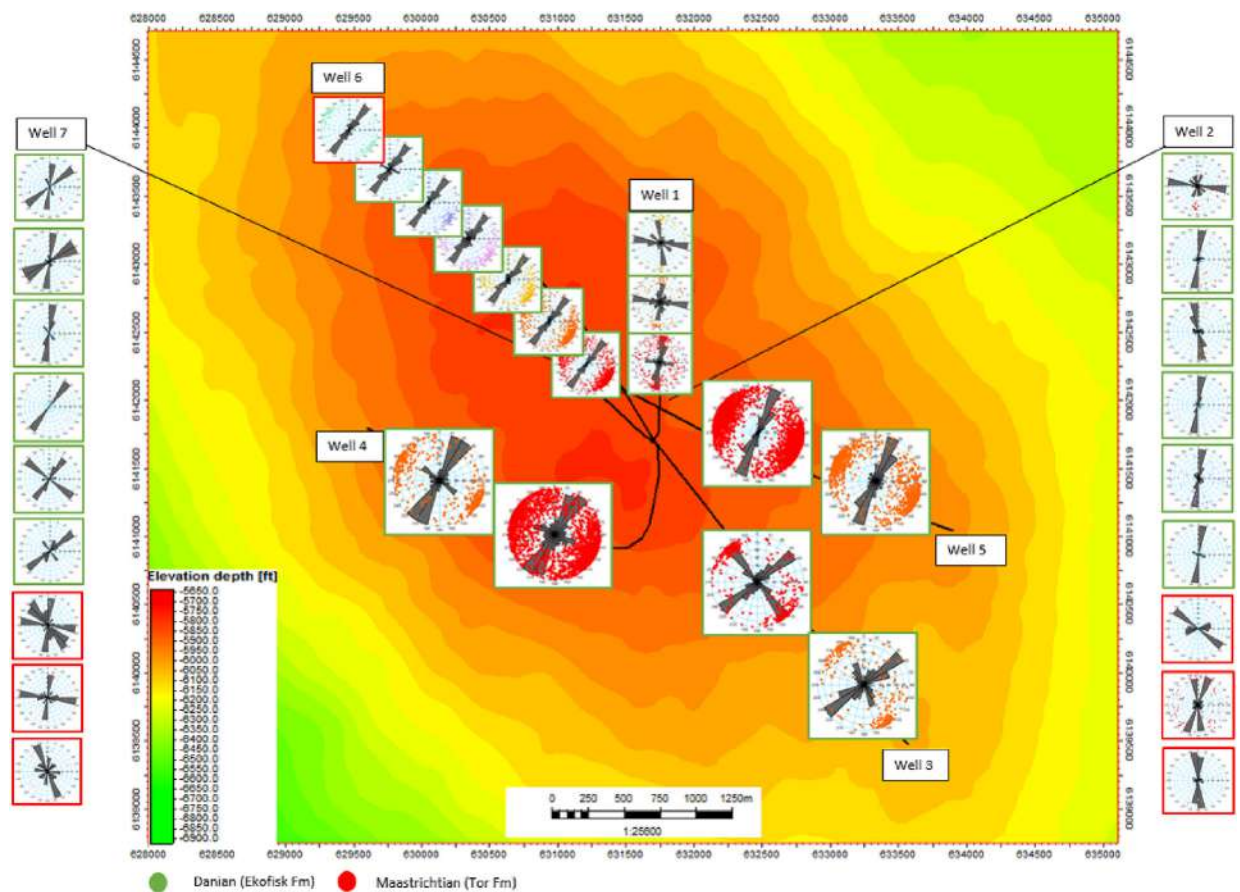


Fig. 14. Fracture point data and corrected rose diagrams in wells 1–7, plotted after unit on the top reservoir depth map.

vertically continuous.

The secondary fracture set consists of fractures striking parallel and perpendicular to the contours of the Kraka Dome. The orientation of these fractures varies between wells, depending on their location on the dome. This fracture set is thought to have formed during salt movements, and it is expected to follow the strain evolution of the Kraka chalk. Because of the positions of wells 5 and 6, the two fracture sets in cannot be distinguished on the basis of orientation in these wells (the dome contours are parallel to the NE/NNE regional trend).

The NNE/NE trending regional fracture set continues into the Tor Formation of wells 2, 6 and 7. However there is much more scatter in the Tor orientation data than in the Ekofisk data. The scattering may be due to varying local stresses during salt movements, however there is insufficient data from the Tor Formation to determine the fracture pattern with confidence.

5. Workflow outcomes: seismic-scale fault trends

The orientation data for the lineations observed on RGB ant-tracked seismic data, and for the large-scale faults interpreted on amplitude seismic are shown in Fig. 15. The ant-tracked structural model contains 25 lineations along the well trajectory. The majority (16) of these lineations strike NNE/NE. Of the 32 large-scale faults interpreted on amplitude seismic, 18 are oriented in a NE direction, while only 3 faults strike NNW. This indicates that the NNE trend is representative for smaller-scale lineations, at the limit of resolution of amplitude seismic. The rose diagram for the ant-tracked lineations shows higher variance than the large-scale faults. This implies greater variation in the

orientation of small scale features in comparison to large-scale fault trends, which we would expect.

Overall the NE/NNE trend of the lineations and the large-scale faults matches the orientation of one of the main sets of fractures observed on borehole images. The main NE/NNE fracture trend can therefore be correlated from wellbore-scale fractures to local lineation scale and to field-wide fault scale (compare Figs. 14 and 15). This suggests that many of the wellbore-scale fractures are genetically related to the seismic-scale faults, and formed in response to a regional stress regime. The resulting fracture and fault set covers a range of scales, from small fractures with lengths ≥ 1 m up to seismic-scale faults with lengths of several km. The increased variance in orientation of the small-scale features may be partly due to uncertainty of the interpretation, and partly due to small-scale local variations in stress regime, for example around larger faults or around the salt diapir.

There are of course many more fractures observed on borehole images than lineaments observed on the RGB ant-tracked seismic data, and most of the individual fractures observed on the borehole images are of a much smaller scale than the lineaments. There is therefore no direct correspondence between fractures on borehole images and lineaments observed on the ant-tracked seismic data. Nor do we see a clear correlation between the density of fractures on borehole images and the location of the lineaments (Fig. 16). This may partly reflect variation in borehole image quality: it was not possible to pick so many fractures in areas of poor image quality.

However, if we filter the fractures from borehole images to only include the resistive fractures, we do see a clear correlation in many wells, between both the distribution and the orientation of resistive

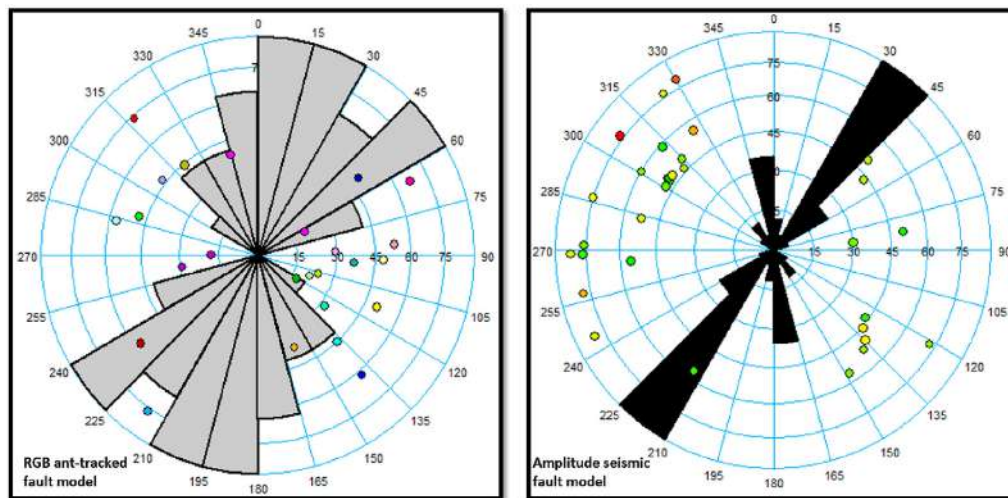


Fig. 15. Rose diagrams of structural lineations interpreted on the RGB ant-tracked volume (left) and the amplitude volume (right).

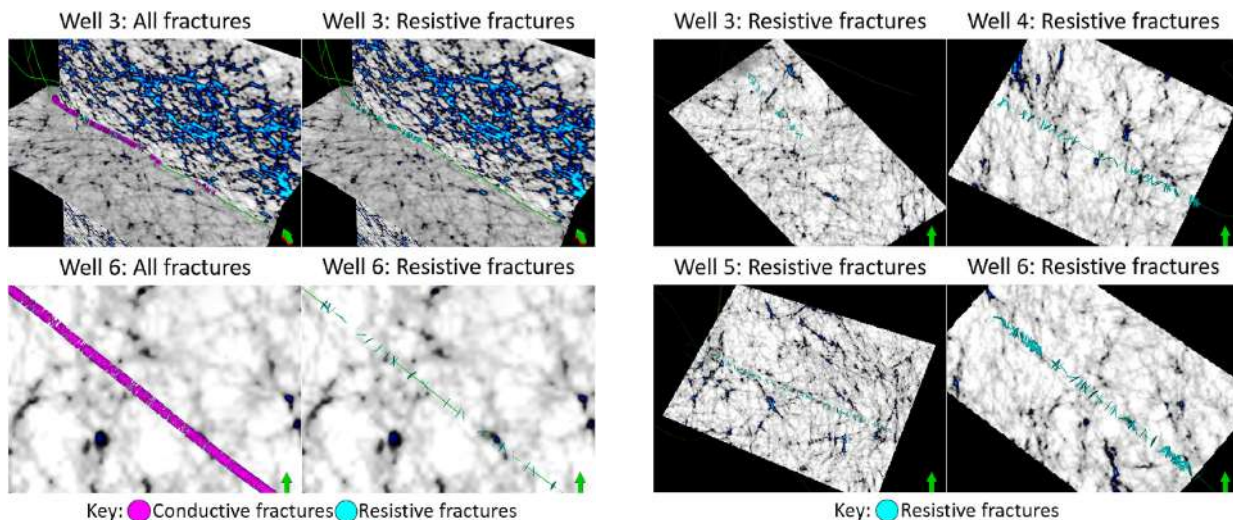


Fig. 16. Comparison of fractures interpreted on borehole images, shown by coloured disks, and lineations observed on RGB ant-tracked seismic data, for wells 3 and 6. The left hand pictures show all fractures interpreted on borehole images, with conductive fractures shown by pink disks and resistive fractures by blue disks, while the right hand pictures show only the resistive fractures. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

Fig. 17. Comparison of the resistive fractures interpreted on borehole images, shown by blue disks, and lineations observed on RGB ant-tracked seismic data, for wells 3, 4, 5 and 6. All wells are shown looking down from above. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

fractures and the seismic lineaments. This is particularly clear for the NW-SE oriented horizontal wells 3, 4, 5 and 6 (Fig. 17). This suggests that the resistive fractures observed on borehole images may be more likely to represent or be associated with larger-scale structures than the conductive fractures. This is quite plausible, as resistive fractures are filled with some resistive material, which may be fault gouge or cement precipitated from a fluid travelling through the fracture. Thus we would expect faults with significant displacement, or fractures associated with such small faults (e.g. in the damage zone), to appear as resistive fractures on borehole images. Conductive fractures, by contrast, could mostly represent a distributed background population of small, unfilled cracks.

6. Summary and conclusions

We have established an integrated workflow for correlation of structural features at different scales in chalk reservoirs. The workflow bridges the scale-gap between well-scale and seismic data sets, and has increased our understanding of the natural fracture system in our test case, Kraka. Such knowledge can be used strategically in optimizing production schemes and obtaining sustainable recovery factors. The combination of BHI- core- and seismic data allow for the extrapolation of fractures away from the borehole.

An obvious next step in this workflow is upscaling, fracture modelling and evaluation of current findings through fluid simulations. Such simulations, verified through production data, might provide feedback to the fracture characterization effort, further improving the model (closed loop).

Application of our approach in the Kraka Field indicate that:

1. A large portion of extensional and chert-associated fractures identified in core are distinguishable in borehole images. Stylolites, stylolite-associated fractures, fractures located in bioturbated zones and well-parallel fractures are however difficult to differentiate in the BHIs.
2. Borehole images are imperative in distinguishing cemented from open fractures, as cemented fractures may be opened during the coring process. BHIs thus increase our ability to constrain fluid flow along the fracture network.
3. Extensional and chert-associated fractures are commonly represented by partial sinusoids in BHIs, suggesting they are either short or only partially open or cemented.
4. Manual dip-picking was deemed requisite because of:
 - The large portion of fractures represented by partial sinusoids.
 - Relatively poor image quality (compared to newer BHI data).
 - Internal resistivity variations, which may “confuse” automatic dip-picking tools.
5. For the ant-tracked algorithm, higher vertical resolution can be achieved through RGB image processing, compared to wavelet based extraction of structural features. Both ant-tracked volumes display structural trends that are below the resolution of amplitude seismic.
6. Fractures picked on BHIs correlate to large-scale regional trends and to features picked on ant-tracked seismic data. This strengthens the case of a consistent regional stress field that scales down to local stresses observed at the BHI and core scale. We can therefore extrapolate fractures away from the wellbores and calibrate 3D models (e.g. discrete fracture network models).

In our test case, the results of the workflow show that the Ekofisk Formation of the Kraka Field is characterized by steep fractures striking NE and NNE, parallel or near-parallel to the maximum horizontal stress in the area. Fractures in Kraka occur as swarms and as isolated features. Moderate fracture clustering occurs in the majority of horizontal wells, as well as in the vertical well. The greatest tendency for fracture swarm occurrence is observed in the horizontal well with the lowest associated fracture density.

The main fracture trend, established from borehole images and core, is present in the Ekofisk sections of all horizontal/deviated wellbores and has been confirmed by core data. Results from the vertical well 7 indicate that the dominant NE/NNE trend of the Ekofisk Formation is vertically continuous. A secondary fracture set of fractures striking parallel and perpendicular to the contours of the Kraka Dome was identified. The orientation of these fractures varies between wells, depending on their location on the dome. This fracture set likely developed during salt movements and is expected to follow the strain evolution of the Kraka chalk.

The main NNE/NE fracture trend can be correlated from well scale to ant-tracked scale. Faults mapped on amplitude seismics can also be identified in the ant-tracked cube. In the amplitude model, faults mainly trend NE, indicating that the NNE trend is representative for smaller-scale lineations (well scale to ant-tracked scale).

This integrated study proves invaluable in testing assumptions in building fracture models and the subsequent upscaling process and will be useful for validating a geomechanically based DFN for the Kraka Field.

Acknowledgements

The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre (DHRTC) under the Advanced Water Flooding programme (AWF project no: 16/00787/

ginet).

We thank DUC for providing data access and F. Amour for his useful input. We would also like to extend our thanks to Schlumberger for providing licenses for Petrel, eXchroma and Techlog.

Appendix A. Supplementary data

Supplementary data to this article can be found online at <https://doi.org/10.1016/j.marpetgeo.2019.104065>.

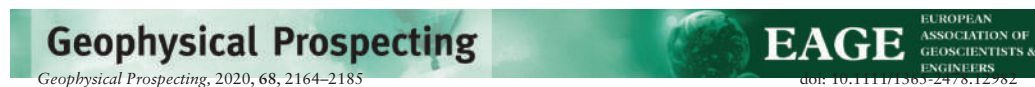
References

- Cox, D.R., Lewis, P.A.W., 1966. *The Statistical Analysis of Series of Events*. Springer Netherlands.
- Fernø, M.A., 2012. Enhanced oil recovery in fractured reservoirs. In: *Introduction to Enhanced Oil Recovery (EOR) Processes and Bioremediation of Oil-Contaminated Sites*. InTech.
- Gillespie, P., Howard, C., Walsh, J., Watterson, J., 1993. Measurement and characterisation of spatial distributions of fractures. *Tectonophysics* 226, 113–141. [https://doi.org/10.1016/0040-1951\(93\)90114-y](https://doi.org/10.1016/0040-1951(93)90114-y).
- Gillespie, P., Walsh, J., Watterson, J., Bonson, C., Manzocchi, T., 2001. Scaling relationships of joint and vein arrays from the burren, co. clare, Ireland. *J. Struct. Geol.* 23, 183–201. [https://doi.org/10.1016/S0191-8141\(00\)00090-0](https://doi.org/10.1016/S0191-8141(00)00090-0).
- Jørgensen, L., Andersen, P., 1991. Integrated study of the kraka field. In: *Offshore Europe*, Society of Petroleum Engineers.
- Klinkby, L., Kristensen, L., Nielsen, E.B., Zinck-Jørgensen, K., Stemmerik, L., 2005. Mapping and characterization of thin chalk reservoirs using data integration: the kraka field, Danish north sea. *Pet. Geosci.* 11, 113–124. <https://doi.org/10.1144/1354-079304-632>.
- Koestler, A., Reksten, K., 1992. Insight into the 3d fracture network of an exposed analogue of fractured chalk reservoirs—the lågerdorf case. In: *Proc. Fourth North Sea Chalk Symposium*.
- Laake, A., 2015. Structural interpretation in color — a new RGB processing application for seismic data. *Interpretation* 3, SC1–SC8. .
- Marrett, R., 1996. Aggregate properties of fracture populations. *J. Struct. Geol.* 18, 169–178. [https://doi.org/10.1016/S0191-8141\(96\)80042-3](https://doi.org/10.1016/S0191-8141(96)80042-3).
- Marrett, R., Allmendinger, R.W., 1991. Estimates of strain due to brittle faulting: sampling of fault populations. *J. Struct. Geol.* 13, 735–738. [https://doi.org/10.1016/0191-8141\(91\)90034-g](https://doi.org/10.1016/0191-8141(91)90034-g).
- Marrett, R., Allmendinger, R.W., 1992. Amount of extension on “small” faults: an example from the viking graben. *Geology* 20, 47. [https://doi.org/10.1130/0091-7613\(1992\)020<0047:aeosf>2.3.co;2](https://doi.org/10.1130/0091-7613(1992)020<0047:aeosf>2.3.co;2).
- Møller, J.J., Rasmussen, E.S., 2003. Middle jurassic–early cretaceous rifting of the Danish central graben. The Jurassic of Denmark and Greenland. *Geol. Surv. Den. Greenl. Bull.* 1, 247–264.
- Olson, J.E., 2003. Sublinear scaling of fracture aperture versus length: an exception or the rule? *J. Geophys. Res.: Solid Earth* 108. <https://doi.org/10.1029/2001jb000419>.
- Olson, J.E., 2004. Predicting fracture swarms—the influence of subcritical crack growth and the crack-tip process zone on joint spacing in rock. *Geol. Soc. Lond. Spec. Publ.* 231, 73–88. <https://doi.org/10.1144/GSL.SP.2004.231.01.05>.
- Olson, J.E., Yuan, Q., Holder, J., Rijken, P., 2001. Constraining the spatial distribution of fracture networks in naturally fractured reservoirs using fracture mechanics and core measurements. In: *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers.
- Peacock, D., Harris, S., Mauldon, M., 2003. Use of curved scanlines and boreholes to predict fracture frequencies. *J. Struct. Geol.* 25, 109–119. [https://doi.org/10.1016/S0191-8141\(02\)00016-0](https://doi.org/10.1016/S0191-8141(02)00016-0).
- Pedersen, S.I., Randen, T., Sonneland, L., Steen, Ø., 2002. Automatic fault extraction using artificial ants. In: *SEG Technical Program Expanded Abstracts 2002*. Society of Exploration Geophysicists.
- Rank-Friend, M., Elders, C.F., 2004. The evolution and growth of central graben salt structures, salt dome province, Danish north sea. *Geol. Soc. Lond. Mem.* 29, 149–164. <https://doi.org/10.1144/gsl.mem.2004.029.01.15>.
- Rasmussen, E.S., Vejebak, O.V., Bidstrup, T., Piasecki, S., Dybkjær, K., 2005. Late cenozoic depositional history of the Danish north sea basin: implications for the petroleum systems in the kraka, halfdan, siri and nini fields. *Geol. Soc. Lond. Pet. Geol. Conf. Ser.* 6, 1347–1358. <https://doi.org/10.1144/0061347>.
- Schlumberger, 2006. Fault resolution improved for norsk hydro's varg field. case study: variance cube software enhances imaging and reduces processing time. http://www.slb.com/media/Files/software/case_studies/varcube_cs.ashx, Accessed date: 20 November 2017.
- Terzaghi, R.D., 1965. Sources of error in joint surveys. *Geotechnique* 15, 287–304. <https://doi.org/10.1680/geot.1965.15.3.287>.
- Westaway, R., 1994. Quantitative analysis of populations of small faults. *J. Struct. Geol.* 16, 1259–1273. [https://doi.org/10.1016/0191-8141\(94\)90068-x](https://doi.org/10.1016/0191-8141(94)90068-x).

B.2 Deep neural network application for 4D seismic inversion to changes in pressure and saturation: Optimizing the use of synthetic training datasets

Abstract: In this work, we tackle the challenge of quantitative estimation of reservoir dynamic property variations during a period of production, directly from four-dimensional seismic data in the amplitude domain. We employ a deep neural network to invert four-dimensional seismic amplitude maps to the simultaneous changes in pressure, water and gas saturations. The method is applied to a real field data case, where, as is common in such applications, the data measured at the wells are insufficient for properly training deep neural networks, thus, the network is trained on synthetic data. Training on synthetic data offers much freedom in designing a training dataset, therefore, it is important to understand the impact of the data distribution on the inversion results. To define the best way to construct a synthetic training dataset, we perform a study on four different approaches to populating the training set making remarks on data sizes, network generality and the impact of physics-based constraints. Using the results of a reservoir simulation model to populate our training datasets, we demonstrate the benefits of restricting training samples to fluid flow consistent combinations in the dynamic reservoir property domain. With this the network learns the physical correlations present in the training set, incorporating this information into the inference process, which allows it to make inferences on properties to which the seismic data are most uncertain. Additionally, we demonstrate the importance of applying regularization techniques such as adding noise to the synthetic data for training and show a possibility of estimating uncertainties in the inversion results by training multiple networks.

G. C  rte, J. S. Dramsch, H. Amini, and C. MacBeth (2020). “Deep neural network application for 4D seismic inversion to changes in pressure and saturation: Optimizing the use of synthetic training datasets”. In: *Geophysical Prospecting* 68.7. Published, Appendix B.2, pp. 2164–2185



Deep neural network application for 4D seismic inversion to changes in pressure and saturation: Optimizing the use of synthetic training datasets

Gustavo Côrte^{1*}, Jesper Dramsch², Hamed Amini^{1,†} and Colin MacBeth¹

¹Heriot-Watt University, Edinburgh, EH14 4AS, UK, and ²Technical University of Denmark, Copenhagen, Denmark

Received February 2020, revision accepted May 2020

ABSTRACT

In this work, we tackle the challenge of quantitative estimation of reservoir dynamic property variations during a period of production, directly from four-dimensional seismic data in the amplitude domain. We employ a deep neural network to invert four-dimensional seismic amplitude maps to the simultaneous changes in pressure, water and gas saturations. The method is applied to a real field data case, where, as is common in such applications, the data measured at the wells are insufficient for properly training deep neural networks, thus, the network is trained on synthetic data. Training on synthetic data offers much freedom in designing a training dataset, therefore, it is important to understand the impact of the data distribution on the inversion results. To define the best way to construct a synthetic training dataset, we perform a study on four different approaches to populating the training set making remarks on data sizes, network generality and the impact of physics-based constraints. Using the results of a reservoir simulation model to populate our training datasets, we demonstrate the benefits of restricting training samples to fluid flow consistent combinations in the dynamic reservoir property domain. With this the network learns the physical correlations present in the training set, incorporating this information into the inference process, which allows it to make inferences on properties to which the seismic data are most uncertain. Additionally, we demonstrate the importance of applying regularization techniques such as adding noise to the synthetic data for training and show a possibility of estimating uncertainties in the inversion results by training multiple networks.

Key words: Reservoir geophysics, Time lapse seismic, Inversion, Machine learning.

INTRODUCTION

Estimating dynamic reservoir property change during a period of field production from four-dimensional (4D) seismic data has been a challenge and ambition for geoscientists in the oil and gas industry. These estimates are appealing for reservoir

monitoring and history matching purposes, because 4D seismic data offer information about reservoir property changes across the whole reservoir, at a specific production time. It complements well production information, which is spatially sparse but temporally dense. 4D seismic data provide information for the space between wells. But this information is encoded into the measured seismic amplitudes. So, we need to comprehend how the changes occurring inside the reservoir affect the seismic amplitudes we measure.

[†]Currently at Aker BP, Stavanger, Norway

*E-mail: gac2@hw.ac.uk

Along field production, the reservoir goes through constant change in properties such as fluid saturation, pore pressure, temperature, or even to changes in the reservoir rock architecture itself due to compaction and dissolution. The change in each of these properties has an independent impact on the seismic data, but they seldom act alone. Water injection for example leads to an increase in water saturation and an increase in pressure in the vicinities of the injector well. The observed 4D seismic amplitudes are a superposition of all the effects caused by the simultaneous variations in any dynamic property. The challenge is in quantitatively estimating the simultaneous contribution of each reservoir property to the final observed data. As is common in geophysical inversion, this is an underdetermined problem, prone to ambiguities and highly uncertain. Seismic information is limited and cannot provide enough independent measurements to characterize the whole reservoir state.

The information present in the variation of 4D amplitudes with offset (4D AVO) is crucial for quantifying multiple simultaneous reservoir property changes. We can highlight two major theoretical studies that use analytical solutions to show the possibility of quantifying the changes in both pressure and saturation directly from the 4D AVO data. In the first, Landrø (2001) follows a linearization of Smith and Gidlow's approximation to the reflection coefficient equation (Smith and Gidlow, 1987) to analytically derive a linear relation linking the 4D AVO gradient/intercept seismic attributes to the changes in two reservoir properties, pressure and water/oil saturation. The derived equations depend on rock physics-related parameters that can be estimated using laboratory measurements. In a different approach, Alvarez and Macbeth (2014) follow a linearization of the Aki and Richards' (1980) approximation to the reflection coefficient equation to derive an angle-dependent linear relation between the changes in pressure and oil/water saturations and the 4D seismic amplitudes. This relation also depends on reservoir petro-elastic parameters. Additionally, MacBeth *et al.* (2006) developed a data-based inversion method that assumes a linear link between any 4D seismic attribute and the changes in pressure and saturation. The parameters in the equations here are not directly related to any petro-elastic property, instead, they need to be previously calibrated using repeated well measurements, in a similar manner as the way neural networks are trained using direct observations. In this study, the authors run a principal component analysis to determine the best 4D seismic attributes for the simultaneous quantification of pressure and saturation changes (Florich, 2006). The authors conclude that the 4D AVO related attributes contribute the most to the

inversion process and that the separation of effects could not be done without this AVO information.

Other methods take advantage of the 4D AVO information in many domains (impedance, gradient/intercept, seismic amplitudes) to invert for the changes in different reservoir properties such as pressure, compaction, and the saturations of water and gas (Trani *et al.*, 2011; Coleou *et al.*, 2013; Corzo *et al.*, 2013; Davolio *et al.*, 2013; Omofoma, 2017; Wong, 2017; Côte *et al.*, 2019). Most of these studies stress that there is a good deal of ambiguity and uncertainty in the solutions, thus any available external information should be used to constrain and/or regularize the inversion process (Blanchard and Thore, 2008; Blanchard, 2012). External information may come from the wells, as in MacBeth *et al.* (2006) and Coleou *et al.* (2013), where the authors use well-injected and produced volumes as global constraints to the saturation results. Reservoir simulation models can also be used to provide information to guide the inversion results. Davolio *et al.* (2013) and Omofoma (2017) use multiple realizations of a reservoir simulation model to define local hard bounds, constraining the possible inversion results. Côte *et al.* (2019) use the results of a history matched simulation model as local prior information in a Bayesian inversion approach to regularize the solution and provide soft constraints to the inversion results. Additionally, 4D seismic time-shift measurements have also been used as a data-based source of information in simultaneous inversion processes (Trani *et al.*, 2011; Thore and Hubans, 2012). In neural network solutions, we do not have the possibility of applying direct constraints to the inference process. Constraining the training dataset does not guarantee constrained results either, as the network can extrapolate beyond the training dataset. In this paper, we show a few techniques that can be used in the construction of the network architecture and training dataset that contribute to regularizing and constraining the inversion results.

The construction of the training dataset is a critical step that has great impact on the inversion results. Neural network applications, as opposed to most of the mentioned studies, do not rely on a physical model to establish the links between the seismic and reservoir domains. Instead, they rely on a training dataset composed of real input–output measurements, learning from it the non-linear relations that link inputs to outputs. The training dataset defines the 'physical' model that is used in the inversion, so it is important that it contains a good physical representation of the whole problem. In this case, a good training dataset should represent the whole reservoir, containing the global variability on reservoir quality and the possible dynamic property combinations.

2166 G. Côte et al.

Measured data to compose a training dataset can only come from repeated well measurements at seismic acquisition times. MacBeth *et al.* (2006) use well data in a model calibration approach analogous to neural network training. The authors pre-define a linear equational link between the seismic and reservoir domains and iteratively calibrate the equations' parameters to fit the data measured at a few well locations. The main difference to neural network training is that deep neural networks (DNNs) contain thousands more parameters to be calibrated, leading to much more complex non-linear relations. Consequently, deep neural networks need a much larger amount of data for satisfactory training. Nonetheless, Cao and Roy (2017) perform a synthetic study showing that a neural network can also be trained successfully using only information at well locations in a 4D reservoir property inversion application. In real reservoir cases though, the necessary repeated saturation well logs are not common and may be lacking as whole, as is true in our case study. This type of data is sparse, and it can be argued that it may be biased to good reservoir areas, where the wells are located, and thus, incapable of representing the entire reservoir. More often than not, just the well data are not sufficient to properly train a neural network.

The alternative is to use synthetic data to help in the construction of the training dataset. Ayzenberg and Liu (2014) present a real reservoir case of a neural network application to 4D pressure and saturation inversion where the authors populate their training dataset with reservoir simulation results and real seismic observations at a few well-understood areas. To extend their training dataset beyond the wells, they begin a shift to synthetic data, but only on the reservoir domain, keeping the real observed seismic data. Xue *et al.* (2019) use a fully synthetic dataset to train their neural network to quantify the changes in water saturation on a real reservoir case. The authors make use of a wedge model as a static frame and random sampling of the dynamic domain. More recently, Zhong *et al.* (2020) presented a solution using convolutional generative adversarial networks to invert impedance change images to reservoir property changes. Their convolutional approach analyses full images, incorporating a spatial correlation aspect into the inference results. For this reason, their synthetic training dataset is composed of full reservoir images, created by running 300 reservoir simulations with varying static models. Although convolutional networks are the state-of-the-art in image analysis, they require an immense amount of previous work to prepare synthetic training datasets.

This paper presents a DNN application to inverting 4D AVO seismic data into the simultaneous changes in three reser-

voir properties: pressure (ΔP), water saturation (ΔS_w) and gas saturation (ΔS_g). It provides a quick and practical alternative to more well-established inversion methodologies. As a good platform for comparison, we present a Bayesian model-based inversion approach applied to the same dataset in Côte *et al.* (2019), and a direct comparison of methods in Dramschet *et al.* (2019a).

The DNN is trained with synthetic data and applied to real 4D seismic data from a North Sea field. We use a reservoir simulator to seismic modelling approach (Sim2Seis) to construct four synthetic training datasets with the objective of assessing the impact of the distribution of data in the training dataset on the quality of the inversion results when applied to a real 4D seismic dataset. The training datasets presented differ essentially on how much external physical information is used to constrain and distribute the data. We show the value of using physics informed and fluid flow consistent realizations to create a realistic distribution of data in a synthetic training dataset. Furthermore, we show the importance of training the DNN on noisy synthetic data and the possibility of estimating uncertainties in the results by training multiple DNN models with varying signal to noise levels. With this we address the problems of constraining the results with external physical information and regularizing solutions to avoid overfitting of the training data and inverting noise.

FIELD AND DATASET

The field is composed of stacked turbidite channel and sheet-like sands ranging from 5 to 30 m in thickness and 25–30% in porosity. It is highly compartmentalized both laterally and vertically due to faults and intercalating shales. The sandstone reservoir is present in four adjacent fault blocks. Faults between blocks are sealing, creating four isolated segments with varying water oil contacts (Fig. 1b). This whole structure dips and thickens to the north-west (Fig. 1a). Inside each segment there are faults that may be sealing or not, leading to a few isolated compartments and a highly complex geological setup. Figure 1(c) shows the vertical sum of pore volume in the reservoir, where we see clearly the channel features. Detailed explanations of the depositional and stratigraphic evolution of the region can be found in Ebdon *et al.* (1995) and Lamers and Carmichael (1999).

The initial pressure in the field was only around 3 MPa above bubble point pressure, making pressure maintenance to prevent gas exsolution the main production strategy. To maintain pressure, water injectors were drilled in the water leg, on the west flank of the reservoir and in other select zones

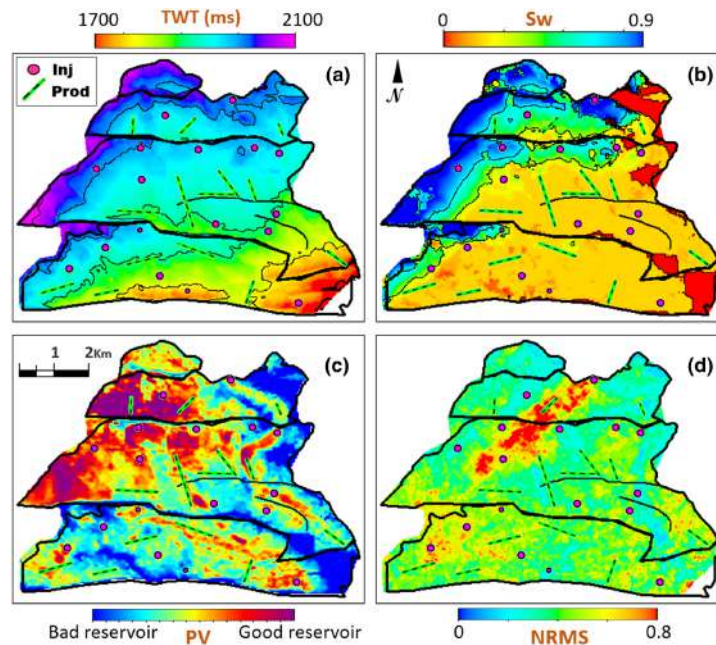


Figure 1 (a) Top reservoir horizon in time, (b) initial water saturation for reservoir sandstone, (c) pore volume for reservoir and (d) NRMS measure of non-repeatability.

around the reservoir. Even so, production in this complex compartmentalized structure led to areas with strong pressurization due to water injection into isolated compartments, while other areas lack the pressure support and experience gas release due to pressure depletion below bubble point pressure. This creates a complex dynamic setup on top of an already complex static framework. The challenge is to use four-dimensional (4D) seismic data to quantitatively estimate simultaneous changes in three dynamic properties: pressure (ΔP), water saturation (ΔS_w) and gas saturation (ΔS_g) across the reservoir. Both pressure and gas effects on seismic data are non-linear, so the inversion method should deal properly with the non-linearities due to changes in these two properties.

The production strategy for this reservoir included regular 4D seismic acquisitions to aid in monitoring reservoir production. In this paper, we present the results of the method applied to one of the many monitor seismic acquisitions acquired along the field life. The reservoir is thin to seismic standards, being identified in a seismic quadrature section as one single

trough (Fig. 2). For this reason, all of the analysis is done in map form. The seismic data used for inversion (Fig. 3) are the time-lapse difference in the sum of negative amplitudes map attribute (ΔSNA), extracted from quadrature seismic volumes along the reservoir time window. This map extraction consists of a vertical sum of the negative seismic amplitudes between the top and bottom reservoir horizons (shown in Fig. 2) for the baseline and monitor volumes, followed by a subtraction of these two maps (monitor – baseline). Calculated time-shifts are very small and show no correlation with the seismic amplitudes or production data, so unfortunately, they were not useful. We use the pre-production seismic acquisition as the baseline for generating the 4D seismic maps.

Figure 1(d) shows the normalized root mean squared (NRMS) map extracted from the monitor-baseline pair used for inversion. This was calculated in a 400 ms time window, 100 ms above the reservoir, so that it is away from any production-related effects, but deep enough not to capture acquisition footprints. NRMS is a measure of comparison between two seismic traces. When extracted in the overburden

2168 G. Côte et al.

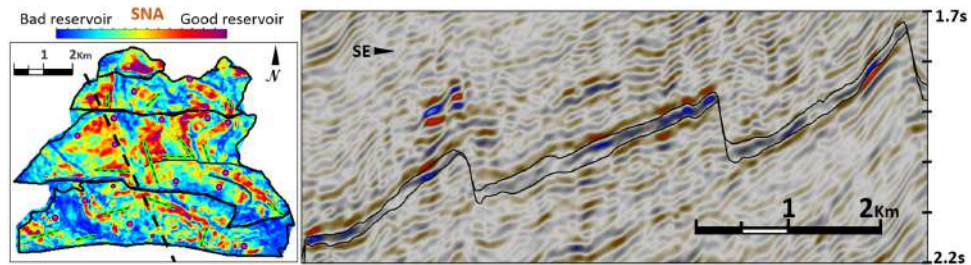


Figure 2 Sum of the negative amplitude (SNA) seismic map attribute for the baseline acquisition (left), extracted from the quadrature full stack seismic 1996 baseline acquisition (right). Seismic section on the right shows the top and bottom reservoir horizons between which seismic attributes are extracted.

region, NRMS maps are interpreted as a measure of non-repeatability between two seismic acquisitions, providing a relative estimation to the 4D seismic data quality across the reservoir. NRMS values range from 0 to 2. Values of 0 represent perfectly correlated traces, 1 is observed for uncorrelated random noise traces while 2 is observed for anti-correlated traces. It is commonly presented as percentage values, with a multiplication of the calculated NRMS values by 100. Calculated NRMS values range from 20% to 40%, except for the undershoot zone, where repeatability is poor due to the presence of a production platform. The poor repeatability in this area has a strong impact on the quality of the signal and, as we will see, in the inversion results for this area.

To aid in interpretations, Fig. 4 shows vertical average maps of the reservoir simulation results. In all maps in Figures 3 and 4, the colour scales are adjusted to represent softening effects as yellow–red and hardening effects in blue–green. Softening effects are defined as those that are related to a

decrease in the bulk reservoir rock seismic impedance, while hardening effects relate to increases in reservoir impedance. For example, increase in water saturation lead to an increase in reservoir impedance because water impedance is higher than the oil impedance. Increases in gas saturation on the other hand lead to decreases in the reservoir impedance because gas' seismic impedance is lower than that of the oil. For this reason, their colour scales are inverted. This scheme simplifies the comparison of reservoir property maps and seismic maps.

In Figs 3 and 4 some areas of interest are circled, showing important features that we will use as guides for a qualitative assessment of the inversion results. This initial interpretation is inherited from previous four-dimensional amplitudes with offset (4D AVO) studies done for this dataset (Côte *et al.*, 2019). Circled areas represent the reservoir property that dominates each seismic anomaly. This depends on the seismic sensitivity to changes in each property and on what other property changes are occurring simultaneously. Seismic

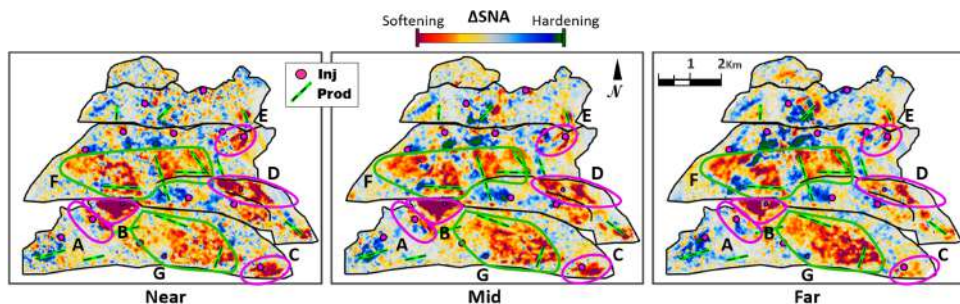


Figure 3 Seismic data used for inversion: time-lapse change in the sum of negative amplitudes attributes extracted from the near, mid and far stack.

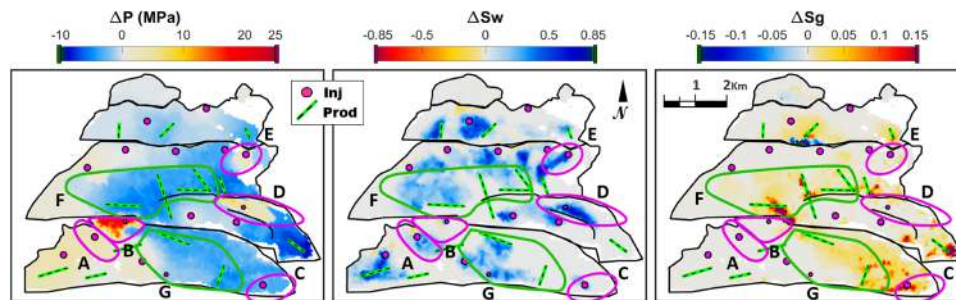


Figure 4 Vertical average maps of the results of the reservoir flow simulation for the changes in pressure (ΔP), water saturation (ΔS_w) and gas saturation (ΔS_g).

amplitudes may be more sensitive to changes in some property over other. If changes in both these properties are superposed, the seismic effects related to one property will overcome and dominate over the other. Uncertainties tend to be larger in the properties that have their effects dominated by co-occurring changes in other properties to which the seismic data are more sensitive.

Zones circled in magenta (A, B, C, D and E) are well-understood softening signals related to pressurization around water injectors. In this case, the hardening signals related to the increase in water saturation are overcome and dominated by the stronger softening signals related to the pressure increase. Zones circled in green (F and G) are well-understood softening signals related to gas saturation increase. In these zones, the gas saturation softening response dominates, but it is always in competition with hardening signals due to water saturation increase. In zone F, the water comes mainly from the aquifer located to the north and west, aided by injectors in the water leg. This zone is particularly complicated because, aside from the water-gas competition, it lies under a platform in an area of low seismic repeatability (Fig. 1d), thus the seismic data here are very noisy and uncertain. The AVO gradient is especially affected, crippling the data capability of differentiating between pressurization-related softening effects and gas saturation related ones. For this reason, inversion results in zone F may show leakage between gas and pressure effects. In zone G, the water comes from the two injectors placed on its southwest edges. We see from the simulation results (Fig. 4) that a considerable amount of water has been injected in this zone, but no hardening signal can be seen in Fig. 3. As seismic data are much more sensitive to increases in gas than in water saturations, the gas-related response dominates.

We see some hardening signals related to water saturation increase in Fig. 3, but no evident hardening signal related to pressure depletion. The non-linear nature of pressure effects on 4D seismic data means that even though pressure increase leads to strong softening signals, pressure depletion results in very low hardening anomalies. To complicate further, in the present case, pressure depletion is always accompanied by gas coming out of solution, so the gas softening effects always dominate over the weak hardening pressure depletion signals. This makes it particularly difficult to quantify pressure depletion values from the 4D seismic data.

The main challenges for this inversion are to quantify the pressure increase values in the pressurized compartments, to differentiate between pressure related and gas related softening signals, to determine areas of pressure depletion and to locate water fronts in areas where water saturation related hardening signals are dominated by other competing effects.

DEEP NEURAL NETWORK ARCHITECTURE AND TRAINING

We employ a deep neural network (DNN) with the encoder-decoder architecture to translate the mapped sum of negative amplitudes (ΔSNA) seismic attributes (Fig. 3) into the corresponding changes in three reservoir properties: ΔP , ΔS_w and ΔS_g . Although we mention maps, this network is not working on full images, instead it makes pixel-wise inferences, analysing each pixel individually and independently of others. In the training phase, it uses all the pixels (or samples) provided in the training dataset to define one general non-linear function that best links the values given in the input domain to the output domain. In the inference phase, it applies this non-linear function individually to each pixel in the map

2170 G. Côte et al.

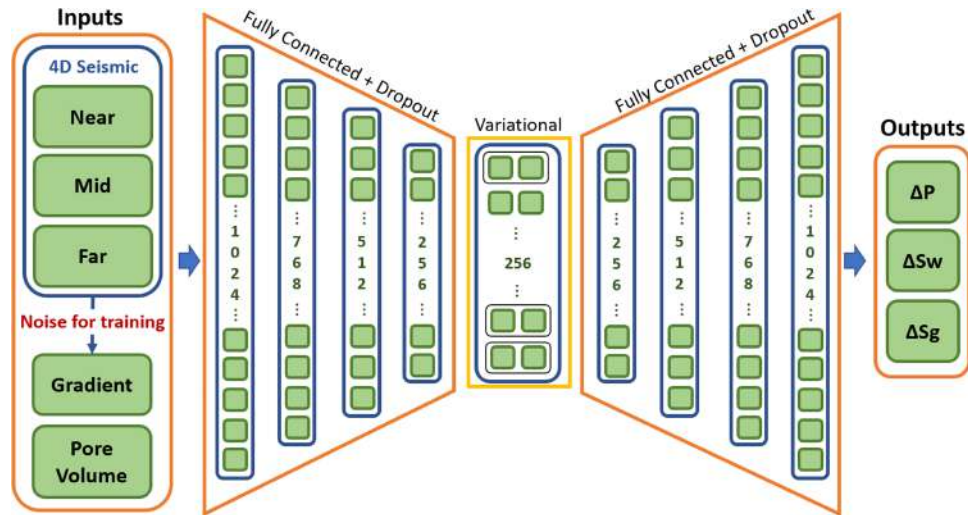


Figure 5 Deep neural network architecture.

provided. Being so, there is no lateral correlation constraint or smoothing technique to ensure lateral correlation and control noise content. For this reason, we employ three important techniques that help to avoid overfitting and inverting noise: dropout regularization (Srivastava *et al.*, 2014), variational Bayes encoding layer (Kingma and Welling, 2014) and training with noisy samples (Bishop, 1995).

The full DNN architecture consists of four encoding layers that compress the input information from 1024 neurons in the first layer to 256 neurons in the last, a central layer where variational encoding is implemented and a mirror decoder structure that decompresses the information back into 1024 neurons (Fig. 5). Compressing the information serves as forcing function for the network to learn the most meaningful features in the data in regard to the optimization objective. In each of the encoder-decoder layers we use a dropout regularization technique in the training phase, this randomly excludes 20% of the neuron connections in each training iteration. Dropout regularization is commonly used as a technique to prevent overfitting of the training data, leading to a more general model and helping in dealing with noisy datasets (Srivastava *et al.*, 2014).

The central encoding layer is arranged so that each neuron defines two outputs which are used to define Gaussian

distributions (mean and standard deviation). In the following layer, each neuron draws one random value from each Gaussian distribution. Consequently, a neuron from the central layer feeds slightly different values to each neuron in the following layer, as opposed to feeding the same value to all neurons as it is done in all other connections in the network. Using variational encoding in the central layer, instead of a fixed link from encoding to decoding, provides a flexibility to the network, making it more general and robust to noise (Kingma and Welling, 2014). To be able to train the network with back propagation of the gradients, we use the 'reparameterization trick' as in Kingma *et al.* (2015). To construct this architecture, we used a Tree of Parzen scheme for estimation of the hyperparameters (Bergstra *et al.*, 2013). This is an optimization scheme that uses a subset of the training data to find the best hyperparameters for the problem at hand. Adjusted hyperparameters were the number of layers and neurons per layer.

The input layer contains the time-lapse difference in the ΔSNA seismic attribute, extracted for the near, mid and far angle-stacks. From these three 4D seismic attributes, the network calculates the four-dimensional amplitudes with offset gradient to be used as an input as well. In the training phase, before calculating the gradient, we add white Gaussian noise to the synthetic data. This step is crucial for achieving meaningful results when making inferences on the noisy seismic

data. Training with noisy synthetics is equivalent to Tikhonov regularization of inversion processes (Bishop, 1995), so it controls overfitting and prevents the DNN from treating the noise as signal. The magnitude of the added noise is controlled by one single parameter, the signal to noise ratio, that defines the standard deviation of the Gaussian distribution from which random noise values are drawn. In a later section of this paper, we present an analysis of the impact of this noise parameter on the inversion results and elaborate on how to assess the performance of the trained models and select an optimal value for this parameter.

In addition to the time-lapse seismic attributes, we also include in the inputs the reservoir pore volume, calculated from the reservoir simulation model. This static parameter is relevant as reservoir pore volume acts as a scalar on the 4D seismic amplitudes, leading to stronger responses in areas with higher reservoir pore volume. We observed that the addition of this static parameter within the network architecture is essential in achieving a more accurate regression result. The pore volume as an input parameter abstracts the information the neural network has to learn from the seismic input maps, alleviating the learning process for the network (Dramsch *et al.*, 2019b).

