Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Erik Arne Huso

Generative Adversarial Networks for Seismic Interpretation

CycleGAN: A novel approach

Master's thesis in Industrial Mathematics

June 2020

NTNU
Norwegian University of Science and Technology

Erik Arne Huso

Generative Adversarial Networks for Seismic Interpretation

CycleGAN: A novel approach

Master's thesis in Industrial Mathematics
Supervisor: Jo Eidsvik
Co-Supervisors: Kenneth Duffaut and Luc Alberts, IGP
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Summary

Seismic interpretation workflows are a crucial part of oil and gas exploration projects. Even though advances in deep learning research have made it possible to simplify some of the workflows, much of the work is manual, tedious, and time-consuming, despite being trivial.

In this thesis, we apply a deep learning method to a seismic interpretation use case in order to further optimise and automatise it. More specifically, we use a Generative Adversarial Network to build a geological macro model based on a given seismic volume. We do this by turning the problem into a domain translation problem, translating seismic data into a representation of the main features of the geological structures present.

We improve a state-of-the-art, semi-supervised domain mapping method, the *cycle-GAN*, by adding a customised penalty term to its objective functions. By minimising the distance of the *autocorrelation* of the generated image and a reference image, we can increase the interpretation quality by reducing the amount of noise present in the predictions. Quantitatively, we reduce the MSE by 21.5%. We further test the capabilities of this modified model by generating a macro model based on real seismic data from the Dutch F3 block dataset. The results are promising, showing that the model indeed is capable of building a decent macro model, given that it is trained well enough. Compared to an analogue, manual process, this vastly reduces the time spent.

We do extensive analysis on how model performance is affected by more or less training data, and similarly how longer training sessions enhance interpretation quality. We find that both more training data and longer training contribute to making the quality of the macro model better. Using a larger dataset may increase accuracy by as much as 30.9%, while also increasing training time by approximately 170%. Conversely, there can also be concluded that even if the size of the training dataset is small, longer training can compensate for this, and vice versa.

Sammendrag

Arbeidsprosesser der tolkning av seismikk inngår, er en nødvendig del av prosjekter der olje- og gassforekomster oppdages. Selv om nylige fremskritt i forskningen rundt dyp læring har gjort det mulig å forenkle noe av arbeidsflyten, er mye av arbeidet fortsatt manuelt, langtekkelig og ressurskrevende, til tross for at arbeidet er trivielt.

I denne oppgaven bruker vi en dyplæringsmodell til å tolke seismikk for å automatisere og optimimalisere arbeidet. Mer spesifikt så brukes et Generative Adversarial Network til å konstruere en geologisk makromodell basert på et gitt seismisk volum. Vi gjennomfører dette ved å gjøre om problemet til et domene-overføringsproblem, der vi gjør om seismikk til en modell som viser de viktigste geologiske strukturene tilstede.

Vi forbedrer en av de mest avanserte og beste modellene som bruker en form for halvveis veiledning i domene-overføring, cycleGAN, ved å legge til et tilpasset straffeledd til dens tapsfunksjon. Ved å minimere avstanden mellom autokorrelasjonen til et generert og et referansebilde, kan vi forbedre bildekvaliteten ved å redusere forekomsten av støy. Kvantitativt så finner vi at MSE reduseres med 21.5%. Videre tester vi denne modifiserte modellen ved å generere en makromodell basert på ekte seismikk fra det nederlandske F3 blokk-datasettet. Resultatene er lovende, og viser at vår modell er kapabel til å konstruere en god makromodell, gitt at den trenes nok. Sammenlignet med en analog manuell prosess, reduseres tiden brukt kraftig.

Vi gjør omfattende analyse på hvordan modellens prestasjoner avhenger av hvorvidt vi bruker mindre eller mer treningsdata, og også hvordan prestasjonen forbedres ved å trene lenger. Vi finner at både mer treningsdata og lenger trening er med på å forbedre kvaliteten i de genererte bildene. Ved å bruke et litt større datasett til trening kan nøyaktighet forbedres med 30.9%, mens tiden det tar å trene øker med 170%. Samtidig kan det konkluderes med at lenger trening kan kompensere for mindre treningssett, og omvendt.

Preface

This thesis concludes my Master of Science degree in Industrial Mathematics at the Norwegian University of Science and Technology (NTNU), with a specialisation in statistics. It has truly been five magnificent years, and I am thankful for all the knowledge and experience I have been privileged to accumulate during this time. Furthermore, I am grateful for having had the opportunity to study abroad. Experiencing new places and cultures is indeed the key to the enrichment of one's life.

I want to thank my supervisor, Jo Eidsvik, for our productive meetings and guidance, answering my countless number of mails, and for quick adaption when everything changed in March. I would also like to thank my co-supervisors, Kenneth Duffaut and Luc Alberts from the Department of Geoscience and Petroleum, for providing me with a different view on the problem, thus challenging me to look at it from several angles.

Further, I would like to thank my girlfriend, Veronika, for supporting me through thick and thin. These five years would not have been the same without you. Lastly, I would like to thank my friends and family for their support throughout the years, and my fellow students for all the good times.

Trondheim, June 2020
Erik Arne Huso

Table of Contents

Summary i

Sammendrag i

Preface ii

Table of Contents iv

List of Tables v

List of Figures ix

1 Introduction 1

2 Literature Review on GANs 5
2.1 Related GAN models 5
2.2 Papers on deep learning methods for seismic image data 6
2.3 Papers discussing different loss functions 7

3 Background on Seismic Data 9
3.1 Introduction to seismic 9
3.2 Reflection seismology 10
3.3 Data 14
3.3.1 Synthetic data 14
3.3.2 Netherlands Offshore F3 block 18

4 Generative Adversarial Networks 21
4.1 Fundamental deep learning theory 21
4.1.1 Activation functions 22
4.1.2 Learning process 23
4.1.3 Convolutional Neural Networks 26
4.1.4 U-Net 28

4.2 Introduction to GANs . . . 29
4.2.1 Definition and mathematical description . . . 30
4.2.2 Game theory . . . 34
4.2.3 Training GANs . . . 35
4.3 CycleGAN . . . 38
4.3.1 Definition and mathematical description . . . 39
4.3.2 CycleGAN algorithm . . . 41
4.4 Model architecture . . . 41

5 CycleGAN with Additional Penalty Term **43**
5.1 Theory . . . 43
5.1.1 Covariance penalty . . . 43
5.1.2 Correlation penalty . . . 44
5.1.3 Kullback-Leibler Divergence penalty . . . 45
5.2 Results . . . 46
5.2.1 Evaluation metrics . . . 46
5.2.2 Covariance penalty . . . 48
5.2.3 Correlation penalty . . . 50
5.2.4 Kullback-Leibler Divergence penalty . . . 52
5.3 Summary . . . 55

6 Results **57**
6.1 Initial results and tuning of hyperparameter . . . 58
6.2 Rich versus sparse data . . . 59
6.3 Training length . . . 64

7 Discussion **67**
7.1 Main remarks . . . 67
7.2 Value of model in practical use . . . 71

8 Conclusion **73**

Bibliography **75**

Appendix **81**

List of Tables

6.1 MSE values for the four cases. 61
6.2 Jaccard index values for the four cases. 61

List of Figures

1.1 Examples of what a GAN imagines would be the visual representation of a celebrity. These people do not exist. Image reproduced from Karras et al. (2018). 2

3.1 An illustration showing how an incoming wave is reflected and refracted with different wave amplitudes dependent on the properties of the two layers. The interface between the two layers is the horizon. 10

3.2 An example of how a RC (reflection coefficient) model in a convolution with a wavelet is used to produce a (synthetic) seismic trace. The RC function shows the different reflection coefficients as a function of depth, and as how the stratigraphy varies. 12

3.3 The illustration shows the relationship between forward and inverse modelling. Figure reproduced from Nanda (2016). 13

3.4 A 2D excerpt of seismic 3D data from the Aasta Hansteen area in the Norwegian Sea. 13

3.5 Visualisation of how the synthetic seismic trace is constructed. The amount of noise added is high. 15

3.6 The image shows a 2D cross section of a geological model including subsurface structures we would like the deep learning model to discover. Specifically horizons, faults and pinch-outs. 16

3.7 Model and belonging seismic data. Different layers of model displayed with different colours. 17

3.8 A 3D view of the geological model of the F3 block. Figure reproduced from Alaudah et al. (2019). 18

3.9 19

4.1 An example of a simple feedforward neural network. In this network, the input layer (in yellow) with input vector $\mathbf{x} = [x_1, x_2, x_3]^T$ are connected to a single output (in green) through two hidden layers, displayed in light blue. Weight matrices W_1, W_2 and W_3 links the neurons together. Moreover, f and O are predefined activation function for the hidden layers and the output, respectively, and $\mathbf{a}$ is the activation outputs from the hidden layers. The network also uses *bias*, b, to adjust the outputs to give a best possible fit. Here, b is the same everywhere, but different bias values for each neuron is also possible. 22
4.3 Example showing how max pooling works. 28
4.4 Architecture of the UNet. It consists of two main parts: a contracting path and an expansive path. In the contractive phase, spatial image information is reduced while feature information is increased. In the expansive phase, feature and spatial information is combined to propagate context information to a higher resolution. Figure reproduced from Ronneberger et al. (2015). 28
4.5 A high-level sketch showing the architecture of a standard GAN. The Generator generates images as close to the target as possible, while the Discriminator tries to separate generated images from the real training samples. 29
4.6 Plot showing the trajectory of simple GD for the value function $V(x, y) = xy$. It will orbit the equilibrium solution at $x = y = 0$ at a constant radius. 37
4.7 The *cycleGAN* architecture. The model consists of two separate GANs, mapping images from opposite domains. Figure reproduced from Wang and Deng (2018). 39

5.1 A figure illustrating how the basic cycleGAN occasionally makes predictions that are noisy. Notice the area in the upper left part of the predicted image. 44
5.2 Comparison of accuracies of baseline model (without any additional penalty terms) and a model where a covariance distance penalty is added. We observe how the MSE has sudden drops in the plots where additional penalty is added. 48
5.3 MSE of model with additional penalty term trained for 300 epochs. . . . 49
5.4 50
5.5 MSE of the model accuracy where $\gamma = 0.01$ and $\gamma = 0$. The plots represents a mean of 10 simulations each, to get a more representative result. . 51
5.6 MSE of the model accuracy where n denotes the size of the training set. Here, γ is set to 0.01 where correlation penalty is included. The data is based on the mean of five simulations. 52
5.7 Comparison of model output with and without additional penalty term. The model is trained for 50 epochs in all four cases and $\gamma = 0$ and $\gamma = 0.01$ respectively. 52
5.8 An illustration of how the KLD evaluates the distance in distribution of histograms of a reference and a generated image. 53
5.9 53

5.10 MSE of an average of 5 sessions with $\gamma_K = 0.01$ and $\gamma_K = 0$. The reduction of MSE when KLD penalty is added is significant. 54
5.11 Comparison of model output with and without additional KLD penalty term. The model is trained for 150 epochs in all four cases and $\gamma_K = 0$ and $\gamma_K = 0.01$ respectively. 55

6.1 A figure illustrating how we extract lines from a given volume. The light blue area represents the seismic volume, while the light yellow planes indicates evenly extracted 2D lines. 57
6.2 58
6.3 Model-generated outputs compared to its reference. The model is trained for 150 epochs in both cases. 59
6.4 A figure illustrating how a prediction may differ from its true label. Green pixels indicates areas where the difference between the generated label and the actual label is bigger than some small threshold (here, 0.1 is used). . 59
6.5 A special case where the model tries to include more colours/horizons than what is present in the seismic. 60
6.6 Model output after a training session of 60 epochs. 60
6.7 MSE for models trained on rich and sparse datasets. The rich dataset is based on 20 inlines and 20 crosslines. The sparse dataset consists of 5 inlines and 5 crosslines. Complex seismic indicates the Eastern region, while simple seismic indicates training and predicting on the Western region. 61
6.8 Comparison of model outputs after 50, 150 and 300 epochs, training on a sparse and a rich dataset. 62
6.9 Comparison of model outputs trained on sparse and rich datasets for a horisontal slice across the volume, at depth 210. 64
6.10 MSE for three sessions where the model is trained on a sparse dataset, with a benchmark consisting of several sessions averaged into one where no additional penalty term is included. 65
6.11 Comparison of model outputs after a various number of epochs. We compare output from two different inlines. 66

7.1 Model output from a case where we train on complex seismic but predict on simple seismic. The model is trained for 300 epochs. 68
7.2 Time plotted as a function of epochs for training sessions on a sparse and a rich dataset. 70

Chapter 1

Introduction

In a time where the industry faces some of the biggest challenges ever seen, with dropping oil prices, a coming, necessary adaption to a greener world and general uncertainty of what the future brings, the need of a digital transformation - the digital leap - is larger than ever. The industry needs to ameliorate their workflows and increase profitability in current and future projects. Digitalisation is viewed as one of the main solutions to overcome the massive challenges.

The exploration phase of oil and gas (i.e. hydrocarbons) projects has always been an extensive process, as there is high risk involved in investing in well drilling operations. High confidence in possible prospects is required as the drilling operations are very costly. At the same time, many of the workflows in the exploration phase are still highly manual and thus expensive. Geoscientists are, for example, still highly involved in interpretation procedures. Ultimately, the experience and knowledge of a geoscientist will always be a crucial ingredient in the exploration process. But there are also parts in the current interpretation workflows where human intervention is excessive. Making these parts fully automatic could release resources that would be better spent elsewhere, for instance on working with reducing emission footprints in the industry.

Technologies like machine learning (ML) are sets of tools that potentially can be - and already are - applied to these certain problems to automatise and digitalise them. We now experience a ”boom” in oil and gas industry where ML methods are applied to many problems to simplify workflows. Many of these attempts turn out to be successful, proving to be valuable assets in the industry. Deep learning, a branch of machine learning, is one of the most popular sets of methods existing in this field today. Recent advances in research here have led to several interesting new concepts. The research on how these concepts can solve real problems like hydrocarbon exploration problems is ever-growing.

In this project, we apply one of these deep learning methods, a *Generative Adversarial Network* (GAN), on an interpretation use case. We investigate how this method can be used in creating a workflow for automatically picking the brightest horizon events in seismic volume data, creating a geologic, three-dimensional macro model of the volume.

Since they first saw the light of day in 2014, GANs have gained vast amounts of pop-

ularity, and a rapid increase in various applications.

It has also been called *"The most interesting idea in machine learning in the last decade"* by artificial intelligence (AI) pioneer Yann LeCun[1]. GANs have shown to produce very impressive and realistic results in image generating, and has been used in everything from generating fake celebrity images (as seen in Fig 1.1) to medical imaging to image resolution enhancing and 3D object generation.

Figure 1.1: Examples of what a GAN imagines would be the visual representation of a celebrity. These people do not exist. Image reproduced from Karras et al. (2018).

It is said to be making deep learning more *human* by letting the network *create* instead of just predicting an output based on training data. It has a semi-supervised approach in the sense that it tries to learn underlying features in sample images without any other supervision than being told how a target image *could* look like. Because of this it accommodates powerful capacities, and together with the momentum it gets from the current GAN hype, it is interesting to see if it can be used as a tool in oil and gas exploration.

The most popular applications seen so far have been related to image generation, for instance in generating super-realistic human faces (Karras et al., 2018). But also image translation, e.g. turning a photograph into artistic painting style or turning a satellite aerial photograph into a map-like view (Isola et al., 2017), has become popular and demonstrated the versatility of the GANs. In the seismic world, most of the research done on applying these generative models have focused on seismic data reconstruction (Siahkoohi et al., 2018), processing (Picetti et al., 2018) or to generate distributions for priors for geological structures used in seismic inversion problems (Mosser et al., 2018a). In this project, we reconstruct and further develop some of the work that was first performed by Mosser et al.

[1] `http://yann.lecun.com/`

(2018b). Here, they used a GAN to construct a subsurface geological two-dimensional model based on seismic data input.

Initially, we will do this by using synthetic data, but the main aspect of the thesis will be analyse model performance on real data. We will investigate how the model performs given data of different degrees of complexity and varying training dataset size, and try to expose and understand the weaker and stronger points of using such a model for a task like this. We will also make new contributions to the deep learning research field by improving the proposed GAN model using statistical methods. The proposed model we use in this thesis is a so-called *cycleGAN*, a GAN model designed for image translation. We perform analysis on a basic cycleGAN model by comparing model accuracy between datasets containing relatively much data (rich datasets) and less data (sparse datasets), and we measure how model performance varies with the length of training. We say *relatively* here, as one of the strengths of the model we are using is that it is capable of generating good quality output even when training set size is small.

We also seek to increase performance by implementing additional penalty terms to the model, leading to more precision in model output predictions. Although some of this has been done in previous research studies, such an extensive analysis study and further development on the model itself has never been done before in this field.

Outline

Throughout this master thesis we will explain and investigate different aspects related to the stated objectives. Below follows an outline of the contents of the thesis:

- Chapter 2 contains a brief summary of a literature survey performed in order to gain knowledge and expand our view on topics related to that of this thesis.
- In Chapter 3 we present some basic theory on seismic data and seismic data processing, as well as a brief review of the data we make use of in this thesis.
- In Chapter 4 we present some essential deep learning theory as well as an introduction to the theory of GANs along with some theoretical results. We also present the cycleGAN framework.
- Chapter 5 concerns the work related to modifying the basic cycleGAN model. We present the theory behind the penalty terms as well as the results from tests where we apply them.
- Chapter 6 contains the results from applying our modified model on a real seismic dataset.
- Chapter 7 contains a general discussion regarding the results from the previous chapter and some reflections on the fitness of our model when applied to a real case.
- In Chapter 8 we draw a conclusion to what we have seen in this thesis and what possibilities that lies in further research related to the subjects discussed.

An extended abstract based on the results of this thesis was also made and submitted to the *EAGE Digital*[2] 2020 conference. The abstract can be found in the appendix.

[2] `https://eage.eventsair.com/digital2020/`

Chapter 2

Literature Review on GANs

During the preparation phase of this thesis, a literature review was performed. This was necessary to gather knowledge, understanding and inspiration from subjects closely related to and in vicinity of the topic of the thesis. GANs are a rather new field of research, hailing from 2014, and facing a rapid growth of interest from the deep learning community. It is therefore highly useful to explore what this new technology can do, and try to keep up with the latest research conducted. The following sections provide a summary of the most interesting findings from this survey.

2.1 Related GAN models

The amount of research conducted on GAN models after the first paper was published in 2014, has exploded and promising new technology is rapidly introduced. For the topic of the thesis, we are interested in GAN models which can be utilised in domain mapping, much like the original CycleGAN (Zhu et al., 2017). One of these new GAN successors is the *StarGAN* (Choi et al., 2018), a unified GAN for multi-domain image-to-image translation. Despite this, only one generator is needed. The loss function of the generator consists of the traditional adversarial loss, a domain classification loss and a reconstruction loss. StarGAN is able to produce a high quality domain translation. Its successor, StarGAN v2 (Choi et al., 2019) is also interesting, where the domain label is replaced by a domain specific style code that can represent diverse styles of a specific domain.

There exists several interesting model approaches for domain translation. The *Cycada* (Hoffman et al., 2018), is an alternative, cycle-consistent method which improves pixel-level and low-level shifts. It represents a cycle-consistent adversarial domain adaptation method that unifies cycle-consistent adversarial models with adversarial adaptation methods. *AttentionGAN* (Chen et al., 2018) is a method of unpaired image-to-image translation using an attention-guided GAN. The method offers more stable GAN training and thus improves performance. The paper explores attention masks and content masks so that image backgrounds are better preserved and low-level features of the input is captured. *InstaGAN* (Mo et al., 2019) proposes a method for multi-domain image translations. This

method also preserves background as well as improving the quality of instance-generated segmentations. A paper by Jiang et al. (2019) explores how image-to-image translations can be carried out using segmentation guiding. This approach makes the generated images spatially controllable. The paper by Benaim and Wolf (2017) takes on a different approach to one-sided image translation. By learning a mapping that maintains the distance between a pair of samples, a one-sided mapping can be performed. More specifically, this is done by using target domain identity and distance constraints.

In a paper by Shen et al. (2020) the focus is on improving the performance of the cycleGAN model by improving the cycle-consistency objective. They show that the cycle-consistency rule allows for non-unique mappings. By separating the loss functions between the transfer operations, higher accuracy in mapping is achieved. A paper by Fu et al. (2019) introduces a one-sided unsupervised domain mapping model, building on the cycleGAN model but including an extra geometry-consistent constraint. The loss can be seen as a reconstruction loss that relies on a predefined geometric transformation function, dependent on the problem. The *Mind2Mind* (Frégier and Gouray, 2019) model is based on the principle of transfer learning, and the paper discusses how this can be applied in GANs. It is shown that cases where transfer learning has been used, convergence is faster. There are promising results, albeit little research has yet been conducted on the subject. *TempoGAN* (Xie et al., 2018) is a generative model addressing the super-resolution problem for fluid flows. By combining temporal and spatial discriminators and customised penalty terms the model is able to generate high-quality reproductions of fluid flows.

2.2 Papers on deep learning methods for seismic image data

Mosser et al. (2018b) uses a cycleGAN model to perform inverse modelling on mostly synthetic seismic data, and obtains some promising results. This shows that GANs may be a suitable method for working with seismic data.

In the paper *Seismic data interpolation using CycleGAN* by Kaur et al. (2019), a modified cycleGAN model is used to perform seismic trace interpolation. The model is modified by adding a self-distance loss (also discussed in Benaim and Wolf (2017)), basically comparing the left side of the generated image by the right side.

In Lu (2019), a summary of where the industry currently stands when it comes to deep learning in seismic research is given. One example that is highlighted, is how GANs are shown to provide impressive results in seismic trace reconstructions. In the paper by Picetti et al. (2018), a method for improving seismic image processing using GANs is discussed. The aim of the paper is to develop a model that can produce a high-quality seismic image from migrated seismic data. Enhancing seismic imaging quality is also discussed in Lu (2019), where a neural network in combination with full waveform inversion is proposed. The predictive abilities of a neural network is used to support the full waveform inversion. The predictions from a neural network is used to correct and support the inversion procedure so that a symbiosis between the two methods takes place. In Akhmadiev and Kanfar (2019), GANs are used to compensate for lateral mispositioning of the images and where the lack of *a priori* information of the velocity model is significant. Here, the

cycleGAN model architecture is slightly changed to increase seismic image defocusing performance. In Cho et al. (2020), tracking horizons using a convolutional neural network is discussed. The authors demonstrate how this can be more accurate than manual tracking using conventional methods, and also with a small training dataset. Horizon tracking is also discussed in Koryagin et al. (2020), but with additional focus on generalisation of the method so that the trained network can perform well on a different type of seismic than it was trained on. The network uses binary classification to detect horizons, using a specific Dice coefficient loss function. In Wu et al. (2019) horizon tracking is demonstrated using an encoder-decoder network. By treating each seismic trace as a 1D image, and by extracting the low-level features that resembles reflectors the authors show that the method can perform at least as well as when standard tracking methods are applied. In Huang et al. (2005) the neural network property of local connection is utilised to extract reflectors from seismic data through horizon linking. They use cellular neural networks, which are networks where only neighbouring neuron communicate with each other. An energy function is applied to work as a constraint, helping the network detecting the horizons.

2.3 Papers discussing different loss functions

We have reviewed papers discussing conventional deep learning methods, but where the objective functions are changed to enhance accuracy in the model outputs. In He et al. (2020) a method for cell detection in cases where the data is only partially or incorrectly labelled is proposed. A cycleGAN model is modified by adding a penalty term consisting of the measure of the binary classified image multiplied by a background image - which yields the desired information.

Nabian and Meidani (2020) propose a framework for using physics-driven regularisers to train neural networks. The regularisers are constructed so that they help the network avoid violating the constraints governed by the problem it solves. Physics-based regularisers are also used in Pan et al. (2018) where they act as penalty in an image restoration case. A measure of the difference between a physics-based mapping and the actual output helps increase accuracy. In Yang et al. (2018) a GAN framework for solving stochastic differential equations is discussed. The model learns not only from a training set, but also from governing physical laws which works as further guidance in the training process. In a paper by Hiasa et al. (2018), a modified cycleGAN model is used to generate medical images used to aid in orthopedic procedures. The cycleGAN model is modified by adding a gradient consistency loss in order to improve accuracy at image boundaries.

In Wu et al. (2020) and Yang et al. (2019), statistical and physical constraints are added to the objective function to improve accuracy and convergence rate in modelling chaotic dynamical systems. This is done by helping the generator to decrease weight space so that the generator and the discriminator can be more synchronised in their learning. The physical constraints are approximate physical conservation laws, while the statistical constraints are a measure of the difference between the covariance structures of the data distribution and the generated distribution.

Chapter 3

Background on Seismic Data

To get a better perception of the essence of this thesis, a basic understanding of seismic data and seismic data processing is necessary. This chapter will contain a short introduction to the subject, as well as a presentation of the data we later make use of.

