UiO : Department of Geosciences
University of Oslo

Attenuation of Seismic Interference Noise with Convolutional Neural Networks

An attempt to improve the efficiency of seismic processing

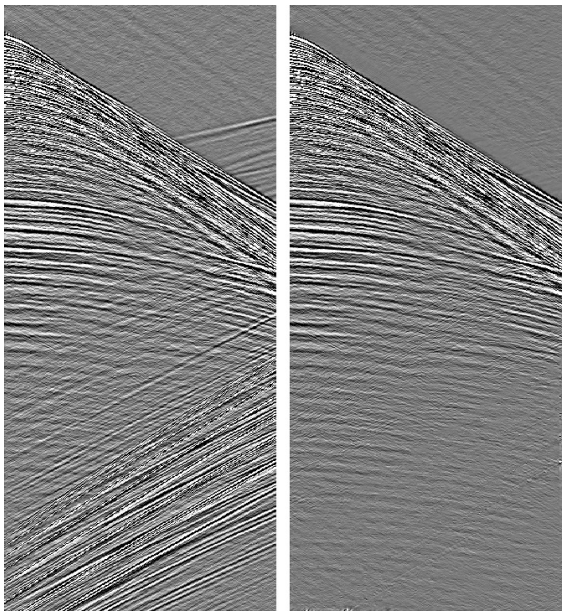
Sigmund Slang

Master's Thesis, Spring 2019

Attenuation of Seismic Interference Noise with Convolutional Neural Networks

An attempt to improve the efficiency of seismic processing

Sigmund Slang



© 2019 Sigmund Slang

Attenuation of Seismic Interference Noise with Convolutional Neural Networks

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Acknowledgements

This thesis work was done in collaboration between the University of Oslo and CGG and I thank CGG, CGG MCNV and Equinor for data used in this thesis. Network figures are inspired by work done by Haris Iqbal, and I thank him for letting me utilize his work.

Firstly, I would like to thank my supervisors Prof. Leiv J. Gelius, Dr. Thomas Elboth and Steven McDonald. Weekly meetings with Prof. Leiv Gelius, at the Department of Geosciences, the University of Oslo, has brought constructive criticism, broad knowledge and strict deadlines which helped me structure my work and thesis. Numerous days has been spent at the CGG office. Dr. Thomas Elboth has used countless hours discussing and brainstorming problems with me. His enthusiasm and knowledge has been inspiring and helpful for the thesis work. Steven McDonald has helped a lot with his broad knowledge in seismic data interpretation and processing.

Secondly, I would like to thank Jing Sun and Thomas Greiner for countless discussions regarding machine learning, which have helped me gain new insight to the field and pushed me to a higher level. Monthly meetings with both and many hours at CGG spent with Jing has helped this thesis become what it is.

Finally I would like to thank friends and family for moral support and good company during this process. A special thanks to my peers for making countless late nights at the university fun and bearable. And Benedicte, who has been cheering me on this entire time, keeping my spirit up when the days got long and the nights short.

Sigmund Slang
May, 2019

Abstract

The aim of this thesis has been to look at the possibility of using a convolutional neural network to attenuate seismic interference (SI) noise in marine seismic data. Modern SI-denoising algorithms employed by the industry are efficient and normally yield very good results. However, they are often time consuming. Using neural networks to do real time denoising could significantly improve denoising efficiency, and thereby save time and money for processing companies.

The dataset used in this thesis consisted of two lines of marine seismic field data, where one line was denoised marine seismic data and the second line contained "pure" seismic interference noise recorded in 2015 in the North Sea. These were combined to create datasets needed to train the convolutional neural networks.

Four different models were applied in this thesis: Classification CNN, Autoencoder, No Downscaling CNN and U-NET. The Classification CNN was implemented asa proof of concept to test whether the network could differentiate between noise-contaminated data and clean data, which yielded good results. The autoencoder gave poor denoising results, showing significant loss of geological signal. Both the No Downscaling CNN and the U-NET gave good results, where U-NET performed best. It performed good denoising, leaving only a very small residual in special cases of conflicting dip. The network required approximately 0.02s for denoising a shot gather, proving that real time denoising of marine seismic data is possible with the use of convolutional neural networks.

The results from this thesis has been published as an Expanded Abstract and accepted for oral presentation at the EAGE Annual 2019. The abstract is included in Appendix B. This thesis has also contributed to a journal paper entitled *A convolutional neural network approach to deblending* submitted to Geophysics (status is moderate revision) (Sun et al., 2019).

Abbrevations

- AE Autoencoder
- ANN Artificial Neural Network
- AVO Amplitude versus offset
- CC Cross-Correlation
- CNN Convolutional Neural Network
- CV Convolutional
- FC Fully connected
- GPU Graphical processing unit
- HPF Hydrostatic pressure fluctuations
- ML Machine Learning
- NDCNN No Downscaling Convolutional Neural Network
- PSM Pre-stack migration
- ReLU Rectified linear unit
- SI Seismic interference
- SNR Signal-to-noise ratio
- TWT Two-way travel time

Contents

1 | Introduction

This thesis is about the use of machine learning to attenuate noise in marine seismic data. However, before discussing seismic data in any detail, we will start off with an introduciton to machine learning. Machine learning (ML) is a popular term these days. The media mention artificial intelligence (AI) as the new future, while others are more sceptical, afraid for what it might become. Everyone has watched movies with robots that are able to think by themselves. Machine learning is often seen as this complex, difficult and abstract concept. This used to be true, but there has been major changes the last few years. Machine learning is currently used by all major internet platforms, such as: Google, Facebook, Instagram, Amazon, etc. These platforms utilize machine learning in many different ways, but one of the most common is embedded in the search platform of Google.

Google has in recent years developed a software package called Tensorflow (Abadi et al., 2015). This is an open source package tailor made for machine learning, compatible with multiple well known programming languages. Tensorflow has a relatively high user threshold, which has opened up for wrappers, such as Keras (Chollet et al., 2015), to use mahine learning libraries as back-end. All of these packages and libraries has opened the world of machine learning to everyone. If you have access to a computer, you can start with machine learning for free with relative ease. The recent availability of powerful GPUs has also opened for the possibility of running high demand jobs, even at home. Open source software and powerful hardware is the main reason for the explosion of interest in machine learning. This thesis will use Keras with a Tensorflow back-end to run a specific type of machine learning, namely Convolutional Neural Networks (CNNs).

An important thing to mention is the current limitations of machine learning. Machine learning is domain specific. It is possible to handle specific tasks, if designed and used correctly, with high precision. Today it is, however, diffucult or even impossible to create general domain machine learning. To put this into perspective, you design a network which can remove certain types of noise from images and gain good results. This has been done multiple times and is the basis for this thesis. Designing a network which can handle speech, talk back, organize files, brew your

daily morning coffee and find 5 news articles of choice at the same time is however limited. All these tasks can be done separately, but creating a general purpose network is still a task yet to be done. When AI is a reality, this task will maybe be feasible, but it is likely that a few decades will pass before the human kind will come anywhere near creating an actual AI.

1.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are networks which learn the structure of input data and execute desired tasks with this data (Gershenson, 2003). The input data can be what ever the user desires such as: pictures, text files, spread sheets, music or other sound signals etc. This might seem confusing, so let us use some real world examples. Assume you are working in an office which has thousands of pages with important data stored in boxes. This data does not exist in digital format and is to be digitized and uploaded to a filing system and then stored in respective folders. If this task has to be done by human hands, all the data has to be scanned and then looked over, manually page by page. Imagine the hours needed for this task to be completed. Instead of having humans do the sorting, computers can be trained to do this automatically. All the data can be digitized and the computer does the rest.

Another usage of ANNs can be image classification. According to reports from Cisco (2018), approximately 80% of the world's internet traffic will be video by 2021. Video is simply moving images, and can thus be interpreted as such by computers. To be able to monitor all this activity, computers need to learn the format, learn what to look for in images. Assume a case where a person is given two images, one of a cat and one of a dog. The human brain will comprehend what these images contain reflex-like without any effort. The person will have no problem labeling the images, choosing which contains a dog and which contains a cat. If a computer is presented with the same problem, it has no understanding of what to look for to tell whether it is a cat, a dog or even an airplane. It can, however, be trained to learn characteristics of specific shapes. Training an ANN is therefore needed for it to work desirably. The theory and structure of an ANN is complex, but the essence of it is rather simple. Data is fed into the network and the network deals with this data accordingly. Assume the task is to classify cats and dogs as mentioned above. If a picture of a cat is fed into the network, we want it to label this image as a cat

and vice versa. This thesis will revolve around the usage of Convolutional Neural Networks (CNNs) for image processing.

1.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are the most commonly used architecture for image processing. A CNN, or any type of neural network, is a cluster of nodes connected in a certain pattern. This pattern can be viewed as a layered model where each layer has a certain amount of nodes. A colored (RGB) image in the aspect of human eyes is simply regarded as a two dimensional problem, while a computer regards this type of images in three dimensions: red, green and blue. This is because each color pixel has its own value, meaning each layer in a CNN has to be three dimensional in the case of RGB images. One might then understand the complexity of image recognition in the "eyes" of a computer.

The main idea behind the concept of a CNN originates from work done by Hubel and Wiesel (1959). They experimented with the eyes of anesthetized cats and monkeys to understand how the brain reads information from the visual cortex. They placed a micro electrode in close proximity to a visual axon (neuron) to measure it's action potential. The nervous system uses electric signals and an axon is all or non signal (on/off) changing in frequency. This signal was recorded by the electrode as they introduced various shapes in the receptive field of the animals. The ground breaking discovery they made was proving that different axons respond to different shapes. Certain axons only fired when presented to horizontal edges or shapes, while giving no signal when presented to vertical edges and vice versa for other classes of axons. Kuffler (1953) showed how axons responded well to small spots of light, but not big spots. This is because the axons has a sensitive center, meaning they work locally.

These axons can be compared with the nodes in a CNN where different parts of a CNN catches different features of an image. Similar to axons, a CNN work locally on an image instead of reading everything at the same time. If multiple nodes are connected in a network, such as the axons in the human eye, the entire network will be able to understand complex shapes. A perfectly designed network still require training to be tuned to a specific purpose. To be able to understand how CNNs are trained, these layers has to be visualized and understood. Figure 1.1 visualizes a small scale classification network consisting of three layers. Assume these layers

are respectively: Input layer, hidden layer and fully connected output layer. It is, however important to notice that this is just a sample network and is not how all CNNs will be structured.

Input layer

The input layer is often a convolutional layer in a CNN. A convolution is, briefly explained, a mathematical operation on two functions which explains how much the two functions overlap at a chosen point in a given domain. The convolution process in a convolutional layer will be explained in Chapter 3. A convolutional layer has filters sliding accross the input image. These filters changes the shape and size of the input image and the machine learns different aspects of the data for each filter sliding over the data. Since the convolutional layer and the input layer in this simple CNN (Figure 1.1) is the same, the size of the layer equals the size of the image. This size is the width and height of the image as well as the last dimension, namely the color depth. For a normal RGB image, this depth is 3.

Hidden layer

The next layer in the CNN is the hidden layer. The reason this layer is referred to as hidden is not concise, but mainly because it is hidden from the user. The user only accesses input and output layer, thus making the intermediate layer(s) hidden. CNNs are often larger than the simple model presented here, where multiple hidden layer with different parameters and functions operates differently on the data. The hidden layers estimates the output as a function of the input data. However, the function is unknown, since it is learned by the computer as it is trained.

Fully connected layer: output layer

The last layer is the fully connected (FC) layer. As the network name suggests, each node in the FC layer is connected to every node of the previous layer. The problem mentioned earlier has two possible outcomes: cat or dog. These are referred to as classes. The FC layer takes the output of the precursor layer and computes the probability of the input image being the different classes of the network; a cat or a dog. The output volume of this layer becomes an array of size 1x1x2 since there are 2 classes to choose from in the problem. If the output looks like the following:

[.90 .1] the network has predicted the image to be class number 1, a cat, with a 90% probability and class number 2, a dog, with a 10% probability.

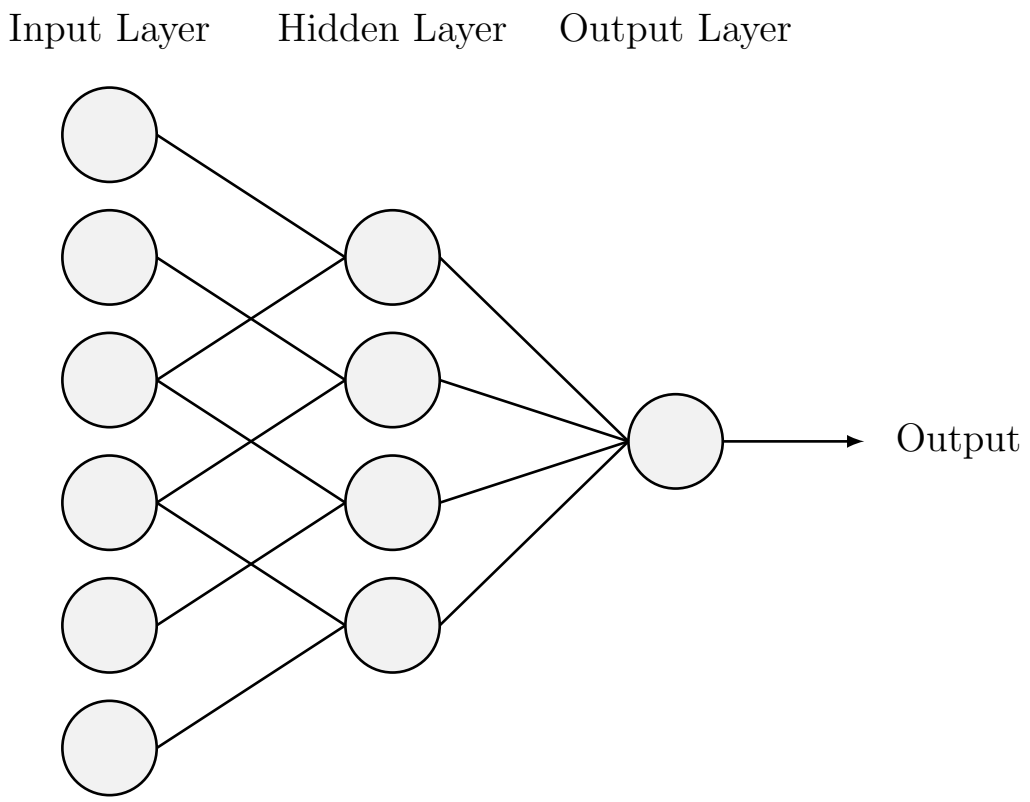


Figure 1.1: *A small scale classification CNN were the first layer is a convolutional layer. All nodes are not connected between layer one and two since convolutional layers work locally.*

Training

The CNN has no way of knowing whether the image is a cat or a dog without being told which is which. The way the computer learns what to look for is by feeding it data which has already been classified. The network gets multiple images of cats and dogs, then tries to tell which class each image belongs to. After it has finished an iteration over the dataset, the real labels are presented to the network and the network optimizes its own parameters accordingly. This process is repeated until the FC layer reaches acceptable accuracy. When the user is pleased with the result, the network should in theory be able to handle other images which has not yet been classified. This is a simplification of the truth, but the concept has a wide range

of uses. This thesis will not use pictures of cats and dogs, but seismic data. The classification approach was explained to ease the understanding of a CNN, but the denoising of seismic data is a regression problem which will be explained in section 3.1.

1.2 Seismics and seismic processing

In this work seismics and seismic processing is understood to be a field in geophysics related to the Earths crust. The definition of seismics used here is referring to the acquisition and processing of seismic data. Seismic data is detailed data of the underground acquired by sending acoustic energy through subsurface layers, whether this be on land or at sea (Musset and Khan, 2009). This data allows for visual representation of deep geological structures and is the main method used by oil companies in the search for hydrocarbon reservoirs. The reflection of the acoustic energy in the subsurface allow us to build up an image of the underground. Since this data is acquired by the use of acoustic energy, the data is always contaminated by noise and requires noise removal in pre-processing steps to be interpreted correctly. Seismic data processing is a large field in geophysics. The processing workflow of seismic data has become more demanding and more complex over the last few years, thus requiring high competence and large amounts of computing power. It is therefore a costly procedure to process seismic data with industrial quality. This thesis uses convolutional neural networks to denoise marine seismic data.

1.2.1 Marine seismic acquisition

Seismic acquisition is the process of generating and recording seismic data. Marine acquisition are typically executed by towing sources and streamers equipped with receivers after a seismic vessel at sea, or by placing them on the earth's surface (Schlumberger, 2017). This thesis uses data collected from marine seismic vessels towing hydrophones, Figure 1.2. As can be seen in the Figure, the energy source generates acoustic energy which is transmitted though the water column down to the seafloor. This energy passes through the subsurface layers and is reflected back to the acoustic receiver arrays which are recording the data. These cables are typically $6 - 10$km long with one sensor every 12.5m per cable. The vessels usually tow $12 - 14$ cables. The normal sampling interval of a seismic survey is 2ms resulting in

big amounts of data generated. An approximation of data generated per 24 hours with one component is:

$$(N_{\text{Channels}} * N_{\text{Cables}} * 500 \text{ samples/s} * 32 \text{ bits} * 3600 * 24)/8 \approx 1.5 \text{ Tb/day} \quad (1.1)$$

These vessels have an estimated operating cost of around 200 000 US $/day making marine seismic acquisition surveys a costly procedure. Because of the cost of surveys, there are on-board staff processing the data in real time, making sure all data is usable. Invalidated data might contain a high amount of noise, rendering it useless. The noise contaminated data might have to be reacquired resulting in a loss of time/profit for the acquisition company. The on-board staff are mainly there to quickly assess the need for re-acquiring data while still in the area.

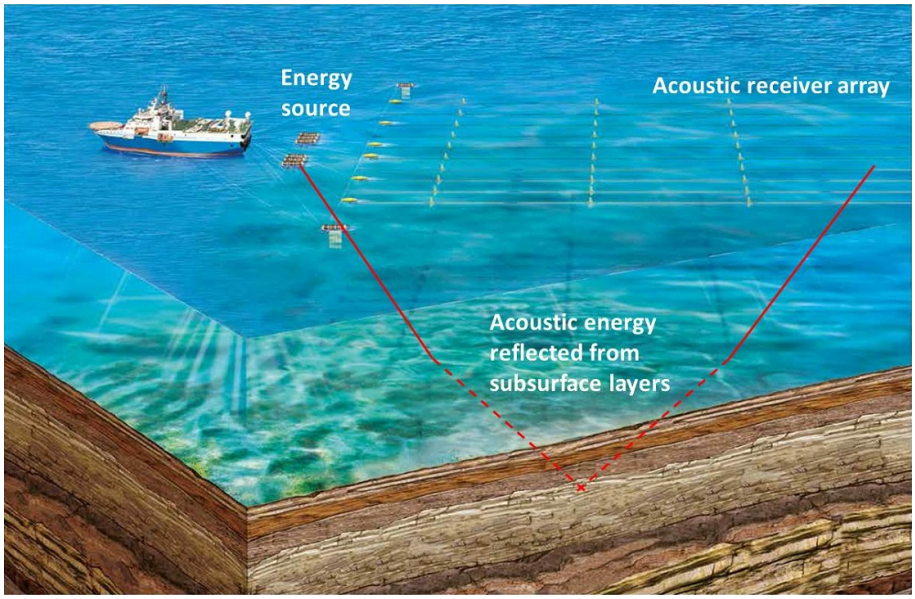


Figure 1.2: *Seismic acquisition (CGG, 2018)*

1.3 Motivation - Definition of thesis

Many types of noise contaminate seismic data such as: swell noise, ambient noise, electric noise and linear noise. Seismic surveys are, as mentioned, a costly procedure, and it would be advantageous if the different categories of noise could be automatically detected and removed. The thesis work was to be carried out in collaboration with CGG in Oslo. During the last few years CGG has made a large number of noise recordings where they have collected various forms of seismic data noise.

The idea of this thesis is to use seismic interference noise records together with normal "noise free" data as a training set for convolutional neural networks and the noise will be attempted removed automatically. This concept is not completely new, but recent papers using CNNs for denoising tend to have the same problem where they use synthetic data, data with limited dynamic range or random noise attenuation (Baardman, 2019; Li et al., 2018; Si and Yuan, 2018; Jin et al., 2018). The data used in this thesis is seismic field data with full dynamic range. Quick and efficient noise removal can potentially increase efficiency on-board acquisition vessels and might lower the need for on-board processors. It might be useful in on-shore processing as a robust and quick way for quality control and improving the quality of the data. These factors may therefore save both time and money for the industry.

Furthermore, it is worth to mention that if we can show that a neural network efficiently can attenuate seismic interference noise, the same network can probably also be trained to tackle other types of noise. It should in if fact also be applicable for processes like deghosting and demultiple. As such, neural network seismic data-processing could become a very valuable approach in the not too distant future.

2 | Noise

Recorded seismic data always contain various types of noise. Seismic noise is essentially all type of undesirable signal in the seismic data. Seismic noise can appear in different patterns with different frequencies and amplitudes. Marine seismic noise can be classified into different categories, all based on the point of reference. In this thesis, the classifications used are: Coherent and incoherent.