Supervised training of neural networks relies on an ensemble of samples of known input-output pairs which define the training dataset. We use the Adam optimization method (Kingma and Ba, 2015) for training. It is a stochastic gradient descent optimization approach with Nesterov momentum and an adaptive learning rate. The algorithm iteratively updates the neuron weights that define the network state, in each iteration outputs are calculated from the inputs provided in the training dataset. The algorithm then compares the calculated outputs to the outputs in the training dataset, using a mean squared error objective function. More details on the architecture and training strategies used can be seen on Dramsch *et al.* (2019b).

The training data are presented to the network in the form of $N \times 1$ vectors, with N being the number of samples in the training dataset. Each of the parameters in the input ($\Delta S_{NA_{near}}$, $\Delta S_{NA_{mid}}$, $\Delta S_{NA_{far}}$ and Pore Volume) and output (ΔP , ΔS_w and ΔS_g) are presented as a separate $N \times 1$ vector.

CONSTRUCTION OF THE SYNTHETIC TRAINING DATASETS

Defining the training dataset is one of the most important steps in a neural network workflow. In the present application, the network represents (or replaces) the physics that links the seismic domain to the reservoir domain, but in fact it has no

knowledge or any information about the physics it represents. It learns this implicitly from the training dataset, so the quality of the training dataset will define how well the network will mimic the physics of the problem and its capability of inferring meaningful results from unseen data.

To construct the training dataset, we need to form an ensemble of input-output realizations. These are independent single pixel realizations of ΔP , ΔS_w , ΔS_g and Pore Volume and their resulting sum of negative amplitudes values for near, mid and far stacks. In the lack of a good and sufficient measured dataset, the alternative is to use synthetic data, generated based on a physical model that represents as best as possible the problem at hand.

In this application, we employ a reservoir simulator to seismic modelling (Sim2Seis) technique (Amini, 2014) to create synthetic seismic data from a reservoir flow simulation model. We use one fixed reservoir model that has been previously history matched to production data. The model grid spans the whole reservoir, so it contains a good representation of the variability of the static reservoir properties. We use this static geological model as the frame for creating four different training datasets that differ in the distribution of the sampled realizations in ΔP , ΔS_w , ΔS_g and Pore Volume. For each realization, we extract a pseudo-log from a certain location in the static reservoir model. The pore volume value is calculated from this pseudo-log, as the vertical sum of the pore volumes for active cells in the reservoir zone. Next, we define a sample realization for the ΔP , ΔS_w and ΔS_g values. These values are distributed vertically in all reservoir cells in the extracted pseudo-log, always respecting initial and residual saturation values in each cell. As this is a map-based approach, we do not model different vertical distributions of fluids or pressure in the reservoir. With this approach, we maintain the vertical resolution of the reservoir model on the static properties, but the dynamic properties represent a vertical average. Angle-stacked seismic traces are then calculated for baseline and monitor states using Sim2Seis and from them the sum of negative amplitudes attributes are extracted along the reservoir time window.

The reservoir model cell dimensions are 50×50 m in the lateral dimensions and 3 m vertically. The regular seismic grid separation is 25 m in both inline and crossline directions. For all synthetic seismic calculations, we use one fixed petro-elastic model that was previously calibrated to the well logs and the observed seismic data (Amini and MacBeth, 2015; Amini, 2018). The petro-elastic model is based on a mixture of sand and shale grains following volume fractions given by reservoir net to gross. Rock frame elastic moduli are calculated

2172 G. Côte et al.

Table 1 Training dataset sizes and training times

Training Datasets	Number of Pseudo-Logs	$\Delta P \times \Delta S_w \times \Delta S_g$ Realizations per Pseudo-Log	Total Number of Samples	Average Training Time (minutes)
1	300	1130	339,000	~17
2	300	475	142,500	~7
3	12,944	7	30,608	~4
4	12,944	100	1,294,400	~60

following Nur's critical porosity model (Nur *et al.*, 1998). Pressure dependence follows the compliance model by MacBeth (2004), using for model parameters the values measured for a sandstone reservoir in the west of Shetland islands, analogous to Shiehallion, which are provided in his paper. Effective fluid elastic moduli are calculated using a homogeneous saturation model and Gassmann's fluid substitution equations (Gassmann, 1951) bring all the pieces together to calculate the saturated rock elastic moduli. Seismic traces are calculated by a convolution of a source wavelet with the reflectivity series calculated with the petro-elastic model. We use separate wavelets for each angle-stack, which are extracted individually from the each of the observed angle-stacked seismic volumes. Wavelets are all zero phase with central frequencies of 29 Hz (near), 25 Hz (mid) and 20 Hz (far).

We tested four training datasets with the objective of assessing the impact of data size and the use of physics informed realizations to populate the training dataset. The amount of physics information used to construct the training datasets increases from set 1 to 2 and to 3. For Training dataset 4, data augmentation techniques are used to assess the impact of dataset size on the inversion results, while keeping the similar levels of physics information as in Training dataset 3.

Table 1 shows a comparison of the amount of data, how it is distributed and the resulting training runtimes for all four training datasets presented. Figure 6 shows the global distribution of data in each training dataset.

Regularly sampled realizations (Training datasets 1 and 2)

In neural network applications, it is often desirable to have general trained networks that can be applied satisfactorily to many different cases. In the current application, this would mean training one general network that can be applied to many reservoirs with differing static and dynamic situations. The most general training dataset should contain realizations representing all possible situations, both on the static (Pore Volume) and on the dynamic ($\Delta P \times \Delta S_w \times \Delta S_g$) domains.

Training dataset 1 is constructed in a way as to be the most general. For this, we generate synthetic realizations for every point in a regular four-dimensional sample grid (Pore Volume $\times \Delta P \times \Delta S_w \times \Delta S_g$). The sample grid covers the whole range of possible situations on all four properties. To keep the saturation values realistic, we apply a unity constraint on the sum of the saturations, and always respect residual oil and water saturations, so that as an example, if $\Delta S_w = 0.5$, then ΔS_g is constrained to values between 0 and 0.15. For each Pore Volume value defined in the sample grid, we extract one single pseudo-log from the simulation model that best represents this Pore Volume value. We then proceed to calculating time-lapse seismic traces for all realizations in the dynamic domain. The reservoir simulation model is used here only to define the static frame for calculating synthetic data, but the reservoir simulation results are not used.

The same sampling strategy is used for Training datasets 1 and 2. The only difference between these datasets is one simple constraint used in Training dataset 2. This constraint comes from our external understanding of the physical processes governing the problem at hand. As has been mentioned, in this reservoir, the increase of gas saturation is a response to pressure depletion. As pressure goes below bubble point pressure, gas comes out of solution from the oil phase. Wherever pressure increases from initial pressure, which is above bubble point pressure, we expect no gas saturation change to occur. Thus, training sample realizations containing simultaneous increases in pressure and gas saturation are not representative of the reality analysed and could be interfering negatively in the solutions. For Training dataset 2, we delete all samples with simultaneous increase in pressure and gas saturation, consequently making it less general, more specific to the reservoir situation. The comparison of Training datasets 1 and 2 pinpoints the impact of one simple constraint, showing the benefits that can be achieved by adding one bit of physical information to constrain the realizations in the synthetic training dataset.

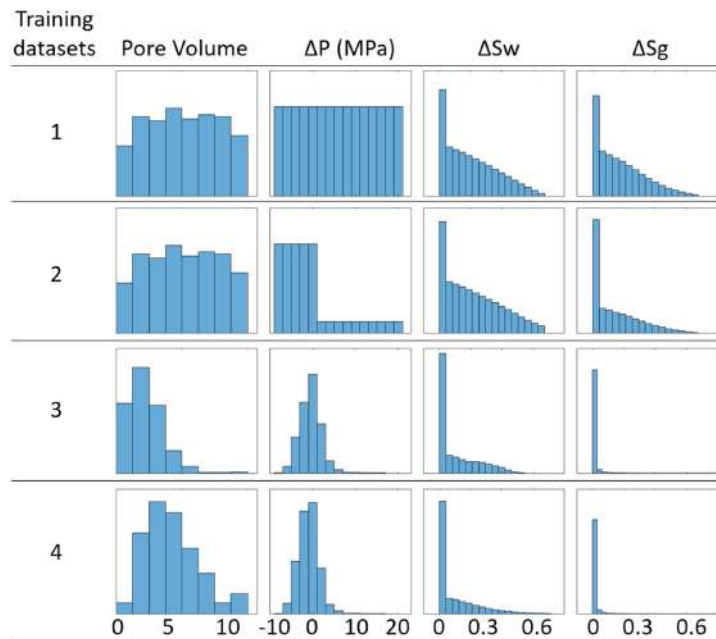


Figure 6 Histograms showing the global distribution of data in Training datasets 1, 2, 3 and 4.

Fluid flow consistent realizations (Training datasets 3 and 4)

A full 3D reservoir flow simulation offers $\Delta P \times \Delta S_w \times \Delta S_g$ realizations that respect a full range of physical processes, including the causal relationship between pressure depletion and gas saturation increase, but also processes related to wettability, capillary forces, relative permeability, etc. Restricting our training samples to the combinations offered by the results of a reservoir simulation model means that the training dataset will respect all of these physical processes and thus be even more realistic and similar to what an unbiased measured dataset would offer. The resulting training datasets are even less general and more specific to the reservoir situation. Reservoir simulation results are used in Training datasets 3 and 4.

For Training dataset 3, we use the reservoir simulation results as the only $\Delta P \times \Delta S_w \times \Delta S_g$ combinations in the training dataset. To maintain consistency with the previous approaches we do not run simulator to seismic modelling directly on the reservoir simulation results, first we extract vertical average maps for ΔP , ΔS_w and ΔS_g at a given time-step,

then we apply the same forward modelling process as previously described (distributing the averaged maps vertically in the extracted pseudo-logs). We extracted simulation results for eight time-steps corresponding to the seismic acquisition dates. This choice on time-steps was done simply because it is common to use a reservoir simulator to seismic modelling workflow at seismic acquisition times to compare synthetic with observed seismic data, so in this case we did not need to extract additional simulation results, all the data were already available and had been generated and used for other previous purposes, this would usually also be true in a regular industry setup. There is no real need to restrict the time-steps to seismic acquisition dates though.

There is a great difference on the distribution of data using this approach (Fig. 6). In this case, we run synthetic traces for every trace location in the simulation model, with this we have 12,944 static pseudo-logs instead of the 300 of the previous models, on the other hand, we only have seven $\Delta P \times \Delta S_w \times \Delta S_g$ realizations per pseudo-log. Thus, pore volume is much more finely sampled, and the global distribution of pore volume is no longer uniform, here it resembles the global

2174 G. Côte et al.

Table 2 Performances and training SNRs of the best models in the synthetic and well validations

Training Datasets	Synthetic Validation					Well Validation	
	Total NMSE	ΔP NMSE	ΔS_w NMSE	ΔS_g NMSE	Training SNR	ΔP NMSE	Training SNR
1	3.08	2.68	2.19	4.37	21.5	6.60	39.5
2	1.56	0.71	1.66	2.31	42	2.50	19
3	0.66	0.62	0.70	0.64	17	0.46	14
4	0.56	0.50	0.73	0.45	11	0.40	12

distribution that could be found in reality. Regarding the dynamic domain sampling, though the simulation results cannot be taken as the real reservoir fluid flow state, the global distributions should resemble reality, as the model has been history matched to well production and injection volumes and pressure measurements. The resulting training dataset is smaller in total number of samples but is much more representative of the reality of the reservoir.

It remains true that DNNs benefit from larger amounts of data. For this reason, in Training dataset 4 we make an effort to augment the previous training dataset, while maintaining all the physical relationships present in the reservoir simulation results. We do this by grouping all the dynamic domain samples ($\Delta P \times \Delta S_w \times \Delta S_g$) in the previous training dataset to create an ensemble of possible samples that respect the reservoir flow physics. For each static pseudo-log extracted from the simulation model, we draw 100 random samples from this global ensemble of dynamic realizations to run time-lapse synthetic seismic traces. In practice, we take the reservoir simulation results found at one trace location and apply it to a different trace location, always respecting end member saturation limits. This approach maintains global distributions in pore volume, ΔP , ΔS_w and ΔS_g that are similar to Training dataset 3 (Fig. 6), while augmenting the data size by 100 times.

MODEL PERFORMANCE QUANTIFICATION

To assess the performance of each trained model, we use two validation approaches. The first one is based on synthetic data, where ground truth is exact, but the seismic data are more well behaved and may not represent the real observed data properly. The second is applied to the observed data itself, using as ground truth for validation the well measurements at seismic acquisition times. In the second approach, the seismic data are a good representation of the real data, but the ground truth is not exact and carries itself some uncertainty. A summary of the performance quantification results for all four training datasets can be found in Table 2.

As has been mentioned, we add random noise to the synthetic training samples in the training phase, to take advantage of the regularization property this technique carries (Bishop, 1995). A deep neural network that is trained on noisy synthetics is more generalized and capable of interpreting the noisy character of the observed data and avoid overfitting this noise. This capability varies with the amplitude of the noise added. If we add too much noise, this corrupts the amplitudes with off-set information in the training phase and inferences are compromised. To assess the impact of the training noise parameter on the inference performance, we train 100 models for each training dataset, with varying training noise levels, ranging from 0 to 50% in noise to signal ratio (SNR). Intuitively, the best performance should be achieved by the model that is trained using the same SNR levels as are present in the inference data. As the final objective is to apply the inversion to the real observed 4D seismic, it is worth estimating the SNR level present in this dataset. For this, we consider the noise power as the root mean square (RMS) of the observed seismic amplitudes in the northernmost segment, where no production has occurred and no 4D seismic signal is expected, and the signal power as the RMS across the whole reservoir. Observed 4D seismic data noise level was estimated to be 14%.

Synthetic validation

In order to make a fair comparison between the performances achieved with the four different training datasets, we created an additional set of synthetic data, which is not used for training in any of the models. We want the validation dataset to represent the real data as well as possible, so that the performances calculated can be interpreted as the capability of a certain model to make correct inferences on the real data. For this reason, the validation dataset is constructed by extracting the reservoir simulation results for one separate time step, which is not used in Training datasets 3 and 4. With this the validation dataset contains 12,944 samples. Then, we add random noise with 14% noise to signal ratio (SNR)

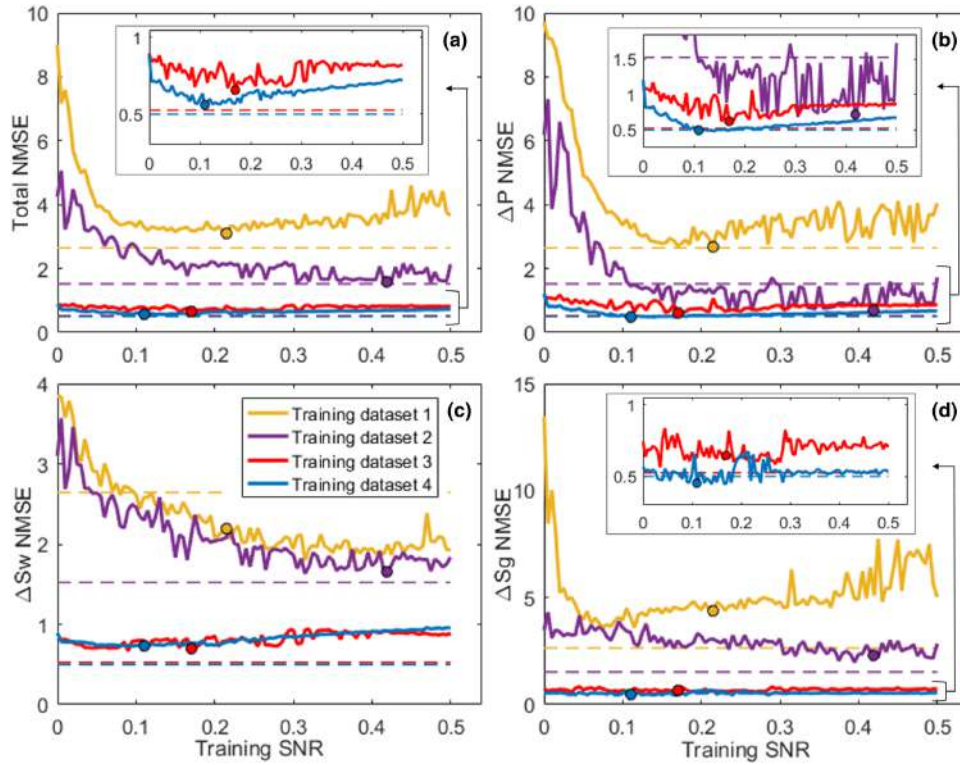


Figure 7 NMSE results for the synthetic validation. (a) Total, (b) ΔP , (c) ΔS_w and (d) ΔS_g .

levels, so that the synthetic validation dataset contains noise levels comparable to the observed 4D seismic. We apply the inference step to the same noisy synthetic validation dataset for all trained models and assess the performance by comparing the inference results to the reservoir property values in the validation dataset. Comparisons are made individually for each reservoir property using the normalized mean squared error (NMSE) metric, these are then individually normalized using the standard deviation of each target reservoir property and averaged to achieve a global performance metric.

Figure 7 shows plots for the performances achieved as a function of the training noise to signal ratio (SNR) for all four training datasets. Plots (b)–(d) show the performances on each property individually and plot (a) shows the global performances for all the trained models. To provide a reference

we also show the performances achieved in the ideal noiseless case (dashed lines), where the models are trained without noise addition and applied to a noiseless version of the synthetic validation dataset. Models that achieved the best total performance for each training dataset are highlighted in Fig. 7 and their results are plotted in map form in Fig. 8.

We observe a global increase in performance (Fig. 7a) with the addition of training noise for all training datasets. Best performances do not achieve the performance values for the ideal noiseless synthetic case (dashed lines) but are all considerably better than when no training noise is added. The addition of training noise is responsible for a decrease of 30–60% in the total normalized mean squared error (NMSE). For Training datasets 1, 3 and 4, the total NMSE reaches a region of lower values with training SNRs around the noise level applied to the validation dataset (14%). Training

2176 G. Côte et al.

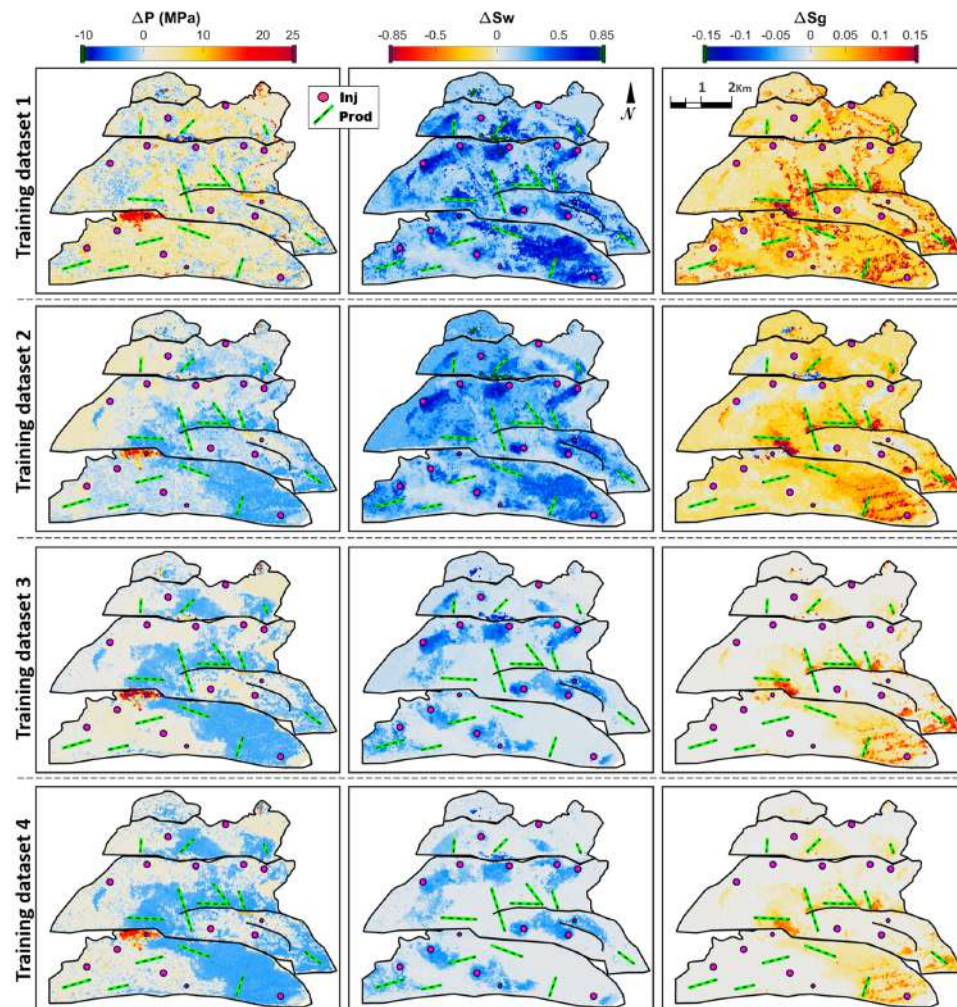


Figure 8 Map results from the best models in the synthetic validation.

dataset 2, on the other hand, shows an increase in performance with training SNRs up to 49%. Intriguingly, for this training dataset, it has been beneficial to train the model with noise levels three times higher than the noise content of the data it was applied to.

In a comparison between the training datasets, we see a global increase in the performances from Training dataset 1 to 2(a). This is mostly due to better estimations of the pressure (b) and the gas saturation (d). Performances for the water saturation estimation are similar for Training datasets 1 and 2.

This shows us that the use of a simple physics-based constraint on the pressure and gas saturation has a positive impact on the estimations of these two properties, while it has little impact on the water saturation estimations. Training datasets 3 and 4 show considerably better performances than the two previous ones, showing how much of a benefit can be achieved by using training datasets populated with fluid flow consistent samples. Training dataset 4 achieves slightly better total performances (a) than Training dataset 3, indicating that the data augmentation has been beneficial to the quality of inferences. Again this is due to a difference in performances for estimating pressure and gas saturation, as water saturation performances are similar. For these two training datasets, the addition of noise has no impact on the performance of the gas saturation estimations.

Well data validation

Although we have made an effort to create a validation dataset that mimics the real data, the performances achieved when applying a model to synthetic data may not represent the truth when the model is applied to the observed four-dimensional seismic data. As has been mentioned, labelled data for the real case are only present as measurements made at wells during seismic acquisition times. For Schiehallion, we only have bottom hole pressure measurements, which is the main reason why we need synthetic data for training in the first place. There is no real data to assess the inference results on the saturation values, but we can nonetheless assess the pressure estimations only. The synthetic validation indicates that total performances are mostly driven by the performance on estimating pressures. This provides some confidence in evaluating the models using well bottom hole pressure measurements only, as the performance on the pressure estimates may be a reasonable representation of the global performances.

When comparing well bottom hole pressures to seismic inverted pressures, it is important to keep in mind that the pressure data measured at wells are not exact ground truth, it carries uncertainties due to mainly two reasons:

- Spatial: In deviated wells, there is uncertainty in locating where along the well perforations the bottom hole pressure measurements should be related to. This is not an uncertainty intrinsic to the well measurements, it becomes relevant only when comparing inverted pressures along the well perforations to the measured values.
- Temporal: A seismic acquisition may take weeks or even months to be finished. Along this time, reservoir pressures

are not constant and bottom hole pressure measurements may vary substantially. There is uncertainty in selecting the ideal time to extract bottom hole pressure measurements for the validation (Omofoma *et al.*, 2019).

Figure 9 shows the normalized root mean squared (NMSE) values achieved by all the trained models. In general, the results corroborate the analysis made in the synthetic validation study. All training datasets present an increase in performance with the addition of training noise and a region of higher performances around 14%, the estimated signal to noise ratio (SNR) for the data they are applied to. In addition to these, Training datasets 1 and 2 show also other models with comparably good performances with higher training SNR values. Training dataset 4 performs only slightly better than Training dataset 3, considering the orders of magnitude higher than computational cost.

Although each trained model offers a deterministic solution, training multiple models with varying SNR values offers the possibility of entering a statistical mind-set. As we have multiple models that present comparable performances, we can create a more general solution by averaging the results of a few best performance models. Additionally, we can use the standard deviation of these solutions as an estimate to the uncertainties in each estimation. Inversion results presented in the next section are all averages of the 10 best solutions, assessed using this well NMSE, out of the 100 trained models. The models used for average and standard deviation quantifications are marked in Figure 9.

INVERSION RESULTS

To provide a visual representation of the benefits of training our deep neural network (DNN) with noise, in Figure 10 we present the inversion results when the DNNs are trained without noise (Training SNR = 0). Figure 11 shows the final inversion results, corresponding to the average between the solutions of the 10 best models selected in the well data validation study. Zero training signal to noise ratio (SNR) results are extremely noisy for Training datasets 1 and 2. The noise content is lower for Training datasets 3 and 4, results for water and gas saturations are similar to the final inversion results but noisier. It is the pressure results that improve most by adding training noise.

In this section, we interpret the final inversion results (Fig. 11) for each training dataset individually. Interpretations are mostly qualitative, using the circled zones as references. Zones circled in magenta are pressure increase areas and zones circled in green are gas accumulation areas.

2178 G. Côte et al.

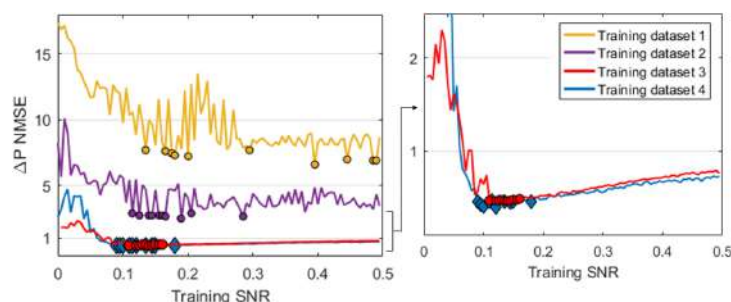


Figure 9 NMSE results for the well data validation.

Training dataset 1

The inversion results for this approach are quite noisy, especially in the gas and the water saturations. This model is incapable of defining well the areas that are impacted by gas saturation, showing very high increases in gas saturation all across the reservoir. Areas dominated by water saturation-related hardening signals are marked reasonably well, but we see a biased background resulting in increases in water saturation across the whole reservoir as well. Furthermore, water saturation values are above the physical limits in many areas, all water saturation values shown in dark green (limit of the colour scale) are above the 85% physical limit. The maximum values for water saturation are of around 1.7, which would mean an increase of 170% of the total porosity in water saturation. Although we can apply constraints to the training data, this does not guarantee that the results will also be constrained, deep neural networks are capable of extrapolating the training data and there is no other way to apply constraints to the inference results. This may lead to unrealistic solutions if the training dataset is inappropriate.

In the pressure results we see well marked pressure increase in all the zones circled in magenta, which is what is expected, but quantitatively the values are far from what is measured at the wells. These areas are where the pressure effects dominate the seismic signal; everywhere else the pressure results are noisy and present an apparent bias towards pressure increase. We also see some leakage from gas saturation-related softening signals into pressure results in zone F. As has been mentioned, in this case, the seismic data are nearly insensitive to pressure depletion. Furthermore, pressure depletion is accompanied by gas breakout, and the seismic data are very sensitive to gas saturation increases. When these two effects are superposed, the much stronger gas effects dominate so that

pressure depletion effects are comparable to noise. The seismic data do not offer any considerable information on pressure depletion (Côte *et al.*, 2019), so it is comprehensible that we see no pressure depletion in the pressure estimations, aside from noise.

Training dataset 2

The use of the constraint in the training dataset has a positive impact on the estimations of changes in pressure and gas saturation (Fig. 11). Pressure results are less biased and leakage in zone F is better controlled. Gas saturation results now mark reasonably well the areas of gas saturation dominance (zones G and F), but we still see a biased background showing gas saturation increases of around 7% across the whole reservoir. Water saturation results are nearly not affected by the constraint, estimations remain biased and values are above the physical limits in many areas.

Interestingly, we now see pressure depletion values in correlation with gas saturation increase in zone G. It is unlikely that these results are based on information from the seismic data so it could be regarded as noise, but as we will see clearly in the results of Training dataset 3 and 4, this is an indication that the network is learning an additional level of information from the training dataset.

Training dataset 3

We can see that having a better representation of reality in the training dataset samples pays off in the quality of the results (Fig. 11). For Training dataset 3 we have much clearer results, noise is contained for all three dynamic properties and we no longer see the general bias present in the water and gas saturation estimations for the previous models, instead, we have

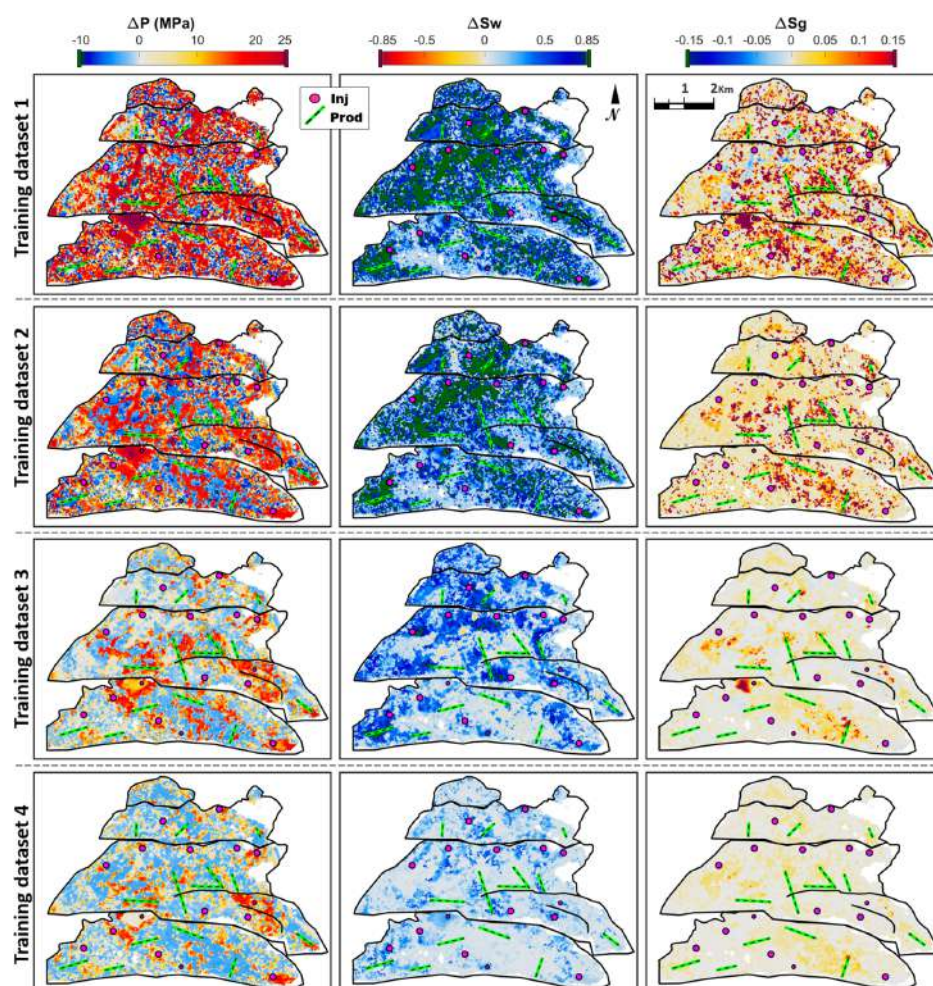


Figure 10 DNN inversion results using training SNR = 0, for training datasets 1, 2, 3 and 4.

a general background of zero values in the areas where there is no seismic evidence of change in each dynamic property. This leads to more contained water estimations forming well-defined bodies that can be connected to the injection wells. Quantitatively, water and gas saturation values are more realistic, falling below the physically possible threshold, except for the water anomalies in the undershoot zone, where the seismic

data are corrupted by low repeatability issues (Fig. 1d). Water front estimations are compromised in some areas where water effects on seismic are dominated by pressure (zones C and E) or gas saturation (zones F and G) effects. On the southwest edge of zone G, there are two injectors that inject a considerable amount of water (as can be seen in the reservoir simulation results in Fig. 4), but we have no seismic evidence of where

2180 G. Côte et al.

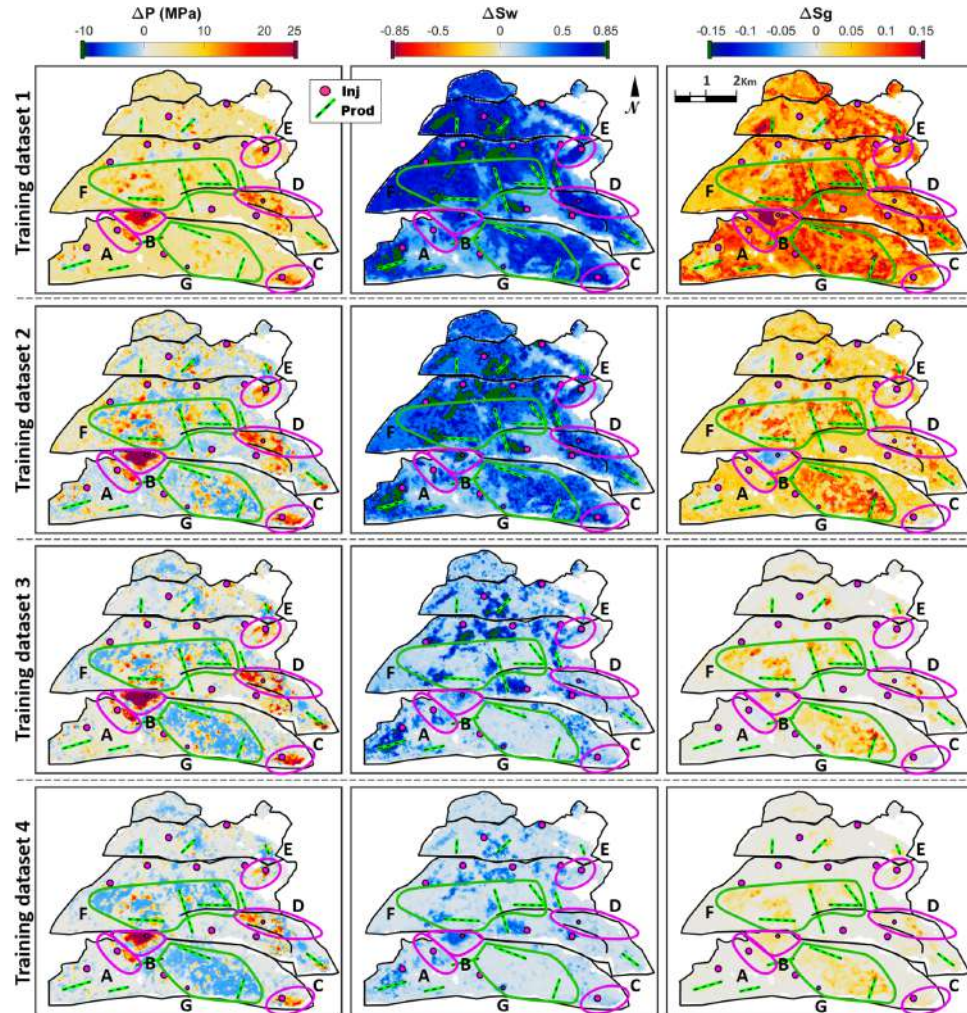


Figure 11 Final DNN inversion results for training datasets 1, 2, 3 and 4. Maps represent the average between the 10 best solutions. Zones circled in magenta are pressure increase areas and zones circled in green are gas accumulation areas.

this water has gone in the inversion results, because the water effects are completely obscured by the gas accumulation here. In zone E, pressure increase effects dominate the seismic response, here the water saturation results are also inconsistent with the amount of water that has been injected into this zone.

Pressure increase is seen in all areas where it dominates the seismic response (magenta circles), furthermore, the quantitative values are more in line with the pressures measured at the wells. Outside these zones we have a global trend of zero values and, interestingly, we see pressure depletion values in

correlation with gas saturation increase (zones F and G). It is clear that this information is not coming from the seismic data. If any seismic information related to pressure depletion could be detected, it would be away from those gas accumulation zones, where the pressure effect is not so overwhelmingly dominated. Instead, the inversion results show pressure depletion values only in direct correlation with the gas saturation increase. This pressure depletion – gas saturation increase correlation is present in the training dataset, as gas exsolution is one of the physical processes modelled by the reservoir simulation. This is a clear indication that the deep neural network (DNN) is learning the correlations present in the training dataset and using this information to make inferences on the dominated properties, where the seismic data cannot provide useful information. We also see a curious gas saturation decrease accompanying the pressure increase in zone B. As there is no initial gas in the reservoir, gas saturation decrease is not present in the training datasets. It does represent a possible reality between monitors, but as we use a pre-production baseline, negative changes in gas saturations are not realistic. We see this result as an indication that the DNN is not only learning the correlation between pressure depletion and gas saturation increase, but also extrapolating this correlation to result in gas saturation decrease in pressure increase zones. This extrapolation is undesirable, as it leads to non-physical results, but it cannot be prevented.

Knowing that the DNN learns not only relations that link input to output but also the correlations between variables in the training dataset emphasizes the argument that the dynamic domain sampling used for creating the training dataset should be retained to physically realistic combinations, because the DNN will embed this physical knowledge into the non-linear transformations it learns. It is also important to note that the data size of this training dataset is much smaller than in the previous models, which means training runtimes are also much faster (Table 1). In this case, it is not necessarily the size of the dataset that matters most, instead the ability of this dataset to represent specifically the global reality of the problem is more important.

Training dataset 4

Augmenting the data size in this approach affected the results only slightly (Fig. 11), but nonetheless the results are generally more consistent. In the pressure results, we see less noise and leakage in the areas dominated by gas (zones F and G). Water saturation results are now below the physical limit everywhere in the map, though the shape of all the water bodies

is very similar to the previous case and the definition of water-fronts does not improve in areas of overwhelming dominance by other properties (zones B, C, D, F and G). Gas saturation results are similar but generally smaller than with Training dataset 3 and zone B no longer shows unrealistic gas saturation decrease values. Given the uncertainty in the estimations, the uncertainty in the reservoir simulation results in this property and the lack of measured saturation logs, it is impossible to define which gas saturation solution is more precise.

Globally the results using Training dataset 4 are slightly better than in model 3, but this comes at a high computational cost. In Table 1, we see that Training dataset 3 contains the smallest data size and quickest runtimes and nonetheless it performs much better than Training datasets 1 and 2, and nearly as well as Training dataset 4, which takes 15 times longer to train. It is also relevant to consider the time it takes to compute the synthetic seismic data to build these training datasets, which is around 100 times longer in Training dataset 4 than in 3.

From this analysis, we see the critical importance in constraining the realizations of our synthetic training dataset to realistic physics informed and fluid flow consistent combinations that represent the specific problem at hand. This constraining will make the model less general, so it should not be applied to a different case that may not contain the same constraint assumptions (e.g. gas injection, gas caps and reservoir compaction), instead it will be more specialized to provide the best results to one specific case.

UNCERTAINTIES

The presented deep neural network workflow is essentially a deterministic solution, but as discussed, we could produce multiple slightly different but equally viable solutions varying the signal to noise ratio (SNR) parameter for training. We can use these multiple solutions to create a simple estimation to uncertainties in the results. Uncertainties here represent the instability in the solutions with varying SNR values. Figure 12 shows maps of the standard deviations of the selected 10 best solutions for all four training datasets.

For Training dataset 1, uncertainty results offer little useful information. From Training dataset 1 to 4 gradually the uncertainties decrease globally, but some patches stand out with high uncertainty values. For the water saturation, uncertainties tend to be higher in the pressure increase zones, and around the high normalized root mean squared (NRMS) zone, which is represented well in the uncertainty results. On the other hand, uncertainty results do not represent well the

2182 G. Côte et al.

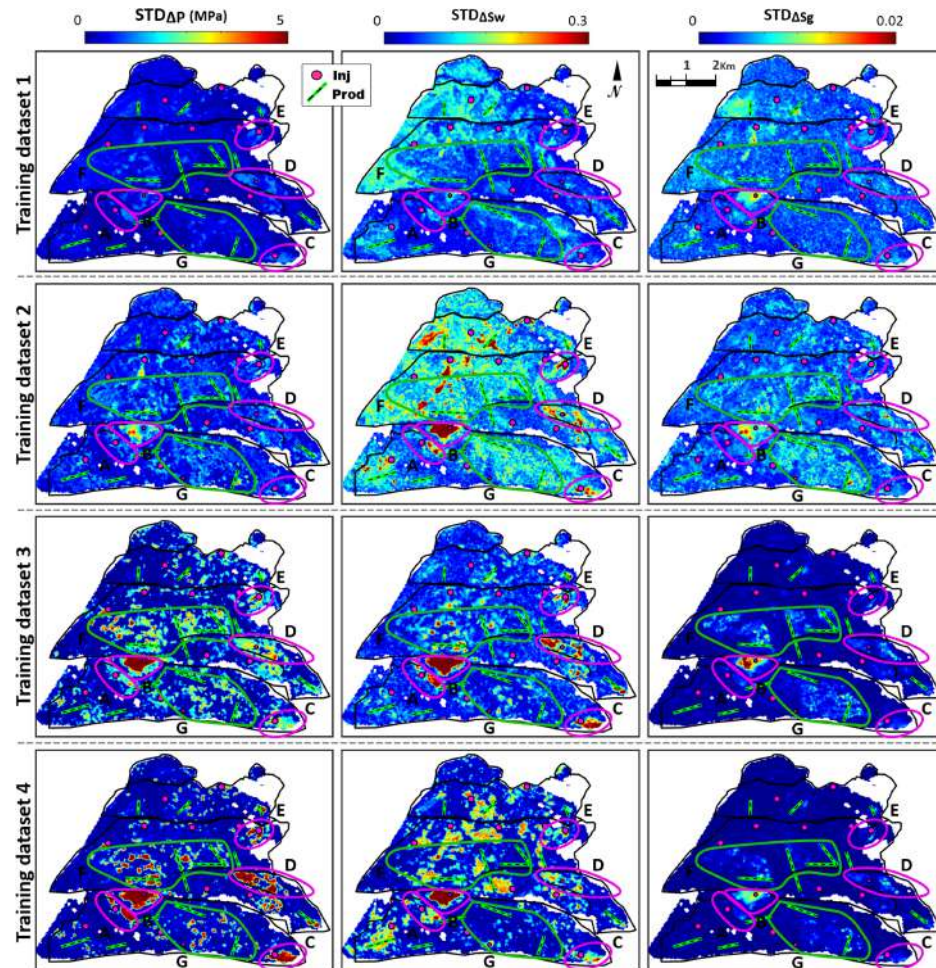


Figure 12 Uncertainty estimations for the four training datasets.

uncertainties expected in areas where water saturation effects are dominated by gas saturation effects (Zones D and E). Uncertainty results for pressure seem to show a direct correlation with the pressure values themselves. This is the opposite of what is expected, as seismic data become more sensitive to pressure as pressure increases. It is unclear what value can be brought from these types of uncertainty estimations, as they do not represent seismic or modelling uncertainties. One thing

does stand out though. Uncertainty values for all three properties highlight zone B as a high uncertainty area.

Zone B is an isolated compartment that was pressurized due to water injection. The injector well was online for around three years and measured pressure increases as high as 20 MPa in 2004, when the seismic acquisition was shot. A previous feasibility study in the area indicated that in order to reach the observed seismic amplitudes, the pressures would need to

be above the estimated rock fracture limit. There are many assumptions in the current petro-elastic model that make it unfit for representing the elastic behaviour of a fractured rock. If the rock has indeed been fractured due to high injection pressures, this means the petro-elastic model used in the creation of the training dataset cannot represent well the rock physics of this zone. Thus, in zone B we have an example of the use of an inadequate training dataset, constructed with an improper rock physics model. This results in solutions that are more unstable with small changes in the noise parameter, which reflects as high uncertainty values. So the uncertainty estimations here were useful for identifying a zone where the synthetic training data are unfit and thus inversion results are not trustworthy.

CONCLUSIONS

The present study has shown that deep neural networks (DNNs) trained exclusively using synthetic data can provide good solutions to the problem of inverting time-lapse seismic data to the simultaneous changes in pressure, water saturation and gas saturation. We show clear evidence of the benefit of adding noise to the synthetic data in the training phase to achieve less noisy and more accurate estimations. Training multiple models with varying training signal to noise ratio (SNR) values and assessing model performance using well measured data offers a possibility of selecting multiple equally probable solutions to create a more general average result. This may also lead to an estimation of the related uncertainties, by calculating the standard deviation of the solutions provided by the selected models. Uncertainty estimations here represent instability in the solutions with respect to the noise parameter and may indicate areas where the training dataset is inadequate.

We show the critical importance of using physics informed sampling of the dynamic domain in creating the training dataset, illustrated by the gradual increase in inversion quality from Training datasets 1 to 3. From datasets 1 to 2, we see that adding a simple external physical knowledge to constrain the samples has a positive impact on the solutions. Using the results of a three-dimensional reservoir flow simulation as the only samples in the dynamic domain (Training datasets 3 and 4) guarantees that the training dataset respects all physical processes modelled by the reservoir simulation. Training datasets then contain physical boundary constraints, physical correlations between dynamic domain parameters and global distributions that resemble the reality of the reservoir. This removes bias in the results and prevents the model from extrapolating beyond the training dataset and leading to extreme un-

realistic results. Solutions also become more stable, less noisy and more precise.

Additionally, we observe that when trained using fluid flow consistent data, the DNN learns not only the relations that link input to output, but also the correlations present in the training dataset, making use of those correlations to make inferences on dominated properties, where the seismic data lack information. This is observed in this case in the correlations between pressure depletion and gas saturation increase. Learning these correlations allows the DNN to resolve some ambiguity present in the seismic data, resulting in better solutions both in the pressure and in the gas saturation results.


We show that, in the present application, an ideal training dataset is one that resembles the most what an unbiased measured dataset would be, both in the sense of containing all the physical correlations in the dynamic domain and also maintaining a realistic global distribution on all related properties. This makes the DNN model less generalized, more specific to the problem at hand, so it should not be applied to describe situations that do not respect the constraints used in the training dataset.

Augmenting the data size while maintaining physical constraints achieved slightly better solutions, improving noise content and leakage in the results. This comes at a high computational cost though. In this application, it seems more important to constrain the sample realizations to physically informed and fluid flow consistent combinations than to chase large data sizes.