3.1 Introduction to seismic

Seismic data analysis is a large field, including everything from seismic data acquisition, seismic processing and seismic interpretation. We will not delve deep into many of these subjects in this thesis, but interested readers can learn more about this in classic textbooks like *Exploration Seismology* (Sheriff and Geldart, 1995). The use of seismic data is one of the most crucial parts of today's oil and gas exploration. With seismic data, geoscientists can get a visualisation of the subsurface of the Earth through mapping of geostructures by soundwaves. The process can be compared to how ultrasound is used in medicine, or how marine vessels use sonar to navigate and detect objects.

A seismic survey is performed by emitting sound pulses into the subsurface. They are reflected at different depths dependent on the contrast between the *Primary-* and *Secondary* wave velocity and density between the different rock layers. The waves move with different speed, and primary waves are the fastest of the two. An interface, a bedding surface or a sudden change in the lithology, is called a *horizon*. The crust of the Earth consists of numerous horizons separating layers of different texture and characteristics from each other due to evolution, weather and the dynamics of the Earth's inside. Figure 3.1 shows a horizon between two layers.

For offshore seismic surveys, sound pulses from seismic vessels are reflected back to sensors either placed on the seabed or towed behind the vessel as cables. There are several categories of surveys: two-dimensional (2D) surveys provide a relatively low-resolution cross-section of the subsurface right beneath the sensor cable (the cross-section consists of a vertical depth axis and a horizontal length axis), and is usually used in new exploration areas as reconnaissance data. Three-dimensional (3D) seismic provides a more detailed view, as it also includes a third spatial dimension (width) and is thus a volume. A 3D

seismic volume consists of 2D seismic *inlines* and *crosslines*, where crosslines are lines (or slices) perpendicular to the direction in which the data was acquired, and inlines are lines parallel to this direction. 3D data is often used in appraisal phases of the exploration, where a more detailed view of the underground is needed, as 3D data offers a better resolution of the spatial appearance of subsurface structures. Four-dimensional (4D) data consists of stacks of repeated 3D surveys of the same area, and is mostly used to detect changes over time in a hydrocarbon reservoir where production or injection is performed.

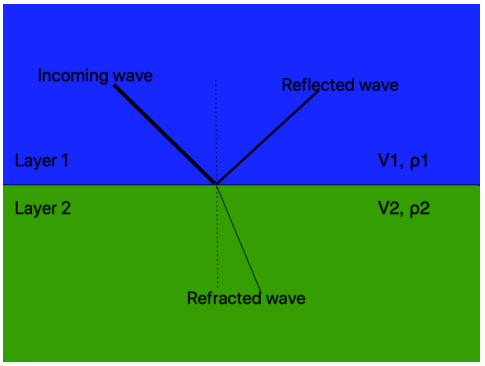


Figure 3.1: An illustration showing how an incoming wave is reflected and refracted with different wave amplitudes dependent on the properties of the two layers. The interface between the two layers is the horizon.

3.2 Reflection seismology

Seismic waves can be described as mechanical perturbations travelling in the Earth's interior with a speed governed by the medium in which the waves are travelling. The seismic velocity can be used to calculate the acoustic impedance of the wave, which is crucial to further determine the reflectivity of the wave. The impedance is defined through the relation

$$I = V\rho, \tag{3.1}$$

where ρ is the density of the medium the wave is travelling through and its velocity V. The velocity V can be either that of a P-wave (V_p) or that of an S-wave (V_s) depending on the mode of interest. The resulting impedance would then be either P-impedance ($V_P\rho$) or shear impedance ($V_S\rho$).

When the wave hits an interface between two different materials with different properties, some of the wave energy will be reflected back to the surface while the rest is refracted through. By generating these seismic waves and then measuring the time before they are reflected back to the surface, one can reconstruct the pathways of the waves so that an image of the underground can be built. See e.g. Yilmaz (2001) for more on this.

The amplitude of a reflected wave can be predicted by multiplying the amplitude of the incident wave with a reflection coefficient, RC, which is found by looking at the

impedance contrast between the two materials:

$$\text{RC} = \frac{I_2 - I_1}{I_2 + I_1}, \tag{3.2}$$

with I_1 and I_2 being the respective impedance of the media. To find the amplitude of the refracted wave, the transmission coefficient, $T = 1 - R$ can be applied in a similar fashion. By investigating the changes in the reflector strengths, changes in the seismic impedance can be mapped, which in turn also can be utilised to infer changes in the properties of the rocks, like density and elastic modulus. We will not cover this in further detail here, as it is not directly relevant to the studies conducted in this thesis. In Figure 3.1 we see an example of how a wave behaves when encountering an interface between two strata.

Forward modelling

Forward modelling of seismic data is used to create synthetic seismic data based on the geological properties of a designated area. The synthetic seismic traces are compared with real seismic data acquired in the field, to see if they agree. If the accuracy is within a given tolerance, the geological model can be used as a reasonably accurate representation of the underground. If not, the geological assumptions are altered and new synthetic data is made. This happens iteratively until a satisfactory match is achieved. Forward modelling can, in many ways, be seen as the opposite of *inverse* modelling, where the geological model parameters are computed from real data. In both cases, the final objective is the same: to determine the geological structure and lithology of the subsurface.

One essential part of forward modelling is the calculation of synthetic seismic traces. There exist numerous methods for seismic modelling, and among the simplest ones for calculating seismic traces are convolution-based models (Sen, 2006). In this method, the seismic, ultimately a time series, $S(x, t)$ is computed using an assumed source function $\omega(t)$, a wavelet, in a convolution with the reflection function $\text{RC}(x, t)$. This function shows how the reflections vary with depth. This is based on the different reflection coefficients R in the subsurface. From this, we get

$$S(x, t) = \omega(t) * \text{RC}(x, t), \tag{3.3}$$

where x is the spatial location and t is the two-way vertical seismic travel time, the time it takes for a wave to travel from surface and back after being reflected. A commonly used wavelet is the Ricker wavelet (Ricker, 1953),

$$\omega(t) = (1 - 2\pi^2 \omega_{max} t^2) e^{-\pi^2 \omega_{max}^2 t^2}, \tag{3.4}$$

where $\omega(t)$ is the amplitude of the wavelet at time t and ω_{max} is the maximum frequency of the wavelet. In Figure 3.2 we illustrate how the reflection function (to the left) is convolved with a wavelet to produce a seismic trace, a single 1D representation of the seismic.

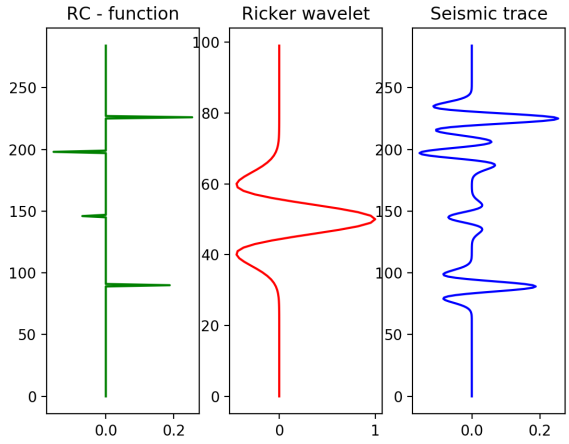


Figure 3.2: An example of how a RC (reflection coefficient) model in a convolution with a wavelet is used to produce a (synthetic) seismic trace. The RC function shows the different reflection coefficients as a function of depth, and as how the stratigraphy varies.

Inverse modelling

Working with seismic data can in many cases be considered to be an *inverse problem*: to develop an abstract model, in this case a model of the structure and properties of the Earth's crust, by using collected data and physical laws. Being an inverse problem also implicates that the output usually is not unique, because more than one model may fit the data. It therefore requires additional information to decide on an appropriate model. Knowledge about the local geophysical and geological conditions is necessary to narrow down the range of potential models. Inversion is an iterative process, where one tries to minimise an objective function which says something about the distance between the estimated and the "true" model. The estimated model is based on acquired seismic data while the "true" model may be based on actual data from the subsurface, e.g. well logs (boreholes from which rock property data is extracted). Then, the inversion parameters transforming the seismic data are iteratively adjusted so that eventually a model that explains as much of the properties of the subsurface area as possible, is found. This is analogous to minimising the objective function. As previously described, this is in many ways the opposite of forward modelling, and this relationship is illustrated in Figure 3.3.

Inverse problems are sensitive to errors, as many assumptions are being made in the process, and great care must be taken when interpreting a seismic survey. Typically, the recorded signals are subject to extensive preprocessing before they are ready to be interpreted.

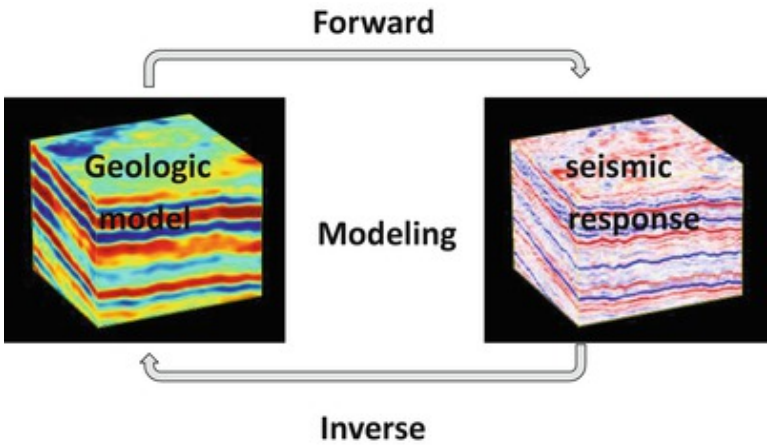


Figure 3.3: The illustration shows the relationship between forward and inverse modelling. Figure reproduced from Nanda (2016).

Seismic interpretation

Seismic interpretation is a key part of accumulating understanding and knowledge about the subsurface of the Earth. The goal is to obtain a coherent geological story from processed seismic reflection data. By tracing and correlating the continuous reflectors in 2D or 3D seismic data, a basis for geological interpretation can be established.

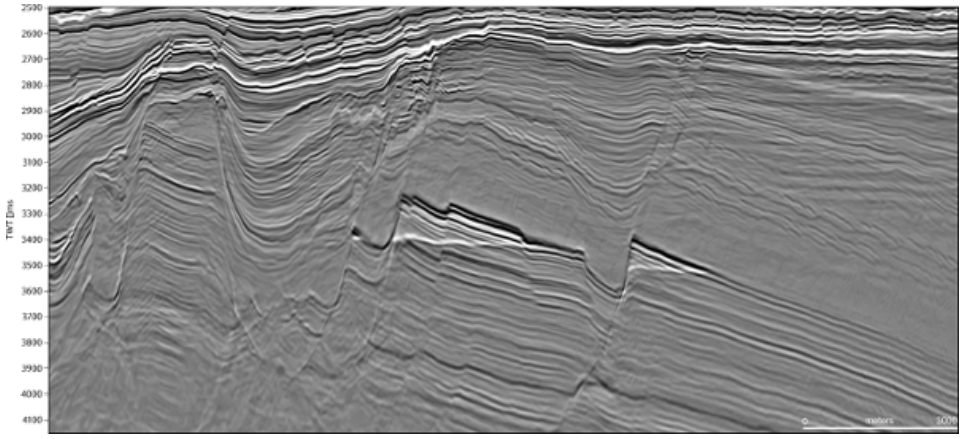


Figure 3.4: A 2D excerpt of seismic 3D data from the Aasta Hansteen area in the Norwegian Sea. Image reproduced from *Norsk Petroleum* [1].

An example of post-processed seismic reflection data is displayed in Figure 3.4 . The aim of doing interpretations on images like this is to produce structural maps that represent

[1] https://www.norskpetroleum.no/en/exploration/seismic-surveys/

the spatial variation in depth of certain geological layers and features, e.g. a geological macro model. These models are important tools in identifying hydrocarbon deposits or *traps*; locations where hydrocarbon reservoirs can be found. In Figure 3.6, two different types of traps are illustrated, a fault and a pinch-out. A fault is an example of a *stratigraphic* trap, which are formed as a result of the deposition in the sedimentary rocks. A pinch-out is an example of a *structural* trap, formed as a consequence of some structural deformation of rock layers. Being able to identify these structures can thus be an important tool in hydrocarbon detection.

However, due to noise and processing steps, seismic images often are of low quality. This leads to some extent of uncertainty in the interpretations and there is often more than one solution that fits the data. Because of this, the need for human proficiency and presence in these interpretation workflows are necessary, as well as being crucial for obtaining geological understanding and knowledge of the area of interest, prior to a possible well drilling.

Extracting key horizons and surfaces is an important part of the interpretation workflow, but even though semi-automatic horizon tracking tools have been available since the 1990s (Harrigan et al. (1992); Leggett et al. (1994); Dorn (1998)) most of them still rely on a certain amount of manual effort by a human. Setting so-called seed points (points from which regions where correlation is high can be tracked) on target geological features, i.e. horizons, so that the deterministic tracking algorithm can interpolate on nearby reflectors with high correlation, must be done manually. The tracking is not fool-proof, so a manual quality check for possible miscorrelations must be performed regularly. Some seismic also needs reinterpretation. This may be the case if the seismic is reprocessed or if it belongs to a 4D seismic dataset. Performing tasks like this on large amounts of seismic data is tedious, subjective and time-consuming albeit being almost trivial.

With computational power being ever-increasing, data science makes room for new methods for solving problems. This is also the case for problems related to extracting information from seismic data. Although the standards of such solutions must be very high, there are methods in the domain of machine learning that may meet them, as we later in this thesis will see.

3.3 Data

In order to perform the experiments involved in this thesis we use both synthetic and real data. The synthetic data is used in an initial phase, mostly in the penalty term exploration phase. We use synthetic data here so that we can evaluate model performance on specifically customised seismic data. However, testing the final model with real data is the most interesting part, and thus poses the main part of the results and analysis later displayed in this thesis.

3.3.1 Synthetic data

For parts of this thesis, we have focused on creating synthetic subsurface models and seismic data for initial testing of the modified deep learning model. For this, we have used the principle of forward modelling on a synthetic subsurface rock model with artificial

density-velocity pairs for each of a predefined number of layers. Hence, by knowing properties like the density and the wave velocity through a layer, we can find the impedance I and furthermore construct a reflection coefficient model. By using a convolution with a wavelet (thus following the steps of Eq. 3.1 - 3.3) we can construct seismic data of the subsurface model.

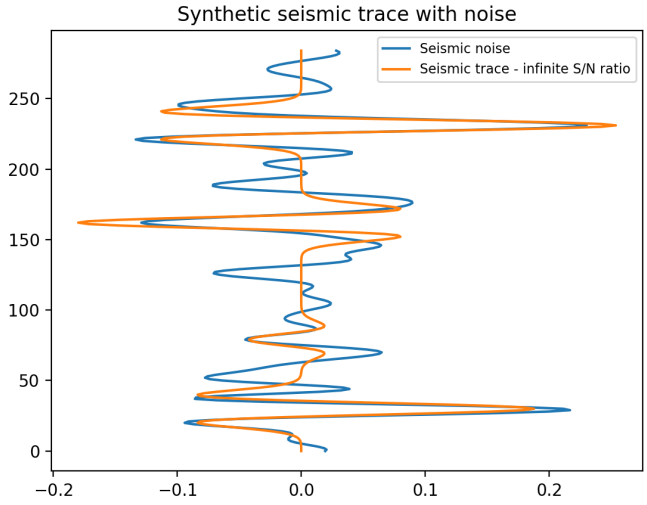


Figure 3.5: Visualisation of how the synthetic seismic trace is constructed. The amount of noise added is high.

Figure 3.5 depicts two seismic traces. One where the Signal-to-noise ratio (S/N ratio) is infinite, meaning that there is no noise present, and one where the S/N ratio is much lower, i.e. a more noisy trace. We use seismic data with noise in this project in order to imitate genuine data.

Specifically, in order to generate the synthetic seismic volume $\mathcal{M}$ we first specify the size of the lateral area. We fill this grid area with samples from a normal distribution $N(0, \tau^2)$. This area $\mathcal{S}_j$ forms the basis of a single layer j.

Depending on how many stratigraphic layers the volume contains (in these experiments we generate and use a model with 4 layers), we add noise, $\eta \sim N(0, 1)$, to each of them to create dissimilarities between them. We end up with $\mathcal{M} = \{\mathcal{S}_1, ..., \mathcal{S}_4\}$. The layers are then smoothed using a convolution with a Gaussian filter of the form

$$g(u, v) = e^{-\left(\frac{u^2}{a} + \frac{v^2}{b}\right)}. \tag{3.5}$$

Here, u and v are spatial positions in $\mathcal{S}_j$ and $a = b = 600$ are positive constants controlling the magnitude of smoothing. Using this smoothing approach leads to a volume where the layers are similar to each other, but with certain individual characteristics. In this way, we obtain more authentic data. The layer generating procedure for 3D cases is illustrated in Algorithm 1.

Algorithm 1: 3D Interface

Result: Multiple 3D Interfaces
g is a Gaussian filter;
$\mathcal{M} = \emptyset$;
$m \times n$ size of area of the layers;
for *1:# layers* **do**
  generate a 2D array $\mathcal{S}_j$ consisting of random values from $N(0, \tau^2)$ in a grid of size $m \times n$;
  add noise $\eta \sim N(0, 1)$ to $\mathcal{S}_j$;
  convolve the grid $\mathcal{S}_j$ with g;
  rescale $\mathcal{S}_j$;
  add $\mathcal{S}_j$ to $\mathcal{M}$;
end
return *volume* $\mathcal{M}$

The layers are placed at different depths in the three-dimensional matrix and for each layer, the spatial coordinate of one specific entry denotes the depth of the layer at that location. To finally generate synthetic seismic we follow the same steps as previously described: the stratification is formed by using artificial (V, ρ)-pairs to form impedance. Furthermore, we add independent noise, $\epsilon \sim N(0, \sigma^2)$ to the whole volume, to decrease signal-to-noise ratio in the seismic and thus make it harder to interpret. Here, $\sigma^2 = 0.08$ is used.

We also increase complexity by creating a fault plane on a layer, similarly to how it is illustrated in Figure 3.6. We do this by "lowering" a subsection of one of the two mid-layers, creating a vertical fault plane.

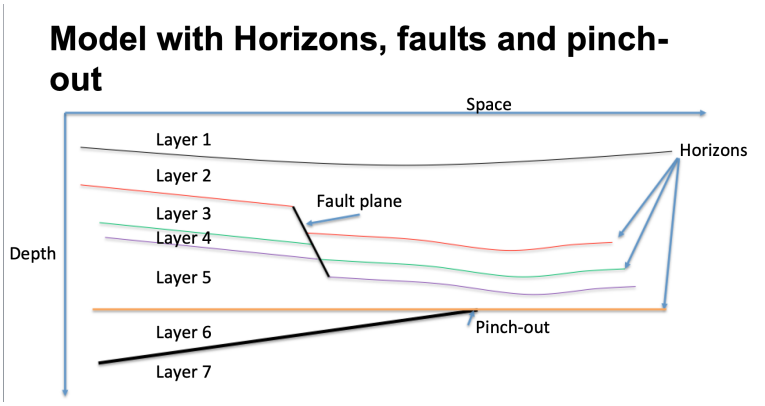
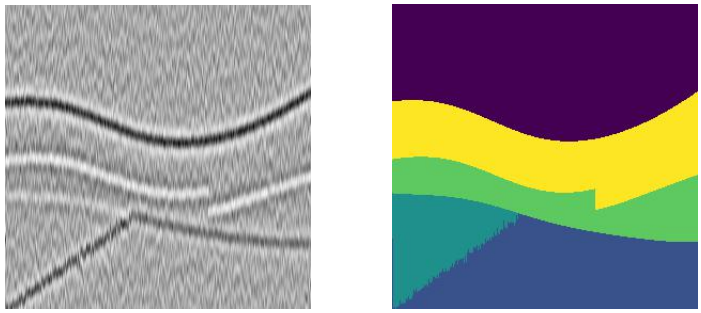


Figure 3.6: The image shows a 2D cross section of a geological model including subsurface structures we would like the deep learning model to discover. Specifically horizons, faults and pinch-outs.

Additionally, a pinch-out is added. We modify the lowest layer by letting the first part be a linear plane. When this plane intersects the interface higher up, we let the plane follow

this layer for the rest of the length of the section. In this way, we get a slightly simplified version of the model displayed in Figure 3.6.

An inline of the resulting synthetic volume is displayed in Figure 3.7.



(a) Example of seismic data with moderate amounts of noise, containing 4 reflectors, a vertical fault plane and a pinch-out.

(b) Example of geological model. We observe that the model contains 4 stratigraphic layers, a vertical fault plane and a pinch-out.

Figure 3.7: Model and belonging seismic data. Different layers of model displayed with different colours.

3.3.2 Netherlands Offshore F3 block

When we have found an appropriate cycleGAN model setup, we will proceed to experimenting with real data. For this we use the F3 block dataset[2] from offshore Netherlands. This dataset is publicly available and is widely used for experiments related to machine learning implementations. In addition to the original dataset we also use labelled data prepared by Alaudah et al. (2019). The labelled 3D model is based on 26 well logs as well as the provided 3D seismic data.

The Netherlands F3 dataset is a seismic survey (made by using acoustic impedance) of about 384km^2 in the Dutch offshore portion of the *Central Graben* basin in the North sea. The survey was acquired in 1987. From Figure 3.8, we can see that the volume consist of several geological structures, both horizontally and vertically. In this project, these are the main structures, or rather the horizons separating them, that we try to identify. In this thesis, we will divide the block into two regions: the Western and the Eastern region. The Eastern region is characterised by more complex structures, while the Western is of a more simple form.

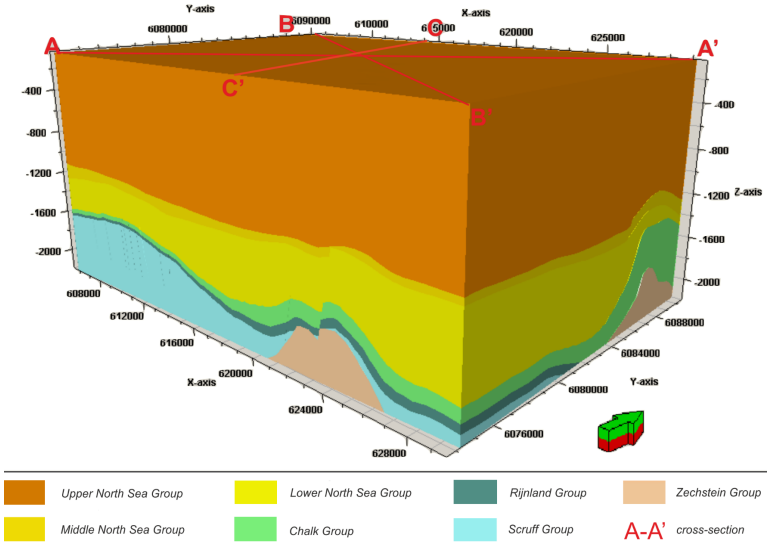
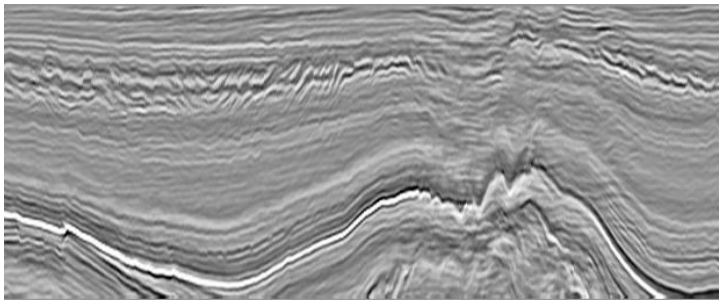


Figure 3.8: A 3D view of the geological model of the F3 block. Figure reproduced from Alaudah et al. (2019).

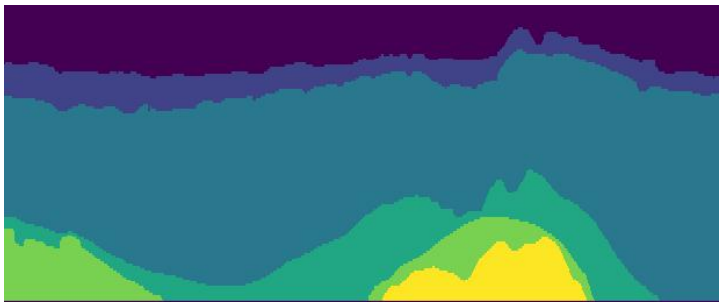
An excerpt showing both a seismic and a labelled inline is shown in Figure 3.9. The excerpt is showing a part of the Eastern region of the volume.

The labelled dataset is created by using dedicated modelling software (Petrel[3]). Some built-in functions are used in the processing: *seeded 3D autotracking* is used to interpret the horizons. A *polygon editing* tool is used to interpret the main fault surfaces and fault networks. Based on the interpreted faults and horizons, preliminary modelling is con-

[2] `https://terranubis.com/datainfo/Netherlands-Offshore-F3-Block-Complete`
[3] `https://www.software.slb.com/products/petrel`



(a) Seismic inline from the Netherlands F3 block dataset.



(b) Labelled inline from the Netherlands F3 block dataset.