2.1 Incoherent noise

Incoherent seismic noise is noise appearing more or less randomly in the data and may be referred to as 'random' noise. Incoherent noise has little or no correlation with neighbouring channels, nor along the same channel, meaning it has a low spatial and temporal correlation (Kumar and Ahmed, 2011). The lack of correlation made stacking a common way to remove incoherent noise. Stacking is, however, obsolete and has been exchanged with various approaches of noise filtering (Sanchis and Hanssen, 2011; Chen and Sacchi, 2015). These methods are more suitable for the industry as most processing steps are done pre-stack.

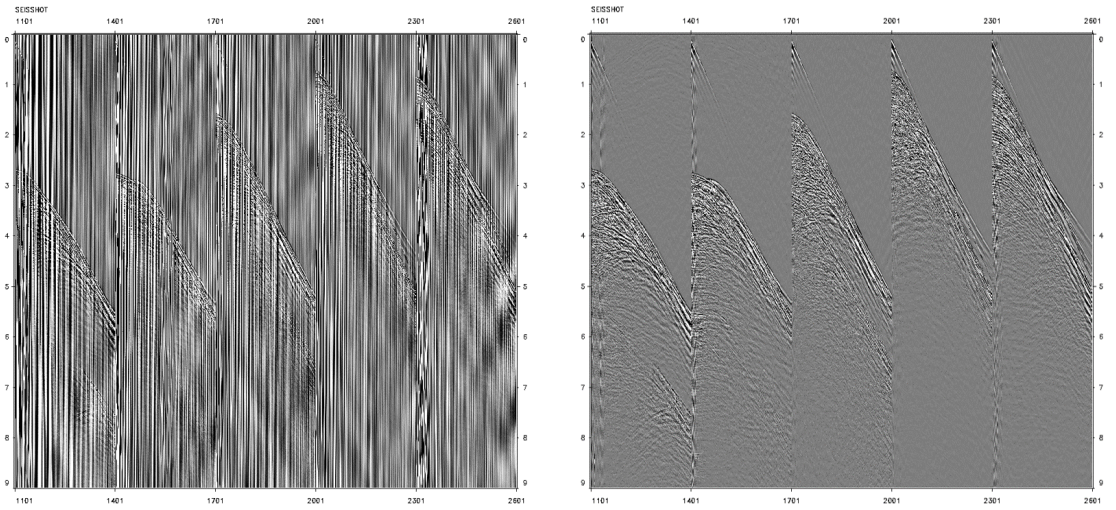
2.1.1 Swell noise

Swell noise is a type of marine seismic noise which is mainly generated by sea surface waves (Elboth et al., 2009b). It can, in certain exceptions, show signs of coherency, but is mainly incoherent. Swell noise is a broad term and can be divided into two different types of noise, dependant on how it is generated.

Hydrostatic Pressure Fluctuations

The most common type of swell noise is generated by hydrostatic pressure fluctuations (HPF) which are low frequency longitudinal sea surface waves generated by rough weather conditions, called swells. This specific type of swell noise will be referred to as HPF generated noise and is given by Kundu (1977) as:

$$w(x,z) = A\omega e^{-kz}\sin(kx - \omega t) \tag{2.1}$$



(a) *Swell noise* ***(b)*** *Swell noise removed*

Figure 2.1: *Figure (a) shows consecutive shotgathers contaminated with high amounts of swell noise generated by hydrostatic pressure fluctuations. Figure (b) shows the same shotgathers after application of a low-cut filter. The swell noise as seen in Figure 2.1a is completely removed, rendering clear seismic shot gathers.*

where z is depth, A is the amplitude of the surface waves, k is the wave number and ω is the angular frequency (Elboth et al., 2009b). Swells have a typical amplitude of approximately $1 - 10$m in open sea, a wavelength of approximately 100m, and a period of approximately 8 seconds. These hydrostatic pressure fluctuations generate the most dominant type of noise to contaminate marine seismic data, which can be regarded as random incoherent noise (Dondurur and Karslı, 2012). The reason these waves generate so much noise is due to the large amplitudes of the waves. The pressure change registered by the receivers when oscillating ± 0.01m is large compared to pressure from reflected sub-surface waves. The effect of a low-cut filter in the Fourier domain can be viewed in Figure 2.2 where the red and blue line represent the seismic data plotted respectively before and after application of a low-cut filter.

Vortex swell

The second type of swell noise is generated by interactions between the streamer and mounted equipment, and the water. This causes vibrations in the streamer. The vessel is slowly oscillating up and down which creates a transversal wave propagating down through the receiver cables. These waves typically have an amplitude of a

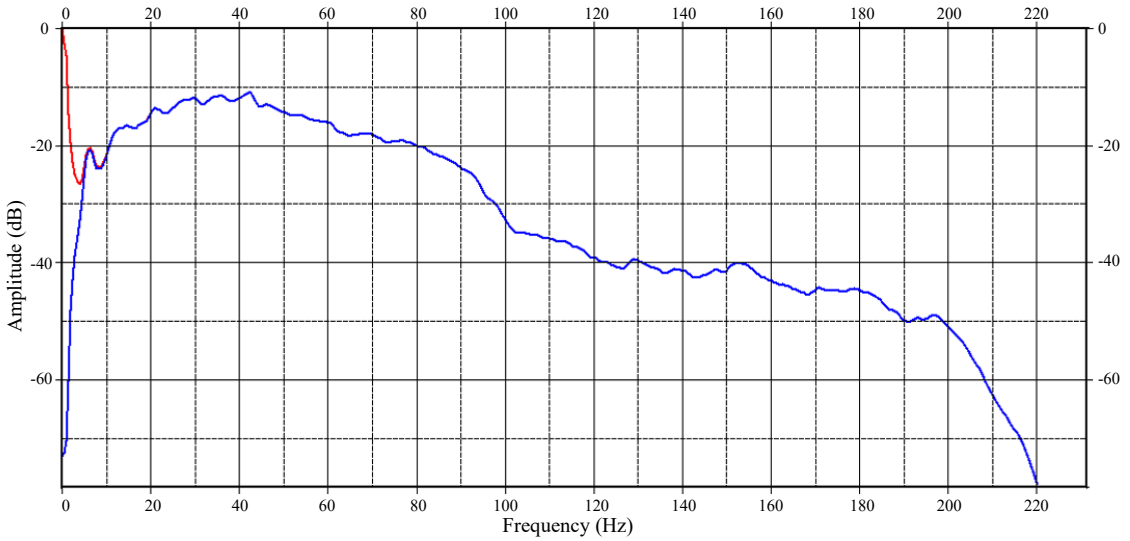


Figure 2.2: *Fourier spectrum of a seismic shot illustrating the removal process of hydrostatic pressure noise. Red line is the data before low-cut filter is applied and the blue line is after.*

few cm and a phase velocity of approximately 50m/s in the transversal direction and approximately 1500m/s in the longitudinal direction. The oscillation of the cables create a type of vertices called Von Karman vertices. These vertices travel in the water and hit neighbouring receivers, thus resulting in a change of pressure. This pressure change is recorded as a type of swell noise, further referred to as vortex swell. The characteristic appearance of vortex swell noise can be seen in Figure 2.3 where it appears as spikes (Elboth et al., 2009a). As can be seen in this figure, the vortex swell has affected multiple traces in the same shot gather, but without much coherency. Previous or later shot gathers will therefore, if vortex swell noise is present, have spikes at different areas. The frequency range of vortex swell noise is approximately $0-10$Hz which makes it difficult to remove it by low-cut filters, because this frequency band also contains important geological data. This type of swell noise is therefore typically removed with sliding F-X filtering in a statistical approach as can be seen in Figure 2.4. Vortex swell can be categorized as random incoherent noise, but since it is generated by an interaction between the streamers and the turbulence, it can in theory be anticipated and reduced. Contractor companies are therefore continuously working on ways of reducing this type of motion, thus reducing the generation of vertices in the water column.

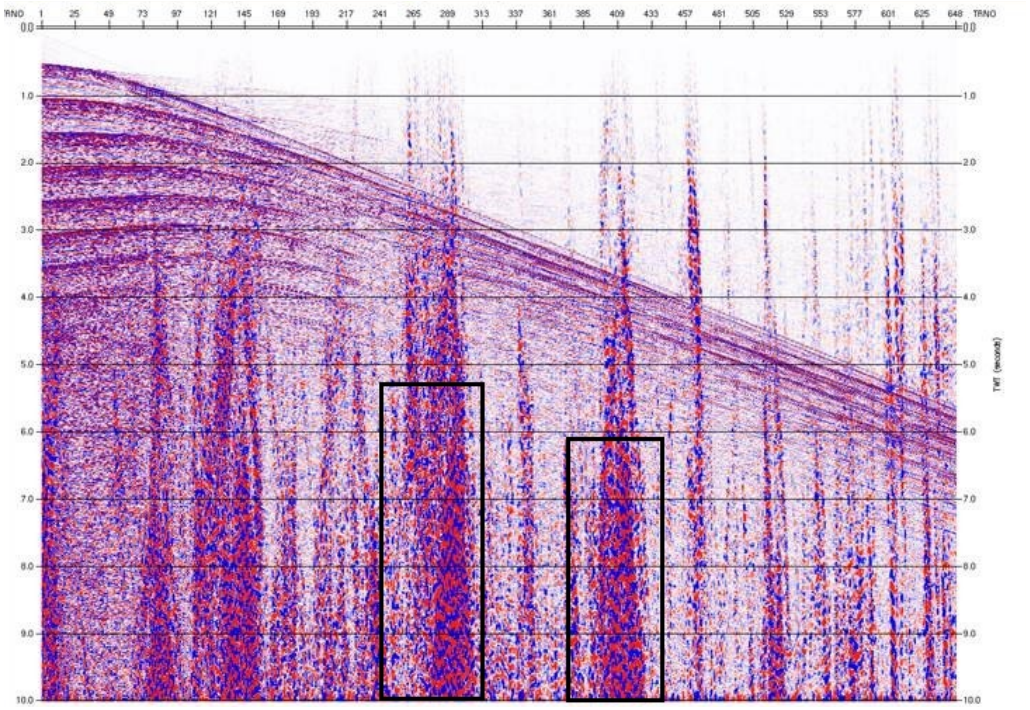


Figure 2.3: *Figure illustrating high contamination of swell noise. Examples of swell noise are marked in black boxes.*

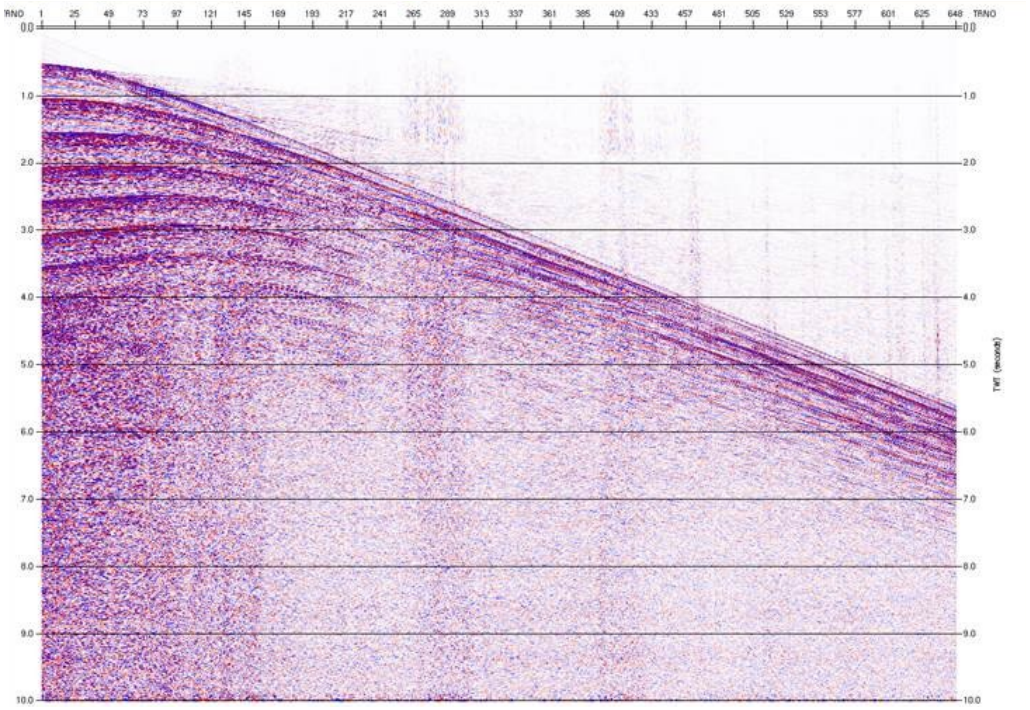


Figure 2.4: *Figure 2.3 after swell noise removal. There are still some residual noise apparent in the Figure.*

2.2 Coherent noise

Coherent seismic noise is noise appearing in some sort of coherency between channels, streamers or in time. If the frequency spectrum of the noise is significantly different from the geological signal, frequency filtering might suffice. There are other different approaches such as multi-channel filtering or spatial averaging for denoise removal, but coherent noise tends to need a more sophisticated approach (Larner et al., 1983).

2.2.1 Tugging noise

Tugging noise can also be regarded as a type of swell noise, but it is very coherent. It is generated by longitudinal tugging from the vessel and from the tail buoy (Elboth et al., 2009a). This type of tugging noise can be seen in Figure 2.5 where it appears as linear events near the edges of the data. It is normally most visible on the first and last sections of the streamer, closer to the origin. Longitudinal tugging has a moveout of approximately 1500m/s and a broadband frequency range of approximately $0-200$ Hz making standard cut-off filtering impossible to use. A common way to remove logitudinal tugging noise is to transform contaminated common channel data to $\tau - p$ domain and mute sections containing tugging noise. The muted areas are then transformed back to time-offset domain and (adaptively) subtracted from the original data.

2.2.2 Seismic interference noise

Marine seismic interference (SI) noise is a type of coherent noise occurring when energy from nearby marine seismic source vessels are recorded. This noise tend to be well preserved over larger distances (Jansen, 2013). SI noise is a well known problem for seismic contractor companies and causes artifacts in the data. These artifacts are coherent noise patterns and have a characteristic appearance as linear events with high amplitude (Akbulut et al., 2005). Figure 2.7 shows SI for five consecutive shot records. The characteristic linear events are clearly visible as they appear at different arrival times in each record. The angles of incidence for SI noise might differ from survey to survey depending on the relative placement of its origin to the receiver. Because of this, SI noise might overlap with sub-surface layer reflections which often have significantly lower amplitudes. SI noise might therefore

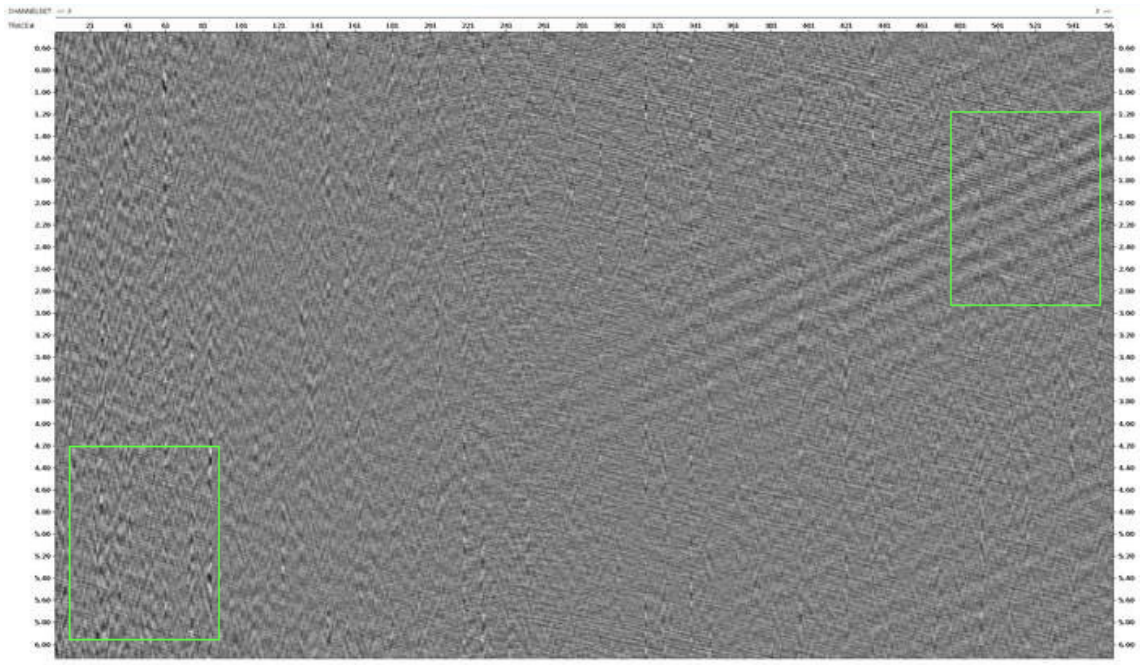


Figure 2.5: *Tugging noise in seismic data. There is tugging noise present both from vessel and tail buoy marked in green squares*

be harmful to processing operations such as deghosting, demultiple, velocity estimations and amplitude versus offset (AVO) analysis (Gulunay et al., 2005). The common workflow for SI removal is illustrated in Figure 2.6. Shotgathers containing SI noise is transformed to $\tau - p$ domain and further sorted to common-p gathers. Each p-value is then iterated over for each common-p gather using a denoising algorithm of choice. When the iteration is complete the data is sorted back to $\tau - p$ and further transformed back to time-space. The data presented in time-space domain is just SI noise which is then subtracted from the original data, thus removing the SI from the real data using an adaptive formulation.

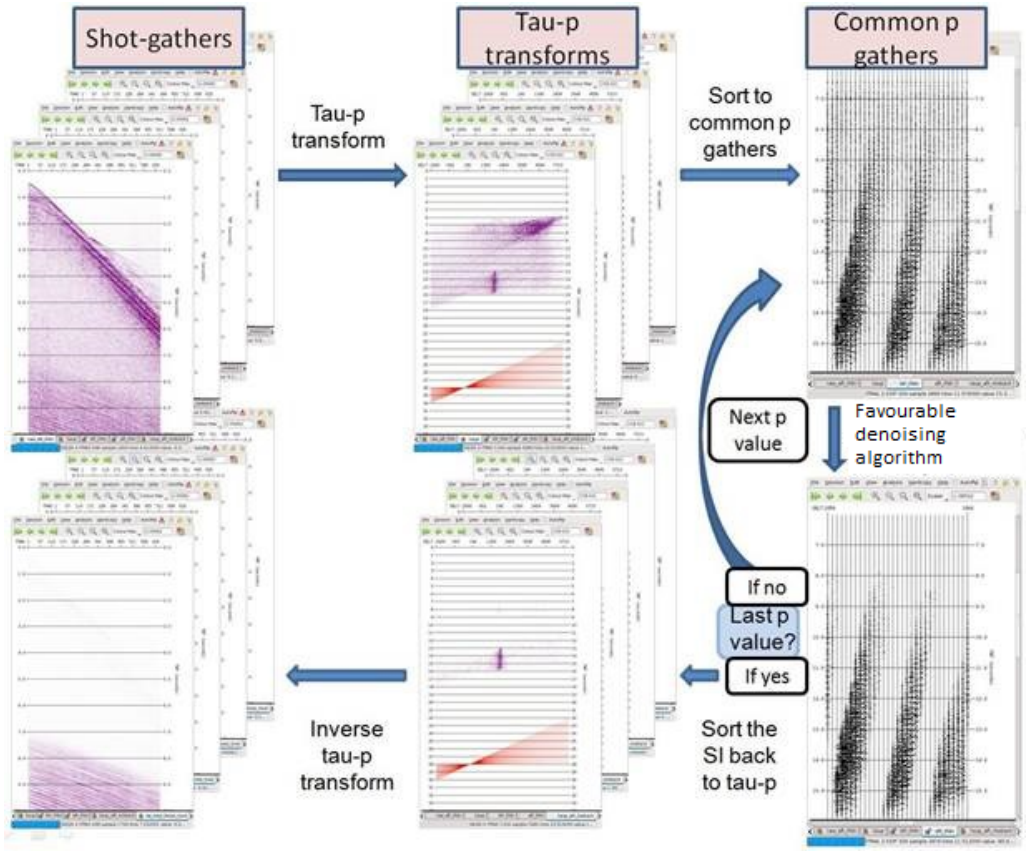


Figure 2.6: *Workflow chart illustrating the process of removing SI (Jansen, 2013).*

According to Akbulut et al. (2005) and more recently Laurain et al. (2015) it has been common for contractor companies to schedule a time-sharing plan due to high levels of SI. This type of plan is a costly measure, since it can cause substantial downtime, but have become better coordinated in the later years. Seismic vessels operating in the North Sea at present date has a downtime of a few percent, reducing the cost compared to the Mexico Gulf in the mid eighties. Although such a system causes little downtime and reduces the most harmful SI-noise, there is still SI-noise present. The most harmful SI-noise is broadside noise, according to processors at CGG. SI-noise intercepting from the side have a similar moveout as reflections meaning there are almost no kintematic difference. Seismic interference can be recorded within a large proximity to the recording vessel. The most harmful SI-noise originates from less than approximately 40 km away, explaining why the problem remains.