ACKNOWLEDGEMENTS

We thank the sponsors of the Edinburgh Time-Lapse Project, Phase VII (AkerBP, BP, CGG, Chevron, ConocoPhillips, ENI, ExxonMobil, Hess, Landmark, Maersk, Nexen, Norsar, OMV, PGS, Petrobras, Shell, Equinor, Woodside and Taqa) for supporting this research and Schlumberger for providing Eclipse software. This work was financed in part by the Brazilian National Research Council, CNPq (200014/2016-1) and the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding program. This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF). We thank Linda Hodgson and Ross Walder for informative discussions. The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Gustavo Côte  <https://orcid.org/0000-0002-5063-2036>

2184 G. Côte et al.

Jesper Dramsch  <https://orcid.org/0000-0001-8273-905X>
Hamed Amini  <https://orcid.org/0000-0001-9588-6374>
Colin MacBeth  <https://orcid.org/0000-0001-8593-3456>

REFERENCES

- Aki, K. and Richards, P. (1980) *Quantitative Seismology: Theory and Methods VI*. New York, NY: W.H. Freeman & Co.
- Alvarez, E. and Macbeth, C. (2014) An insightful parametrization for the flatlander's interpretation of time-lapsed seismic data. *Geophysical Prospecting*, 62, 75–96.
- Amini, H. (2014) *A pragmatic approach to simulator-to-seismic modelling for 4D seismic interpretation*. PhD Thesis, Heriot-Watt University.
- Amini, H. (2018) Comparison of Xu-White, simplified Xu-White (Keys and Xu) and Nur's critical porosity in shaley sands. 80th EAGE Annual meeting, Copenhagen, Denmark, Extended Abstracts, We A11 08.
- Amini, H. and MacBeth, C. (2015) Calibration of rock stress-sensitivity using 4D seismic data. 77th EAGE Annual meeting, Madrid, Spain, Extended Abstracts, We A11 08.
- Ayzenberg, M. and Liu, S. (2014) Saturation and pressure inversion - from 4D seismic to reservoir model updating. 76th EAGE Annual meeting, Amsterdam, Netherlands, Extended Abstracts, We G102 16.
- Bergstra, J., Yamins, D. and Cox D.D. (2013) Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. *30th International Conference on Machine Learning, ICML, Atlanta, Georgia, USA*, 115–123.
- Bishop, C.M. (1995) Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7, 108–116.
- Blanchard, T.D. (2012) Inherent uncertainties in 4D AVO and the implications on pressure and saturation inversion. 74th EAGE Annual meeting, Copenhagen, Denmark, Extended Abstracts, Y020.
- Blanchard, T.D. and Thore, P. (2008) Introducing prior information to pressure and saturation inversion: the key for success? 83rd SEG Annual meeting, Houston, Texas, USA, Extended Abstracts, 4971–4975.
- Cao, J. and Roy, B. (2017) Time-lapse reservoir property change estimation from seismic using machine learning. *The Leading Edge*, 36, 234–238.
- Coleou, T., Roustiau, A., Machecler, I., Ayzenberg, M., Fayemendy, C., Skjei, N., et al. (2013) 4D Petrophysical seismic inversion - case studies. 75th EAGE Annual meeting, London, UK, Extended Abstracts, We 17 03.
- Côte, G., MacBeth, C. and Amini, H. (2019) North Sea field application of 4D Bayesian inversion to pressure and saturation changes. 81st EAGE Annual meeting, London, UK, Extended Abstracts, Tu_P04_09.
- Corzo, M., Macbeth, C. and Barkved, O. (2013) Estimation of pore-pressure change in a compacting reservoir from time-lapse seismic data. *Geophysical Prospecting*, 61, 1022–1034.
- Davolio, A., Maschio, C. and Schiozer, D. (2013) A methodology to constrain pressure and saturation estimation from 4D seismic using multiple simulation models and observed data. *Journal of Petroleum Science and Engineering*, 105, 51–61.
- Dramsch, J., Corte, G., Amini, H., Lüthje, M. and MacBeth, C. (2019a) Deep learning application for 4D pressure saturation inversion compared to Bayesian inversion on North Sea Data. Second EAGE Workshop Practical Reservoir Monitoring, We PRM 11.
- Dramsch, J., Corte, G., Amini, H., MacBeth, C. and Lüthje, M. (2019b) Including physics in deep learning - an example from 4D seismic pressure saturation inversion. 81st EAGE Annual meeting, London, UK, Extended Abstracts, WS10_05.
- Ebdon, C.C., Granger, P.J., Johnson, H.D. and Evans, A.M. (1995) Early Tertiary evolution and sequence stratigraphy of the Faeroe-Shetland basin: implications for hydrocarbon prospectivity. *Geological Society Special Publication*, 90, 51–69.
- Florichich, M. (2006) *An engineering-consistent approach for pressure and saturation estimation from time-lapse seismic data*. PhD Thesis, Heriot-Watt University.
- Gassmann, F. (1951) Über die Elastizität poröser Medien. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 96, 1–23.
- Kingma, D.P. and Ba, J. (2015) Adam: a method for stochastic optimization. *International Conference for Learning Representations, San Diego, California, USA*.
- Kingma, D.P. and Welling, M. (2014). Auto-encoding variational Bayes. *Proceedings of International Conference for Learning Representations, Banff, Canada*.
- Kingma, D.P., Salimans, T. and Welling, M. (2015) Variational dropout and the local reparameterization trick. *Advances in Neural Information Processing Systems*, 28, 2575–2583.
- Lamers, E. and Carmichael, S.M.M. (1999) The Paleocene deepwater sandstone play West of Shetland. *Petroleum Geology Conference series*, 5, 645–659.
- Landrø, M. (2001) Discrimination between pressure and fluid saturation changes from time lapse seismic data. *Geophysics*, 66, 836–844.
- MacBeth, C. (2004) A classification for the pressure-sensitivity properties of a sandstone rock frame. *Geophysics*, 69, 497–510.
- MacBeth, C., Florichich, M. and Soldo, J. (2006) Going quantitative with 4D seismic analysis. *Geophysical Prospecting*, 54, 303–317.
- Nur, A., Mavko, G., Dvorkin, J. and Galmudi, D. (1998) Critical porosity: a key to relating physical properties to porosity in rocks. *The Leading Edge*, 17, 357–362.
- Omofofoma, V. (2017) *The quantification of pressure and saturation changes in clastic reservoirs using 4D seismic data*. PhD Thesis, Heriot-Watt University.
- Omofofoma, V., MacBeth, C. and Amini, H. (2019) Intra-survey reservoir fluctuations – implications for quantitative 4D seismic analysis. *Geophysical Prospecting*, 67, 282–297.
- Smith, G.C. and Gidlow, P.M. (1987) Weighted stacking for rock property estimation and detection of gas. *Geophysical Prospecting*, 35, 993–1014.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 1929–1958.

Deep neural network 4D seismic inversion 2185

- Thore, P. and Hubans, C. (2012) 4D seismic-to-well tying, a key step towards 4D inversion. *Geophysics*, 77(6), R227–R238.
- Trani, M., Arts, R., Leeuwenburgh, O. and Brouwer, J. (2011) Estimation of changes in saturation and pressure from 4D seismic AVO and time-shift analysis. *Geophysics*, 76(2), C1–C17.
- Wong, M. Y. (2017). *Pressure and saturation estimation from prn time-lapse seismic data for a compacting reservoir*. PhD Thesis, Heriot-Watt University.
- Xue, Y., Araujo, M., Wang, K., Lopez, J., Kumar, G. and Brew, G. (2019) Machine Learning to reduce cycle time for time-lapse seismic data assimilation into reservoir management. 81st EAGE Annual meeting, London, UK, Extended Abstracts, We_R07_08.
- Zhong, Z., Sun, A.Y. and Wu, X. (2020) Inversion of time-lapse seismic reservoir monitoring data using cycleGAN: a deep learning-based approach for estimating dynamic reservoir property changes. *Journal of Geophysical Research: Solid Earth*, 125, e2019JB018408.

APPENDIX C

Conference Papers

C.1 Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks

Abstract: Traditional physics-based approaches to infer sub-surface properties such as full-waveform inversion or reflectivity inversion are time-consuming and computationally expensive. We present a deep-learning technique that eliminates the need for these computationally complex methods by posing the problem as one of domain transfer. Our solution is based on a deep convolutional generative adversarial network and dramatically reduces computation time. Training based on two different types of synthetic data produced a neural net that generates realistic velocity models when applied to a real data set. The system’s ability to generalize means it is robust against the inherent occurrence of velocity errors and artifacts in both training and test datasets.

L. Mosser, W. Kimman, J. S. Drams, S. Purves, A. De la Fuente Briceño, and G. Ganssle (June 2018c). “Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks”. In: *80th EAGE Conference and Exhibition 2018*. Published, Appendix C. EAGE. DOI: 10.3997/2214-4609.201800734. URL: <https://doi.org/10.3997/2214-4609.201800734>

Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks

Lukas Mosser

Department of Earth Science and Engineering
Imperial College London
lukas.mosser15@imperial.ac.uk

Wouter Kimman

Open Energi Ltd.
wouter.kimman@gmail.com

Jesper Dramsch

Technical University of Denmark
jesper@dramsch.net

Steve Purves

Euclidity SL
steve@euclidity.com

Alfredo De la Fuente

Skolkovo Institute of Science and Technology
alfredo.delafuente@skoltech.ru

Graham Ganssle

Expero Inc.
graham.ganssle@experoinc.com

Abstract

Traditional physics-based approaches to infer sub-surface properties such as full-waveform inversion or reflectivity inversion are time consuming and computationally expensive. We present a deep-learning technique that eliminates the need for these computationally complex methods by posing the problem as one of domain transfer. Our solution is based on a deep convolutional generative adversarial network and dramatically reduces computation time. Training based on two different types of synthetic data produced a neural network that generates realistic velocity models when applied to a real data set. The system's ability to generalize means it is robust against the inherent occurrence of velocity errors and artifacts in both training and test datasets.

1 Introduction

The task of inferring subsurface geological structures from depth-domain seismic data is a computationally demanding process that frequently appears in geophysical studies and hydrocarbon exploration. Typically, seismic inversion is performed by means of wave inversion methods of a simple prior model of the subsurface and using a backpropagation loop (Lailly et al., 1983; Tarantola, 1984) to iteratively reduce the mismatch between the observed seismic data and the computed synthetic model (Pratt et al., 1998; Virieux and Operto, 2009). Although this approach leads to satisfactory results in practice, it requires an overwhelming amount of computer resources with no guarantee of global convergence; making it inappropriate when time and computing constraints are strict or when we need to perform the same task for a number of geological scenarios.

As an alternative, we propose a data-driven approach that uses deep generative neural networks to formulate the seismic forward and inversion process as a domain transfer problem, which allows us to learn two functions from the datasets: 1) a function to map from the seismic geo-model to the seismic amplitude domain 2) a function to map from seismic amplitude to the geo-model domain. One of the main advantages of this approach comes from the fact that the training step of the algorithm does not require a set of paired input-output images in the dataset.

We present examples of the resulting forward and inverted datasets using the domain transfer method, based on simple synthetic structural models, as well as the Marmousi 2D dataset. Finally, we highlight challenges and possible applications of the proposed approach.

2 Theory

Texture transfer or neural style transfer is an area of research in computer vision. Gatys et al. (2016) used an iterative process to transfer camera photographs into a desired artistic style. They showed results that extracted the features of pre-trained VGG networks (Simonyan and Zisserman, 2014) to model the desired output. This is a computationally expensive iterative process. However, Johnson et al. (2016) specialise a single network per textural style, removing the need to solve an iterative minimisation problem. Isola et al. (2016) reframed the problem in the sense of a domain transfer problem. Here a generative model could be built that transfers the original data to the artistic style. Particularly, a generative adversarial network (GAN) was used with pair-wise corresponding images. Zhu et al. (2017) loosened the constraint on pair-wise training data in a cycle-consistent GAN that learned transfer function between domains. Seismic inversion is an expensive iterative task similar to the computer vision problem discussed here. We use neural style transfer to find a transfer function from seismic amplitude data to velocity functions. We show that this process can benefit from the advancements in deep learning and computer vision.

Deep convolutional generative adversarial networks (DCGAN) consist of two powerful neural networks that learn by competition (Goodfellow et al., 2014; Radford et al., 2015). The generator network \mathbf{G} draws samples from a noise prior or so-called latent space. The generated output is presented to the discriminator network \mathbf{D} in a randomised switch with real data. The discriminator determines whether the output is generated by \mathbf{G} or real. A loss function determines the rate at which both networks learn. In this case, \mathbf{G} gets better at generating realistic outputs and \mathbf{D} improves the ability to evaluate the realism of inputs. In a cycle-consistent setup, we train two GANs in parallel. The generator \mathbf{G} learns the forward generative model. The second generator network \mathbf{F} learns the inverse generative model. The GANs are set up to perform a full circle in the calculation. Input from domain \mathbf{X} is mapped to domain \mathbf{Y} by generator \mathbf{G} , then generator \mathbf{F} maps the result from domain \mathbf{Y} to domain \mathbf{X} . Ideally, the output of the cycle resembles the input so that $\mathbf{F}(\mathbf{G}(x_i)) \approx x_i$.

Both networks \mathbf{G} and \mathbf{F} are subject to an adversarial loss objective. The adversarial loss from the network of Zhu et al. (2017) is defined as:

$$\mathcal{L}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)} [\log D_Y(y)] + \mathbb{E}_{y \sim p_{data}(x)} [\log(1 - D_Y(G(x)))] \quad (1)$$

with the second adversarial loss being equivalent as $\mathcal{L}(F, D_X, Y, X)$.

The cycle of the two GANs has to be consistent in the forward pass $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ as well as the backward pass $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$. Zhu et al. (2017) formalise the cycle consistency loss as follows:

$$\mathcal{L}(G, F) = \mathbb{E}_{x \sim p_{data}(x)} [\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)} [\|G(F(y)) - y\|_1] \quad (2)$$

The combined objective function is simply:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \cdot \mathcal{L}_{cyc}(G, F), \quad (3)$$

where λ is a tuning parameter to weight the relative importance of the networks \mathbf{G} and \mathbf{F} . Enforcing cycle consistency ensures that the data produced by the networks is statistically representative of and bounded by the training dataset, a property that makes the architecture suitable for use in seismic inversion.

3 Convolutional Synthetic Seismic Data

Initially, the network has been tested on geological models with a variety of features. We use a geological modelling package to generate realistic model data with multiple layers with varying velocity/impedance and thickness, folding, faulting and dyke intrusions. The synthetic seismic was generated by convolving the associated reflectivity with a Ricker wavelet.

Figure 1 shows the input, result and reference for the generative networks \mathbf{G} and \mathbf{F} . The first row shows the forward pass from the model domain to seismic domain. A comparison with reference data shows a good match in both the structure and amplitudes.

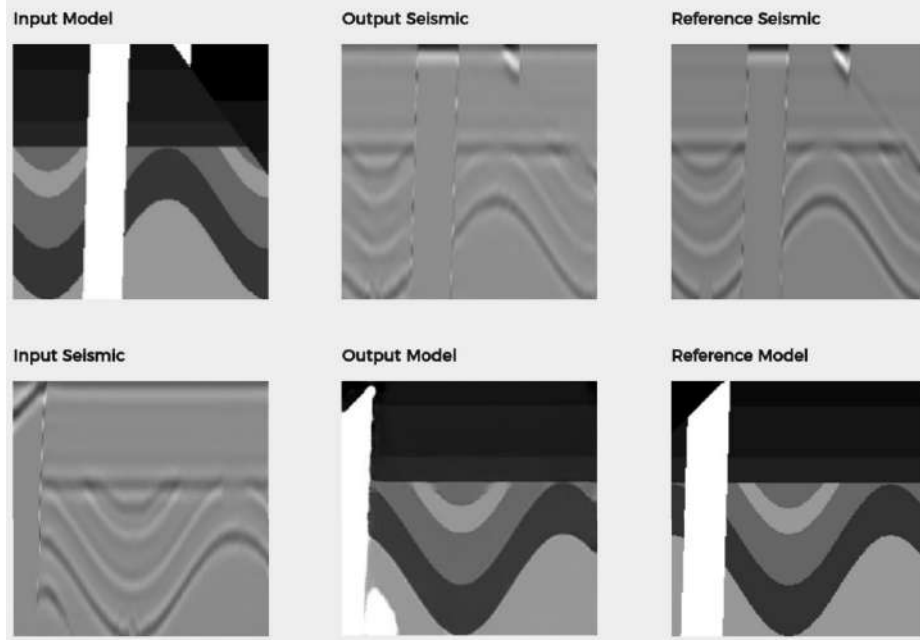


Figure 1: We evaluate the performance of the forward model \mathbf{G} and inverse operator \mathbf{F} based on unseen data of a dyke-anticline training dataset. The first row shows the forward pass through network \mathbf{G} from the velocity model to the seismic domain. Row 2 shows the forward pass of network \mathbf{F} from the seismic to the model domain. In both cases excellent agreement can be found with the reference images (Column 3).

4 Marmousi2 Synthetic Seismic Data

True seismic with its associated velocity errors and noise-related artifacts presents a much bigger challenge compared to the first example. To demonstrate the robustness of the method, we train on synthetic pre-stack Kirchhoff depth migrated seismic of the (elastic) Marmousi2 dataset (Martin, 2004). The data patches extracted from the data show much greater variability and less bias towards high velocity dykes that were prevalent in the convolutional synthetic seismic.

The training was further improved by two pre-processing steps: contrast-enhancement and histogram equalization. The 2D patches we extract from the model, processed and fed to the network are shown in figure 2.

5 Results

We test the improved network \mathbf{F} by taking real data ("Dutch F3", left panel figure 3) as input, that the network has never seen before. The section suffers from migration artifacts (bottom left), and occasional non-continuous reflectors, often a problem for computer vision algorithms. Low contrast regions on top of a high contrast region shows different internal structures and geometries that the network likely has not seen before.

Figure 3 shows the result of the mapping process of network \mathbf{F} . The run-time of this seismic inversion process (the network's operation (\mathbf{F})) is in the order of seconds (GPU time). High contrast areas from the seismic have been identified accordingly. The generated model shows large velocity contrasts where strong reflections occur and changes more smoothly otherwise. The fault is preserved in the

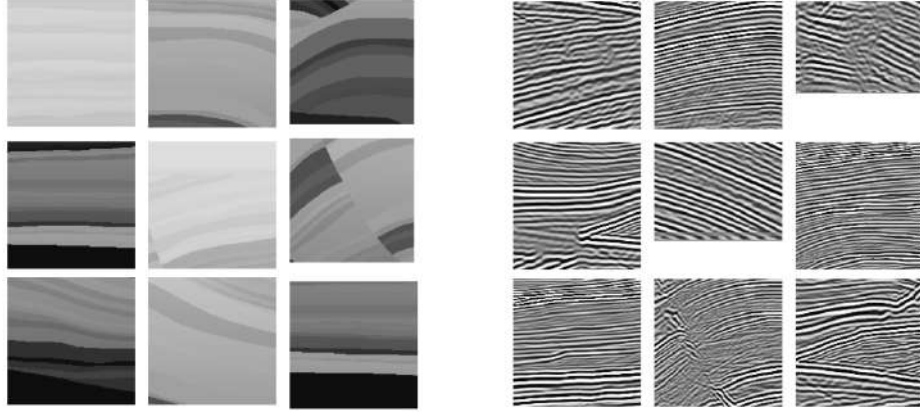


Figure 2: Example training patches extracted from the Marmousi2 synthetic model by Martin (2004). On the left velocities and on the right synthetic seismic forward models are shown.

velocity model, while the velocity model shows some continuity of velocities across the fault where appropriate.

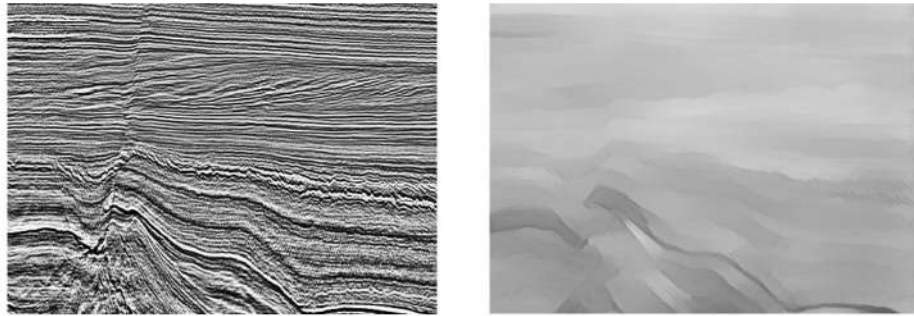


Figure 3: (left) Input seismic to network \mathbf{F} to test the seismic inversion performance of generative adversarial networks. (right) Extracted velocity model generated by network \mathbf{F} .

6 Conclusions

We have presented a method to generalise seismic forward and inverse modeling approaches using domain transfer methods. Sets of training images of two-dimensional synthetic velocity models and forward models have been used to train a pair of deep convolutional neural networks. Once trained, these networks allow extremely fast extraction of estimated velocity fields and geological structure showing qualitatively good results on unseen seismic observations such as the F3 dataset. In our experimentation so far the technique appears to be particularly robust even when training is performed on synthetic datasets containing velocity errors and noise artifacts, providing convincing forward pass results on seismic data from the field. We believe the cyclic consistency constraint within the architecture and the associated relaxation of the requirement of perfectly matched paired input-output images plays a key role in stabilising the network, making this transfer possible.

C.2 Correlation of Fractures From Core, Borehole Images and Seismic Data in a Chalk Reservoir in the Danish North Sea

Abstract: We present an integrated fracture study in the Ekofisk chalk reservoir of the Kraka Field, offshore Denmark, based on core, borehole images and seismic data. The core contains numerous fractures ranging from short (cm-scale) fractures, mostly associated with chert or stylolites, to large (m-scale) open, slickensided fractures likely related to halokinesis. On borehole images, especially larger fractures are identified, coinciding in dip and dip-azimuth. Seismic data at an approximate resolution of 40m would not resolve these local features around the well-bore. We show that chromatic analysis combined with an ant-tracking algorithm extracts several lineaments (> m-scale) from the seismic data. These correlate closely in orientation and distribution with the fractures logged in the well data. It is likely that these represent fracture corridors, small faults or damage zones in the chalk. The seismic data therefore provides a valuable method for mapping the size, orientation and connectivity of fracture zones away from the well. This gives insights into the scalability of local stress fields, and fracture distributions.

T. M. Aabø, J. S. Dramsch, M. Welch, and M. Lühje (June 2017). “Correlation of Fractures From Core, Borehole Images and Seismic Data in a Chalk Reservoir in the Danish North Sea”. In: *79th EAGE Conference and Exhibition 2017*. Published, Appendix C.2. EAGE. DOI: 10.3997/2214-4609.201701283. URL: <https://doi.org/10.3997/2214-4609.201701283>



Introduction

The Kraka oil Field is a salt-induced anticlinal structure located in the southernmost tip of the Danish Central Graben (Danish North Sea). It is produced through natural depletion of the Danian Ekofisk Fm, an overpressured, naturally fractured chalk reservoir. Ekofisk chalk is a mono-mineralic carbonate rock that consists of 96 - 99% calcite (CaCO_3), non-carbonate biogenic particles and small amounts of clay particles (Abramovitz et al., 2010). The Danian of the Kraka Field is divided into an upper porous zone (units D1 – D3) and a lower tight zone (D4 – D5). Porosities in the porous zone are in the range of 25 - 35%, and vary only slightly across the field (Klinkby et al., 2005). In the reservoir, silica occurs as continuous chert bands and isolated chert nodules. The matrix permeability is less than 1mD, however, the effective permeability is approximately 20 times that due to the presence of fractures.

Tectonic fractures related to halokinesis are the main permeability enhancers. Smaller fractures associated with cherts and stylolites may however be important for local permeability enhancement. Fractures in Kraka occur in swarms (Jorgensen et al., 1991). Fracture spacing, orientations and connectivity in the field are currently not well constrained.

In this extended abstract we compare fractures and fracture zones observed at different scales on core, borehole images (FMS and FMI) and ant-tracked seismic volumes, and show that we can correlate between them. BHI and core data are highly complementary. Borehole images are cheaper, provide true orientations and survey the reservoir in-situ, so we can differentiate open and closed fractures under reservoir conditions. Cores allow for direct observations, analyses at nano and micro scale and laboratory experiments. Seismic is used to map faults and fracture zones away from the borehole and to identify regional structural trends. Combining the three data types allows us to extrapolate fractures away from the borehole, and will serve as inputs into a mechanically based discrete fracture model (DFN), which will improve future well planning and EOR activities.

Method and Theory

Micro-resistivity images from FMS and FMI tools are available in one vertical well and seven horizontal or deviated wells. The surveyed wells were drilled in the time period between 1989 and 1997 and raw data has been reprocessed for this study. Due to the high resistivity of chalk, caving and tool sticking, image quality is poor in many places. Chalk sections directly below chert bands have been particularly hard to resolve, as the chert bands do not have planar surfaces and so cause errors in the pad alignment stage of processing. The cherts, being highly resistive compared to the chalk, are in turn well resolved. A large portion of the fracture swarms in Kraka are chert associated. Additionally, cherts enable depth matching between borehole images and cores, as depth shifts along wells vary by up to 8 ft

Cores are available in three of the wells logged by BHI tools: Well 1 (deviated), Well 2 (horizontal), and Well 3 (vertical). This abstract focuses on analyses from Well 1, where the core-recovery percentage is highest. Relative fracture orientations measured in core have been reoriented, depth shifted and are plotted alongside image picks.

Fracture picks from BHIs and cores are subsequently compared to a structural framework derived from seismic images. Seismic amplitude cubes acquired in 2012 have a vertical resolution in the order of 40 m. Therefore, high-fidelity information from the Kraka amplitude cube has been extracted using a gapped chromatic method that is based on structurally sharpened satellite RGB/HSV processing (Laake, 2015). The resulting structural cube serves as an input for an algorithm that systematically analyze the data, mimicking the "swarm intelligence" of ants (Pedersen et al., 2005). The algorithm extracts structural lineaments and assigns confidence levels depending on the length and width of the path of segments, to enhance subtle compaction features and small-scale faulting that are essential to the interpretation of the Kraka chalk field. The additional structural information is used as an opaque overlay on the conventional amplitude cube to guide the seismic interpretation and to avoid misclassification of noise or acquisition artifacts. Centimeter- to meter-scale fractures identified in borehole images and cores are compared to these larger structural lineations in 3D.



Correlation of Core and Borehole Images

In the BHI data from the Kraka Field, sinusoids representing bedding (chalk and marl) are continuous across borehole images. Most fractures are however only represented by partial sinusoids, either because they are short or because they are only partially open or cemented. Comparison with core data, when available, is imperative in determining which partial signals should be picked. Lessons learned from cored wells are transferrable to BHI-surveyed wells without core.

The advantage of core is that we can identify smaller-scale stylolite associated fractures that are not detectable on images because the image resolution is about 1-2mm. Most open chert associated fractures are however visible, due to the large resistivity contrast between cherts and water-based drilling mud. The length of the chert-associated fractures depend on the size of the chert band or nodule, and varies between 10 and 50 cm in Well 1.

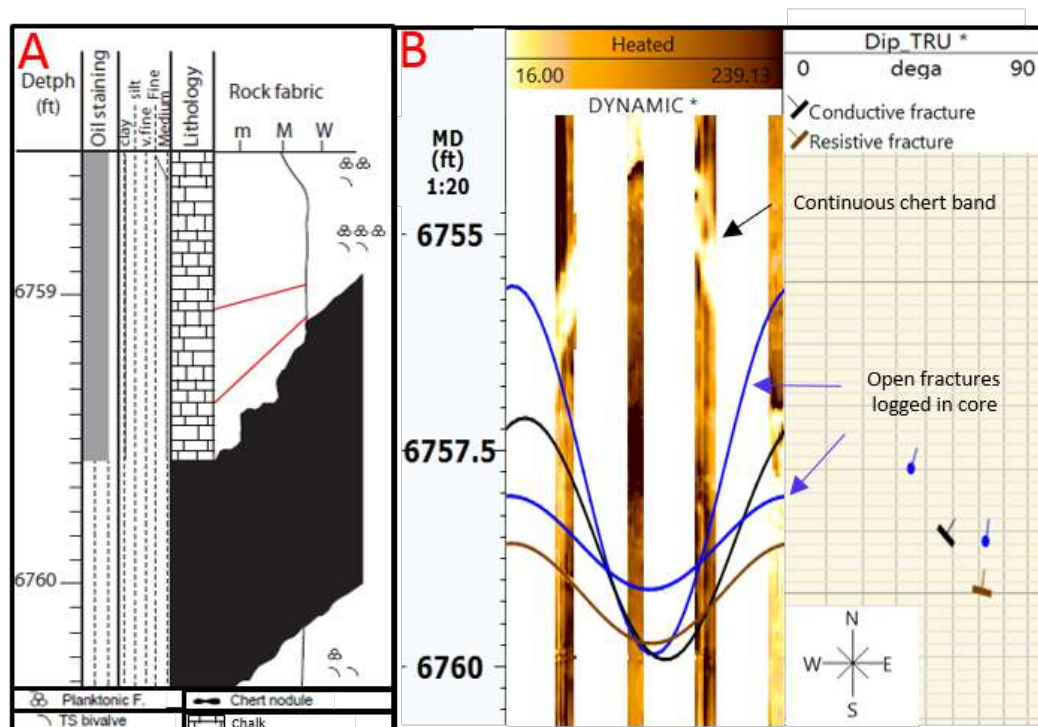


Figure 1 Core log (a) with faults in red and Core-to-BHI correlation (b) of fractured interval in Well 1

Figure 1 shows a logged core interval (a) with corresponding BHI section (b) from Well 1. The core interval contains two natural fractures, represented by blue tadpoles in the BHI. Both fractures are open (non-cemented) in the core, but are recognized as natural fractures by the presence of slickensides. The logged fractures coincide with one open (conductive) and one closed (resistive) fracture picked on the borehole image. The conductive fracture is associated with the continuous chert band, while the resistive feature is believed to represent a tectonic fracture. The relative timing of silica formation and saltdoming in Kraka is yet to be determined. However, as one of the closely-spaced fractures is cemented, while the other is not, it is reasonable to assume that they represent different fracture-generations.

The dip and azimuth of both fractures identified in BHI match the orientation of the fractures logged in core within 12°. Small discrepancies are to be expected, as core must be reoriented manually to calculate true orientations, so fracture orientations from BHIs are commonly considered the most reliable, while the presence and type of fractures can be identified in the core.



Correlation of Seismic and Well Data

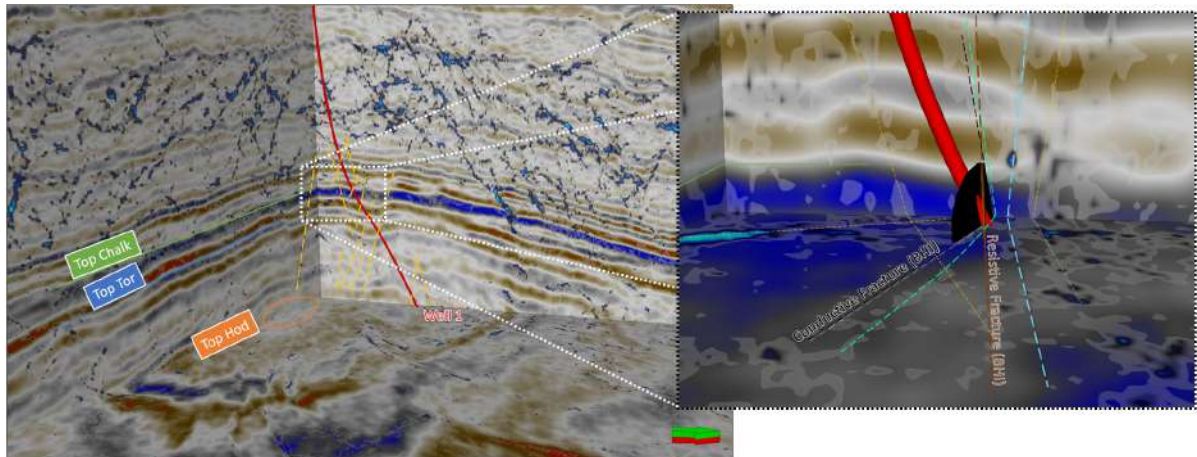


Figure 2 Seismic overlaid with ant-tracked chromatic structural cube and close-up of interpretation compared to BHI fracture discs.

In figure 2 we show the ant-track as transparent overlay over the seismic amplitude. The Top Hod to Top Chalk interval and the highly fractured overburden in the Kraka field are imaged. The z-plane in figure 2 cuts the inner chalk reservoir on Top Hod level. On this plane several lineations can be identified that strike radially relative to the anticlinal reservoir structure. The primary reservoir between Top Tor and Top Chalk shows several discontinuities and amplitude variations. On the cross-section we see conjugate inclined ($< 45^\circ$) faults cut through the reservoir and extend into the overburden and the underlying chalk package. Amplitude variations within the reservoir reflect compaction effects and possibly the influence of fluids saturation. The overburden is heavily fractured with low-throw (15m) conjugate faults that are highlighted with high confidence levels (blue) by the ant-tracking algorithm. The faults are parallel with a spacing between 50 m and 100 m and may be reactivated during depletion. This dynamic overburden must be corrected for in 4D seismic analysis.

The close-up in figure two highlights the faulting at reservoir depth. The z-plane was adjusted to reflect the middle Danian data. Cross-cutting faults along the wells are easily identified.

High-confidence features along the well may be production related. The close-up also increases the visibility of lower confidence features from the ant-track algorithm. These also reflect low-throw conjugate faulting cutting the reservoir and compaction related features are highly visible in this display.

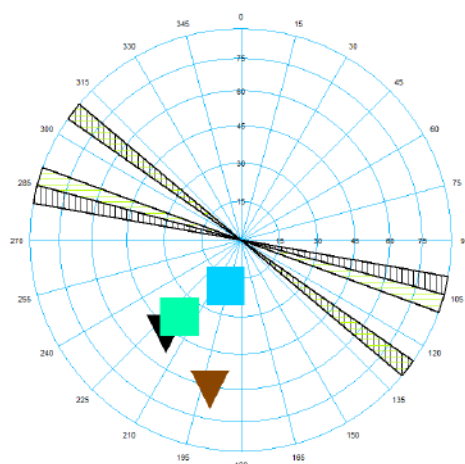


Figure 3 Rose diagram with stereonet overlay, representing BHI picks as triangles and seismic fault as squares

Along Well 1 cross-cutting faults are clearly visible. In the close-up, the two fracture picks from the BHI (Fig. 1) are imaged as discs corresponding to dip angle and azimuth. The Kraka reservoir is highly fractured and heterogeneities are clearly visible as amplitude variations on a seismic scale.

The dip and the dip-azimuth of the conductive fracture (black) corresponds well to the fault (green) from the seismic interpretation. The dip-azimuth of the resistive feature (brown) aligns with the seismic feature (blue), the dip angle deviates, however. The rose diagram in figure 3 shows the azimuth at a 5° interval coinciding for both fractures. The stereonet overlay also confirms



the qualitative assessment that the dip angle of the resistive fracture is steeper than that of the turquoise fault plane and a good match of the conductive fracture with the blue fault plane.

These results show that seismic data and careful post-processing shows stress trends that are reflected at the subseismic scale by BHI and core data. Seismic data is not capable of distinguishing open and closed fractures. Parallel and conjugate faults (yellow) in figure 2 strengthen the case of a consistent regional stress field that scales down to local stresses observed at the BHI and core scale. These can serve as input to building a field scale DFN.

Conclusions

Integrated comparisons of core, borehole image and seismic structural data in the Kraka Field indicate that:

- Many of the fractures seen on core can also be identified on borehole images, especially chert associated fractures. However stylolite associated fractures identified in core are not visible on borehole images. Chert associated fractures, extensional fractures and faults are commonly represented by partial sinusoids in BHIs, suggesting they are either short or only partially open or cemented.
- Borehole images are imperative in distinguishing cemented and open fractures, and thus better constrain fluid flow along the fracture network.
- Chromatic method and ant-track algorithm allows us to image subtle faults, fracture zones and compaction features not obvious on amplitude cubes.
- Structural features picked on BHIs correlate to large-scale regional trends and to features picked on ant-tracked seismic data. This allows us to extrapolate them away from the wellbores and calibrate 3D models (e.g. discrete fracture network models).

This integrated study proves invaluable in testing assumptions in building fracture models and the subsequent upscaling process.

Acknowledgements

The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding programme. We thank DUC for providing access to the data. We thank Frédéric Amour and Solomon Seyum for valuable discussion. We would also like to extend our thanks Schlumberger for providing licenses for Petrel, eXchroma and Techlog and fFA for the GeoTeric license, which were used in to interpret this data.

References

- Abramovitz, T., Andersen, C., Jakobsen, F., Kristensen, L. and Sheldon, E. [2010] 3D seismic mapping and porosity variation of intra-chalk units in the southern Danish North Sea. In: *Geological Society, London, Petroleum Geology Conference series*, 7. Geological Society of London, 537–548.
- Jorgensen, L., Andersen, P. et al. [1991] Integrated study of the Kraka Field. In: *Offshore Europe*. Society of Petroleum Engineers.
- Klinkby, L., Kristensen, L., Nielsen, E.B., Zinck-Jørgensen, K. and Stemmerik, L. [2005] Mapping and characterization of thin chalk reservoirs using data integration: the Kraka Field, Danish North Sea. *Petroleum Geoscience*, **11**(2), 113–124.
- Laake, A. [2015] Structural interpretation in color - A new RGB processing application for seismic data. *Interpretation*, **3**(1), SC1–SC8.
- Pedersen, S.I., Randen, T., Sonneland, L. and Steen, O. [2005] *Automatic fault extraction using artificial ants*. 512–515.

APPENDIX D

Workshop Papers

D.1 Information Theory Considerations in Patch-based Training of Deep Neural Networks on Seismic Time-Series

Abstract: Recent advances in machine learning relies on convolutional deep neural networks. These are often trained on cropped image patches. Pertaining to non-stationary seismic signals this may introduce low frequency noise and non-generalizability.

J. S. Dramschi and M. L  thje (2018f). “Information Theory Considerations In Patch-Based Training Of Deep Neural Networks On Seismic Time-Series”. In: *First EAGE/PESGB Workshop Machine Learning*. Published, Appendix D. EAGE. DOI: 10.3997/2214-4609.201803020. URL: <https://doi.org/10.3997/2214-4609.201803020>



Introduction

Sampling in physics-based applications and digital signal processing has long been recognised as an essential constraint. The Nyquist-Shannon theorem is the most prominent information theorem that prevents aliasing in seismic data (Seibt, 2006). Sampling has to be considered an essential part of a machine learning pipeline to avoid the implicit bias of learnt decision boundaries and joint distributions.

Machine Learning algorithms, particularly deep convolutional neural networks (CNN) often learn on patches of data. In many applications, the dynamic range of the data is additionally converted from 32-bit floats to 8-bit integers. This loss of dynamic range often speeds up training of networks and stabilises convergence at the loss of accuracy. However, investigations into precision have shown that this effect may be negligible (Holi and Hwang, 1993). Patch-based image training in machine learning usually takes smaller windows of data. The ImageNet challenge (Deng et al., 2009) provides 256x256 pixel images, which sets standards for many machine learning architectures.

Theory

Seismic traces are often sampled at 4ms and contain several hundred to thousands of samples. The Nyquist-Shannon theorem applies to high-frequency bounds only. However, we propose that a lower bound has to be adhered to when applying real-valued transformations to data before reconstruction. Low-frequency aliasing can be seen as a DC offset, where DC is the value at 0 Hz. This effect has been studied in non-stationary signals in applications such as seismic frequency decomposition (Chakraborty and Okaya 1995).

In statistical learning, many applications learn implicit joint distributions of the data. These are often approximated by multivariate distributions or transformations that operate solely on real-valued signals instead of complex signals (Hirose, 2003). This is equivalent to a mean shift of the data, as well as noise of the mean and may hinder convergence of the algorithms and diminish results. Inference on images that can appropriately sample low frequencies, due to a larger size, could lead to non-generalizability of the data due to implicit bias, which is the antithesis of machine learning.

We propose a low-frequency boundary, which follows the Nyquist-Shannon sampling theorem. With $f_{ny} = \frac{1}{2T}$, where T is the maximum period resolvable in the time series. This is due to the fact that we treat cutouts of a non-stationary signal as representative of the entire series and therefore, have to infer stationarity within the available bandwidth.

Example: Neural Network - Single Neuron

Neurons in neural networks are described by the activation $\sigma(w \cdot x + b)$, where w is the network weights, x is the input data, b is the network bias, and sigma is a non-linear activation function. A common non-linear activation is the rectified linear unit (RELU) $\sigma(x) = \max(0, x)$. Considering the inference stage, the network weights w and biases b are fixed, x is the only variable parameter. Learning on a mean-shift of q of an arbitrary distribution over x leads to $\sigma(w \cdot (x + q) + b)$, which increases the neuron response by q , weighted by w . At inference, the mean-shift over larger inference data disappears, introducing an additional bias of $w \cdot q$ before non-linear activation. This training bias may lead to prediction errors of the neuron and consequently the full neural network.

Example: Dutch F3 Seismic data

We use a randomly selected trace from the Dutch F3 dataset. The total recording time is 4 seconds with 1001 samples sampled at 4 ms. The sampling interval of 4ms allows for a maximum frequency of 125 Hz. We compare the reconstruction of the signal from the real part of the frequency spectrum for non-overlapping patches. The frequency content of real-valued stationary traces would be similar, whether a trace is split into parts or whole.

The frequency content in Figure 1 shows that properly tapered data introduces a DC offset and the phase spectrum cannot be reconstructed fully. For a window of 101 samples at 4 ms, we get the lower Nyquist frequency of ~ 12.5 Hz. A patch of 256 samples at 4ms has the lower bound of ~ 5 Hz. We propose that a high-pass filter at training may improve convergence. Transfer learning on larger patches with fewer epochs then recovers low-frequency information, while keeping training times attainable.

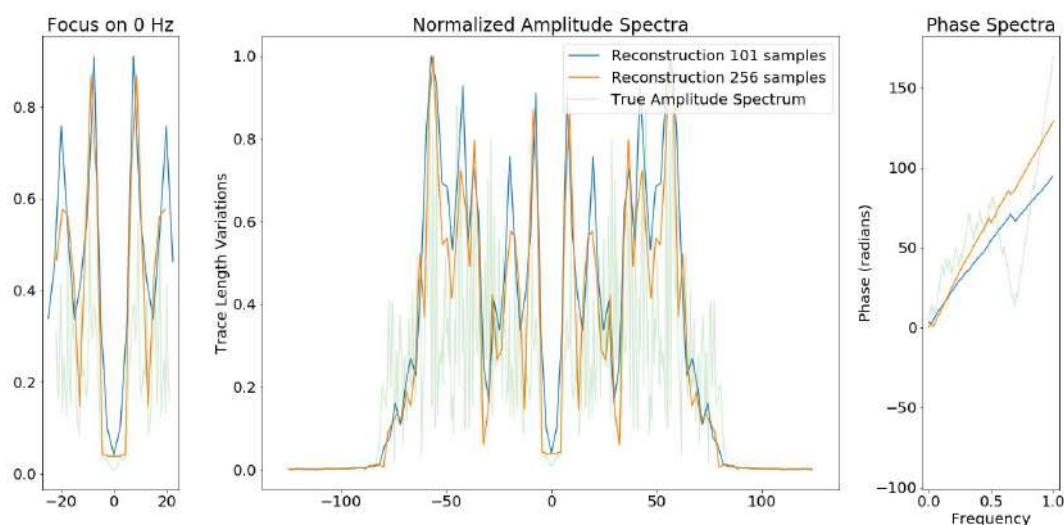


Figure 1 We present different sizes of cutouts, with 101 and 256 samples respectively. In the middle, the full normalised amplitude spectra are presented. On the right, the according phase spectra are presented. On the left, we focus on the frequency content of the amplitude spectra around 0 Hz. The cutouts were Hanning tapered, however, a clear DC offset appears with decreasing patch size.

Conclusions

We investigate the frequency content in non-overlapping patch-based seismic data. Non-overlapping patches may introduce low-frequency noise that translates to a mean-shift of learnt distributions. Further investigations into frequency responses of Convolutional Neural Networks (CNN) and the computation thereof, which is common in the frequency domain, should be undertaken. The authors note that signal processing paradigms apply to image-based CNNs and tapering of time-series before Fourier transformation is essential.

Acknowledgements

The research leading to these results has received funding from the Danish Hydrocarbon Research and Technology Centre under the Advanced Water Flooding program. The authors thank Matthias Schneider for fruitful discussion and dGB for providing the F3 dataset.

References

- Avijit Chakraborty and David Okaya (1995). "Frequency-time decomposition of seismic data using wavelet-based methods." *GEOPHYSICS*, 60(6), 1906-1916
- Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (pp. 248-255). Ieee.
- Hirose, A. (2003). Complex-Valued Neural Networks: An Introduction. In *Complex-Valued Neural Networks: Theories and Applications* (pp. 1-6).
- Holi, J. L., & Hwang, J. N. (1993). Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, (3), 281-290.
- Seibt, P. (2006). *Algorithmic Information Theory*. Springer.

APPENDIX E

Reproducible Code

E.1 Unsupervised Geological Image Segmentation

```
1 import numpy as np
2 from scipy import ndimage
3 import skimage
4 import cv2
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from skimage.morphology import watershed
8 from sklearn.mixture import GaussianMixture
9 from sklearn.preprocessing import normalize
10
11 from PIL import Image
```

```
1 plt.rcParams['figure.figsize'] = (10, 10)
```

```
1 bsem_grey = cv2.imread('bsem.png',0)
```

```
1 def sem_segment(input_image, threshold_low="x", threshold_hi="x",
2                 intraporosity=4):
3     if (threshold_low == "x"):
4         threshold_low = 0
5     if (threshold_hi == "x"):
6         threshold_hi = np.max(input_image)
7     binary_img = (input_image > threshold_low) & (input_image <
8         threshold_hi)
9     filled_img = ndimage.binary_fill_holes(binary_img)
10
11     inter = ndimage.binary_erosion(filled_img ^ binary_img, iterations=
12         intraporosity)
13     label_im, nb_labels = ndimage.label(inter.astype(np.int16))
14     wsfll = watershed(input_image, label_im, mask=np.invert(input_image >
15         threshold*.75))
```

```

12
13     intrapores = ndimage.binary_fill_holes(wsfll.astype(bool))
14
15     filled_img = ndimage.binary_fill_holes(binary_img) ^ intrapores
16
17     eroded_img = ndimage.binary_erosion(filled_img)
18     reconstruct_img = ndimage.binary_propagation(eroded_img, mask=ndimage.
19         filters.gaussian_filter(binary_img,10,order=2))
20     tmp = np.invert(reconstruct_img)
21     eroded_tmp = ndimage.binary_erosion(tmp)
22     final_mask = np.invert(ndimage.binary_propagation(eroded_tmp, mask=
23         ndimage.filters.gaussian_filter(tmp,10,order=2)))
24     return final_mask

```

```

1  classif = GaussianMixture(n_components=2,n_init=7)
2  classif.fit(bsem_grey.reshape((bsem_grey.size, 1)))
3  threshold = np.mean(classif.means_)

```

```

1  GMM_reconstruct_final = sem_segment(bsem_grey,threshold_low=threshold*.95)

```

```

1  means = classif.means_
2  means_order = np.argsort(means,axis=0)
3  low_mean = np.mean(means[(means_order==0) | (means_order==1)])
4  print("Please check if these are ok mean values.\n", classif.means_)

```

Please check if these are ok mean values. $[[201.98997646] \ [75.80232926]]$

```

1  binary_img = bsem_grey > threshold
2
3  plt.subplot(131)
4  plt.imshow(bsem_grey, cmap='gray')
5  plt.axis('off')
6  plt.subplot(132)
7  plt.imshow(binary_img, cmap='gray')
8  plt.axis('off')
9  plt.subplot(133)
10 plt.imshow(bsem_grey*sem_segment(bsem_grey,threshold_low=low_mean*.95,
11     intraporosity=3), cmap='gray')
12 plt.axis('off')
13 plt.savefig('segmentation_small.png')
14 plt.show()

```

```

1 plt.imshow(bsem_grey*sem_segment(bsem_grey,threshold_low=low_mean*.95,
    intraporosity=3),cmap='gray')
2 plt.title('Sediment Map')

```

```

1 plt.imshow(bsem_grey*np.invert(sem_segment(bsem_grey,threshold_low=
    low_mean*.95)),cmap='gray')
2 plt.title('Background Map')
3 bsem_grey.shape

```

```

1 print("The porosity is {:.2f}%".format(100 * (1-(np.count_nonzero(
    GMM_reconstruct_final) / ( GMM_reconstruct_final.shape[1] *
    GMM_reconstruct_final.shape[0])))))
2
3 The porosity is 45.70\%

```

```

1 X = bsem_grey.reshape((bsem_grey.size, 1))
2 flat_classif = GaussianMixture(n_components=2,n_init=7)
3 flat_classif.fit(X)
4 x = np.array(np.linspace(0,255,896*1024)).reshape(-1,1)
5 Z = -flat_classif.score_samples(x)

```

```

1 import matplotlib
2 font = {'size' : 18}
3
4 matplotlib.rc('font', **font)
5
6 fig, ax = plt.subplots()
7
8 ax.imshow([[0,1],[0,1]], cmap=plt.cm.gray, interpolation='bicubic', extent
    =(0, 255, 5, 7.25), alpha=1)
9
10 ax.plot(x, Z, 'r', label='Negative Log Likelihood')
11 plt.axvline(x=np.mean(classif.means_), label='GMM Decision Boundary')
12 plt.axvline(x=141, color='b', label='Histogram Decision Boundary')
13 plt.legend()
14 plt.title('Negative log-likelihood predicted by a GMM')
15 plt.axis('tight')
16 plt.savefig('GMM_decision_boundary.png', dpi = 200, bbox_inches='tight',
    pad_inches = 0)

```

```

1 from skimage import measure
2 def disk_structure(n):

```

```

3     struct = np.zeros((2 * n + 1, 2 * n + 1))
4     x, y = np.indices((2 * n + 1, 2 * n + 1))
5     mask = (x - n)**2 + (y - n)**2 <= n**2
6     struct[mask] = 1
7     return struct.astype(np.bool)
8
9
10 def granulometry(data, sizes=None):
11     s = max(data.shape)
12     if sizes is None:
13         sizes = range(1, s//2, 2)
14     granulo = [ndimage.binary_opening(data, \
15         structure=disk_structure(n)).sum() for n in sizes]
16     return granulo

```

```

1 label_im, nb_labels = ndimage.label(GMM_reconstruct_final)
2 print("Detected {} grains.".format(nb_labels))
3 sizes = ndimage.sum(GMM_reconstruct_final, label_im, range(nb_labels + 1))
4 mean_vals = ndimage.sum(GMM_reconstruct_final, label_im, range(1,
    nb_labels + 1))

```

Detected 490 grains.

```

1 opened_small = ndimage.binary_opening(GMM_reconstruct_final, structure=
    disk_structure(2))
2 opened = ndimage.binary_opening(GMM_reconstruct_final, structure=
    disk_structure(5))
3 opened_more = ndimage.binary_opening(GMM_reconstruct_final, structure=
    disk_structure(15))

```

```

1 plt.imshow(bsem_grey, cmap=plt.cm.gray)
2 plt.contour(opened, [0.1], colors='deepskyblue', linewidths=2)
3 plt.axis('off')
4 plt.show()

```

```

1 plt.imshow(bsem_grey, cmap=plt.cm.gray)
2 plt.contour(opened_more, [0.1], colors='darkorange', linewidths=2)
3 plt.axis('off')
4 plt.show()

```

```

1 plt.imshow(bsem_grey, cmap=plt.cm.gray)
2 plt.contour(opened_small, [0.1], colors='darkorange', linewidths=2)
3 plt.axis('off')

```

```
4 plt.savefig('segmentation_small.png')
5 plt.show()
```

E.1.1 Grain Calculations

```
1 from skimage import measure
2 scale = 50/1000
3
4 label_im, nb_labels = ndimage.label(opened_small)
5
6 GMM_reconstruct_final
7
8 props = measure.regionprops(label_im)
```

```
1 total_area=[]
2 total_diameter=[]
3
4 for p in props:
5     total_area.append(p.area)
6     total_diameter.append(p.equivalent_diameter)
```

```
1 plt.hist(np.log(total_area),bins=60)
2 plt.title('Log Distribution of grain sizes')
3 plt.savefig('grain_sizes.png', dpi = 200, bbox_inches='tight', pad_inches
    = 0)
```

```
1 from math import pi
2 total_error = np.divide(np.subtract(total_area,np.multiply(total_diameter,
    pi)),total_area)
3 plt.hist(total_error,bins=60)
4 plt.title('Deviation from circular grain shape.')
5 plt.savefig('circ_dev.png', dpi = 200, bbox_inches='tight', pad_inches =
    0)
```

```
1 peri = measure.perimeter(GMM_reconstruct_final, neighbourhood=8)
2 print("The perimeter is {:.3f} micrometers.".format(peri*scale))
```

The perimeter is 4235.111 micrometers.