Figure 3.9

ducted using *horizon modelling*. Finally, the *structural modelling* module is used to create the 3D geological model, as seen in Figure 3.8.

Chapter 4

Generative Adversarial Networks

This chapter will cover the theory behind some of the models this project seeks to utilise, as well as some essential deep learning theory. In the first section we will present some general deep learning theory around concepts like neural networks, both structurally and how they learn. Further, we introduce the basic GAN with some theoretical results and lastly the cycleGAN, explaining the structure and how it functions.

4.1 Fundamental deep learning theory

An *Artificial Neural Network* (ANN or simply NN) is a computational system inspired by how neurons in the human brain work in a learning process. Modelling the behaviour of neurons was first performed in the 1940s (McCulloch and Pitts, 1943), and the first, simple neural network - the *Perceptron* - was introduced in Rosenblatt (1958). But it is not until the last few decades that the concept has gained popularity, due to processing power finally reaching an acceptable level.

Most of today's ANNs consist of interconnected groups of nodes, organised in one or several layers usually forming a directed, weighted graph. The weights dictate the relative importance of the relationship between two nodes. Each node is representing a neuron, which receives an input (usually weighted information from the preceding layer), combines this with its internal state (an *activation function*) and produces an output, which is sent to the connected neurons in the succeeding layer. A somewhat simplified output a_k of one neuron k can be summarised by

$$a_k = \phi\left(w_k^T x + b\right), \tag{4.1}$$

with $w_k^T x$ being the dot product between signals (outputs) x from connected neurons from previously in the graph and their belonging weights w_k, b is the *bias* and ϕ some activation function (further explained in the next section). Networks with this directed, acyclic graph architecture are called *feedforward* neural networks (Zell et al., 1994). Such a network is visualised in Figure 4.1.

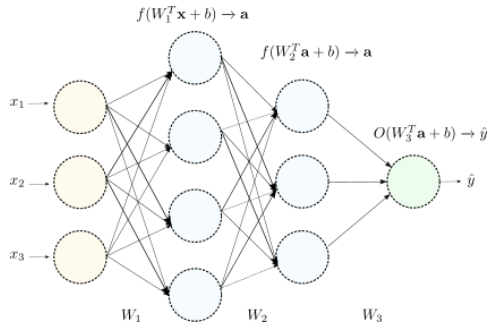


Figure 4.1: An example of a simple feedforward neural network. In this network, the input layer (in yellow) with input vector $\mathbf{x} = [x_1, x_2, x_3]^T$ are connected to a single output (in green) through two hidden layers, displayed in light blue. Weight matrices W_1, W_2 and W_3 links the neurons together. Moreover, f and O are predefined activation function for the hidden layers and the output, respectively, and $\mathbf{a}$ is the activation outputs from the hidden layers. The network also uses *bias*, b, to adjust the outputs to give a best possible fit. Here, b is the same everywhere, but different bias values for each neuron is also possible.

4.1.1 Activation functions

The activation functions in ANNs are inspired by how the rate of action potential that is released from an actual neuron cell varies in a biological neural network. Without activation functions, ANNs would just be linear mapping functions, as the only mathematical operations in the network would be dot-products between a weight matrix and an input vector. In order to make the network capable of solving more complex tasks, non-linearity in the form of activation functions are introduced.

In its simplest form, the activation function would be binary: the neuron is either activated or not activated, based on whether the input value to the neuron is below or exceeds a given threshold. However, for NNs applied to nontrivial problems, one would prefer to use a more complex non-linear activation function. In many cases, it is necessary to shift the activation function either to the right or to the left in order for the network to successfully learn. In these cases we add a constant term, the *bias*. This is analogous to adding the constant term b to a linear function $f(x) = ax + b$ to get a better fit to the data.

There exists a variety of different activation functions, based on which use it is designated for. For output layers, *sigmoid* and *softmax* functions are among the most common choices. These functions return values between 0 and 1 and are thus well-suited for classification tasks. The sigmoid function,

$$S(x) = \frac{1}{1 + e^{-x}}, \tag{4.2}$$

is usually applied in binary classification tasks, while softmax is more often used when there are more than two classes involved. It is defined as

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{4.3}$$

where K is the number of classes and the index i denotes the probability for class i.

For hidden layers, the *tanh* function is often used:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{4.4}$$

However, the arguably most popular activation function for hidden layers is called *rectified linear unit (ReLU)* (Hahnloser et al., 2000), and is defined as

$$f(x) = \max\{0, x\}. \tag{4.5}$$

A general problem with activation functions like the sigmoid and the tanh function is that they are saturating functions, meaning that they have finite upper and lower bounds. As we later will see, training NNs involves finding gradients in order to update the weights between the layers. The saturating functions will have a very small gradient everywhere but in the middle of their range, which makes the weight updates difficult when input values are outside of this narrow range. This is called the *vanishing gradient problem* (Hochreiter et al., 2001). ReLU is preferred as an activation function because it only saturates in one direction, and has a constant gradient in the other direction, in addition to being a simple function and thus computationally efficient. Because of these properties, ReLU in many cases reduces time until convergence in NNs (Krizhevsky et al., 2012).

There exists numerous activation functions based on the ReLU. A popular alternative is the *leaky ReLU* (Maas et al., 2013), defined as

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise,} \end{cases} \tag{4.6}$$

where α usually is equal to 0.01. This activation function prevents cases where the original ReLU would return gradients of value 0 and thus resulting in learning being stopped.

4.1.2 Learning process

Learning in NNs happens through iterative adjustment of the weights connecting the neurons, e.g. by using a technique called *backpropagation* (Rumelhart et al., 1988). The output of the network is compared to the labels of the data through some predefined cost function, C, with the intention of minimising the difference iteratively. Hence, to train a NN, a training dataset is necessary in order to teach the network what its output should look like. A commonly used cost function is the *quadratic cost*, defined as

$$C = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - a_i^L\right)^2, \tag{4.7}$$

where y is the labelled output, a^L is the prediction from the network and N is the number of individual training samples i. The outputs might be scalar, as here, but can also be vectors. To be used in backpropagation, the cost function must be able to be written as an average over cost functions C_i for individual training samples. For classification tasks,

the *cross-entropy* is a commonly used alternative to the quadratic cost, which we will also return to later in this thesis.

The difference between the labels from the training data and the predictions is called the *loss*, and is essentially the errors the network makes. This error is fed back through the layers, from last to first, and the corresponding weights are adjusted to reduce the error. After repeating this training cycle, called an *epoch*, for a sufficiently large number of times, the error usually converges to some constant state and the network can be said to have learned a certain target function.

More specifically, backpropagation evaluates the total derivative of the cost function as a product of the derivatives between every layer, from last to first, hence the term *backwards propagated error*. The derivative is in turn used in an optimisation procedure to minimise the cost function C. Since we wish to monitor how the weights influence the error, we work with the gradients of the cost function with respect to the weights, W.

In a feedforward NN, the only way a weight in a layer l affects the loss is through the effect it has on the next layer $l+1$. This effect is linear, and by letting $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ denote the gradient of the error of neuron j in layer l, we can compute the gradient of each layer recursively, starting from the last layer, L. For weights between layer $l-1$ and l, we have the vector of weights $W_j^l = (w_{jk}^l)_{k=1}^m$, where w_{jk}^l is the weight between the k-th node in layer $l-1$ and j-th node in l, and there is m nodes in layer $l-1$. We let

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L) \tag{4.8}$$

and

$$\delta^{l-1} = (f^{l-1})'(W^l)^T \delta^l, \tag{4.9}$$

where a_j^L is the j-th activation output and f is an activation function. We have here defined $z_j^l := (W_j^l)^T a^{l-1} + b_j^l$, where b_j^l is the bias of the j-th neuron in the l-th layer, n is the number of weights in $l-1$. By using e.g. the *chain rule*, it can be shown that

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \quad \text{and} \quad \frac{\partial C}{\partial b_j^l} = \delta_j^l. \tag{4.10}$$

The gradients are then used in an optimisation algorithm in order to minimise the cost, C, and thus *learn* i.e. improve the predictions on a given task.

Optimisation algorithms

Optimisation involves finding the maximum or minimum of a real function by a systematic and repeated procedure of choosing an allowed input and calculating the resulting function value. A well-known optimisation algorithm is the *gradient descent* (GD), originally proposed in 1847 (Lemaréchal, 2012). It is a simple, first-order iterative algorithm for finding a local minimum of a differentiable function.

The algorithm is based on the observation that for a defined and differentiable function $F(\mathbf{x})$ in a neighborhood of a point $\mathbf{a}$, $F(\mathbf{x})$ decreases fastest in the direction of the negative gradient in that point, $-\nabla F(\mathbf{a})$. Based on this, one can establish an iterative procedure to find the local minimum:

$$\mathbf{x} \leftarrow \mathbf{x} - \eta_k \nabla F(\mathbf{x}), \tag{4.11}$$

where η_k denotes the step size, or *learning rate*, at iteration k. Usually, the learning rate is kept constant.

An algorithm that is very popular in training deep learning models is *stochastic gradient descent* (SGD), a stochastic approximation of the gradient descent algorithm (Robbins and Monro, 1951). It is well-suited for problems involving large amounts of data, because of its computational efficiency. Instead of calculating the gradient of the entire dataset, it estimates the gradient through using a smaller subset of the data. For an objective function on the form

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w), \tag{4.12}$$

where n is the number of observations in a data set and Q_i is the value evaluated at the i-th observation. We pick a *mini-batch* $B_k \subseteq \{1, ..., n\}$, of the indexes of the observations with size $|B_k| = m \ll n$. For *true* SGD the size of this subset is 1. We can describe the iterative procedure at iteration k as

$$w \leftarrow w - \eta_k \frac{1}{m}\sum_{i \in B_k} \nabla Q_i(w), \tag{4.13}$$

where B_k is the sampled mini-batch at step k and m is the size of the subset. The full dataset is shuffled, and the m samples are drawn randomly. Using $m > 1$ helps reduce variance (since more data is used to compute the estimate). However, the larger the mini-batch is, the more computation time increases.

Optimisation algorithms for deep learning models is an active research area. One of the new and promising algorithms that have challenged algorithms like SGD is *Adam* (Kingma and Ba, 2014). Adam is an abbreviation for *Adaptive Moment Estimation*, and it utilises both averages of gradients as well as second moments of the gradients iteratively. For given parameters $w^{(i)}$ and loss function $L^{(i)}$ at step i, the parameter update is given by the following equations:

$$\begin{aligned} m^{(i+1)} &\leftarrow \beta_1 m^{(i)} + (1-\beta_1)\nabla_w L^{(i)} \\ v^{(i+1)} &\leftarrow \beta_2 v^{(i)} + (1-\beta_2)\left(\nabla_w L^{(i)}\right)^2, \end{aligned} \tag{4.14}$$

with m and v being first and second moment estimate vectors respectively, with initial values $m^{(0)} = v^{(0)} = 0$. Furthermore, β_1, $\beta_2 \in [0, 1)$ are forgetting factors (exponential decay rates for the moment estimates). The first and second moments are defined as

$$\begin{aligned} \hat{m} &= \frac{m^{(i+1)}}{1-\beta_1^{i+1}} \\ \hat{v} &= \frac{v^{(i+1)}}{1-\beta_2^{i+1}}, \end{aligned} \tag{4.15}$$

so that the weight updates will be

$$w^{(i+1)} \leftarrow w^{(i)} - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}, \tag{4.16}$$

where η is the learning rate and ϵ a small number to prevent dividing by zero.

4.1.3 Convolutional Neural Networks

One of the most important classes of NNs are *Convolutional Neural Networks* (CNNs) (LeCun et al. (1998); Krizhevsky et al. (2012)). CNNs are widely used in visual image analysis, natural language processing and recommender systems (Bhandare et al., 2016).

Receptive field

Convolutional networks are inspired by biological processes, i.e. how neurons are organised to resemble the system in the visual cortex of animals. Individual cortical neurons only respond to stimuli in a certain region of the visual field which leads to eventually splitting the image into several, smaller parts. This is mirrored in these NNs, where every neuron receives input from a restricted subarea, usually a square shaped area, of the previous layer. This is called *the receptive field* of a neuron. Applying this philosophy into CNNs is a part of the reason why these networks are well-suited for image processing tasks.

Convolutional layers

CNNs may consist of regular, fully connected layers (layers where neurons receive input from every single neuron in the previous layer), but they mainly consist of *convolutional layers*: layers where the neurons receive input from a restricted subarea of the previous layer (the receptive field), so that only the local relationships of the input are examined.

A convolutional layer, or a convolutional *filter*, consists of one or several *kernels*. A kernel is essentially a matrix consisting of a numbers forming a distinctive pattern, used to detect e.g. low-level features in a target image. A kernel for detecting edges in an image could be:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

During a regular forward pass in a CNN, the kernels are convolved (in practice a sliding dot product) across the width and height of the input. For a kernel K of shape $M \times N$, the generated output of the layer, a feature map F, would look like

$$F(i,j) = I(i,j) * K = \sum_{m=1}^{M} \sum_{n=1}^{N} K(m,n) I(i-m, j-n), \tag{4.17}$$

for each element of F and where I is the input image. The process is illustrated in Figure 4.2. As in ordinary NNs, we introduce non-linearity through activation functions. The output from the convolutions is passed through an activation function for each receptive field, i.e. a layer of activation functions, like the ReLU function.

The kernel weights are learned as in ordinary NNs - usually through the backpropagation algorithm, and results in the network learning filters that activate when it detects a certain type of features at a specific spatial position in the input. As one moves deeper into the network, the filters learn to detect more complex patterns as feature outputs from previous layers are combined.

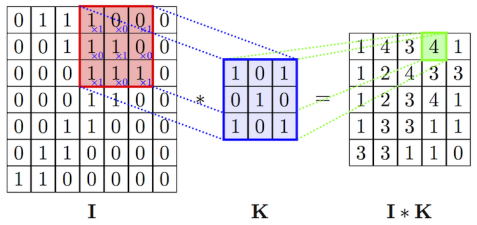


Figure 4.2: Example showing how a convolutional operation in a convolutional layer works. The input image I is here convolved with the kernel K to produce the output on the right. Figure reproduced from Hossain and Sajib (2019).

Utilising the receptive field in the layer structure helps reducing the number of parameters greatly, which facilitates building deeper networks (more hidden layers) without *overfitting*. Overfitting would mean producing output that is too similar to a particular set of (training) data so that prediction accuracy when facing unseen test data is greatly reduced. The reduced size of parameters also helps resolving the problem of vanishing or exploding gradients seen in ordinary NNs, as well as speeding up computations.

By using kernels, the layer is also able to reduce the dimensionality and compress the image. How much the spatial dimensions are reduced in the layer is controlled by several hyperparameters, e.g. *stride*, which controls how many pixels the filters jump when it slides around. In some cases it is convenient to pad the input volume with zeros on the border. This is called *padding*, and the padding hyperparameter controls the output spatial volume.

Pooling

Another frequent tool used in CNNs is called *pooling*. This is a method of down-sampling: lowering the resolution of the input, but keeping the most important structures. Intuitively, this is because the exact location of a certain feature is of less importance compared to the location relative to other features. This is useful in order to reduce the number of parameters in the network, and to reduce the risk of overfitting. It also makes the sample more robust to changes in the location of the feature the (convolutional) layer is detecting. Usually, the pooling layers are periodically placed between successive convolutional layers in a CNN.

There are some specified pooling operations, usually maximum or average. Arguably the most popular pooling technique is *max pooling*. Here, when the pooling operation is performed, the input is down-sampled by only picking the maximum value of the patch. A pooling layer is often added after a convolutional layer so that the pooling operation is down-sampling the output from the convolutional layer. This is illustrated in Figure 4.3.

By utilising this architecture, the different layers learns to recognise specific shapes or features in an image before merging all of these parts into a final representation. The information the network now has concerning the input can then be further processed to produce a desired output depending on the purpose of the network.

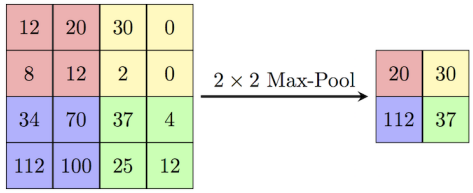


Figure 4.3: Example showing how max pooling works.

4.1.4 U-Net

A recent application of the CNN philosophy that has shown to perform well on image segementation tasks, is the *U-Net* (Ronneberger et al., 2015). It was initially developed to perform biomedical segmentation tasks, but has shown to exceed regular CNNs in a variety of segmentation tasks due to its ability to convert an image into a feature vector and from this, reconstruct a segmented image.

The architecture of the U-Net consists of three coherent sections. A contraction part (called the *encoder*), an expansion part (the *decoder*) and a bottleneck connecting the two. In the contraction phase the spatial information of an input image is reduced while its feature information is increased through repeated convolutional layers (each followed by a ReLU and a max pooling operation). Then, in the expansive phase, the spatial and feature information is combined through a series of upsampling convolutions to reconstruct the segmented image. The architecture of the U-Net is shown in Figure 4.4.

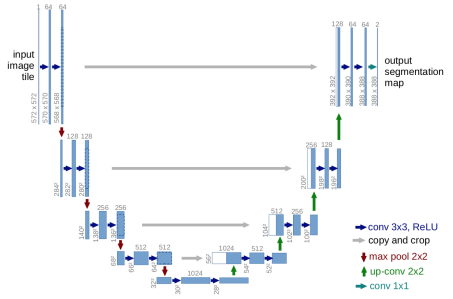


Figure 4.4: Architecture of the UNet. It consists of two main parts: a contracting path and an expansive path. In the contractive phase, spatial image information is reduced while feature information is increased. In the expansive phase, feature and spatial information is combined to propagate context information to a higher resolution. Figure reproduced from Ronneberger et al. (2015).

4.2 Introduction to GANs

GANs are a branch of deep learning where the main objective is to map from latent space to a particular data distribution of interest, often to generate some synthetic output, be it images of people or art. More specifically, GANs are models combining a generative and a discriminative approach to form an *adversarial* method, where the generative and the discriminative methods oppose each other. It consists of a generative model whose aim is to generate candidates that is as close to the targeted data distribution as possible, while a discriminative model evaluates these and tries to learn which samples are real and which are generated. When the generative model thus tries to cause the discriminative model to make an incorrect prediction, it generates adversarial examples.

An analogue to this interaction could be the generator working as a con artist trying to produce fake currency, while the discriminator would be a police investigator trying to separate fake currency made by the con artist from real currency (Goodfellow, 2016). As both the investigator and the artist learn from experience, their abilities to produce and uncover fake currency increase. The competition between the two continues until the fake currency is indistinguishable from the real. The GAN will then have reached convergence. Figure 4.5 illustrates the architecture of such a network.

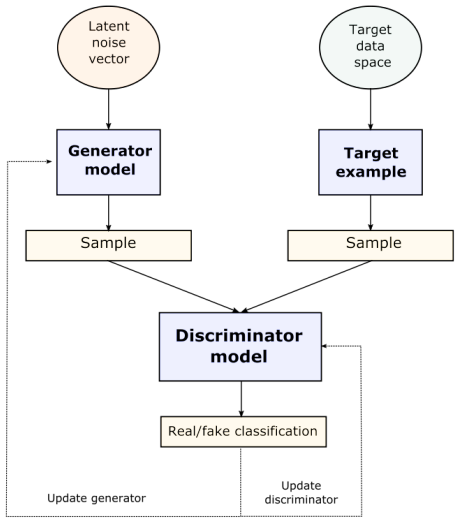


Figure 4.5: A high-level sketch showing the architecture of a standard GAN. The Generator generates images as close to the target as possible, while the Discriminator tries to separate generated images from the real training samples.

4.2.1 Definition and mathematical description

As research in the field has progressed, numerous versions of GANs has been developed. What they have in common is that they are extensions or further developments and innovations of the fundamental GAN model based on the work of Goodfellow et al. (2014).

Framework

We let $\mathcal{Z} \subseteq \mathbb{R}^\ell$ and $\mathcal{X} \subseteq \mathbb{R}^d$ be two ambient data spaces. Further, we let $p_{\mathbf{z}}$ be a prior distribution over $\mathcal{Z}$, typically a uniform $U(0,1)$ or Gaussian distribution $N(\mu, \sigma^2)$, and p_{data} be the distribution of real data points over $\mathcal{X}$. We also let $\mathbf{z} \sim p_{\mathbf{z}}$ be a latent noise vector. This latent vector $\mathbf{z}$ is mapped to form candidate distributions working as samples via the generator G, which induces its own distribution p_{G} from the input so that if $\mathbf{z} \sim p_{\mathbf{z}}$ then p_{G} is the distribution such that $G(\mathbf{z}) \sim p_{\text{G}}$.

Ideally, p_{G} converges to a good estimator of p_{data}, the distribution of the target data so that $G : \mathcal{Z} \to \mathcal{X}$. The discriminator D takes the sample as input and outputs the probability that the sample belongs to p_{data}, or in other words $D : \mathcal{X} \to [0,1]$.

In this framework, both the generator and the discriminator are differentiable functions, $G(\mathbf{z}; \boldsymbol{\theta}_g)$ and $D(\mathbf{x}; \boldsymbol{\theta}_d)$. The vectors $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_g$ represent the parameters of the functions D and G, respectively. For now, the functions are defined to be multilayer perceptrons; feedforward neural networks as described in Section 4.1, but in more recent years, deep convolutional neural networks are more often used (Radford et al., 2015).

Discriminator cost function

The discriminator wants to maximise the probability of assigning the correct label to both the generated samples and the examples from the training set. The common way to define such a cost function $\mathcal{L}_D$ is to use binary cross-entropy, which is also used here. Cross-entropy is defined as

$$H(p, q) = -\sum_{x \in \mathcal{P}} p(x) \log q(x), \tag{4.18}$$

for discrete probability distributions p and q defined over the same support $\mathcal{P}$.

Cross-entropy is widely used in ML loss functions as a measure of the dissimilarity between the true distribution p_i and predicted distribution q_i for an outcome i. Thus, in a binary setting with a sigmoidal probability curve we have

$$q_{y=1} = \hat{y} \quad \text{and} \quad q_{y=0} = 1 - \hat{y},$$

for outcomes $y = 1$ or $y = 0$. The cross-entropy between p and q can then be expressed as

$$H(p, q) = -\sum_i p_i \log(q_i) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) = H(\hat{y}, y). \tag{4.19}$$

In the context of *vanilla* GANs (ordinary GANs) we use two minibatches of data, one from the target dataset, i.e. with label 1, and one from the generated samples, with label 0 (Goodfellow, 2016). A minibatch here is described about as before: it can be said to be a

subset of a dataset, so that for a training set $\mathcal{S}$ we have $\mathcal{B} \subsetneq \mathcal{S}$, where the minibatch $\mathcal{B}$ has a size larger than 1.

In the following short derivation we denote by $D(\cdot)$ and $G(\cdot)$ the discriminator and generator functions, omitting their parameters for ease of notation. Now, let us consider single data points $x \sim p_{\text{data}}(x)$ and $z \sim p_z(z)$. If we let $y = 1$ denote the label for data coming from $p_{\text{data}}(x)$ and $\hat{y} = D(x)$ we obtain the following cross-entropy:

$$H\left(D(x), 1\right) = -\log\left(D(x)\right). \tag{4.20}$$

Furthermore, for the second batch, letting $y = 0$ denote data coming from $p_z(z)$ and $\hat{y} = D(G(z))$ yields

$$H\left(D(G(z), 0)\right) = -\log\left(1 - D(G(z))\right). \tag{4.21}$$

For one data point x_i and its belonging label y_i we can then write (with a slight abuse of notation):

$$H\left(D, (x_i, y_i)\right) = -y_i \log\left(D(x_i)\right) - (1 - y_i)\log\left(1 - D(x_i)\right). \tag{4.22}$$

By reviewing the total performance of the discriminator, i.e. for all points $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$ and $\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})$, we sum the every individual entropy so that we end up with:

$$H\left(D, (x_k, y_k)_{k=1}^N\right) = -\sum_{k=1}^{N} y_k \log\left(D(x_k)\right) - \sum_{k=1}^{N}(1 - y_k)\log\left(1 - D(x_k)\right), \tag{4.23}$$

where N is the number of samples, and where we know that $x_i = G(z_i)$ when $z_i \sim p_z$. If we let $N \to \infty$ we can replace the sums with expectations, using that

$$\mathbb{E}[\mathbb{X}] = \sum_{i=1}^{\infty} x_i p_i, \tag{4.24}$$

for outcomes $(x_1, x_2, ...)$ with belonging probabilities $(p_1, p_2, ...)$. For a continuous probability density function we trade the sum with an integral.

If we also assume that the labels y_i are evenly split in 1 and 0, and letting $\mathcal{L}_D$ denote the cross-entropy of the discriminator (so that we also end up with using the parameters $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_g$), we arrive at

$$\mathcal{L}_D(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D(\mathbf{x}; \boldsymbol{\theta}_d) - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_z} \log\left(1 - D(G(\mathbf{z}; \boldsymbol{\theta}_g); \boldsymbol{\theta}_d)\right). \tag{4.25}$$

Here, $D(\mathbf{x}; \cdot)$ is the discriminator's estimate of the probability that the real data distribution $\mathbf{x}$ is real, and $D(G(\mathbf{z}; \cdot); \cdot)$ is the estimate that a generated distribution - thus fake - is real.