Move-out of seismic interference

Seismic interference (SI) noise mainly travels in the water column with the seafloor and sea surface acting as reflectors. Small parts of the energy may propagate in sub-surface layers, but will make little to no impact in the data due to high attenuation factors. SI noise has a rather distinguishable appearance, mentioned in previous section, but this strictly linear structure only yields when the origin of noise is further away than approximately 40km. For SI noise events originating from sources within a proximity of approximately 40km, the events appear curved. Figure 2.7 illustrates five SI-noise contaminated shot gathers with varying angle and distance to the source vessel. Shot number one and two show SI noise coming from abeam. This noise is almost linear due to the distance (40+ km) between the source and reciever vessel. The alignment of the noise is similar to the seismic signal since they originate from the same direction. Shot number three shows two white boxes visualizing SI-noise coming from abeam at a relative close proximity to the ship (15-20 km). There are some curvature present which might align with the seafloor data. This type of SI noise is difficult since it kinematically appear similar to reflection data. Shot number 4 shows SI coming from the side of the boat within close proximity (6 km). The noise appears with much curvature due to the close distance of the source vessel. The amplitude of the noise is high and may mask the underlying geologic reflection data. Shot number five show SI coming from ahead with a large distance (40+ km) to the source vessel. The noise appears similar to shot number one, but mirrored along the depth direction due to distance of origin. SI noise traveling in the water column has various velocity depending on level of salinity in the water, but it can be approximated to 1500m/s.

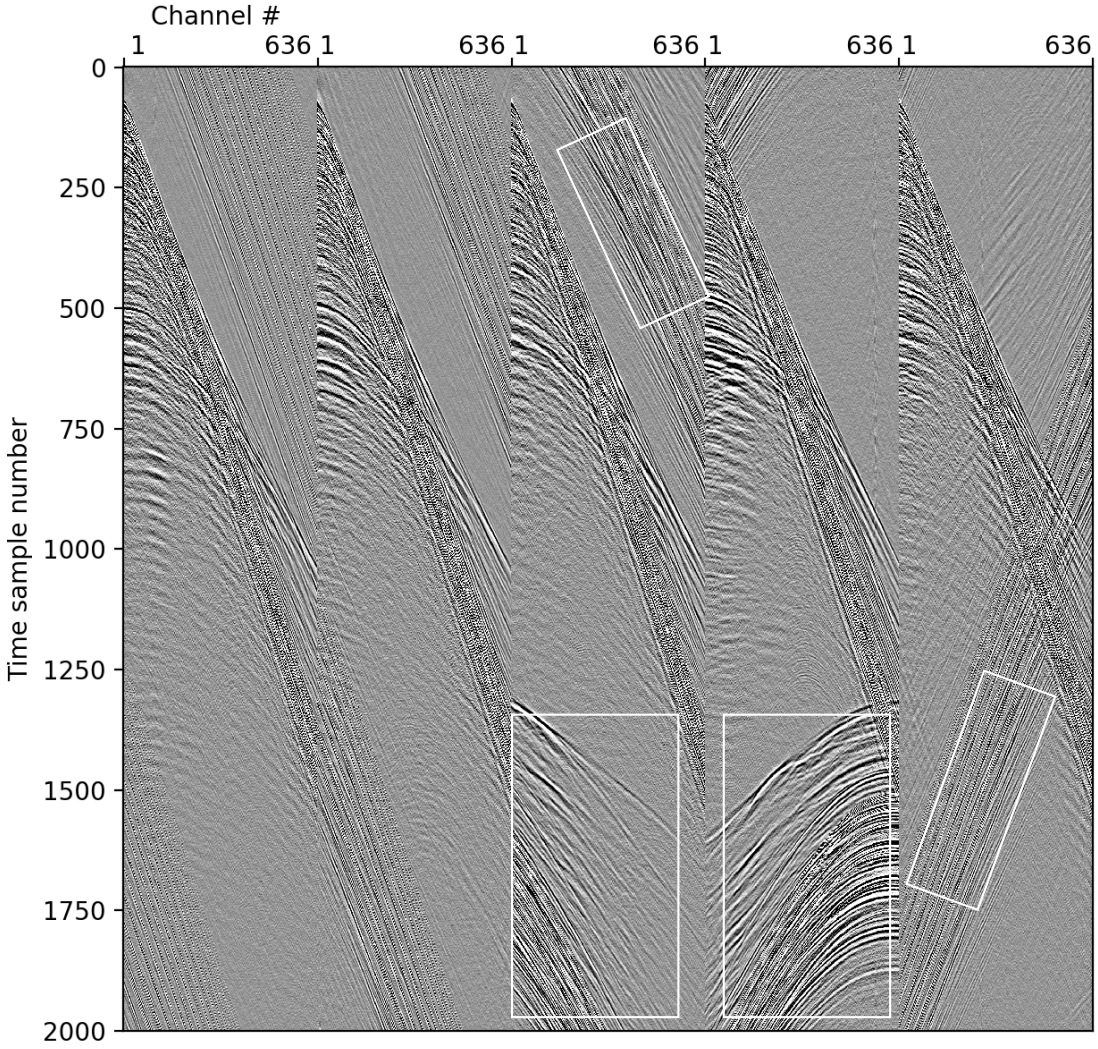


Figure 2.7: *Consecutive shotgathers displaying seismic interference. White boxes are marking different characteristics of SI in three of the shots, although SI is present in every shot.*

As mentioned eralier, the goal of this thesis is to investigate the feasibility of using machine learning -or more specifically convolutional neural networks for attenuating SI-noise.

3 | Neural Networks

Machine learning (ML) can be regarded as a trained approach to function estimation. Convolutional Neural Networks (CNNs) are no exception. When CNNs are applied to a problem, the network is trying to estimate a function $f(x) = \hat{y}$ such that $\hat{y} \approx y$ where x is the input, $\hat{y}$ is the output and y is the ground truth. In this thesis the input data is noise contaminated marine seismic data, and the ground truth is noise free data.

3.1 Classification vs. Regression

The type of network mentioned in 1.1.1 is a convolutional neural network designed for classification. The purpose of this thesis is to remove noise from seismic data, which can be classified as a type of regression problem. In ML there are generally two types of problems: Supervised and Unsupervised learning (Chapelle et al., 2006).

Unsupervised learning is when the user has a dataset and applies various machine learning method to structure the dataset, learn the structure of the dataset or other deeper features without a ground truth (Chapelle et al., 2006). It is therefore hard to estimate the performance of the model. This thesis will, however, only use supervised learning. In supervised learning, the ground truth is known. The type of ground truth defines whether the problem is a regression problem or a classification problem. Classification and regression are two major parts of ML. According to Mohri et al. (2012) classification is the concept of dividing input items into categories, while regression is the concept of predicting a value for each item.

Classification can be regarded as a sorting problem. The input data consists of a given number of classes where the network has to put correct item into correct class, thus making classification a discrete problem. The most basic type of classification is a True/False problem, but there are multiclass problems with a lot more complexity as well. A common example related to CNNs is classifying the public CIFAR-10 dataset, which is a dataset consisting of $6 \cdot 10^4$ small color images of size $32x32$ (Krizhevsky, 2009). The dataset has 10 different classes such as; horse, cat and airplane. When classifying such a dataset, the network has to make a prediction

on what the input image is and then put the corresponding label on the image. A loss function checks whether this labeling is correct or not and gives a measure on the performance of the network. If the network classifies an airplane as a cat, it will be caught by the loss function and the performance measure will be corrected accordingly.

Regression is used to predict continuous values, compared to classification which predicts discrete values. Regression is not an ML term exclusively. It comes from statistics where regression is often used to predict a relationship between a number of measurements, or the estimated point for a certain value. Linear regression is a well known example. If a dataset consists of multiple points spred over a grid, linear regression will estimate the best fitting line for all points. Regression is not limited to first order, as linear regression, and multivariate regression is widely used in both ML and by statisticians.

The problem presented in this thesis can be regarded as a regression problem. The network recieves noise contaminated seismic data as input and has to predict a noise free output. For every single data point in the input image, the network has to predict a noise free output value as close to the ground truth as possible. To be able to construct a network capable of such a task, the structure has to be understood.

3.2 Structure - Mathematical approach

The definition of a CNN is rather loose and according to Goodfellow et al. (2016) a CNN has to have at least one convolutional layer to be classified as a CNN. Each network has to be tailor made for each specific application. However, there are certain guidelines which can be followed. This thesis uses marine seismic data which can be regarded as black and white (BW) images. As mentioned in Chapter 1, an image in the eyes of a computer has one additional dimension, namely the color dimension. The color depth of a BW image is one, meaning marine seismic data can be regarded as a 2D matrix, not 3D as colored images. Before the different building blocks of a neural network can be explained in detail, certain important aspects have to be explained.

3.2.1 Forward Propagation

There are multiple key concepts in machine learning which might seem confusing at first. Terms like: weights, biases and nodes are in no way intuitive, but important to understand. Weights are essentially a decimal number which is used when calculating the output of a layer in a neural network. A bias is a value which is added to the output to reduce the chances of unwanted effects such as "dead neurons", which will be explained at a later point. Understanding the mathematics behind a standard neural network will ease the understanding of the specific building blocks. The first important concept in a neural network is the forward propagation, as explained by Ellacott (1997). A neural network, as mentioned in the introduction to this chapter, can be regarded as a representation of a function, f, mapping the input, $\mathbf{x}$, to the output, $\hat{\mathbf{y}}$. A neural network consists of multiple nodes arranged in a network structure. Figure 3.1 is a good way of visualizing how a node works and, in a larger scale, how the calculations of a neural network functions. In this case, the input, $\mathbf{x}$, consists of three values: $\mathbf{x} = [x_1, x_2, x_3]$. The input is fed to the node which has one weight, w_i, for each value in $\mathbf{x}$. The first intermediate step in the neuron is the linear combination of weights and input:

$$z = \sum_{i=1}^{m} w_i x_i + b \tag{3.1}$$

where m denotes the number of inputs (in this case 3) and the weights w are multiplied with each corresponding value x_i plus a bias addition. This can be vectorized, as shown in Figure 3.1, where

$$z = \mathbf{w}^T \mathbf{x} + b, \tag{3.2}$$

and T denotes taking the transpose of weight vector $\mathbf{w}$. A more generalized approach for multiple nodes can be written as

$$\mathbf{z} = \mathbf{W}^T \mathbf{x} + \mathbf{b} \tag{3.3}$$

Where $\mathbf{W}^T$ is a matrix with weights. The last intermediate step is passing $\mathbf{z}$ through a non-linear activation function. In this example, we simply apply one of the most common activation functions used: ReLU (Rectified Linear Unit). The output is then calculated as follows:

$$\sigma(\mathbf{z}) = \hat{\mathbf{y}} \tag{3.4}$$

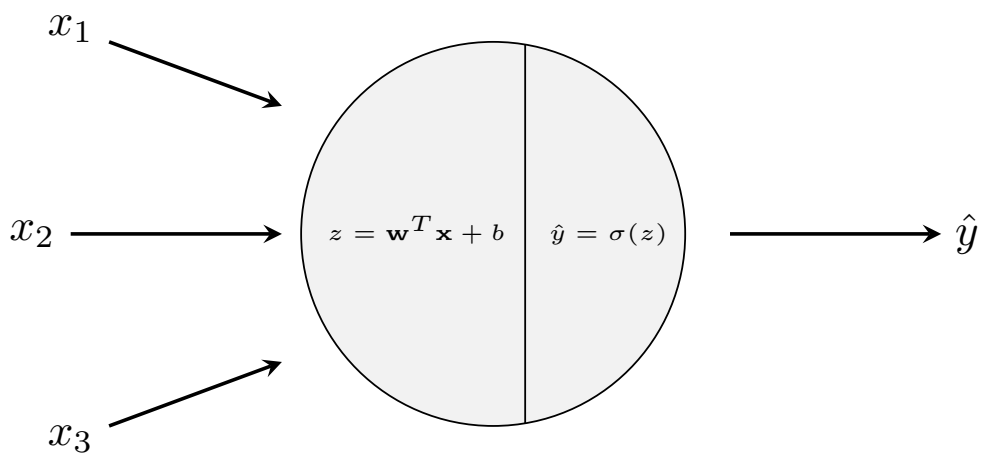


Figure 3.1: *Illustration of a node with three inputs, x_1, x_2, x_3, and one output, $\hat{y}$. The calculations computed in the node are visualized in a vectorized manner.*

where

$$\sigma(\mathbf{z}) = max(\mathbf{z}, 0) \tag{3.5}$$

This is the main process of a neural network and is referred to as the forward propagation or the forward pass. The input is multiplied with weights and then passed through a non-linear activation function. The next step in understanding neural networks is to realize the importance of activation functions.

3.2.2 Activation functions

Activation functions represent a key feature in neural networks and introduce non-linearity (Agostinelli et al., 2014). A linear function is, as the name suggest, a straight line when plotted. Such functions are easy to solve, but they are limited in their complexity. If all activation functions in a neural network were linear, the entire network could be represented as a collection of linear operations, i.e. a more conventional regression problem trying to fit the best line through a number of samples. Most types of modern problems solved with neural networks have no linear approximations or solutions. It is therefore benefitial to introduce non-linearity, thus activation functions.

Another required characteristic of an activation function is that it needs to be

differentiable. This is because the networks training process and a specific step called backwards propagation. During backwards propagation, the network adjusts its weights and biases based on the error between output and ground truth and all the neurons in the network. A gradient of the non-linear function is calculated to optimize the network performance, which is not possible if the activation function is non-differentiable. The concept of back propagation will be discussed in more detail at a later point.

To summarize, an activation has to be non-linear and differentiable. The most common used activation functions are: Sigmoid, TanH, ReLU and Leaky ReLU as seen in Figure 3.2.

Sigmoid

Sigmoid is a differentiable function bounded within the range $[0, 1]$ (Han and Moraga, 1995). It is defined as

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{3.6}$$

making the function centered around 0.5 and giving it a characteristic "S-shaped" curve. Sigmoid fits all necessary requirements, but has been proven non-optimal for certain machine learning problems.

One of the drawbacks is something called vanishing gradients (Nwankpa et al., 2018). When the output from a Sigmoid approaches 0 or 1, the gradient of the function is close to zero. During backwards propagation, these gradients are multiplied to the neurons, causing them to send little to zero signals. This is a phenomenon often called "Dead Neurons". Once a neuron is stuck at close to no output, it will not recover and thus will not contribute to the network.

Another problem of a sigmoid is that it is not zero centered (Nwankpa et al., 2018). The way Sigmoid is defined, all output will be positive. When multiplying with the gradient during back propagation, the weights tends to be moved too far in either direction making Sigmoid an inefficient function which is slow to converge. Although there are a lot of downsides to Sigmoid, it has proven useful when employed at the output layer for certain network architectures.

TanH

TanH (Hyperbolic tangent) is a 'scaled' version of Sigmoid and defined as

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{3.7}$$

TanH is defined in the range $[-1, 1]$ and is therefore zero centered. Because of this characteristic, TanH often leads to a better optimization than Sigmoid and tends to be preferred (Karlik and Olgac, 2011). There are still downsides to TanH and as the Sigmoid, it also suffers from vanishing and exploding gradients. Since it may cause "Dead Neurons" it is less and less common in use. The reason it is still employed is due to its boundary range. Since this is defined between $[-1, 1]$ it is possible to normalize the input values to this range, thus easing the data handling needed before feeding data to the network.

Rectified Linear Unit

Rectified Linear Unit (ReLU) is an activation function which has become more popular during recent years. It is known for its simplicity defined as

$$a(x) = \max(x, 0). \tag{3.8}$$

This means that ReLU is equal to x if $x \geq 0$ and equal to 0 otherwise. The simplicity of ReLU means that it is faster. It requires less computational power to run and has even proved to improve results for certain types of deep neural networks (Glorot et al., 2011). ReLU is non-differentiable at zero which, in theory, makes it a bad choice for an activation function. It is, however, differentiable at all other places and can be filled in as a 1 or 0 where the input is 0. A downside to ReLU is that it, as TanH and Sigmoid, can cause "Dead Neurons" (Maas et al., 2013). A way of reducing the chance of dead neurons is implementing a small slope for negative arguments in the ReLU.

Leaky ReLU

Leaky ReLU is a commonly used version of ReLU which adjusts for the vanishing gradient problem. It introduces a small slope instead of forcing $a(x) = 0$ when $x < 0$ and is defined as

$$a(x) = \max(x, \alpha x) \tag{3.9}$$

where α is often set to 0.01. It has the same characteristics as ReLU, but a reduced chance of causing "Dead Neurons" as the gradient is non-zero for $x < 0$ (Maas et al., 2013).

3.2.3 Loss

The loss function in a neural network is a way of estimating how well the model is preforming. It is not arbitrary which function to use. If the wrong loss function is implemented, the model might preform badly and choosing the best fit is therefore important. The most common functions used in denoising are Mean Squared Error (MSE) and Mean Absolute Error (MAE). Huber loss and Binary cross-entropy (BCE) are also included in this thesis. All loss functions can be viewed in figure 3.3.

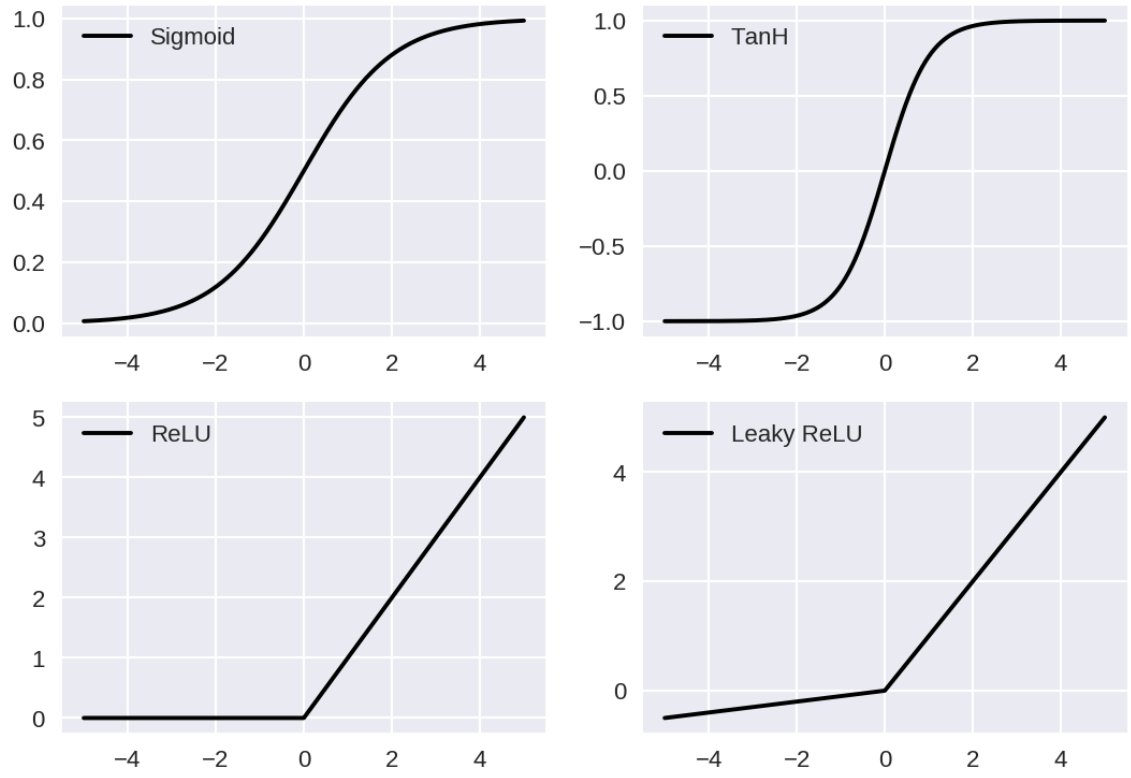


Figure 3.2: *Illustration of the functional curves of the four most common activation functions used in neural networks: Sigmoid, TanH, ReLU and Leaky ReLU*

Mean Square Error - L^2

The mean square error (MSE) is a common way of measuring the loss in a neural network (Zhao et al., 2017). It is defined by the equation

$$h(x) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.10}$$

where $\hat{y}$ is the predicted value and y is the true value. MSE is a loss function which penalizes high errors since it is based on a squared error. This means that MSE is sensitive to outliers which can be a problem in certain cases.

Mean Absolute Error - L^1

The mean absolute error (MAE) is another common loss function used in CNNs. It is defined by equation

$$h(x) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{3.11}$$

where $\hat{y}$ is the predicted value and y is the true value. In contradiction to MSE, MAE is not sensitive to outliers and will not penalize high errors. This means that MAE is more robust than MSE (Willmott and Matsuura, 2005). There is, however, a downside to using MAE. The gradient of MAE has the exact same slope as long as the difference $\neq 0$. This might cause issues during training where the network might struggle with converging to a minima due to high gradients.