E.2 Transfer learning in Automatic Seismic Interpretation

First we'll import all the libraries we need down the line. We also set the "random seed", so results can be reproduced by avid readers. Keras should report using the Tensorflow backend, otherwise reproducibility cannot be guaranteed.

```
1 import numpy as np
2 import pandas as pd
3 import obspy
4 import keras
5 import time
6 from keras_tqdm import TQDMNotebookCallback
7 from tqdm import trange, tqdm_notebook
8 import matplotlib.pyplot as plt
9
10
11 import tensorflow as tf
12 from obspy.io.segy.segy import _read_segy
13 from sklearn.model_selection import train_test_split
14
15 np.random.seed(42)
16 %matplotlib notebook
```

For experimentation with network models, we keep the keras imports separate, to reduce loading time.

```
1 from keras.models import Sequential, Model, clone_model
2 from keras.layers import Conv2D, Dense, Activation, Flatten, Dropout,
   Input
3 from keras.layers.normalization import BatchNormalization
4 from keras.preprocessing.image import ImageDataGenerator
5 from keras.optimizers import SGD
```

We need to define some parameters. As we are using Transfer learning, we have to adjust these parameters to fit into the network that we use and test.

```
1 patch_size = 64 # for ResNet50 put 244
2 batch_size = 256
3 num_channels = 1
4 num_classes = 9
5 all_examples = 158812
6 num_examples = 7500
7 epochs = 20
8 steps=450
9 sampler = list(range(all_examples))
10
11 opt = 'adam'
```



```

12 lossfkt = ['categorical_crossentropy']
13 metrica = ['mae', 'acc']

```

Here we test, whether we are running on CPU or GPU. We want to run on GPU, if it's not in the device list. It will be slow.

```

1 from tensorflow.python.client import device_lib
2 print(device_lib.list_local_devices())
3 # It should say GPU here. Otherwise your model will run sloooow.

```

E.2.1 Data Loading

Now let's load the F3 data and read three slices. The labeled data, as well as, a distal inline and a crossline.

```

1 filename = 'data/Dutch Government_F3_entire_8bit_seismic.segy'
2
3 t0=time.time()
4 stream0 = _read_segy(filename, headonly=True)
5 print('--> data read in {:.1f} sec'.format(time.time()-t0)) #Thanks to
   aadm
6
7 t0=time.time()
8
9 labeled_data = np.stack(t.data for t in stream0.traces if t.header.
   for_3d_poststack_data_this_field_is_for_in_line_number == 339).T
10 inline_data = np.stack(t.data for t in stream0.traces if t.header.
   for_3d_poststack_data_this_field_is_for_in_line_number == 500).T
11 xline_data = np.stack(t.data for t in stream0.traces if t.header.
   for_3d_poststack_data_this_field_is_for_cross_line_number == 500).T
12
13 print('--> created slices in {:.1f} sec'.format(time.time()-t0))

```

E.2.2 Helper Functions

From these slices, we need to extract patches. While, we could do that before and save them as array or image data, using a generator that utilizes the CPU, while the GPU trains the network is a bit more storage- and memory-friendly. `patch_extractor2D()` automates the patch-extraction and pads sides, where necessary.

Then we build `acc_assess()` to format our test accuracy assessment nicely, because we're lazy and retyping it for every model we build is a nuisance.

All functions are accompanied with a little sanity check. While this is not automated testing (like TDD), it does help to make sure, our function works as intended.

```

1 def patch_extractor2D(img,mid_x,mid_y,patch_size,dimensions=1):
2     try:
3         x,y,c = img.shape
4     except ValueError:
5         x,y = img.shape
6         c=1
7     patch=np.pad(img, patch_size//2, 'constant', constant_values=0)[mid_y
8         :mid_y+patch_size,mid_x:mid_x+patch_size] #because it's padded we
9         don't subtract half patches all the tim
10    if c != dimensions:
11        tmp_patch = np.zeros((patch_size,patch_size,dimensions))
12        for uia in range(dimensions):
13            tmp_patch[:, :,uia] = patch
14        return tmp_patch
15    return patch
16
17 image=np.random.rand(10,10)//.1
18 print(image)
19
20 patch_extractor2D(image,10,10,4,1)

```

```

1 def acc_assess(data,loss=['categorical_crossentropy'],metrics=['acc']):
2     if not isinstance(loss, list):
3         try:
4             loss = [loss]
5         except:
6             raise("Loss must be list.")
7     if not isinstance(metrics, list):
8         try:
9             metrics = [metrics]
10        except:
11            raise("Metrics must be list.")
12    out='The test loss is {:.3f}\n'.format(data[0])
13    for i, metric in enumerate(metrics):
14        if metric in 'mae':
15            out += "The total mean error on the test is {:.3f}\n".format(
16                data[i+1])
17        if metric in 'accuracy':
18            out += "The test accuracy is {:.1f}%\n".format(data[i+1]*100)
19    return out
20 print(acc_assess([1,2,3], 'bla', ["acc", "mae"]))

```

E.2.3 Exploratory Data Analysis

We need to load and check our labels.

```

1 labels = pd.read_csv('data/classification.ixz', delimiter=" ", names=["
2     Inline", "Xline", "Time", "Class"])

```

```
2 labels.describe()
```

```
1 labels["Xline"]-=300-1
2 labels["Time"] = labels["Time"]//4
3 labels.describe()
```

```
1 labeled_data.shape
```

```
1 fig2 = plt.figure(figsize=(15.0, 10.0))
2 vml = np.percentile(labeled_data, 99)
3 img1 = plt.imshow(labeled_data, cmap="Greys", vmin=-vml, vmax=vml, aspect=
    'auto')
4 plt.yticks(np.arange(0, 462, 100), np.arange(0, 462*4, 400))
5 plt.xlabel('Trace Location')
6 plt.ylabel('Time [ms]')
7 plt.savefig('labeled_data.png', bbox_inches='tight')
8 plt.show()
```

```
1 fig2 = plt.figure(figsize=(15.0, 10.0))
2 vmx = np.percentile(xline_data, 99)
3 plt.imshow(xline_data, cmap="Greys", vmin=-vmx, vmax=vmx, aspect='auto')
4 plt.yticks(np.arange(0, 462, 100), np.arange(0, 462*4, 400))
5 plt.xlabel('Trace Location')
6 plt.ylabel('Time [ms]')
7 plt.savefig('xline_data.png', bbox_inches='tight')
8 plt.show()
```

```
1 fig2 = plt.figure(figsize=(15.0, 10.0))
2 vmy = np.percentile(inline_data, 99)
3 plt.imshow(inline_data, cmap="Greys", vmin=-vmy, vmax=vmy, aspect='auto')
4 plt.yticks(np.arange(0, 462, 100), np.arange(0, 462*4, 400))
5 plt.xlabel('Trace Location')
6 plt.ylabel('Time [ms]')
7 plt.savefig('inline_data.png', bbox_inches='tight')
8 plt.show()
```

```
1 fig2 = plt.figure(figsize=(15.0, 10.0))
2 img2 = plt.imshow(labeled_data, cmap="Greys", vmin=-vml, vmax=vml, aspect=
    'auto')
3 img1 = plt.scatter(labels["Xline"], labels[["Time"]], c=labels[["Class"]],
    cmap='Dark2', alpha=0.03)
```

```

4 plt.yticks(np.arange(0, 462, 100), np.arange(0, 462*4, 400))
5 plt.xlabel('Trace Location')
6 plt.ylabel('Time [ms]')
7 plt.savefig('label.png', bbox_inches='tight')
8 plt.show()

```

E.2.4 Train the Network

Now we perform a test-train split. Then we can validate the results of our experiment.

```

1 train_data, test_data, train_samples, test_samples = train_test_split(
2     labels, sampler, random_state=42)
3 print(train_data.shape, test_data.shape)

```

This is the ‘keras’ data generator that wraps the ‘patch_extractor2D()’.

```

1 class SeismicSequence(keras.utils.Sequence):
2     def __init__(self, img, x_set, t_set, y_set, patch_size, batch_size,
3         dimensions):
4         self.slice = img
5         self.X, self.t = x_set, t_set
6         self.batch_size = batch_size
7         self.patch_size = patch_size
8         self.dimensions = dimensions
9         self.label = y_set
10
11     def __len__(self):
12         return len(self.X) // self.batch_size
13
14     def __getitem__(self, idx):
15         sampler = np.random.permutation(len(self.X))
16         samples = sampler[idx*self.batch_size:(idx+1)*self.batch_size]
17         labels = keras.utils.to_categorical(self.label[samples],
18             num_classes=9)
19         if self.dimensions == 1:
20             return np.expand_dims(np.array([patch_extractor2D(self.slice,
21                 self.X[x], self.t[x], self.patch_size, self.dimensions) for x
22                 in samples]), axis=4), labels
23         else:
24             return np.array([patch_extractor2D(self.slice, self.X[x], self.t
25                 [x], self.patch_size, self.dimensions) for x in samples]),
26                 labels

```

We define several callbacks for keras. The training should be stopped early, if the validation loss or the categorical cross entropy do not improve within the defined patience. Checkpoints are written to ‘tmp.h5’ for every epoch.

```

1 earlystop1 = keras.callbacks.EarlyStopping(monitor='val_loss',
2                                           min_delta=0,
3                                           patience=3,
4                                           verbose=0, mode='auto')
5
6 earlystop2 = keras.callbacks.EarlyStopping(monitor='val_acc',
7                                           min_delta=0,
8                                           patience=3,
9                                           verbose=0, mode='auto')
10
11 checkpoint = keras.callbacks.ModelCheckpoint('tmp.h5',
12                                             monitor='val_loss',
13                                             verbose=0,
14                                             save_best_only=False,
15                                             save_weights_only=False,
16                                             mode='auto',
17                                             period=1)
18
19 callbacklist = [TQDMNotebookCallback(leave_inner=True, leave_outer=True),
20                earlystop1, earlystop2, checkpoint]

```

E.2.5 Waldeland CNN

The model introduced by Waldeland, reproduced from MalenoV. Compared to today's standards this is a relatively shallow CNN. We train the network from scratch.

```

1 tf.logging.set_verbosity(tf.logging.ERROR)
2
3 model_vanilla = Sequential()
4 model_vanilla.add(Conv2D(50, (5, 5), padding='same', input_shape=(
5     patch_size, patch_size, 1), strides=(4, 4), data_format="channels_last",
6     name = 'conv_layer1'))
7 model_vanilla.add(BatchNormalization())
8 model_vanilla.add(Activation('relu'))
9 model_vanilla.add(Conv2D(50, (3, 3), strides=(2, 2), padding = 'same', name
10     = 'conv_layer2'))
11 model_vanilla.add(Dropout(0.5))
12 model_vanilla.add(BatchNormalization())
13 model_vanilla.add(Activation('relu'))
14 model_vanilla.add(Conv2D(50, (3, 3), strides=(2, 2), padding= 'same', name
15     = 'conv_layer3'))
16 model_vanilla.add(Dropout(0.4))
17 model_vanilla.add(BatchNormalization())
18 model_vanilla.add(Activation('relu'))
19 model_vanilla.add(Conv2D(50, (3, 3), strides=(2, 2), padding= 'same', name
20     = 'conv_layer4'))
21 model_vanilla.add(Dropout(0.2))
22 model_vanilla.add(BatchNormalization())

```

```

18 model_vanilla.add(Activation('relu'))
19 model_vanilla.add(Conv2D(50, (3, 3), strides=(2, 2), padding= 'same', name
   = 'conv_layer5'))
20 model_vanilla.add(Flatten())
21 model_vanilla.add(Dense(50, name = 'dense_layer1'))
22 model_vanilla.add(BatchNormalization())
23 model_vanilla.add(Activation('relu'))
24 model_vanilla.add(Dense(10, name = 'attribute_layer'))
25 model_vanilla.add(BatchNormalization())
26 model_vanilla.add(Activation('relu'))
27 model_vanilla.add(Dense(num_classes, name = 'pre-softmax_layer'))
28 model_vanilla.add(BatchNormalization())
29 model_vanilla.add(Activation('softmax'))
30
31 model_vanilla.compile(loss=lossfkt,
32                       optimizer=opt,
33                       metrics=metrica)

```

```

1 t0=time.time()
2
3 hist_vanilla = model_vanilla.fit_generator(
4     SeismicSequence(
5         labeled_data,
6         train_data["Xline"].values,
7         train_data["Time"].values,
8         train_data["Class"].values,
9         patch_size,
10        batch_size,
11        1),
12    steps_per_epoch=steps,
13    validation_data = SeismicSequence(
14        labeled_data,
15        test_data["Xline"].values,
16        test_data["Time"].values,
17        test_data["Class"].values,
18        patch_size,
19        batch_size,
20        1),
21    validation_steps = len(test_samples)//batch_size,
22    epochs = epochs,
23    verbose = 0,
24    callbacks = callbacklist)
25
26 print('--> Training for Waldeland CNN took {:.1f} sec'.format(time.time()-
   t0)) #Thanks to aadm

```

```

1 model_vanilla.save("vanilla_model.h5")

```

```

1 vanillascore=model_vanilla.evaluate(np.expand_dims(np.array([
    patch_extractor2D(labeled_data,labels["Xline"][x],labels["Time"][x],64)
    for x in test_samples]), axis=4),keras.utils.to_categorical(labels["
    Class"][test_samples], num_classes=9), verbose=0)
2 print(acc_assess(vanillascore,lossfkt,metrica))

```

Looking at the metric on training as well as validation gives a good overview, if we are doing appropriate training or if we are overfitting.

```

1 print(hist_vanilla.history.keys())
2 plt.plot(hist_vanilla.history['acc'])
3 plt.plot(hist_vanilla.history['val_acc'])
4 plt.title('model accuracy')
5 plt.ylabel('accuracy')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()

```

```

1 # summarize history for loss
2 plt.plot(hist_vanilla.history['loss'])
3 plt.plot(hist_vanilla.history['val_loss'])
4 plt.title('model loss')
5 plt.ylabel('loss')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()

```

```

1 t_max, y_max = xline_data.shape
2
3 half_patch = patch_size//2
4
5 predx = np.full_like(xline_data,-1)
6
7 for space in tqdm_notebook(range(y_max),desc='Space'):
8     for depth in tqdm_notebook(range(t_max),leave=False, desc='Time'):
9         predx[depth,space] = np.argmax(model_vanilla.predict(np.
            expand_dims(np.expand_dims(patch_extractor2D(xline_data,space,
            depth,patch_size), axis=0), axis=4)))

```

```

1 np.save('vanilla_predx.npy',predx,allow_pickle=False)

```



```
1 plt.imshow(predx)
```

```
1 fig2 = plt.figure(figsize=(15.0, 10.0))
2 img2 = plt.imshow(xline_data, cmap="Greys", vmin=-vmx, vmax=vmx, aspect='
    auto')
3 img1 = plt.imshow(predx, aspect='auto', cmap="Dark2", alpha=0.5)
4 plt.savefig('pred1_x.png', bbox_inches='tight')
5 plt.show()
```

```
1 t_max, y_max = inline_data.shape
2
3 half_patch = patch_size//2
4
5 predi = np.full_like(inline_data, -1)
6
7 for space in tqdm_notebook(range(y_max), desc='Space'):
8     for depth in tqdm_notebook(range(t_max), leave=False, desc='Time'):
9         predi[depth, space] = np.argmax(model_vanilla.predict(np.
            expand_dims(np.expand_dims(patch_extractor2D(inline_data, space,
            depth, patch_size), axis=0), axis=4)))
```

```
1 np.save('vanilla_predi.npy', predi, allow_pickle=False)
```

```
1 predi = np.load('vanilla_predi.npy')
2 plt.imshow(predi)
```

```
1 np.save('vanilla_predi.npy', predi, allow_pickle=False)
2 fig2 = plt.figure(figsize=(15.0, 10.0))
3 img2 = plt.imshow(inline_data, cmap="Greys", vmin=-vmy, vmax=vmy, aspect='
    auto')
4 img1 = plt.imshow(predi, aspect='auto', cmap="Dark2", alpha=0.5)
5 plt.yticks(np.arange(0, 462, 100), np.arange(0, 462*4, 400))
6 plt.xlabel('Trace Location')
7 plt.ylabel('Time [ms]')
8 plt.savefig('pred1_i.png', bbox_inches='tight')
9 plt.show()
```

E.2.6 VGG16 Transfer Learning

We import the VGG16 model trained on the ImageNet dataset. We freeze all layers and cut off the classification part. We can then retrain the classification neurons, to see if the filters generalize to seismic data.

```
1 from keras.applications.vgg16 import VGG16
2 from keras import backend as K
3 K.set_image_dim_ordering('tf')
```

```
1 input_tensor = Input(shape=(patch_size, patch_size, 3))
2 base_model = keras.applications.vgg16.VGG16(include_top=False, weights='
    imagenet', input_tensor=input_tensor, input_shape=None)
3
4 for layer in base_model.layers[:8]:
5     layer.trainable = False
```

```
1 x = base_model.output
2 x = Flatten()(x)
3 x = Dense(256, name = 'dense_layer1')(x)
4 x = BatchNormalization()(x)
5 x = Activation('relu')(x)
6 x = Dropout(.5)(x)
7 x = Dense(num_classes, name = 'pre-softmax_layer')(x)
8 x = BatchNormalization()(x)
9 x = Activation('softmax')(x)
10
11 vgg = Model(input=base_model.input, output=x)
```

```
1 sgd = SGD(lr=1e-4, decay=1e-6, momentum=0.9, nesterov=True)
2 vgg.compile(loss=lossfkt,
3             optimizer=sgd,
4             metrics=metrics)
```

```
1 t0 = time.time()
2
3 vgg_hist = vgg.fit_generator(
4     SeismicSequence(
5         labeled_data,
6         train_data["Xline"].values,
7         train_data["Time"].values,
8         train_data["Class"].values,
9         patch_size,
10        batch_size,
11        3),
```

```

12     steps_per_epoch=steps,
13     validation_data = SeismicSequence(
14         labeled_data,
15         test_data["Xline"].values,
16         test_data["Time"].values,
17         test_data["Class"].values,
18         patch_size,
19         batch_size,
20         3),
21     validation_steps = len(test_data)//batch_size,
22     epochs = epochs,
23     verbose = 0,
24     callbacks = callbacklist)
25
26 print('--> Training for VGG transfer took {:.1f} sec'.format(time.time()-
    t0))

```

```

1 vgg.save('vgg_model.h5')

```

```

1 vgg_score=vgg.evaluate(np.array([patch_extractor2D(labeled_data, labels["
    Xline"][x], labels["Time"][x], 64, 3) for x in test_samples]), keras.utils
    .to_categorical(labels["Class"][test_samples], num_classes=9))
2 print(acc_assess(vgg_score, lossfkt, metrica))

```

```

1 print(hist_vanilla.history.keys())
2 plt.plot(vgg_hist.history['acc'])
3 plt.plot(vgg_hist.history['val_acc'])
4 plt.title('model accuracy')
5 plt.ylabel('accuracy')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()

```

```

1 # summarize history for loss
2 plt.plot(vgg_hist.history['loss'])
3 plt.plot(vgg_hist.history['val_loss'])
4 plt.title('model loss')
5 plt.ylabel('loss')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()

```

```

1 t_max, y_max = xline_data.shape
2
3 half_patch = patch_size//2
4
5 vgg_predx = np.full_like(xline_data,-1)
6
7 for space in tqdm_notebook(range(y_max),desc='Space'):
8     for depth in tqdm_notebook(range(t_max),leave=False, desc='Time'):
9         vgg_predx[depth,space] = np.argmax(vgg.predict(np.expand_dims(
            patch_extractor2D(xline_data,space,depth,patch_size,3), axis=0)
        )))

```

```

1 np.save('vgg_predx.npy',vgg_predx,allow_pickle=False)

```

```

1 plt.imshow(vgg_predx)

```

```

1 vgg_predx=np.load('vgg_predx.npy')
2 fig2 = plt.figure(figsize=(15.0, 10.0))
3 img2 = plt.imshow(xline_data, cmap="Greys", vmin=-vmx, vmax=vmx, aspect='
    auto')
4 img1 = plt.imshow(vgg_predx, aspect='auto', cmap="Dark2", alpha=0.5)
5 plt.yticks(np.arange(0, 462, 100), np.arange(0, 462*4, 400))
6 plt.xlabel('Trace Location')
7 plt.ylabel('Time [ms]')
8 plt.savefig('vgg1_x.png', bbox_inches='tight')
9 plt.show()

```

```

1 t_max, y_max = inline_data.shape
2
3 half_patch = patch_size//2
4
5 vgg_predi = np.full_like(inline_data,-1)
6
7 for space in tqdm_notebook(range(y_max),desc='Space'):
8     for depth in tqdm_notebook(range(t_max),leave=False, desc='Time'):
9         vgg_predi[depth,space] = np.argmax(vgg.predict(np.expand_dims(
            patch_extractor2D(inline_data,space,depth,patch_size,3), axis
            =0)))

```

```

1 np.save('vgg_predi.npy',vgg_predi,allow_pickle=False)

```

```

1 plt.imshow(vgg_predi)

2 vgg_predi= np.load('vgg_predi.npy')
3 fig2 = plt.figure(figsize=(15.0, 10.0))
4 img2 = plt.imshow(inline_data, cmap="Greys", vmin=-vmy, vmax=vmy, aspect='
    auto')
5 img1 = plt.imshow(vgg_predi, aspect='auto', cmap="Dark2", alpha=0.5)
6 plt.yticks(np.arange(0, 462, 100), np.arange(0, 462*4, 400))
7 plt.xlabel('Trace Location')
8 plt.ylabel('Time [ms]')
9 plt.savefig('vgg1_i.png', bbox_inches='tight')
10 plt.show()

```

E.2.7 ResNet50 Transfer Learning

We import the ResNet50 that was trained on the ImageNet data and freeze all layers, like we did for the VGG16. Then we retrain the classifier to see if the learned filters generalize on seismic data.

```

1 from keras.applications.resnet50 import ResNet50
2 from keras import backend as K
3 K.set_image_dim_ordering('tf')
4 patch_size=244

5 input_tensor = Input(shape=(patch_size,patch_size,3))
6 res_base = ResNet50(include_top=False, weights='imagenet', input_tensor=
    input_tensor, input_shape=None, pooling=None)
7
8 for layer in res_base.layers[:45]:
9     layer.trainable = False

10 q = res_base.output
11 q = Flatten()(q)
12 q = BatchNormalization()(q)
13 q = Activation('relu')(q)
14 q = Dense(10,name = 'attribute_layer')(q)
15 q = BatchNormalization()(q)
16 q = Activation('relu')(q)
17 q = Dense(num_classes, name = 'pre-softmax_layer')(q)
18 q = BatchNormalization()(q)
19 q = Activation('softmax')(q)
20 resnet = Model(input=res_base.input, output=q)

```

```

1 sgd = SGD(lr=1e-3, decay=1e-6, momentum=0.9, nesterov=True)
2 resnet.compile(loss=lossfkt,
3               optimizer=sgd,
4               metrics=metrca)

```

```

1 t0 = time.time()
2
3 batch_size=50
4 res_hist = resnet.fit_generator(
5     SeismicSequence(
6         labeled_data,
7         train_data["Xline"].values,
8         train_data["Time"].values,
9         train_data["Class"].values,
10        patch_size,
11        batch_size,
12        3),
13    steps_per_epoch=steps,
14    validation_data = SeismicSequence(
15        labeled_data,
16        test_data["Xline"].values,
17        test_data["Time"].values,
18        test_data["Class"].values,
19        patch_size,
20        batch_size,
21        3),
22    validation_steps = len(test_data)//batch_size,
23    epochs = epochs,
24    verbose = 0,
25    callbacks = callbacklist)
26
27 print('--> Training for ResNet transfer took {:.1f} sec'.format(time.time
    ()-t0))

```

```

1 resnet.save('resnet_model.h5')

```

```

1 resnetscore=resnet.evaluate(np.array([patch_extractor2D(labeled_data,
    labels["Xline"][x],labels["Time"][x],patch_size,3) for x in
    test_samples]), keras.utils.to_categorical(labels["Class"][test_samples
    ], num_classes=9))
2 print(acc_assess(resnetscore,lossfkt,metrca))

```

```

1 t_max, y_max = xline_data.shape
2
3 half_patch = patch_size//2
4
5 resnet_predx = np.full_like(xline_data, -1)
6
7 for space in tqdm_notebook(range(y_max), desc='Space'):
8     for depth in tqdm_notebook(range(t_max), leave=False, desc='Time'):
9         resnet_predx[depth, space] = np.argmax(resnet.predict(np.
            expand_dims(patch_extractor2D(xline_data, space, depth, patch_size
            , 3), axis=0)))

```

```

1 np.save('resnet_predx.npy', resnet_predx, allow_pickle=False)

```

```

1 plt.imshow(resnet_predx)

```

```

1 fig2 = plt.figure(figsize=(15.0, 10.0))
2 img2 = plt.imshow(xline_data, cmap="Greys", vmin=-vmx, vmax=vmx, aspect='
    auto')
3 img1 = plt.imshow(resnet_predx, aspect='auto', cmap="Dark2", alpha=0.8)
4 plt.savefig('resnet_x.png', bbox_inches='tight')
5 plt.show()

```

```

1 t_max, y_max = inline_data.shape
2
3 half_patch = patch_size//2
4
5 resnet_predi = np.full_like(inline_data, -1)
6
7 for space in tqdm_notebook(range(y_max-400, y_max-300), desc='Space'):
8     for depth in tqdm_notebook(range(t_max-400, t_max-300), leave=False,
        desc='Time'):
9         resnet_predi[depth, space] = np.argmax(resnet.predict(np.
            expand_dims(patch_extractor2D(inline_data, space, depth,
            patch_size, 3), axis=0)))

```

```

1 np.save('resnet_predi.npy', resnet_predi, allow_pickle=False)

```

```

1 plt.imshow(resnet_predi)

```



```

1 fig2 = plt.figure(figsize=(15.0, 10.0))
2 img2 = plt.imshow(inline_data, cmap="Greys", vmin=-vmy, vmax=vmy, aspect='
    auto')
3 img1 = plt.imshow(resnet_predi, aspect='auto', cmap="Dark2", alpha=0.8)
4 plt.savefig('resnet_i.png', bbox_inches='tight')
5 plt.show()

```

```

1 print(res_hist.history.keys())
2 plt.plot(res_hist.history['acc'])
3 plt.plot(res_hist.history['val_acc'])
4 plt.title('model accuracy')
5 plt.ylabel('accuracy')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()

```

```

1 # summarize history for loss
2 plt.plot(res_hist.history['loss'])
3 plt.plot(res_hist.history['val_loss'])
4 plt.title('model loss')
5 plt.ylabel('loss')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()

```

```

1 plot_model(resnet, to_file='model_resnet.png')
2 plot_model(resnet, to_file='model_resnet_shapes.png', show_shapes=True)
3 SVG(model_to_dot(resnet).create(prog='dot', format='svg'))

```

E.2.8 Model Summary

We can see the summaries of the layers in the model definitions. Leveraging high-dimensional CNNs that are already trained can be very valuable.

```

1 model_vanilla.summary()

```

```

1 vgg.summary()

```

```
1 resnet.summary()
```

E.3 Complex-valued neural networks

```
1 import numpy as np
2 from skimage.util.shape import view_as_windows
3 from skimage.io import imsave
4 from scipy.signal import hilbert
5
6 # Load Data
7 # Get it here: https://github.com/olivesgatech/
   facies_classification_benchmark
8 # Original here: https://terranubis.com/datainfo/Netherlands-Offshore-F3-
   Block-Complete
9 train_seismic = np.load("data/train/train_seismic.npy")
10
11 # Calculate Complex traces
12 train_hilbert = np.zeros_like(train_seismic, dtype=np.complex)
13 for x in range(train_hilbert.shape[0]):
14     for y in range(train_hilbert.shape[1]):
15         train_hilbert[x,y,:] = hilbert(train_seismic[x,y,:])
16
17 train_complex = train_hilbert
18 train_complex = train_hilbert - train_seismic
19
20 # Generate Patch Data
21 patch_size = 64
22 patch_size = 64
23
24 stride = 8
25
26 patch_shape = (1, patch_size, patch_size)
27
28 real_data = view_as_windows(train_seismic, patch_shape, step=stride)
29 cmplx_data = view_as_windows(train_complex, patch_shape, step=stride)
30
31 # Train - Validation Split
32 p = .9
33 val_split = np.random.choice(a=[False, True], size=real_data.shape[0:3], p
   =[p, 1-p])
34
35 # Inline Data
36 real = []
37 cmplx = []
38
39 for a in range(real_data.shape[0]):
40     for b in range(real_data.shape[1]):
```

```

41         for c in range(real_data.shape[2]):
42             real.append(np.squeeze(real_data[a,b,c,0,:,:]).T)
43             cmplx_patch = np.squeeze(cmplx_data[a,b,c,0,:,:]).T
44             cmplx.append(np.stack([np.real(cmplx_patch), np.imag(
45                 cmplx_patch)], axis=2))
46 np.save('patch_data/i_real.npy', real)
47 np.save('patch_data/i_cmplx.npy', cmplx)
48
49 # Crossline Data
50 patch_shape = (patch_size, 1, patch_size)
51
52 real_data = view_as_windows(train_seismic, patch_shape, step=stride)
53 cmplx_data = view_as_windows(train_complex, patch_shape, step=stride)
54
55 real = []
56 cmplx = []
57
58 for a in range(real_data.shape[0]):
59     for b in range(real_data.shape[1]):
60         for c in range(real_data.shape[2]):
61             real.append(np.squeeze(real_data[a,b,c,:,:0,:]).T)
62             cmplx_patch = np.squeeze(cmplx_data[a,b,c,:,:0,:]).T
63             cmplx.append(np.stack([np.real(cmplx_patch), np.imag(
64                 cmplx_patch)], axis=2))
65 np.save('patch_data/x_real.npy', real)
66 np.save('patch_data/x_cmplx.npy', cmplx)

```

```

1 import os
2 os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"    # see issue #152
3 os.environ["CUDA_VISIBLE_DEVICES"]="4,5,6,7"
4
5 import numpy as np
6 from sklearn.model_selection import train_test_split
7
8 import complexnn
9
10 import keras
11 from keras import models
12 from keras import layers
13 from keras import optimizers
14 from keras.utils import multi_gpu_model
15
16 act_fn = 'elu'
17
18 def CAE(input_shape=None, x=0):
19
20     model = models.Sequential()
21     # Block 1

```

```

22 model.add(complexnn.conv.ComplexConv2D(2**(x+1), (3, 3), activation=
    act_fn, padding='same', input_shape=input_shape))
23 model.add(complexnn.conv.ComplexConv2D(2**(x+1), (3, 3), strides=(2,2)
    , activation=act_fn, padding='same'))
24 model.add(complexnn.bn.ComplexBatchNormalization())
25 model.add(complexnn.conv.ComplexConv2D(2**(x+2), (3, 3), strides=(2,2)
    , activation=act_fn, padding='same'))
26 model.add(complexnn.bn.ComplexBatchNormalization())
27 model.add(complexnn.conv.ComplexConv2D(2**(x+3), (3, 3), strides=(2,2)
    , activation=act_fn, padding='same'))
28 model.add(complexnn.bn.ComplexBatchNormalization())
29 model.add(complexnn.conv.ComplexConv2D(2**(x+4), (3, 3), strides=(2,2)
    , activation=act_fn, padding='same'))
30 model.add(complexnn.bn.ComplexBatchNormalization())
31
32 model.add(complexnn.conv.ComplexConv2D(2**(x+5), (3, 3), activation=
    act_fn, padding='same'))
33
34 model.add(complexnn.conv.ComplexConv2D(2**(x+4), (3, 3), strides=(2,2)
    , transposed=True, activation=act_fn, padding='same'))
35 model.add(complexnn.bn.ComplexBatchNormalization())
36 model.add(complexnn.conv.ComplexConv2D(2**(x+3), (3, 3), strides=(2,2)
    , transposed=True, activation=act_fn, padding='same'))
37 model.add(complexnn.bn.ComplexBatchNormalization())
38 model.add(complexnn.conv.ComplexConv2D(2**(x+2), (3, 3), strides=(2,2)
    , transposed=True, activation=act_fn, padding='same'))
39 model.add(complexnn.bn.ComplexBatchNormalization())
40 model.add(complexnn.conv.ComplexConv2D(2**(x+1), 3, strides=(2,2),
    transposed=True, activation = act_fn, padding = 'same'))
41 model.add(complexnn.bn.ComplexBatchNormalization())
42 model.add(complexnn.conv.ComplexConv2D(2**(x+1), 3, activation =
    act_fn, padding = 'same'))
43 model.add(complexnn.conv.ComplexConv2D(1 , 3, activation = act_fn,
    padding = 'same'))
44 return model
45
46 def RAE(input_shape=None,x=0):
47
48 model = models.Sequential()
49 model.add(layers.Conv2D(2**(x+1), (3, 3), activation=act_fn, padding='
    same', kernel_initializer = 'he_normal', input_shape=input_shape))
50 model.add(layers.Conv2D(2**(x+1), (3, 3), activation=act_fn, strides
    =(2,2), padding='same', kernel_initializer = 'he_normal'))
51 model.add(layers.BatchNormalization())
52 model.add(layers.Conv2D(2**(x+2), (3, 3), activation=act_fn, strides
    =(2,2), padding='same', kernel_initializer = 'he_normal'))
53 model.add(layers.BatchNormalization())
54 model.add(layers.Conv2D(2**(x+3), (3, 3), activation=act_fn, strides
    =(2,2), padding='same', kernel_initializer = 'he_normal'))
55 model.add(layers.BatchNormalization())
56 model.add(layers.Conv2D(2**(x+4), (3, 3), activation=act_fn, strides
    =(2,2), padding='same', kernel_initializer = 'he_normal'))

```

```

57     model.add(layers.BatchNormalization())
58     model.add(layers.Conv2D(2**(x+5), (3, 3), activation=act_fn, padding='
    same', kernel_initializer = 'he_normal'))
59     model.add(layers.Conv2DTranspose(2**(x+4), (3, 3), activation=act_fn,
    strides=(2,2), padding='same', kernel_initializer = 'he_normal'))
60     model.add(layers.BatchNormalization())
61     model.add(layers.Conv2DTranspose(2**(x+3), (3, 3), activation=act_fn,
    strides=(2,2), padding='same', kernel_initializer = 'he_normal'))
62     model.add(layers.BatchNormalization())
63     model.add(layers.Conv2DTranspose(2**(x+2), (3, 3), activation=act_fn,
    strides=(2,2), padding='same', kernel_initializer = 'he_normal'))
64     model.add(layers.BatchNormalization())
65     model.add(layers.Conv2DTranspose(2**(x+1), 3, activation=act_fn,
    strides=(2,2), padding='same', kernel_initializer = 'he_normal'))
66     model.add(layers.BatchNormalization())
67     model.add(layers.Conv2D(2**(x+1), 3, activation=act_fn, padding='same'
    , kernel_initializer = 'he_normal'))
68     model.add(layers.Conv2D(1, (3, 3), activation=act_fn, padding='same',
    kernel_initializer = 'he_normal'))
69     return model
70
71 def par_train(X_train,X_test,model,filename):
72     par_model = multi_gpu_model(model, gpus=4)
73
74     par_model.compile(optimizer=optimizers.Adam(), loss='mse', metrics=["
    mae"])
75
76     csv_cb = keras.callbacks.CSVLogger('../logs/'+filename+'.csv')
77
78     par_model.fit(X_train,
79                 X_train,
80                 epochs=100,
81                 verbose=2,
82                 batch_size=16,
83                 shuffle=True,
84                 validation_data=(X_test, X_test),
85                 callbacks=[csv_cb])
86
87     par_model.save(filename+'.hd5')
88
89 def print_summary(model, filename):
90     with open('../descriptions/'+filename + '_report.txt','w') as fh:
91         # Pass the file handle in as a lambda function to make it callable
92         model.summary(print_fn=lambda x: fh.write(x + '\n'))
93
94
95 print('=====\n===Data Loading=====\n=====')
96 cmplx = np.concatenate([np.load("../patch_data/x_cmplx.npy"), np.load("../
    patch_data/i_cmplx.npy")])
97 real = np.concatenate([np.expand_dims(np.load("../patch_data/i_real.npy"),
    axis=3), np.expand_dims(np.load("../patch_data/x_real.npy"), axis=3)])
98

```

```

99 print('=====\n==Data Splitting==\n=====')
100 X_train_cmplx, X_test_cmplx, X_train_real, X_test_real, = train_test_split
    (cmplx, real, test_size=0.25, random_state=42)
101 del(cmplx)
102 del(real)
103
104 print('=====\n===Print Summary===\n=====')
105
106 print_summary(CAE((64,64,2),0), 'cmplx_mini')
107 print_summary(CAE((64,64,2),1), 'cmplx_small')
108 print_summary(CAE((64,64,2),2), 'cmplx_big')
109 print_summary(RAE((64,64,1),1), 'real_mini')
110 print_summary(RAE((64,64,1),2), 'real_small')
111 print_summary(RAE((64,64,1),3), 'real_big')
112
113
114 print('=====\n=====Train=====')
115 for common_seed in [33,42,12345,914872,552926,175937,528286]:
116     #for common_seed in [33,]:
117         from tensorflow import set_random_seed
118         np.random.seed(common_seed)
119         set_random_seed(common_seed)
120
121         import complexnn
122
123         import keras
124         from keras import models
125         from keras import layers
126         from keras import optimizers
127         from keras.utils import multi_gpu_model
128
129         #print('=====\n=====Train R 0=====\n=====')
130         #par_train(X_train_real, X_test_real,RAE((None,None,1),1),'real_mini')
            # Real Mini
131         print('=====\n=====Train R S=====\n=====')
132         par_train(X_train_real, X_test_real,RAE((None,None,1),2),'real_small')
            # Real Small
133         print('=====\n=====Train C S=====\n=====')
134         par_train(X_train_cmplx, X_test_cmplx,CAE((None,None,2),1),'
            cmplx_small') # Complex Small
135         print('=====\n=====Train C L=====\n=====')
136         par_train(X_train_cmplx, X_test_cmplx,CAE((None,None,2),2),'cmplx_big'
            ) # Complex Large
137         print('=====\n=====Train R L=====\n=====')
138         par_train(X_train_real, X_test_real,RAE((None,None,1),3),'real_large')
            # Real Large
139         #print('=====\n=====Train C 0=====\n=====')
140         #par_train(X_train_cmplx, X_test_cmplx,CAE((None,None,2),0),'
            cmplx_mini') # Complex Mini

```

```

1 import keras

```

```

2 import numpy as np
3 import complexnn
4
5 from sklearn.model_selection import train_test_split
6
7 CCAE = keras.models.load_model(
8     "cmplx_big33.hd5",
9     custom_objects={
10         "ComplexConv2D": complexnn.conv.ComplexConv2D,
11         "ComplexBatchNormalization": complexnn.bn.
12             ComplexBatchNormalization,
13     },
14 )
15 sCCAЕ = keras.models.load_model(
16     "cmplx_small33.hd5",
17     custom_objects={
18         "ComplexConv2D": complexnn.conv.ComplexConv2D,
19         "ComplexBatchNormalization": complexnn.bn.
20             ComplexBatchNormalization,
21     },
22 )
23
24 RCAE = keras.models.load_model("real_small33.hd5")
25 bRCAE = keras.models.load_model("real_big33.hd5")
26
27
28 big_seismic = np.rot90(np.load("../data/test_once/test1_seismic.npy")
29     [0:1,:,:], axes=(1,2), k=3)
30
31 from scipy.signal import hilbert
32 tmp_complex = hilbert(np.squeeze(big_seismic), axis=0)
33 big_complex = np.expand_dims(np.stack([np.real(tmp_complex), np.imag(
34     tmp_complex)],axis=2),axis=0)
35
36 bc_pred = np.squeeze(CCAE.predict(big_complex))[:255,:701]
37 sc_pred = np.squeeze(sCCAЕ.predict(big_complex))[:255,:701]
38
39 sr_pred = np.squeeze(RCAE.predict(np.expand_dims(big_seismic,axis=3)))
40     [:255,:701]
41 br_pred = np.squeeze(bRCAE.predict(np.expand_dims(big_seismic,axis=3)))
42     [:255,:701]
43
44 np.savez('predictions.npz', truth=big_seismic[0], small_complex=sc_pred[:,
45     :, 0], big_complex=bc_pred[:, :, 0], small_real=np.squeeze(sr_pred),
46     big_real=np.squeeze(br_pred))

```

```

1 import numpy as np
2 from scipy import fftpack

```



```

3 import matplotlib.pyplot as plt
4 import matplotlib.patches as patches
5
6
7 # Load data
8 keys = ['truth', 'small_complex', 'small_real', 'big_complex', 'big_real']
9
10 with np.load("predictions.npz") as data:
11     x,y = data['truth'].shape
12     seismic = np.zeros((x,y,5))
13     for i, q in enumerate(keys):
14         seismic[:, :, i] = data[q]
15
16 # Prepare Patches
17 f3_offset = 0.832 # Top crop of data
18
19 bottom_xt = [350,675,130,250]
20 top_xt = [50,300,30,100]
21 silent_xt = [500, 695,70, 120]
22
23 bottom = seismic[bottom_xt[2]:bottom_xt[3],bottom_xt[0]:bottom_xt[1],:]
24 top = seismic[top_xt[2]:top_xt[3],top_xt[0]:top_xt[1],:]
25 silent = seismic[silent_xt[2]:silent_xt[3],silent_xt[0]:silent_xt[1],:]
26
27 # Generate annotated ground truth
28 fig, ax = plt.subplots(figsize=(20,7))
29 im = ax.imshow(seismic[:, :, 0], vmin=-1, vmax=1, aspect='auto', extent=[0,
    seismic.shape[1]*25, seismic.shape[0]*.004+f3_offset, f3_offset])
30 ax.set_title("Full Seismic Data")
31 ax.set_ylabel("Time [s]")
32 ax.set_xlabel("Offset [m]")
33
34 bc = 'k'
35 tc = 'w'
36 sc = 'r'
37
38 brect = patches.Rectangle((bottom_xt[0]*25,bottom_xt[2]*0.004+f3_offset),(
    bottom_xt[1]-bottom_xt[0])*25,(bottom_xt[3]-bottom_xt[2])*0.004,
    linewidth=1.25, edgecolor=bc,facecolor='none')
39 trect = patches.Rectangle((top_xt[0]*25,top_xt[2]*0.004+f3_offset),(top_xt
    [1]-top_xt[0])*25,(top_xt[3]-top_xt[2])*0.004,linewidth=1, edgecolor=tc
    ,facecolor='none')
40 srect = patches.Rectangle((silent_xt[0]*25,silent_xt[2]*0.004+f3_offset),(
    silent_xt[1]-silent_xt[0])*25,(silent_xt[3]-silent_xt[2])*0.004,
    linewidth=1, edgecolor=sc,facecolor='none')
41
42 # Add the patch to the Axes
43 ax.add_patch(brect)
44 ax.add_patch(trect)
45 ax.add_patch(srect)
46
47 ax.annotate('Top', ((top_xt[0]+10)*25,f3_offset+(top_xt[2]+10)*0.004),

```

```

        color=tc, weight='bold', fontsize=16, ha='left', va='center')
48 ax.annotate('Bottom', ((bottom_xt[0]+10)*25,f3_offset+(bottom_xt[2]+10)
    *0.004), color=bc, weight='bold', fontsize=16, ha='left', va='center')
49 ax.annotate('Silent', ((silent_xt[0]+10)*25,f3_offset+(silent_xt[2]+10)
    *0.004), color=sc, weight='bold', fontsize=16, ha='left', va='center')
50
51 fig.colorbar(im, ax=ax)
52 fig.tight_layout()
53 # fig.show()
54 fig.savefig(f"figures/seismic.png", bbox_inches='tight', dpi=200)
55
56 plt.imshow(seismic[:, :, 0])
57
58 plt.imshow(seismic[50:150, 500:, 0])
59
60
61 def rms(a,b):
62     return np.sqrt(np.power(a-b,2).mean())
63
64 def mae(a,b):
65     return np.abs(a-b).mean()
66
67 def format_seis(data, keys, keyword=""):
68     for i in range(data.shape[-1]):
69         print(f"{keys[i]:13s} {keyword}:\t | RMS: {rms(data[:, :, 0], data
           [:, :, i]):.4f}\t | MAE: {mae(data[:, :, 0], data[:, :, i]):.4f}")
70
71 def plot_seis(data, keys, size=(20,7), keyword="", xtent=None):
72     for i in range(data.shape[-1]):
73         fig, ax = plt.subplots(figsize=size)
74         if xtent:
75             im = ax.imshow(data[:, :, i], vmin=-1, vmax=1, aspect='auto',
                extent=[xtent[0]*25,xtent[1]*25,xtent[3]*.004+f3_offset,
                xtent[2]*.004+f3_offset])
76         else:
77             im = ax.imshow(data[:, :, i], vmin=-1, vmax=1, aspect='auto',
                extent=[0,data.shape[1]*25,data.shape[0]*.004+f3_offset,
                f3_offset])
78         ax.set_title(keys[i].replace("_", " ").title())
79         ax.set_ylabel("Time [s]")
80         ax.set_xlabel("Offset [m]")
81         fig.colorbar(im, ax=ax)
82         fig.tight_layout()
83         # fig.show()
84         fig.savefig(f"figures/{keyword}_{keys[i]}.png", bbox_inches='tight
            ', dpi=200)
85
86 def plot_fk(data, keys, size=(5,5), keyword=""):
87     for i in range(data.shape[-1]):
88         M, N = data[:, :, i].shape
89         fft2 = fftpack.fft2(data[:, :, i])
90         f_mag = np.abs(fft2)

```

```

91     f_mag = fftpack.fftfreq(f_mag)
92     f_mag = np.log(1 + f_mag)
93     fig, ax = plt.subplots(figsize=size)
94     q, p = fftpack.fftfreq(M, d=.004), fftpack.fftfreq(N, d=25/1000)
95     im = ax.imshow(f_mag[:M//2,N//2:9*N//16], aspect='auto', vmin=0,
96                   vmax=8,
97                   extent=(0, max(p)/8, 0, max(q)))
98     ax.set_title(keys[i].replace("_", " ").title())
99     ax.set_ylabel("Frequency [Hz]")
100    ax.set_xlabel("Wavenumber [km$^{-1}$]")
101    fig.colorbar(im, ax=ax)
102    fig.tight_layout()
103    # fig.show()
104    fig.savefig(f"figures/{keyword}_fk_{keys[i]}.png", bbox_inches='
105               tight', dpi=200)
106
107 def plot_fk_diff(data, keys, size=(5,5), keyword=""):
108     for i in range(1,data.shape[-1]):
109         M, N = data[:, :, i].shape
110         fft2 = fftpack.fft2(data[:, :, i])
111         f_mag = np.abs(fft2) - np.abs(fftpack.fft2(data[:, :, 0]))
112         f_mag = fftpack.fftfreq(f_mag)
113         #f_mag = np.log(1 + f_mag)
114         vm = np.abs(f_mag).max()
115         q, p = fftpack.fftfreq(M, d=.004), fftpack.fftfreq(N, d=25/1000)
116         fig, ax = plt.subplots(figsize=size)
117         im = ax.imshow(f_mag[:M//2,:], aspect='auto', cmap='RdBu', vmin=-
118                       vm, vmax=vm,
119                       extent=(min(p), max(p), 0, max(q)))
120         ax.set_title(keys[i].replace("_", " ").title())
121         ax.set_ylabel("Frequency [Hz]")
122         ax.set_xlabel("Wavenumber [km$^{-1}$]")
123         fig.colorbar(im, ax=ax)
124         fig.tight_layout()
125         # fig.show()
126         fig.savefig(f"figures/{keyword}_fk_diff_{keys[i]}.png",
127                   bbox_inches='tight', dpi=200)
128
129 print(seismic.shape[1], [x*seismic.shape[1]//y for x,y in [(1,2), (3,4),
130                   (5,8), (9,16)]], [x*seismic.shape[1]//y-seismic.shape[1]//2 for x,y in
131                   [(1,2), (3,4), (5,8), (9,16)]])
132
133 # Let's first calculate the rms and mae on the full seismic and the
134 cutouts.
135 format_seis(seismic, keys, "full")
136 format_seis(bottom, keys, "bottom")
137 format_seis(top, keys, "top")
138 format_seis(silent, keys, "silent")
139
140 # Plot seismic
141 plot_seis(seismic, keys, (20,7), "full")
142 plot_seis(bottom, keys, (10,6), "bottom", bottom_xt)

```

```
136 plot_seis(top,keys,(10,6), "top", top_xt)
137 plot_seis(silent,keys,(10,6), "silent", silent_xt)
138
139 # Plot FK representation of Seismic
140 plot_fk(seismic,keys,keyword="full")
141 plot_fk(top,keys,keyword="top")
142 plot_fk(bottom,keys,keyword="bottom")
143 plot_fk(silent,keys,keyword="silent")
144
145 # Plot difference of FK images
146 plot_fk_diff(seismic,keys,keyword="full")
147 plot_fk_diff(top,keys,keyword="top")
148 plot_fk_diff(bottom,keys,keyword="bottom")
149 plot_fk_diff(silent,keys,keyword="silent")
```

E.4 Machine Learning in 4D Seismic Inversion

E.4.1 4D Neural Network Inversion Training

This notebook generates a model for the notebook in ['02 - 4D-Inversion-Field-Data'](02 - 4D-Inversion-Field-Data).

```
1 import os
2
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5
6 import pandas as pd
7 import numpy as np
8
9 from scipy.io import loadmat, savemat
10
11 from sklearn.model_selection import train_test_split
12 %load_ext autoreload
13 %autoreload 2
```

```
1 from numpy.random import seed
2 seed(42)
3 from tensorflow import set_random_seed
4 set_random_seed(42)
```

E.4.1.1 Data Loading

The data was provided in Matlab data format. It is available as Pressure, Gas Saturation and Water Saturation maps together and to increase training data, also available as isolated effects. Additionally, pore volume maps turned out to be essential for a successful inversion.