Hence, the performance of the discriminator depends on both the functions D and G and thus on the parameters $\boldsymbol{\theta}_d$ and $\boldsymbol{\theta}_g$ when the networks are NNs or CNNs. This cost function is common for most discriminators used in GANs. The discriminator can only control its own parameters and similarly for the generator. This makes it possible to describe the problem as a game and not only an optimisation problem.

Generator cost function

In order to play the game between the discriminator and the generator, we also need a cost function for the generator. This function varies more across the different versions of these networks, but the simplest one would be

$$\mathcal{L}_G = -\mathcal{L}_D, \tag{4.26}$$

so that the discriminator and the generator are parts of a zero-sum game, a balance between players where they try to minimise their own maximum loss, and maximise their own minimum gain (see Maschler et al. (2018) and Section 4.2.2 in this thesis for more on this). The total gains of the players will directly sum to zero. With $\mathcal{L}_G$ being directly tied to $\mathcal{L}_D$ we can summarise the total value function using the discriminator's performance so that

$$V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = -\mathcal{L}_D(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d), \tag{4.27}$$

and since games like this zero-sum one are defined by using minimising and maximising operations, so-called *minimax* rules, we find the solution as

$$\arg\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d). \tag{4.28}$$

The generator wants to minimise the probability of the discriminator correctly classifying the generated images as being fake, and the discriminator wants to maximise the probability of correctly classifying a given input image. A solution to this expression is the so-called *Nash equilibrium* (Nash, 1951), named after the famous mathematician John F. Nash, widely used in game theory (Von Neumann et al., 2007) which these zero-sum games fall under.

As game theory play a big part in the GAN philosophy, utilising it can thus play an important part in the training of these networks. We therefore present a brief introduction on the subject in Section 4.2.2.

Theoretical results

The choice of generator cost function we use here does not work very well in practice as the assumptions made here rely on convexity, while highly non-convex functions are often used. However, the cost function is useful when it comes to theoretical analysis of the capabilities of GANs. In practice, we often have to settle with approximations of these results.

It can be shown that learning in this particular zero-sum game is similar to minimising the *Jensen-Shannon Divergence* (JSD), a finite and symmetric, altered version of the *Kullback-Leibler Divergence* (KLD) (Kullback and Leibler, 1951), between model distribution and the data. For two probability measures η and μ, where η is absolutely continuous w.r.t. μ, the KLD is defined as

$$\mathrm{KL}(\eta||\mu) = \mathbb{E}_{x\sim\eta}\left[\log\frac{\eta(x)}{\mu(x)}\right], \tag{4.29}$$

where the "||" operator indicates divergence, i.e. how η differs from μ.

In the following derivation we assume that the generator is fixed, and we try to find the optimal discriminator D_G^* by finding

$$\max_D V(G, D). \tag{4.30}$$

By considering the integral version of Equation 4.25, and simplifying by writing $D(\mathbf{x}; \boldsymbol{\theta}_d)$ and $G(\mathbf{z}; \boldsymbol{\theta}_g)$ as $D(\mathbf{x})$ and $G(\mathbf{z})$, we get

$$V(G, D) = \int_{\mathcal{X}} p_{\text{data}}(\mathbf{x}) \log\left(D(\mathbf{x})\right) d\mathbf{x} + \int_{\mathcal{Z}} p_{\mathbf{z}}(\mathbf{z}) \log\left(1 - D(G(\mathbf{z}))\right) d\mathbf{z} \tag{4.31}$$

$$= \int_{\mathcal{X}} \left[p_{\text{data}}(\mathbf{x}) \log\left(D(\mathbf{x})\right) + p_{\text{G}}(\mathbf{x}) \log\left(1 - D(\mathbf{x})\right)\right] d\mathbf{x}, \tag{4.32}$$

where the transition in the last line is possible due to the *Radon-Nikodym theorem* (Nikodym, 1930).

Furthermore, it can easily be shown that for any $(a, b) \in \mathbb{R}^2 \setminus \{(0, 0)\}$ the function $y \mapsto a \log y + b \log(1 - y)$ reaches its maximum in $[0, 1]$ at $\frac{a}{a+b}$. Thus, V has its maximum in $[0, 1]$ at $\frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})}$, which is then $D_G^*(\mathbf{x})$.

We now turn to find the optimal generator, G^*. The optimal generator minimising the loss function occurs when $D = D_G^*$. Hence,

$$G^* = \arg\min_G V(D_G^*, G). \tag{4.33}$$

Subtracting the optimal discriminator $D = D_G^*$ into Equation 4.32 yields

$$G^* = \arg\min_G \left\{ \int_{\mathcal{X}} \left[p_{\text{data}}(\mathbf{x}) \log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})} + p_{\text{G}}(\mathbf{x}) \log \frac{p_{\text{G}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})}\right] d\mathbf{x} \right\}. \tag{4.34}$$

We now manipulate the above equation by adding and subtracting the terms $\log(2) p_{\text{data}}(\mathbf{x})$ and $\log(2) p_{\text{G}}(\mathbf{x})$. We end up with

$$G^* = \arg\min_G \left\{ \int_{\mathcal{X}} \Big[\big(\log(2) - \log(2)\big) p_{\text{data}}(\mathbf{x}) + p_{\text{data}}(\mathbf{x}) \log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})} + \big(\log(2) - \log(2)\big) p_{\text{G}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x}) \log \frac{p_{\text{G}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})} \Big] d\mathbf{x} \right\} \tag{4.35}$$

which, after collecting and reorganising terms can be written as minimising

$$\begin{aligned} G = &- \log 2 \int_{\mathcal{X}} \left(p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})\right) d\mathbf{x} \\ &+ \int_{\mathcal{X}} p_{\text{data}}(\mathbf{x}) \left[\log 2 + \log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})}\right] d\mathbf{x} \\ &+ \int_{\mathcal{X}} p_{\text{G}}(\mathbf{x}) \left[\log 2 + \log \frac{p_{\text{G}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_{\text{G}}(\mathbf{x})}\right] d\mathbf{x}. \end{aligned} \tag{4.36}$$

The first term in the above equation equals $-\log 4$, and by noting that the last two terms can be slightly manipulated (using that $\log(ab) = \log a + \log b$) and write them as expectations, we get

$$G^* = \arg\min_G \left\{ -\log 4 + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_{\text{G}}(\mathbf{x})}{2}} \right] + \mathbb{E}_{\mathbf{x}\sim p_{\text{G}}} \left[\log \frac{p_{\text{G}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x})+p_{\text{G}}(\mathbf{x})}{2}} \right] \right\}. \tag{4.37}$$

Moreover, we make use of the definition of the KLD to write

$$G^* = \arg\min_G \left\{ -\log 4 + \text{KL}\left(p_{\text{data}} \middle|\middle| \frac{p_{\text{data}} + p_{\text{G}}}{2} \right) + \text{KL}\left(p_{\text{G}} \middle|\middle| \frac{p_{\text{data}} + p_{\text{G}}}{2} \right) \right\}, \tag{4.38}$$

and finally

$$G^* = \arg\min_G \left\{ -\log 4 + 2\text{JSD}(p_{\text{data}}||p_{\text{G}}) \right\}, \tag{4.39}$$

since we by definition of JSD have the relation

$$\text{JSD}(P||Q) = \frac{1}{2}\text{KL}(P||M) + \frac{1}{2}\text{KL}(Q||M), \tag{4.40}$$

with $M = \frac{P+Q}{2}$.

The JSD$(\cdot)$ is always non-negative. Additionally, it is zero only when the distributions are equal. Hence, the optimal generator G^* is found when the generator replicates the distribution from the data perfectly and $p_{\text{data}} = p_{\text{G}}$.

Thus, it can be shown that if the players' policies are updated in function space, the equilibrium state will be reached. If we assume that the outputs from G comes from p_{G} we can again write

$$V(G, D) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim p_{\text{G}}} [\log (1 - D_G(\mathbf{x}))] = U(p_{\text{G}}, D). \tag{4.41}$$

$U(p_{\text{G}}, D)$ can be proven to be convex in p_{G} by linearity of expectations, which also implies that the optimisation problem converges. But as mentioned in the beginning of this section, these results rely on convexity. In reality, this is not the case as deep neural nets, which are most often used in GANs, are highly non-convex. Therefore, many alternative solutions are proposed to achieve this, e.g. more complex loss functions, but in general this makes GANs hard to train and unstable while doing so.

Additionally, when solving specific tasks it is necessary to build an architecture that suits the problem so that the model is as fit as possible for solving the task. Therefore, there exist numerous different successors that all are based on the same underlying concept of GANs.

4.2.2 Game theory

We next provide a bit more depth and shed some light on the game theory aspect of the GANs. Game theory can be used to apply a more direct *adversarial* approach to explain GAN training and to overcome some of the present challenges related to training them. In GANs, a game is played between the generator and the discriminator. In a *strategic* two-player game (like this), each player chooses their plan of action conclusively, and simultaneously. It has the following definition (Osborne and Rubinstein, 1994):

Definition 4.2.1. A strategic two-player game $\langle (A_i), (u_i) \rangle_{i=1,2}$, for players $i = 1, 2$ consist of

- a non-empty set A_i including the actions available for player i.
- an utility function (a payoff) $u_i : A \to \mathbb{R}$, where $A = A_1 \times A_2$.

If the set A_i is finite we say that the game is finite.

As previously mentioned, the theoretical setup discussed in the previous section suggests that the generator and the discriminator of a GAN plays a minimax game, which for a two-player strategic game can be defined as

$$\overline{u_i} = \min_{a_{-i} \in A_i} \max_{a_i \in A_i} u_i(a_i, a_{-i}), \tag{4.42}$$

where $\overline{u_i}$ is the largest value the player can be sure to get when the actions of the other player (a_{-i}) is known. These games can be zero-sum games if, given that it is a strategic two-player game with $\langle (A_i), (u_i) \rangle_{i=1,2}$, we have $u_1 = -u_2$. Zero-sum games are thus games where the gain of utility of a player is exactly balanced by the loss of its opponent, and vice versa. GANs are zero-sum games in this manner because either the generator "wins" by tricking the discriminator, or the discriminator "wins" by correctly identifying the generated image leading to the generator "losing".

Theoretically, a game like this can reach a point where neither player can gain anything by deviating from their strategy, given that the players know the optimal strategy of their opponent. This situation is known as a Nash equilibrium, and can, in a strategic two-player game, be formally defined as:

Definition 4.2.2. A Nash equilibrium of a strategic two-player game $\langle (A_i), (u_i) \rangle_{i=1,2}$, is a profile $a^* \in A$ of actions with the property that for $i = 1, 2$ we have

$$u_i(a^*_{-i}, a^*_i) \geq u_i(a^*_{-i}, a_i) \quad \forall a_i \in A_i. \tag{4.43}$$

In the GAN setting discussed in this thesis so far, given that we ought to pick generators and discriminators from the spaces $\mathcal{G}$ and $\mathcal{D}$, the Nash equilibria (G^*, D^*) for every $G \in \mathcal{G}$ and every $D \in \mathcal{D}$ (or rather the tuples $(\boldsymbol{\theta}_g^*, \boldsymbol{\theta}_d^*)$) are such that

$$V(G^*, D) \leq V(G^*, D^*) \leq V(G, D^*) \tag{4.44}$$

for a value function V, is satisfied (Farnia and Ozdaglar, 2020). By exploiting this adversarial nature of GANs, new approaches to train them can be developed, possibly exceeding the traditional optimisation algorithms. One example of such an approach is to use a *stochastic forward-backward algorithm* (Franci and Grammatico, 2020). However, this is still an open and highly active research problem.

4.2.3 Training GANs

Training GANs involves balancing the training of two separate networks at the same time, making sure that neither of the networks learn too fast, or at the expense of the other network. In practice, this can be very difficult to achieve which also results in GANs being hard to train properly. This has also lead to an extensive search in finding objective functions and losses that stabilise and improve performance.

Nash equilibrium

In GANs, a Nash equilibrium is − as we have discussed − reached when the loss functions of the generator and the discriminator cannot be lowered and convergence is reached. Theoretically, this occurs when the generator is able to reproduce the true data distribution and the discriminator is able to successfully classify all samples.

In practice, getting to this state is difficult as the strategy of the opponent usually is unknown and variable. Even when an equilibrium theoretically exists, simple optimisation algorithms may struggle to find the optimum. This can be illustrated by this simple example, using plain gradient descent:

We consider a minimax game with two players both controlling scalar values (replacing the generator and discriminator), so that the value function V is

$$V(x, y) = xy, \tag{4.45}$$

where the player controlling x wishes to minimise and the player controlling y wishes to maximise (Goodfellow, 2016). It is clear that a Nash equilibrium exits, for $x = y = 0$. Suppose we try to learn this equilibrium using a simple optimisation algorithm like GD (see Equation 4.11). To simplify the problem, we let GD be a continuous time process with an infinitesimal learning rate. Then the trajectory of that of GD for this problem can be defined as the following system of equations:

$$\begin{aligned} \frac{dx}{dt} &= -\frac{\partial}{\partial x} V\left(x(t), y(t)\right) \\ \frac{dy}{dt} &= \frac{\partial}{\partial y} V\left(x(t), y(t)\right), \end{aligned} \tag{4.46}$$

which evaluates to

$$\begin{aligned} \frac{dx}{dt} &= -y(t) \\ \frac{dy}{dt} &= x(t). \end{aligned} \tag{4.47}$$

If we now differentiate the latter expression in Equation 4.47, we get

$$\frac{d^2y}{dt^2} = \frac{dx}{dt} = -y(t). \tag{4.48}$$

Solving this differential equation finally yields

$$\begin{aligned} x(t) &= x_0 \cos(t) - y_0 \sin(t) \\ y(t) &= x_0 \sin(t) + y_0 \cos(t), \end{aligned} \tag{4.49}$$

for initial values x_0, y_0. If we plot this trajectory, as in Figure 4.6, we observe that it will forever orbit the equilibrium and never reach it. Choosing a larger learning rate would lead to a trajectory spiralling outwards, away from the equilibrium.

This simple example illustrates that even for some trivial games, certain optimisation algorithms might not converge. In a game of two players, optimisation would mean

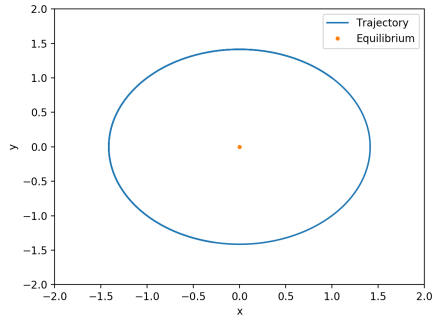


Figure 4.6: Plot showing the trajectory of simple GD for the value function $V(x, y) = xy$. It will orbit the equilibrium solution at $x = y = 0$ at a constant radius.

”downhill progress”. But even if each player moves downhill on their own update, the same update might move the other player uphill. Sometimes an equilibrium is reached for both players, but often they just repeatedly undo their opponent's progress. This illustrates the scenario we face. For GANs, there exist no theoretical prediction whether convergence is guaranteed using simple optimisation algorithms. Developing a fitting theoretical foundation and developing optimisation algorithms guaranteed to converge remains an open field of research.

Mode collapse

One of the many challenges in the learning process of GANs is avoiding *mode collapse* or *the Helvetica scenario* (Goodfellow et al., 2014). A mode collapse is occurring when the output of the model is the same no matter what the input looks like. The task of the generator is to produce the output that is most likely to be real from the discriminator's point of view. Usually, when the generator tries to produce the same small subset of outputs, the discriminator would learn to always reject that output. But if the discriminator gets stuck in a local minima there is a chance that it does not discover this optimal strategy and accepts the output from the generator. The generator produces a large imbalance of modes of the target distribution which ultimately deteriorates its capabilities to find other plausible modes. Both networks are thus over-optimised to exploit short-term opponent weaknesses to try to win the game. The game turns into a ever-lasting swirl that does not converge.

An attempt to fix this issue is to introduce minibatch features (Salimans et al., 2016). This allows the discriminator to compare an example to a minibatch of generated samples and a minibatch of real samples. By comparing the example to the minibatches, the discriminator can decide if the example is unusually similar to generated samples. This method is shown to produce excellent results.

A different attempt to overcome the issue is through *unrolled GANs* (Metz et al., 2016). The idea is to build a computational graph that describes k steps of learning in the discriminator and backpropagate through all these steps to calculate the gradient of the generator.

This helps stabilising the gradient and helps reducing mode collapse, even for small k. There are also numerous other versions of GANs whose main goal is to reduce mode collapse.

Vanishing gradient

Another major problem with training GAN models is the occurrence of vanishing gradients. The problem is particularly present in the vanilla GAN setup. It can be shown (Arjovsky and Bottou, 2017) that, for the vanilla GAN,

$$\lim_{||D-D^*||\to 0} \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}} \left[\log\left(1 - D(g_{\boldsymbol{\theta}}(z))\right)\right] = 0. \tag{4.50}$$

This shows that as the discriminator improves, it successfully rejects generator samples with high confidence, the gradient of the generator vanishes and training process slows down before it eventually stops. To overcome this, one would have to be extremely precise in the amount the discriminator is trained for and carefully interplay the improvements of the generator and the discriminator or use another, more robust gradient step for the generator. The latter is the most common choice.

4.3 CycleGAN

In this project, we will study one of the many successors of the original GAN model, namely the *cycleGAN* (Zhu et al., 2017). The model was proposed as a generalisation of the *pix2pix* model, where image-to-image translation is conducted using conditional GANs (Isola et al., 2017). In the *pix2pix* model, the generator and discriminator use structured data to learn the distribution of the desired output in order to do the mapping. Thus, this requires supervised training with pairs of images $\{x_i, y_i\}_{i=1}^N$ as dataset.

However, in real life paired datasets like this seldom exists without extensive preprocessing. Developing a model which is not dependent on structured training of this sort is therefore highly attractive. This is something *cycleGAN* addresses, making the translation completely independent of data that is paired. The model is taking one more step towards an unsupervised method, using only images from the two domains it wishes to map between, but where any direct correspondence between the domains is absent. This means that if we were to translate horses to zebras, we train the model with images with one thing in common: they would contain either a horse or a zebra. The model then learns the characteristics of horses and the characteristics of zebras, and − given an image of a horse − identify the horse and replace it with a zebra with the identical shape as that of the horse.

The general design of the model also avoids having to rely on task-specific and predefined similarity-functions, nor assuming that the input and output of the model must lie in the same low-dimensional embedding space. This makes the model a versatile, all-purpose model, suitable for a variety of tasks.

Figure 4.7 shows the architecture of the cycleGAN and how the two GANs map input between two domains.

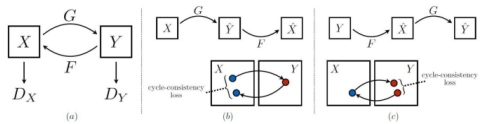


Figure 4.7: The *cycleGAN* architecture. The model consists of two separate GANs, mapping images from opposite domains. Figure reproduced from Wang and Deng (2018).

4.3.1 Definition and mathematical description

The model is fundamentally based on *cycle-consistency*: if one performs a mapping $\{G : \mathbf{x} \rightarrow \mathbf{y}\}$, and then a mapping in the opposite direction $\{F : \mathbf{y} \rightarrow \mathbf{x}\}$ one is back at where one started. We thus ideally want $\mathbf{x} \rightarrow G(\mathbf{x}) \rightarrow F(G(\mathbf{x})) \rightarrow \mathbf{x}$, and the other way around. This can be utilised to form a cycle-consistent loss, where $||\mathbf{x} - F(G(\mathbf{x}))||_1$ is the forward-consistent loss and $||\mathbf{y} - G(F(\mathbf{y}))||_1$ is the backward-consistent loss, and where $|| \cdot ||_p$ is the p-norm, defined as

$$||x||_p := \left(\sum_{i \in \mathbb{N}} |x_i|^p \right)^{1/p}. \tag{4.51}$$

Optimising this loss will help push F and G to be consistent with each other. We define two domains of images X and Y, with training data $\{\mathbf{x}_i\}_{i=1}^N \in X$ and $\{\mathbf{y}_i\}_{i=1}^N \in Y$ respectively. Furthermore, we denote their common data distribution as $x \sim p_{\text{data}}(\mathbf{x})$ and $y \sim p_{\text{data}}(\mathbf{y})$. The model itself will consist of two separate GANs. We introduce two generators G and F as previously defined, performing the mappings $\{G : X \rightarrow Y\}$ and $\{F : Y \rightarrow X\}$. We also introduce two discriminators D_Y and D_X, where D_X tries to distinguish between images $\{\mathbf{x}\}$ and $F(\mathbf{y})$ and, similarly, D_Y between images $\{\mathbf{y}\}$ and $G(\mathbf{x})$. In theory, the two GANs presented so far will have their own adversarial loss, similar to the loss presented in the GAN section. So for the generator G and its discriminator D_Y we get:

$$\mathcal{L}_{\text{Adv}}(G, D_Y) = \mathbb{E}_{\mathbf{y} \sim p_{\text{data}(\mathbf{y})}} \left[\log D_Y(\mathbf{y}) \right] + \mathbb{E}_{\mathbf{x} \sim p_{\text{data}(\mathbf{x})}} \left[\log \left(1 - D_Y(G(\mathbf{x})) \right) \right], \tag{4.52}$$

and similarly for F and D_X. But as we have previously discussed, the theoretical loss functions are of little practical use. Instead, the authors of the original cycleGAN paper apply a least squares (LS) loss, based on the *LSGAN* implementation by Mao et al. (2017). In general, a LS loss can be defined as $||\mathbf{o} - \hat{\mathbf{o}}||_2^2$, for labels $\mathbf{o}$ and predictions $\hat{\mathbf{o}}$. The loss used in the LSGAN paper (and thus also here), is described as

$$\mathcal{L}_{\text{LSGAN}}(G, D_Y) = \mathbb{E}_{\mathbf{y} \sim p_{\text{data}(\mathbf{y})}} \left[\left(D_Y(\mathbf{y}) - 1 \right)^2 \right] + \mathbb{E}_{\mathbf{x} \sim p_{\text{data}(\mathbf{x})}} \left[D_Y(G(\mathbf{x}))^2 \right], \tag{4.53}$$

and similar for the opposite translation. This loss provides more stable training and also increases output quality (Zhu et al., 2017). In addition, we will also use the cycle-consistent loss to measure the performance of the complete model. Translating images from one domain to another using unpaired data is an under-constrained problem. The cycle-consistent

loss helps reducing the size of the space of possible mapping functions, as it requires e.g. $F(G(\mathbf{x})) \approx \mathbf{x}$ and $G(F(\mathbf{y})) \approx \mathbf{y}$, and is a crucial part of the cycleGAN structure. The cyclic loss is defined as

$$\mathcal{L}_{\text{Cyc}}(G, F) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[||F(G(\mathbf{x})) - \mathbf{x}||_1\right] + \mathbb{E}_{\mathbf{y}\sim p_{\text{data}}(\mathbf{y})}\left[||G(F(\mathbf{y})) - \mathbf{y}||_1\right]. \quad (4.54)$$

Lastly, *identity loss* is used to preserve colour composition between the translations so that the generators are heavier constrained when choosing the tint of its generated images. This loss is meant to guide the generators to perform a near-identity mapping of the target domain. Here, it can be seen as not making too drastically alterations to model input. The identity loss is defined as

$$\mathcal{L}_I(G, F) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{y})}\left[||G(\mathbf{y}) - \mathbf{y}||_1\right] + \mathbb{E}_{\mathbf{y}\sim p_{\text{data}}(\mathbf{y})}\left[||F(\mathbf{x}) - \mathbf{x}||_1\right]. \quad (4.55)$$

The total objective for the loss of the cycleGAN model will thus be

$$\mathcal{L}_{\text{cycleGAN}}(G, F, D_X, D_Y) = \mathcal{L}_{\text{LSGAN}}(G, D_Y) + \mathcal{L}_{\text{LSGAN}}(F, D_X) + \lambda_C \mathcal{L}_{\text{Cyc}}(G, F) + \lambda_I \mathcal{L}_I(G, F), \quad (4.56)$$

where λ_C, λ_I are penalising weights. Since we can view this system of GANs as a GAN in itself, we seek to find the solution of the following minimax problem:

$$\arg\min_{G,F} \max_{D_X, D_Y} \mathcal{L}_{\text{cycleGAN}}(G, F, D_X, D_Y). \quad (4.57)$$

We then opt to train the model by optimising the loss function over the parameters of the generators and discriminators to find $(\boldsymbol{\theta}_g^*, \boldsymbol{\theta}_{d_Y}^*)$ and $(\boldsymbol{\theta}_f^*, \boldsymbol{\theta}_{d_X}^*)$.

4.3.2 CycleGAN algorithm

To better understand how the cycleGAN works, we include a short block of pseudocode explaining the training procedure of the basic model.

Algorithm 2: Training procedure for a basic cycleGAN model in pseudocode.