Huber Loss

The Huber loss is a combination of MSE and MAE which utilizes the best of both functions. It is defined by the following case wise function (Huber, 1964)

$$h(x) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| < \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta & \text{otherwise} \end{cases} \tag{3.12}$$

If the absolute difference between $\hat{y}$ and y is less than a given value, δ, the function calculates the MSE loss. Otherwise, a modified MAE is calculated as can be seen from the equation 3.12.

Binary Cross-Entropy

Binary cross-entropy (BCE) is, as the name suggests, a loss function restricted to binary problems. It is also referred to as the logistic loss or log-likelihood (Murphy, 2012). For binary problems, the output is either 0 or 1. Assume a case where the network is to label dog and cat: the label for dog is 1 and cat is 0. Using the equation for BCE gives:

$$h(x) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot (1 - p(y_i)) \tag{3.13}$$

where $p(y_i)$ is the probability for the input being a dog. If this equals 0, the network is certain the input is not a dog, and therefore a cat. If more than two labels are introduced to this loss function it will not work given the binary restriction.

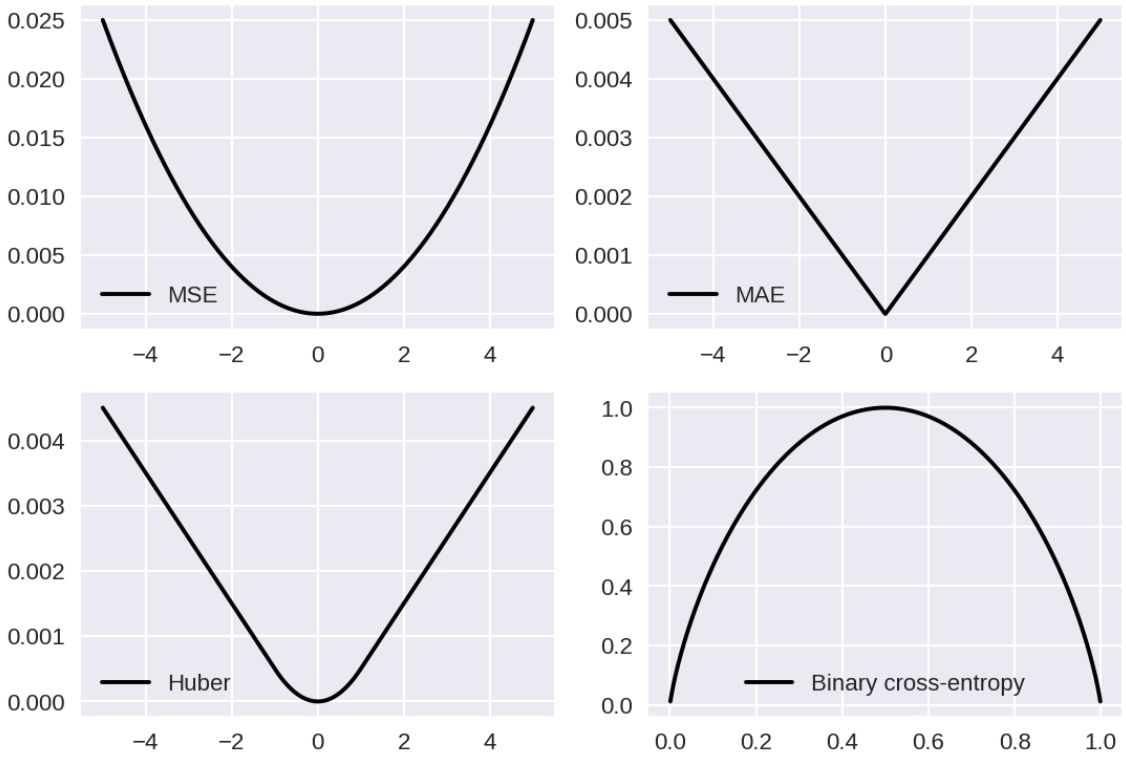


Figure 3.3: *Illustration of the functional curves of the loss functions presented in this thesis: MSE, MAE, Huber and BCE*

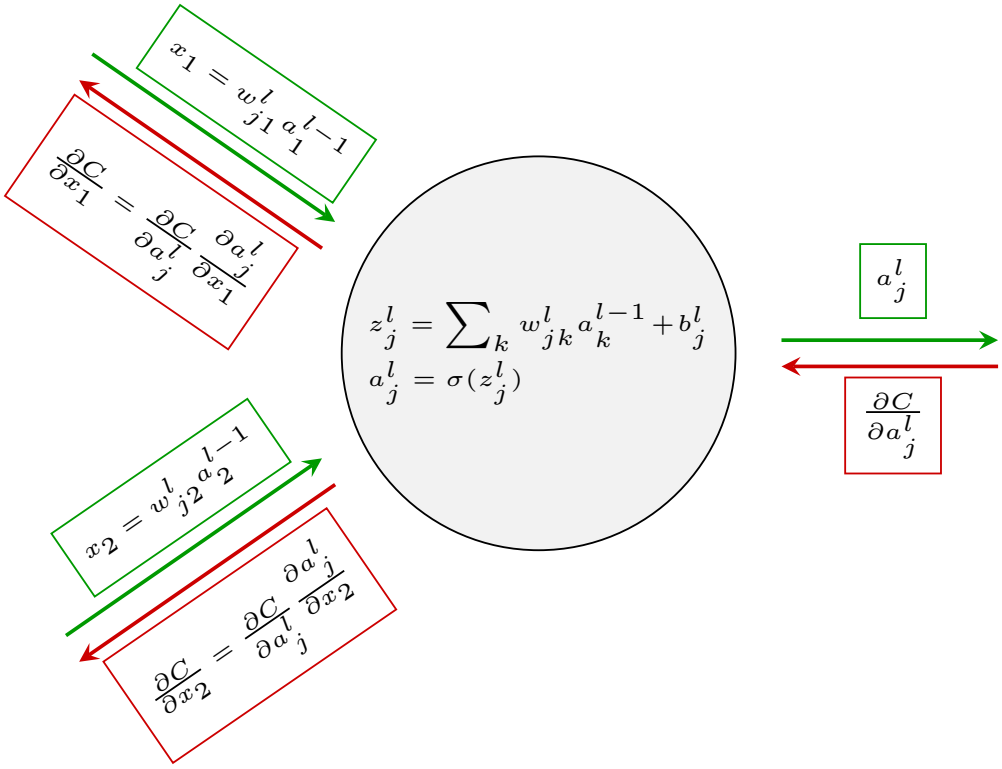


Figure 3.4: *Figure of a node illustrating the relationship between Forward and Backward propagation. Forward propagation is marked in green and Backward in red. Adapted from (Sood, 2018).*

3.2.4 Backward propagation

Backward propagation (backprop), as mentioned above, is a key feature of every neural network. It is the process where the network updates its weights based on the loss between the ground truth and calculated output. To understand how backprop works, the forward propagation has to be well understood. The theoretical framework used in this section is based on Nielsen (2015). The activation output from one neuron is given as

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) = \sigma\left(z^l_j\right) \tag{3.14}$$

where a_j^l is the activation output, w_{jk}^l are the weights, b_j^l denotes bias and σ is the activation function. The indices j, k and l denotes respectively node number in layer, node number in previous layer, and layer number. Equation 3.14 is the same equation as 3.1 and 3.4 combined, but generalized to a larger model. Corresponding vectorized form of equation 3.14 is

$$\mathbf{a}^l = \sigma\left(\mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l\right) \tag{3.15}$$

with same annotations as in equation 3.14. To update the weights, we need to calculate the gradients of the loss function with respect to weights and biases, $\frac{\partial C}{\partial w_{jk}}$ and $\frac{\partial C}{\partial b_j}$. Let y be the ground truth, and $\hat{y}$ be the output of the network which can be rewritten as $a^L(x)$ being the output from the last activation function (L denotes the number of layers in the network). Define now the quantity

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.16}$$

δ_j^l which denotes the error in the j^{th} neuron in the l^{th} layer. z_j^l is the weighted input to a node, as in equation 3.2. ∂C denotes the gradient of the loss function which will be utilized in the optimizer (section 3.2.5). Back propagation starts at the last layer of the network (denoted L) and, as the name suggest, propagates backwards through the network, Figure 3.4. The Figure illustrates an arbitrary node in the network where green arrows symbolizes forward pass and red arrows symbolizes backward propagation. The first step of backprop is therefore calculating the error in the output layer

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \tag{3.17}$$

where $\partial C/\partial a_j^L$ measures how fast the cost is changing as a function of the j^{th} activation output. $\sigma'(z_j^L)$ measures how fast the activation function, σ, is changing at z_j^L. Corresponding vectorized form of equation 3.17 is

$$\boldsymbol{\delta}^L = \nabla_a C \odot \sigma'\left(\mathbf{z}^L\right) \tag{3.18}$$

where $\odot$ denotes the Hadamard product (element-wise multiplication). The next step is creating a generalized equation for the remainder of the network

$$\boldsymbol{\delta}^l = \left(\left(\mathbf{W}^{l+1}\right)^T \boldsymbol{\delta}^{l+1}\right) \odot \sigma'\left(\mathbf{z}^l\right) \tag{3.19}$$

which propagates through the network from the last layer to the first layer. The next step of in the training regime is to update the weights and biases. The update process depends on which optimizer is used.

3.2.5 Optimizer

The optimizer is an important building stone in a neural network. It is used alongside backward propagation and changes the weights and biases based on the gradient of the loss function. In essence, an optimizer has two tasks: finding the direction in which to move the weights and biases and the distance in which to move them (Mitchell, 1997). There are multiple different optimizers, but most have one thing in common; they are based on gradient descent.

Gradient Descent

Gradient descent is the basis of most optimizers and the most popular optimizer used for neural networks (Ruder, 2016). Gradient Descent is defined by

$$\theta_t = \theta_{t-1} - \frac{\eta}{m}\left(\frac{\partial C}{\partial \theta_t}\right) \tag{3.20}$$

where η is the learning rate, m denotes the batch size, θ denotes either weights or biases at time step t and C is the loss as explained in section 3.2.4. It minimizes the loss function by use of its gradient. It checks whether the gradient is positive or negative - whether the loss is increasing or decreasing and updates θ_t correspondingly.

RMSprop

RMSprop (Root Mean Square propagation) is an attempt to optimize gradient descent proposed by Hinton et al. (2012). It is defined by

$$g_{\theta_t} = \beta \cdot g_{\theta_{t-1}} + (1-\beta)\left(\frac{\partial C}{\partial \theta_t}\right)^2 \tag{3.21}$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{m\sqrt{g_{\theta_t}}}\left(\frac{\partial C}{\partial \theta_t}\right) \tag{3.22}$$

where η is the learning rate and g_{θ_t} is the moving average of the squared gradients at time step t. More over, m denotes the batch size, θ denotes either weights or biases, C denotes the loss as explained in section 3.2.4 and β is a user defined scalar.

Comparison

The main difference between RMSprop and Gradient Descent is apparent in the equations listed. The consequence is that Gradient Descent is more efficient to calculate and will therefore have a lower run time compared to RMSprop. Gradient descent converges relatively fast for large datasets, but might get stuck at saddlepoints. Gradient squared optimizers, such as RMSprop, converges faster and tend to find better results as they seem less likely to get stuck at saddlepoints (Ruder, 2016).

3.3 Layers

A CNN is built up by layers and understanding the function of each layer is a key concept when building a network. The different layers have different properties and functions. If each layer is generalized, the function of a layer in a CNN is transforming an input volume to an output volume with some differentiable function (Karpathy, ND). This differentiable function is the activation function, as mentioned in the previous section. The most common layer used in all neural networks are also common in classification CNNs, namely the fully connected layer.

3.3.1 Fully connected layer

The fully connected (FC) layer (often called dense layer) is a key building block in classification networks. They are the main building blocks in the Multilayer Perceptron (MLP). MLP is, as the name suggests, multiple perceptrons set into a specific system. The perceptron is one of the first neural networks and was created by Rosenblatt (1958). He first published the perceptron as a possible model for how the brain stores and organizes the information. This model, as can be viewed in Widrow and Lehr (1990), consists of analogue valued input connected to six threshold elements and passed through a desired response node. A way of describing this small scale network in modern ways is with one node as in equation 3.1, followed by a threshold. Minsky and Papert (1969) mentioned in their book how the perceptron is limited to linear problems, highlighting the constraints of the model. Prior to this publication, Rosenblatt (1961) had already created connected perceptrons known as MLP. It is at least three layers deep where every node, except for the input layer, uses non-linear

activation functions. An MLP is therefore not restricted to linear problems. An MLP is a feed-forward type of network often used for classification. Feed-forward means that the data-flow in the network travels from A to B directly with no loop or feedback. A feed-forward network approximates a function $y = f(x; \theta)$ and learns the parameters θ for the best approximation of f (Goodfellow et al., 2016).

A fully connected layer is a layer where every node in the current layer is connected to every node in the following layer. FC layers are well suited for classification. Assume a network structure designed for image classification. The input, x, is an image of size $[32 \times 32]$ where the input data consists of 10 different classes, e.g. the CIFAR dataset (Krizhevsky, 2009). The network takes these images as input and maps them to a specific category. For an FC layer to be able to take input, the input has first to be represented as a one-dimensional array. The input image therefore has to be concatenated into an array of size $32 \cdot 32 = 1024$. The FC layer will then have 1024 weights for each node in the layer, one for each entry in the input. Figure 1.1 is a small scale network. If all nodes between layer one and two are interconnected, this could illustrate an FC network. By using the same example as above, for each hidden layer, the network has 4096 weights per layer. This shows why FC layers scales badly with images and is mostly used in latent layers during classification. The more common way of handling full size images with neural networks is using convolutional (CV) layers. This is because FC layers work globally, while CV layers work locally.

3.3.2 Convolutional layer

A convolutional (CV) layer is a layer which, suggested by its name, convolves the input. The operation in a CV layer in Tensorflow is, however, not a convolution but a cross-correlation and calling it a convolution is factually wrong (Tensorflow, 2019). The literature and entire field of ML using Tensorflow refers to this operation as a convolution, therefore once the term convolution is used in this thesis, it is actually referring to a cross-correlation. The math behind both a convolution and a cross-correlation will be explained in detail below, but to ease the transition, the general movement and behaviour of a CV layer will be explained first.

The behaviour of a Convolutional layer

Each convolutional layer has assigned a set of filters (kernels) where all have the same size. These filters are normally square matrices consisting of decimal values with a chosen size. For educational purposes, assume the filters are 3×3 filters with values $x \in [0, 1]$. The CV layer slides the filters over the input data step wise, typically referred to as convolving the data. The step length is referred to as stride and is a user defined parameter. If stride equals one, the filters moves one pixel between each computation. For each step, the filters compute the elementwise multiplication between the filter values and the image values, which are then summed and assigned to the corresponding pixel in the output image. This process can be viewed in Figure 3.5. Although the filters operate in 3D, they create a 2D activation map for each filter. This means that the depth of the output volume equals number of filters in the layer. This process is the convolution of the data, and the basis of this specific layer type.

As can be seen in Figure 3.5, if no padding is applied, a CV layer will reduce the image size. The size reduction is dependant on the filter size, the stride and the number of filters in the layer. By using Figure 3.5 as a template, we have an input volume with size $[5 \times 5 \times 3]$. We set padding to 0, stride to 1 and use 3 filters. The width of the output volume then becomes:

$$\frac{W - F + 2P}{S} + 1 = \frac{5 - 3 + 2 \cdot 0}{1} + 1 = 3 \tag{3.23}$$

where W is the width of the input volume, F is the filter size, P is the padding and S is the stride. The same procedure regarding the height of the output volume can be repeated with equation 3.23, but since the input volume is squared, the width and height is equal. The total volume of the output then becomes: $[3 \times 3 \times 3]$ where the depth is equal to the number of filters in the CV layer.

There are two main reason for why CV layers are favourable in image processing. The first and foremost reason is that CV layers scale well with increasing image size (Goodfellow et al., 2016). The weights in a CV layer is not dependant on the input volume size. This means that with static parameters in the CV layer and with increasing input volume size, the number of weights will remain the same. The number of weights in a CV layer is given by $F_{\text{height}} \times F_{\text{width}} \times N_{\text{channels}}$ where channels are the depth of the input volume. This means that even for large input volumes, CV layers are computationally efficient. The second reason for why CV layers are

favourable to image processing compared to FC layers is due to the spatial view. Since CV layers consists of filters sliding across the input data, they can detect edges, shapes or patterns which are connected over multiple pixels. According to Goodfellow et al. (2016), CNNs are well suited for 2D images where neighboring pixels are connected in a larger pattern. It should therefore be possible to denoise seismic data with localized and 'random' noise and we assume that CNNs will be able to handle seismic data, given its continuous nature and 2D structure.

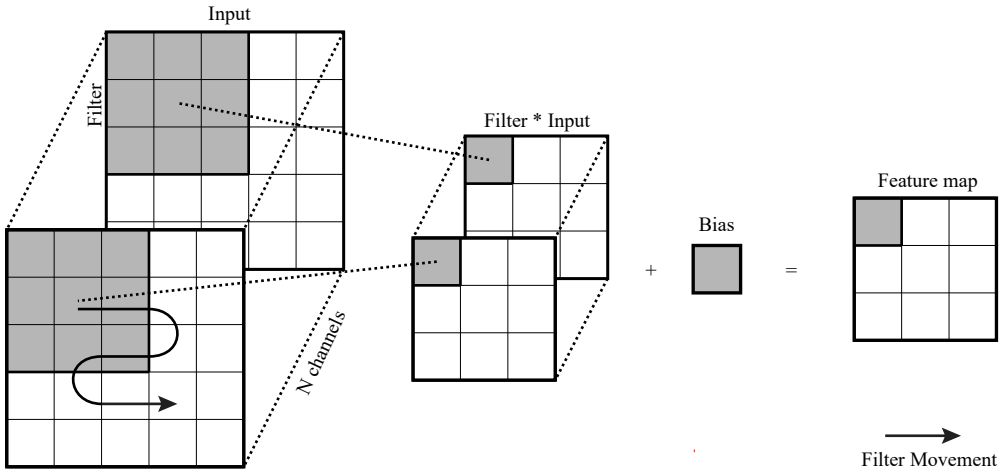


Figure 3.5: *Figure illustrating how a convolutional layer operates on data. In this case, no padding is applied which results in a size reduction of the output data. The input data has multiple channels, which also yields for the filter sets. In this example, the layers has a single filter set. The output therefore becomes a 2-dimensional feature map which is the summation of the elementwise product of filter and input channel. The black arrow on input illustrates how the filter moves across the input volume.*

The "convolution" operation

Convolution (CV) (or convolving) is a mathematical operation which describe the relationship between three functions (Smith, 1997). It is a measure of how two functions effect each other. Convolution can be expressed in multiple dimension, and the 2D operation can be expressed grid wise as follows:

$$y = (g * F) = \sum_{u=1}^{W} \sum_{v=1}^{H} g[u, v] F[i - u, j - v] \tag{3.24}$$

where y is a measure of how much f and g overlap in the given domain. W and H denote the width and height of the filter. Although it is called a convolutional layer, the actual mathematical operation carried out (in Tensorflow) is a cross-correlation (CC). In contrary to convolution, CC is a measurement of the similarity or relation between two functions (Lahiri, 2016). Although there is a difference in the purpose of CC and CV, the definition of cross-correlation has a lot of similarities to convolution and is given as follow:

$$y = (g \star F) = \sum_{u=1}^{W} \sum_{v=1}^{H} g[u, v] F[i + u, j + v] \tag{3.25}$$

Figure 3.6 is an illustration of the difference between the two operations. The filter used in CC is the original filter. The convolution rotates the filter in both x and y direction which therefore results in different outputs for the two operations.