Each map contains values for ΔP , ΔS_g , ΔS_w maps and simulated seismic difference maps for near, mid and far in the 'dSNA_syn_' words.

```

1 def data_gen_train_full():
2     seed(42)
3     set_random_seed(42)
4     location = "data"
5     filename = "Seis2PS_NN_training_input"
6     suffixes = ["", "_Ponly", "_SGonly", "_SWonly"]
7     #suffixes = [""]
8     out = pd.DataFrame()
9     for suff in suffixes:
10        seis_ps = loadmat(os.path.join(location, filename+suff))
11        headers = ["dP", "dSg", "dSw", "dSNA_syn_nr", "dSNA_syn_md", "
12                  dSNA_syn_fr"]
13        print(suff)
14        for q in range(len(seis_ps["dSg"][0])):
15            tost = pd.DataFrame()
16            for x in headers:
17                tost[x] = seis_ps[x][0][q].ravel()
18            if suff == "":
19                pore = "Pore_volume"
20                pore_data = seis_ps[pore].ravel()
21                tost[pore] = pore_data
22            else:
23                tost[pore] = pore_data
24            out = out.append(tost)
25        out = out.dropna()
26        out = out.loc[~(out[["dP", "dSg", "dSw"]]==0).all(axis=1)]
27        out = out.sample(frac=1).reset_index(drop=True)
28
29        y_train = out[["dP", "dSg", "dSw"]]
30        X_train = out[["dSNA_syn_nr", "dSNA_syn_md", "dSNA_syn_fr", "
31                      Pore_volume"]]
32
33    return X_train, y_train

```

```

1 def data_gen_test(q):
2     seed(42)
3     set_random_seed(42)
4     location = "data"
5     filename = "Seis2PS_NN_training_input"
6     seis_ps = loadmat(os.path.join(location, filename))
7     headers = ["dP", "dSg", "dSw", "dSNA_syn_nr", "dSNA_syn_md", "
8               dSNA_syn_fr"]

```

```

8     tost = pd.DataFrame()
9     for x in headers:
10         tost[x] = seis_ps[x][0][q].ravel()
11
12     pore = "Pore_volume"
13     pore_data = seis_ps[pore].ravel()
14     tost[pore] = pore_data
15
16     out = tost.dropna()
17     out = out.loc[~(out[["dP", "dSg", "dSw"]]==0).all(axis=1)]
18
19     y_train = out[["dP", "dSg", "dSw"]]
20     X_train = out[["dSNA_syn_nr", "dSNA_syn_md", "dSNA_syn_fr", "
        Pore_volume"]]
21
22     return X_train, y_train

```

E.4.1.2 Model Building

The initial architecture was generated from an encoder decoder architecture using ‘hyperas’ to optimize width, depth and dropout rate for predicting one synthetic map from the other time steps.

```

1 import tensorflow as tf
2 import keras
3 import keras.backend as K
4 from keras.models import Model
5 from keras.layers import Input, Dense, AlphaDropout, Dropout, Lambda,
    GaussianNoise, BatchNormalization, Concatenate
6 from keras import regularizers
7 from keras import optimizers
8 from keras import callbacks
9
10 from keras_tqdm import TQDMNotebookCallback
11
12 #from hyperopt import Trials, STATUS_OK, tpe
13 #from hyperas import optim
14 #from hyperas.distributions import choice, quniform, uniform, loguniform
15
16 from sklearn.metrics import r2_score
17 %autoreload 2

```

```

1 def r_square(y_true, y_pred):
2     from keras import backend as K
3     SS_res = K.sum(K.square(y_true - y_pred))
4     SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
5     return (1 - SS_res/(SS_tot + K.epsilon()))

```

```

6
7 def r_square_loss(y_true, y_pred):
8     SS_res = K.sum(K.square(y_true - y_pred))
9     SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
10    return 1 - (1 - SS_res/(SS_tot + K.epsilon()))
11
12 def huber_loss(y_true, y_pred, clip_delta=.35):
13     error = y_true - y_pred
14     cond = tf.keras.backend.abs(error) < clip_delta
15     squared_loss = 0.5 * tf.keras.backend.square(error)
16     linear_loss = clip_delta * (tf.keras.backend.abs(error) - 0.5 *
17         clip_delta)
18
19    return tf.where(cond, squared_loss, linear_loss)

```

```

1 def sampling(args):
2     """Reparameterization trick by sampling fr an isotropic unit Gaussian.
3     # Arguments
4         args (tensor): mean and log of variance of Q(z|X)
5     # Returns
6         z (tensor): sampled latent vector
7     """
8
9     z_mean, z_log_var = args
10    batch = K.shape(z_mean)[0]
11    dim = K.int_shape(z_mean)[1]
12    # by default, random_normal has mean=0 and std=1.0
13    epsilon = K.random_normal(shape=(batch, dim))
14    return z_mean + K.exp(0.5 * z_log_var) * epsilon
15
16 def build_vae(near_off mid_off, far_off, noise=0.01, lr=1e-4):
17     seed(42)
18     set_random_seed(42)
19
20     alpha_dropout = 0.20
21     encoding_dims = 256
22     growth_factor = 1
23
24     layers = 4
25
26
27     mid_near_offset = mid-near
28     far_near_offset = far-near
29     far_mid_offset = far-mid
30
31     near = Input(shape=(1,), name="near_input")
32     mid = Input(shape=(1,), name="mid_input")
33     far = Input(shape=(1,), name="far_input")
34     pore = Input(shape=(1,), name="pore_input")
35     noisy_near = GaussianNoise(noise)(near)
36     noisy_mid = GaussianNoise(noise)(mid)

```



```

37     noisy_far = GaussianNoise(noise)(far)
38     mid_near = Lambda(lambda inputs: ( inputs[0] - inputs[1] ) / (
        mid_near_offset ))([noisy_mid, noisy_near])
39     far_near = Lambda(lambda inputs: ( inputs[0] - inputs[1] ) / (
        far_near_offset ))([noisy_far, noisy_near])
40     far_mid = Lambda(lambda inputs: ( inputs[0] - inputs[1] ) / (
        far_mid_offset ))([noisy_far, noisy_mid])
41
42     input_gradient = Concatenate()([noisy_near, noisy_mid, noisy_far,
        mid_near, far_near, far_mid, pore])
43
44     encoded = Dense(encoding_dims*growth_factor*layers, activation="relu",
        name="encoder_0")(input_gradient)
45     encoded = Dropout(alpha_dropout)(encoded)
46     #encoded = BatchNormalization()(encoded)
47
48     for q in range(layers):
49         encoded = Dense(encoding_dims*growth_factor*(layers-q), activation
            ="relu", name="encoder_"+str(q+1))(encoded)
50         encoded = Dropout(alpha_dropout)(encoded)
51         #encoded = BatchNormalization()(encoded)
52
53     z_mean = Dense(encoding_dims, name='z_mean')(encoded)
54     z_log_var = Dense(encoding_dims, name='z_log_var')(encoded)
55
56     # use reparameterization trick to push the sampling out as input
57     # note that "output_shape" isn't necessary with the TensorFlow backend
58     deep_down = Lambda(sampling, name='z')([z_mean, z_log_var])
59
60     decoded0 = Dense(encoding_dims*growth_factor, activation="relu", name=
        "decoder_0")(deep_down)
61
62     for q in range(2, layers):
63         #decoded0 = BatchNormalization()(decoded0)
64         decoded0 = Dropout(alpha_dropout)(decoded0)
65         decoded0 = Dense(encoding_dims*growth_factor*q, activation="relu",
            name="decoder_"+str(q-1))(decoded0)
66
67     output0 = Dense(encoding_dims*growth_factor*layers, activation="linear
        ")(decoded0)
68     dP = Dense(1, activation="linear", name="dP")(output0)
69     output1 = Dense(encoding_dims*growth_factor*layers, activation="linear
        ")(decoded0)
70     dSw = Dense(1, activation="linear", name="dSw")(output1)
71     output2 = Dense(encoding_dims*growth_factor*layers, activation="linear
        ")(decoded0)
72     dSg = Dense(1, activation="linear", name="dSg")(output2)
73
74     model = Model(inputs=[near, mid, far, pore], output=[dP, dSw, dSg])
75
76     model.compile(loss="mse", optimizer=optimizers.Nadam(lr=lr), metrics=[
        r_square])

```

```

77
78     model.summary()
79
80     return model

```

```

1 X_train, y_train = data_gen_train_full()

```

```

1 model = build_vae(10,20,30,noise=0.00,lr=5e-4)

```

```

1 earlystop0 = callbacks.EarlyStopping(monitor='val_loss',min_delta=0,
    patience=5,verbose=1,mode='auto', restore_best_weights=True)
2 earlystop1 = callbacks.EarlyStopping(monitor='val_loss',min_delta=0,
    patience=11,verbose=1,mode='auto', restore_best_weights=True)
3 earlystop2 = callbacks.EarlyStopping(monitor='val_loss',min_delta=0,
    patience=51,verbose=1,mode='auto', restore_best_weights=False)
4 ir1 = callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience
    =5, verbose = 1, min_lr=0)
5 ir2 = callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience
    =15, cooldown=10, verbose = 1, min_lr=0)

```

E.4.1.3 Pre-Training without Noise

Clearly the network should converge on the model data.

```

1 result = model.fit(
2     [X_train["dSNA_syn_nr"],X_train["dSNA_syn_md"],X_train["dSNA_syn_fr"],
3     X_train["Pore_volume"]],
4     [y_train["dP"],y_train["dSw"],y_train["dSg"]],
5     batch_size= 400,
6     epochs=2000,
7     verbose=0,
8     shuffle=True,
9     validation_split=0.01,
10    callbacks = [TQDMNotebookCallback(leave_inner=True,leave_outer=True),
        earlystop1,ir1],

```

```

1 model.save("pre.hd5")

```

E.4.1.4 Fine-Tuning On Noisy Data

The final model has to expect noisy data to be able to transfer it to field data, we therefore rebuild the model and load the weights from the pre-trained network. This model will not converge particularly nor will it perform well on the synthetic data.

```

1 model = build_vae(noise=0.02,lr=1e-4)
2 model.load_weights("pre.hd5")
3 model.compile(loss="mse", optimizer=optimizers.Nadam(lr=1e-4), metrics=[
4     r_square])
5 result = model.fit(
6     [X_train["dSNA_syn_nr"],X_train["dSNA_syn_md"],X_train["dSNA_syn_fr"],
7     X_train["Pore_volume"]],
8     [y_train["dP"],y_train["dSw"],y_train["dSg"]],
9     batch_size= 500,
10    epochs=2000,
11    verbose=0,
12    shuffle=True,
13    validation_split=0.01,
14    callbacks = [TQDMNotebookCallback(leave_inner=True,leave_outer=True),
15                earllystop2,ir2],
16 )
17 model.save("bbest.hd5")

```

```

1 experiment = "publication"

```

```

1 model = keras.models.load_model("best.hd5", custom_objects={'r_square':
2     r_square, 'huber_loss': huber_loss})

```

E.4.1.5 Test

Generate the test data and evaluate the model on this data. You may notice that the "test" data is contained within the train data, which would be relevant, if the actual evaluation was done on the synthetic data. During the build phase, this map would be excluded from the train data to get valid results. (This is important.)

```

1 X_test, y_test = data_gen_test(3)

```

```

1 print("Evalutation on unseen data:")
2 print(model.evaluate([X_test["dSNA_syn_nr"],X_test["dSNA_syn_md"],X_test["
3     dSNA_syn_fr"],X_test["Pore_volume"]], [y_test["dP"],y_test["dSw"],
4     y_test["dSg"]]))

```

```

1 %%timeit
2 preddata = model.predict([X_test["dSNA_syn_nr"],X_test["dSNA_syn_md"],
    X_test["dSNA_syn_fr"],X_test["Pore_volume"]])

```

E.4.1.6 Plotting

The plotting rearranges the sample-wise prediction into the original map shape and deletes predictions, where data is not available.

‘model_shape’ saves a mask of the data, we apply this map to the predictions to get rid of predictions on samples that did not contain actual data.

```

1 location = "data"
2 filename = "Seis2PS_NN_training_input"
3 q=3
4 pore_volume = loadmat(os.path.join(location,filename))["Pore_volume"]
5 seis_ps = loadmat(os.path.join(location,filename))
6 ravel_data = seis_ps["dSNA_syn_nr"][0][q].ravel()
7 model_shape = ~np.isnan(ravel_data)
8
9 plt.figure(figsize=(15,20))
10 plt.subplot(2, 2, 1)
11 vmax=np.max(np.abs(seis_ps["dSNA_syn_nr"][0][q]))
12 plt.imshow(seis_ps["dSNA_syn_nr"][0][q], cmap="seismic_r", vmin=-vmax,
    vmax=vmax, aspect='auto')
13 plt.colorbar()
14 plt.title("Near")
15 plt.subplot(2, 2, 2)
16 vmax=np.max(np.abs(seis_ps["dSNA_syn_md"][0][q]))
17 plt.imshow(seis_ps["dSNA_syn_md"][0][q], cmap="seismic_r", vmin=-vmax,
    vmax=vmax, aspect='auto')
18 plt.colorbar()
19 plt.title("Mid")
20 plt.subplot(2, 2, 3)
21 vmax=np.max(np.abs(seis_ps["dSNA_syn_fr"][0][q]))
22 plt.imshow(seis_ps["dSNA_syn_fr"][0][q], cmap="seismic_r", vmin=-vmax,
    vmax=vmax, aspect='auto')
23 plt.colorbar()
24 plt.title("Far")
25 plt.subplot(2, 2, 4)
26 vmax=np.nanmax(seis_ps["Pore_volume"])
27 plt.imshow(seis_ps["Pore_volume"], cmap="seismic_r", vmin=-vmax, vmax=vmax,
    , aspect='auto')
28 plt.colorbar()
29 plt.title("Pore Volume")
30 plt.savefig("Seismic-input.png")
31 plt.show()

```

```

1 data = np.full_like(seis_ps["dSg"][0][0], np.nan)
2 counter = 0
3 for i, m in enumerate(model_shape):
4     if m:
5         data[np.unravel_index(i, data.shape)] = preddata[0][counter]
6         counter += 1
7 vmax = np.nanmax([np.abs(data), np.abs(seis_ps["dP"][0][q])])
8 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20,10))
9
10 im = axes[0].imshow(data, cmap="seismic", vmin=-vmax, vmax=vmax, aspect='
    equal')
11 axes[0].set_title("Neural Network dP "+experiment)
12
13 im = axes[1].imshow(seis_ps["dP"][0][q], cmap="seismic", vmin=-vmax, vmax=
    vmax, aspect='equal')
14 axes[1].set_title("Test Data dP")
15
16 fig.tight_layout()
17 fig.subplots_adjust(right=0.8)
18 cbar_ax = fig.add_axes([0.82, 0.2, 0.01, 0.6])
19 fig.colorbar(im, cax=cbar_ax)
20
21 blerg = "dP"
22 mat_dict = {blerg+"_nn_data": data, blerg: seis_ps[blerg][0][q]}
23 savemat('matlab/'+blerg+'.mat', mat_dict)
24
25 fig.savefig("Bestfit-dP-"+experiment+".png")
26 fig.show()

```

```

1 plt.figure(figsize=(10,10))
2 plt.imshow(data-seis_ps["dP"][0][q], cmap="seismic", vmin=-vmax, vmax=vmax
    )
3 plt.title("Misfit dP "+experiment)
4 plt.colorbar()
5 plt.tight_layout()
6 plt.savefig("Bestfit_diff-dP-"+experiment+".png")
7 plt.show()

```

```

1 data = np.full_like(seis_ps["dSg"][0][0], np.nan)
2 counter = 0
3 for i, m in enumerate(model_shape):
4     if m:
5         data[np.unravel_index(i, data.shape)] = preddata[1][counter]
6         counter += 1
7
8 vmax = np.nanmax([np.abs(data), np.abs(seis_ps["dSw"][0][q])])
9 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20,10))
10

```

```

11 im = axes[0].imshow(data, cmap="seismic_r", vmin=-vmax, vmax=vmax, aspect=
    'equal')
12 axes[0].set_title("Neural Network dSw "+experiment)
13
14 im = axes[1].imshow(seis_ps["dSw"][0][q], cmap="seismic_r", vmin=-vmax,
    vmax=vmax, aspect='equal')
15 axes[1].set_title("Test Data dSw")
16
17 fig.tight_layout()
18 fig.subplots_adjust(right=0.8)
19 cbar_ax = fig.add_axes([0.82, 0.2, 0.01, 0.6])
20 fig.colorbar(im, cax=cbar_ax)
21
22 blerg = "dSw"
23 mat_dict = {blerg+"_nn_data": data, blerg: seis_ps[blerg][0][q]}
24 savemat('matlab/'+blerg+'.mat', mat_dict)
25
26 fig.savefig("Bestfit-dSw-"+experiment+".png")
27 fig.show()

```

```

1 plt.figure(figsize=(10,10))
2 plt.imshow(data-seis_ps["dSw"][0][q], cmap="seismic_r", vmin=-vmax, vmax=
    vmax)
3 plt.title("Misfit dSw "+experiment)
4 plt.colorbar()
5 plt.tight_layout()
6 plt.savefig("Bestfit_diff-dSw-"+experiment+".png")
7 plt.show()

```

```

1 data = np.full_like(seis_ps["dSg"][0][0], np.nan)
2 counter = 0
3 for i, m in enumerate(model_shape):
4     if m:
5         data[np.unravel_index(i, data.shape)] = preddata[2][counter]
6         counter += 1
7
8 vmax = np.nanmax([np.abs(data), np.abs(seis_ps["dSg"][0][q])])
9 fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20,10))
10
11 im = axes[0].imshow(data, cmap="seismic_r", vmin=-vmax, vmax=vmax, aspect
    ='equal')
12 axes[0].set_title("Neural Network dSg "+experiment)
13
14 im = axes[1].imshow(seis_ps["dSg"][0][q], cmap="seismic_r", vmin=-vmax,
    vmax=vmax, aspect='equal')
15 axes[1].set_title("Test Data dSg")
16
17 fig.tight_layout()
18 fig.subplots_adjust(right=0.8)

```

```

19 cbar_ax = fig.add_axes([0.85, 0.2, 0.01, 0.6])
20 fig.colorbar(im, cax=cbar_ax)
21
22 blerg = "dSg"
23 mat_dict = {blerg+"_nn_data": data, blerg: seis_ps[blerg][0][q]}
24 savemat('matlab/'+blerg+'.mat', mat_dict)
25
26 fig.savefig("Bestfit-dSg-"+experiment+".png")
27 fig.show()

```

```

1 plt.figure(figsize=(10,10))
2 plt.imshow(data-seis_ps["dSg"][0][q], cmap="seismic_r", vmin=-vmax, vmax=
    vmax)
3 plt.title("Misfit dSg "+experiment)
4 plt.colorbar()
5 plt.tight_layout()
6 plt.savefig("Bestfit_diff-dSg-"+experiment+".png")
7 plt.show()

```

E.4.2 4D Inversion on Field Data

This notebook requires a trained model from the notebook in [‘01 - 4D-Pressure-Saturation-Inversion’](01 - 4D-Pressure-Saturation-Inversion.ipynb).

```

1 import os
2
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5
6 import pandas as pd
7 import numpy as np
8
9 from scipy.io import loadmat, savemat
10
11 from sklearn.model_selection import train_test_split
12 %load_ext autoreload
13 %autoreload 2

```

```

1 from numpy.random import seed
2 seed(42)
3 from tensorflow import set_random_seed
4 set_random_seed(42)

```


E.4.2.1 Data

This notebook evaluates the network on field data.

```

1 def data_gen(timestep):
2     seed(42)
3     set_random_seed(42)
4     #suffixes = [""]
5     location = "data"
6     filename = "Seis2PS_NN_training_input"
7     pore_volume = loadmat(os.path.join(location, filename))["Pore_volume"]
8     location = "data"
9     filename = "Seis2PS_NN_training_input_obs"
10    seis_ps = loadmat(os.path.join(location, filename))
11    pore_volume
12    headers = ["dSNA_nr", "dSNA_md", "dSNA_fr"]
13    out_test = pd.DataFrame()
14    tost = pd.DataFrame()
15    for x in headers:
16        ravel_data = seis_ps[x][0][timestep]
17        out_test[x] = ravel_data.ravel()
18
19    out_test["Pore_volume"] = pore_volume.ravel()
20
21
22
23    return out_test
24 data_gen(1).describe()

```

```

1 import tensorflow as tf
2 import keras
3 import keras.backend as K
4 from keras.models import Model
5 from keras.layers import Input, Dense, AlphaDropout, Dropout
6 from keras import regularizers
7 from keras import optimizers
8 from keras import callbacks
9
10 from keras_tqdm import TQDMNotebookCallback
11
12 from hyperopt import Trials, STATUS_OK, tpe
13 from hyperas import optim
14 from hyperas.distributions import choice, quniform, uniform, loguniform
15
16 from sklearn.metrics import r2_score
17 %autoreload 2

```

```

1 def r_square(y_true, y_pred):
2     from keras import backend as K

```

```

3     SS_res = K.sum(K.square(y_true - y_pred))
4     SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
5     return (1 - SS_res/(SS_tot + K.epsilon()))
6
7 def r_square_loss(y_true, y_pred):
8     SS_res = K.sum(K.square(y_true - y_pred))
9     SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
10    return 1 - (1 - SS_res/(SS_tot + K.epsilon()))
11
12 def huber_loss(y_true, y_pred, clip_delta=.35):
13     error = y_true - y_pred
14     cond = tf.keras.backend.abs(error) < clip_delta
15     squared_loss = 0.5 * tf.keras.backend.square(error)
16     linear_loss = clip_delta * (tf.keras.backend.abs(error) - 0.5 *
17                               clip_delta)
18
19    return tf.where(cond, squared_loss, linear_loss)

```

```

1 experiment = "publication"

```

E.4.2.2 Model Loading

We load the noise-trained model. We have to provide the custom_objects 'r_square', 'huber_loss' for the experiments.

```

1 model = keras.models.load_model("best.hd5", custom_objects={'r_square':
    r_square, 'huber_loss': huber_loss})

```

```

1 model.summary()

```

```

1 q = 3
2 X_test = data_gen(q)

```

```

1 preddata = model.predict(X_test)

```

E.4.2.3 Plotting

Plot the resulting inversion and the networks.

```

1 for q in range(3,4):
2     X_test = data_gen(q)
3     preddata = model.predict([X_test["dSNA_nr"],X_test["dSNA_md"],X_test["
4         dSNA_fr"],X_test["Pore_volume"]])
5     fig = plt.figure(figsize=(15, 10))
6     plt.subplot(231)
7     vmax=np.max(np.abs(seis_ps["dSNA_nr"][0][q]))
8     plt.imshow(seis_ps["dSNA_nr"][0][q], cmap="seismic_r", vmin=-vmax,
9         vmax=vmax, aspect='equal')
10    plt.colorbar()
11    plt.title("Near")
12    plt.subplot(232)
13    vmax=np.max(np.abs(seis_ps["dSNA_md"][0][q]))
14    plt.imshow(seis_ps["dSNA_md"][0][q], cmap="seismic_r", vmin=-vmax,
15        vmax=vmax, aspect='equal')
16    plt.colorbar()
17    plt.title("Mid")
18    plt.subplot(233)
19    vmax=np.max(np.abs(seis_ps["dSNA_fr"][0][q]))
20    plt.imshow(seis_ps["dSNA_fr"][0][q], cmap="seismic_r", vmin=-vmax,
21        vmax=vmax, aspect='equal')
22    plt.colorbar()
23    plt.title("Far")
24
25    ax0 = plt.subplot(234)
26    data = preddata[0].reshape(seis_ps["dSNA_md"][0][0].shape)
27    vmax = np.nanmax(np.abs(data))
28    im0 = ax0.imshow(data, cmap="seismic", vmin=-vmax, vmax=vmax, aspect='
29        equal')
30    ax0.set_title("Neural Network dP "+experiment)
31    fig.colorbar(im0, ax=ax0)
32
33    ax1 = plt.subplot(235)
34    data = preddata[1].reshape(seis_ps["dSNA_md"][0][0].shape)
35    vmax = np.nanmax(np.abs(data))
36    im1 = ax1.imshow(data, cmap="seismic_r", vmin=-vmax, vmax=vmax, aspect
37        ='equal')
38    ax1.set_title("Neural Network dSw")
39    fig.colorbar(im1, ax=ax1)
40
41    ax2 = plt.subplot(236)
42    data = preddata[2].reshape(seis_ps["dSNA_md"][0][0].shape)
43    vmax = np.nanmax(np.abs(data))
44    im2 = ax2.imshow(data, cmap="seismic", vmin=-vmax, vmax=vmax, aspect='
45        equal')
46    ax2.set_title("Neural Network dSg")
47    fig.colorbar(im2, ax=ax2)

```

```

43     blerg = "obs"
44     mat_dict = {blerg+"dP_nn_data": preddata[0].reshape(seis_ps["dSNA_md"
45         ] [0] [0].shape),
46                 blerg+"dSw_nn_data": preddata[1].reshape(seis_ps["dSNA_md"
47         ] [0] [0].shape),
48                 blerg+"dSg_nn_data": preddata[2].reshape(seis_ps["dSNA_md"
49         ] [0] [0].shape),
50                 blerg+"near": seis_ps["dSNA_nr"] [0] [q],
51                 blerg+"mid": seis_ps["dSNA_md"] [0] [q],
52                 blerg+"far": seis_ps["dSNA_fr"] [0] [q],}
53     savemat('matlab/'+blerg+'.mat', mat_dict)

    fig.savefig("Observed-vae-gradient-basemod2-timestep"+str(q)+".png")
    fig.show()

```

```

1 from keras.utils import plot_model
2 plot_model(model, to_file='model.png', show_shapes=True)

```

```

1 from IPython.display import SVG
2 from keras.utils.vis_utils import model_to_dot
3
4 SVG(model_to_dot(model).create(prog='dot', format='svg'))

```

```

1 X_test.shape

```

```

1 preddata[0].shape

```

```

1 location = "data"
2 filename = "Seis2PS_NN_training_input"
3 pore_volume = loadmat(os.path.join(location,filename))["Pore_volume"]

```

```

1 location = "data"
2 filename = "Seis2PS_NN_training_input_obs"
3 seis_ps = loadmat(os.path.join(location,filename))
4 headers = list(seis_ps.keys())[3:]
5 plt.imshow(pore_volume)

```

```
1 seis_ps["dSNA_nr"][0][0].shape
```

```
1 plt.figure(figsize=(15,20))
2 plt.subplot(2, 2, 1)
3 vmax=np.max(np.abs(seis_ps["dSNA_nr"][0][q]))
4 plt.imshow(seis_ps["dSNA_nr"][0][q], cmap="seismic_r", vmin=-vmax, vmax=
   vmax, aspect='auto')
5 plt.colorbar()
6 plt.title("Near")
7 plt.subplot(2, 2, 2)
8 vmax=np.max(np.abs(seis_ps["dSNA_md"][0][q]))
9 plt.imshow(seis_ps["dSNA_md"][0][q], cmap="seismic_r", vmin=-vmax, vmax=
   vmax, aspect='auto')
10 plt.colorbar()
11 plt.title("Mid")
12 plt.subplot(2, 2, 3)
13 vmax=np.max(np.abs(seis_ps["dSNA_fr"][0][q]))
14 plt.imshow(seis_ps["dSNA_fr"][0][q], cmap="seismic_r", vmin=-vmax, vmax=
   vmax, aspect='auto')
15 plt.colorbar()
16 plt.title("Far")
17 plt.subplot(2, 2, 4)
18 vmax=np.nanmax(seis_ps["Pore_volume"])
19 plt.imshow(seis_ps["Pore_volume"], cmap="seismic_r", vmin=-vmax, vmax=vmax
   , aspect='auto')
20 plt.colorbar()
21 plt.title("Pore Volume")
22 plt.savefig("Seismic-input.png")
23 plt.show()
```

```
1 import click
2 import glob
3 from tqdm import tqdm
4 import segyio
5 import numpy as np
6 from numpy.lib import stride_tricks
7 from time import time
8
9 def cutup(data, blk, strd):
10     sh = np.array(data.shape)
11     blk = np.asarray(blk)
12     strd = np.asarray(strd)
13     nbl = (sh - blk) // strd + 1
14     strides = np.r_[data.strides * strd, data.strides]
15     dims = np.r_[nbl, blk]
16     data6 = stride_tricks.as_strided(data, strides=strides, shape=dims)
17     return data6#.reshape(-1, *blk)
18
```

```

19 for filename in glob.iglob("../data/*.sgy"):
20     start_time = time()
21     numpy_path = filename[:-4]+"_256.npy"
22     cube_path = filename[:-4]+"_cube.npy"
23     print(f"=== Converting {filename[:-4].split('/')[1]} ===")
24     print("Opening file")
25     with segyio.open(filename, 'r', strict=False) as segy:
26         print("Precomputing Statistics")
27         f_time = time()
28         ilines=np.unique(segy.attributes(segyio.TraceField.INLINE_3D)[:])
29         mi = min(ilines)
30         mia = max(ilines)
31         il = len(np.unique(ilines))
32         xlines=np.unique(segy.attributes(segyio.TraceField.CROSSLINE_3D)
33                             [:])
34         mx = min(xlines)
35         mxa = max(xlines)
36         xl = len(np.unique(xlines))
37         out = np.full((il-600, xl-600, len(segy.samples)), np.nan, dtype=np.
38                             float32)
39         print(f" - in {time() - f_time:.2f} seconds")
40         print("Extracting Traces")
41         f_time = time()
42         for x in tqdm(range(len(segy.trace))):
43             i = segy.attributes(segyio.TraceField.INLINE_3D)[x]-mi
44             c = segy.attributes(segyio.TraceField.CROSSLINE_3D)[x]-mx
45             if (i >= 300) and (i < il - 300) and (c >= 300) and (c < xl -
46                     300):
47                 out[i-300, c-300, :] = segy.trace[x]
48         print(f" - in {time() - f_time:.2f} seconds")
49
50         print("Normalization")
51         f_time = time()
52         out_clip = np.percentile(np.abs(out), 99)
53         out = np.clip(out, -out_clip, out_clip)
54         out /= max(abs(out.min()), out.max())
55         print(f" - in {time() - f_time:.2f} seconds")
56
57         print("Reshaping Data to Batches")
58         f_time = time()
59         i, j, k = (64, 64, 256-64) # Batch Dimension
60         s_i, s_j, s_k = (2, 2, 1.2) # Strides
61         y = cutup(out, (i, j, k), (int(i/s_i), int(j/s_j), int(k/s_k))).
62             reshape(-1, i, j, k, 1)
63         print(f" - in {time() - f_time:.2f} seconds")
64
65         print("Saving File")
66         f_time = time()
67         np.save(numpy_path, y)
68         np.save(cube_path, out)
69         print(f" - in {time() - f_time:.2f} seconds")

```

```

67     print("Testing Load")
68     f_time = time()
69     z = np.load(numpy_path)
70     np.testing.assert_array_equal(y,z)
71     del y
72     del z
73     z = np.load(cube_path)
74     np.testing.assert_array_equal(out,z)
75     del z
76     print(f" - in {time() - f_time:.2f} seconds")
77     print("--- --- ---")
78     print(f"Total: {time() - start_time:.2f} seconds")
79     print("=== === ===", end="\n\n")
80
81 print("Conversion Completed.")

```

```

1  """
2  train atlas-based alignment with MICCAI2018 version of VoxelMorph,
3  specifically adding uncertainty estimation and diffeomorphic transforms.
4  """
5
6  # python imports
7  import os
8  import glob
9  import sys
10 import random
11 from argparse import ArgumentParser
12
13 # third-party imports
14 import tensorflow as tf
15 import numpy as np
16 from keras.backend.tensorflow_backend import set_session
17 from keras.optimizers import Adam
18 from keras.callbacks import ModelCheckpoint, CSVLogger, TerminateOnNaN,
19     ReduceLROnPlateau, EarlyStopping
20
21 # project imports
22 import datagenerators
23 import networks
24 import losses
25
26 vm_dir = '/home/jdram/voxelmorph/'
27 sys.path.append(os.path.join(vm_dir, 'ext', 'neuron'))
28 import neuron.callbacks as nrn_gen
29
30 # export PYTHONPATH=$PYTHONPATH:/home/jdram/voxelmorph/ext/neuron:/home/
31     jdram/voxelmorph/ext/pynd-lib:/home/jdram/voxelmorph/ext/pytools-lib/
32
33 def train(data_dir,
34         atlas_file,
35         model_dir,

```

```

34         gpu_id,
35         lr,
36         nb_epochs,
37         prior_lambda,
38         image_sigma,
39         steps_per_epoch,
40         batch_size,
41         load_model_file,
42         bidir,
43         bool_cc,
44         initial_epoch=0):
45     """
46     model training function
47     :param data_dir: folder with npz files for each subject.
48     :param atlas_file: atlas filename. So far we support npz file with a '
49         vol' variable
50     :param model_dir: model folder to save to
51     :param gpu_id: integer specifying the gpu to use
52     :param lr: learning rate
53     :param nb_epochs: number of training iterations
54     :param prior_lambda: the prior_lambda, the scalar in front of the
55         smoothing laplacian, in MICCAI paper
56     :param image_sigma: the image sigma in MICCAI paper
57     :param steps_per_epoch: frequency with which to save models
58     :param batch_size: Optional, default of 1. can be larger, depends on
59         GPU memory and volume size
60     :param load_model_file: optional h5 model file to initialize with
61     :param bidir: logical whether to use bidirectional cost function
62     :param bool_cc: Train CC or MICCAI version
63     """
64
65     # load atlas from provided files. The atlas we used is 160x192x224.
66     #atlas_vol = np.load(atlas_file)['vol'][np.newaxis, ..., np.newaxis]
67     vm_dir = '/home/jdram/voxelmorph/'
68     base = np.load(os.path.join(vm_dir, "data", "
69         ts12_dan_a88_fin_o_trim_adpc_002661_256.npy"))
70     monitor = np.load(os.path.join(vm_dir, "data", "
71         ts12_dan_a05_fin_o_trim_adpc_002682_256.npy"))
72     #base = np.load(os.path.join(vm_dir, "data", "
73         ts12_dan_a88_fin_o_trim_adpc_002661_abs.npy"))
74     #monitor = np.load(os.path.join(vm_dir, "data", "
75         ts12_dan_a05_fin_o_trim_adpc_002682_abs.npy"))
76
77     #vol_size = (64, 64, 64)
78     vol_size = (64, 64, 256-64)
79     #vol_size = (128, 128, 256)
80
81     # prepare data files
82     # for the CVPR and MICCAI papers, we have data arranged in train/
83         validate/test folders
84     # inside each folder is a /vols/ and a /asegs/ folder with the volumes
85     # and segmentations. All of our papers use npz formatted data.

```



```

78 #train_vol_names = glob.glob(os.path.join(data_dir, '*.npy'))
79 #random.shuffle(train_vol_names) # shuffle volume list
80 #assert len(train_vol_names) > 0, "Could not find any training data"
81
82 # Diffeomorphic network architecture used in MICCAI 2018 paper
83 nf_enc = [32,64,64,64]
84 nf_dec = [64,64,64,64,32,3]
85
86 # prepare model folder
87 if not os.path.isdir(model_dir):
88     os.mkdir(model_dir)
89 tf.reset_default_graph()
90
91 if bool_cc:
92     pre_net = "cc_"
93 else:
94     if bidir:
95         pre_net = "miccai_bidir_"
96     else:
97         pre_net = "miccai_"
98
99
100 # gpu handling
101 gpu = '/device:GPU:%d' % int(gpu_id) # gpu_id
102 os.environ["CUDA_VISIBLE_DEVICES"] = gpu_id
103 config = tf.ConfigProto()
104 config.gpu_options.allow_growth = True
105 config.allow_soft_placement = True
106 set_session(tf.Session(config=config))
107
108 # prepare the model
109 with tf.device(gpu):
110     # prepare the model
111     # in the CVPR layout, the model takes in [image_1, image_2] and
112     # outputs [warped_image_1, flow]
113     # in the experiments, we use image_2 as atlas
114     if bool_cc:
115         model = networks.cvpr2018_net(vol_size, nf_enc, nf_dec)
116     else:
117         model = networks.miccai2018_net(vol_size, nf_enc, nf_dec,
118                                         bidir=bidir, vel_resize=.5)
119
120 # load initial weights
121 if load_model_file is not None and load_model_file != "":
122     print('loading', load_model_file)
123     model.load_weights(load_model_file)
124
125 # save first iteration
126 model.save(os.path.join(model_dir, f'{pre_net}{initial_epoch:02d}.
127                        h5'))
128 model.summary()

```

```

127
128     if bool_cc:
129         model_losses = [losses.NCC().loss, losses.Grad('l2').loss]
130         loss_weights = [1.0, 0.01] # recommend 1.0 for ncc, 0.01 for
            mse
131     else:
132         flow_vol_shape = model.outputs[-1].shape[1:-1]
133         loss_class = losses.Miccai2018(image_sigma, prior_lambda,
            flow_vol_shape=flow_vol_shape)
134         if bidir:
135             model_losses = [loss_class.recon_loss, loss_class.
                recon_loss, loss_class.kl_loss]
136             loss_weights = [0.5, 0.5, 1]
137         else:
138             model_losses = [loss_class.recon_loss, loss_class.kl_loss]
139             loss_weights = [1, 1]
140
141     segy_gen = datagenerators.segy_gen(base, monitor, batch_size=
        batch_size)
142
143     # prepare callbacks
144     save_file_name = os.path.join(model_dir, pre_net+'{epoch:02d}.h5')
145
146     with tf.device(gpu):
147         # fit generator
148         save_callback = ModelCheckpoint(save_file_name, period=5)
149         csv_cb = CSVLogger(f'{pre_net}log.csv')
150         nan_cb = TerminateOnNaN()
151         rlr_cb = ReduceLROnPlateau(monitor='loss', verbose=1)
152         els_cb = EarlyStopping(monitor='loss', patience=15, verbose=1,
            restore_best_weights=True)
153         cbs = [save_callback, csv_cb, nan_cb, rlr_cb, els_cb]
154         mg_model = model
155
156         # compile
157         mg_model.compile(optimizer=Adam(lr=lr), loss=model_losses,
            loss_weights=loss_weights)
158
159
160
161         mg_model.fit([base, monitor],[monitor, np.zeros_like(base)],
162             initial_epoch=initial_epoch,
163             batch_size=8,
164             epochs=nb_epochs,
165             callbacks=cbs,
166             #steps_per_epoch=steps_per_epoch,
167             verbose=1)
168
169
170 if __name__ == "__main__":
171     parser = ArgumentParser()
172

```

```

173 parser.add_argument("--data_dir", type=str, default='/home/jdram/
    voxelmorph/data',
174                        help="data folder")
175
176 parser.add_argument("--atlas_file", type=str,
177                      dest="atlas_file", default='/home/jdram/voxelmorph
    /data/atlas_norm.npz',
178                      help="atlas file")
179 parser.add_argument("--model_dir", type=str,
180                      dest="model_dir", default='/home/jdram/voxelmorph/
    models/',
181                      help="models folder")
182 parser.add_argument("--gpu", type=str, default="6",
183                      dest="gpu_id", help="gpu id number")
184 parser.add_argument("--lr", type=float,
185                      dest="lr", default=1e-4, help="learning rate")
186 parser.add_argument("--epochs", type=int,
187                      dest="nb_epochs", default=350,
188                      help="number of iterations")
189 parser.add_argument("--prior_lambda", type=float,
190                      dest="prior_lambda", default=10,
191                      help="prior_lambda regularization parameter")
192 parser.add_argument("--image_sigma", type=float,
193                      dest="image_sigma", default=0.02,
194                      help="image noise parameter")
195 parser.add_argument("--steps_per_epoch", type=int,
196                      dest="steps_per_epoch", default=100,
197                      help="frequency of model saves")
198 parser.add_argument("--batch_size", type=int,
199                      dest="batch_size", default=1,
200                      help="batch_size")
201 parser.add_argument("--load_model_file", type=str,
202                      dest="load_model_file",
203                      help="optional h5 model file to initialize with")
204 parser.add_argument("--bidir", type=int,
205                      dest="bidir", default=0,
206                      help="whether to use bidirectional cost function")
207 parser.add_argument("--initial_epoch", type=int,
208                      dest="initial_epoch", default=0,
209                      help="first epoch")
210 parser.add_argument("--cc", type=bool,
211                      dest="bool_cc", default=False,
212                      help="Train MICCAI diffeomorphism version or CC.")
213
214 args = parser.parse_args()
215 train(**vars(args))

```

```

1 """
2 Example script to register two volumes with VoxelMorph models
3 Please make sure to use trained models appropriately.