Result: Returns a cycleGAN model trained for a given number of epochs.
Initialisation: randomisation of weights;
Sample minibatch of samples $\{x^{(1)}, ..., x^{(m)}\}$ from domain X;
Sample minibatch of samples $\{y^{(1)}, ..., y^{(m)}\}$ from domain Y;
for *number of epochs* **do**
  for *each* $(x^{(i)}, y^{(i)})$ *in minibatch* **do**
    Optimise discriminator loss functions

$$\mathcal{L}_{\text{real}}^{(D)} = (D_X(x^{(i)}) - 1)^2 + (D_Y(y^{(i)}) - 1)^2$$

    and

$$\mathcal{L}_{\text{fake}}^{(D)} = \Big(D_Y(G(x^{(i)}))\Big)^2 + \Big(D_X(F(y^{(i)}))\Big)^2.$$

    Update the discriminators;
    Optimise generator loss functions

$$\mathcal{L}^{(F)} = \Big(D_X(F(y^{(i)})) - 1\Big)^2 + \lambda_C \mathcal{L}_{\text{Cyc}}^{(Y \to X \to Y)} + \lambda_I \mathcal{L}_{\text{I}}^{(Y \to X \to Y)}$$

    and

$$\mathcal{L}^{(G)} = \Big(D_Y(G(x^{(i)})) - 1\Big)^2 + \lambda_C \mathcal{L}_{\text{Cyc}}^{(X \to Y \to X)} + \lambda_I \mathcal{L}_{\text{I}}^{(X \to Y \to X)}.$$

    Update the generators;
  end
end

The $\mathcal{L}_{\text{Cyc}}$ is computed as described in the previous section, i.e. with

$$\mathcal{L}_{\text{Cyc}}^{(X \to Y \to X)} = ||y^{(i)} - G(F(y^{(i)}))||_1 \tag{4.58}$$

for the $X \to Y \to X$ translation, while the $\mathcal{L}_{\text{I}}$ is computed as

$$\mathcal{L}_{\text{I}}^{(X \to Y \to X)} = ||y^{(i)} - G(y^{(i)})||_1, \tag{4.59}$$

for the same translation. The opposite translation is analogous in both cases. To optimise the losses we use an optimisation algorithm like Adam or SGD.

4.4 Model architecture

For the experiments conducted later in this thesis we use a cycleGAN model similar to the one described previously in this chapter. The generators used are based on the U-Net

architecture (see Section 4.1.4). The idea here is that the design of the network is fit and able to capture the seismic features and reproduce them accurately. The encoder of the U-Net consists of 9 convolutional layers with ReLU as activation function, and instance normalisation (Ulyanov et al. (2016)): a normalisation procedure normalising across the channels of each training image. The decoder is built by 7 transposed convolutional layers, instance normalisation, *dropout* (a regularisation technique for reducing overfitting in NNs) applied to the three first layers, and leakyReLU activation function. Additionally, there are skip connections between the encoder and decoder. Skip architecture skips some layers in the network. Exactly why this works is still unclear, but it has shown to improve network performance (He et al., 2016).

The discriminators in the model we use are based on so-called *PatchGANs* (Li and Wand, 2016). A patchGAN works by dividing the actual image into patches of size equal to its receptive field, similar to how CNNs operate. Then, it models the image as a *Markov Random Field* (Kindermann, 1980), where pixels separated by more than a patch are assumed to be independent. It will classify each of these patches as either real or fake. This procedure is run convolutionally across an input image, averaging the responses to provide a final classification. Layers in this network consists of regular convolutional operations, with instance normalisation and leakyReLU activation functions. The size of the receptive field used here is 70×70 pixels.

To describe the U-Net and PatchGAN architectures we have been using in this thesis, we use a special notation. We define the following: we let `c7s1-k` denote a 7×7 block consisting of convolutions, instance normalisation and ReLU with k filters and a stride of 1. Further, we let `dk` denote a 3×3 block of convolutions, instance normalisation and ReLU with k filters and a stride of 2. `Rk` indicates a residual block containing two 3×3 convolutional layers, both with the same number of filters k. Lastly, `uk` denotes a 3×3 block with convolutions, instance normalisation and ReLU with k filters and a stride of $\frac{1}{2}$. Thus, we can describe the encoder part of the U-Net we use as `c7s1-64`, `d128`, `d256`, `R256`, `R512`, `R512`, `R512`, `R512`, `R512`, and the decoder part as `R512`, `R512`, `R512`, `R512`, `R256`, `u128`, `c7s1-3`.

For the PatchGAN, we define: `Ck` a convolutional, instance normalisation, leakyReLU block with k filters and a stride of 2. Then, we can describe the architecture as: `C64`, `C128`, `C256`, `C512`.

Model parameters

The model uses an Adam optimiser to learn, with the following parameters: learning rate $\eta = 0.0002$, forgetting factors $\beta_1 = 0.5$ and $\beta_2 = 0.999$, and $\epsilon = 1 \cdot 10^{-7}$. For the model loss function we use $\lambda_C = 10$ and $\lambda_I = 5$. The weights are initialised from a Gaussian distribution, $N(0, 0.02)$. Whenever leakyReLU is applied as an activation function, a slope of 0.2 is used.

Chapter 5

CycleGAN with Additional Penalty Term

One of the main purposes of this thesis is to build an improved cycleGAN model. In this chapter we present some theory and results related to different strategies. We first introduce the theory behind the ideas, and then their respective results.

5.1 Theory

A central objective is to increase accuracy of a model based on the cycleGAN architecture. This may be done by addressing a certain flaw or weakness in the predictions the model makes and try to learn it to avoid the mistake by introducing a penalty. The penalty accentuates the flaw and helps the model correct it.

The penalty is placed on the two generators, guiding them to produce more precise interpretations. Perhaps the most challenging aspect when it comes to penalties in a cycleGAN framework is that it relies on unpaired training data. This means that we are unable to e.g. compare a generated image directly to its reference because this image will most likely not be available at the time. A regular least squares loss will for example be of little use, since the motif of the images will be slightly different, at best. Hence, instead of penalising the quality of the generated image explicitly, we will need to penalise by comparing certain features that are present in every image, i.e. by focusing on the underlying distribution the images all come from. For this, statistical tools may be appropriate to use.

5.1.1 Covariance penalty

Early tests using the basic cycleGAN model have shown that generated images may suffer from a high level of noise in certain areas of the image. This noise appears as clusters of pixels with a higher or lower intensity than the surrounding area would indicate. A case illustrating this is shown in Figure 5.1.

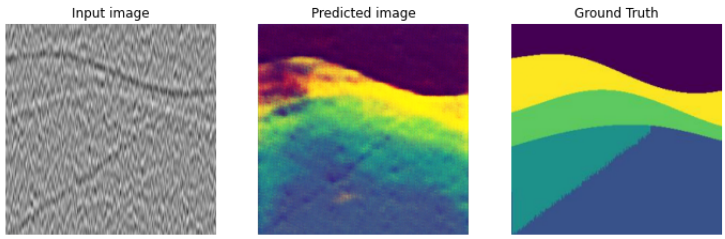


Figure 5.1: A figure illustrating how the basic cycleGAN occasionally makes predictions that are noisy. Notice the area in the upper left part of the predicted image.

To reduce this, one idea is to compute and use the covariances of a generated image and a reference image from the target distribution, and penalise by the distance between the covariances to make the generated output be more similar to the target distribution. The noise that can be found in the generated images can be considered to be outliers of the distribution of the image. By correcting this, i.e. by inducing a high loss value when many outliers are present in the generated image, the number of outlier pixels eventually may be reduced. We end up with the following objective function:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{cycleGAN}}(G, F, D_X, D_Y) + \gamma_C d\left(\Sigma(p_{\text{data}}) - \Sigma(p_{G(\boldsymbol{z})})\right), \quad (5.1)$$

where d is a distance metric, γ_C is a tunable hyperparameter weighting the influence of the penalty term, and Σ denotes the covariance of an image. This proposed solution is also discussed in one of the papers we have reviewed. Wu et al. (2020) did something similar, but on a regular GAN model instead. The cycleGAN objective function we used here is the same as described in Section 4.3. The covariance is computed by considering grayscale images (to reduce image dimensions from 3 to 1) and using a built-in covariance function in our actual implementation. The distance metric used here is the *Frobenius norm*, defined as

$$||A||_F := \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|a_{ij}|^2}, \quad (5.2)$$

where A is a $m \times n$ matrix, with elements a_{ij}. We thus end up with

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{cycleGAN}}(G, F, D_X, D_Y) + \gamma_C||\Sigma(p_{\text{data}}) - \Sigma(p_{G(\boldsymbol{z})})||_F. \quad (5.3)$$

5.1.2 Correlation penalty

As in the previous case, the idea behind the penalty term is based on how statistical properties of a distribution can be used as a regularisation in GAN training as seen in Wu et al. (2020). Now, we try to address the noise artefacts by looking at the correlations in the traces of the image. The idea is that by measuring how similar neighbouring pixels are − either lateral or by depth − and penalise accordingly, one would obtain more smooth,

clean and less distorted outputs. Images where there is noise present are to be harder penalised because of less correlation between neighbouring pixels. Thus, the accuracy of the model interpretations could be slightly increased.

For our model we will have the following objective function

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{cycleGAN}}(G, F, D_X, D_Y) + \gamma ||R_\tau(p_{\text{data}}) - R_\tau(p_{G(\boldsymbol{z})})||_F, \quad (5.4)$$

where R_τ is the correlation of the maximum vicinity of a point of τ units, either lateral or depth-wise. R is calculated using the principle of *autocorrelation* (Gubner, 2006): the correlation between a signal and a delayed copy of itself. It can be computed as

$$R_\tau = \frac{1}{\hat{\sigma}^2(n-\tau)} \sum_{t=1}^{n-\tau} (x_t - \hat{\mu})\,(x_{t+\tau} - \hat{\mu}), \quad (5.5)$$

where $\{x_1, ..., x_n\}$ are measurements and $\hat{\mu}$ and $\hat{\sigma}^2$ are the estimated sample mean and variance of the sequence of measurements. We go through each trace iteratively, computing the autocorrelation for a shift of $1, ..., \tau_{\text{max}}$, and add the vectors of length $|\tau_{\text{max}}|$ belonging to each trace, to a matrix, i.e.

$$R_{\text{Tot},\tau}(I) = \sum_{i=1}^{N} R_\tau(\mathbf{x}_i), \quad (5.6)$$

for an image I with N traces $\mathbf{x}_j$. We end up with a matrix of dimensions $\tau_{\text{max}} \times N$. We do this for each a generated and a reference image and measure the distance between them using the Frobenius norm. The distance is used as a weighted penalty.

5.1.3 Kullback-Leibler Divergence penalty

In this section we explore the capabilities of the Kullback-Leibler divergence (KLD) as a penalty measure. KLD provides a way to measure how similar two probability distributions are, or the "distance" between them.

The idea is the same as before: KLD penalty should help reducing some of the noise present in the generated images, and smooth out the overall appearance. It will penalise depending on how different the generated image, or rather its distribution, is from the reference image. We use KLD in the same way as previously defined, namely

$$\text{KL}(\eta||\mu) = \mathbb{E}_{x\sim\eta}\left[\log \frac{\eta(x)}{\mu(x)}\right], \quad (5.7)$$

for two probability distributions η and μ defined on the same probability space $\mathcal{X}$. We turn every trace (i.e. depth-wise) of the target generated image and the reference image into distributions, p_G and p_{data} respectively. This is done by considering grayscale versions of the images and make a histogram of the pixel intensities in the trace (here, we use 50 bins). We then compute the KLD based on the histogram distributions of traces of the generated and the reference image. This is done for every trace in the images, and the divergences

are added up to a total divergence for the whole image. Based on this, we arrive at the following objective function:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{cycleGAN}}(G, F, D_X, D_Y) + \gamma_K \sum_{i=1}^{N} \text{KL}(p_{Gi} || p_{\text{data}i}), \tag{5.8}$$

where N denotes the number of traces in the images. The parameter γ_K denotes the tuneable hyperparameter weighting the influence of the KLD penalty.

5.2 Results

The tests and evaluations of the additional penalty terms discussed previously in this chapter are displayed here. In these experiments, $N = 40$ images are generated and used for comparison with N reference images based on the same seismic images the generated images as based on. The experiments are performed using a synthetic seismic dataset, and includes faults and a pinch-out (see Section 3.3.1). We focus on seismic data with moderate amount of noise (see Figure 3.7), and vary how much training data is fed to the model and how long learning process lasts, to learn as much as possible about how the penalty affects the performance. As for the implementations, all code is written in *Python*[1] and *Tensorflow*[2] with extensive use of libraries like *Numpy* and *Matplotlib*. The training sessions are run on a *Tesla K80* GPU using *Google Colab*[3].

The quantitative measurements for quality evaluations we mainly use throughout this chapter and the next are computed by using the *mean squared error* (MSE), but *interpretation over union* (IoU) is also used. We omit metrics like PSNR and SSIM because of their similarities to the MSE, thus providing little extra depth to the quality evaluation. Below follows a summary of the named evaluation metrics.

5.2.1 Evaluation metrics

In this thesis, we perform experiments which we need to quantitatively evaluate. For this we use an evaluation metric. There exist numerous evaluation metrics, and for this task we need metrics that may reflect the accuracy of the structural interpretations in a clear way. Therefore, we present a few alternatives commonly used in tasks like these.

Mean squared error

A common metric used to compare predictions to the true labels is the *mean squared error* (MSE). It measures the average of squares of errors, i.e. the average squared difference between predicted values and true values. For images, we may use

$$\text{MSE} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i,j \in Y_n} \left(Y_{nij} - \hat{Y}_{nij} \right)^2, \tag{5.9}$$

[1] www.python.org
[2] www.tensorflow.org
[3] colab.google.com

where we sum up the squared differences between a reference image Y and a generated image $\hat{Y}$, pixel-wise, so that indices i, j represents pixels in images $\{Y\}_{n=1}^{N}$.

Jaccard index

The Jaccard index (Jaccard, 1901), also known as *Intersection over Union* (IoU), is an evaluation metric used for evaluating the amount of overlap between a target and a prediction. It can be defined as the size of the intersection divided by the union of the sample sets, or

$$\text{IoU} = \frac{\text{Area of overlap}}{\text{Area of union}} = \frac{\mid A \cap B \mid}{\mid A \cup B \mid}, \tag{5.10}$$

for two sets A and B. The *intersection* $(A \cap B)$ is comprised of the common pixels we can find both in the prediction and the reference regions, whereas the union $(A \cup B)$ is comprised of all pixels either found in the prediction or the reference regions. A larger value indicates a good segmentation.

PSNR

Peak signal-to-noise ratio (PSNR) is a term measuring the ratio between a noise-free image and a noisy approximation. PSNR (in decibel) is defined via MSE as

$$\text{PSNR} = 10 \cdot \log\left(\frac{\text{MAX}_I^2}{\text{MSE}}\right), \tag{5.11}$$

with MAX_I denoting the maximum number of pixel values of the image.

SSIM

Structural similarity (SSIM) index is a method for measuring the similarity between two images (Wang et al., 2004). SSIM is perception-based, incorporating important perceptual attributes like structural information. Structural information in this sense is based on the fact that spatially neighbouring pixels usually contain a significant inter-dependency, especially when they are close to each other. This dependency carry important information about the general structures in the images. SSIM is computed in the following way:

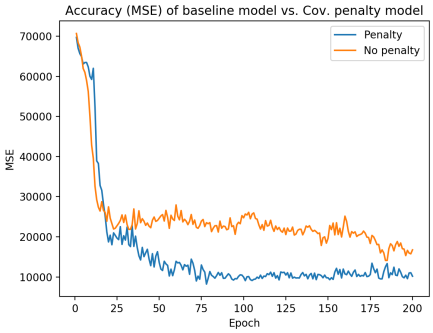
$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \tag{5.12}$$

for two equal-sized images x and y, where μ_x and μ_y denotes the average and σ_x^2 and σ_y^2 the variance of image x and y, with σ_{xy}^2 being their covariance. The variables $c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$ are used avoid instability when the denominator is close to zero, with $k_1 = 0.01$ and $k_2 = 0.03$ being default constants and L is the dynamic range of the pixel values (usually 255).

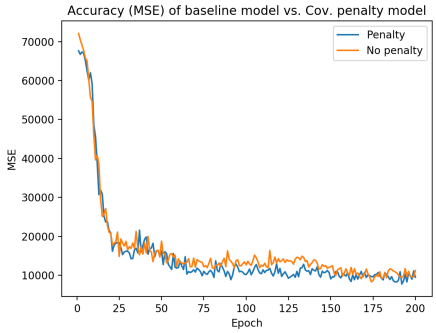
SSIM bases its comparisons on a distortion-free image as reference, i.e. it should be well-suited for comparing a generated image against a reference or ground truth.

5.2.2 Covariance penalty

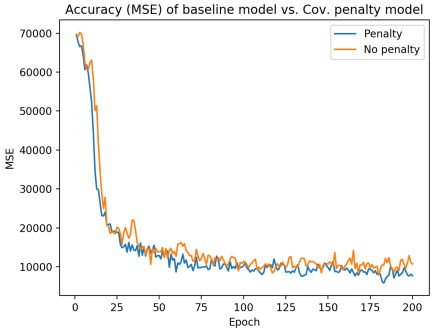
The hypothesis that there is a slightly higher chance of converging to a good local minimum when additional covariance penalty is added, is attempted tested in Figure 5.7. We conducted four sessions with an equal initial set of parameters: number of epochs set to 200, γ_C set to 0.13 (fixed after initial results indicating that this is in the vicinity of an optimal value) and total size of training data, $n = 5$.



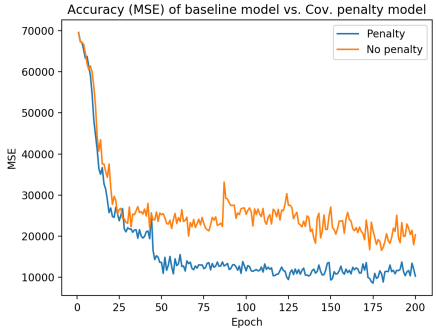
(a) Accuracy plot after first run.



(b) Accuracy plot after second run.



(c) Accuracy plot after third run.



(d) Accuracy plot after fourth run.

Figure 5.2: Comparison of accuracies of baseline model (without any additional penalty terms) and a model where a covariance distance penalty is added. We observe how the MSE has sudden drops in the plots where additional penalty is added.

As we can see from this small test, there might be a tendency of a lower MSE from the outputs where the model also includes the additional penalty term. It is also interesting to see how the MSE suddenly drops in two of the cases for the model with the additional penalty term. However, based on several more training sessions, the difference in MSE is often small and an actual improvement is not very easy to verify from looking at the actual outputs. Based on examining the quality of actual outputs, it is hard to notice any difference in prediction accuracy from when additional penalty is included compared to when it is excluded. This applies at least in smaller training sessions. When training is prolonged, with sessions around 300 epochs or more, there seems to be a higher chance of

a reaching a good local or global minimum so that predictions are of acceptable quality, considering high amount of noise and 5 training images. But there is no guarantee of the model reaching such a minimum, and this does not occur very often.

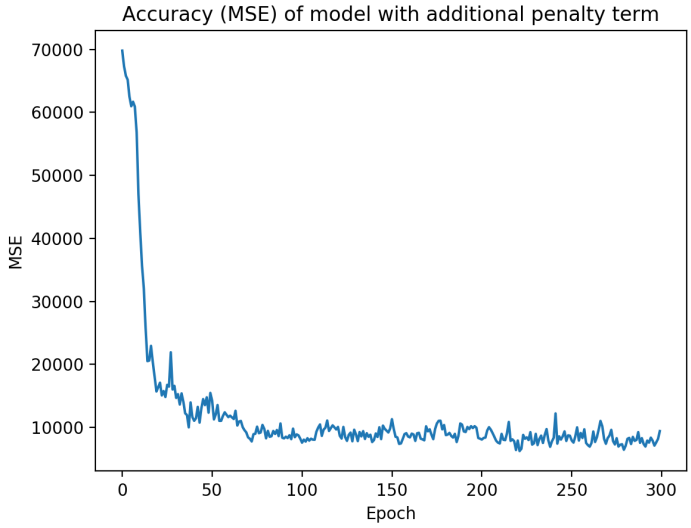


Figure 5.3: MSE of model with additional penalty term trained for 300 epochs.

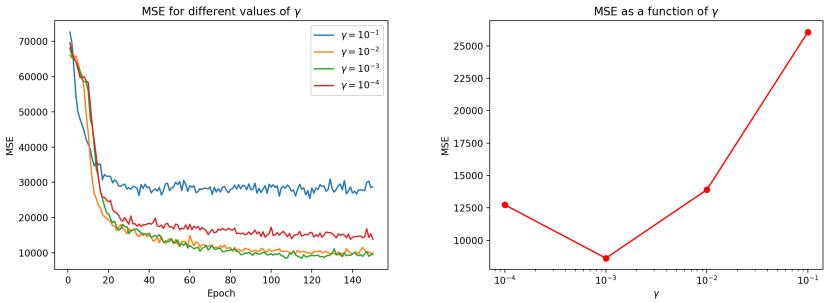
We also did a longer training session, to investigate how the model responds. From looking at the MSE in Figure 5.3 it is clear that this does not improve accuracy. But from looking at how the MSE regularly make sudden drops, there seems to be small increments in accuracy throughout the session, which may indicate that if the model is trained for a much longer period, the overall performance may increase.

Based on this, there are some evidence indicating that the covariance penalty leads to better performance. However, the difference is not striking in the long run. It can be observed that the penalty makes a slight difference in some cases, but more often the performance is about the same as that of the basic cycleGAN model. Additionally, the computation cost is substantial: an epoch may run for up to 80 seconds. If the strategy of using the covariance as a penalty is decided further pursued it would be natural to seek ways to optimise the method. Either through hyperparameter tuning, or investigating more sophisticated ways of capturing the covariance of the distributions, while also decreasing computation time drastically. For now, we proceed with pursuing other penalty options.

5.2.3 Correlation penalty

In this case we focus on measuring the correlation depth-wise. This has proven to yield the best results, as well as being easier to implement. The number of max shifts in the autocorrelation calculation, τ, is here set to be 6, but may probably be optimised through extensive hyperparameter tuning. The data we test the model on here is synthetic seismic data, with a training set size of $n = 5$.

We start with a more thorough investigation of how different values of the hyperparameter γ affects the accuracy. This may give an indication of which range of values may be best suited for weighting the correlation penalty. In Figure 5.4b we show the MSE for four different values of γ, each with a difference in order of 10.



(a) MSE of four different values of γ. The plots are based on a mean of five simulations per parameter value.

(b) Mean MSE of four different values of γ after model has started to learn, i.e. after around 50 epochs.

Figure 5.4

We observe that values of γ in the range of $0.01 - 0.001$ are yielding the best results. Higher values leads to over-penalising, which interrupts the learning process. On the other hand, going lower than 0.001 leads to the correlation penalty being almost insignificant in the learning process. The accuracy here looks to be on level with the accuracy of a model without the additional penalty term.

It is interesting to note that the curves for the first 20 epochs are steeper for higher values of γ, which may indicate that penalising harder in early training is beneficial, but this advantage quickly turns into a disadvantage later, as learning also stops at a higher level.

Further, we wish to investigate whether the accuracy improves or deteriorates when we apply the additional correlation penalty. In Figure 5.5 we see how accuracy varies over 150 epochs for both models. We use $\gamma = 0.01$.

We observe that the additional penalty term indeed helps in reducing MSE. We can further quantify this effect by calculating the *relative change* (in percent), which we here define as

$$\Delta := \frac{x_{\text{after}} - x_{\text{initial}}}{x_{\text{initial}}} \cdot 100\%. \tag{5.13}$$

We use the mean values of MSE for the last 60 epochs in both cases. This yields a relative change of $\Delta = \frac{9601 - 12224}{12224} \cdot 100\% \approx -21.5\%$, which is a significant reduction. We

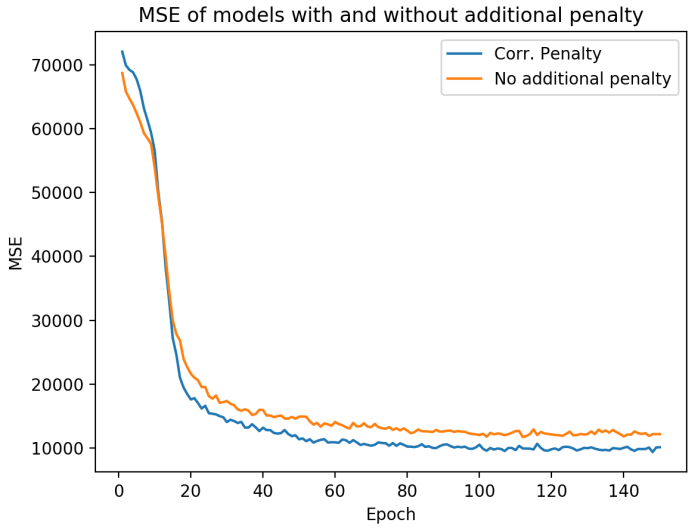


Figure 5.5: MSE of the model accuracy where $\gamma = 0.01$ and $\gamma = 0$. The plots represents a mean of 10 simulations each, to get a more representative result.

further proceed to investigate whether the size of the training set has any influence in how the model with the correlation penalty term performs. This is easy to test, and is visualised in Figure 5.6.