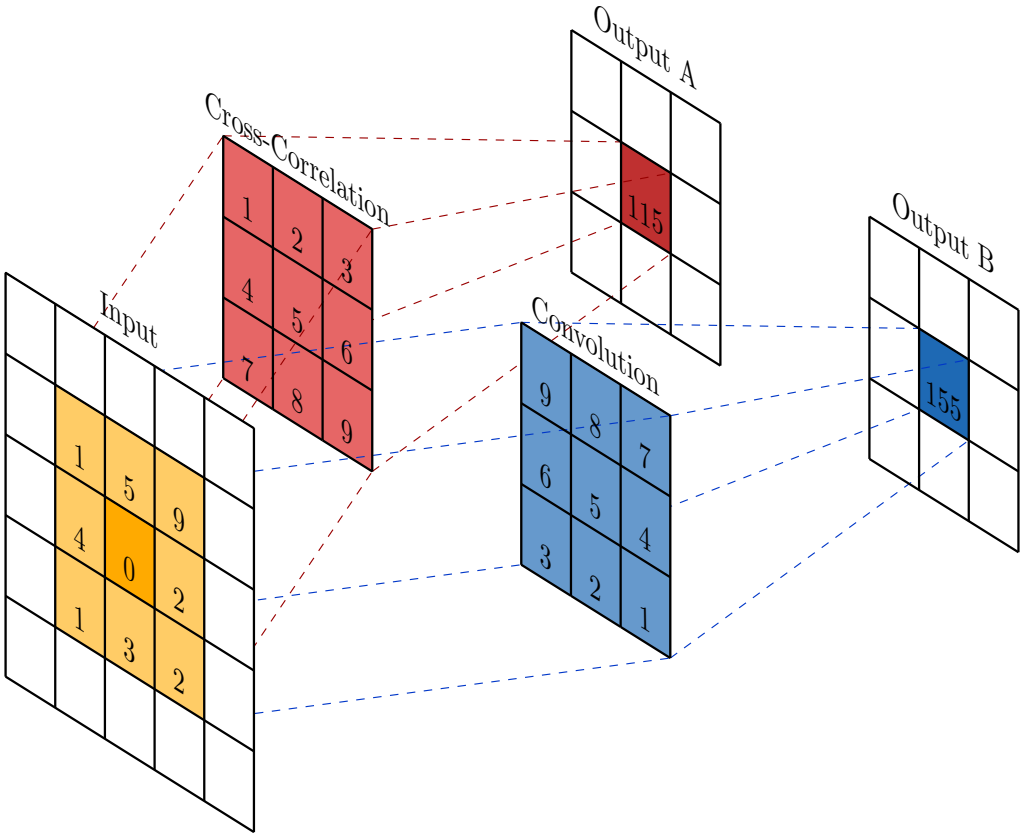


Figure 3.6: *Figure illustrating the difference between correlation and convolution with outputs respectively Output A and Output B. The initial filter has the same alignment as used in Cross-Correlation and is rotated in the Convolution operation.*

It is important to notice the difference and to know what is actually happening in the network, but it is also important to know that both operations converge towards the same output value during training in most cases. If the kernel is initialized with random values, which is common, it makes no difference whether the filter is rotated or not. The only cases where the difference might make an impact is with a network with preset filters. If the user assumes a convolution, a cross-correlation can give wrong results.

3.3.3 Pooling layer

A pooling layer is a layer constructed from local information with similarity to convolutional layers. The difference is that pooling layers down-sample the volume, while a convolutional does operations filter-wise. Pooling layers can be split into three different categories: max, average and min pooling, see Figure 3.7. Let us assume the pooling has a filter size of 2×2. For each region, the layer does a specific operation based on the type of pooling. The pooling types will map 4 values to one output value where: max pooling sets the output value to be the largest of the 4 values, min pooling sets the output value to be the smallest of the 4 values and average pooling computes the average of the 4 values. This process can be viewed in Figure 3.7 where different regions are color coded to ease the understanding. The output of a 2×2 pooling with stride 2 is a down-sampling by a factor of 2 in each direction.

Pooling is much used in machine learning, and the most common type is max pooling because large values tend to be most important when using images. Max pooling is a way for the network to extract the sharpest features of an image, thus the subsequent layers after a max pooling layer will have an image consisting mainly of important features and less abundant information. Reducing the size of the input volume is also a way to increase computational speed. The downside to using pooling is that whenever down-sampling is implemented, information is lost. For networks where images are down-sampled and up-sampled before output, the end results tend to be blurry. Max pooling is therefore a good way for the network to learn sharp features, but might cause lacking output due to information loss.

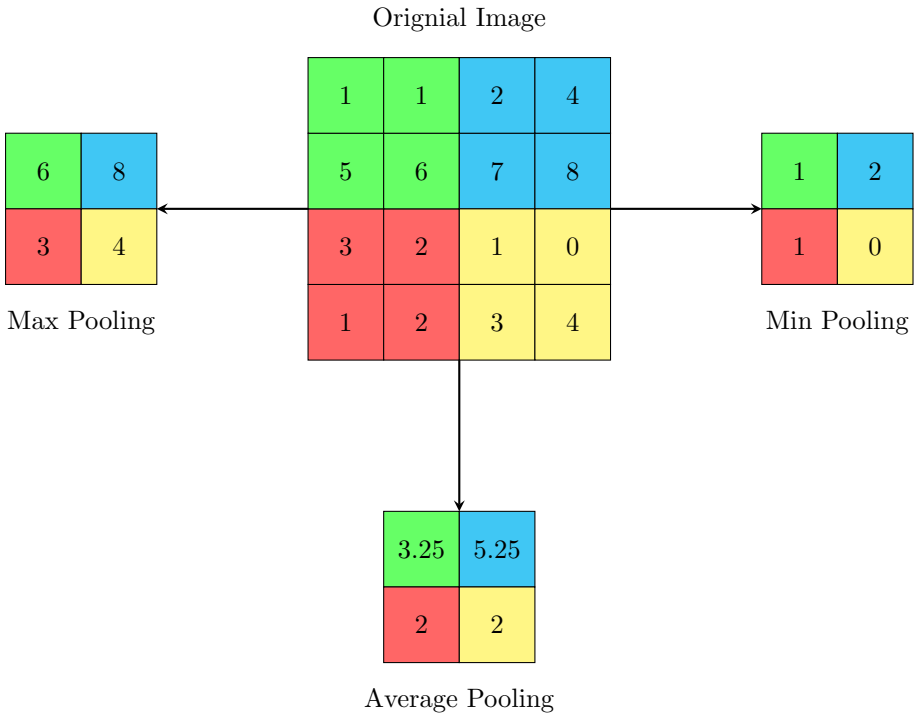


Figure 3.7: *Visual examples of* 2×2 *pooling of different kinds*

3.3.4 Batch Normalization

When using machine learning, it is common to normalize the data before it is passed to the network. The reason for this is because the network operates better on numbers in a narrow range. Assume input data with value range of $[1, 1000]$ compared to $[0.001, 1]$. The range is the same with regards to power, but the network works better with data in the latter range. If values become too large, the chance of exploding gradients increase, thus increasing the artifacts in the data. Normalized input is not normalized after passing through the first layer of the network. A way of counteracting this is introducing batch normalization (BN) as defined by Ioffe and Szegedy (2015) as

$$BN(x_i) = \gamma \left(\frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \beta \tag{3.26}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ denotes the mean and the variance of the input batch x_i. γ and β are two trainable parameters enabling the optimization of these parameters during optimization. BN introduces robustness to the model and might enable the use of higher learning rates, thus increasing the efficiency of the model.

3.3.5 Upsampling layer

The Upsampling layer, often called deconvolution, is essentially a transposed/inverse convolution. The layer operate in a similar manner to convolutional layer, but instead of scaling an image down, the image is scaled up. Deconvolution is a bad name, as it does not undo a convolution, but performs a convolution the opposite way. The output pixel in a normal convolution is the summation of the elementwise multiplication between input region and filter values. The transposed convolution works differently. It multiplies a value in the input with the filter values and copies it to multiple pixels, Figure 3.8. The output from an upsampling layer will be weighted copies of the filter, weighted by the input. If upsampling by a factor of 2, the filter will move 2 pixels in the output for every corresponding pixel in the input. Overlapping regions in the output will be summed (Li et al., 2017).

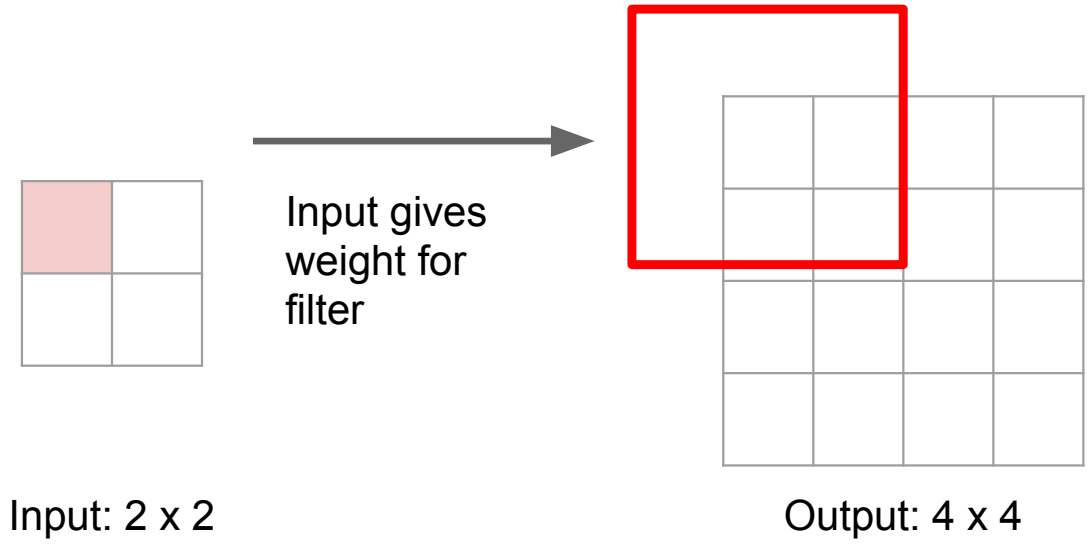


Figure 3.8: *Figure illustrating the process in an upsampling layer for a case with 2 times upsampling with a 3x3 layer (Li et al., 2017).*

4 | Framework

Machine learning and neural networks is a broad field of work. There is a great variance in methods and models based on the problem at hand. However, there is one thing which tends to yield for machine learning as a whole; it is computationally demanding. Thus, machine learning requires a lot of processing power.

4.1 Hardware

The hardware used to run the neural network models in this thesis was provided by CGG. The processing GPU cluster consisted of eight nodes where 4 of the nodes were assigned to this work. Each node has the following specifications:

- CPU: Intel Xeon CPU E5-160 v3, 3.50 GHz, 10Mb cache, (8x)
- GPU: Nvidia GeForce GTX 1080, 8GB, 1.73 GHz, (2x)
- HDD: 62 GB storage (per user)
- OS: Debian 7, Wheezy

Listing the specifications of a computer might be confusing to many, so to put everything into perspective, the server will be compared to retail computers. Macbook pro from Apple is a widespread computer and with the best specifications has a GPU providing approximately 4GB of GPU memory and retails for 26 000 nok (Apple, 2019). Comparing these specifications to the server equals 4 Macbook pros only counting for the GPU memory for a single node. The average retail price for a GeForce GTX 1080 GPU is approximately 8000 nok, proving how much computing power these nodes contain compared to average computers.

It might seem like these specifications are unnecessarily high, but machine learning is a computationally demanding task, especially when used with marine seismic data. An average network used in this thesis used approximately 14.5 GB of the 16 GB GPU memory. If the number of images per iteration was increased, the GPU memory was exhausted.

4.1.1 GPU vs CPU

It might seem strange to some that the computation is done on GPUs rather than CPUs. CPUs have always been the processing part of a computer, while the GPU has been used for picture rendering in terms of movies, games and images. Although this is still the main trend, it is slowly turning. A GPU can be much more efficient than a CPU for certain computational processes. The way this can be explained is to analyze the computational speed versus the memory bandwidth. The computational speed of a CPU is superior to the computational speed of a GPU. If a CPU and a GPU were both introduced to a single operation problem, the CPU would complete the task at fractions of the time compared to a GPU. The reason a GPU can compete with a CPU in regards of machine learning is the memory bandwidth. A high end CPU may have a memory bandwidth of 40 GB/s while a high end GPU might have a memory bandwidth of 600 GB/s.

It is important to keep in mind that since the CPU is much faster for single operations, the task has to be large in number, but low in operation calculation. A high end CPU has 8 cores while a high end GPU has 4096 cores. Each of the cores in a GPU can be regarded as simple while each core in a CPU is sophisticated. As long as the task is simple enough, the 4096 GPU cores will complete it conciderably faster faster than the 8 CPU cores, no matter how sophisticated they are. This is the reason why GPUs are a good choice for machine learning. The calculations done in each node tend to be basic operations such as addition and multiplication, but there are a lot of nodes and a lot of tasks to be fulfilled. The shear memory bandwidth of a GPU therefore fits nicely with deep learning.

4.2 Software

During the last $\sim$5 years, many companies have created open source software tailor made for machine learning. The programming languages these are compatible with tend to narrow the range of choices. It is common now to use either Python or C++ for machine learning as these are the most supported programming languages for such packages. All work in this thesis is programmed in Python 3. Python is a high-level programming language which focuses on being general purpose and an easy code to read (van Rossum, 1995). It has gained a lot of grounds the last 10 years and is slowly taking the position of Matlab since Python is free to use.

Along with Python, the following packages are installed to make Python run the neural networks on GPU:

- Anaconda: Distribution package for Python/R
- CUDA toolkit: Development environment for GPU applications
- cuDNN: Deep Neural Network library for GPU
- Keras: Wrapper for TensorFlow
- TensorFlow: Open source machine learning framework

4.2.1 Nvidia drivers

cuDNN and CUDA Toolkit are libraries made specifically for running Deep Neural Networks on Nvidia based GPUs. cuDNN consists of tuned implementations for common routines used in neural networks (Nvidia, 2019b). It is free for non-comercial use making it a good option for students or personal use. CUDA toolkit is a development environment tailor made for creating applications running on GPUs (Nvidia, 2019a).

To be able to run TensorFlow on an Nvidia GPU, both cuDNN and CUDA Toolkit are required. cuDNN and CUDA also require a specific Nvidia driver to be installed making the installation process tedious. The cuDNN version has to match the CUDA version and the Nvidia driver version. The correct version of TensorFlow then has to be installed on top matching the correct version of Python.

4.2.2 TensorFlow

TensorFlow is an open source software machine learning framework originally developed by Google (Abadi et al., 2015). Tensorflow has implemented almost everything that is used in machine learning to this date. If certain functions or layer types are unavailable, these can be created and relatively easily be included with predefined TensorFlow functions.

4.2.3 Keras

Keras is an overlay for TensorFlow and a couple of other well known machine learning libraries. Keras focuses on being an easy to use high-level machine learning library that lowers the threshold of creating machine learning scripts (Chollet et al., 2015).

4.2.4 Anaconda

Anaconda is a Python (or R) distribution making it easier to install and manage all Python packages (Anaconda, 2019). Instead of downloading every single library needed, Anaconda comes predefined with a large number of packages. If the wanted packages are not installed, simple command line arguments will install the package at the correct location. Anaconda also enables the use of environments, meaning that multiple Python versions can be run on the same computer with different packages installed. Multiple environments mean that different version of TensorFlow and Keras could be installed on the same computer at the same time flawlessly. This was utilized much in the first months of the thesis to understand which versions were most desirable to use.

5 | Method

When designing a model, there are multiple factors to take into account. If certain prerequisites are omitted, the model might under-perform, overfit or behave in other negative ways. One of the most important factors when designing a model is to fit the model to the desired task and dataset.

5.1 Dataset

There are as many models as there are problems to solve. The different datasets existing used for machine learning range over multiple different file types such as text, video, matrices, images and so on. For this thesis a specific format is used; namely marine seismic data. This kind of data is represented as matrices, but can also be regarded as images. It is therefore likely to adapt a model in a similar way as one might when doing image processing with machine learning. There are, however significant differences between conventional images and seismic data.

5.1.1 Seismic data vs. conventional images

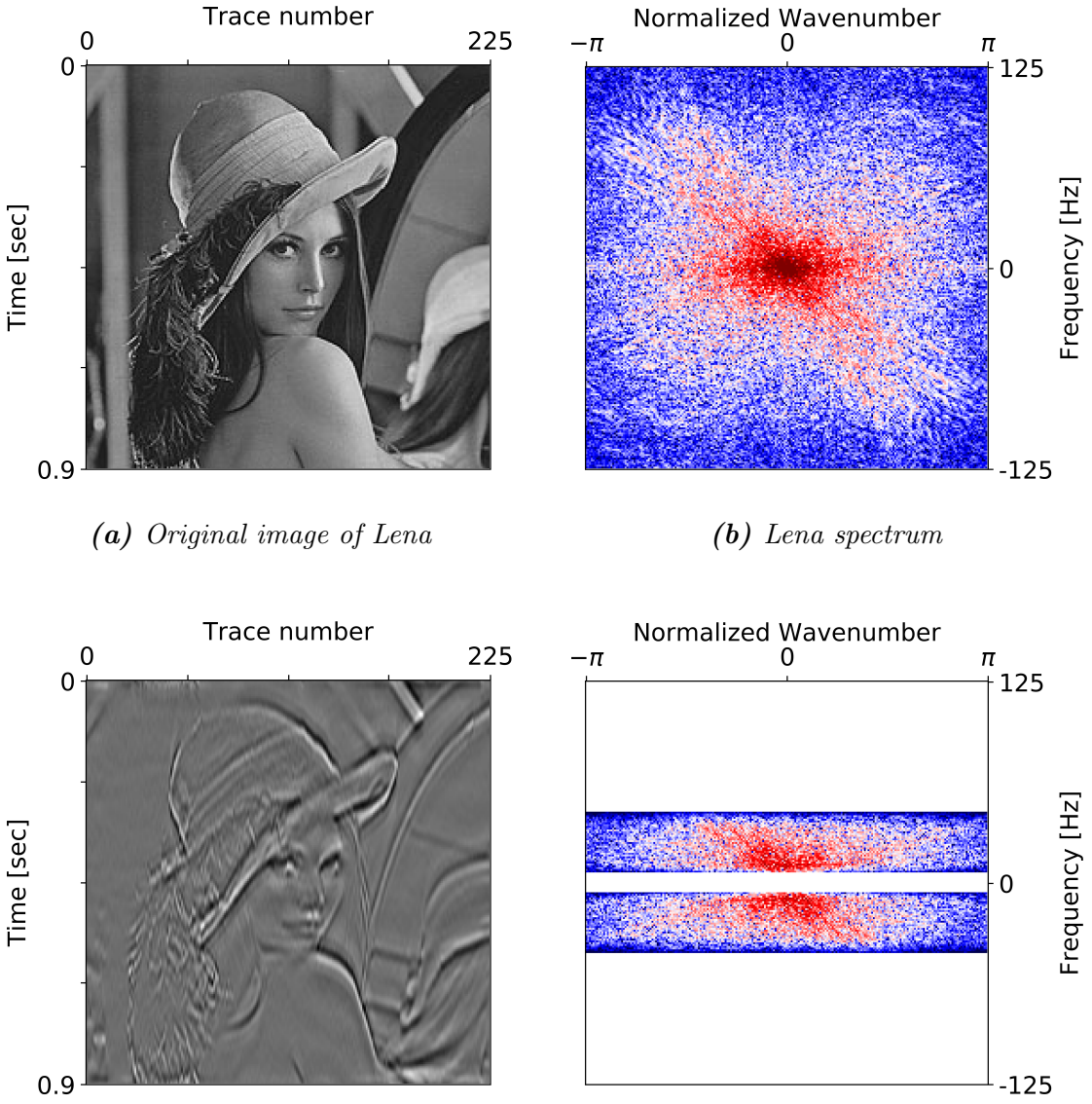
There are multiple different formats to store image data, but the most popular formats tend to be JPEG (Joint Photographic Experts Groups) and PNG (Portable Network Graphics). Both of these image formats can represent up to 24-bit color images or grayscale images. The main difference between the formats is that PNG is normally a lossless format, while JPEG is normally a lossy format (Santa-Cruz and Ebrahimi, 2000). PNG with a lossless format will behave better for image graphics containing hard transitions, while JPEG with a lossy format will be better suited for natural photography. When comparing seismic data to conventional image data the image format is irrelevant as they contain almost the same information. When referring to a conventional image in this section, either PNG or JPEG are representative formats.

Marine seismic data represents acoustic wave recordings, and are therefore different from conventional images. A standard or conventional image has only positive values ranging from $0-255$ per channel with a color depth of 3, thus $256^3 = 16777216$

possible colors. Seismic data values range from approximately $-3 \cdot 10^4$ to $3 \cdot 10^4$ with a color depth equal to 1 since seismics are only greyscale. Another important aspect about seismic data, compared to normal images is its frequency content. A conventional image contains higher frequencies than seismics. Deep seismic, as can be viewed in Figure 2.7, typically will be limited to a frequency range of $5 - 40Hz$.

In case of a conventional image or photography the frequency content is much higher. The source is now replaced by high-frequency natural light and the final image resolution obtained is determined by the camera system. To visualize the basic difference in features between a seismic and conventional image we follow the idea of Liner (2000). The starting point is the commonly used Lena image shown in figure 5.1a. We now convert the axes of this image to seismic measurables. Thus, the horizontal axis will represent trace number (running from 1 to 250), and the vertical axis recording time. In this case a temporal sampling interval of 4 msec has been employed. Figure 5.1b shows the corresponding FK-spectrum of the Lena image (obtained after a 2D Fourier Transform along trace and time axes). To simulate the characteristics of a seismic image, the spectrum is filtered to seismic bandwidth employing a bandpass filter defined by $[5, 10, 30, 40]Hz$, see Figure 5.1d. After applying a double inverse Fourier transform the seismic version of Lena is obtained as shown in Figure 5.1c. From direct comparison between Figures 5.1a and 5.1c the following observations can be made:

- Due to the missing low frequencies, the are shading is lost in the 'seismic image'.
- Due to the missing low frequencies, there is a loss on steeply dipping information in the 'seismic image'.
- Due to the missing higher frequencies, the resolution of the 'seismic image' is much poorer.
- Due to the missing higher frequencies, the finer details are lost in the 'seismic image'.