```

```

4 Let's say we have a model trained to register subject (moving) to atlas (
  fixed)
5 One could run:
6 python register.py --gpu 0 /path/to/test_vol.nii.gz /path/to/atlas_norm.
  nii.gz --out_img /path/to/out.nii.gz --model_file ../models/
  cvpr2018_vm2_cc.h5
7 """
8
9 ###
10 # py imports
11 import os
12 import sys
13 from argparse import ArgumentParser
14
15 # third party
16 import tensorflow as tf
17 import numpy as np
18 import keras
19 from keras.backend.tensorflow_backend import set_session
20 from scipy.interpolate import interpn
21
22 import matplotlib.pyplot as plt
23
24 ###
25 # project
26 sys.path.append('/home/jdram/voxielmorph/src')
27 import networks, losses
28 sys.path.append('/home/jdram/voxielmorph/ext/neuron')
29 import neuron.layers as nrn_layers
30
31 ###
32
33 def register(gpu_id, mov, fix, model_file, out_img, out_warp):
34     """
35     register moving and fixed.
36     """
37     #assert model_file, "A model file is necessary"
38     #assert out_img or out_warp, "output image or warp file needs to be
39         specified"
40
41     # GPU handling
42     if gpu_id is not None:
43         gpu = '/gpu:' + str(gpu_id)
44         os.environ["CUDA_VISIBLE_DEVICES"] = str(gpu_id)
45         config = tf.ConfigProto()
46         config.gpu_options.allow_growth = True
47         config.allow_soft_placement = True
48         set_session(tf.Session(config=config))
49     else:
50         gpu = '/cpu:0'
51
52     # load data

```

```

52 #mov_nii = nib.load(moving)
53 #mov = mov_nii.get_data()[np.newaxis, ..., np.newaxis]
54 #fix_nii = nib.load(fixed)
55 #fix = fix_nii.get_data()[np.newaxis, ..., np.newaxis]
56
57 with tf.device(gpu):
58     # load model
59     loss_class = losses.Miccai2018(0.02, 10, flow_vol_shape=[256-64])
60     custom_objects = {'SpatialTransformer': nrn_layers.
        SpatialTransformer,
61                        'VecInt': nrn_layers.VecInt,
62                        'Sample': networks.Sample,
63                        'Rescale': networks.RescaleDouble,
64                        'Resize': networks.ResizeDouble,
65                        'Negate': networks.Negate,
66                        'recon_loss': loss_class.recon_loss, # values shouldn't
                        matter
67                        'kl_loss': loss_class.kl_loss        # values shouldn't
                        matter
68                    }
69
70
71     net = keras.models.load_model(model_file, custom_objects=
        custom_objects)
72
73     # register
74     [moved, warp] = net.predict([mov, fix])
75
76     return moved, warp
77
78 def plt_commons(title, ix, cbar_label=None, pre='x', name='x', w=0, suff='
    '):
79     plt.title(title.title())
80     plt.ylabel('Time [s]')
81     plt.xlabel(f'{ix} [m]')
82     if cbar_label:
83         cbar = plt.colorbar(pad=0.1, orientation="horizontal", aspect=20)
84         cbar.set_label(cbar_label)
85     plt.savefig(f'{pre}_{ix}_{title}_{w}{suff}.png'.replace(' ', '_').lower
        (), bbox_inches='tight')
86
87 def warp_results(moving, fixed, moved, warped, pre='x'):
88     """
89     Warp Results and plot on CPU
90     moving: monitor
91     fixed: base
92     moved: matched monitor
93     warped: warp vector
94     pre: extra suffix for multiple experiments
95     """
96     seis = np.max([np.abs(moving[0, 32, :, :, 0]).max(), \
97                    np.abs(fixed[0, 32, :, :, 0]).max(), \

```

```

98     np.abs(moved[0, 32, :, :, 0]).max(), \
99     np.abs(moving[0, :, 32, :, 0]).max(), \
100    np.abs(fixed[0, :, 32, :, 0]).max(), \
101    np.abs(moved[0, :, 32, :, 0]).max(), \
102    ])
103
104    orig_warp = warped.copy()
105    orig_warp[:, :, :, :, 3:] = np.exp(orig_warp[:, :, :, :, 3:]/2)
106
107    warped[:, :, :, :, 3:] = np.exp(warped[:, :, :, :, 3:]/2)
108    warped[:, :, :, :, 0:2] *= 12.5
109    warped[:, :, :, :, 3:5] *= 12.5
110    warped[:, :, :, :, 2] *= 0.004 * 1e3
111    warped[:, :, :, :, 5] *= 0.004 * 1e3
112    warp_u = np.max([np.abs(warped[0, 32, :, :, :2]), \
113                    np.abs(warped[0, :, 32, :, :2])])
114
115    warp_ut = np.max([np.abs(warped[0, 32, :, :, 2]), \
116                     np.abs(warped[0, :, 32, :, 2])])
117
118    diff = np.max([np.abs(moving[0, 32, :, :, 0]-fixed[0, 32, :, :, 0]).
119                  max(), \
120                  np.abs(moved[0, 32, :, :, 0]-fixed[0, 32, :, :, 0]).max(), \
121                  ])
122
123    for name, dat in {'monitor': moving, 'base': fixed, 'matched': moved,
124                    'warp': warped}.items():
125        extent = (0, 64*12.5, (256-64)*.004, 0)
126        plt_size= (3,9)
127
128        plt_args = {'extent':extent, 'aspect':'auto', 'interpolation':'
129                    bicubic'}
130
131        for w in range(dat.shape[-1]):
132
133            if w < 3:
134                cmap = 'RdBu'
135
136                if dat.shape[-1] == 1:
137                    va = diff
138                    cb = 'Amplitude'
139
140                    # Difference Images
141                    if not 'base' in name:
142                        plt.figure(figsize=plt_size)
143                        plt.imshow(dat[0, 32, :, :, w].T-fixed[0, 32, :,
144                                :, w].T, cmap=cmap, vmin=-va, vmax=va, **
145                                plt_args)
146                        plt_commons(name+' difference', 'inline', cb, pre=
147                                pre, name=name, w=w, suff='_diff')
148
149                        plt.figure(figsize=plt_size)

```

```

144         plt.imshow(dat[0, :, 32, :, w].T-fixed[0, :, 32,
145                     :, w].T, cmap=cmap, vmin=-va, vmax=va, **
146                     plt_args)
147
148         plt_commons(name+' difference', 'crossline', cb,
149                     pre=pre, name=name, w=w, suff='_diff')
150     va = seis
151
152     else:
153         va = warp_u
154         cb = 'Spatial Shift [m]'
155         if w == 2:
156             va = warp_ut
157             cb = 'Time Shift [ms]'
158
159         # Intentionally left unindented
160         plt.figure(figsize=plt_size)
161         plt.imshow(dat[0, 32, :, :, w].T, cmap=cmap, vmin=-va,
162                     vmax=va, **plt_args)
163         plt_commons(name, 'inline', cb, pre=pre, name=name, w=w)
164
165         plt.figure(figsize=plt_size)
166         plt.imshow(dat[0, :, 32, :, w].T, cmap=cmap, vmin=-va,
167                     vmax=va, **plt_args)
168         plt_commons(name, 'crossline', cb, pre=pre, name=name, w=w
169                     )
170
171     else:
172         cmap = 'viridis'
173         cb = 'Spatial Uncertainty $1\sigma$ [m]'
174         va = warp_u
175         if w == 5:
176             cb = 'Temporal Uncertainty $1\sigma$ [ms]'
177             va = warp_ut
178
179         plt.figure(figsize=plt_size)
180         plt.imshow(dat[0, 32, :, :, w].T, cmap=cmap, **plt_args)
181         plt_commons(name, 'inline', cb, pre=pre, name=name, w=w)
182
183         plt.figure(figsize=plt_size)
184         plt.imshow(dat[0, :, 32, :, w].T, cmap=cmap, **plt_args)
185         plt_commons(name, 'crossline', cb, pre=pre, name=name, w=w
186                     )
187
188     if w == 0:
189         np.save(f'{name}_{pre}.npy', dat)
190     plt.close('all')
191
192     ###
193     vm_dir = '/home/jdram/voxelmorph/'
194     model_path = "/home/jdram/voxelmorph/models/backup/miccai_300_full_deep.h5
195                 " #64x64x192 full unet
196     #model_path = "/home/jdram/voxelmorph/models/backup/miccai_e290_64-64-192

```

```

    _upsample.h5" #64x64x192 upsampled
188
189 monitor_hfd = np.load(os.path.join("/", "scratch", "jdram", "voxelmorph",
    "halfdan", "ts12_hfd_h05_fin_o_trim_adpc_002582_cube.npy"))
190 base_hfd     = np.load(os.path.join("/", "scratch", "jdram", "voxelmorph",
    "halfdan", "ts12_hfd_h93_fin_o_trim_adpc_002568_cube.npy"))
191
192 #k = 350
193 #p = 850
194 #q = 475
195 k = 350
196 p = 200
197 q = 475
198 moving = monitor_hfd[np.newaxis, k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
199 fixed   = base_hfd[np.newaxis,k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
200
201 #%%
202 moved, warped = register(None, moving, fixed, model_path, None, None)
203
204 #%%
205 warp_results(moving, fixed, moved, warped, 'hfd')
206
207 #%%
208
209 monitor = np.load(os.path.join(vm_dir, "data", "
    ts12_dan_a05_fin_o_trim_adpc_002682_cube.npy"))
210 base     = np.load(os.path.join(vm_dir, "data", "
    ts12_dan_a88_fin_o_trim_adpc_002661_cube.npy"))
211
212 k=225
213 q=350
214 p=300
215
216 moving = monitor[np.newaxis, k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
217 fixed   = base[np.newaxis,k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
218
219 #%%
220 moved, warped = register(None, moving, fixed, model_path, None, None)
221
222 #%%
223
224 #np.save("moving_a.npy",moving)
225 #np.save("fixed_a.npy",fixed)
226 #np.save("moved_a.npy",moved)
227 #np.save("warped_a.npy",warped)
228
229 warp_results(moving, fixed, moved, warped, 'a')
230
231
232
233 base2     = np.load(os.path.join(vm_dir, "data", "
    ts12_dan_d05_fin_o_trim_adpc_002696_cube.npy"))

```



```

234 monitor2 = np.load(os.path.join(vm_dir, "data", "
      ts12_dan_d12_fin_o_trim_adpc_002710_cube.npy"))
235
236 moving = monitor2[np.newaxis, k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
237 fixed  = base2[np.newaxis,k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
238
239 ###
240 moved, warped = register(None, moving, fixed, model_path, None, None)
241
242 ###
243 warp_results(moving,fixed,moved,warped,'d')
244
245
246 moving = monitor2[np.newaxis, k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
247 fixed  = monitor[np.newaxis,k-32:k+32,p:p+64,q:q+256-64,np.newaxis]
248
249 ###
250 moved, warped = register(None, moving, fixed, model_path, None, None)
251
252 ###
253 warp_results(moving,fixed,moved,warped,'ad')

```

E.5 Software Manual: Keras Complex

Software manual from the `keras complex` package for complex-valued neural networks in Python 3. Original code by Trabelsi et al. (2017) in Theano. Code ported to Tensorflow, consolidated, packaged, set up with automatic testing and documentation by Dramsch et al. (2019c).

Original Code:

C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal (2017). “Deep complex networks”. In: *arXiv preprint arXiv:1705.09792*

Port to Keras with Tensorflow:

J. S. Dramsch and Contributors (2019c). *Complex-Valued Neural Networks in Keras with Tensorflow*. Open-Source Software. DOI: 10.6084/m9.figshare.9783773. URL: <https://github.com/JesperDramsch/keras-complex>

Keras Complex

Oct 16, 2019

Table of Contents

1	Contents	3
1.1	Introduction	3
1.2	Installation	3
1.3	complexn	3
1.4	How to Contribute	19
1.5	Implementation and Math	19
1.6	Citation	20
2	Indices and tables	23
	Bibliography	25
	Python Module Index	27
	Index	29

Complex-valued convolutions could provide some interesting results in signal processing-based deep learning. A simple(-ish) idea is including explicit phase information of time series in neural networks. This code enables complex-valued convolution in convolutional neural networks in keras with the TensorFlow backend. This makes the network modular and interoperable with standard keras layers and operations.

CHAPTER 1

Contents

1.1 Introduction

Complex-valued convolutions could provide some interesting results in signal processing-based deep learning. A simple(-ish) idea is including explicit phase information of time series in neural networks. This code enables complex-valued convolution in convolutional neural networks in keras with the TensorFlow backend. This makes the network modular and interoperable with standard keras layers and operations.

1.2 Installation

Installation is as easy as
<code>pip install keras-complex</code>
but you'll need to install tensorflow in addition using
<code>pip install tensorflow-gpu</code>
for the GPU version or for the non-GPU version:
<code>pip install tensorflow</code>

1.3 complexnn

1.3.1 complexnn package

Submodules

Keras Complex

complexnn.bn module

`complexnn.bn.ComplexBN` (*input_centred*, *Vrr*, *Vii*, *Vri*, *beta*, *gamma_rr*, *gamma_rri*, *gamma_rii*, *scale=True*, *center=True*, *layernorm=False*, *axis=-1*)

Complex Batch Normalization

Arguments: *input_centred* – input data *Vrr* – Real component of covariance matrix *V Vri* – Imaginary component of covariance matrix *V Vri* – Non-diagonal component of covariance matrix *V beta* – Learnable shift parameter *beta gamma_rr* – Scaling parameter *gamma* - *rr* component of 2x2 matrix *gamma_rri* – Scaling parameter *gamma* - *ri* component of 2x2 matrix *gamma_rii* – Scaling parameter *gamma* - *ii* component of 2x2 matrix

Keyword Arguments: *scale* (bool) – Standardization of input (default: (True)) *center* (bool) – Mean-shift correction (default: (True)) *layernorm* (bool) – Normalization (default: (False)) *axis* (int) – Axis for Standardization (default: {-1})

Raises: ValueError: Dimonsional mismatch

Returns: Batch-Normalized Input

```
class complexnn.bn.ComplexBatchNormalization(axis=-1, momentum=0.9, epsilon=0.0001, center=True, scale=True, beta_initializer='zeros', gamma_ddag_initializer='sqrt_init', gamma_ddag_offset_initializer='zeros', gamma_offset_initializer='zeros', moving_variance_initializer='sqrt_init', moving_covariance_initializer='zeros', beta_regularizer=None, gamma_ddag_regularizer=None, gamma_offset_regularizer=None, beta_constraint=None, gamma_ddag_constraint=None, gamma_offset_constraint=None, **kwargs)
```

Bases: `keras.engine.base_layer.Layer`

Complex version of the real domain Batch normalization layer (Ioffe and Szegedy, 2014). Normalize the activations of the previous complex layer at each batch, i.e. applies a transformation that maintains the mean of a complex unit close to the null vector, the 2 by 2 covariance matrix of a complex unit close to identity and the 2 by 2 relation matrix, also called pseudo-covariance, close to the null matrix. # Arguments

axis: Integer, the axis that should be normalized (typically the features axis). For instance, after a *Conv2D* layer with *data_format="channels_first"*, set *axis=2* in *ComplexBatchNormalization*.

momentum: Momentum for the moving statistics related to the real and imaginary parts.

epsilon: Small float added to each of the variances related to the real and imaginary parts in order to avoid dividing by zero.

center: If True, add offset of *beta* to complex normalized tensor. If False, *beta* is ignored. (*beta* is formed by real_beta and imag_beta)

scale: If True, multiply by the *gamma* matrix. If False, *gamma* is not used.

beta_initializer: Initializer for the real_beta and the imag_beta weight. *gamma_ddag_initializer:* Initializer for the diagonal elements of the *gamma* matrix.

which are the variances of the real part and the imaginary part.

`gamma_off_initializer`: Initializer for the off-diagonal elements of the gamma matrix. `moving_mean_initializer`: Initializer for the moving means. `moving_variance_initializer`: Initializer for the moving variances. `moving_covariance_initializer`: Initializer for the moving covariance of the real and imaginary parts.

`beta_regularizer`: Optional regularizer for the beta weights. `gamma_regularizer`: Optional regularizer for the gamma weights. `beta_constraint`: Optional constraint for the beta weights. `gamma_constraint`: Optional constraint for the gamma weights.

Input shape Arbitrary. Use the keyword argument *input_shape* (tuple of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape Same shape as input.

References

- [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](<https://arxiv.org/abs/1502.03167>)

build(*input_shape*)

Creates the layer weights.

Must be implemented on all layers that have weights.

Arguments

input_shape: **Keras tensor** (*future input to layer*) or list/tuple of Keras tensors to reference for weight shape computations.

call(*inputs*, *training=None*)

This is where the layer's logic lives.

Arguments *inputs*: Input tensor, or list/tuple of input tensors. ****kwargs**: Additional keyword arguments.

Returns A tensor or list/tuple of tensors.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns Python dictionary.

`complexnn.bn.complex_standardization` (*input_centred*, *Vrr*, *Vii*, *Vri*, *layernorm=False*, *axis=-1*)

Complex Standardization of input

Arguments: *input_centred* – Input Tensor *Vrr* – Real component of covariance matrix *V Vii* – Imaginary component of covariance matrix *V Vri* – Non-diagonal component of covariance matrix *V*

Keyword Arguments: *layernorm* (bool) – Normalization (default: {False}) *axis* (int) – Axis for Standardization (default: {-1})

Raises: `ValueError`: Mismatched dimensions

Returns: Complex standardized input

`complexnn.bn.sanitizeinitGet` (*init*)

`complexnn.bn.sanitizeinitSet` (*init*)

`complexnn.bn.sqr_init` (*shape*, *dtype=None*)

complexnn.conv module

`conv.py`

```
class complexnn.conv.ComplexConv (rank, filters, kernel_size, strides=1, padding='valid',
data_format=None, kernel_size, dilation_rate=1, activation=None,
use_bias=True, normalize_weight=False, kernel_initializer='complex', bias_initializer='zeros',
gamma_diag_initializer=<function> sqr_init>, gamma_off_initializer='zeros', kernel_regularizer=None,
bias_regularizer=None, gamma_diag_regularizer=None, gamma_off_regularizer=None, kernel_constraint=None,
bias_constraint=None, gamma_diag_constraint=None, gamma_off_constraint=None, init_criterion='he',
seed=None, spectral_parameterization=False, transpose=False, epsilon=1e-07, **kwargs)
```

Bases: `keras.engine.base_layer.Layer`

Abstract nD complex convolution layer.

This layer creates a complex convolution kernel that is convolved with the layer input to produce a tensor of outputs. If *use_bias* is True, a bias vector is created and added to the outputs. Finally, if *activation* is not `None`, it is applied to the outputs as well.

Arguments:

rank: Integer, the rank of the convolution, e.g., "2" for 2D convolution.

filters: Integer, the dimensionality of the output space, i.e., the number of complex feature maps. It is also the effective number of feature maps for each of the real and imaginary parts. (I.e., the number of complex filters in the convolution) The total effective number of filters is 2 x filters.

kernel_size: An integer or tuple/list of n integers, specifying the dimensions of the convolution window.

strides: An integer or tuple/list of n integers, specifying the strides of the convolution. Specifying any stride value != 1 is incompatible with specifying any *dilation_rate* value != 1.

padding: One of "valid" or "same" (case-insensitive). *data_format*: A string, one of *channels_last* (default) or

channels_first. The ordering of the dimensions in the inputs. *channels_last* corresponds to inputs with shape *(batch..., channels)* while *channels_first* corresponds to inputs with shape *(batch, channels, ...)*. It defaults to the *image_data_format* value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

dilation_rate: An integer or tuple/list of n integers, specifying the dilation rate to use for dilated convolution. Currently, specifying any *dilation_rate* value != 1 is incompatible with specifying any *strides* value != 1.

activation: Activation function to use (see `keras.activations`). If you don't specify anything, no activation is applied (i.e., "linear" activation: $d(x) = x$).

use_bias: Boolean, whether the layer uses a bias vector. *normalize_weight*: Boolean, whether the layer normalizes its complex

weights before convolving the complex input. The complex normalization performed is similar to the one for the batchnorm. Each of the complex kernels is centred and multiplied by the inverse square root of the covariance matrix. Then a complex multiplication is performed as the normalized weights are multiplied by the complex scaling factor gamma.

kernel_initializer: *Initializer for the complex kernel weights* matrix. By default it is 'complex'. The 'complex_independent' and the usual initializers could also be used. (See keras.initializers and init.py).

bias_initializer: *Initializer for the bias vector* (see keras.initializers).

kernel_regularizer: *Regularizer function applied to the kernel weights* matrix (see keras.regularizers).

bias_regularizer: *Regularizer function applied to the bias vector* (see keras.regularizers).

activity_regularizer: *Regularizer function applied to the output of the layer* (its "activation"). (See keras.regularizers).

kernel_constraint: *Constraint function applied to the kernel matrix* (see keras.constraints).

bias_constraint: *Constraint function applied to the bias vector* (see keras.constraints).

spectral_parametrization: *Boolean, whether or not to use a spectral parametrization of the parameters.*

transposed: Boolean, whether or not to use transposed convolution

build (*input_shape*)

call (*inputs*, ***kwargs*)

This is where the layer's logic lives.

Arguments: *inputs:* Input tensor, or list/tuple of input tensors. ***kwargs:* Additional keyword arguments.

Returns: A tensor or list/tuple of tensors.

compute_output_shape (*input_shape*)

Computes the output shape of the layer.

Assumes that the layer will be built to match that input shape provided.

Arguments

input_shape: *Shape tuple (tuple of integers)* or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns: An output shape tuple.

get_config ()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns: Python dictionary.

class `complexnn.Conv, ComplexConv1D` (*filters*, *kernel_size*, *strides*=1, *padding*='valid', *dilation_rate*=1, *activation*=None, *use_bias*=True, *kernel_initializer*='complex', *bias_initializer*='zeros', *kernel_regularizer*=None, *bias_regularizer*=None, *activity_regularizer*=None, *kernel_constraint*=None, *bias_constraint*=None, *seed*=None, *init_criterion*='he', *spectral_parametrization*=False, *transposed*=False, ***kwargs*)

Bases: `complexnn.Conv, ComplexConv`

1D complex convolution layer. This layer creates a complex convolution kernel that is convolved with a complex input layer over a single complex spatial (or temporal) dimension to produce a complex output tensor. If *use_bias* is True, a bias vector is created and added to the complex output. Finally, if *activation* is not None, it is applied each of the real and imaginary parts of the output. When using this layer as the first layer in a model, provide an *input_shape* argument (tuple of integers or None, e.g. (10, 128) for sequences of 10 vectors of 128-dimensional vectors, or (None, 128) for variable-length sequences of 128-dimensional vectors. # Arguments

filters: *Integer, the dimensionality of the output space, i.e.*, the number of complex feature maps.

It is also the effective number of feature maps for each of the real and imaginary parts. (i.e. the number of complex filters in the convolution) The total effective number of filters is 2 x filters.

kernel_size: *An integer or tuple/list of n integers, specifying the dimensions of the convolution window.*

strides: *An integer or tuple/list of a single integer, specifying the stride length of the convolution.* Specifying any stride value != 1 is incompatible with specifying any *dilation_rate* value != 1.

padding: *One of "valid", "causal" or "same" (case-insensitive). "causal" results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order.* See [WaveNet: A Generative Model for Raw Audio, section 2.1] (<https://arxiv.org/abs/1609.03499>).

dilation_rate: *an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution.* Currently, specifying any *dilation_rate* value != 1 is incompatible with specifying any *strides* value != 1.

activation: *Activation function to use* (see keras.activations). If you don't specify anything, no activation is applied (ie. "linear" activation: $af(x) = x$).

use_bias: Boolean, whether the layer uses a bias vector. *normalize_weight:* Boolean, whether the layer normalizes its complex

weights before convolving the complex input. The complex normalization performed is similar to the one for the batchnorm. Each of the complex kernels are centred and multiplied by the inverse square root of covariance matrix. Then, a complex multiplication is performed as the normalized weights are multiplied by the complex scaling factor gamma.

kernel_initializer: *Initializer for the complex kernel weights matrix.*

By default it is 'complex'. The 'complex_independent' and the usual initializers could also be used. (see keras.initializers and init.py).

bias_initializer: *Initializer for the bias vector* (see keras.initializers).

kernel_regularizer: *Regularizer function applied to the kernel weights* matrix (see keras.regularizers).

bias_regularizer: *Regularizer function applied to the bias vector* (see keras.regularizers).

activity_regularizer: *Regularizer function applied to the output of the layer* (its "activation"). (see keras.regularizers).

kernel_constraint: Constraint function applied to the kernel matrix (see keras.constraints).

bias_constraint: Constraint function applied to the bias vector (see keras.constraints).

spectral_parametrization: Whether or not to use a spectral parametrization of the parameters.

transposed: Boolean, whether or not to use transposed convolution

Input shape 3D tensor with shape: *(batch_size, steps, input_dim)*

Output shape 3D tensor with shape: *(batch_size, new_steps, 2 x filters)* steps value might have changed due to padding or strides.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinitialized later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns Python dictionary.

```
class complexnn.conv.ComplexConv2D(filters, kernel_size, strides=(1, 1), padding='valid',
                                     data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='complex',
                                     bias_initializer='zeros', kernel_regularizer=None,
                                     bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, seed=None,
                                     init_criterion='he', spectral_parametrization=False,
                                     transposed=False, **kwargs)
```

Bases: *complexnn.conv.ComplexConv*

2D Complex convolution layer (e.g. spatial convolution over images). This layer creates a complex convolution kernel that is convolved with a complex input layer to produce a complex output tensor. If *use_bias* is True, a complex bias vector is created and added to the outputs. Finally, if *activation* is not *None*, it is applied to both the real and imaginary parts of the output. When using this layer as the first layer in a model, provide the keyword argument *input_shape* (tuple of integers, does not include the sample axis), e.g. *input_shape=(128, 128, 3)* for 128x128 RGB pictures in *data_format='channels_last'*. # Arguments

filters: Integer, the dimensionality of the complex output space (i.e. the number complex feature maps in the convolution). The total effective number of filters or feature maps is 2 x filters.

kernel_size: An integer or tuple/list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any *dilation_rate* value != 1.

padding: one of "valid" or "same" (case-insensitive). data_format: A string.

one of *channels_last* (default) or *channels_first*. The ordering of the dimensions in the inputs. *channels_last* corresponds to inputs with shape *(batch, height, width, channels)* while *channels_first* corresponds to inputs with shape *(batch, channels, height, width)*. It defaults to the *image_data_format* value found in your Keras config file at *~/.keras/keras.json*. If you never set it, then it will be "channels_last".

dilation_rate: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any *dilation_rate* value != 1 is incompatible with specifying any stride value != 1.

activation: Activation function to use (see keras.activations). If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).

use_bias: Boolean, whether the layer uses a bias vector. *normalize_weight*: Boolean, whether the layer normalizes its complex

weights before convolving the complex input. The complex normalization performed is similar to the one for the batchnorm. Each of the complex kernels are centred and multiplied by the inverse square root of covariance matrix. Then, a complex multiplication is performed as the normalized weights are multiplied by the complex scaling factor gamma.

kernel_initializer: Initializer for the complex *kernel* weights matrix.

By default it is 'complex'. The 'complex_independent' and the usual initializers could also be used. (see keras.initializers and init.py).

bias_initializer: Initializer for the bias vector (see keras.initializers).

kernel_regularizer: Regularizer function applied to the *kernel* weights matrix (see keras.regularizers).

bias_regularizer: Regularizer function applied to the bias vector (see keras.regularizers).

activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see keras.regularizers).

kernel_constraint: Constraint function applied to the kernel matrix (see keras.constraints).

bias_constraint: Constraint function applied to the bias vector (see keras.constraints).

spectral_parametrization: Whether or not to use a spectral parametrization of the parameters.

transposed: Boolean, whether or not to use transposed convolution

Input shape 4D tensor with shape: *(samples, channels, rows, cols)* if *data_format='channels_first'* or 4D tensor with shape: *(samples, rows, cols, channels)* if *data_format='channels_last'*.

Output shape 4D tensor with shape: *(samples, 2 x filters, new_rows, new_cols)* if *data_format='channels_first'* or 4D tensor with shape: *(samples, new_rows, new_cols, 2 x filters)* if *data_format='channels_last'*. *rows* and *cols* values might have changed due to padding.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinitialized later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns Python dictionary.


```
class complexnn.Conv.ComplexConv3D(filters, kernel_size, strides=(1, 1, 1), padding='valid',
                                   data_format=None, dilation_rate=(1, 1, 1), activation=None, use_bias=True, kernel_initializer='complex',
                                   bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, seed=None, init_criterion='he', spectral_parametrization=False,
                                   transposed=False, **kwargs)

Bases: complexnn.Conv.ComplexConv

3D convolution layer (e.g. spatial convolution over volumes). This layer creates a complex convolution kernel that is convolved with a complex layer input to produce a complex output tensor. If use_bias is True, a complex bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to each of the real and imaginary parts of the output. When using this layer as the first layer in a model, provide the keyword argument input_shape (tuple of integers, does not include the sample axis), e.g. input_shape=(2, 128, 128, 128, 3) for 128x128x128 volumes with 3 channels, in data_format="channels_last". # Arguments

filters: Integer, the dimensionality of the complex output space (i.e. the number complex feature maps in the convolution). The total effective number of filters or feature maps is 2 x filters.

kernel_size: An integer or tuple/list of 3 integers, specifying the width and height of the 3D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 3 integers, specifying the strides of the convolution along each spatial dimension. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding: one of "valid" or "same" (case-insensitive). data_format: A string.

one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2, spatial_dim3). It defaults to the image_data_format value found in your Keras config file at ~/keras/keras.json. If you never set it, then it will be "channels_last".

dilation_rate: an integer or tuple/list of 3 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.

activation: Activation function to use (see keras.activations). If you don't specify anything, no activation is applied (ie. "linear" activation: af(x) = x).

use_bias: Boolean, whether the layer uses a bias vector. normalize_weight: Boolean, whether the layer normalizes its complex weights before convolving the complex input. The complex normalization performed is similar to the one for the batchnorm. Each of the complex kernels are centred and multiplied by the inverse square root of covariance matrix. Then, a complex multiplication is performed as the normalized weights are multiplied by the complex scaling factor gamma.

kernel_initializer: Initializer for the complex kernel weights matrix. By default it is 'complex'. The 'complex_independent' and the usual initializers could also be used. (see keras.initializers and init.py).

bias_initializer: Initializer for the bias vector (see keras.initializers).
```

```
kernel_regularizer: Regularizer function applied to the kernel weights matrix (see keras.regularizers).

bias_regularizer: Regularizer function applied to the bias vector (see keras.regularizers).

activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see keras.regularizers).

kernel_constraint: Constraint function applied to the kernel matrix (see keras.constraints).

bias_constraint: Constraint function applied to the bias vector (see keras.constraints).

spectral_parametrization: Whether or not to use a spectral parametrization of the parameters.

transposed: Boolean, whether or not to use transposed convolution

# Input shape 5D tensor with shape: (samples, channels, conv_dim1, conv_dim2, conv_dim3) if data_format=channels_first or 3D tensor with shape: (samples, conv_dim1, conv_dim2, conv_dim3, channels) if data_format=channels_last.

# Output shape 5D tensor with shape: (samples, 2 x filters, new_conv_dim1, new_conv_dim2, new_conv_dim3) if data_format=channels_first or 5D tensor with shape: (samples, new_conv_dim1, new_conv_dim2, new_conv_dim3, 2 x filters) if data_format=channels_last. new_conv_dim1, new_conv_dim2 and new_conv_dim3 values might have changed due to padding.

get_config()
Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstanciated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by Network (one layer of abstraction above).

# Returns Python dictionary.

complexnn.Conv.ComplexConvolution1D
alias of complexnn.Conv.ComplexConv1D

complexnn.Conv.ComplexConvolution2D
alias of complexnn.Conv.ComplexConv2D

complexnn.Conv.ComplexConvolution3D
alias of complexnn.Conv.ComplexConv3D

class complexnn.Conv.WeightNormConv(gamma_initializer='ones', gamma_regularizer=None, gamma_constraint=None, epsilon=1e-07, **kwargs)

Bases: keras.layers.convolutional._Conv

build(input_shape)
Creates the layer weights.

Must be implemented on all layers that have weights.

# Arguments

input_shape: Keras tensor (future input to layer) or list/tuple of Keras tensors to reference for weight shape computations.

call(inputs)
This is where the layer's logic lives.

# Arguments inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword arguments.
```

Returns A tensor or list/tuple of tensors.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns Python dictionary.

```
complexnn.conv.conv2d_transpose(inputs, filter, kernel_size=None, filters=None,
                                  strides=(1, 1), padding='SAME', output_padding=None,
                                  data_format='channels_last')
```

Compatibility layer for K.conv2d_transpose

Take a filter defined for forward convolution and adjust it for a transposed convolution.

```
complexnn.conv.conv_transpose_output_length(input_length, filter_size, padding, stride, dilation=1, output_padding=None)
```

Rearrange arguments for compatibility with conv_output_length.

```
complexnn.conv.ifftc(f)
```

Stub

```
complexnn.conv.ifftc2(f)
```

Stub

```
complexnn.conv.sanitizedinitGet(init)
```

```
complexnn.conv.sanitizedinitSet(init)
```

complexnn.dense module

```
class complexnn.dense.ComplexDense(units, activation=None, use_bias=True,
                                     init_criterion='he', kernel_initializer='complex',
                                     bias_initializer='zeros', kernel_regularizer=None,
                                     bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None,
                                     seed=None, **kwargs)
```

Bases: keras.engine.base_layer.Layer

Regular complex densely-connected NN layer. Dense implements the operation: *real_preact = dot(real_input, real_kernel) - dot(imag_input, imag_kernel)* *imag_preact = dot(real_input, imag_kernel) + dot(imag_input, real_kernel)* *output = activation(K.concatenate([real_preact, imag_preact]) + bias)* where *activation* is the element-wise activation function passed as the *activation* argument, *kernel* is a weights matrix created by the layer, and *bias* is a bias vector created by the layer (only applicable if *use_bias* is *True*). Note: if the input to the layer has a rank greater than 2, then AN ERROR MESSAGE IS PRINTED. # Arguments

units: Positive integer, dimensionality of each of the **real part** and the imaginary part. It is actually the number of complex units.

activation: Activation function to use (see keras.activations). If you don't specify anything, no activation is applied (ie. "linear" activation: $af(x) = x$).

use_bias: Boolean, whether the layer uses a bias vector. *kernel_initializer*: Initializer for the complex *kernel* weights matrix.

By default it is 'complex', and the usual initializers could also be used. (see keras.initializers and init.py)

bias_initializer: Initializer for the bias vector (see keras.initializers).

kernel_regularizer: Regularizer function applied to the *kernel* weights matrix (see keras.regularizers).

bias_regularizer: Regularizer function applied to the bias vector (see keras.regularizers).

activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see keras.regularizers).

kernel_constraint: Constraint function applied to the kernel matrix (see keras.constraints).

bias_constraint: Constraint function applied to the bias vector (see keras.constraints).

Input shape a 2D input with shape *(batch_size, input_dim)*.

Output shape For a 2D input with shape *(batch_size, input_dim)*, the output would have shape *(batch_size, units)*.

build(input_shape)

Creates the layer weights.

Must be implemented on all layers that have weights.

Arguments

input_shape: Keras tensor (*future input to layer*) or list/tuple of Keras tensors to reference for weight shape computations.

call(inputs)

This is where the layer's logic lives.

Arguments inputs: Input tensor, or list/tuple of input tensors. ****kwargs**: Additional keyword arguments.

Returns A tensor or list/tuple of tensors.

compute_output_shape(input_shape)

Computes the output shape of the layer.

Assumes that the layer will be built to match that input shape provided.

Arguments

input_shape: Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns An output shape tuple.

get_config()

Returns the config of the layer.

A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

The config of a layer does not include connectivity information, nor the layer class name. These are handled by *Network* (one layer of abstraction above).

Returns Python dictionary.

complexnn.fft module

```
class complexnn.fft.FFT (**kwargs)
    Bases: keras.engine.base_layer.Layer
    call (x, mask=None)
        This is where the layer's logic lives.

    # Arguments
        inputs: Input tensor, or list/tuple of input tensors.
        **kwargs: Additional keyword arguments.

    # Returns
        A tensor or list/tuple of tensors.

class complexnn.fft.FFT2 (**kwargs)
    Bases: keras.engine.base_layer.Layer
    call (x, mask=None)
        This is where the layer's logic lives.

    # Arguments
        inputs: Input tensor, or list/tuple of input tensors.
        **kwargs: Additional keyword arguments.

    # Returns
        A tensor or list/tuple of tensors.

class complexnn.fft.IFFT (**kwargs)
    Bases: keras.engine.base_layer.Layer
    call (x, mask=None)
        This is where the layer's logic lives.

    # Arguments
        inputs: Input tensor, or list/tuple of input tensors.
        **kwargs: Additional keyword arguments.

    # Returns
        A tensor or list/tuple of tensors.

class complexnn.fft.IFFT2 (**kwargs)
    Bases: keras.engine.base_layer.Layer
    call (x, mask=None)
        This is where the layer's logic lives.

    # Arguments
        inputs: Input tensor, or list/tuple of input tensors.
        **kwargs: Additional keyword arguments.

    # Returns
        A tensor or list/tuple of tensors.

complexnn.fft.fft (z)
complexnn.fft.fft2 (x)
complexnn.fft.ifft (z)
complexnn.fft.ifft2 (x)

complexnn.init module

class complexnn.init.ComplexIndependentFilters (kernel_size, input_dim, weight_dim,
nb_filters=None, criterion='glorot', seed=None)
    Bases: keras.initializers.Initializer
    get_config ()
```

```
class complexnn.init.ComplexInit (kernel_size, input_dim, weight_dim, nb_filters=None, criterion='glorot', seed=None)
    Bases: keras.initializers.Initializer
    call complexnn.init.IndependentFilters (kernel_size, input_dim, weight_dim, nb_filters=None, criterion='glorot', seed=None)
    Bases: keras.initializers.Initializer
    get_config ()

class complexnn.init.SqrtInit
    Bases: keras.initializers.Initializer
    complexnn.init.complex_init
        alias of complexnn.init.ComplexInit
    complexnn.init.independent_filters
        alias of complexnn.init.IndependentFilters
    complexnn.init.sqr_init
        alias of complexnn.init.SqrtInit
```

complexnn.norm module

```
class complexnn.norm.ComplexLayerNorm (epsilon=0.0001, axis=-1, center=True, scale=True,
gamma_diag_initializer=<function sqrt_init>, gamma_off_initializer='zeros',
beta_regularizer=None, gamma_diag_regularizer=None,
gamma_off_regularizer=None, beta_constraint=None,
gamma_diag_constraint=None, gamma_off_constraint=None, **kwargs)
    Bases: keras.engine.base_layer.Layer
    build (input_shape)
        Creates the layer weights.

        Must be implemented on all layers that have weights.

    # Arguments
        input_shape: Keras tensor (future input to layer) or list/tuple of Keras tensors to reference for weight shape computations.
    call (inputs)
        This is where the layer's logic lives.

    # Arguments
        inputs: Input tensor, or list/tuple of input tensors.
        **kwargs: Additional keyword arguments.

    # Returns
        A tensor or list/tuple of tensors.

get_config ()
    Returns the config of the layer.

    A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer can be reinstantiated later (without its trained weights) from this configuration.

    The config of a layer does not include connectivity information, nor the layer class name. These are handled by Network (one layer of abstraction above).
```

```
# Returns Python dictionary.
class complexnn.norm.LayerNormalization (epsilon=0.0001, axis=-1, beta_init='zeros',
gamma_init='ones', gamma_regularizer=None, beta_regularizer=None,**kwargs)
Bases: keras.engine.base_layer.Layer
build (input_shape)
Creates the layer weights.
Must be implemented on all layers that have weights.
# Arguments
    input_shape: Keras tensor (future input to layer) or list/tuple of Keras tensors to reference for
weight shape computations.
call (x, mask=None)
This is where the layer's logic lives.
# Arguments
    inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword argu-
ments.
# Returns
    A tensor or list/tuple of tensors.
get_config ()
Returns the config of the layer.
A layer config is a Python dictionary (serializable) containing the configuration of a layer. The same layer
can be reinstantiated later (without its trained weights) from this configuration.
The config of a layer does not include connectivity information, nor the layer class name. These are
handled by Network (one layer of abstraction above).
# Returns
    Python dictionary.
complexnn.norm.LayerNorm (x, axis, epsilon, gamma, beta)

complexnn.pool module

class complexnn.pool.SpectralPooling2D (topf=(0,))
Bases: keras.engine.base_layer.Layer
call (x, mask=None)
This is where the layer's logic lives.
# Arguments
    inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword argu-
ments.
# Returns
    A tensor or list/tuple of tensors.
class complexnn.pool.SpectralPooling2D (**kwargs)
Bases: keras.engine.base_layer.Layer
call (x, mask=None)
This is where the layer's logic lives.
# Arguments
    inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword argu-
ments.
# Returns
    A tensor or list/tuple of tensors.
```

```
complexnn.utils module

class complexnn.utils.GetAbs (**kwargs)
Bases: keras.engine.base_layer.Layer
call (inputs)
This is where the layer's logic lives.
# Arguments
    inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword argu-
ments.
# Returns
    A tensor or list/tuple of tensors.
compute_output_shape (input_shape)
Computes the output shape of the layer.
Assumes that the layer will be built to match that input shape provided.
# Arguments
    input_shape: Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the
layer). Shape tuples can include None for free dimensions, instead of an integer.
# Returns
    An output shape tuple.
class complexnn.utils.GetImag (**kwargs)
Bases: keras.engine.base_layer.Layer
call (inputs)
This is where the layer's logic lives.
# Arguments
    inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword argu-
ments.
# Returns
    A tensor or list/tuple of tensors.
compute_output_shape (input_shape)
Computes the output shape of the layer.
Assumes that the layer will be built to match that input shape provided.
# Arguments
    inputs: Input tensor, or list/tuple of input tensors. **kwargs: Additional keyword argu-
ments.
# Returns
    A tensor or list/tuple of tensors.
compute_output_shape (input_shape)
Computes the output shape of the layer.
Assumes that the layer will be built to match that input shape provided.
# Arguments
```

input_shape: Shape tuple (tuple of integers) or list of shape tuples (one per output tensor of the layer). Shape tuples can include None for free dimensions, instead of an integer.

Returns An output shape tuple.

```
complexnn.utils.get_abs(x)
complexnn.utils.get_imagpart(x)
complexnn.utils.get_realpart(x)
complexnn.utils.getpart_output_shape(input_shape)
```

Module contents

1.4 How to Contribute

1.5 Implementation and Math

Complex convolutional networks provide the benefit of explicitly modelling the phase space of physical systems [TBZ+17]. The complex convolution introduced can be explicitly implemented as convolutions of the real and complex components of both kernels and the data. A complex-valued data matrix in cartesian notation is defined as $\mathbf{M} = M_R + iM_S$ and equally, the complex-valued convolutional kernel is defined as $\mathbf{K} = K_R + iK_S$. The individual coefficients (M_R, M_S, K_R, K_S) are real-valued matrices, considering vectors are special cases of matrices with one of two dimensions being one.

1.5.1 Complex Convolution Math

The math for complex convolutional networks is similar to real-valued convolutions, with real-valued convolutions being:

$$\int_{\mathbf{R}^d} f(y) \cdot g(x - y) dy$$

which generalizes to complex-valued function on \mathbf{R}^d :

$$(f * g)(x) = \int_{\mathbf{R}^d} f(y)g(x - y) dy = \int_{\mathbf{R}^d} f(x - y)g(y) dy.$$

in order for the integral to exist, f and g need to decay sufficiently rapidly at infinity [CC-BY-SA Wiki].

1.5.2 Implementation

Solving the convolution of, implemented by [TBZ+17], translated to keras in [DC19]

$$M' = K * M = (M_R + iM_S) * (K_R + iK_S),$$

we can apply the distributivity of convolutions to obtain

$$M' = \{M_R * K_R - M_S * K_S\} + i\{M_R * K_S + M_S * K_R\},$$

where K is the Kernel and M is a data vector.

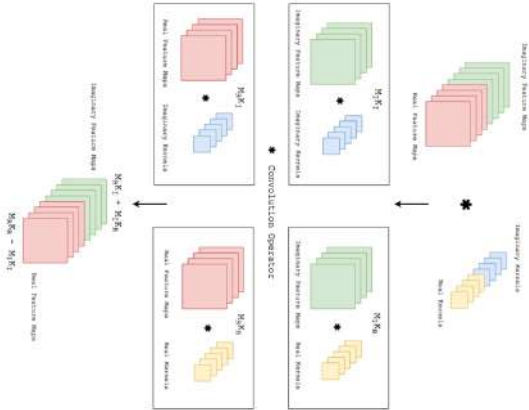


Fig. 1: Complex Convolution implementation (CC-BY [TBZ+17])

1.5.3 Considerations

Complex convolutional neural networks learn by back-propagation. [SSC15] state that the activation functions, as well as the loss function must be complex differentiable (holomorphic). [TBZ+17] suggest that employing complex losses and activation functions is valid for speed, however, refers that [HY12] show that complex-valued networks can be optimized individually with real-valued loss functions and contain piecewise real-valued activations. We reimplement the code [TBZ+17] provides in keras with tensorflow , which provides convenience functions implementing a multitude of real-valued loss functions and activations.

[CC-BY [DLuthjeC19]]

1.6 Citation

Please cite the original work as:

```
@ARTICLE{Trabelsi2017,
  author = "Chihed Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitry Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, Christopher J Pal",
  title = "Deep Complex Networks",
  journal = "arXiv preprint arXiv:1705.09792",
  year = "2017"
```

Cite this software version as:

```
@misc{dramsch2019complex,
  title = {Complex-Valued Neural Networks in Keras with TensorFlow},
  url = {https://figshare.com/articles/Complex-Valued_Neural_Networks_in_
~Keras_with_TensorFlow/9783773/1},
  DOI = {10.6084/m9.figshare.9783773},
  publisher = {figshare},
  author = {Dramschi, Jesper S{\o}ren and Contributors},
  year = {2019}
}
```

CHAPTER 2

Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

Bibliography

- [DC19] Jesper Soeren Dranssch and Contributors. Complex-valued neural networks in keras with tensorflow. 2019. URL: https://figshare.com/articles/Complex_Valued_Neural_Networks_in_Keras_with_Tensorflow/9783773/1, doi:10.6084/m9.figshare.9783773.
- [DLuthjC19] Jesper Sören Dranssch, Mikael Luthje, and Anders Nymark Christensen. Complex-valued neural networks for machine learning on non-stationary physical data. *arXiv preprint arXiv:1905.12321*, 2019.
- [HY12] Akira Hirose and Shota Yoshida. Generalization characteristics of complex-valued feedforward neural networks in relation to signal coherence. *IEEE Transactions on Neural Networks and Learning Systems*, 2012.
- [SSC15] Andy M. Sarroff, Victor Shepardon, and Michael A. Casey. Learning representations using complex-valued nets. *CoRR*, 2015. URL: <http://arxiv.org/abs/1511.06351>, arXiv:1511.06351.
- [TBZ+17] Chibeb Trabelsi, Olexa Blumick, Ying Zhang, Dmitry Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

Python Module Index

C

- `complexnn`, 19
- `complexnn.bn`, 4
- `complexnn.conv`, 6
- `complexnn.dense`, 13
- `complexnn.fft`, 15
- `complexnn.init`, 15
- `complexnn.norm`, 16
- `complexnn.pool`, 17
- `complexnn.utils`, 18

Index

B

build() (complex.bn.ComplexBatchNormalization method), 5

build() (complex.conv.ComplexConv method), 7

build() (complex.conv.WeightNorm_Conv method), 12

build() (complex.dense.ComplexDense method), 14

build() (complex.norm.ComplexLayerNorm method), 16

build() (complex.norm.LayerNormalization method), 17

C

call() (complex.bn.ComplexBatchNormalization method), 5

call() (complex.conv.ComplexConv method), 7

call() (complex.conv.WeightNorm_Conv method), 12

call() (complex.dense.ComplexDense method), 14

call() (complex.fft.FFT method), 15

call() (complex.fft.FFT2 method), 15

call() (complex.fft.IFFT method), 15

call() (complex.fft.IFFT2 method), 15

call() (complex.fft.IFFT2 method), 15

call() (complex.norm.ComplexLayerNorm method), 16

call() (complex.norm.LayerNormalization method), 17

call() (complex.pool.SpectralPooling1D method), 17

call() (complex.pool.SpectralPooling2D method), 17

call() (complex.util.GetAbs method), 18

call() (complex.util.GetImag method), 18

call() (complex.util.GetReal method), 18

complex_init (in module complex.init), 16

complex_standardization() (in module complex.bn), 5

ComplexBatchNormalization (class in complex.bn), 4

ComplexBN() (in module complex.bn), 4

ComplexConv (class in complex.conv), 6

ComplexConv1D (class in complex.conv), 7

ComplexConv2D (class in complex.conv), 9

ComplexConv3D (class in complex.conv), 10

ComplexConvolution1D (in module complex.conv), 12

ComplexConvolution2D (in module complex.conv), 12

ComplexConvolution3D (in module complex.conv), 12

ComplexDense (class in complex.dense), 13

ComplexIndependentFilters (class in complex.init), 15

ComplexInit (class in complex.init), 15

ComplexLayerNorm (class in complex.norm), 16

complex.bn (module), 4

complex.bn.conv (module), 6

complex.bn.dense (module), 13

complex.bn.fft (module), 15

complex.bn.init (module), 15

complex.bn.norm (module), 16

complex.bn.pool (module), 17

complex.bn.util (module), 18

compute_output_shape() (in module complex.conv.ComplexConv method), 7

compute_output_shape() (in module complex.dense.ComplexDense method), 14

compute_output_shape() (in module complex.util.GetAbs method), 18

compute_output_shape() (in module complex.util.GetImag method), 18

compute_output_shape() (in module complex.util.GetReal method), 18

conv2d_transpose() (in module complex.conv.conv_transpose_output_length() (in module complex.conv), 13

F

FFT (class in complex.fft), 15

fft() (in module complex.fft), 15

FFT2 (class in complex.fft), 15

fft2() (in module complex.fft), 15

G

get_abs() (in module complex.util), 19

get_config() (complex.bn.ComplexBatchNormalization method), 5

get_config() (complex.conv.ComplexConv method), 7

get_config() (complex.conv.ComplexConv1D method), 9

get_config() (complex.conv.ComplexConv2D method), 10

get_config() (complex.conv.ComplexConv3D method), 12

get_config() (complex.conv.WeightNorm_Conv method), 13

get_config() (complex.dense.ComplexDense method), 14

get_config() (complex.init.ComplexIndependentFilters method), 15

get_config() (complex.init.IndependentFilters method), 16

get_config() (complex.norm.ComplexLayerNorm method), 16

get_config() (complex.norm.LayerNormalization method), 17

get_imagpart() (in module complex.util), 19

get_realpart() (in module complex.util), 19

GetAbs (class in complex.util), 18

GetImag (class in complex.util), 18

getpart_output_shape() (in module complex.util), 19

GetReal (class in complex.util), 18

I

IEFT (class in complex.fft), 15

ieft() (in module complex.conv), 13

ieft() (in module complex.fft), 15

IEFT2 (class in complex.fft), 15

ieft2() (in module complex.conv), 13

ieft2() (in module complex.fft), 15

independent_filters (in module complex.init), 16

IndependentFilters (class in complex.init), 16

L

layernorm() (in module complex.norm), 17

S

sanitizedinitGet() (in module complex.bn), 5

sanitizedinitGet() (in module complex.bn), 13

sanitizedinitGet() (in module complex.bn), 5

sanitizedinitGet() (in module complex.conv), 13

SpectralPooling1D (class in complex.pool), 17

SpectralPooling2D (class in complex.pool), 17

sqr_init (in module complex.init), 16

sqr_init() (in module complex.bn), 5

SqrInit (class in complex.init), 16

W

WeightNorm_Conv (class in complex.conv), 12

Acronyms

AE	AutoEncoder	69
AI	Artificial Intelligence	
ASI	Automatic Seismic Interpretation	139
AVO	Amplitude versus Offset	2
BSEM	Backscatter Scanning-Electron Microscopy	45
CNN	Convolutional Neural Network	8
DIW	Dynamic Image Warping	114
DL	Deep Learning	
DNN	Deep Neural Network	2
DTW	Dynamic Time Warping	7
EC	Evolutionary Computing.....	57
FCN	Fully Convolutional Network.....	69
FK	Frequency-Wavenumber.....	71
FLOP	Floating-Point Operations	57
FOSS	Free Open Source Software	
GMM	Gaussian mixture model	45
GP	Gaussian Process	
HBR	Hatchell-Bourne-Røste	6
LDDMM	Large Deformation Diffeomorphic Metric Mapping.....	109
ML	Machine Learning	
MSE	Mean Squared Error	
NAS	Neural Architecture Search	57
NLP	Natural Language Processing	
NN	Neural Network.....	5
NRMS	Normalized Root Mean Squared Error.....	6
ODE	Ordinary Differential Equation.....	110
QI	Quantitative Interpretation	7

RF	Random Forest	
RL	Reinforcement Learning	
RNN	Recurrent Neural Network	57
SGD	Stochastic Gradient Descent	59
SIFT	Scale-Invariant Feature Transform	111
SNR	Signal-to-Noise Ratio	56
SOTA	state-of-the-art	59
SVM	Support Vector Machine	
TF	Tensorflow	
TPE	Tree of Parzen Estimator	95
VAE	Variational AutoEncoder	

Bibliography

- Aabø, T. M., J. S. Dramsch, M. Welch, and M. Luthje (June 2017). “Correlation of Fractures From Core, Borehole Images and Seismic Data in a Chalk Reservoir in the Danish North Sea”. In: *79th EAGE Conference and Exhibition 2017*. Published, Appendix C.2. EAGE. DOI: 10.3997/2214-4609.201701283. URL: <https://doi.org/10.3997/2214-4609.201701283> (cit. on p. 184).
- Aabø, T. M., J. S. Dramsch, C. L. Würtzen, S. Seyum, F. Amour, M. Welch, and M. Luthje (2020). “An integrated workflow for fracture characterization in chalk reservoirs, applied to the Kraka Field”. In: *Marine and Petroleum Geology* 112. Published, Appendix B. ISSN: 0264-8172. DOI: <https://doi.org/10.1016/j.marpetgeo.2019.104065>. URL: <http://www.sciencedirect.com/science/article/pii/S026481721930501X> (cit. on p. 143).
- Agterberg, F. (1966). “Markov schemes for multivariate well data”. In: *Proceedings, symposium on applications of computers and operations research in the mineral industries, Pennsylvania State University, State College, Pennsylvania*. Vol. 2, pp. X1–x18 (cit. on p. 12).
- Aizerman, M. A. (1964). “Theoretical foundations of the potential function method in pattern recognition learning”. In: *Automation and remote control* 25, pp. 821–837 (cit. on p. 21).
- Alaudah, Y., P. Michalowicz, M. Alfarraj, and G. AlRegib (2019). “A Machine Learning Benchmark for Facies Classification”. In: *arXiv preprint arXiv:1901.07659* (cit. on pp. 69, 79).
- Alfarraj, M. and G. AlRegib (2019). “Petrophysical property estimation from seismic data using recurrent neural networks”. In: *arXiv preprint arXiv:1901.08623* (cit. on p. 37).
- Alikhassi, A., H. E. Gourabi, and M. Baikpour (2018). “Comparison of inter-and intra-observer variability of breast density assessments using the fourth and fifth editions of Breast Imaging Reporting and Data System”. In: *European journal of radiology open* 5, pp. 67–72 (cit. on p. 55).
- Amini, H. (2018). “Comparison of Xu-White, Simplified Xu-White (Keys & Xu) and Nur’s Critical Porosity in Shaley Sands”. In: *80th EAGE Conference and Exhibition 2018* (cit. on p. 103).
- Anifowose, F., C. Ayadiuno, and F. Rashedian (2017). “Carbonate Reservoir Cementation Factor Modeling Using Wireline Logs and Artificial Intelligence Methodology”. In: *79th EAGE Conference and Exhibition 2017-Workshops*. earthdoc.org. URL: <http://earthdoc.org>

- [//www.earthdoc.org/publication/publicationdetails/?publication=89285](http://www.earthdoc.org/publication/publicationdetails/?publication=89285) (cit. on p. 23).
- Araya-Polo, M., T. Dahlke, C. Frogner, C. Zhang, T. Poggio, and D. Hohl (Mar. 2017). “Automated fault detection without seismic processing”. In: *Lead. Edge* 36.3, pp. 208–214. ISSN: 1070-485X. DOI: 10.1190/tle36030208.1. URL: <https://doi.org/10.1190/tle36030208.1> (cit. on p. 34).
- Arts, R., O. Eiken, A. Chadwick, P. Zweigel, L. van der Meer, and B. Zinszner (July 2004). “Monitoring of CO₂ injected at Sleipner using time-lapse seismic data”. In: *Energy* 29.9-10, pp. 1383–1392. ISSN: 0360-5442. DOI: 10.1016/j.energy.2004.03.072. URL: <http://www.sciencedirect.com/science/article/pii/S0360544204001550> (cit. on p. 5).
- Athey, S. and G. Imbens (2016). “Recursive partitioning for heterogeneous causal effects”. In: *Proceedings of the National Academy of Sciences* 113.27, pp. 7353–7360 (cit. on p. 44).
- Baan, M. van der and C. Jutten (July 2000). “Neural networks in geophysical applications”. In: *Geophysics* 65.4, pp. 1032–1047. ISSN: 0016-8033. DOI: 10.1190/1.1444797. URL: <https://doi.org/10.1190/1.1444797> (cit. on p. 17).
- Babakhin, Y., A. Sanakoyeu, and H. Kitamura (2019a). “Semi-Supervised Segmentation of Salt Bodies in Seismic Images using an Ensemble of Convolutional Neural Networks”. In: *German Conference on Pattern Recognition (GCPR)*, pp. 218–231 (cit. on pp. 36, 69).
- Babakhin, Y., A. Sanakoyeu, and H. Kitamura (2019b). “Semi-supervised segmentation of salt bodies in seismic images using an ensemble of convolutional neural networks”. In: *German Conference on Pattern Recognition*. Springer, pp. 218–231 (cit. on p. 70).
- Bajcsy, R. and S. Kovačič (1989). “Multiresolution elastic matching”. In: *Computer vision, graphics, and image processing* 46.1, pp. 1–21 (cit. on pp. 109, 118).
- Balakrishnan, G., A. Zhao, M. R. Sabuncu, J. Guttag, and A. V. Dalca (2019). “VoxelMorph: a learning framework for deformable medical image registration”. In: *IEEE transactions on medical imaging* (cit. on pp. 111, 116, 121, 122).
- Ballabio, C. and S. Sterlacchini (Jan. 2012). “Support Vector Machines for Landslide Susceptibility Mapping: The Staffora River Basin Case Study, Italy”. In: *Math. Geosci.* 44.1, pp. 47–70. ISSN: 1874-8961, 1874-8953. DOI: 10.1007/s11004-011-9379-9. URL: <https://doi.org/10.1007/s11004-011-9379-9> (cit. on p. 23).
- Bansal, A., S. Ma, D. Ramanan, and Y. Sheikh (2018). “Recycle-gan: Unsupervised video retargeting”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 119–135 (cit. on pp. 111, 120).
- Barnes, A. E. (2007). “A tutorial on complex seismic trace analysis”. In: *Geophysics* 72.6, W33–W43 (cit. on p. 79).
- Baxter, J. (1998). “Theoretical models of learning to learn”. In: *Learning to learn*. Springer, pp. 71–94 (cit. on p. 62).
- Bayes, T. (1763). “LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S”. In: *Philosophical transactions of the Royal Society of London* 53, pp. 370–418 (cit. on p. 10).