We observe that even when the size of the training set is doubled it does not lead to a significant reduction in MSE when the model has converged, even with the correlation penalty. It may appear that a MSE around 10000 is a threshold, at least on this dataset. However, more training data seem to help the model converge earlier. When the training dataset size is reduced the difference between including the penalty term and not including it becomes clearer. The less data is used in training, the more the penalty term helps in accuracy.

In Figure 5.7 we compare generated images from the model with and without additional penalty term. As we can see, when we include the correlation penalty we are able to remove some of the most noticeable noise. However, some of the noise is still present. And the penalty term does not help in correcting segmentation completely. All in all, the correlation penalty provides a small, but noticeable increase in accuracy in the tests we have done.

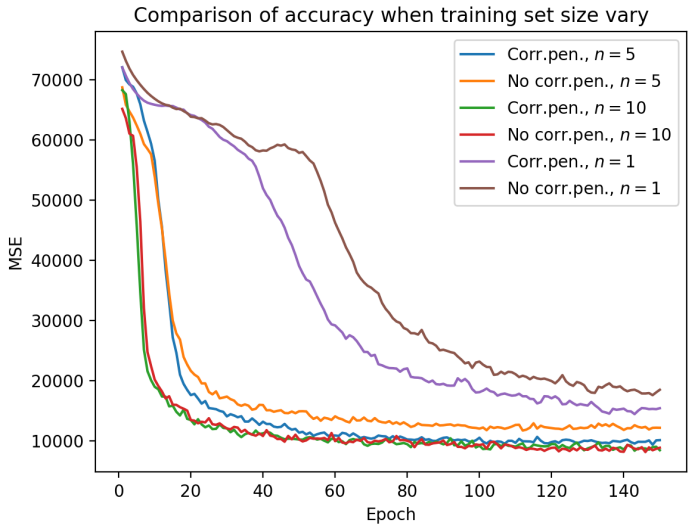
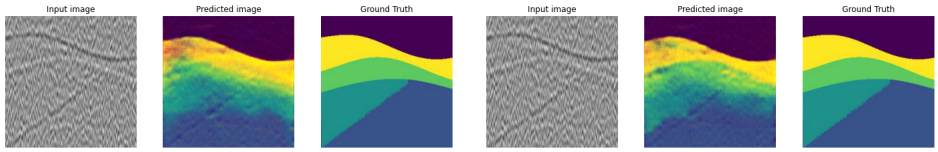
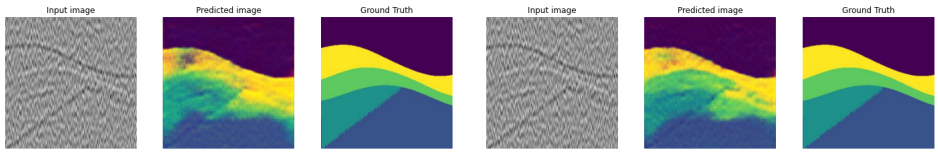


Figure 5.6: MSE of the model accuracy where n denotes the size of the training set. Here, γ is set to 0.01 where correlation penalty is included. The data is based on the mean of five simulations.



(a) Model output without additional penalty term.

(b) Model output with additional penalty term.



(c) Model output without additional penalty term.

(d) Model output with additional penalty term.

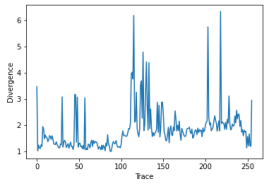
Figure 5.7: Comparison of model output with and without additional penalty term. The model is trained for 50 epochs in all four cases and $\gamma = 0$ and $\gamma = 0.01$ respectively.

5.2.4 Kullback-Leibler Divergence penalty

In Figure 5.8 we illustrate how the KLD evaluates the difference between a random sample image and a generated image. Low KLD means high similarity, and vice versa. As previously stated, since cycleGAN learns with unpaired images, a direct comparison between reference and generated images is impossible. Nonetheless, it is clear that the KLD is able

to find similarities in the structure even if the images are not related. The left parts of the images are the most similar, but from approximately epoch 100, noise and disturbance leads to large oscillations. Time per epoch for this penalty is slightly lower than for the correlation penalty.

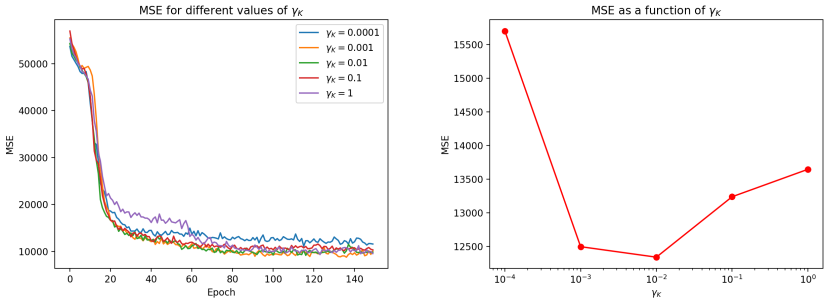
(a) A reference image of a geological model. **(b)** A generated image of a geological model.



(c) The resulting KLD from comparing each trace of the reference image to the generated image.

Figure 5.8: An illustration of how the KLD evaluates the distance in distribution of histograms of a reference and a generated image.

To tune the hyperparameter we run training sessions with γ_K varying from 0.0001 to 1. We run 5 sessions per value of γ_K and average these to get a more accurate indication of how the model performs with the respective weighting of the additional penalty term.



(a) MSE of five different values of γ_K. The plots are based on a mean of five simulations per parameter value.

(b) Mean MSE of five different values of γ_K after model has started to learn, i.e. after around 50 epochs.

Figure 5.9

The results are displayed in Figure 5.9a. As we can see from the plots, the performance

is fairly equal despite the variation in γ_K. However, for the lowest value the MSE is slightly higher overall. This indicates that the regularisation is almost insignificant here, providing almost no guidance to the learning process. For $\gamma_K = 1$ we observe that the performance is irregular, with a sudden drop in MSE at about 60 epochs. This may indicate that the penalty term is regulating the learning process too hard, leading to sub-optimal learning. For the three remaining parameter values, the performance is more even and more tests will be needed to find the optimal hyperparameter value. But for now, we proceed with a value of 0.01, which seems to be in the vicinity of the optimal parameter value here.

If we compare performance with and without the KLD penalty term, we see that there is a significant improvement. This is illustrated in Figure 5.10. Following the procedure from the last section, we quantify the reduction by looking at the relative change, Δ. In this case, we get a reduction of $\Delta = 19.6\%$, which is also significant. We do not include any plot of tests with different training set size in this section, as we assume that the effect is similar to that of Figure 5.6.

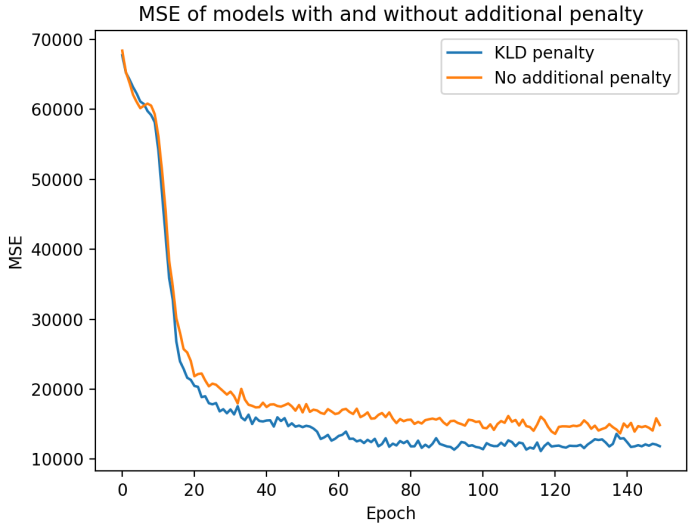
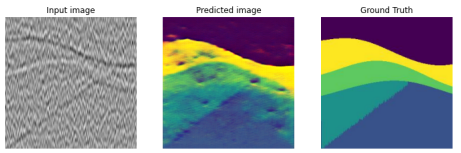


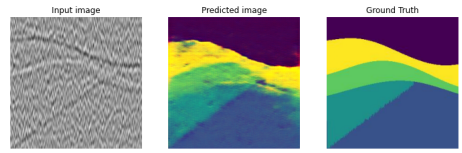
Figure 5.10: MSE of an average of 5 sessions with $\gamma_K = 0.01$ and $\gamma_K = 0$. The reduction of MSE when KLD penalty is added is significant.

We also include a figure showing how the KLD penalty term improves the actual interpretations. In Figure 5.11 we compare outputs of a model with no additional penalty to those of a model with the KLD penalty activated, with $\gamma_K = 0.01$. It is clear that there is an improvement in the interpretations from the model with the KLD penalty. The outputs are more clean, with less occurrence of dots of dark noise, and the visuals are smoother overall. However, this improvement is not always as clear as it is here; some variability in quality when many outputs are compared is present. One reason why this is the case, may be that the penalty term regularises the pixel intensity distribution, which just indi-

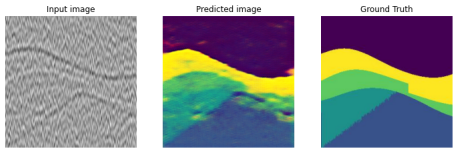
rectly helps reducing the noise. Yet, the figures illustrates that in many cases the additional penalty term helps in enhancing interpretation quality.



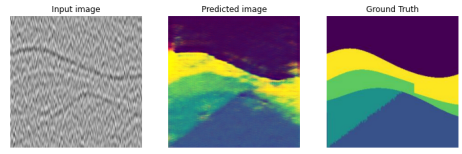
(a) Model output without additional KLD penalty term.



(b) Model output with additional KLD penalty term.



(c) Model output without additional KLD penalty term.



(d) Model output with additional KLD penalty term.

Figure 5.11: Comparison of model output with and without additional KLD penalty term. The model is trained for 150 epochs in all four cases and $\gamma_K = 0$ and $\gamma_K = 0.01$ respectively.

5.3 Summary

In this chapter we have experimented with three different penalty terms. Two of them, the correlation penalty and the KLD penalty terms described in the previous two sections prove to be the most promising. As we have seen, they both provide a significant drop in MSE, with a relative reduction in MSE of 21.5% and 19.6%, respectively. This clearly implies an increase in interpretation quality, with the penalties also not being too computationally expensive. To decide which one of the two that are best suited for the task of interpreting seismic data we may need to do further analysis. For now, we settle with using the correlation penalty, which tests indicate may give a slightly more stable performance overall.

Chapter 6

Results

In this chapter we present the results from training our modified cycleGAN model on real seismic data from the F3 dataset . We present some initial results as well as results where we examine how training with rich and sparse datasets affects performance. We also look at what effect extended training has on the quality of the interpretations. We use inlines and crosslines from the dataset described in Chapter 2, composing a training dataset by evenly extracting lines from the volume. Figure 6.1 is a sketch illustrating this.

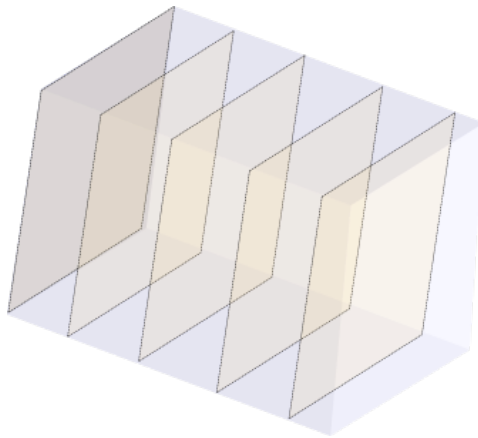
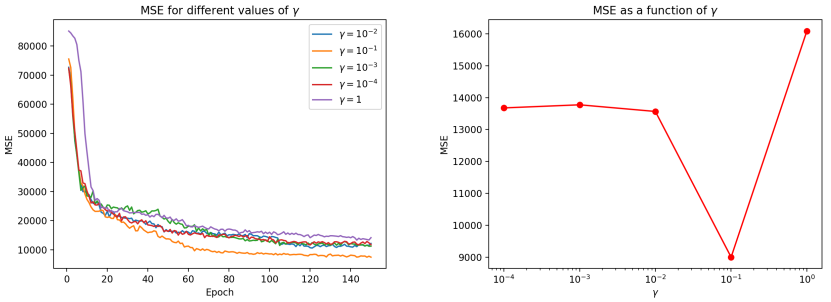


Figure 6.1: A figure illustrating how we extract lines from a given volume. The light blue area represents the seismic volume, while the light yellow planes indicates evenly extracted 2D lines.

For the evaluations, we use MSE and Jaccard Index. As in the previous chapter, the test set used for comparison contains 40 images, extracted from the volume in a similar way as described above.

6.1 Initial results and tuning of hyperparameter

For the real seismic data, we re-tune the hyperparameter γ to see whether the MSE accuracy behaves differently when the training data basis is changed from synthetic to real. We run training sessions with values of γ ranging from 1 to 0.0001. The results are shown in Figure 6.2.



(a) MSE for models where the penalty term hyperparameter varies from 1 to 10^{-4}. Each curve is a result of an average of 5 training sessions.

(b) Mean MSE of four different values of γ after model has started to learn, i.e. after around 50 epochs.

Figure 6.2

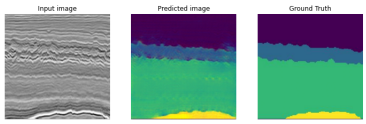
As we observe in Figure 6.2, we get the lowest MSE when $\gamma = 0.1$. This is higher than what we saw when we tested with synthetic data. Although we observe that the plot clearly indicates an optimal value for γ, the general trend of the plot is somewhat ambiguous, and not very similar to Figure 5.9b. More data points would have been preferred to strengthen the assumption that a minimum is present in the vicinity of $\gamma = 0.1$. For now, we proceed with the optimal hyperparameter value, $\gamma_{\text{opt}} = 0.1$.

When we run the first training sessions on the dataset, it is immediately clear that this is a more challenging task than using synthetic data, due to the increased complexity. Hence, more training data is in general needed. Additionally, an initial impression is that the training set should be well-designed in order to facilitate constructive learning. We will see examples of why this is important later.

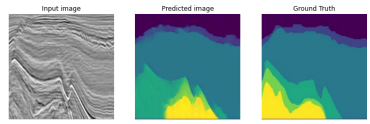
Figure 6.3 shows a couple of the initial results we performed on the two regions of the volume.

We observe that the model is capable of interpreting and reproducing certain shapes and patterns, while others are more difficult. A way to illustrate this is to study which pixels are correctly interpreted, and which are not. This is illustrated in Figure 6.4, where the green pixels in the black-white contrast representation indicates the inaccuracies of the model. In the generated image, the lighter green layer located directly above the yellow surface is not always recognised by the model.

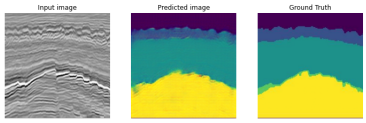
From the figure, it is clear that the model is able to reproduce decent interpretations. Given some initial conditions from the labelled training set (e.g. how many and what colours we want to display, their order etc.) the model is able to interpret the seismic and pick the correct horizons for segmentation in the majority of the cases. However, some



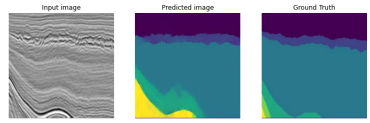
(a) Model output 1 on the Western region.



(b) Model output 1 on the Eastern region.



(c) Model output 2 on the Western region.



(d) Model output 2 on the Eastern region.

Figure 6.3: Model-generated outputs compared to its reference. The model is trained for 150 epochs in both cases.

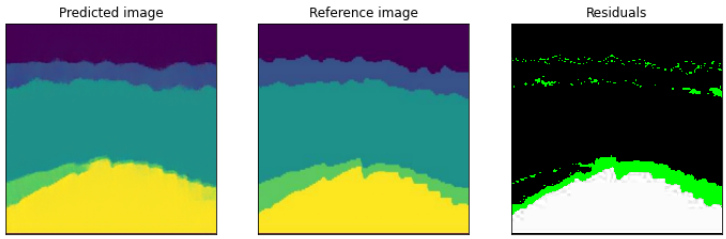


Figure 6.4: A figure illustrating how a prediction may differ from its true label. Green pixels indicates areas where the difference between the generated label and the actual label is bigger than some small threshold (here, 0.1 is used).

predictions are inaccurate. Especially when the seismic includes fewer reflectors than our model anticipates. The model seems determined in always including all the colours. This special case is illustrated in Figure 6.5. This is an illustration of what might happen if we not, through the training set, prepare the model for this special case.

We also include Figure 6.6 to illustrate why the cycleGAN is considered to be practising *steganography*: the art of hiding data inside an ordinary image. As we observe, several reflectors are visible in the generated images. This is seen as a way of ”cheating”. The model can, by hiding data, use the information later to reproduce the original image and thus trick the cyclic loss (Chu et al., 2017). This artefact is particularly visible in the early stages of training, and mostly disappears when the learning matures. It is an interesting demonstration of the smartness, or laziness, of the cycleGAN.

6.2 Rich versus sparse data

To get a more realistic training scenario, we from now on include both inlines and crosslines in the training data. In this way, the model in principle learns the full body of the domain

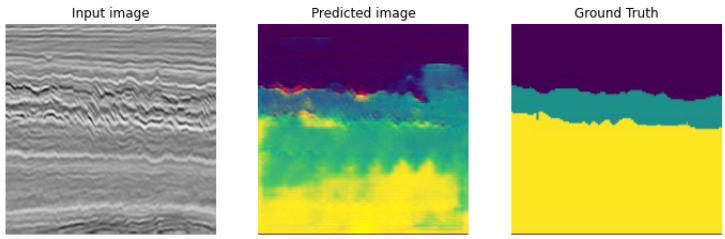
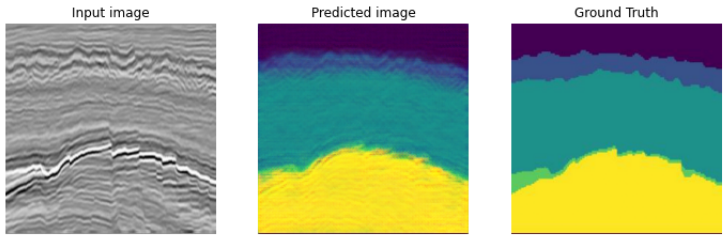
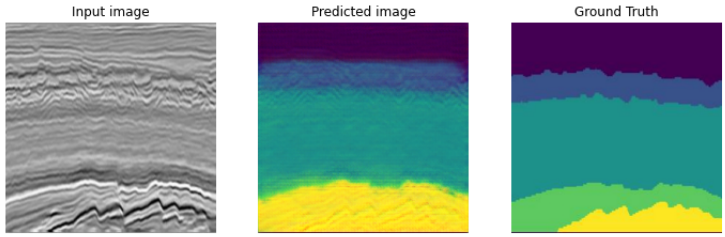


Figure 6.5: A special case where the model tries to include more colours/horizons than what is present in the seismic.



(a) Model output example 1 with disguised, hidden reflectors.



(b) Model output example 2 with disguised, hidden reflectors.

Figure 6.6: Model output after a training session of 60 epochs.

of interest.

In this section, we experiment with how model output quality varies from training with rich and sparse datasets. The seismic volume is, as previously mentioned, divided into smaller regions. This is is done to standardise the input image dimensions. The volume is split into cubes of dimension $286 \times 286 \times 255$.

We define a sparse dataset as a dataset containing 5 inlines and 5 crosslines evenly spread across the domain, in this case the F3 dataset. A rich dataset is defined as containing 20 inlines and 20 crosslines extracted from the domain. How we extract training lines can be seen in Figure 6.1. The size of the training sets we use here would in most settings

involving training a deep learning model, be considered very small. The sparse dataset consists of 1.75% of the inlines and crosslines, while the rich dataset consists of 7.00% of the total number of inlines and crosslines in the volume. We test the model both on the simple and complex seismic, to get a better understanding of the capabilities of the model.

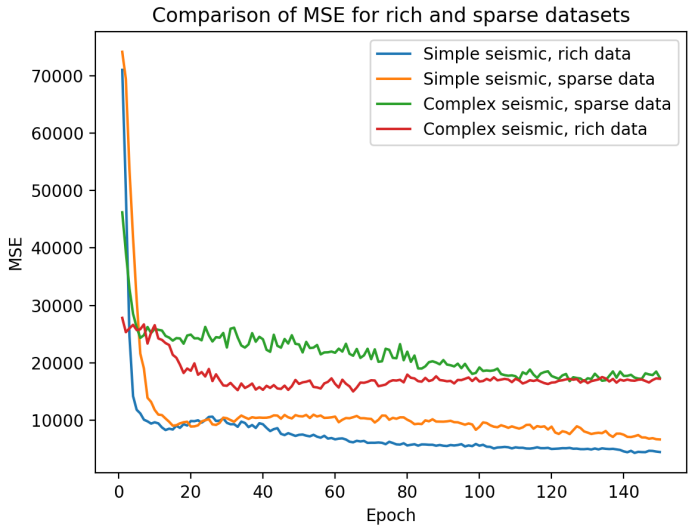


Figure 6.7: MSE for models trained on rich and sparse datasets. The rich dataset is based on 20 inlines and 20 crosslines. The sparse dataset consists of 5 inlines and 5 crosslines. Complex seismic indicates the Eastern region, while simple seismic indicates training and predicting on the Western region.

In Figure 6.7, the difference between using the rich and the sparse dataset for training is visualised. As we can see, the results are reasonable: MSE is lower for the rich datasets than for the sparse datasets. The MSE is also higher when the model is trying to interpret the more complex seismic. We also observe that there noticeably more oscillations in the MSE when the seismic is complex. An explanation for this might be that it is more difficult for the model to understand the structures here, so it keeps doing changes to the interpretations as learning evolves.

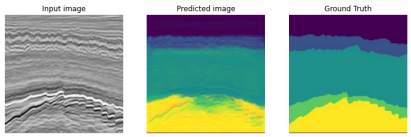
	Simple	Complex
Sparse	6692.77	17945.12
Rich	4623.10	17263.06
Δ	30.9%	3.8 %

Table 6.1: MSE values for the four cases.

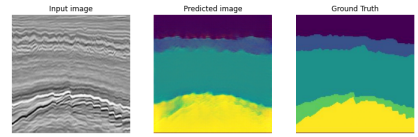
	Simple	Complex
Sparse	0.442	0.286
Rich	0.457	0.293
Δ	3.4%	2.4 %

Table 6.2: Jaccard index values for the four cases.

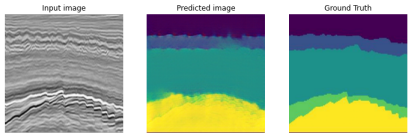
Figure 6.7 is also summarised in table form. Table 6.1 shows the respective MSE values, with Jaccard index values for the four cases in Table 6.2. We also include relative change, here denoted by Δ. The Jaccard index and MSE values are calculated at epoch 150, at the end of training. It should be noted that, usually, a Jaccard index score of 1 implies a perfect segmentation. On this task, however, a very good segmentation scores in the range of $0.5 - 0.6$. This is due to the way we represent images here. To provide some context, the single image displayed in Figure 6.3a has a measured Jaccard index score of 0.51, and MSE of 1368, while Figure 6.3b has Jaccard index score of 0.25 and MSE of 18768. In essence, the tables confirms the impression from the figure. More training data improves interpretation quality. From the numbers, however, the difference is mostly small, with one exception. We observe that the relative change for MSE in simple seismic is as large as 30.9%. The accuracy is approximately the same for both sparse and rich datasets trained on complex seismic, but with a slight improvement. The reason for this may be due to the complex seismic being too difficult to handle for datasets this small, but may also partially be explained by the fact that MSE (and Jaccard index here) are not being very sophisticated evaluation metrics, thus failing to capture a possibly more correct structure for the model trained with the rich dataset. The verdict is that for the complex dataset we work with here, both the size of the training set and the number of epochs we train the model with must be greatly increased to get decent results. We mostly focus on the simpler seismic from this point forward, to narrow our scope slightly.



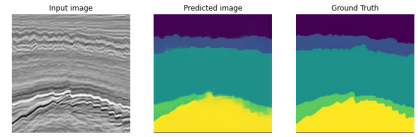
(a) Model output after 50 epochs, trained on the sparse dataset.



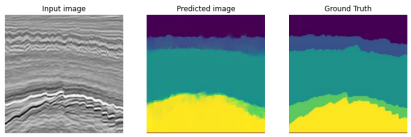
(b) Model output after 50 epochs, trained on the rich dataset.



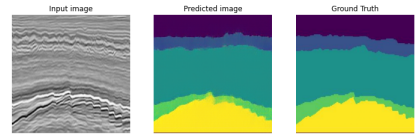
(c) Model output after 150 epochs, trained on the sparse dataset.



(d) Model output after 150 epochs, trained on the rich dataset.