(a) *Original image of Lena*

(b) *Lena spectrum*

(c) *Seismic Band Lena*

(d) *Filtered Lena Spectrum*

Figure 5.1: *Figure (a) visualizes the well known Lena image. The spectrum of Lena can be viewed in Figure (b). Figure (d) is the Lena spectrum filtered to seismic bandwidth [5, 10, 30, 40]Hz. Figure (c) visualizes the seismic band Lena. Adapted from Liner (2000).*

Thus major differences exist between a seismic and conventional image. Direct use of ML design tailored for denoising of conventional images will likely not work in an optimal sense if employed for seismic data. This will be demonstrated in chapter 6 of this thesis.

Figure 5.2: *Figure illustrating augmentation techniques. All images are of the same tennis balls, but they are all augumented in a specific way. From left to right: Original, rotated, translated, scaled, clipped.*

5.1.2 Data Augmentation

In many cases, the dataset available, is rather small in size. Such datasets will likely underfit or overfit on the model. This will in turn tend to limit the possible performance of the network. Despite a well designed network model, without a good dataset to go with the model, it is rendered useless. An important characteristic of untrained neural networks is that a small change in an image, makes it appear as something completely different to the network. If the dataset is too small, one may therefore utilize something called data augmentation. Data augmentation is a scheme which expands the dataset without substantially changing the characteristics of the samples (Taylor and Nitschke, 2017). Figure 5.2 visualizes common augmentation techniques. They include: rotation, translation, scaling and clipping. Rotation is an augmentation technique which is self-explanatory. The image is rotated a certain degree which may significantly change the way the network views the image. Rotation works fine if the image is squared, but once the width differs from the height it requires extra editing before the network can handle the image. Translation is another operation where the image is shifted certain steps. If this shift is large enough, the network will no longer fetch the same features at the same filter locations, meaning it will regard the image as something else. Scaling and clipping are other operations where the image is either zoomed out, or cut creating an effect of zooming in. Both of these approaches require extra editing before the image has the same size as the original data. Data augmentation is a common tool used to expand the size of the dataset.

Permutation

Data Augmentation is a good way of increasing the size of a dataset, but depending on the dataset, there may be more clever ways of increasing the size as well. If the dataset consists of clean files and noise files, where the noise is pure noise, permutation is a possibility. The permutation approach is simply to combine all clean files and all noise files in every possible way. The seismic data mainly used in this thesis consist of clean shots gathers and pure noise gathers. Assume the dataset consists of 500 clean files and 500 noise files. If they are combined with permutation, each noise file is added to each clean file. This results in $500 \cdot 500 = 2.5 \cdot 10^5$ files, thus greatly increasing the size of the dataset. The downside of the permutation approach is that the dataset has to fit certain criteria for permutation to be possible. Although it might seem like a type of data augmentation, permutation is not data augmentation. The dataset is never changed in a specific way, it is simply reorganized, but in a way where it may remove the necessity of any data augmentation.

When the combination and data augmentation is completed, the dataset can be regarded as ready to use. To train a model, the user requires three different datasets. The most common approach is to split the current dataset into three subsets, namely: training, validation and test datasets. A common distribution of data in the three subsets are: 80% of data in the training set, 15% of data in the validation set and the remaining 5% in the test set often referred to as the Pareto Principle (Pocatilu et al., 2010). Once these subsets are created, they have to be kept separated and independent for the remainder of the process to gain statistically reliable results. These subsets are, as their names suggest, used for training, validation and testing.

5.2 Training, validation and testing

To arrive on a network model which handles specific data as desired, it has to be trained. The training of a neural network is a stepwise process iterating multiple times through the dataset. A full iteration is referred to as an epoch and is visualized in Figure 5.3. Each epoch starts with the network being fed one batch, often referred to as mini-batch, of data from the training dataset. The size of a batch is user defined and should often be as large as possible to gain the best results (Smith et al., 2017). The network passes the batch of data through the network and calculates the loss of

the output compared with the ground truth. The network then backpropagates the errors through the network where the optimizer updates weights and biases based on the gradient of the loss, as referred to in chapter 3. These steps are repeated until the entire training dataset has been iterated through. Once there is no more data in the training dataset, the validation begins.

Validation is more or less a repeat of the training regime, but without the optimization of parameters in the network. The network is being fed batches from the validation dataset which is fed through the network and the loss between output and ground truth is calculated. Once there is no more data left in the validation dataset, validation is complete. Validation itself does nothing for the network, but it is an intermediate measure of how the network is performing (James et al., 2013). The training and validation datasets are two different datasets and check whether the network is overfitting to the training dataset or not. Overfitting means that the network is tuned to a specific dataset and will not perform well on any other data. A way of detecting overfitting is if the training loss is decreasing while the validation loss is stationary or increasing. To avoid overfitting, the robustness and versatility of the model has to be increased. Introducing Batch Normalization or Dropout is a common way to reduce the chance of overfitting. Batch Normalization reduces the range of values the neurons can shift around as explained in 3 which might cause fewer spikes and increase accuracy of the model. Dropout simply means that a percentage of the information at user defined locations in the network is removed to reduce the impact of specific layers.

Once both the training and validation datasets have been fed through the network, one epoch is completed. It is common, and typically necessary, to run multiple epochs for the network to gain good results. Once the user is satisfied with the performance of the network, the training is completed. Although the loss functions indicate that the model performs well, it is still important to test the model on a separate dataset to make sure it actually performs as indicated. The testing is separate from both training and validation where the biggest difference is that the network no longer receives the ground truth. The testset is fed into the network model which then produces the results based on the weights and biases learned during training. These results represents an unbiased evaluation of the performance of the network. In this thesis, a select few samples were plotted for visual inspection of the performance on the test set.

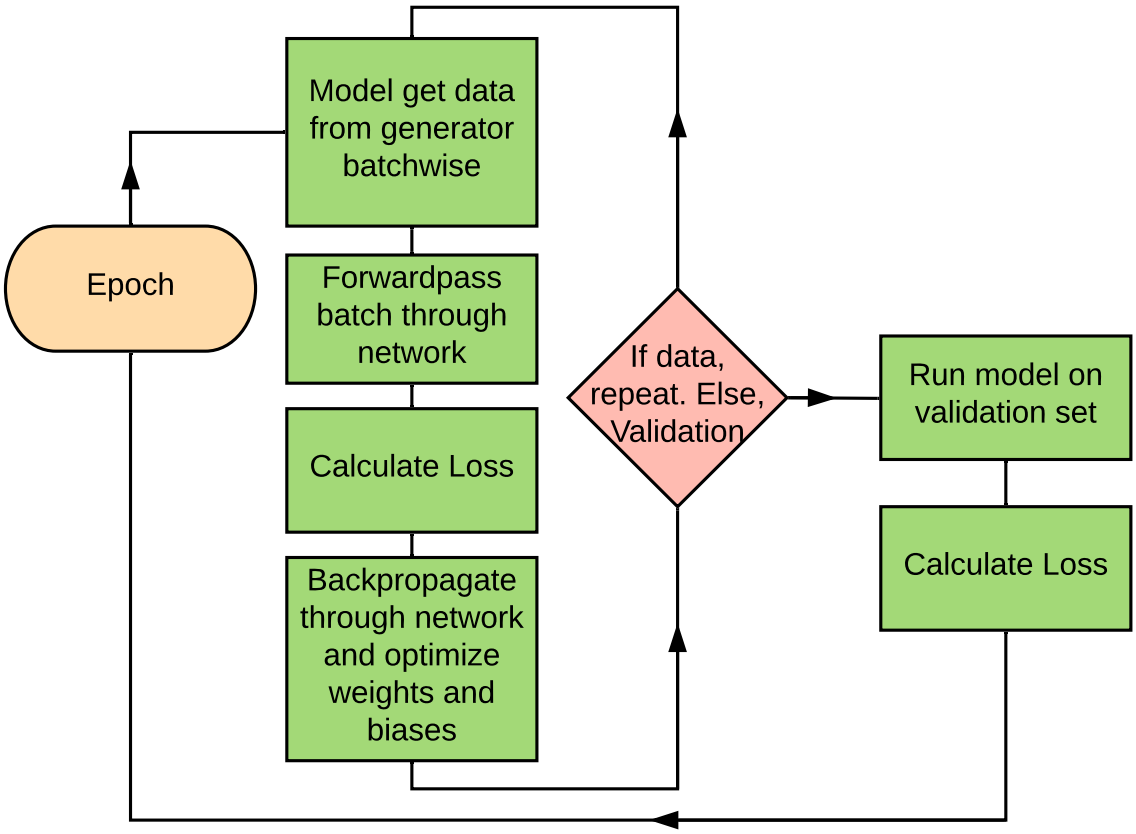


Figure 5.3: *Flowchart visualizing the training process of a network model. The training of a neural network is an iterative process revolving around epochs, marked in red.*

5.3 Creating a network model

Creating a network model might be relatively easy, but it takes some knowledge to actually create a versatile model which performs well. The coding of the network design also varies much based on what ML library is being used (e.g. Tensorflow and Keras). There are, however, certain aspects which applies for every model. The most important part of the construction phase is to adjust the network to the dataset and desired usage. In this thesis, each network model should be designed to handle large images, since seismic data are represented by large image files. Numerous network designs were tested during the work of this thesis, but they can all be narrowed down to three basic structures, namely: Classification CNN, Autoencoder (AE), and No downscaling CNN (NDCNN).

5.3.1 Classification CNN

The main job of a classification network, as mentioned in Chapter 3, is to label input data and try to fit it to the correct class. A classification network can be designed in various ways, but in this thesis a convolutional approach is used since the input data are images. When data is fed into a network, it is read as a three dimensional array of numbers. In this thesis work, the purpose of the classifier was to detect whether a seismic shot gather contains noise or not. The output of the network should be a one-dimensional array containing two elements: noise contaminated and clean. This in terms means that the network has to reduce the number of samples throughout the network as it learns the structure of the data. The network has to label the image as noisy or clean correspondingly. Since the network only has two classes to divide the data into, the problem can be interpreted as a binary problem: True or False.

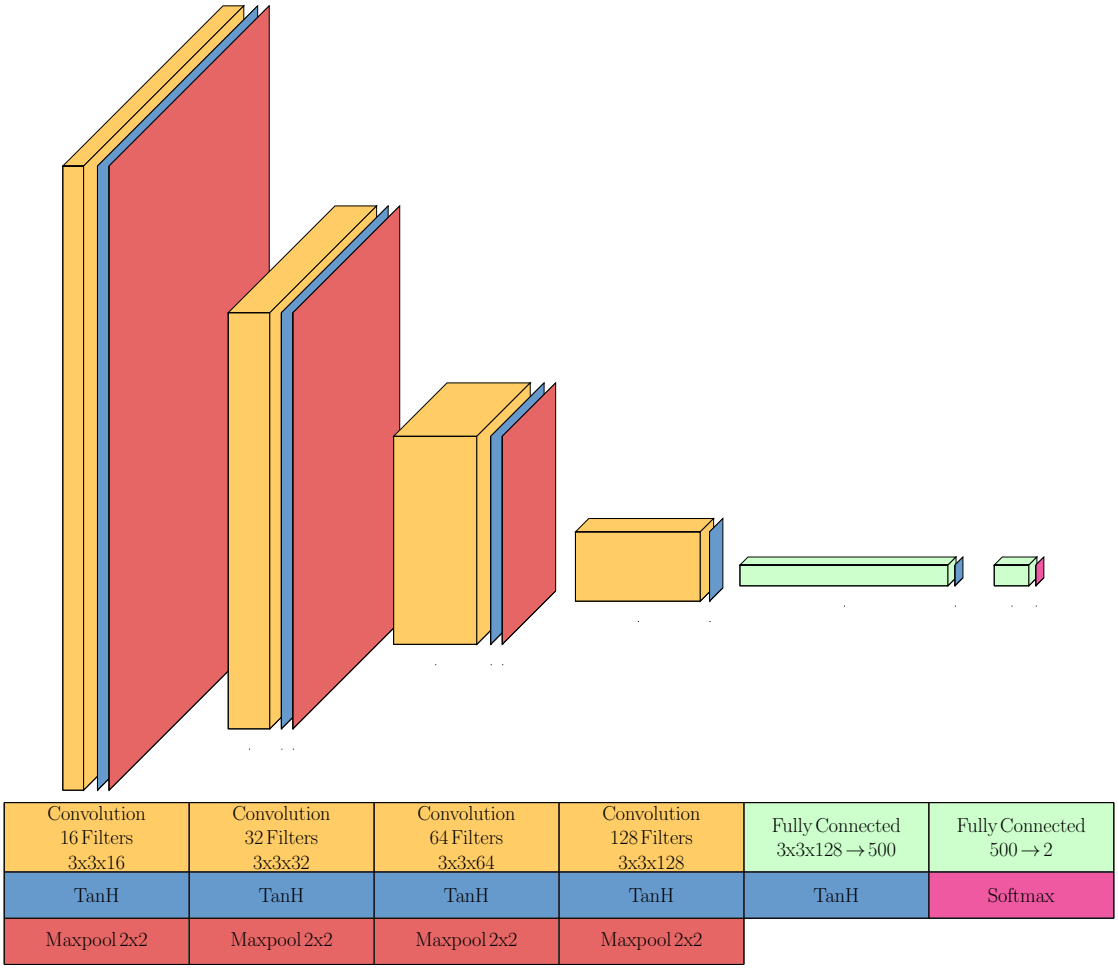


Figure 5.4: *Figure visualizing the structure of the classification CNN.*

One classification network used in this thesis is shown in Figure 5.4. The model consists of a 9-layer network where the image decreases in size throughout the layers. The first part of the network is a CNN consisting of 4 convolutional layers with a TanH activation function. All filters have the same size of $[3 \times 3]$ where each layer has double the amount of filters as the previous layer, meaning the number of filters for each layer from conv layer 1 to conv layer 4 is $[16, 32, 64, 128]$. Between each convolutional layer is a max pooling layer halving the size of the image before it is fed further into the network. The second part of the network is represented by two consecutive fully connected (FC) layers. The data is flattened and fed into the FC layers outputting a two dimensional vector of size $[1 \times 2]$ representing the two classes: clean data and noisy data. The loss function is Binary cross-entropy since the problem is a binary problem.

Convolutional (CV) layers are used both because the input data are images, since CV layers have less parameters than FC-layers (chapter 3), and because they can detect a pattern between neighbouring pixels. Since the network is only going to tell whether the the data is noise contaminated or not, it only needs specific features, such as the characteristics of the noise. Max pooling is applied to reduce the size of the data between each CV layer as well as enhancing the most dominating features. The size reduction between each CV layer forces them to fetch different features as the input to each layer is substantially different, increasing the robustness of the model. The fully connected layers are useful once the size of the image is reduced. They calculate and return the probability of the two different classes based on the calculations done by the convolutional layers.

5.3.2 Autoencoder

Autoencoders (AE) are commonly used for image processing in machine learning. They tend to be built up by convolutional layers, but have a specific structure separating them from other types of network. Opposite to the classification case, where the image is downscaled through the network (Figure 5.4), an AE downscales to a latent layer and then upscales to original size. The essence of an AE might not strike as a good architecture for image processing, but have proven to yield good results in various cases (Gondara, 2016; Chaitanya et al., 2017; Xie et al., 2012).

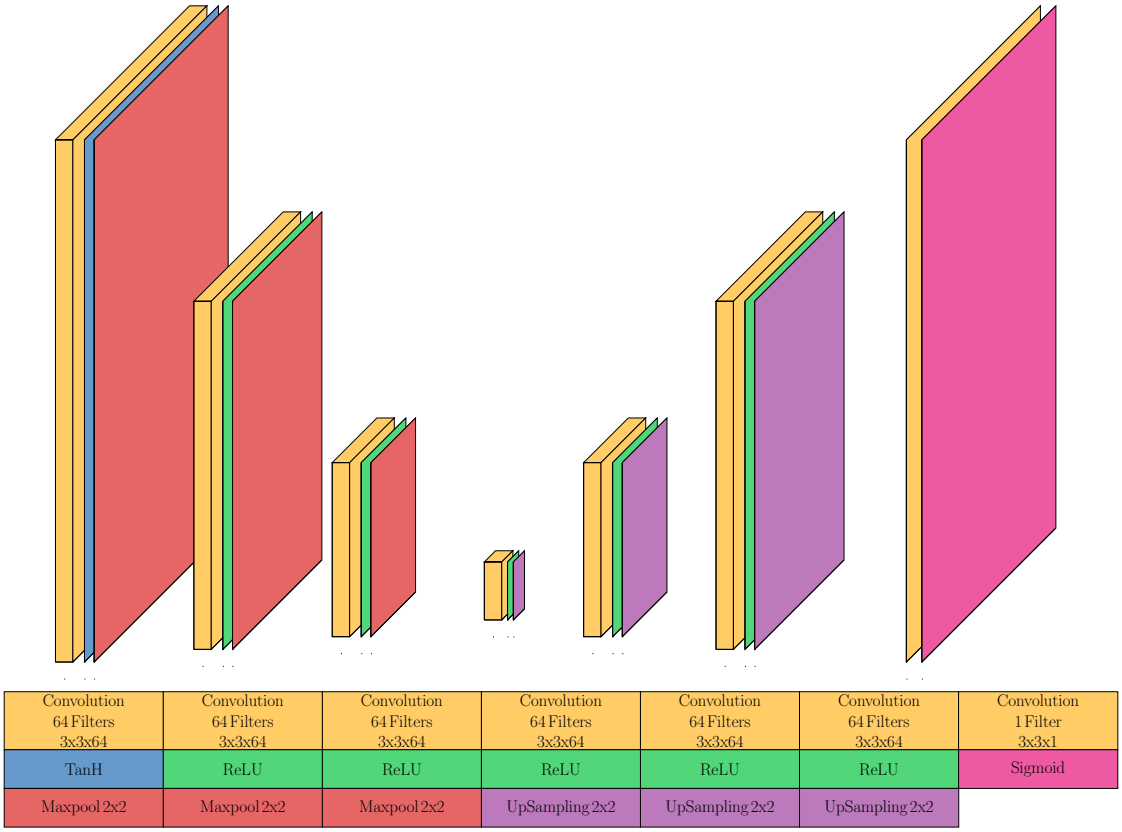


Figure 5.5: *Figure visualizing the structure of an autoencoder.*

The theory behind an autoencoder is to represent data at a latent level. To do this, the network has to compress the data, thus separate important features, while omitting less important features. A way of regarding the process is looking at it through nodes. Assume the input to the network is 1×5 while the latent layer is 1×2. All the entries in input array will represent one node in the input layer, but while the input layer has 5 nodes, the latent layer has only 2 nodes. To be able to represent the input at the latent layer, parts of the data has to be removed, while keeping the main structure. The network has to learn which features in the data are important and which are not. This process is a form of compression. If the dataset has certain features which repeats in many of the images, the AE will learn said features and omit the ones which tend to be random or seldom. After the data is downscaled to the latent level, the network tries to recreate the data minimizing the difference between the output and the input. The characteristics of an autoencoder fits well with noise removal. If the noise appearance varies from image to image, it will be regarded as a feature which is not important and can therefore be removed. The AE network was implemented as an example of the

misperformance of conventional networks when applied to the seismic case. Due to the major differences in the characteristics of a seismic and conventional image, and also due to the more complex type of noise in the seismic case the performance of a standard AE is not expected to be good in case of seismic denoising.

A typical AE network tested in this thesis can be seen in Figure 5.5. It consisted of 7 convolutoinal layers where the first and last layer used respectively the TanH and Sigmoid activation functions and the intermediate layers used ReLU. The network used filters with size $[3x3x64]$ throughout the network, except for the last layer where number of filters were reduced to 1.

5.3.3 No Downscaling CNN - NDCNN

As data is passed through a convolutional neural network, each layer receives slightly changed data given the operations made by previous layers. Because of this characteristic, each layer should focus on different features in the data with no downscaling present. By keeping the full image size throughout the network, the chance of loss is reduced, which is an important aspect for seismic images. Each time an image is compressed, certain information is lost and the more the data being downsampled, the higher the chance for that information to be signal, and not noise. This is the essence behind No Downscaling CNN (NDCNN). Instead of having to recreate the data from a compressed version, the network only has to augment the data and remove the specific undesired parts e.g. remove the noise while keeping the signal intact.

Figure 5.6 visualizes a typical NDCNN model used in this thesis. The network consists of 8 convolutional layers. The last layer has no activation function and only one filter to generate a single output. Each intermediate convolutional layer is followed by a Leaky ReLU activation function and a Batch Normalization layer before being passed through to the next block. Batch Normalization along with Leaky ReLU are used in the model to potentially gain faster learning and reducing the chance of "dead neurons" as explained in chapter 3. A residual layer is added between conv layer 2 and conv layer 3. The first two convolutional layers have large filters to try to capture the large scale features in the data by increasing the field of view. This is due to the shear size of the input data. The output of conv layer 2 is added with the input data to make sure that the remaining part of the network catches the more local features present in the data.