- Beg, M. F., M. I. Miller, A. Trouvé, and L. Younes (2005). “Computing large deformation metric mappings via geodesic flows of diffeomorphisms”. In: *International journal of computer vision* 61.2, pp. 139–157 (cit. on pp. 109, 110, 118).
- Belson, W. A. (1959). “Matching and prediction on the principle of biological classification”. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 8.2, pp. 65–75 (cit. on p. 11).
- Bergen, K. J., P. A. Johnson, V. Maarten, and G. C. Beroza (2019). “Machine learning for data-driven discovery in solid Earth geoscience”. In: *Science* 363.6433, eaau0323 (cit. on p. 9).
- Bergstra, J. S., R. Bardenet, Y. Bengio, and B. Kégl (2011). “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger. Curran Associates, Inc., pp. 2546–2554. URL: <http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf> (cit. on p. 104).
- Bergstra, J., B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox (2015). “Hyperopt: a python library for model selection and hyperparameter optimization”. In: *Computational Science & Discovery* 8.1, p. 014008 (cit. on p. 95).
- Bergstra, J., D. Yamins, and D. D. Cox (2013). “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures”. In: (cit. on p. 104).
- Bestagini, P., V. Lipari, and S. Tubaro (Aug. 2017). “A machine learning approach to facies classification using well logs”. In: *SEG Technical Program Expanded Abstracts 2017*. SEG Technical Program Expanded Abstracts. Society of Exploration Geophysicists, pp. 2137–2142. DOI: 10.1190/segam2017-17729805.1. URL: <https://doi.org/10.1190/segam2017-17729805.1> (cit. on p. 25).
- Beyreuther, M. and J. Wassermann (Dec. 2008). “Continuous earthquake detection and classification using discrete Hidden Markov Models”. In: *Geophys. J. Int.* 175.3, pp. 1055–1066. ISSN: 0956-540x. DOI: 10.1111/j.1365-246X.2008.03921.x. URL: <https://academic.oup.com/gji/article-abstract/175/3/1055/634811> (cit. on p. 26).
- Bicego, M., C. Acosta-Muñoz, and M. Orozco-Alzate (June 2013). “Classification of Seismic Volcanic Signals Using Hidden-Markov-Model-Based Generative Embeddings”. In: *IEEE Trans. Geosci. Remote Sens.* 51.6, pp. 3400–3409. ISSN: 0196-2892. DOI: 10.1109/tgrs.2012.2220370. URL: <http://dx.doi.org/10.1109/TGRS.2012.2220370> (cit. on p. 26).
- Bishop, C. M. (1995). “Training with noise is equivalent to Tikhonov regularization”. In: *Neural computation* 7.1, pp. 108–116 (cit. on p. 95).
- Blondelle, H., A. Juneja, J. Micaelli, and P. Neri (2017). “Machine Learning can extract the information needed for modelling and data analysing from unstructured documents”. In: *79th EAGE Conference and Exhibition 2017-Workshops*. earthdoc.org. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=89273> (cit. on p. 37).
- Blouin, M., A. Caté, L. Perozzi, and E. Gloaguen (2017). “Automated facies prediction in drillholes using machine learning”. In: *79th EAGE Conference and Exhibition*

- 2017-Workshops. earthdoc.org. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=89276> (cit. on p. 25).
- Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. (2016). “End to end learning for self-driving cars”. In: *arXiv preprint arXiv:1604.07316* (cit. on p. 42).
- Bond, C. E., A. D. Gibbs, Z. K. Shipton, S. Jones, et al. (2007). “What do you think this is? ‘Conceptual uncertainty’ in geoscience interpretation”. In: *GSA today* 17.11, p. 4 (cit. on pp. 41, 55).
- Breiman, L. (2001). “Random forests”. In: *Machine learning* 45.1, pp. 5–32 (cit. on pp. 13, 22).
- Brown, T. B., D. Mané, A. Roy, M. Abadi, and J. Gilmer (2017). “Adversarial patch”. In: *arXiv preprint arXiv:1712.09665* (cit. on p. 56).
- Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. (2020). “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165* (cit. on p. 59).
- Bruges: Bag of really useful geophysical equations and stuff (2016). URL: <https://github.com/agile-geoscience/bruges> (cit. on p. xiii).
- Bryson, A. E. (1961). “A gradient method for optimizing multi-stage allocation processes”. In: *Proc. Harvard Univ. Symposium on digital computers and their applications*. Vol. 72 (cit. on p. 11).
- Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux (2013). “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122 (cit. on pp. 18, 19).
- Cao, J. and B. Roy (Mar. 2017). “Time-lapse reservoir property change estimation from seismic using machine learning”. In: *Lead. Edge* 36.3, pp. 234–238. ISSN: 1070-485x. DOI: 10.1190/tle36030234.1. URL: <https://doi.org/10.1190/tle36030234.1> (cit. on p. 25).
- Castagna, J. P. and M. M. Backus (1993). *Offset-dependent reflectivity—Theory and practice of AVO analysis*. Society of Exploration Geophysicists (cit. on p. 93).
- Caté, A., L. Perozzi, E. Gloaguen, and M. Blouin (Mar. 2017). “Machine learning as a tool for geologists”. In: *Lead. Edge* 36.3, pp. 215–219. ISSN: 1070-485x. DOI: 10.1190/tle36030215.1. URL: <https://doi.org/10.1190/tle36030215.1> (cit. on p. 26).
- Caté, A., E. Schetselaar, P. Mercier-Langevin, and P.-S. Ross (2018). “Classification of lithostratigraphic and alteration units from drillhole lithogeochemical data using machine learning: A case study from the Lalor volcanogenic massive sulphide deposit, Snow Lake, Manitoba, Canada”. In: *J. Geochem. Explor.* 188, pp. 216–228. ISSN: 0375-6742. URL: <https://www.sciencedirect.com/science/article/pii/S0375674217305083> (cit. on pp. 23, 25).
- Chaki, S., A. Routray, and W. K. Mohanty (Mar. 2018). “Well-Log and Seismic Data Integration for Reservoir Characterization: A Signal Processing and Machine-Learning Perspective”. In: *IEEE Signal Process. Mag.* 35.2, pp. 72–81. ISSN: 1053-5888. DOI:

- 10.1109/msp.2017.2776602. URL: <http://dx.doi.org/10.1109/MSP.2017.2776602> (cit. on p. 23).
- Chan, S. and A. H. Elsheikh (2017). “Parametrization and generation of geological models with generative adversarial networks”. In: *arXiv preprint arXiv:1708.01810* (cit. on p. 37).
- Chang, C.-C. and C.-J. Lin (May 2011). “LIBSVM: A Library for Support Vector Machines”. In: *ACM Trans. Intell. Syst. Technol.* 2.3, 27:1–27:27. ISSN: 2157-6904. DOI: 10.1145/1961189.1961199. URL: <http://doi.acm.org/10.1145/1961189.1961199> (cit. on pp. 11, 18).
- Chen, T. and C. Guestrin (2016). “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Kdd ’16. San Francisco, California, USA: Acm, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: <http://doi.acm.org/10.1145/2939672.2939785> (cit. on p. 11).
- Chen, Y., H. Fang, B. Xu, Z. Yan, Y. Kalantidis, M. Rohrbach, S. Yan, and J. Feng (2019). “Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution”. In: *arXiv preprint arXiv:1904.05049* (cit. on p. 141).
- Cherrett, A., I. Escobar, and H. Hansen (2011). “Fast deterministic geostatistical inversion”. In: *73rd EAGE Conference and Exhibition incorporating SPE EUROPEC 2011* (cit. on pp. 109, 117).
- Chevitarese, D., D. Szwarcman, R. M. D. Silva, and E. V. Brazil (2018). “Seismic facies segmentation using deep learning”. In: *ACE 2018 Annual Convention & Exhibition*. searchanddiscovery.com. URL: http://www.searchanddiscovery.com/documents/2018/42286chevitarese/ndx%5C_chevitarese.pdf (cit. on p. 35).
- Chilès, J.-P. and N. Desassis (2018). “Fifty years of kriging”. In: *Handbook of mathematical geosciences*. Springer, pp. 589–612 (cit. on p. 12).
- Chiles, J. and P. Chauvet (1975). “Kriging: a method for cartography of the sea floor”. In: *The International Hydrographic Review* (cit. on p. 12).
- Ching, T., D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, M. Zietz, M. M. Hoffman, W. Xie, G. L. Rosen, B. J. Lengerich, J. Israeli, J. Lanchantin, S. Woloszynek, A. E. Carpenter, A. Shrikumar, J. Xu, E. M. Cofer, C. A. Lavender, S. C. Turaga, A. M. Alexandari, Z. Lu, D. J. Harris, D. DeCaprio, Y. Qi, A. Kundaje, Y. Peng, L. K. Wiley, M. H. S. Segler, S. M. Boca, S. J. Swamidass, A. Huang, A. Gitter, and C. S. Greene (Apr. 2018). “Opportunities and obstacles for deep learning in biology and medicine”. en. In: *J. R. Soc. Interface* 15.141. ISSN: 1742-5689, 1742-5662. DOI: 10.1098/rsif.2017.0387. URL: <http://dx.doi.org/10.1098/rsif.2017.0387> (cit. on p. 9).
- Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (cit. on p. 37).
- Chollet, F. et al. (2015a). *Keras*. <https://keras.io> (cit. on pp. 62, 76, 122).
- Chollet, F. et al. (2015b). *Keras*. <https://github.com/fchollet/keras> (cit. on pp. 100, 104).

- Chopra, S., R. Hadsell, Y. LeCun, et al. (2005). “Learning a similarity metric discriminatively, with application to face verification”. In: *CVPR (1)*, pp. 539–546 (cit. on pp. 111, 119).
- Christensen, G. E., R. D. Rabbitt, and M. I. Miller (1994). “3D brain mapping using a deformable neuroanatomy”. In: *Physics in Medicine & Biology* 39.3, p. 609 (cit. on p. 118).
- Ciresan, D. C., U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber (2011). “Flexible, high performance convolutional neural networks for image classification”. In: *Twenty-Second International Joint Conference on Artificial Intelligence* (cit. on p. 13).
- Collobert, R., S. Bengio, and J. Mariéthoz (2002). *Torch: a modular machine learning software library*. Tech. rep. Idiap (cit. on pp. 11, 18).
- Corte, G., J. S. Dramsch, C. MacBeth, and H. Amini (2020). “Deep Neural Network Application for 4D Seismic Inversion to Pressure and Saturation: Enhancing Training Data Sets”. In: *82nd EAGE Annual Conference & Exhibition*. Vol. 2020. 1. Published. European Association of Geoscientists & Engineers, pp. 1–5 (cit. on pp. 108, 140).
- Côte, G., J. S. Dramsch, H. Amini, and C. MacBeth (2020). “Deep neural network application for 4D seismic inversion to changes in pressure and saturation: Optimizing the use of synthetic training datasets”. In: *Geophysical Prospecting* 68.7. Published, Appendix B.2, pp. 2164–2185 (cit. on pp. 108, 140, 156).
- Corte, G., C. MacBeth, and H. Amini (submitted 2019). “North Sea field application of 4D Bayesian inversion to pressure and saturation changes”. In: *81st EAGE Conference & Exhibition 2019*. submitted (cit. on p. 103).
- Cortes, C. and V. Vapnik (1995). “Support-vector networks”. In: *Machine learning* 20.3, pp. 273–297 (cit. on pp. 11, 13).
- Cover, T. and P. Hart (1967). “Nearest neighbor pattern classification”. In: *IEEE transactions on information theory* 13.1, pp. 21–27 (cit. on p. 11).
- Cracknell, M. J. and A. M. Reading (2013). “The upside of uncertainty: Identification of lithology contact zones from airborne geophysics and satellite data using random forests and support vector machines”. In: *Geophysics* 78.3, Wb113–wb126 (cit. on p. 23).
- Cressie, N. (1990). “The origins of kriging”. In: *Mathematical geology* 22.3, pp. 239–252 (cit. on p. 15).
- Dahl, G. E., T. N. Sainath, and G. E. Hinton (May 2013). “Improving deep neural networks for LVCSR using rectified linear units and dropout”. In: *2013 IEEE International Conference on Acoustics Speech and Signal Processing*. IEEE. DOI: 10.1109/icassp.2013.6639346. URL: <https://doi.org/10.1109/icassp.2013.6639346> (cit. on p. 61).
- Dalca, A. V., G. Balakrishnan, B. Fischl, P. Golland, J. Guttag, J. E. Iglesias, M. Rakic, M. R. Sabuncu, E. Yu, A. Zhao, et al. (2018a). *Voxelmorph*. voxelmorph.mit.edu (cit. on p. 122).
- Dalca, A. V., G. Balakrishnan, J. Guttag, and M. R. Sabuncu (2018b). “Unsupervised learning for fast probabilistic diffeomorphic registration”. In: *International Confer-*

- ence on Medical Image Computing and Computer-Assisted Intervention*. Springer, pp. 729–738 (cit. on pp. 111, 121–123).
- Dammeier, F., J. R. Moore, C. Hammer, F. Haslinger, and S. Loew (Feb. 2016). “Automatic detection of alpine rockslides in continuous seismic data using hidden Markov models”. In: *J. Geophys. Res. Earth Surf.* of the Ser. Lect. Notes in Comput. Sci 121.2, pp. 351–371. ISSN: 2169-9003. DOI: 10.1002/2015jf003647. URL: <http://doi.wiley.com/10.1002/2015JF003647> (cit. on p. 26).
- Dechter, R. (1986). *Learning while searching in constraint-satisfaction problems*. University of California, Computer Science Department, Cognitive Systems ... (cit. on p. 14).
- Delhomme, J. (1978). “Kriging in the hydrosociences”. In: *Advances in water resources* 1, pp. 251–266 (cit. on p. 12).
- Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009a). “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09* (cit. on p. 61).
- Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009b). “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, pp. 248–255 (cit. on pp. 11, 26).
- Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2018). “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (cit. on p. 42).
- DeVries, P. M., F. Viégas, M. Wattenberg, and B. J. Meade (2018). “Deep learning of aftershock patterns following large earthquakes”. In: *Nature* 560.7720, p. 632 (cit. on p. 41).
- Di, H., M. Shafiq, and G. AlRegib (2017a). “Multi-attribute k-means cluster analysis for salt boundary detection”. In: *79th EAGE Conference and Exhibition*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=88632> (cit. on p. 26).
- Di, H., M. Shafiq, and G. AlRegib (Aug. 2017b). “Seismic-fault detection based on multiattribute support vector machine analysis”. In: *SEG Technical Program Expanded Abstracts 2017*. SEG Technical Program Expanded Abstracts. Society of Exploration Geophysicists, pp. 2039–2044. DOI: 10.1190/segam2017-17748277.1. URL: <https://doi.org/10.1190/segam2017-17748277.1> (cit. on p. 23).
- Di, H., Z. Wang, and G. AlRegib (2018). “Deep convolutional neural networks for seismic salt-body delineation”. In: *AAPG Annual Convention and*. URL: http://www.searchanddiscovery.com/documents/2018/70360di/ndx%5C_di.pdf (cit. on p. 34).
- Dodge, D. A. and D. B. Harris (2016). “Large-scale test of dynamic correlation processors: Implications for correlation-based seismic pipelines”. In: *Bull. Seismol. Soc. Am.* URL: <https://pubs.geoscienceworld.org/ssa/bssa/article-abstract/106/2/435/332173> (cit. on p. 25).
- Dony, R. D. and S. Haykin (1995). “Neural network approaches to image compression”. In: *Proceedings of the IEEE* 83.2, pp. 288–303 (cit. on p. 94).
- Dosovitskiy, A., P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox (2015). “FlowNet: Learning optical flow with convolutional

- networks". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2758–2766 (cit. on pp. 110, 119).
- Dowla, F. U., S. R. Taylor, and R. W. Anderson (Oct. 1990). "Seismic discrimination with artificial neural networks: Preliminary results with regional spectral data". In: *Bull. Seismol. Soc. Am.* 80.5, pp. 1346–1373. ISSN: 0037-1106. URL: <https://pubs.geoscienceworld.org/ssa/bssa/article-abstract/80/5/1346/119382> (cit. on p. 12).
- Dramsch, J. S., G. Corte, H. Amini, M. Lüthje, and C. MacBeth (Feb. 2019a). *Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data*. URL: eartharxiv.org/sj7zf (cit. on pp. 97, 99).
- Dramsch, J. S. (July 2020a). *3D decision volume of SVM, Random Forest, and Deep Neural Network*. DOI: 10.6084/m9.figshare.12640226.v1. URL: https://figshare.com/articles/media/3D%5C_decision%5C_volume%5C_of%5C_SVM%5C_Random%5C_Forest%5C_and%5C_Deep%5C_Neural%5C_Network/12640226/1 (cit. on pp. 21, 31, 32).
- Dramsch, J. S. (Aug. 2020b). *4D seismic warping voxelmorph code*. DOI: 10.6084/m9.figshare.12808910.v1 (cit. on p. 123).
- Dramsch, J. S. (July 2020c). *Code for 70 Years of Machine Learning in Geoscience in Review*. DOI: 10.6084/m9.figshare.12666140.v1 (cit. on p. 18).
- Dramsch, J. S. (Dec. 15, 2018a). *Reproducible Code: Deep-learning seismic facies on state-of-the-art CNN architectures*. DOI: 10.6084/m9.figshare.7227545. URL: <https://github.com/JesperDramsch/seismic-transfer-learning> (cit. on pp. 70, 139).
- Dramsch, J. S. (2018b). *Reproducible Code: Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk*. URL: <https://github.com/JesperDramsch/backscatter-sem-segmentation> (cit. on p. 54).
- Dramsch, J. (2019a). "Machine Learning in 4D Seismic Data Analysis: Deep Neural Networks in Geophysics". English. PhD thesis (cit. on p. 11).
- Dramsch, J. S. (2019b). *Reproducible Code: Complex-valued neural networks for machine learning on non-stationary physical data*. URL: <https://github.com/JesperDramsch/Complex-CNN-Seismic> (cit. on p. 91).
- Dramsch, J. S. (Sept. 2020d). "70 years of machine learning in geoscience in review". In: *Advances in Geophysics*. Ed. by B. Moseley and L. Krischer. Published, Peer-Reviewed, Section 2.2. Academic Press. Chap. 4. ISBN: 9780128216699 (cit. on pp. 2, 8, 9, 44).
- Dramsch, J. S. (2020e). *Code for 70 Years of Machine Learning in Geoscience in Review*. DOI: 10.6084/m9.figshare.12666140.v1 (cit. on p. 44).
- Dramsch, J. S., F. Amour, and M. Lüthje (2018a). "Gaussian Mixture Models For Robust Unsupervised Scanning-Electron Microscopy Image Segmentation Of North Sea Chalk". In: *First EAGE/PESGB Workshop Machine Learning*. Published, Chapter 3. EAGE. DOI: 10.3997/2214-4609.201803014. URL: <https://doi.org/10.3997/2214-4609.201803014> (cit. on pp. 2, 39, 49, 139).
- Dramsch, J. S., A. N. Christensen, C. MacBeth, and M. Lüthje (2019b). "Deep Unsupervised 4D Seismic 3D Time-Shift Estimation with Convolutional Neural Networks".

- In: *IEEE Transactions in Geoscience and Remote Sensing*. In Review, Chapter 7 (cit. on pp. 2, 36, 39, 56, 109, 111, 112, 115, 140).
- Dramsch, J. S. and Contributors (2018b). *Awesome Open Geoscience*. Maintainer. URL: <https://github.com/softwareunderground/awesome-open-geoscience> (cit. on p. xiii).
- Dramsch, J. S. and Contributors (2019c). *Complex-Valued Neural Networks in Keras with Tensorflow*. Open-Source Software. DOI: 10.6084/m9.figshare.9783773. URL: <https://github.com/JesperDramsch/keras-complex> (cit. on pp. 91, 250).
- Dramsch, J. S., G. Corte, H. Amini, M. L  thje, and C. MacBeth (2019d). “Deep Learning Application for 4D Pressure Saturation Inversion Compared to Bayesian Inversion on North Sea Data”. In: *Second EAGE Workshop Practical Reservoir Monitoring 2019*. Published, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201900028 (cit. on pp. 2, 96, 97, 108, 140).
- Dramsch, J. S., G. Corte, H. Amini, C. MacBeth, and M. L  thje (2019e). “Including Physics in Deep Learning – An Example from 4D Seismic Pressure Saturation Inversion”. In: *81st EAGE Conference and Exhibition 2019 Workshop Programme*. Published, Chapter 6. EAGE. DOI: 10.3997/2214-4609.201901967. URL: <https://doi.org/10.3997/2214-4609.201901967> (cit. on pp. 2, 94, 96, 102, 108, 140).
- Dramsch, J. S. and M. L  thje (2018c). *Deep Learning: From Cats to 4D Seismic - Reducing cycle time and model training cost in asset management*. Tech. rep. Danish Hydrocarbon Research and Technology Centre. DOI: 10.6084/m9.figshare.7422629 (cit. on p. 2).
- Dramsch, J. S. and M. L  thje (2018d). “Deep-learning seismic facies on state-of-the-art CNN architectures”. In: *SEG Technical Program Expanded Abstracts 2018*. Published, Chapter 4. Society of Exploration Geophysicists, pp. 2036–2040. DOI: 10.1190/segam2018-2996783.1. URL: <https://doi.org/10.1190/segam2018-2996783.1> (cit. on pp. 33–36, 55, 56, 61, 97).
- Dramsch, J. S. and M. L  thje (Nov. 2018e). “Information Theory Considerations In Patch-Based Training Of Deep Neural Networks On Seismic Time-Series”. In: *First EAGE/PESGB Workshop Machine Learning*. EAGE. DOI: 10.3997/2214-4609.201803020. URL: <https://doi.org/10.3997/2214-4609.201803020> (cit. on p. 78).
- Dramsch, J. S. and M. L  thje (2018f). “Information Theory Considerations In Patch-Based Training Of Deep Neural Networks On Seismic Time-Series”. In: *First EAGE/PESGB Workshop Machine Learning*. Published, Appendix D. EAGE. DOI: 10.3997/2214-4609.201803020. URL: <https://doi.org/10.3997/2214-4609.201803020> (cit. on p. 189).
- Dramsch, J. S., M. L  thje, and A. N. Christensen (2019f). “Complex-valued neural networks for machine learning on non-stationary physical data”. In: *Computers & Geoscience*. Accepted, Chapter 5 (cit. on pp. 2, 71, 72, 91, 139).
- Dreyfus, S. (1962). “The numerical solution of variational problems”. In: *Journal of Mathematical Analysis and Applications* 5.1, pp. 30–45 (cit. on p. 11).
- Dubrule, O. (1984). “Comparing splines and kriging”. In: *Computers & Geosciences* 10.2-3, pp. 327–338 (cit. on p. 12).

- Dupont, E., T. Zhang, P. Tilke, L. Liang, and W. Bailey (2018). “Generating realistic geology conditioned on physical measurements with generative adversarial networks”. In: *arXiv preprint arXiv:1802.03065* (cit. on p. 37).
- Duvenaud, D. (2014). “Automatic model construction with Gaussian processes”. PhD thesis. University of Cambridge (cit. on p. 16).
- Engel, J., K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts (2019). “Gansynth: Adversarial neural audio synthesis”. In: *arXiv preprint arXiv:1902.08710* (cit. on p. 36).
- Feng, X.-T. and M. Seto (July 1998). “Neural network dynamic modelling of rock microfracturing sequences under triaxial compressive stress conditions”. In: *Tectonophysics* 292.3, pp. 293–309. ISSN: 0040-1951. DOI: 10.1016/S0040-1951(98)00072-9. URL: <http://www.sciencedirect.com/science/article/pii/S0040195198000729> (cit. on p. 15).
- Ferreira, R., E. V. Brazil, R. Silva, et al. (2018). “Texture-Based Similarity Graph to Aid Seismic Interpretation”. In: *ACE 2018 Annual*. URL: http://www.searchanddiscovery.com/documents/2018/70365ferreira/ndx%5C_ferreira.pdf (cit. on p. 26).
- Forel, D., T. Benz, and W. D. Pennington (2005). *Seismic Data Processing with Seismic Un*x: A 2D Seismic Data Processing Primer*. Society of Exploration Geophysicists (cit. on p. 56).
- Fukushima, K. (1980). “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4, pp. 193–202 (cit. on p. 11).
- Gadot, D. and L. Wolf (2016). “PatchBatch: A batch augmented loss for optical flow”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4236–4245 (cit. on pp. 111, 119).
- Ghaderi, A. and M. Landrø (2005). “Pre-stack estimation of time-lapse seismic velocity changes—an example from the Sleipner CO₂-sequestration project”. In: *Greenhouse Gas Control Technologies 7*. Elsevier, pp. 633–641 (cit. on p. 7).
- Giorgino, T. et al. (2009). “Computing and visualizing dynamic time warping alignments in R: the dtw package”. In: *Journal of statistical Software* 31.7, pp. 1–24 (cit. on p. 114).
- Goodfellow, I. J., J. Shlens, and C. Szegedy (2014a). “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (cit. on p. 56).
- Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press (cit. on p. 28).
- Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014b). “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., pp. 2672–2680. URL: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf> (cit. on pp. 11, 36).
- Görtler, J., R. Kehlbeck, and O. Deussen (2019). “A Visual Exploration of Gaussian Processes”. In: *Distill*. <https://distill.pub/2019/visual-exploration-gaussian-processes>. DOI: 10.23915/distill.00017 (cit. on p. 16).

- Goshtasby, A. (1988). “Image registration by local approximation methods.” In: *Image and Vision Computing* 6.4, pp. 255–261 (cit. on pp. 109, 118).
- Gramstad, O. and M. Nickel (2018). “Automated Top Salt Interpretation Using a Deep Convolutional Net”. In: *80th EAGE Conference and Exhibition 2018*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=92117> (cit. on p. 34).
- Graves, A. (2012). “Sequence transduction with recurrent neural networks”. In: *arXiv preprint arXiv:1211.3711* (cit. on p. 43).
- Griffin, D. and J. Lim (1984). In: vol. 32. 2. IEEE, pp. 236–243. DOI: 10.1109/icassp.1983.1172092. URL: <https://doi.org/10.1109/icassp.1983.1172092> (cit. on p. 72).
- Grün, F., C. Rupprecht, N. Navab, and F. Tombari (2016). “A taxonomy and library for visualizing learned features in convolutional neural networks”. In: *arXiv preprint arXiv:1606.07757* (cit. on p. 97).
- Guillen, P., G. Larrazabal*, G. González, D. Bumber, and R. Vilalta (Aug. 2015). “Supervised learning to detect salt body”. In: *SEG Technical Program Expanded Abstracts 2015*. SEG Technical Program Expanded Abstracts. Society of Exploration Geophysicists, pp. 1826–1829. DOI: 10.1190/segam2015-5931401.1. URL: <https://doi.org/10.1190/segam2015-5931401.1> (cit. on p. 25).
- Guitton, A. (2018). “3D Convolutional Neural Networks for Fault Interpretation”. In: *80th EAGE Conference and Exhibition 2018*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=92118> (cit. on p. 34).
- Güney, F. and A. Geiger (2016). “Deep discrete flow”. In: *Asian Conference on Computer Vision*. Springer, pp. 207–224 (cit. on pp. 111, 119).
- Guo, R., Y. S. Zhang, H. Lin, and W. Liu (2017). “Sweet Spot Interpretation from Multiple Attributes: Machine Learning and Neural Networks Technologies”. In: *First EAGE/AMGP/AMGE Latin*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=90731> (cit. on p. 35).
- Gupta, I., C. Rai, C. Sondergeld, and D. Devegowda (Feb. 2018). “Rock typing in the Upper Devonian-Lower Mississippian Woodford Shale Formation, Oklahoma, USA”. In: *Interpretation* 6.1, Sc55–sc66. ISSN: 2324-8858. DOI: 10.1190/int-2017-0015.1. URL: <https://doi.org/10.1190/INT-2017-0015.1> (cit. on p. 23).
- Hahnloser, R. H., R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung (2000). “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. In: *Nature* 405.6789, pp. 947–951 (cit. on p. 27).
- Hajimoradlou, A., G. Roberti, and D. Poole (2019). “Predicting landslides using contour aligning convolutional neural networks”. In: *arXiv: Computer Vision and Pattern Recognition* (cit. on p. 36).
- Hale, D. (Mar. 2013a). “Methods to compute fault images, extract fault surfaces, and estimate fault throws from 3D seismic images”. In: *Geophysics* 78.2, O33–o43. ISSN: 0016-8033. DOI: 10.1190/geo2012-0331.1. URL: <https://doi.org/10.1190/geo2012-0331.1> (cit. on pp. 26, 109).
- Hale, D. (2006). “An efficient method for computing local cross-correlations of multi-dimensional signals”. In: *CWP Report* 656 (cit. on p. 117).

- Hale, D. (Mar. 2013b). “Dynamic warping of seismic images”. In: *GEOPHYSICS* 78.2, S105–S115. ISSN: 0016-8033. DOI: 10.1190/geo2012-0327.1. URL: <http://library.seg.org/doi/10.1190/geo2012-0327.1> (cit. on pp. 7, 112, 114).
- Hale, D. (2013c). “Dynamic warping of seismic images”. In: *Geophysics* 78.2, S105–S115 (cit. on pp. 109, 116, 128).
- Hall, B. (Oct. 2016). “Facies classification using machine learning”. In: *Lead. Edge* 35.10, pp. 906–909. ISSN: 1070-485x. DOI: 10.1190/tle35100906.1. URL: <https://doi.org/10.1190/tle35100906.1> (cit. on p. 23).
- Hall, M. and B. Hall (Mar. 2017). “Distributed collaborative prediction: Results of the machine learning contest”. In: *Lead. Edge* 36.3, pp. 267–269. ISSN: 1070-485x. DOI: 10.1190/tle36030267.1. URL: <https://doi.org/10.1190/tle36030267.1> (cit. on p. 25).
- Hall, S. A., C. MacBeth, O. I. Barkved, and P. Wild (Jan. 2002). “Time-lapse seismic monitoring of compaction and subsidence at Valhall through cross-matching and interpreted warping of 3D streamer and OBC data”. In: *SEG Technical Program Expanded Abstracts 2002*. Society of Exploration Geophysicists, pp. 1696–1699. DOI: 10.1190/1.1817004. URL: <http://library.seg.org/doi/abs/10.1190/1.1817004> (cit. on pp. 6, 7).
- Hall, S. A., C. MacBeth, O. I. Barkved, and P. Wild (2005). “Cross-matching with interpreted warping of 3D streamer and 3D ocean-bottom-cable data at Valhall for time-lapse assessment”. In: *Geophysical Prospecting* 53.2, pp. 283–297 (cit. on p. 117).
- Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media (cit. on p. 96).
- Hatchell, P. J., S. J. Bourne, and T. Netherlands. (Jan. 2005a). “Measuring reservoir compaction using time-lapse timeshifts”. In: *SEG/Houston 2005 Annual Meeting*, pp. 2500–2504. DOI: 10.1190/1.2148230. URL: <http://library.seg.org/doi/abs/10.1190/1.2148230> (cit. on p. 6).
- Hatchell, P. and S. Bourne (2005b). “Rocks under strain: Strain-induced time-lapse time shifts are observed for depleting reservoirs”. In: *The Leading Edge* 24.12, pp. 1222–1225 (cit. on pp. 109, 116).
- Hatchell, P., S. Bourne, and T. Netherlands (Dec. 2005c). “Rocks under strain: Strain-induced time-lapse time shifts are observed for depleting reservoirs”. In: *Lead. Edge* 24.12, pp. 1222–1225. ISSN: 1070-485x. DOI: 10.1190/1.2149624. URL: <http://library.seg.org/doi/10.1190/1.2149624> (cit. on p. 6).
- He, K., X. Zhang, S. Ren, and J. Sun (2015). “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034 (cit. on p. 97).
- He, K., X. Zhang, S. Ren, and J. Sun (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on pp. 33, 34, 57, 65, 141).
- He, T., Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li (2019). “Bag of tricks for image classification with convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 558–567 (cit. on p. 141).

- Hermes, L., D. Frieauff, J. Puzicha, and J. M. Buhmann (1999). "Support vector machines for land usage classification in Landsat TM imagery". In: *IEEE 1999 International Geoscience and Remote Sensing Symposium. IGARSS'99 (Cat. No. 99CH36293)*. Vol. 1. Ieee, pp. 348–350 (cit. on p. 13).
- Herwanger, J. (2015). "Seismic Geomechanics - How to build and calibrate geomechanical models using 3D and 4D seismic data". In: *EAGE Education Tour*, pp. 1–219 (cit. on p. 6).
- Higham, N. J. (2005). "The scaling and squaring method for the matrix exponential revisited". In: *SIAM Journal on Matrix Analysis and Applications* 26.4, pp. 1179–1193 (cit. on p. 121).
- Hinton, G. E. and R. R. Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pp. 504–507 (cit. on pp. 69, 77).
- Hirose, A. and S. Yoshida (Apr. 2012). "Generalization Characteristics of Complex-Valued Feedforward Neural Networks in Relation to Signal Coherence". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.4, pp. 541–551. DOI: 10.1109/tnnls.2012.2183613. URL: <https://doi.org/10.1109/tnnls.2012.2183613> (cit. on p. 76).
- Ho, T. K. (1995). "Random decision forests". In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. Ieee, pp. 278–282 (cit. on pp. 11, 13).
- Hochreiter, S. (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116 (cit. on p. 56).
- Hochreiter, S., Y. Bengio, P. Frasconi, J. Schmidhuber, et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies* (cit. on p. 27).
- Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780 (cit. on pp. 11, 13, 37).
- Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8, pp. 2554–2558 (cit. on p. 11).
- Hornik, K., M. Stinchcombe, and H. White (Jan. 1989). "Multilayer feedforward networks are universal approximators". In: *Neural Netw.* 2.5, pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. URL: <http://www.sciencedirect.com/science/article/pii/0893608089900208> (cit. on p. 17).
- Huang, G., Z. Liu, L. Van Der Maaten, and K. Q. Weinberger (2017). "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708 (cit. on pp. 57, 141).
- Huang, K. Y., W. R. I. Chang, and H. T. Yen (1990). "Self-organizing neural network for picking seismic horizons". In: *SEG Technical Program Expanded*, pp. 313–316. URL: <https://library.seg.org/doi/pdf/10.1190/1.1890183> (cit. on p. 12).
- Hui, T.-W., X. Tang, and C. Change Loy (2018). "Liteflownet: A lightweight convolutional neural network for optical flow estimation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8981–8989 (cit. on pp. 111, 119).

- Huijbregts, C. and G. Matheron (1970). “Universal kriging (an optimal method for estimating and contouring in trend surface analysis): 9th Intern”. In: *Sym. on Decisionmaking in the Mineral Industries (proceedings to be published by Canadian Inst. Mining)*, Montreal (cit. on p. 12).
- Hulbert, C., B. Rouet-Leduc, C. X. Ren, J. Riviere, D. C. Bolton, C. Marone, and P. A. Johnson (Jan. 2018). “Estimating the Physical State of a Laboratory Slow Slipping Fault from Seismic Signals”. In: arXiv: 1801.07806 [physics.geo-ph]. URL: <http://arxiv.org/abs/1801.07806> (cit. on p. 25).
- Ildstad, C. R. and P. Bormann (Oct. 22, 2017). *MalenoV_nD (MAchine LEarNing Of Voxels)*. Version d647f07. URL: <https://github.com/bolgebrygg/MalenoV> (cit. on pp. 34, 59).
- Ilg, E., N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox (2017). “FlowNet 2.0: Evolution of optical flow estimation with deep networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2462–2470 (cit. on pp. 111, 119).
- Ioffe, S. and C. Szegedy (2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (cit. on pp. 36, 76, 78, 81, 97, 141).
- Iqbal, H. (Dec. 2018). *HarisIqbal88/PlotNeuralNet v1.0.0*. DOI: 10.5281/zenodo.2526396. URL: <https://doi.org/10.5281/zenodo.2526396> (cit. on p. 33).
- Isola, P., J.-Y. Zhu, T. Zhou, and A. A. Efros (2017). “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134 (cit. on p. 120).
- Itakura, F. (1975). “Minimum Prediction Residual Principle Applied to Speech Recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1, pp. 67–72. ISSN: 00963518. DOI: 10.1109/TASSP.1975.1162641 (cit. on pp. 112, 113).
- Jeong, J., E. Park, W. S. Han, and K.-Y. Kim (Oct. 2014). “A novel data assimilation methodology for predicting lithology based on sequence labeling algorithms”. In: *J. Geophys. Res. [Solid Earth]* 119.10, pp. 7503–7520. ISSN: 2169-9313. DOI: 10.1002/2014jb011279. URL: <http://doi.wiley.com/10.1002/2014JB011279> (cit. on p. 26).
- Jewett, M. A., C. Bombardier, D. Caron, M. R. Ryan, R. R. Gray, E. L. S. Louis, S. J. Witchell, S. Kumra, and K. E. Psihramis (1992). “Potential for inter-observer and intra-observer variability in x-ray review to establish stone-free rates after lithotripsy”. In: *The Journal of urology* 147.3, pp. 559–562 (cit. on p. 41).
- Johnston, D. H. (Jan. 2013a). “6. Seismic Processing of 4D Data”. In: *Practical Applications of Time-lapse Seismic Data*. Society of Exploration Geophysicists, pp. 103–126. DOI: 10.1190/1.9781560803126.ch6. URL: <http://library.seg.org/doi/abs/10.1190/1.9781560803126.ch6> (cit. on pp. 5, 6).
- Johnston, D. H. (Jan. 2013b). *Practical Applications of Time-lapse Seismic Data*. Society of Exploration Geophysicists. ISBN: 9781560803072. DOI: 10.1190/1.9781560803126. URL: <http://library.seg.org/doi/book/10.1190/1.9781560803126> (cit. on pp. 5, 6).

- Jumper, J., K. Tunyasuvunakool, P. Kohli, D. Hassabis, and A. Team (Apr. 8, 2020). *Computational predictions of protein structures associated with COVID-19*. Tech. rep. URL: <https://deepmind.com/research/open-source/computational-predictions-of-protein-structures-associated-with-COVID-19> (cit. on p. 42).
- Kadurin, A., S. Nikolenko, K. Khrabrov, A. Aliper, and A. Zhavoronkov (Sept. 2017). “druGAN: An Advanced Generative Adversarial Autoencoder Model for de Novo Generation of New Molecules with Desired Molecular Properties in Silico”. en. In: *Mol. Pharm.* 14.9, pp. 3098–3104. ISSN: 1543-8384, 1543-8392. DOI: 10.1021/acs.molpharmaceut.7b00346. URL: <http://dx.doi.org/10.1021/acs.molpharmaceut.7b00346> (cit. on p. 9).
- Karpatne, A., W. Watkins, J. Read, and V. Kumar (2017). “Physics-guided neural networks (pgnn): An application in lake temperature modeling”. In: *arXiv preprint arXiv:1710.11431* (cit. on p. 97).
- Karra, S., D. O’Malley, J. D. Hyman, H. S. Viswanathan, and G. Srinivasan (Mar. 2018). “Modeling flow and transport in fracture networks using graphs”. en. In: *Phys Rev E* 97.3-1, p. 033304. ISSN: 2470-0053, 2470-0045. DOI: 10.1103/PhysRevE.97.033304. URL: <http://dx.doi.org/10.1103/PhysRevE.97.033304> (cit. on p. 26).
- Kelley, H. J. (1960). “Gradient theory of optimal flight paths”. In: *Ars Journal* 30.10, pp. 947–954 (cit. on p. 11).
- Kendall, A., Y. Gal, and R. Cipolla (2018). “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491 (cit. on p. 42).
- Al-Khawari, H., R. P. Athyal, O. Al-Saeed, P. N. Sada, S. Al-Muthairi, and A. Al-Awadhi (2010). “Inter-and intraobserver variation between radiologists in the detection of abnormal parenchymal lung changes on high-resolution computed tomography”. In: *Annals of Saudi medicine* 30.2, pp. 129–133 (cit. on p. 55).
- Khoshnevis, N. and R. Taborda (2018). “Prioritizing ground-motion validation metrics using semisupervised and supervised learning”. In: *Bull. Seismol. Soc. Am.* URL: <https://pubs.geoscienceworld.org/ssa/bssa/article-abstract/108/4/2248/536309> (cit. on p. 26).
- Kim, B., H. Kim, K. Kim, S. Kim, and J. Kim (2019). “Learning not to learn: Training deep neural networks with biased data”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9012–9020 (cit. on p. 39).
- Kingma, D. P. and J. Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv. eprint: arXiv:1412.6980* (cit. on pp. 30, 58, 81, 123).
- Kingma, D. P. and M. Welling (2013). “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (cit. on pp. 94, 100, 122).
- Kingma, D. P., T. Salimans, and M. Welling (2015). “Variational dropout and the local reparameterization trick”. In: *Advances in Neural Information Processing Systems*, pp. 2575–2583 (cit. on pp. 116, 121).
- Klein, S., M. Staring, K. Murphy, M. A. Viergever, and J. P. Pluim (2009). “Elastix: a toolbox for intensity-based medical image registration”. In: *IEEE transactions on medical imaging* 29.1, pp. 196–205 (cit. on p. 118).