(e) Model output after 300 epochs, trained on the sparse dataset.



(f) Model output after 300 epochs, trained on the rich dataset.

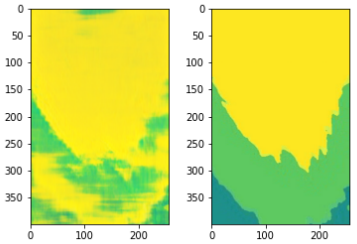
Figure 6.8: Comparison of model outputs after 50, 150 and 300 epochs, training on a sparse and a rich dataset.

To visualise how these various training conditions affects the model performance, we refer to Figure 6.8. Here, we add results from training sessions with three different lengths, $50, 150$ and 300 epochs. The figure shows how more data and more epochs influence the interpretations, when trained on simple seismic. Clearly, the poorest results are obtained with the fewest epochs and the smallest training set. And for the sparse dataset, it seems like obtaining a "correct" segmentation, takes time. Even after 150 epochs, the model has not found a way to successfully separate the two lower, main horizons. As Figure 6.8f indicates, a clear segmentation may be possible after many enough epochs. After 300 epochs, there are signs showing that constructive learning still takes place.

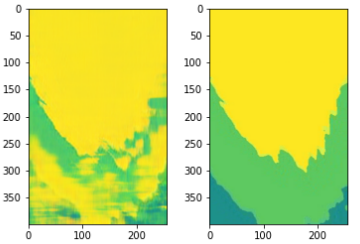
When we train on the rich dataset, this goes faster: already after 150 epochs we achieve an almost clear segmentation in the lower region of the image. The upper region of the image is still a bit different from the labelled image it is compared to, but overall the output is tidy with a minimal presence of noise artefacts.We explore further what effect epochs have on quality in greater detail in the next section.

We are also interested in how good the model predictions are overall. To study this, we examine a horizontal slice in the predicted macro model. This is illustrated in Figure 6.9.

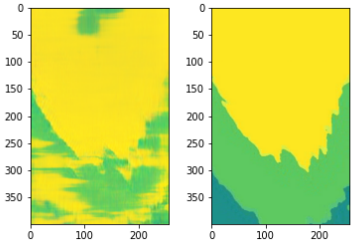
As we can see, improvement over a total volume is far slower than for single cross-sections. This is also anticipated, given that there are far more details in a slice covering the whole volume than there are in a single lateral slice. However, it is worth mentioning that the model is these cases the model is not trained on any seismic from about line 285 and onwards. This is also visible in the figure. The type of seismic we find in this area is thus completely unseen. This shows that out model has some restrictions when it comes to facing data it has not learned anything about. However, it may be able to reproduce certain areas accurately also here.



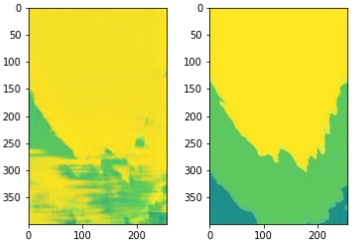
(a) Model output after 150 epochs, trained on the sparse dataset.



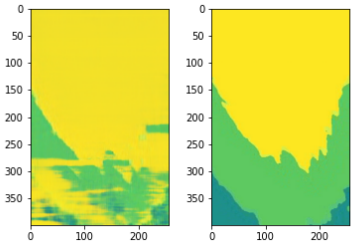
(b) Model output after 150 epochs, trained on the rich dataset.



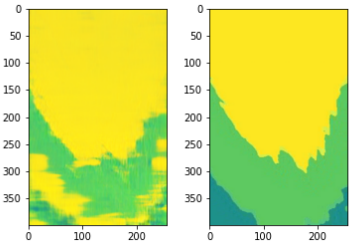
(c) Model output after 300 epochs, trained on the sparse dataset.



(d) Model output after 300 epochs, trained on the rich dataset.



(e) Model output after 600 epochs, trained on the sparse dataset.



(f) Model output after 600 epochs, trained on the rich dataset.

Figure 6.9: Comparison of model outputs trained on sparse and rich datasets for a horisontal slice across the volume, at depth 210.

6.3 Training length

To investigate whether the model interpretations can be improved even more, we extend the training process. We let the model train for 600 epochs on a sparse dataset and observe if there are any reductions in MSE and evaluate prediction quality manually. Figure 6.10 shows the MSE for three training sessions of extended length, compared to an averaged set of sessions where no additional penalty is added to provide some context. As we observe, convergence is apparently reached eventually. It may seem like the phase where conver-

gence has been reached (the time when the small oscillations have stopped) is varying. In the second training session, this may have been reached just before reaching 600 epochs, while it occurs much earlier for the first session. In general, as we have previously seen, we obtain better predictions when we include the additional penalty term. However, in what may here be described as the early learning phase (from 0 to around 150 epochs) we observe that the oscillations are larger when we include the additional penalty. A reason for this might be that we then allow the model objective function to jump from minimum to minimum to a larger extent, increasing the probability to eventually find a (close to) optimal local minimum.

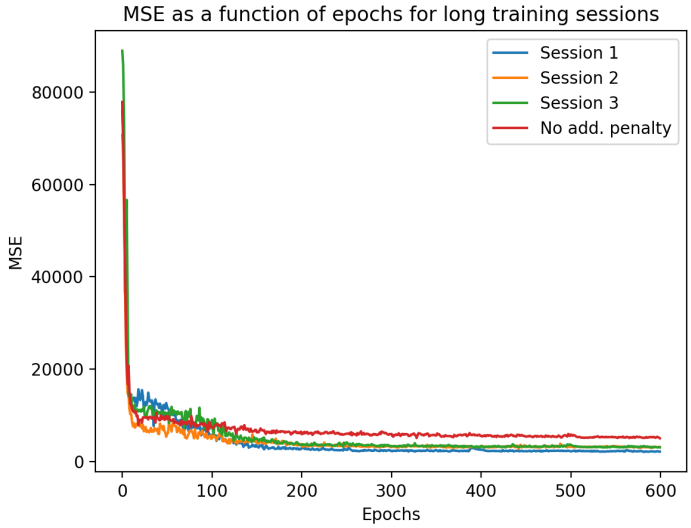
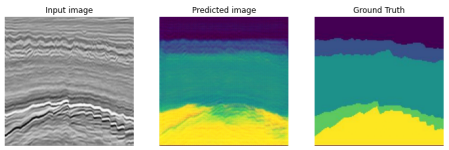


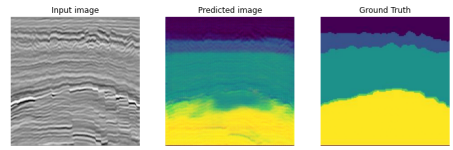
Figure 6.10: MSE for three sessions where the model is trained on a sparse dataset, with a benchmark consisting of several sessions averaged into one where no additional penalty term is included.

In Figure 6.11 we compare model outputs from training sessions of varying length. As we have previously seen, interpretation quality improves continuously when we run long training sessions. This is also evident here. After 600 epochs, the cross-sections provided here seem to be fairly accurately interpreted. Although this is the case here, this progress may vary from session to session. This is due to the randomness in the learning process, which prompts the local minima the model finds to not always be equal. This also means that they vary in quality. However, these plots indicate that the probability of finding a good local minima definitively increases when training sessions are longer.

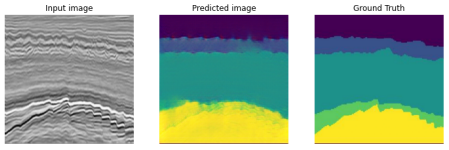
Time per epoch is of interest, as it is an indication of the efficiency of the model as a useful tool. Note here that we throughout this chapter are using the extended model: the model where the correlation penalty term is included. This leads to higher accuracy, but also to epochs being slightly slower. Tests show that the reduction in speed per epoch when the penalty term is added is about 30%. The performance of the GPU we have used through *Google Colab* varies, probably due to variable traffic and load, but time per epoch



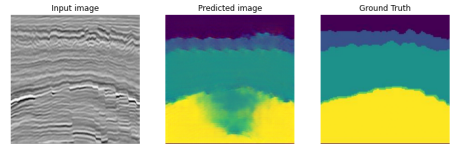
(a) Model prediction of inline 1 after 50 epochs, trained on the sparse dataset.



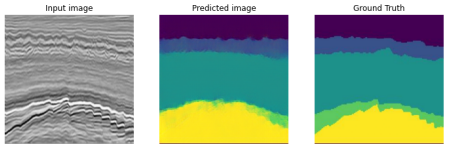
(b) Model prediction of inline 2 after 50 epochs, trained on the sparse dataset.



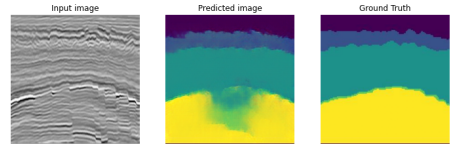
(c) Model prediction of inline 1 after 150 epochs, trained on the sparse dataset.



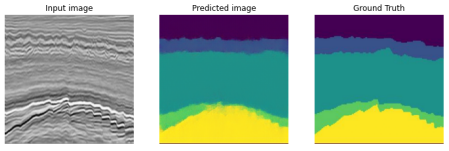
(d) Model prediction of inline 2 after 150 epochs, trained on the sparse dataset.



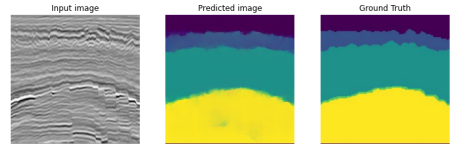
(e) Model prediction of inline 1 after 300 epochs, trained on the sparse dataset.



(f) Model prediction of inline 2 after 300 epochs, trained on the sparse dataset.



(g) Model prediction of inline 1 after 600 epochs, trained on the sparse dataset.



(h) Model prediction of inline 2 after 600 epochs, trained on the sparse dataset.

Figure 6.11: Comparison of model outputs after a various number of epochs. We compare output from two different inlines.

when training on the sparse dataset seem to be around 9 to 15 seconds. For the rich dataset, time per epoch varies between 26 and 44 seconds. This indicates a relative change in time per epoch of about 170%, assuming times in the middle of the respective intervals. If time is to be seriously considered when using the model, there is much to gain with using a smaller training set.

Chapter 7

Discussion

In this chapter we provide a more in-depth discussion where we reflect on the results presented in the previous chapter, and what value that lies within our model in practical use, including its strengths and weaknesses. We also include some suggestions to peers interested in this particular subject, based on the experiences we have accumulated throughout this project, particularly related to interesting alternatives to pursue in future studies.

7.1 Main remarks

The initial results on real data show that the hyperparameter value γ that was found in Chapter 5 needs to be adjusted when tested with real data. The new value clearly improves the performance of the model. This emphasises the importance of correctly tuning hyperparameters in order to get optimal results. In this case it would be beneficial to include more data points in the tuning process to determine the correct value with confidence. Additionally, it illustrates how vulnerable such a deep learning model can be. There are a variety of different, tunable parameters involved in the architecture of models like the one we apply here. A slight inaccuracy in tuning one of them may corrupt the results. Therefore, a serious amount of time should be invested in taking care of the hyperparameters involved.

Nevertheless, from the results it is clear that the model also exceeds the basic cycleGAN model when applied to a real data case. We have mainly used a modified model where the correlation penalty term is added, but using a model with the KLD penalty term instead might have yielded promising results on real data as well. The fact that we are able to increase the performance in these two cases also gives a reason to believe that there lies potential in developing penalty terms with greater sophistication and theoretical embedding. This may further increase model performance, as seemingly, the results based on the vanilla cycleGAN model is far from optimal.

Moreover, we have in this thesis solely focused on enhancing performance through modifying the objective functions in the GANs, leaving no attention to the residual parts of the architecture. There might lie unreleased potential in experimenting with gener-

ator/discriminator setups, e.g. replacing the U-Net generator with newer technology (a promising replacement could be the HRNet (Wang et al., 2020)), or just make incremental changes to the U-Net structure itself. Furthermore, it may be useful to review the codework as better solutions to certain parts of the framework definitively exist.

It is fair to say that the current overall setup presented in this thesis is far from optimised, and is working merely as a demonstration of what the promising technology of GANs is capable of showing us.

When we turn to the actual results, it is obvious that an increase in training dataset size, i.e. from a sparse to a rich dataset, leads to an increase in prediction accuracy. We experience a 30.9% relative increase in MSE accuracy when working with simple seismic, as well as an increase (albeit lower) when the seismic is complex. This is well anticipated from general deep learning theory. And this especially comes to sight in regions of the images which are harder to interpret. For instance, the lower part of the a certain part of the seismic contains a reflector which is difficult to capture (as seen in Figures 6.8 and 6.11). More data makes the model more capable of handling the strenuous parts, like this area. It is also evident from the results that the complex region is too demanding to produce good quality results overall, given the small datasets we have been working with in this thesis. Failure to produce good results here also displays the limits of the model, and how it is yet not able to fully understand the fundamental principles behind the seismic. What a further increase in training set size could lead to in this matter seems to be a plausible option to explore.

An increase in training set size also leads to slower epochs. Going from the sparse to the rich dataset increases time per epoch with approximately 170%. This highlights the trade-off between size of the dataset and the training time, which ultimately should be a choice made by the user, based on the overall goals determined in the actual exploration project.

An important aspect of the training dataset discussion is its composition. Even though the GAN is able to learn abstractions, its abilities are somewhat restricted. Here, it can best be visualised by training a model on exclusively complex seismic (eastern region) and try to predict on the simpler seismic structures in the western region. There are few similarities in the geological shapes in the two regions, and the model will attempt to fit certain complex looking structures to simple seismic. This is illustrated in Figure 7.1.

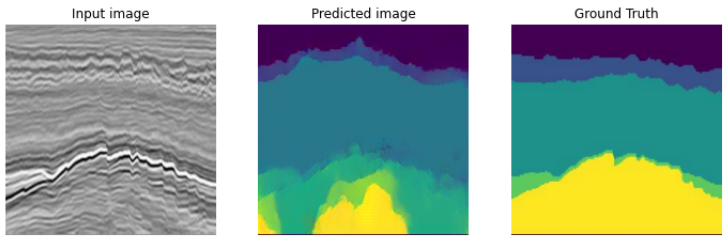


Figure 7.1: Model output from a case where we train on complex seismic but predict on simple seismic. The model is trained for 300 epochs.

Due to this problem, an interesting topic to pursue in further work would be to apply a mild form of *transfer learning* (Pratt, 1993) to the model. Transfer learning is the ability to store knowledge about one problem and apply it to a different, but related problem. Here, this may be used to learn the model to generalise seismic data and possibly make it more robust when exposed to geological structures dissimilar to those it has been trained on.

In general, to ensure that the base for model predictions is as good as possible, deliberate dataset preprocessing is a necessity. The training dataset should reflect the target geological macro model extensively. This e.g. requires diversity in training data so that decent results across the target volume is obtainable. A good strategy is to pick data evenly across the volume, to make sure that most of the structures in the seismic volume is represented in the training data. As we have seen, Figure 6.5 is a special case illustrating how things can go wrong when the model comes across something it does not comprehend because of a lack of explicit learning. This discussion also sheds light on another aspect of the human versus machine debate: even though we want to leave certain trivial and tedious tasks to be automatised with the aid of deep learning, humans will still be imperative in facilitating the task to be automatised.

It can also be argued about what would be a representable number of training images used to predict on a volume in a real case. What we may conclude with on this matter is that the more complex and variable the seismic is, the more data is needed to ensure good quality in model output. However, even this is not a guarantee for good results, because as we have seen: training GANs can be difficult, and there is always a risk of converging to a suboptimal local minimum.

Moreover, based on the results we can conclude that the arbitrary cutoff we initially fixed at 150 epochs in many cases is not sufficient for producing accurate enough predictions. This is especially evident in cases where the training dataset is sparse. Even though training is slower, i.e. it takes more epochs to reach the same level of accuracy as with a rich dataset, improvement does not stop at 150 or 300 epochs. Based on visual inspections, we can see that even after 600 epochs, incremental improvements in some cases still may take place. Learning progress is varying in individual training sessions, so that the amount the model has learned after a given time may be different from session to session. However, it is evident that letting the training last for more epochs is beneficial for model output quality, up to a certain extent. This observation helps emphasise the fact that eventually there are other ways than to continuously increase training dataset size to achieve predictions of good quality.

This is also illustrated in Figure 7.2, which shows the time perspectives of training with rich and sparse datasets. As we can see from the figure, a model trained on a sparse dataset may reach about 750 epochs by the same time a model trained on the rich dataset reaches 300 epochs. This takes about 2.8 hours. Hence, there is evidently a clear trade-off between choosing to include more data in the training process and how much time is spent on the training itself. This knowledge may be useful in cases where the amount of good quality training data is limited. As we have seen, training the model on a sparse dataset for approximately 3 hours in some cases is sufficient for decent results. This is also considered to be a relatively short training time for a deep learning model, which in many cases could take days. In other cases, perhaps in places where the geology is more complex, letting the model train over night with a richer dataset might be a profitable trade for better results.

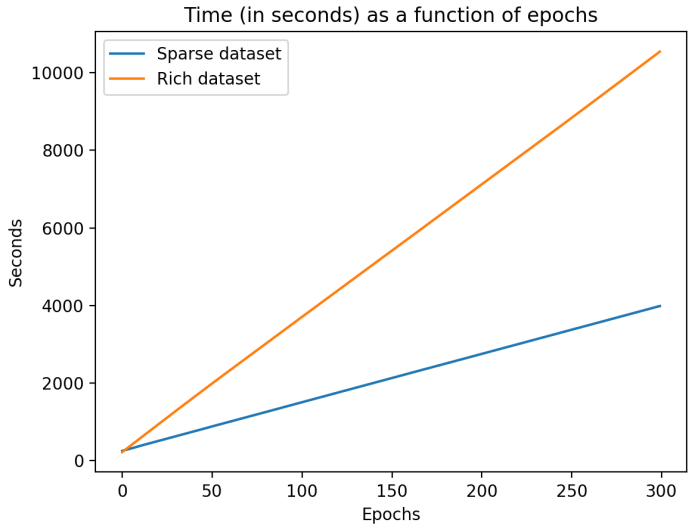


Figure 7.2: Time plotted as a function of epochs for training sessions on a sparse and a rich dataset.

A question that emerges is how long one in general would want to train the model to ensure *good enough* interpretations. In order to determine this, we need to define what is *good enough*. This is also something that is likely to be a highly subjective opinion. With *good enough* interpretations, we here assume interpretations that with a fair accuracy is able to reproduce the geological structures, e.g. clearly highlights the different strata. Based on several experiments we suggest that training the model for $600 - 900$ epochs should be enough to, with a high confidence, provide good interpretations. This applies for simpler seismic data, like the type we find in the western region of our dataset. For more complex seismic, longer training must be expected in order to produce decent quality output.

Lastly, the way we choose to evaluate the quality of the generated outputs in this thesis should also briefly be addressed. We have mostly been using the MSE as an evaluation metric. However, this metric has its flaws when it comes to image evaluation. Comparing images pixel by pixel may say something about how similar the single pixels are, but does not completely capture the similarity or dissimilarity of the structure of the images as a whole. In this sense, MSE may be viewed as somewhat ”primitive”. It has proven to be difficult to find an evaluation metric that allows for differentiating incremental, but significant, improvements in the structural interpretations. For future studies, a hybrid between the MSE and the IoU measuring the per-pixel segmentation accuracy could perhaps be an improvement. Nevertheless, using MSE plots along with visual inspections seem to provide an acceptable compromise and solution to the problem, for now.

7.2 Value of model in practical use

The results we provide in this thesis is a great demonstration of what problems deep learning is capable of potentially simplifying. One of the criteria such a model should satisfy in order to be useful in practice is *scalability*, i.e. be able to handle large datasets without consuming proportional amounts of resources. Our model proves to be scalable, given a training dataset. Once the model is trained, predictions are generated almost instantly. Moreover, the time it takes to train a model is, as we have seen, highly manageable. The limited amount of training data the model needs is one of its strengths. We only need a very small portion of the total volume (about $1.75\% - 7.00\%$) as training data and train the model for only a few hours in order to most likely receive decent results. If done manually, this process could potentially take weeks. This is a sound proof of value. But this again comes with a certain cost: the training set composition should be well-thought out if good quality results are to be expected. Nevertheless, the value here is significant.

The somewhat limited ability of making abstractions beyond what the model has been trained on, and therefore its limited robustness currently restricts its usefulness to some extent. With the current setup we are dependent on data engineering to compose an optimal training dataset for learning, which is a key to achieve high quality results. It is beyond doubt that the value of such a model would increase significantly if it was able to understand seismic data on a fundamental, deeper level. This is also something that potentially be targeted in further studies.

When we compare our model to similar, already existing models, we can conclude that what the model potentially lacks in interpretation accuracy it simultaneously gains in speed and small training set size. It is also in possession of one clear advantage: the absence of the need of paired training data, which comes in handy in a field where good quality training data is scarce. It proves to be a competitive alternative to already existing methodology used to solve these types of problems.

Chapter 8

Conclusion

In this thesis we have explored the capabilities of a modified cycleGAN model through testing it on subsurface data segmentation. More specifically, we have assessed its abilities to interpret seismic data in order to build a geological macro model of a given volume. As the work conducted in this thesis is inspired by Mosser et al. (2018b), some of our results can be viewed as a validation of the results published in the named paper. However, our scope spans slightly wider. We improve the deep learning tool used and perform extensive analysis on the significance of the length of training and quantity of training data, widening the general understanding of this technology further and giving a thorough demonstration of its capabilities.

To do this, we have experimented with including additional penalty terms to a vanilla cycleGAN model with the intention of denoising its generated images. For this, we used three different approaches, mainly targeting statistical properties in the images. First, we experimented with the covariance of the image distributions. The idea here is that images with less noise will have a lower covariance overall. Thus, by penalising by a weighted distance between the covariances of a generated image and a reference image, we would obtain clearer predictions. Secondly, we looked at the correlations of the images. By considering the traces of the images, we penalise by a weighted distance between the autocorrelations of the generated image and a reference image. Lastly, we experiment with the KLD between a generated image and a reference image. With KLD being a measure of the difference between two distributions, the idea is that this difference between a noisy, generated image and a reference image in general would be larger when the generated image is noisier, and thus result in a larger penalty.

Based on the conducted experiments involving all three penalty versions, we conclude that both the correlation penalty term and the KLD penalty term achieve promising results in combination with a vanilla cycleGAN, clearly reducing the amount of noise in the predicted images. Quantitatively, we find that MSE is relatively reduced with 21.5% and 19.6% for the correlation penalty and the KLD penalty, respectively. For the second part of this thesis, we choose to proceed with a model including the correlation penalty term.

Next, we test this modified cycleGAN on real seismic data from the Dutch F3 block

dataset. We evaluate how the model performs on a variety of cases related to training dataset size and the length of training sessions. We find that the more data the model is given for training, the better the quality of the predictions become. However, even though the model is given less training data (as little as 1.75% of the total dataset), it is still able to produce decent predictions across the whole volume, given that the training time is reasonably long and that the seismic is quite simple.

Ultimately, given that we train the model with very little data, and that the training time is relatively short (also compared to other deep learning methods), we still receive useful output. This is evidence of the potential value in utilising this type of model for this use case, even in practice.

The model can possibly be improved by a thorough review of its architecture, by experimenting with a different setup through replacing and changing key parts, e.g. the generator or discriminator networks, changing the existing layout, or improving the general code. Improving the code could also decrease computation time, leading to faster epochs.

Further, more penalty research may be conducted. It could be interesting to investigate whether gradient-based penalties can improve the edge detection (see e.g. Vincent et al. (2009)) and thus result in more accurate horizon interpretations. Exploring the concept of transfer learning in GANs would also be interesting. If this is successfully implemented in a model, it could lead to increased robustness in performance as it makes the model less prone to misinterpretation of unseen data. Little research has yet been conducted in this field, but promising results can be found in Wang et al. (2018) and Frégier and Gouray (2019).

An even more relevant and realistic application to this project would be to use more local data. The subsurface structures vary greatly from area to area, implying that data from the Netherlands may not be very similar to the type of data we find on the Norwegian Continental Shelf. Testing the model on data from eastern parts of the North Sea or the Norwegian sea (for instance the *Norne* dataset) may thus be of great interest in future studies.

Bibliography

Akhmadiev, R., Kanfar, R.S., 2019. Subsurface Imaging using GANs. N.p. URL: http://cs229.stanford.edu/proj2019aut/.

Alaudah, Y., Michałowicz, P., Alfarraj, M., AlRegib, G., 2019. A machine-learning benchmark for facies classification. Interpretation 7, SE175–SE187.

Arjovsky, M., Bottou, L., 2017. Towards principled methods for training generative adversarial networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net. URL: https://openreview.net/forum?id=Hk4_qw5xe.

Benaim, S., Wolf, L., 2017. One-sided unsupervised domain mapping, in: Advances in neural information processing systems, pp. 752–762.

Bhandare, A., Bhide, M., Gokhale, P., Chandavarkar, R., 2016. Applications of convolutional neural networks. International Journal of Computer Science and Information Technologies 7, 2206–2215.