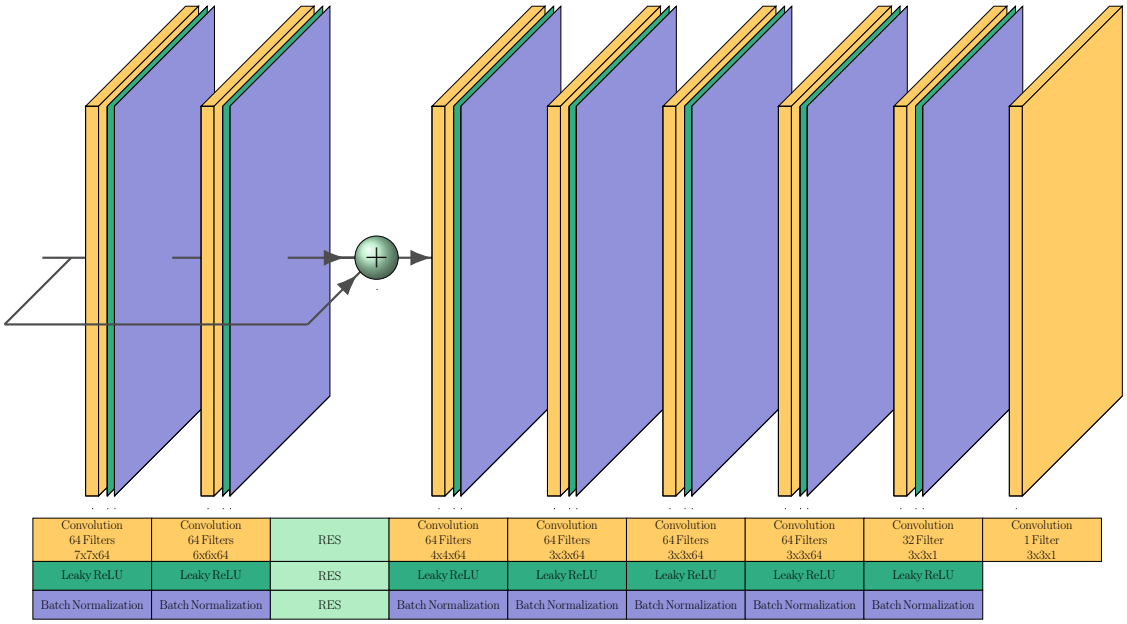


Figure 5.6: *Figure visualizing a NDCNN model.*

5.3.4 U-NET

The U-NET is a model which uses characteristics from both NDCNN and AE. The network compresses images to a latent level in the same manner as an AE network, thus removing random features in the data. The data is then re-sampled back to original size where less noise is present. The compression is a good way of removing noise, but it might cause unwanted loss. NDCNN was a way of counteracting the potential loss from AE, but a model with no downscaling will demand more processing power. U-NET utilizes the characteristic downscaling as seen in an autoencoder, but has residual layers passing higher resolution outputs between each upscaling, as can be seen in Figure 5.7. The model will receive efficiency and denoising capabilities from the AE structure, but keep the structure of the data as it receives higher resolution outputs after downscaling.

Figure 5.7 visualizes a U-NET model used in this thesis. The network 7 convolutional layers each followed by Leaky ReLU, except for the last layer which has no activation function. The first two convolutional layers employ max pooling to reduce the size of the data. There are two consecutive layers at latent level, where the last employ upsampling. Before passed to the next layer, a residual layer is employed combining data from the previous layer of same size. This is repeated for the next upscaling process, as seen in Figure 5.7. Once the data reaches original

size, it is passed through a convolutional layer, and then fed to the output layer. The first two convolutional layers employ large filters, as NDCNN, to capture large scale features.

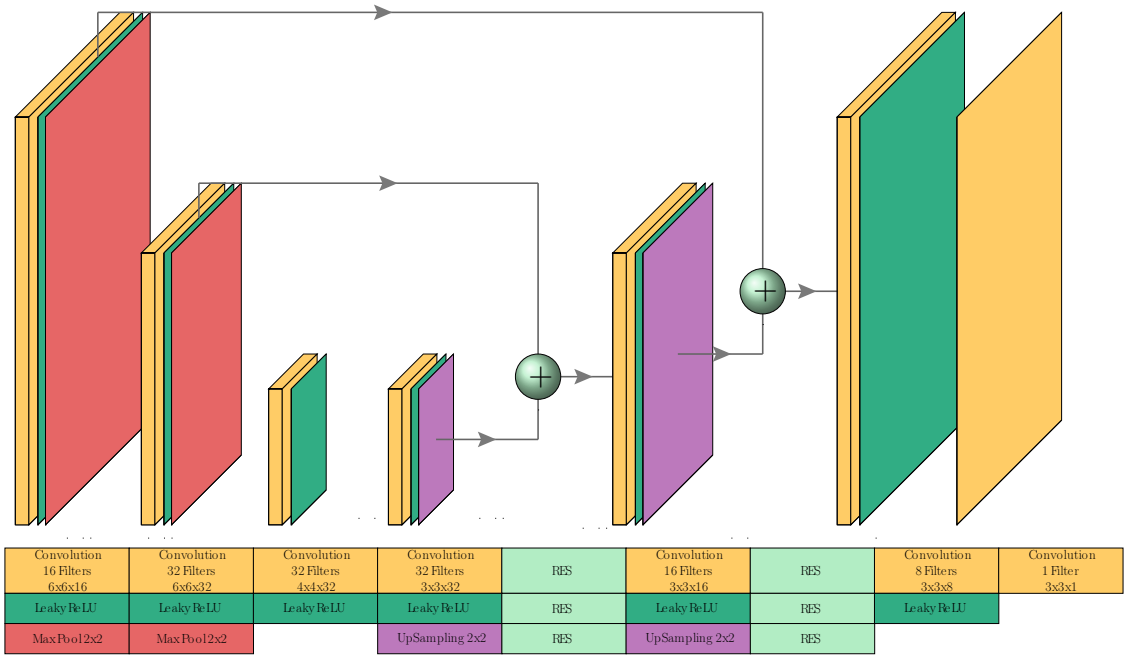


Figure 5.7: *Figure visualizing the structure of a U-NET model.*

5.4 Application of the network model

When both the dataset is correctly sorted and the network model is created, the model can be applied to the dataset. Figure 5.8 is a visualization of the full workflow needed when starting from scratch with a dataset to a final output from the network. Figure 5.3 illustrates in more detail the training step, as explained in section 5.2. To make the network read the data, a data generator has to be created.

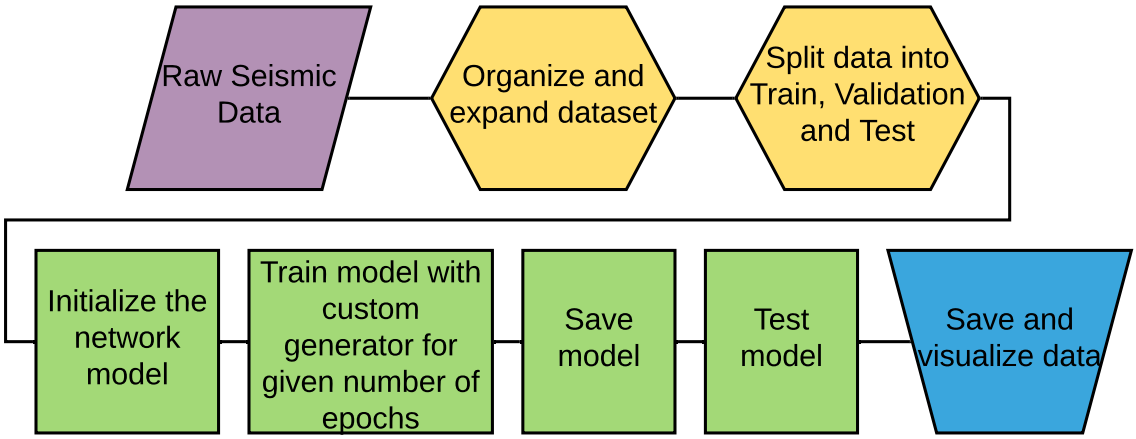


Figure 5.8: *Flowchart illustrating the process of creating and training a model. It is generalized, but has a few aspects which are specific to this thesis.*

5.4.1 Data generator

To initiate a model, the data has to be fed to the network in a correct manner. There are multiple ways of data feeding, but it is important to optimize this process as it tends to become a bottleneck. As explained in chapter 4, each core on a GPU is slow, but a GPU has a large amount of nodes and can thus handle much data at the same time. The data is stored on a hard drive and has to be fed to the GPU faster than the GPU can compute the data to reduce the running time. If the GPU has to wait for data, it has nodes which are not computing. This means that the model is not optimized and will be slower than necessary. It is common for machine learning libraries to include pre-built data generator functions which feed data to the GPU with little to no latency. These pre-built generator functions tend to be file restricted and might not fit for the desired file format. That was the case for this thesis, and the data generator function had to be custom created.

When writing a custom made generator function, every aspect has to be considered to increase the efficiency. Python is a versatile programming language, but it can be rather inefficient if all code is written in the native language. However, there are multiple libraries which can, if used correctly, greatly increase the execution speed of a program. The library used to create the custom made generator was NumPy, which is a library adding support for multi-dimensional arrays and ma-

trices, and linear algebraic operations running a C backend (Oliphant, 2006). The custom made generator is almost entirely written with NumPy objects and feed data to the network with little latency.

5.4.2 Saving results, testing and visualizing data

The training duration of a neural network can vary considerably based on the size of the network and dimensions of the input data. Due to this, it is important to save the network model both during training and after training is completed. The saved model will contain updated weights, biases and status of optimizer. The training process might shut down unexpectedly and cause hours of training time lost if no intermediate saving was implemented. Once the training process is completed, the finished model is easily tested on a new dataset; the test set. The results gained from the test set are the most important results, as the loss function might not pick up on certain errors.

Visualizing the data from testing is essential to gain knowledge about the strengths and flaws of a model. If 90% of the pixels in an output image resemble the ground truth, the network will likely yield a good loss number, but if the remaining 10% contains artifacts, the actual performance of the model might still be poor. This is important when using ML for seismics, as low amplitude reflections contain valuable information. To try and capture all aspects of the data, four different steps were visualized for each test image. The ground truth, contaminated input, output, and the difference between ground truth and output. The Fourier spectra of the images were also plotted to gain insight in the performance in multiple domains, thus increasing the knowledge of the performance of the network model.

5.4.3 History

The timeline of this thesis is an important aspect to prove how and why the final model came to be. The first step of this thesis work was to investigate whether a computer could or could not recognise the noise present in the data. If the computer could not differentiate between clean and noise contaminated data, denoising could be challenging. The data had to be structured in a suitable way for the computer to read them and then be fed to a classification network, similar to the one visualized in figure 5.4. As mentioned in section 5.4.1, it is common for machine learning

libraries to have pre-built generator functions. The seismic data was stored as PNG-files and fed to the network by utilizing one of these generator functions. Model yielded resonable results, proving that the network could differentiate between noise contaminated data and clean data.

The next step was to create a denoising network. The data was zero padded to keep the size through the layers. Given the common usage of autoencoders as well as their characteristics, mentioned in section 5.3.2, it was a reasonable network model to test. The initial AE model was similar to the model represented in figure 5.5, but with slight changes in activation and loss functions. The loss function used was binary cross-entropy (see Section 3.2.3). This design gave non-optimal results. Changing the loss function to Mean Square Error improved the results, but they were still not optimal. To try and increase the robustness of the model, the data had to be fed to the network in a better manner. Instead of the data being fed to the network as PNG-files, the custom generator fed them in as NPY-files which is a NumPy format. This opened the possibility of using a permutation approach on the dataset, greatly increasing the number of input samples. A dataset containing 10^6 samples is large, and therefore 10% of the dataset was used. The results from the autoencoder improved, but it removed too much signal in the compression during the network, resulting in images looking synthetic and lacking essential information.

To counter for the data loss during compression, another design approach was implemented and tested, namely NDCNN. Instead of compression the image and then decode it, the image was kept at full size throughout the network. The network then only had to remove noise, instead of recreating the entire image. This network model had the same amount of layers as the autoencoder, but without the downscaling. It was also evident that the zero padding caused edge effects and a mirror padding function was implemented to counter these effects 5.9. The results from NDCNN yielded slightly less signal loss, but a lot more residual noise. To capture the full range of all frequencies, the first layers of the network had large diluted filters. A residual layer connecting input data and output from the large size filter layer was added to ensure that the last part of the network has a full field of view. Introduction of batch normalization and leaky ReLU produced results with close to no signal loss, but with a good amount of noise removed. This model can be viewed in Figure 5.6.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	0	0
0	0	6	7	8	9	10	0	0
0	0	11	12	13	14	15	0	0
0	0	16	17	18	19	20	0	0
0	0	21	22	23	24	25	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

(a) *Zero padding*

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
23	22	21	22	23	24	25	24	23
18	17	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13

(b) *Mirror padding*

Figure 5.9: *Figures (a) Zero padding and (b) Mirror padding visualized.*

NDCNN required long training time, and the results still had some residuals left. A third denoising design was implemented employing the characteristics of both NDCNN and AE combined, namely a U-NET. This network structure contained the compression structure of an AE, but had residual layers inheriting from earlier higher resolution layers to counter for unwanted information loss during compression. This model was much more efficient and gave significantly improved results. The U-NET can be viewed in Figure 5.7

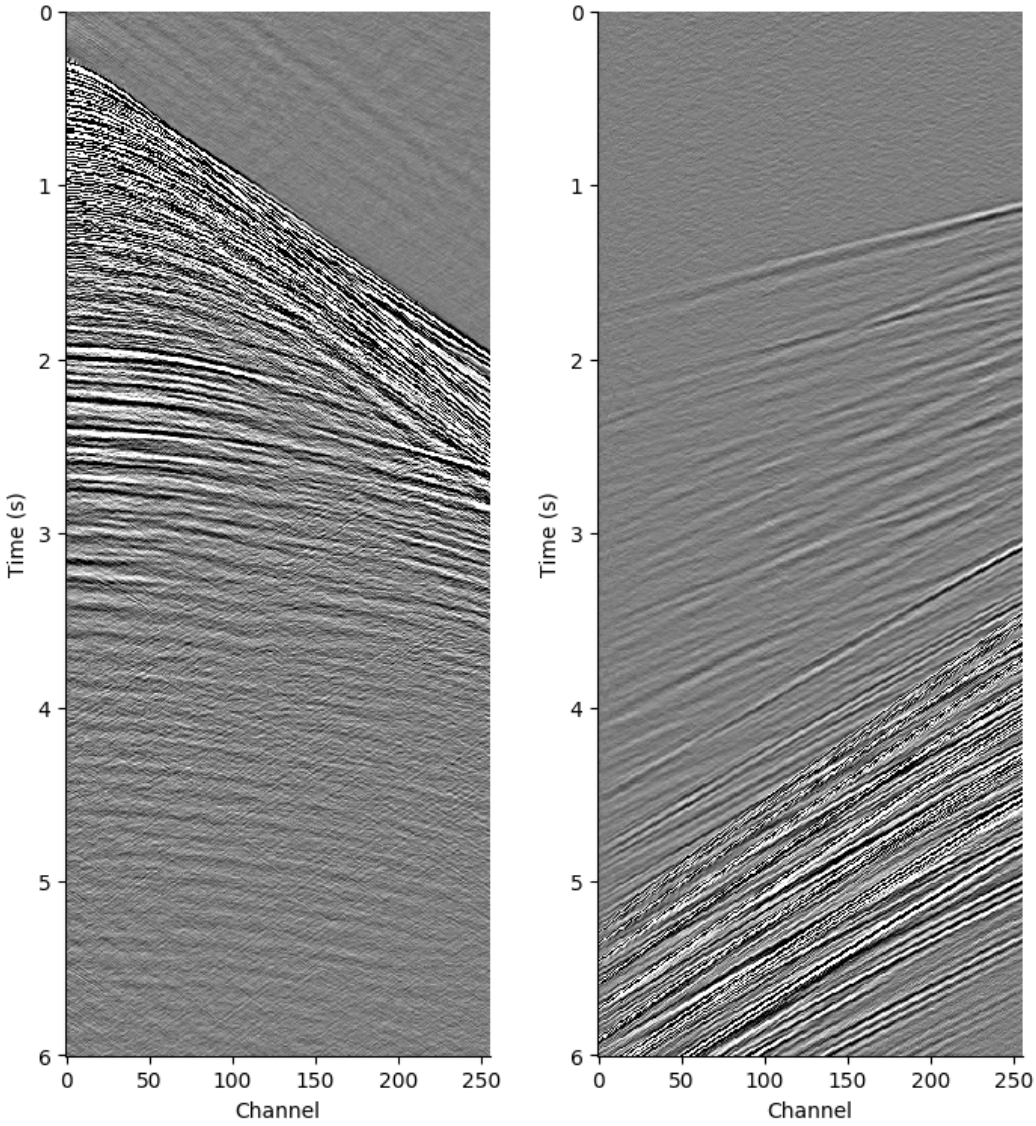
6 | Results

This chapter presents results from the various network models applied in this thesis work. The dataset will be presented as well as characteristics of the different network models.

6.1 Dataset

The dataset used in this thesis is marine seismic data acquired by CGG in the North Sea. Such data is always contaminated by various types of noise to some extent (as explained in Chapter 2) which might require denoising before used in neural networks. This problem was avoided since CGG had "pure" SI-noise was recorded separately. The data used in this thesis consists of two lines of marine seismic data where one line contains 482 industrially denoised, and therefore nearly noise-free marine seismic records, and the second line contains 800 records of "pure" seismic interference (SI) noise. The noise records, as can be viewed in Figure 6.1b, were acquired by two vessels traveling in opposite direction with one vessel shooting and one vessel recording. The data was divided into 8 second gathers, coinsiding with the shot point interval of the shooting vessel. All data was recorded with a $2ms$ sampling rate. The clean data, as can be viewed in Figure 6.1a, was both shot and acquired by the towing vessel in good weather conditions and put through an industrial processing denoising workflow.

Seismic datasets with a $2ms$ sampling rate hold 500 time samples for each second. It is common to view events down to at least 6 second two-way travel time (TWT) for North Sea data thus resulting in 3000 time samples for each trace. This is considered large compared to conventional images in neural networks, see Chapter 5. The data was therefore downsampled to $4ms$ to reduce the size of each shot record entering the network. The data had an original record time of 9 second TWT, which was cut down to 6 second to further reduce computation time and memory consumption. The final input size of the seismic images was $1500x256$, as far offset traces were omitted to increase computational efficiency.



(a) *Clean data* ***(b)*** *Noise data*

Figure 6.1: *Example shot gathers from the datasets where (a) shows a seismic shot record from the "clean" dataset and (b) shows "pure" SI-noise from the noise dataset. The shotgathers presented in this figure will be used in numerous examples in this chapter.*

6.1.1 Data scaling

As referred to in Chapter 3, neural networks ideally works on data scaled to $[0-1]$ range with fairly uniformly distributed values. The four different scaling methods used in this thesis were: normalization (Figure 6.2a and Equation 6.4), thresholding (Figure 6.2b and Equation 6.3), square root (Figure 6.2c and Equation 6.2) and cubic

root (Figure 6.2d and Equation 6.1). In these figures the output from the network is marked with blue, and the difference between input and output marked with red. It has to be mentioned that the last three scaling methods will not normalize the data between 0 and 1. Normalization was therefore applied afterwards to fit the data to the desired range. All versions were run on similar models displaying the same shotgather with the same noise. These scaling methods were tested to narrow the dynamic range of the amplitude values in the seismic data. The figures presented are cutouts from larger sections. These sections are included in the Appendix A (Figures A.1 - A.4), and will be referred to correspondingly.

There are clear differences between the scaled models. However, the most striking difference is between cubic root and the others. The output from cubic root, as seen in Figure 6.2d, is completely washed out. The amplitudes in the entire image are scaled up and almost no apparent geological information is visible, as can be seen in Figure A.1. The difference plot A.1 contains more geology than the output and it seems likely that the dynamic range became too narrow for the network to learn features from the data. The cubic root scaling was implemented as follows

$$f(x) = \begin{cases} \sqrt[3]{x}, & x > 0 \\ -\sqrt[3]{|x|}, & x < 0 \end{cases} \tag{6.1}$$

where x is a matrix and the negative values are forced positive before applying the cubic root to remove any imaginary numbers.

The square root performs better than cubic root, but is still sub-optimal as there are a lot of residual noise left in the output, Figure 6.2c. The noise has areas with quite high amplitudes and the entire section shows washout suppressing the underlying geological signal. There are many details present in the difference, which demonstrates that the washout is rather significant and introduces a lot of artifacts in the data. Figure A.2 shows that the output reconstructs reflections between 2-3 second TWT, but the difference shows substantial loss in the same time range. The geological signals underlying the intercepting SI-noise are partly reconstructed, but there are still apparent reflections visible in the zoomed difference section, Figure 6.2c. There is a likelihood that the issue with the cubic root yields for the square root as well, where the dynamic range might become too narrow. The square root

was implemented as follows

$$f(x) = \begin{cases} \sqrt{x}, & x > 0 \\ -\sqrt{|x|}, & x < 0 \end{cases} \tag{6.2}$$

where x is a matrix and the negative values are forced positive before applying the square root to remove any imaginary numbers.