- Koch, G., R. Zemel, and R. Salakhutdinov (2015). “Siamese neural networks for one-shot image recognition”. In: *ICML deep learning workshop*. Vol. 2. Lille (cit. on p. 39).
- Kolmogorov, A. N. (1939). “Sur l’interpolation et extrapolation des suites stationnaires”. In: *CR Acad Sci* 208, pp. 2043–2045 (cit. on p. 10).
- Kong, Q., D. T. Trugman, Z. E. Ross, M. J. Bianco, B. J. Meade, and P. Gerstoft (2019). “Machine learning in seismology: Turning data into insights”. In: *Seismological Research Letters* 90.1, pp. 3–14 (cit. on p. 9).
- Kratzert, F., D. Klotz, G. Shalev, G. Klambauer, S. Hochreiter, and G. Nearing (2019). “Benchmarking a catchment-aware Long Short-Term Memory Network (LSTM) for large-scale hydrological modeling”. In: *arXiv preprint arXiv:1907.08456* (cit. on p. 37).
- Krige, D. G. (1951). “A statistical approach to some mine valuation and allied problems on the Witwatersrand”. English. PhD thesis. Johannesburg (cit. on pp. 10, 11).
- Krischer, L. and A. Fichtner (2017). “Generating seismograms with deep neural networks”. In: *Agufm* 2017, S41d–03 (cit. on p. 37).
- Krishnan, R. G., U. Shalit, and D. Sontag (2015). “Deep kalman filters”. In: *arXiv preprint arXiv:1511.05121* (cit. on p. 43).
- Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012a). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems* 25. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1097–1105. URL: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf> (cit. on p. 61).
- Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012b). “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*, pp. 1097–1105 (cit. on pp. 26, 141).
- Krumbein, W. C. and M. F. Dacey (1969). “Markov chains and embedded Markov chains in geology”. In: *Journal of the International Association for Mathematical Geology* 1.1, pp. 79–96 (cit. on p. 11).
- Kuehn, N. M., C. Riggelsen, et al. (2011). “Modeling the joint probability of earthquake, site, and ground-motion parameters using Bayesian networks”. In: *Bulletin of the*. URL: <https://pubs.geoscienceworld.org/ssa/bssa/article-abstract/101/1/235/349494> (cit. on p. 26).
- Kuzma, H. A. (2003). “A support vector machine for avo interpretation”. In: *SEG Technical Program Expanded Abstracts 2003*. Society of Exploration Geophysicists, pp. 181–184 (cit. on p. 13).
- Kvalsvik, J. and Contributors (2019). *SegyIO*. Version 1.8.6. URL: <https://github.com/equinor/segyio/> (cit. on p. 42).
- Landrø, M. (May 2001). “Discrimination between pressure and fluid saturation changes from time-lapse seismic data”. In: *Geophysics* 66.3, pp. 836–844. ISSN: 0016-8033. DOI: 10.1190/1.1444973. URL: <http://library.seg.org/doi/abs/10.1190/1.1444973> (cit. on p. 7).
- Lary, D. J., A. H. Alavi, A. H. Gandomi, and A. L. Walker (Jan. 2016). “Machine learning in geosciences and remote sensing”. In: *Geoscience Frontiers* 7.1, pp. 3–10. ISSN: 1674-

9871. DOI: 10.1016/j.gsf.2015.07.003. URL: <http://www.sciencedirect.com/science/article/pii/S1674987115000821> (cit. on p. 9).
- Lecun, Y. (1989). “Generalization and network design strategies”. In: *Connectionism in perspective*. Ed. by R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels. Accessed on Mon, November 20, 2017. Elsevier. URL: <http://yann.lecun.com/exdb/publis/pdf/lecun-89.pdf> (cit. on p. 61).
- LeCun, Y., Y. Bengio, and G. Hinton (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444 (cit. on p. 11).
- LeCun, Y., P. Haffner, L. Bottou, and Y. Bengio (1999). “Object recognition with gradient-based learning”. In: *Shape, contour and grouping in computer vision*. Springer, pp. 319–345 (cit. on p. 73).
- Legendre, A. M. (1805). *Nouvelles méthodes pour la détermination des orbites des comètes*. F. Didot (cit. on p. 10).
- Li, G., Y. Qiao, Y. Zheng, Y. Li, and W. Wu (2019). “Semi-supervised learning based on generative adversarial network and its applied to lithology recognition”. In: *IEEE Access* 7, pp. 67428–67437 (cit. on p. 70).
- Li, J. and J. Castagna (Jan. 2004). “Support Vector Machine (SVM) pattern recognition to AVO classification”. In: *Geophys. Res. Lett.* 31.2, p. 948. ISSN: 0094-8276. DOI: 10.1029/2003gl018299. URL: <http://doi.wiley.com/10.1029/2003GL018299> (cit. on p. 23).
- Lin, M., Q. Chen, and S. Yan (2013). “Network in network”. In: *arXiv preprint arXiv:1312.4400* (cit. on p. 33).
- Lin, T.-Y., P. Goyal, R. Girshick, K. He, and P. Dollár (2017). “Focal loss for dense object detection”. In: *arXiv preprint arXiv:1708.02002* (cit. on p. 61).
- Lindsay, B. G. (1995). “Mixture models: theory, geometry and applications”. In: *NSF-CBMS regional conference series in probability and statistics*. JSTOR, pp. i–163 (cit. on p. 50).
- Liner, C. L. (May 2002). “Phase, phase, phase”. In: *The Leading Edge* 21.5, pp. 456–457. DOI: 10.1190/1.1885500. URL: <https://doi.org/10.1190/1.1885500> (cit. on p. 73).
- Linnainmaa, S. (1970). “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. In: *Master’s Thesis (in Finnish), Univ. Helsinki*, pp. 6–7 (cit. on p. 11).
- Liu, M., W. Li, M. Jervis, and P. Nivlet (2019a). “3D seismic facies classification using convolutional neural network and semi-supervised generative adversarial network”. In: *SEG Technical Program Expanded Abstracts 2019*. Society of Exploration Geophysicists, pp. 4995–4999 (cit. on p. 70).
- Liu, X., P. He, W. Chen, and J. Gao (2019b). “Multi-task deep neural networks for natural language understanding”. In: *arXiv preprint arXiv:1901.11504* (cit. on p. 56).
- Liu, Y., Z. Chen, L. Wang, Y. Zhang, Z. Liu, and Y. Shuai (Mar. 2015). “Quantitative seismic interpretations to detect biogenic gas accumulations: a case study from Qaidam Basin, China”. In: *Bull. Can. Petrol. Geol.* 63.1, pp. 108–121. ISSN: 0007-4802. DOI: 10.2113/gscpgbull.63.1.108. URL: <https://pubs.geoscienceworld.org/cspg/bcp/article-abstract/63/1/108/455952> (cit. on p. 23).

- Londoño, J. M. and H. Kumagai (2018). “4D seismic tomography of Nevado del Ruiz Volcano, Colombia, 2000–2016”. In: *Journal of Volcanology and Geothermal Research* 358, pp. 105–123 (cit. on p. 5).
- Long, J., E. Shelhamer, and T. Darrell (2015). “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440 (cit. on p. 61).
- Lu, P., M. Morris, S. Brazell, C. Comiskey, and Y. Xiao (2018). “Using generative adversarial networks to improve deep-learning fault interpretation networks”. In: *The Leading Edge* 37.8, pp. 578–583 (cit. on p. 37).
- Lumley, D. E. (1995). *Seismic time-lapse monitoring of subsurface fluid flow*. 91. Stanford University (cit. on p. 6).
- Lumley, D. E. (Jan. 2001). “Time-lapse seismic reservoir monitoring”. In: *Geophysics* 66.1, pp. 50–53. ISSN: 0016-8033. DOI: 10.1190/1.1444921. URL: <http://library.seg.org/doi/abs/10.1190/1.1444921> (cit. on p. 6).
- Lundberg, S. M., B. Nair, M. S. Vavilala, M. Horibe, M. J. Eisses, T. Adams, D. E. Liston, D. K.-W. Low, S.-F. Newman, J. Kim, et al. (2018). “Explainable machine-learning predictions for the prevention of hypoxaemia during surgery”. In: *Nature biomedical engineering* 2.10, pp. 749–760 (cit. on p. 43).
- Luo, S. and D. Hale (Aug. 2014). “Least-squares migration in the presence of velocity errors”. In: *SEG Technical Program Expanded Abstracts 2014*. 5. Society of Exploration Geophysicists, pp. 3980–3984. DOI: 10.1190/segam2014-1367.1. URL: <http://library.seg.org/doi/abs/10.1190/segam2014-1367.1> (cit. on p. 112).
- Luo, X., W. Zhou, W. Wang, Y. Zhu, and J. Deng (2017). “Attention-based relation extraction with bidirectional gated recurrent unit and highway network in the analysis of geological data”. In: *IEEE Access* 6, pp. 5705–5715 (cit. on p. 37).
- Ma, J., Z. Jiang, Q. Tian, and G. D. Couples (2012). “Classification of Digital Rocks by Machine Learning”. In: *ECMOR XIII-13th European*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=62262> (cit. on p. 23).
- MacBeth, C., M.-D. Mangriotis, and H. Amini (2019). “Post-stack 4D seismic time-shifts: Interpretation and evaluation”. In: *Geophysical Prospecting* 67.1, pp. 3–31 (cit. on p. 6, 115).
- Maggi, A., V. Ferrazzini, C. Hibert, F. Beauducel, P. Boissier, and A. Amemoutou (May 2017). “Implementation of a Multistation Approach for Automated Event Classification at Piton de la Fournaise Volcano”. In: *Seismol. Res. Lett.* 88.3, pp. 878–891. ISSN: 0895-0695. DOI: 10.1785/0220160189. URL: <https://pubs.geoscienceworld.org/ssa/srl/article-abstract/88/3/878/284054> (cit. on p. 25).
- Mahajan, D., R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten (2018). “Exploring the limits of weakly supervised pretraining”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 181–196 (cit. on p. 59).
- Malfante, M., M. D. Mura, J. Metaxian, J. I. Mars, O. Macedo, and A. Inza (Mar. 2018). “Machine Learning for Volcano-Seismic Signals: Challenges and Perspectives”. In: *IEEE Signal Process. Mag.* 35.2, pp. 20–30. ISSN: 1053-5888. DOI: 10.1109/msp.

- 2017.2779166. URL: <http://dx.doi.org/10.1109/MSP.2017.2779166> (cit. on p. 23).
- Mardan, A., A. Javaherian, et al. (2017). “Channel Characterization Using Support Vector Machine”. In: *79th EAGE Conference*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=89283> (cit. on p. 23).
- Marjanović, M., M. Kovačević, B. Bajat, and V. Voženilek (Nov. 2011). “Landslide susceptibility assessment using SVM machine learning algorithm”. In: *Eng. Geol.* 123.3, pp. 225–234. ISSN: 0013-7952. DOI: 10.1016/j.enggeo.2011.09.006. URL: <http://www.sciencedirect.com/science/article/pii/S0013795211002195> (cit. on p. 23).
- Markov, A. A. (1906). “Rasprostranenie zakona bol’shih chisel na velichiny, zavisyaschie drug ot druga”. In: *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete* 15.135-156, p. 18 (cit. on pp. 10, 289).
- Markov, A. A. (1971). “Extension of the limit theorems of probability theory to a sum of variables connected in a chain”. In: *Dynamic probabilistic systems* 1. Reprint in English of (Markov, 1906), pp. 552–577 (cit. on p. 10).
- Martinelli, G., J. Eidsvik, R. Sinding-Larsen, et al. (2013). “Building Bayesian networks from basin-modelling scenarios for improved geological decision making”. In: *Petroleum*. URL: <http://pg.lyellcollection.org/content/early/2013/06/24/petgeo2012-057.abstract> (cit. on p. 26).
- Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: <http://tensorflow.org/> (cit. on pp. xiii, 11, 26, 62, 76, 122).
- Masotti, M., R. Campanini, L. Mazzacurati, S. Falsaperla, H. Langer, and S. Spampinato (Apr. 2008). “TREMOrEC: A software utility for automatic classification of volcanic tremor”. In: *Geochem. Geophys. Geosyst.* 9.4. ISSN: 1525-2027. DOI: 10.1029/2007gc001860. URL: <http://doi.wiley.com/10.1029/2007GC001860> (cit. on p. 23).
- Masotti, M., S. Falsaperla, H. Langer, S. Spampinato, and R. Campanini (Oct. 2006). “Application of Support Vector Machine to the classification of volcanic tremor at Etna, Italy”. In: *Geophys. Res. Lett.* 33.20, p. 113. ISSN: 0094-8276. DOI: 10.1029/2006gl027441. URL: <http://doi.wiley.com/10.1029/2006GL027441> (cit. on p. 23).
- Matalas, N. C. (1967). “Mathematical assessment of synthetic hydrology”. In: *Water Resources Research* 3.4, pp. 937–945 (cit. on p. 12).
- Matheron, G. (1963). “Principles of geostatistics”. In: *Economic geology* 58.8, pp. 1246–1266 (cit. on p. 10).

- Matheron, G. et al. (1981). “Splines and kriging; their formal equivalence”. In: (cit. on p. 12).
- Mavko, G., T. Mukerji, and J. Dvorkin (2003). “The Rock Physics Handbook”. In: (cit. on p. 73).
- McCormack, M. (Jan. 1991). “Neural computing in geophysics”. In: *Lead. Edge* 10.1, pp. 11–15. ISSN: 1070-485x. DOI: 10.1190/1.1436771. URL: <https://doi.org/10.1190/1.1436771> (cit. on p. 13).
- McErlean, A., D. M. Panicek, E. C. Zabor, C. S. Moskowitz, R. Bitar, R. J. Motzer, H. Hricak, and M. S. Ginsberg (2013). “Intra- and Interobserver Variability in CT Measurements in Oncology”. In: *Radiology* 269.2, pp. 451–459. DOI: 10.1148/radiol.13122665. URL: <https://doi.org/10.1148/radiol.13122665> (cit. on p. 55).
- McKinney, W. et al. (2010). “Data structures for statistical computing in python”. In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX, pp. 51–56 (cit. on p. xiv).
- Mehri, S., K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. C. Courville, and Y. Bengio (2016). “SampleRNN: An Unconditional End-to-End Neural Audio Generation Model”. In: *CoRR* abs/1612.07837. arXiv: 1612.07837. URL: <http://arxiv.org/abs/1612.07837> (cit. on p. 72).
- Meister, S., J. Hur, and S. Roth (2018). “UnFlow: Unsupervised learning of optical flow with a bidirectional census loss”. In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on pp. 111, 119).
- Mignan, A. and M. Broccardo (2019a). “A Deeper Look into ‘Deep Learning of After-shock Patterns Following Large Earthquakes’: Illustrating First Principles in Neural Network Physical Interpretability”. In: *International Work-Conference on Artificial Neural Networks*. Springer, pp. 3–14 (cit. on p. 41).
- Mignan, A. and M. Broccardo (2019b). “One neuron versus deep learning in aftershock prediction”. In: *Nature* 574.7776, E1–e3 (cit. on pp. 41, 42).
- Miller, M. I., A. Trouvé, and L. Younes (2015). “Hamiltonian systems and optimal control in computational anatomy: 100 years since D’Arcy Thompson”. In: *Annual review of biomedical engineering* 17, pp. 447–509 (cit. on p. 110).
- Mitchell, T. M. et al. (1997). *Machine learning* (cit. on p. 10).
- Miyauchi, M., M. Seki, A. Watanabe, and A. Miyauchi (1993). “Interpretation of optical flow through complex neural network”. In: *New Trends in Neural Computation*. Springer Berlin Heidelberg, pp. 645–650. DOI: 10.1007/3-540-56798-4_215. URL: https://doi.org/10.1007/3-540-56798-4_215 (cit. on p. 73).
- Mjolsness, E. and D. DeCoste (Sept. 2001). “Machine learning for science: state of the art and future prospects”. en. In: *Science* 293.5537, pp. 2051–2055. ISSN: 0036-8075. DOI: 10.1126/science.293.5537.2051. URL: <http://dx.doi.org/10.1126/science.293.5537.2051> (cit. on p. 17).
- Modarres, M. H., R. Aversa, S. Cozzini, R. Ciancio, A. Leto, and G. P. Brandino (2017). “Neural network for nanoscience scanning electron microscope image recognition”. In: *Scientific reports* 7.1, p. 13282 (cit. on p. 50).
- Mosser, L., R. Oliveira, and M. Steventon (2019). “Probabilistic Seismic Interpretation Using Bayesian Neural Networks”. In: *81st EAGE Conference and Exhibition 2019*.

- Vol. 2019. 1. European Association of Geoscientists & Engineers, pp. 1–5 (cit. on p. 43).
- Mosser, L., O. Dubrule, and M. J. Blunt (Oct. 2017). “Reconstruction of three-dimensional porous media using generative adversarial neural networks”. en. In: *Phys Rev E* 96.4-1, p. 043309. ISSN: 2470-0053, 2470-0045. DOI: 10.1103/PhysRevE.96.043309. URL: <http://dx.doi.org/10.1103/PhysRevE.96.043309> (cit. on p. 37).
- Mosser, L., O. Dubrule, and M. J. Blunt (Feb. 2018a). “Conditioning of three-dimensional generative adversarial networks for pore and reservoir-scale models”. In: arXiv: 1802.05622 [stat.ML]. URL: <http://arxiv.org/abs/1802.05622> (cit. on p. 37).
- Mosser, L., O. Dubrule, and M. J. Blunt (June 2018b). “Stochastic seismic waveform inversion using generative adversarial networks as a geological prior”. In: *Mathematical Geosciences* 52.1, pp. 53–79. arXiv: 1806.03720 [physics.geo-ph]. URL: <http://arxiv.org/abs/1806.03720> (cit. on p. 37).
- Mosser, L., W. Kimman, J. S. Dransch, S. Purves, A. De la Fuente Briceño, and G. Ganssle (June 2018c). “Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks”. In: *80th EAGE Conference and Exhibition 2018. Published, Appendix C*. EAGE. DOI: 10.3997/2214-4609.201800734. URL: <https://doi.org/10.3997/2214-4609.201800734> (cit. on pp. 37, 120, 179).
- Nash, J. (1951). “Non-cooperative games”. In: *Annals of mathematics*, pp. 286–295 (cit. on p. 36).
- Neal, R. M. (1996). *Bayesian learning for neural networks*. Springer Science & Business Media (cit. on p. 17).
- Newendorp, P. D. (1976). “Decision analysis for petroleum exploration”. In: (cit. on p. 12).
- Ochoa, L. H., L. F. Niño, and C. A. Vargas (Jan. 2018). “Fast magnitude determination using a single seismological station record implementing machine learning techniques”. In: *Geodesy and Geodynamics* 9.1, pp. 34–41. ISSN: 1674-9847. DOI: 10.1016/j.geog.2017.03.010. URL: <http://www.sciencedirect.com/science/article/pii/S1674984717300058> (cit. on p. 23).
- Ohrnberger, M. (2001). [No title]. https://www.researchgate.net/profile/Matthias_Ohrnberger/publication/252958874_Continuous_Automatic_Classification_of_Seismic_Signals_of_Volcanic-Origin_at_Mt_Merapi_Java_Indonesia/links/573ffe3e08aea45ee84504a4/Continuous-Automatic-Classification-of-Seismic-Signals-of-Volcanic-Origin-at-Mt-Merapi-Java-Indonesia.pdf. Accessed: 2018-12-17. URL: https://www.researchgate.net/profile/Matthias_Ohrnberger/publication/252958874_Continuous_Automatic_Classification_of_Seismic_Signals_of_Volcanic-Origin_at_Mt_Merapi_Java_Indonesia/links/573ffe3e08aea45ee84504a4/Continuous-Automatic-Classification-of-Seismic-Signals-of-Volcanic-Origin-at-Mt-Merapi-Java-Indonesia.pdf (cit. on p. 26).
- Oliveira, L. de, M. Paganini, and B. Nachman (2017). “Learning particle physics by example: location-aware generative adversarial networks for physics synthesis”. In: *Computing and Software for Big Science* (cit. on p. 97).

- Oord, A. van den, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu (2016). “WaveNet: A Generative Model for Raw Audio”. In: *CoRR* abs/1609.03499. arXiv: 1609.03499. URL: <http://arxiv.org/abs/1609.03499> (cit. on p. 72).
- Oord, A. van den, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. van den Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, and D. Hassabis (2017). “Parallel WaveNet: Fast High-Fidelity Speech Synthesis”. In: *CoRR* abs/1711.10433. arXiv: 1711.10433. URL: <http://arxiv.org/abs/1711.10433> (cit. on pp. 72, 73).
- Paganini, M., L. de Oliveira, and B. Nachman (2017). “CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks”. In: arXiv: 1705.02355 [hep-ex] (cit. on p. 72).
- Paganini, M., L. de Oliveira, and B. Nachman (2018). “CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks”. In: *Physical Review D* 97.1, p. 014021 (cit. on p. 36).
- Panakkat, A. and H. Adeli (2007). “Neural network models for earthquake magnitude prediction using multiple seismicity indicators”. In: *International journal of neural systems* 17.01, pp. 13–33 (cit. on p. 39).
- Pasolli, E., F. Melgani, and M. Donelli (July 2009). “Automatic Analysis of GPR Images: A Pattern-Recognition Approach”. In: *IEEE Trans. Geosci. Remote Sens.* 47.7, pp. 2206–2217. ISSN: 0196-2892. DOI: 10.1109/tgrs.2009.2012701. URL: <http://dx.doi.org/10.1109/TGRS.2009.2012701> (cit. on p. 23).
- Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (2017). “Automatic Differentiation in PyTorch”. In: *NIPS Autodiff Workshop* (cit. on p. 18).
- Pearl, J. (2012). “The do-calculus revisited”. In: *arXiv preprint arXiv:1210.4852* (cit. on p. 26).
- Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on pp. xiv, 11, 18, 51).
- Poulton, M., B. Sternberg, and C. Glass (Dec. 1992). “Location of subsurface targets in geophysical data using neural networks”. In: *Geophysics* 57.12, pp. 1534–1544. ISSN: 0016-8033. DOI: 10.1190/1.1443221. URL: <https://doi.org/10.1190/1.1443221> (cit. on p. 15).
- Prenger, R., R. Valle, and B. Catanzaro (2018). “WaveGlow: A Flow-based Generative Network for Speech Synthesis”. In: *CoRR* abs/1811.00002. arXiv: 1811.00002. URL: <http://arxiv.org/abs/1811.00002> (cit. on p. 72).
- Preston, F. W. and J. Henderson (1964). *Fourier series characterization of cyclic sediments for stratigraphic correlation*. Kansas Geological Survey (cit. on p. 11).
- Purves, S., B. Alaei, and E. Larsen (2018). “Bootstrapping Machine-Learning Based Seismic Fault Interpretation”. In: *ACE 2018 Annual Convention &*. URL: <http://www>.

- searchanddiscovery.com/abstracts/html/2018/ace2018/abstracts/2856016.html (cit. on p. 35).
- Purves, S., B. Alaei, and D. Lolis (2019). “Towards Subsurface ML Metrics”. In: *81st EAGE Conference and Exhibition 2019 Workshop Programme* (cit. on p. 56).
- Purves, S. (Oct. 2014). “Phase and the Hilbert transform”. In: *The Leading Edge* 33.10, pp. 1164–1166. DOI: 10.1190/tle33101164.1. URL: <https://doi.org/10.1190/tle33101164.1> (cit. on p. 73).
- Ramcharan, A., K. Baranowski, P. McCloskey, B. Ahmed, J. Legg, and D. P. Hughes (2017). “Deep learning for image-based cassava disease detection”. In: *Frontiers in plant science* 8, p. 1852 (cit. on p. 42).
- Ramcharan, A., P. McCloskey, K. Baranowski, N. Mbilinyi, L. Mrisho, M. Ndalawha, J. Legg, and D. P. Hughes (2019). “A mobile-based deep learning model for cassava disease diagnosis”. In: *Frontiers in plant science* 10, p. 272 (cit. on p. 42).
- Ranjan, A. and M. J. Black (2017). “Optical flow estimation using a spatial pyramid network”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4161–4170 (cit. on pp. 110, 111, 119).
- Rasmussen, C. E. (2003). “Gaussian processes in machine learning”. In: *Summer School on Machine Learning*. Springer, pp. 63–71 (cit. on p. 10).
- Real, E., A. Aggarwal, Y. Huang, and Q. V. Le (2019). “Regularized evolution for image classifier architecture search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 4780–4789 (cit. on pp. 57, 141).
- Recht, B., R. Roelofs, L. Schmidt, and V. Shankar (2019). “Do imagenet classifiers generalize to imagenet?” In: *arXiv preprint arXiv:1902.10811* (cit. on p. 24).
- Reddy, R. and G. Bonham-Carter (1991). “A Decision-Tree Approach to Mineral Potential Mapping in Snow Lake Area, Manitoba”. In: *Canadian Journal of Remote Sensing* 17.2, pp. 191–200. DOI: 10.1080/07038992.1991.10855292. eprint: <https://doi.org/10.1080/07038992.1991.10855292>. URL: <https://doi.org/10.1080/07038992.1991.10855292> (cit. on p. 12).
- Richardson, A. (Jan. 2018). “Seismic Full-Waveform Inversion Using Deep Learning Tools and Techniques”. In: arXiv: 1801.07232 [physics.geo-ph]. URL: <http://arxiv.org/abs/1801.07232> (cit. on p. 37).
- Rickett, J. E. and D. Lumley (July 2001). “Cross-equalization data processing for time-lapse seismic reservoir monitoring: A case study from the Gulf of Mexico”. In: *Geophysics* 66.4, pp. 1015–1025. ISSN: 0016-8033. DOI: 10.1190/1.1487049. URL: <http://library.seg.org/doi/abs/10.1190/1.1487049> (cit. on p. 7).
- Rickett, J., L. Duranti, T. Hudson, B. Regel, and N. Hodgson (2007a). “4D time strain and the seismic signature of geomechanical compaction at Genesis”. In: *The Leading Edge* 26.5, pp. 644–647 (cit. on pp. 109, 117).
- Rickett, J., L. Duranti, S. Ramon, U. T. Hudson, B. Regel, and N. Orleans (May 2007b). “4D time strain and the seismic signature of geomechanical compaction at Genesis”. In: *Lead. Edge* 26.5, pp. 644–647. ISSN: 1070-485x. DOI: 10.1190/1.2737103. URL: <http://library.seg.org/doi/10.1190/1.2737103> (cit. on p. 7).

- Roden, R. and C. W. Chen (2017). “Interpretation of DHI characteristics with machine learning”. In: *First Break*. ISSN: 0263-5046. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=88069> (cit. on p. 35).
- Roden, R. and H. Sepúlveda (July 1999). “The significance of phase to the interpreter: Practical guidelines for phase analysis”. In: *The Leading Edge* 18.7, pp. 774–777. DOI: 10.1190/1.1438375. URL: <https://doi.org/10.1190/1.1438375> (cit. on p. 73).
- Roeth, G. and A. Tarantola (1994). “Neural networks and inversion of seismic data”. In: *Journal of Geophysical Research: Solid Earth* 99.B4, pp. 6753–6768. DOI: 10.1029/93JB01563. URL: <https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/93JB01563> (cit. on p. 97).
- Rolnick, D., P. L. Donti, L. H. Kaack, K. Kochanski, A. Lacoste, K. Sankaran, A. S. Ross, N. Milojevic-Dupont, N. Jaques, A. Waldman-Brown, et al. (2019). “Tackling climate change with machine learning”. In: *arXiv preprint arXiv:1906.05433* (cit. on p. 10).
- Ronneberger, O., P. Fischer, and T. Brox (2015a). “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241 (cit. on pp. 35, 69).
- Ronneberger, O., P. Fischer, and T. Brox (2015b). “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241 (cit. on pp. 50, 111, 121).
- Rosenblatt, F. (1958). “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6, p. 386 (cit. on p. 13).
- Ross, Z. E., M. A. Meier, and E. Hauksson (2018a). “P-wave arrival picking and first-motion polarity determination with deep learning”. In: *J. Geophys. Res.* ISSN: 0148-0227. URL: <https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2017JB015251> (cit. on p. 39).
- Ross, Z. E., M.-A. Meier, E. Hauksson, and T. H. Heaton (May 2018b). “Generalized Seismic Phase Detection with Deep Learning”. In: *Bulletin of the Seismological Society of America* 108.5a, pp. 2894–2901. arXiv: 1805.01075 [physics.geo-ph]. URL: <http://arxiv.org/abs/1805.01075> (cit. on p. 39).
- Røste, T., A. Stovas, and M. Landrø (2006). “Estimation of layer thickness and velocity changes using 4D prestack seismic data”. In: *Geophysics* 71.6, S219–s234 (cit. on p. 6).
- Röth, G. and A. Tarantola (1994). “Neural networks and inversion of seismic data”. In: *J. Geophys. Res.* 99.B4, p. 6753. ISSN: 0148-0227. DOI: 10.1029/93jb01563. URL: <http://doi.wiley.com/10.1029/93JB01563> (cit. on p. 14).
- Rouet-Leduc, B., C. Hulbert, D. C. Bolton, et al. (2018). “Estimating fault friction from seismic signals in the laboratory”. In: *Geophysical*. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/2017GL076708> (cit. on p. 25).
- Rouet-Leduc, B., C. Hulbert, N. Lubbers, et al. (2017). “Machine learning predicts laboratory earthquakes”. In: *Geophysical*. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/2017GL074677> (cit. on p. 25).

- Ruder, S. (2016). “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (cit. on p. 62).
- Rumelhart, D., G. Hinton, and R. Williams (1988a). “Learning Internal Representations by Error Propagation”. In: *Readings in Cognitive Science*. Elsevier, pp. 399–421. DOI: 10.1016/b978-1-4832-1446-7.50035-2. URL: <https://doi.org/10.1016/b978-1-4832-1446-7.50035-2> (cit. on p. 61).
- Rumelhart, D. E., G. E. Hinton, R. J. Williams, et al. (1988b). “Learning representations by back-propagating errors”. In: *Cognitive modeling* 5.3, p. 1 (cit. on p. 11).
- Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y (cit. on p. 26).
- Russell, S. J. and P. Norvig (2010). *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education. ISBN: 978-0-13-207148-2. URL: http://vig.pearsoned.com/store/product/1,1207,store-12521%5C_isbn-0136042597,00.html (cit. on pp. 11, 13).
- Rutherford Ildstad, C. and P. Bormann (2017). “MalenoV”. Machine learning of Voxels (cit. on p. 61).
- Sakoe, H. and S. Chiba (Feb. 1978). “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1, pp. 43–49. ISSN: 0096-3518. DOI: 10.1109/TASSP.1978.1163055. URL: <http://ieeexplore.ieee.org/document/1163055/> (cit. on pp. 112, 114).
- Samuel, A. L. (1959). “Some studies in machine learning using the game of checkers”. In: *IBM Journal of research and development* 3.3, pp. 210–229 (cit. on p. 10).
- Saporetti, C. M., L. G. da Fonseca, E. Pereira, and L. C. de Oliveira (Aug. 2018). “Machine learning approaches for petrographic classification of carbonate-siliciclastic rocks using well logs and textural information”. In: *J. Appl. Geophys.* 155, pp. 217–225. ISSN: 0926-9851. DOI: 10.1016/j.jappgeo.2018.06.012. URL: <http://www.sciencedirect.com/science/article/pii/S092698511630667X> (cit. on pp. 23, 25, 26).
- Sarroff, A. M. (May 2018). *Complex Neural Networks for Audio*. Tech. rep. TR2018-859. Hanover, NH: Dartmouth College, Computer Science. URL: <http://www.cs.dartmouth.edu/~trdata/reports/TR2018-859.pdf> (cit. on p. 73).
- Sarroff, A. M., V. Shepardson, and M. A. Casey (2015). “Learning Representations Using Complex-Valued Nets”. In: *CoRR* abs/1511.06351. arXiv: 1511.06351. URL: <http://arxiv.org/abs/1511.06351> (cit. on p. 76).
- Scarpiniti, M., D. Vigliano, R. Parisi, and A. Uncini (June 2008). “Generalized splitting functions for blind separation of complex signals”. In: *Neurocomputing* 71.10-12, pp. 2245–2270. DOI: 10.1016/j.neucom.2007.07.037. URL: <https://doi.org/10.1016/j.neucom.2007.07.037> (cit. on p. 73).
- Schütt, K. T., F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko (Jan. 2017a). “Quantum-chemical insights from deep tensor neural networks”. en. In: *Nat. Commun.* 8, p. 13890. ISSN: 2041-1723. DOI: 10.1038/ncomms13890. URL: <http://dx.doi.org/10.1038/ncomms13890> (cit. on p. 9).

- Schütt, K. T., F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko (2017b). “Quantum-chemical insights from deep tensor neural networks”. In: *Nature communications* 8, p. 13890 (cit. on p. 97).
- Schwarzacher, W. (1972). “The semi-Markov process as a general sedimentation model”. In: *Mathematical Models of Sedimentary Processes*. Springer, pp. 247–268 (cit. on p. 12).
- Sen, S., S. Kainkaryam, C. Ong, and A. Sharma (2020). “SaltNet: A production-scale deep learning pipeline for automated salt model building”. In: *The Leading Edge* 39.3, pp. 195–203 (cit. on p. 36).
- Serra, J. and L. Vincent (1992). “An overview of morphological filtering”. In: *Circuits, Systems and Signal Processing* 11.1, pp. 47–108 (cit. on p. 51).
- Shah, R. and L. Innig (May 7, 2019). *Aftershock Issues*. Version 9632948. URL: https://github.com/rajshah4/aftershocks%5C_issues (cit. on pp. 41, 42).
- Shashidhara, B. M., M. Scott, and A. Marburg (2020). “Instance Segmentation of Benthic Scale Worms at a Hydrothermal Site”. In: *The IEEE Winter Conference on Applications of Computer Vision*, pp. 1314–1323 (cit. on p. 36).
- Shen, C. (2018). “A transdisciplinary review of deep learning research and its relevance for water resources scientists”. In: *Water Resources Research* 54.11, pp. 8558–8593 (cit. on p. 9).
- Shen, D., G. Wu, and H.-I. Suk (June 2017). “Deep Learning in Medical Image Analysis”. en. In: *Annu. Rev. Biomed. Eng.* 19, pp. 221–248. ISSN: 1523-9829, 1545-4274. DOI: 10.1146/annurev-bioeng-071516-044442. URL: <http://dx.doi.org/10.1146/annurev-bioeng-071516-044442> (cit. on p. 9).
- Simonyan, K. and A. Zisserman (2014a). “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (cit. on pp. 33, 141).
- Simonyan, K. and A. Zisserman (2014b). “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (cit. on p. 62).
- Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15, pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html> (cit. on p. 61).
- Stein, F. (2004). “Efficient computation of optical flow using the census transform”. In: *Joint Pattern Recognition Symposium*. Springer, pp. 79–86 (cit. on pp. 111, 119).
- Su, J., D. V. Vargas, and K. Sakurai (2019). “One pixel attack for fooling deep neural networks”. In: *IEEE Transactions on Evolutionary Computation* (cit. on p. 56).
- Suksmono, A. B. and A. Hirose (Mar. 2002). “Adaptive noise reduction of InSAR images based on a complex-valued MRF model and its application to phase unwrapping problem”. In: *IEEE Transactions on Geoscience and Remote Sensing* 40.3, pp. 699–709. DOI: 10.1109/tgrs.2002.1000329. URL: <https://doi.org/10.1109/tgrs.2002.1000329> (cit. on p. 73).
- Sun, A. Y., B. R. Scanlon, Z. Zhang, D. Walling, S. N. Bhanja, A. Mukherjee, and Z. Zhong (2019). “Combining Physically Based Modeling and Deep Learning for Fusing GRACE Satellite Data: Can We Learn From Mismatch?” In: *Water Resources Research* 55.2, pp. 1179–1195 (cit. on p. 36).

- Sun, D., X. Yang, M.-Y. Liu, and J. Kautz (2018). “PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8934–8943 (cit. on pp. 111, 119).
- Sutskever, I., J. Martens, G. Dahl, and G. Hinton (June 2013). “On the importance of initialization and momentum in deep learning”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: Pmlr, pp. 1139–1147. URL: <http://proceedings.mlr.press/v28/sutskever13.html> (cit. on p. 30).
- Szegedy, C., S. Ioffe, V. Vanhoucke, and A. A. Alemi (2017). “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence* (cit. on p. 141).
- Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9 (cit. on pp. 57, 141).
- Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826 (cit. on p. 141).
- Talarico, E., W. Leão, and D. Grana (2019). “Comparison of Recursive Neural Network and Markov Chain Models in Facies Inversion”. In: *Petroleum Geostatistics 2019*. Vol. 2019. 1. European Association of Geoscientists & Engineers, pp. 1–5 (cit. on p. 37).
- Tan, M., B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le (2019a). “Mnasnet: Platform-aware neural architecture search for mobile”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828 (cit. on p. 141).
- Tan, M. and Q. V. Le (2019b). “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *arXiv preprint arXiv:1905.11946* (cit. on pp. 57, 141).
- Tan, M. and Q. V. Le (2019c). “MixConv: Mixed Depthwise Convolutional Kernels”. In: *CoRR*, abs/1907.09595 (cit. on p. 141).
- Tancik, M., P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng (2020). “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”. In: *arXiv preprint arXiv:2006.10739* (cit. on p. 43).
- Taner, M. T., F. Koehler, and R. Sheriff (1979). “Complex seismic trace analysis”. In: *Geophysics* 44.6, pp. 1041–1063 (cit. on p. 79).
- Tarantola, A. (2005). *Inverse problem theory and methods for model parameter estimation*. Siam (cit. on p. 7).
- Team, T. D. (May 2016). “Theano: A Python framework for fast computation of mathematical expressions”. In: *arXiv e-prints* abs/1605.02688. URL: <http://arxiv.org/abs/1605.02688> (cit. on p. 18).
- Titos, M., A. Bueno, L. García, M. C. Benítez, and J. Ibañez (2018). “Detection and classification of continuous volcano-seismic signals with recurrent neural networks”.

- In: *IEEE Transactions on Geoscience and Remote Sensing* 57.4, pp. 1936–1948 (cit. on p. 37).
- Touvron, H., A. Vedaldi, M. Douze, and H. Jégou (2019). “Fixing the train-test resolution discrepancy”. In: *arXiv preprint arXiv:1906.06423* (cit. on p. 141).
- Trabelsi, C., O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal (2017). “Deep complex networks”. In: *arXiv preprint arXiv:1705.09792* (cit. on pp. 43, 74–76, 250).
- Turing, A. M. (Oct. 1950). “I.—Computing Machinery and Intelligence”. In: *Mind* Lix.236, pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: <http://oup.prod.sis.lan/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf>. URL: <https://doi.org/10.1093/mind/LIX.236.433> (cit. on p. 10).
- Uieda, L. (2018). “Verde: Processing and gridding spatial data using Green’s functions”. In: *Journal of Open Source Software* 3.29, p. 957. ISSN: 2475-9066. DOI: 10.21105/joss.00957 (cit. on p. 42).
- Ulyanov, D., A. Vedaldi, and V. Lempitsky (2018). “Deep image prior”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9446–9454 (cit. on p. 97).
- Valentine, A. and L. M. Kalnins (2016). “An introduction to learning algorithms and potential applications in geomorphometry and earth surface dynamics.” In: *Earth surface dynamics*. 4, pp. 445–460 (cit. on p. 9).
- Valera, M., Z. Guo, P. Kelly, S. Matz, V. A. Cantu, A. G. Percus, J. D. Hyman, G. Srinivasan, and H. S. Viswanathan (May 2017). “Machine learning for graph-based representations of three-dimensional discrete fracture networks”. In: *Computational Geosciences* 22.3, pp. 695–710. ISSN: 1573-1499. DOI: 10.1007/s10596-018-9720-1. arXiv: 1705.09866 [physics.geo-ph]. URL: <http://arxiv.org/abs/1705.09866> (cit. on p. 25).
- Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). “Attention is all you need”. In: *Advances in neural information processing systems*, pp. 5998–6008 (cit. on p. 39).
- Waldeland, A. U. and A. Solberg (2017). “Salt classification using deep learning”. In: *79th EAGE Conference and Exhibition*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=88635> (cit. on p. 34).
- Waldeland, A., A. Jensen, L. Gelius, and A. Solberg (July 2018). “Convolutional neural networks for automated seismic interpretation”. In: *Lead. Edge* 37.7, pp. 529–537. ISSN: 1070-485X. DOI: 10.1190/tle37070529.1. URL: <https://doi.org/10.1190/tle37070529.1> (cit. on p. 34).
- Waldeland, A. and A. Solberg (May 2016). “3D Attributes and Classification of Salt Bodies on Unlabelled Datasets”. In: *78th EAGE Conference and Exhibition 2016*. EAGE Publications BV. DOI: 10.3997/2214-4609.201600880. URL: <https://doi.org/10.3997/2214-4609.201600880> (cit. on pp. 58, 61, 62).
- Wang, H., J. F. Wellmann, Z. Li, X. Wang, and R. Y. Liang (Feb. 2017a). “A Segmentation Approach for Stochastic Geological Modeling Using Hidden Markov Random Fields”. In: *Math. Geosci.* 49.2, pp. 145–177. ISSN: 1874-8961, 1874-8953. DOI:

- 10.1007/s11004-016-9663-9. URL: <https://doi.org/10.1007/s11004-016-9663-9> (cit. on p. 26).
- Wang, K., J. Lomask, and F. Segovia (Aug. 2017b). “Automatic, geologic layer-constrained well-seismic tie through blocked dynamic warping”. In: *Interpretation* 5.3, Sj81–sj90. ISSN: 2324-8858. DOI: 10.1190/int-2016-0160.1. URL: <https://doi.org/10.1190/INT-2016-0160.1> (cit. on p. 26).
- Wang, L. X. and J. M. Mendel (1992). “Adaptive minimum prediction-error deconvolution and source wavelet estimation using Hopfield neural networks”. In: *Geophysics*. ISSN: 0016-8033. URL: <https://library.seg.org/doi/abs/10.1190/1.1443281> (cit. on p. 37).
- Wang, Z., H. Di, M. A. Shafiq, Y. Alaudah, and G. AlRegib (2018). “Successful leveraging of image processing and machine learning in seismic structural interpretation: A review”. In: *The Leading Edge* 37.6, pp. 451–461 (cit. on p. 9).
- Watkins, C. J. C. H. (1989). “Learning from delayed rewards”. In: (cit. on p. 11).
- Wei, S., O. Yonglin, Z. Qingcai, H. Jiaqiang, et al. (2018). “Unsupervised Machine Learning: K-means Clustering Velocity Semblance Auto-Picking”. In: *80th EAGE Conference*. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=92299> (cit. on p. 26).
- Weinzaepfel, P., J. Revaud, Z. Harchaoui, and C. Schmid (2013). “DeepFlow: Large displacement optical flow with deep matching”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1385–1392 (cit. on pp. 111, 119).
- Welly: Manage subsurface well data (2015). URL: <https://github.com/agile-geoscience/welly> (cit. on p. xiii).
- Wickman, F. (1968). “Repose period patterns of volcanoes. V. General discussion and a tentative stochastic model”. In: *Arkiv for Mineralogi och Geologi* 4.5, p. 351 (cit. on p. 12).
- Widrow, B. and M. Lehr (1990). “30 years of adaptive neural networks: perceptron Madaline, and backpropagation”. In: *Proceedings of the IEEE* 78.9, pp. 1415–1442. DOI: 10.1109/5.58323. URL: <https://doi.org/10.1109/5.58323> (cit. on p. 61).
- Williams, C. K. (1998). “Prediction with Gaussian processes: From linear regression to linear prediction and beyond”. In: *Learning in graphical models*. Springer, pp. 599–621 (cit. on p. 15).
- Williams, C. K. and C. E. Rasmussen (2006). *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA (cit. on pp. 16, 17).
- Wirgin, A. (2004). “The inverse crime”. In: *arXiv preprint math-ph/0401050* (cit. on pp. 39, 55, 115).
- Witten, I. H., E. Frank, and M. A. Hall (2005). “Practical machine learning tools and techniques”. In: *Morgan Kaufmann*, p. 578 (cit. on p. 18).
- Worrall, D. E., S. J. Garbin, D. Turmukhambetov, and G. J. Brostow (2017). “Interpretable transformations with encoder-decoder networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5726–5735 (cit. on p. 94).
- Wu, H. and B. Zhang (Apr. 2018). “A deep convolutional encoder-decoder neural network in assisting seismic horizon tracking”. In: arXiv: 1804.06814 [physics.geo-ph]. URL: <http://arxiv.org/abs/1804.06814> (cit. on p. 35).

- Xie, P., A. Zhou, and B. Chai (2019a). “The application of long short-term memory (LSTM) method on displacement prediction of multifactor-induced landslides”. In: *IEEE Access* 7, pp. 54305–54311 (cit. on p. 37).
- Xie, Q., E. Hovy, M.-T. Luong, and Q. V. Le (2019b). “Self-training with Noisy Student improves ImageNet classification”. In: *arXiv preprint arXiv:1911.04252* (cit. on pp. 28, 43).
- Xie, S., R. Girshick, P. Dollár, Z. Tu, and K. He (2017). “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500 (cit. on p. 141).
- Xie, X., H. Qin, C. Yu, and L. Liu (Dec. 2013). “An automatic recognition algorithm for GPR images of RC structure voids”. In: *J. Appl. Geophys.* 99, pp. 125–134. ISSN: 0926-9851. DOI: 10.1016/j.jappgeo.2013.02.016. URL: <http://www.sciencedirect.com/science/article/pii/S0926985113000487> (cit. on p. 23).
- Xu, B., N. Wang, T. Chen, and M. Li (2015). “Empirical evaluation of rectified activations in convolutional network”. In: *arXiv preprint arXiv:1505.00853* (cit. on p. 27).
- Yilmaz, Ö. (Jan. 2001a). *Seismic Data Analysis*. Society of Exploration Geophysicists. DOI: 10.1190/1.9781560801580. URL: <https://doi.org/10.1190/1.9781560801580> (cit. on p. 61).
- Yilmaz, Ö. (2001b). *Seismic data analysis*. Vol. 1. Society of exploration geophysicists Tulsa, OK (cit. on p. 72).
- Yilmaz, Ö. (Mar. 2003). *Seismic data analysis : processing, inversion, and interpretation of seismic data*. en. Society of Exploration Geophysicists. ISBN: 9781560800941. DOI: 10.1190/1.9781560801580. URL: <https://market.android.com/details?id=book-kYeioAEACAAJ> (cit. on pp. 5, 6).
- Yu, T., D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine (2019). “Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning”. In: *arXiv preprint arXiv:1910.10897* (cit. on p. 56).
- Zabih, R. and J. Woodfill (1994). “Non-parametric local transforms for computing visual correspondence”. In: *European conference on computer vision*. Springer, pp. 151–158 (cit. on p. 119).
- Zabihi Naeini, E., H. Hoeber, G. Poole, and H. R. Siahkoochi (2009). “Simultaneous multitivintage time-shift estimation”. In: *Geophysics* 74.5, pp. V109–V121 (cit. on pp. 109, 116).
- Zhang, Y. and K. V. Paulson (1997). “Magnetotelluric inversion using regularized Hopfield neural networks”. In: *Geophys. Prospect.* ISSN: 0016-8025. URL: <http://www.earthdoc.org/publication/publicationdetails/?publication=33881> (cit. on p. 15).
- Zhao, T., F. Li, and K. Marfurt (May 2017a). “Constraining self-organizing map facies analysis with stratigraphy: An approach to increase the credibility in automatic seismic facies classification”. In: *Interpretation* 5.2, T163–t171. ISSN: 2324-8858. DOI: 10.1190/int-2016-0132.1. URL: <https://doi.org/10.1190/INT-2016-0132.1> (cit. on p. 35).

- Zhao, X. and J. M. Mendel (1988). “Minimum-variance deconvolution using artificial neural networks”. In: *SEG Technical Program Expanded Abstracts*. URL: <https://library.seg.org/doi/pdf/10.1190/1.1892433> (cit. on p. 12).
- Zhao, Z. and L. Gross (Aug. 2017b). “Using supervised machine learning to distinguish microseismic from noise events”. In: *SEG Technical Program Expanded Abstracts 2017*. SEG Technical Program Expanded Abstracts. Society of Exploration Geophysicists, pp. 2918–2923. DOI: 10.1190/segam2017-17727697.1. URL: <https://doi.org/10.1190/segam2017-17727697.1> (cit. on p. 23).
- Zheng, H., J. Fu, T. Mei, and J. Luo (2017). “Learning multi-attention convolutional neural network for fine-grained image recognition”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 5209–5217 (cit. on p. 39).
- Zhu, J.-Y., T. Park, P. Isola, and A. A. Efros (2017). “Unpaired image-to-image translation using cycle-consistent adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232 (cit. on p. 119).
- Zhu, W. and G. C. Beroza (Mar. 2018). “PhaseNet: A Deep-Neural-Network-Based Seismic Arrival Time Picking Method”. In: arXiv:1803.03211 [physics.geo-ph]. URL: <http://arxiv.org/abs/1803.03211> (cit. on p. 36).
- Zitova, B. and J. Flusser (2003). “Image registration methods: a survey”. In: *Image and vision computing* 21.11, pp. 977–1000 (cit. on pp. 109, 117, 118).
- Zoph, B., V. Vasudevan, J. Shlens, and Q. V. Le (2018). “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710 (cit. on pp. 57, 141).
- Zuo, R., Y. Xiong, J. Wang, and E. J. M. Carranza (2019). “Deep learning and its application in geochemical mapping”. In: *Earth-science reviews* (cit. on p. 9).