Chen, X., Xu, C., Yang, X., Tao, D., 2018. Attention-gan for object transfiguration in wild images, in: Proceedings of the European Conference on Computer Vision (ECCV), pp. 164–180.

Cho, Y., Jeong, D., Jun, H., 2020. Semi-auto horizon tracking guided by strata histograms generated with transdimensional markov-chain monte carlo. Geophysical Prospecting , 1456–1475.

Choi, Y., Choi, M., Kim, M., Ha, J.W., Kim, S., Choo, J., 2018. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8789–8797.

Choi, Y., Uh, Y., Yoo, J., Ha, J.W., 2019. Stargan v2: Diverse image synthesis for multiple domains. arXiv preprint arXiv:1912.01865 .

Chu, C., Zhmoginov, A., Sandler, M., 2017. Cyclegan, a master of steganography. arXiv preprint arXiv:1712.02950 .

Dorn, G.A., 1998. Modern 3-d seismic interpretation. The Leading Edge 17, 1262–1262.

Farnia, F., Ozdaglar, A., 2020. Gans may have no nash equilibria. arXiv preprint arXiv:2002.09124 .

Franci, B., Grammatico, S., 2020. A game-theoretic approach for generative adversarial networks. arXiv preprint arXiv:2003.13637 .

Frégier, Y., Gouray, J.B., 2019. Mind2mind: transfer learning for gans. arXiv preprint arXiv:1906.11613 .

Fu, H., Gong, M., Wang, C., Batmanghelich, K., Zhang, K., Tao, D., 2019. Geometry-consistent generative adversarial networks for one-sided unsupervised domain mapping, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2427–2436.

Goodfellow, I., 2016. Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160 .

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets, in: Advances in neural information processing systems, pp. 2672–2680.

Gubner, J.A., 2006. Probability and random processes for electrical and computer engineers. Cambridge University Press.

Hahnloser, R.H., Sarpeshkar, R., Mahowald, M.A., Douglas, R.J., Seung, H.S., 2000. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature 405, 947.

Harrigan, E., Kroh, J., Sandham, W., Durrani, T., 1992. Seismic horizon picking using an artificial neural network, in: [Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing, IEEE. pp. 105–108.

He, J., Wang, C., Jiang, D., Li, Z., Liu, Y., Zhang, T., 2020. Cyclegan with an improved loss function for cell detection using partly labeled images. IEEE Journal of Biomedical and Health Informatics , 1–1?

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

Hiasa, Y., Otake, Y., Takao, M., Matsuoka, T., Takashima, K., Carass, A., Prince, J.L., Sugano, N., Sato, Y., 2018. Cross-modality image synthesis from unpaired data using cyclegan, in: International workshop on simulation and synthesis in medical imaging, Springer. pp. 31–41.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al., 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. A field guide to dynamical recurrent neural networks. IEEE Press.

Hoffman, J., Tzeng, E., Park, T., Zhu, J.Y., Isola, P., Saenko, K., Efros, A., Darrell, T., 2018. CyCADA: Cycle-consistent adversarial domain adaptation, in: Proceedings of the 35th International Conference on Machine Learning, PMLR. pp. 1989–1998.

Hossain, M.A., Sajib, M.S.A., 2019. Classification of image using convolutional neural network (cnn). Global Journal of Computer Science and Technology .

Huang, K.Y., Chang, C.H., Hsieh, W.S., Hsieh, S.C., Wang, L.K., Tsai, F.J., 2005. Cellular neural network for seismic horizon picking, in: 2005 9th International Workshop on Cellular Neural Networks and Their Applications, IEEE. pp. 219–222.

Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1125–1134.

Jaccard, P., 1901. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. Bull Soc Vaudoise Sci Nat 37, 241–272.

Jiang, S., Tao, Z., Fu, Y., 2019. Segmentation guided image-to-image translation with adversarial networks, in: 2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019), IEEE. pp. 1–7.

Karras, T., Aila, T., Laine, S., Lehtinen, J., 2018. Progressive growing of GANs for improved quality, stability, and variation, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=Hk99zCeAb.

Kaur, H., Pham, N., Fomel, S., 2019. Seismic data interpolation using cyclegan, in: SEG Technical Program Expanded Abstracts 2019. Society of Exploration Geophysicists, pp. 2202–2206.

Kindermann, R., 1980. Markov random fields and their applications. American mathematical society .

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. International Conference on Learning Representations .

Koryagin, A., Mylzenova, D., Khudorozhkov, R., Tsimfer, S., 2020. Seismic horizon detection with neural networks. arXiv preprint arXiv:2001.03390 .

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

Kullback, S., Leibler, R.A., 1951. On information and sufficiency. The annals of mathematical statistics 22, 79–86.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324.

Leggett, M., Sandham, W., Durrani, T., 1994. 3d seismic horizon tracking using an artificial neural network, in: 56th EAEG Meeting, European Association of Geoscientists & Engineers. pp. cp–47.

Lemaréchal, C., 2012. Cauchy and the gradient method. Doc Math Extra 251, 254.

Li, C., Wand, M., 2016. Precomputed real-time texture synthesis with markovian generative adversarial networks, in: European Conference on Computer Vision, Springer. pp. 702–716.

Lu, P., 2019. Deep learning realm for geophysics: Seismic acquisition, processing, interpretation, and inversion. arXiv preprint arXiv:1909.06486 .

Maas, A.L., Hannun, A.Y., Ng, A.Y., 2013. Rectifier nonlinearities improve neural network acoustic models, in: Proc. icml, p. 3.

Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Paul Smolley, S., 2017. Least squares generative adversarial networks, in: Proceedings of the IEEE International Conference on Computer Vision, pp. 2794–2802.

Maschler, M., Solan, E., Hellman, Z., Borns, M., Zamir, S., 2018. Game Theory. Cambridge University Press.

McCulloch, W.S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics 5, 115–133.

Metz, L., Poole, B., Pfau, D., Sohl-Dickstein, J., 2016. Unrolled generative adversarial networks. URL: `https://openreview.net/pdf?id=BydrOIcle`.

Mo, S., Cho, M., Shin, J., 2019. Instagan: Instance-aware image-to-image translation, in: ICLR 2019.

Mosser, L., Dubrule, O., Blunt, M., 2018a. Stochastic seismic waveform inversion using generative adversarial networks as a geological prior, in: First EAGE/PESGB Workshop Machine Learning.

Mosser, L., Kimman, W., Dramsch, J., Purves, S., De la Fuente Briceño, A., Ganssle, G., 2018b. Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks, in: 80th EAGE Conference and Exhibition 2018, European Association of Geoscientists & Engineers. pp. 1–5.

Nabian, M.A., Meidani, H., 2020. Physics-driven regularization of deep neural networks for enhanced engineering design and analysis. Journal of Computing and Information Science in Engineering 20, 1–14.

Nanda, N.C., 2016. Seismic modelling and inversion, in: Seismic Data Interpretation and Evaluation for Hydrocarbon Exploration and Production. Springer, pp. 187–204.

Nash, J., 1951. Non-cooperative games. Annals of mathematics , 286–295.

Nikodym, O., 1930. Sur une généralisation des intégrales de mj radon. Fundamenta Mathematicae 15, 131–179.

Osborne, M.J., Rubinstein, A., 1994. A course in game theory. MIT press.

Pan, J., Liu, Y., Dong, J., Zhang, J., Ren, J., Tang, J., Tai, Y.W., Yang, M.H., 2018. Physics-based generative adversarial models for image restoration and beyond. arXiv preprint arXiv:1808.00605 .

Picetti, F., Lipari, V., Bestagini, P., Tubaro, S., 2018. A generative adversarial network for seismic imaging applications, in: 88th Society of Exploration Geophysicists International Exposition and Annual Meeting, SEG 2018, pp. 2231–2235.

Pratt, L.Y., 1993. Discriminability-based transfer between neural networks, in: Advances in neural information processing systems, pp. 204–211.

Radford, A., Metz, L., Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 .

Ricker, N., 1953. The form and laws of propagation of seismic wavelets. Geophysics 18, 10–40.

Robbins, H., Monro, S., 1951. A stochastic approximation method. The annals of mathematical statistics , 400–407.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer. pp. 234–241.

Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65, 386.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., et al., 1988. Learning representations by back-propagating errors. Cognitive modeling 5, 1.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X., 2016. Improved techniques for training gans, in: Advances in neural information processing systems, pp. 2234–2242.

Sen, M., 2006. Seismic Inversion. Society of Petroleum Engineers, U.S.A.

Shen, Z., Zhou, S.K., Chen, Y., Georgescu, B., Liu, X., Huang, T., 2020. One-to-one mapping for unpaired image-to-image translation, in: The IEEE Winter Conference on Applications of Computer Vision, pp. 1170–1179.

Sheriff, R.E., Geldart, L.P., 1995. Exploration seismology. Cambridge university press.

Siahkoohi, A., Kumar, R., Herrmann, F., 2018. Seismic data reconstruction with generative adversarial networks, in: 80th EAGE Conference and Exhibition 2018.

Ulyanov, D., Vedaldi, A., Lempitsky, V.S., 2016. Instance normalization: The missing ingredient for fast stylization. CoRR abs/1607.08022.

Vincent, O.R., Folorunso, O., et al., 2009. A descriptive algorithm for sobel image edge detection, in: Proceedings of Informing Science & IT Education Conference (InSITE), Informing Science Institute California. pp. 97–107.

Von Neumann, J., Morgenstern, O., Kuhn, H.W., 2007. Theory of games and economic behavior (commemorative edition). Princeton university press.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al., 2020. Deep high-resolution representation learning for visual recognition. IEEE transactions on pattern analysis and machine intelligence .

Wang, M., Deng, W., 2018. Deep visual domain adaptation: A survey. Neurocomputing 312, 135–153.

Wang, Y., Wu, C., Herranz, L., van de Weijer, J., Gonzalez-Garcia, A., Raducanu, B., 2018. Transferring gans: generating images from limited data, in: Proceedings of the European Conference on Computer Vision (ECCV), pp. 218–234.

Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P., 2004. Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing 13, 600–612.

Wu, H., Zhang, B., Lin, T., Cao, D., Lou, Y., 2019. Semiautomated seismic horizon interpretation using the encoder-decoder convolutional neural network. Geophysics 84, B403–B417.

Wu, J.L., Kashinath, K., Albert, A., Chirila, D., Xiao, H., et al., 2020. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. Journal of Computational Physics 406, 109209.

Xie, Y., Franz, E., Chu, M., Thuerey, N., 2018. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. ACM Transactions on Graphics (TOG) 37, 1–15.

Yang, L., Zhang, D., Karniadakis, G.E., 2018. Physics-informed generative adversarial networks for stochastic differential equations. arXiv preprint arXiv:1811.02033 .

Yang, Z., Wu, J.L., Xiao, H., 2019. Enforcing deterministic constraints on generative adversarial networks for emulating physical systems. arXiv preprint arXiv:1911.06671 .

Yilmaz, Ö., 2001. Seismic data analysis: Processing, inversion, and interpretation of seismic data. Society of exploration geophysicists.

Zell, A., Mache, N., Huebner, R., Mamier, G., Vogt, M., Schmalzl, M., Herrmann, K.U., 1994. Snns (stuttgart neural network simulator), in: Neural Network Simulation Environments. Springer, pp. 165–186.

Zhu, J.Y., Park, T., Isola, P., Efros, A.A., 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks, in: Proceedings of the IEEE international conference on computer vision, pp. 2223–2232.

Appendix

Extended abstract, based on the results in the thesis, submitted for the EAGE Digital 2020 conference is shown on the next page.

Introduction

Seismic interpretation workflows tend to rely on much manual labour involving human presence for various tasks. These tasks, which could for instance entail the picking of key geological horizons from a three-dimensional seismic cube, are in many cases tedious, time-consuming and resource demanding. With technology evolving, many new tools have become available for seismic processing workflows, and in recent years there has been much focus on machine learning approaches to facilitate elements of this work process (Dorn, 1998). Semi-automated horizon picking has been around since the 1990s (Harrigan et al., 1992; Leggett et al., 1994), and with the developments in computing resources and algorithms, there is by today enormous opportunities for automising the horizon picking as well as other seismic processing tasks. Machine learning techniques will never replace human knowledge, but they can ease the interpreters role in workflows, so the professional work is spent where it is most needed.

Recent advances in deep learning research has led to the development of a promising class of methods called Generative Adversarial Networks (GANs) (Goodfellow et al., 2014). These generative nature of these methods means that they are highly capable of generating images that resemble what is seen with only limited training data, at least compared to that of other deep learning approaches. Mosser et al. (2018) showed promising results when applying GAN to seismic data set.

In this paper, we present a method for creating a geological macro model through seismic segmentation by using semi-supervised deep learning GANs. We build on the work done by Mosser et al. (2018), by transforming the problem into that of domain translation. In particular, we use a state-of-the art method for unsupervised domain translation, the *CycleGAN* (Zhu et al., 2017) model, and combine this with a penalty function for the autocorrelation in the data and in the generated images. The approach is tested on different levels of training, which in our case meands labeling or picking of geological horizons. We compare results obtained by sparse training where only five inlines and five crosslines are labeled, and rich training where twenty inlines and twenty crosslines are labeled.

GANs and Cycle GANs

The newly developed GANs consist of two neural networks that play a game against each other. As a simplification, one could say that a GAN is a neural network with a different neural network working as a loss function. One of the networks, the generator G, tries to fool the other network (the discriminator D) by generating images as close to some target distribution as possible. The discriminator evaluates both generated images and images from the target distribution, and tries to separate the generated (fake) images from the real (target distribution). Both the networks improve over time: the generator generates better replicas, and the discriminator improves its evaluation accuracy.

The cycleGAN extends the original setup by having two GANs. This has been advantageous in situations where one aims to have a domain translator between two domains X and Y. One generator G learns a forward generative model, from X to Y, while the other generator F learns an inverse generative model, mapping from Y to X. These two GANs work together by forming a cycle-consistent setup (Zhu et al., 2017). Ideally, the output resembles the input, so that for a given input $\mathbf{x} \in X$, one gets $F(G(\mathbf{x})) \approx \mathbf{x}$.

Each GAN in the setup has its own adversarial loss, so for (G, D_Y) we have

$$\mathscr{L}_{LSGAN}(G, D_Y) = E_{\mathbf{y}\sim p_{data(\mathbf{y})}}\left[(D_Y(\mathbf{y})-1)^2\right] + E_{\mathbf{x}\sim p_{data(\mathbf{x})}}\left[D_Y(G(\mathbf{x}))^2\right] \tag{1}$$

and similarly for (F, D_X). To ensure cycle-consistency, i.e. for both the forward pass $\mathbf{x}[]F(\mathbf{x})[]G(F(\mathbf{x})) \approx \mathbf{x}$ and the opposite, backward pass $\mathbf{y}[]G(\mathbf{y})[]F(G(\mathbf{y})) \approx \mathbf{y}$ the cyclic loss (Zhu et al., 2017) is defined as

$$\mathscr{L}_{Cyc}(G, F) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[||F(G(\mathbf{x})) - \mathbf{x}||_1\right] + E_{\mathbf{y}\sim p_{data}(\mathbf{y})}\left[||G(F(\mathbf{y})) - \mathbf{y}||_1\right], \tag{2}$$

where $||\cdot||_1$ denotes the *1-norm*, defined in the following way: $||\mathbf{x}||_1 := \sum_{i=1}^{n} |x_i|$ for a vector $\mathbf{x}$ of length n. Additionally, identity loss is used to preserve colour balance in the translations,

$$\mathscr{L}_I(G, F) = E_{\mathbf{x}\sim p_{data}(\mathbf{y})}\left[||G(\mathbf{y}) - \mathbf{y}||_1\right] + E_{\mathbf{y}\sim p_{data}(\mathbf{y})}\left[||F(\mathbf{x}) - \mathbf{x}||_1\right]. \tag{3}$$

The complete cycleGAN loss function is thus

$$\mathscr{L}_{CycleGAN}(G,F,D_X,D_Y) = \mathscr{L}_{LSGAN}(G,D_Y) + \mathscr{L}_{LSGAN}(F,D_X) + \lambda_C \mathscr{L}_{Cyc}(G,F) + \lambda_I \mathscr{L}_I(G,F), \quad (4)$$

where λ_C, λ_I are penalising weights.

Modified Loss Function in GANs

Because of the semi-supervised properties of the cycleGAN, i.e. the unpaired data training, no direct comparison between the generated image and its reference is possible. Hence, we propose to use statistical methods to capture the underlying similarities between the distributions of the generated and the reference images (Wu et al., 2020). In doing so, we introduce the correlation penalty to the cycleGAN model. This is used to maintain reasonable smoothness in generated images, leading to higher quality in the interpretations.

Our modified loss function is defined as

$$\mathscr{L}(G,F,D_X,D_Y) = \mathscr{L}_{cycleGAN}(G,F,D_X,D_Y) + \gamma ||R_\tau(p_{data}) - R_\tau(p_{G(z)})||_F, \quad (5)$$

where $||\cdot||_F$ denotes the Frobenius norm, summing the squared entries of a given matrix and then taking the squared root of the sum. Furthermore, γ is a penalty weight and R_τ is the correlation in the vicinity of a point in the depth direction on the seismic grid domain. Mathematically, for distance τ the correlation is defined as

$$R_\tau = \frac{1}{\hat{\sigma}^2(n-\tau)} \sum_{t=1}^{n-\tau} (x_t - \hat{\mu})(x_{t+\tau} - \hat{\mu}), \quad (6)$$

where $\{x_1, ..., x_n\}$ are measurements and $\hat{\mu}$ and $\hat{\sigma}^2$ are estimated sample mean and variance of the sequence of measurements. We compute the correlations for trace by trace, in sequential order, before summing them up, i.e

$$R_{Tot,\tau}(I) = \sum_{i=1}^{N} R_\tau(\mathbf{x}_i), \quad (7)$$

for an image I with N traces. We add each of the traveltime distances up to a specified τ_{max} to our matrix R_τ and do this for both our generated image and a reference image.

Case Study and Network Characteristics

We train the model on a dataset containing real seismic data; the publicly available *Netherlands F3 block* dataset. We also use labelled data based on this particular seismic dataset, prepared by Alaudah et al. (2019), in order to let the model learn the domain transfer. The training data thus consist of unpaired images from both a seismic dataset and a labelled macro model dataset of the F3 volume.

In a comparative study, we use a sparse dataset (5 inlines and 5 crosslines) consisting of 1.75% of the total dataset, and a rich dataset (20 inlines and 20 crosslines) consisting of 7.0% of the total dataset. The inlines and crosslines are evenly extracted from the seismic volume.

The generators in our cycleGAN model are so-called *U-Nets* (Ronneberger et al., 2015), while the discriminators are *PatchGANs* (Zhu et al., 2017). The model is implemented in Python and TensorFlow, and run on a Tesla K80 GPU through *Google Colab*.

The model is trained for up to 300 training iterations, or *epochs*. Here, one epoch takes about 13 seconds for the sparse dataset and about 35 seconds when trained on the rich dataset.

Results

We show the improvements the modified cycleGAN model with correlation loss in Figure 1. This is shown for the mean square error (MSE) for the test data, where the model is trained on the sparse data.

The proposed model leads to a relative decrease in MSE of 21.5%, due to a significant reduction of noise present in the generated images.

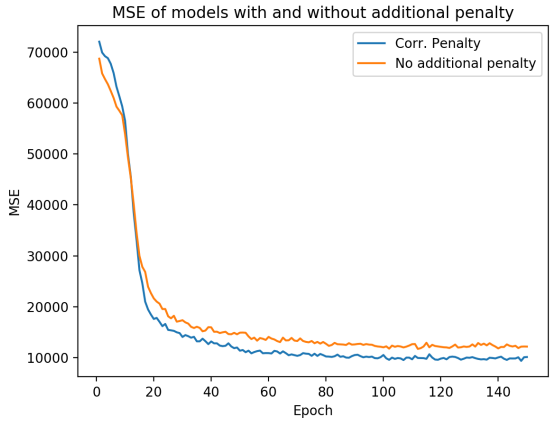
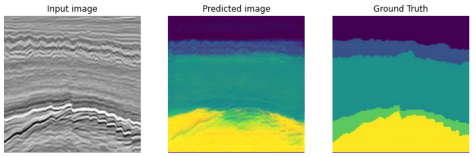
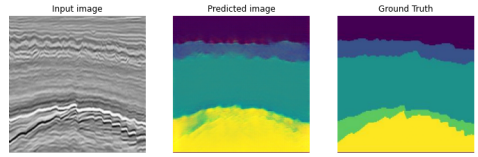


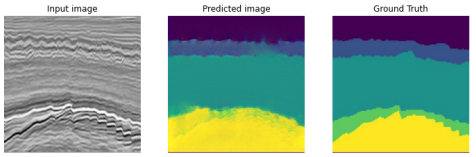
Figure 1: Comparison of mean squared error (MSE) of GAN models with and without the proposed penalty term. The correlation penalty appears to give faster convergence and better predictions.



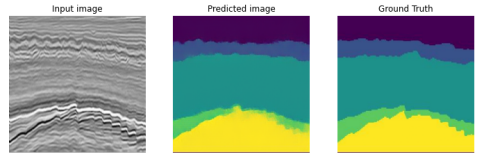
(a) Model output after 50 epochs, trained on the sparse dataset.



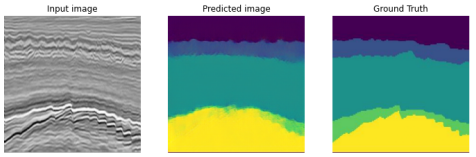
(b) Model output after 50 epochs, trained on the rich dataset.



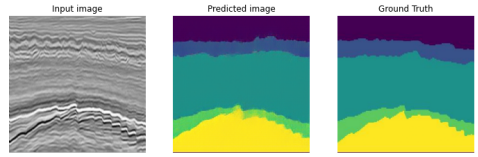
(c) Model output after 150 epochs, trained on the sparse dataset.



(d) Model output after 150 epochs, trained on the rich dataset.



(e) Model output after 300 epochs, trained on the sparse dataset.



(f) Model output after 300 epochs, trained on the rich dataset.

Figure 2: Comparison of model outputs after 50, 150 and 300 epochs. Sparse training (left) is based on 10 labeled seismic lines, while rich training (right) is based on 40 labeled seismic lines.

In Figure 2 we present results showing the actual interpretations made by the GAN model. This is shown for one test line in the seismic data sets, and the displayed results are compared to their reference images

(Ground Truth). We train the modified CycleGAN on the sparse (left) and the rich (right) training data. The model is able to obtain reasonable results even in the case with sparse data. Still, it tends to miss more detail in the geological horizons for the sparse situation. We observe that the size of the dataset has an impact on how many epochs are required to run before a decent interpretation is found. This is clearly faster when the training basis is richer.

The results show that there are different alternatives to pursue when seeking decent quality in model output, either by including more data in the training or to let the model train longer. We believe that the method presented here can be used as an aid in building geological macro models because of its low computation cost and rapid predictions.

Conclusions

We have presented a method to perform seismic inversion using a semi-supervised approach only trained on a small dataset. The method presented here is a modified cycleGAN model where we have added an additional correlation penalty term to further enhance interpretation quality. The trained model allow for very fast macro model building, given a seismic volume. The model proves to be robust, yielding accurate interpretations on unseen seismic data. This is a demonstration of what these semi-supervised deep learning methods are capable of.

Acknowledgements

We thank the Norwegian Research Council and the industry partners of the GAMES consortium at NTNU for financial support (grant No. 294404).

References

Alaudah, Y., Michałowicz, P., Alfarraj, M. and AlRegib, G. [2019] A machine-learning benchmark for facies classification. *Interpretation*, **7**(3), SE175–SE187.

Dorn, G.A. [1998] Modern 3-D seismic interpretation. *The Leading Edge*, **17**(9), 1262–1262.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. [2014] Generative adversarial nets. In: *Advances in neural information processing systems*. 2672–2680.

Harrigan, E., Kroh, J., Sandham, W. and Durrani, T. [1992] Seismic horizon picking using an artificial neural network. In: *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 3. IEEE, 105–108.

Leggett, M., Sandham, W. and Durrani, T. [1994] 3D Seismic horizon tracking using an artificial neural network. In: *56th EAEG Meeting*. European Association of Geoscientists & Engineers, cp–47.

Mosser, L., Kimman, W., Dramsch, J., Purves, S., De la Fuente Briceño, A. and Ganssle, G. [2018] Rapid seismic domain transfer: Seismic velocity inversion and modeling using deep generative neural networks. In: *80th EAGE Conference and Exhibition 2018*, 2018. European Association of Geoscientists & Engineers, 1–5.

Ronneberger, O., Fischer, P. and Brox, T. [2015] U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

Wu, J.L., Kashinath, K., Albert, A., Chirila, D., Xiao, H. et al. [2020] Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. *Journal of Computational Physics*, **406**, 109–209.

Zhu, J.Y., Park, T., Isola, P. and Efros, A.A. [2017] Unpaired image-to-image translation using cycle-consistent adversarial networks. In: *Proceedings of the IEEE international conference on computer vision*. 2223–2232.

NTNU

Norwegian University of
Science and Technology