The thresholding preformed much better than both of the root functions. Seismic data has a large dynamic range, where certain areas are high in amplitude, but not of much interest in reflection seismics; such as the refracted events. Thresholding was applied to force the high amplitudes down and was implemented as follows

$$f(x) = \begin{cases} 0.01x, & |x| > 500 \\ x, & |x| < 500 \end{cases} \tag{6.3}$$

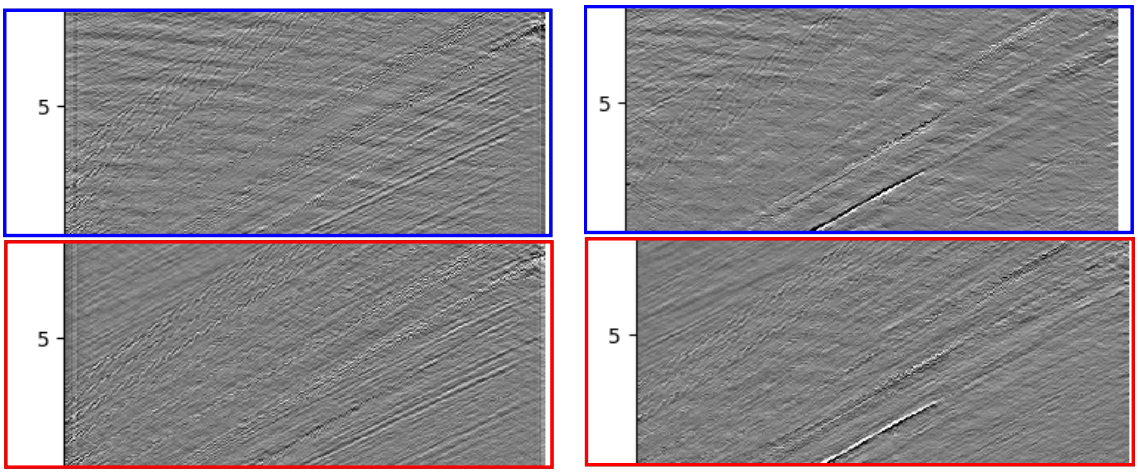
where all values exceeding an absolute value of 500 were multiplied by 0.01 effectively scaling them down by a factor of 100. The results from the thresholding can be viewed in Figures 6.2b and A.3 where it is evident that it performs better than both the square and cubic roots. Residual noise exists in the output, but the noise is significantly attenuated. The underlying geology is kept relatively well, but still geological information is lost, as can be seen in the output, Figure 6.2b.

Normalization was, as mentioned, applied in combination with all three scaling methods discussed above to make sure all values lie between 0 and 1 before passed to the network. The normalization applied is defined as follows

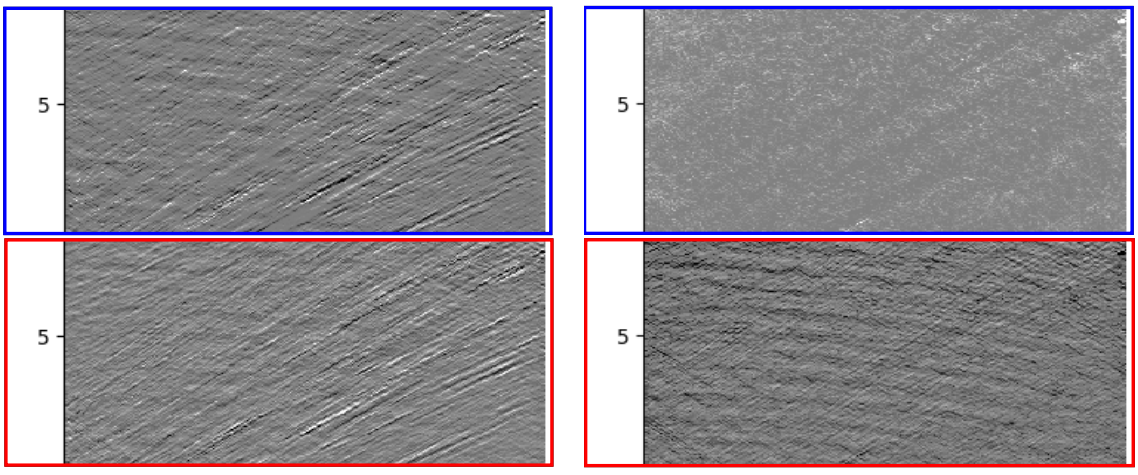
$$f(x) = 0.5 \left(\frac{x}{max(|x|)} + 1 \right) \tag{6.4}$$

where the input, x, was divided by the absolute maximum value of input and then shifted to lie between 0 and 1. The result from the use of only normalization can be viewed in Figures 6.2a and A.4. Residual noise is present in the data, but it is substantially attenuated. It is evident that pure normalization performs better than both root methods, and also slightly better than thresholding, Figure 6.2b. The residual noise in the output from the pure normalization distributes over a larger area than in case of the thresholding, but at the same time keeps the underlying deeper geology more intact. Although thresholding might remove slightly more noise, the most important aspect is to not damage the underlying signals. The spikes left in the threshold, Figure 6.2b, also have substantially higher amplitudes

than the residual in the pure normalization output. The thresholding performs better around the refracted events, compared to the pure normalization. Another important aspect favouring normalization is that the process is reversible, while thresholding is permanent. Due to these aspects, the scaling method used in this thesis was chosen as pure normalization defined by equation 6.4.

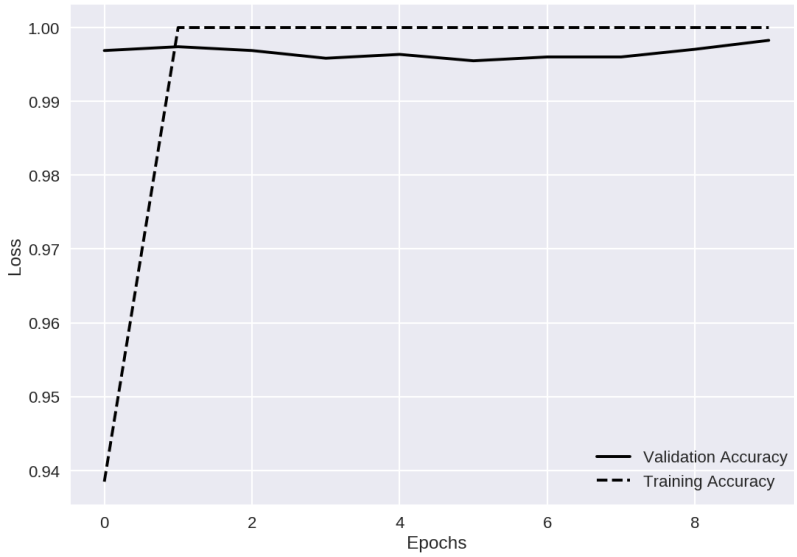


(a) *Normalization* ***(b)*** *Threshold*

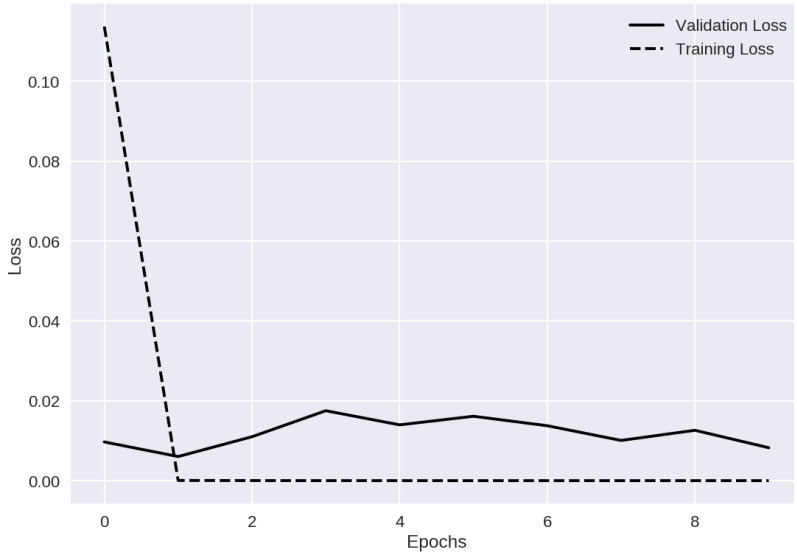


(c) *Square Root* ***(d)*** *Cubic Root*

Figure 6.2: *Figure visualizing the impact of the four different scaling methods applied to the data. Figures (a), (b), (c), and (d) respectively visualize Normalization, Threshold, Square Root and Cubic Root. The blue and red boxes show respectively output and difference (ground truth - output). The figures are cutouts from larger sections found in Appendix A (Figures A.1 - A.4).*



(a) *Accuracy*



(b) *Loss*

Figure 6.3: *(a) Accuracy and (b) loss results from the classification convolutional neural network, visualized in Figure 5.4.*

6.2 Classification

The classification network was, as mentioned in 5.4.3, created as a proof of concept to whether the network could understand the structure of SI-noise and differentiate between the noise and the signal. The network is visualized in Figure 5.4 and is a 6-layer model employing max pooling to compress the image and fully connected

layers to classify the output. Figure 6.3a visualizes the accuracy performance of the classification network where the accuracy quickly rises to almost 100% both for the training and validation set. The loss from the network can be viewed in Figure 6.3b. The training loss stabilizes at a value of approximately 10^{-7} while the validation loss oscillates slightly, but reaches a value of about 10^{-4} after 9 epochs. The classification network clearly performs well on both the training and validation datasets, reaching almost perfect results after a few epochs.

6.3 Autoencoder

Two types of autoencoder networks were tested in this thesis, further referred to as AE1 and AE2, see Table 6.1. Autoencoders are, as mentioned in section 5.4.3, commonly used in conventional image denoising.

AE1	AE2
Conv 32x3x3	Conv 64x3x3
Max Pool	Max Pool
Conv 32x3x3	Conv 64x3x3
Max Pool	Max Pool
Conv 32x3x3	Conv 64x3x3
-	Conv 64x3x3
Up Sampling	Up Sampling
Conv 32x3x3	Conv 64x3x3
Up Sampling	Up Sampling
Conv 1x3x3	Conv 1x3x3

Table 6.1: *Table listing the structure of the autoencoder models used in this thesis: Conventional (AE1) and Custom (AE2). Both models used ReLU as activation functions and Sigmoid in the output layer. Autoencoder 2 had TanH as activation function in the first layer.*

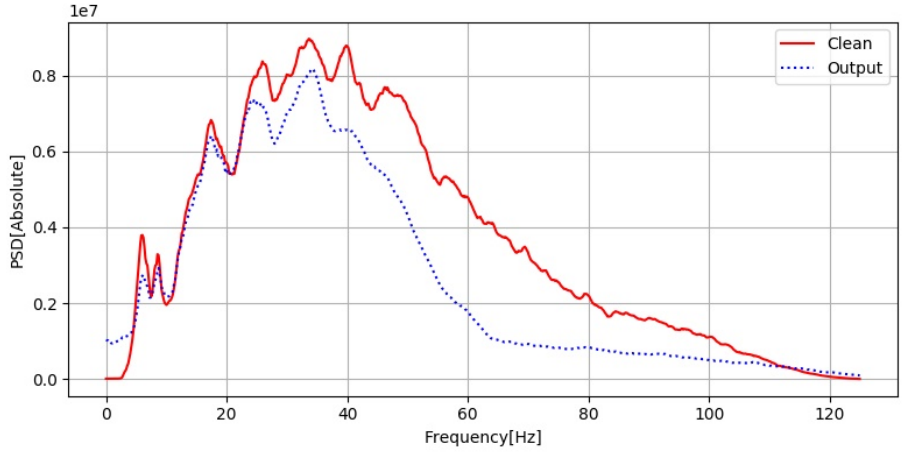
The denoising results form AE1 can be viewed in Figure 6.5. The output, Figure 6.5c, shows some SI-noise removed, but there are still significant SI-noise left in the output data. The zoomed section around 5 second TWT (two-way traveltime), Figure 6.5iii, highlights an area with significant noise residual. The noise overshadows

the underlying signal and it is difficult to see whether the signal is kept intact due to the low signal-to-noise ratio (SNR). Box vi in Figure 6.5d highlights the same part in the difference between input and output showing weak residual geological signals and high amounts of residual noise. The residual noise is pixelated which is likely an artifact introduced by the compression in the model. Boxes i and iv respectively Figures 6.5c and 6.5d highlight an area around 2-3 second TWT with strong reflections and multiples. Box i shows a seemingly clear and noise free reconstruction of this section, but on comparison with box iv in the difference shows that there are substantial loss in signal in this area. The box ii highlighted in Figure 6.5c around 4 second TWT shows a high SNR as there are little noise present in this area. Direct comparison with box v in Figure 6.5d shows that some geological information has been lost. Given the amplitudes of the input in this area, the loss is quite substantial. It is evident that the model struggles with recreating the refracted arrival, as can be seen in Figure 6.5d. Careful inspection also show edge effects along both vertical edges, creating vertical artifacts in the data. Figure 6.4a shows the frequency spectrum of the input and output. It is clear that the model struggles with recreating the frequency band as it performs increasingly worse with increasing frequency from around $25Hz$ and throughout the spectrum. The frequencies below $25Hz$ are recreated relatively well. Figure 6.4b show the loss plot for AE1, where the training loss stabilizes at $3 \cdot 10^{-5}$ after 15 epochs while the validation loss stabilizes at $5 \cdot 10^{-5}$.

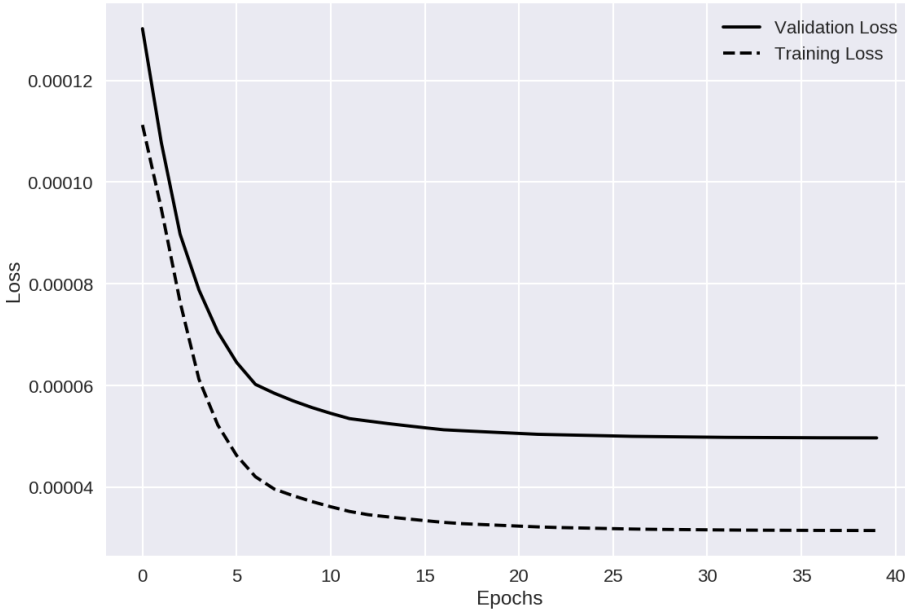
The results from the use of AE2 can be seen in Figure 6.6. This autoencoder shares many of the characteristics as seen in the results from AE1, Table 6.1. There are residual SI-noise left in the output, Figure 6.6c, introducing artifacts in the data appearing as dipping features with a constant low amplitude as can be seen in box iii in 6.6c. Comparing the zoomed sections (boxes i and iv) respectively in Figures 6.6c and 6.6d shows that the reflections around 2-3 second TWT are relatively well recreated, but with some loss. The model performs well at low amplitudes with high SNR as can be seen in boxes ii and iv in respectively Figures 6.6c and 6.6d where there are no apparent geological data present in the difference. Both the residual SI-noise and the refracted arrivals show pixelated artifacts. The frequency spectra of Figures 6.6a and 6.6c are given in Figure 6.7a, showing that the network recreates the entire frequency band relatively well, but has some deviations at higher frequencies $(70 - 125Hz)$. The loss plot for AE2 is given in Figure 6.7b where the

training loss is stable at a value of $1 \cdot 10^{-5}$ while the validation loss is increasing throughout every epoch with a final value of $1.4 \cdot 10^{-4}$ at 20 epochs.

The most apparent difference between the models is the edge effect present in AE1, Figure 6.5, which is not present in AE2, Figure 6.6. The residual noise in the outputs of the models vary as AE1 has higher amplitude loss, while AE2 has lower amplitude residual. AE2 also shows patches in the noise residual with constant low amplitudes, which is not present in AE 1. The shallow part of the SI-noise intercepting at about 1 second TWT is better attenuated in AE1 as there are more apparent residual left in the output from AE2. The outputs from both models appear synthetic, and they both perform relatively poorly. AE2, although performing poorly, performs substantially better than AE1 as there are a lot less geological information left in the difference from AE2, Figure 6.6d, compared to the difference from AE1, Figure 6.5d. AE2 also performs better denoising than AE1. The loss function indicates that AE1 performs better than AE2, but the output and frequency spectra of AE2 are noticeably better. However, AE2 over estimates the higher frequencies, producing higher amplitude signal for frequencies above 80 Hz.



(a) *Frequency spectrum of AE1*



(b) *Loss plot from AE1*

Figure 6.4: *(a) Frequency spectrum and (b) loss plot for AE1*

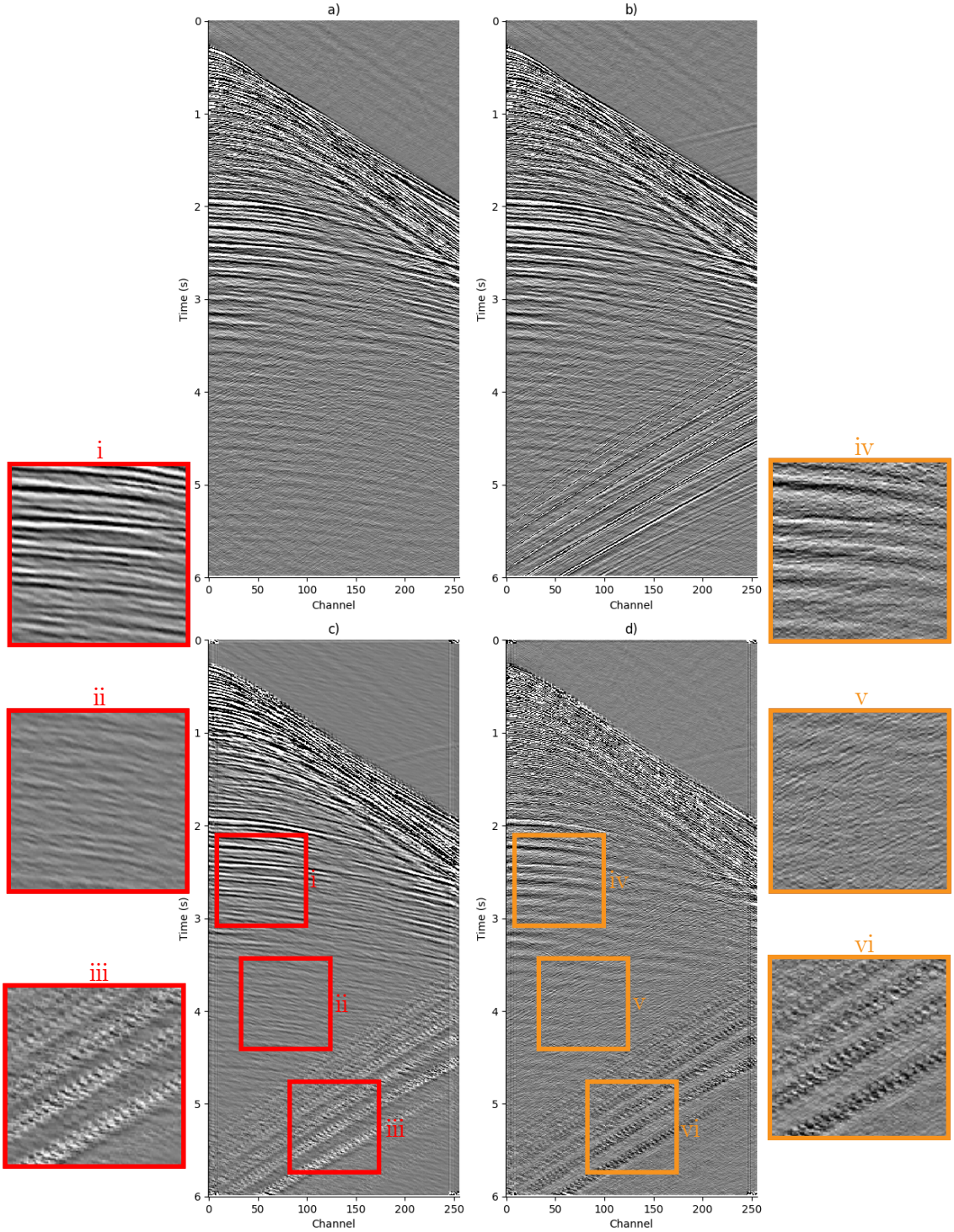


Figure 6.5: *AE1 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*

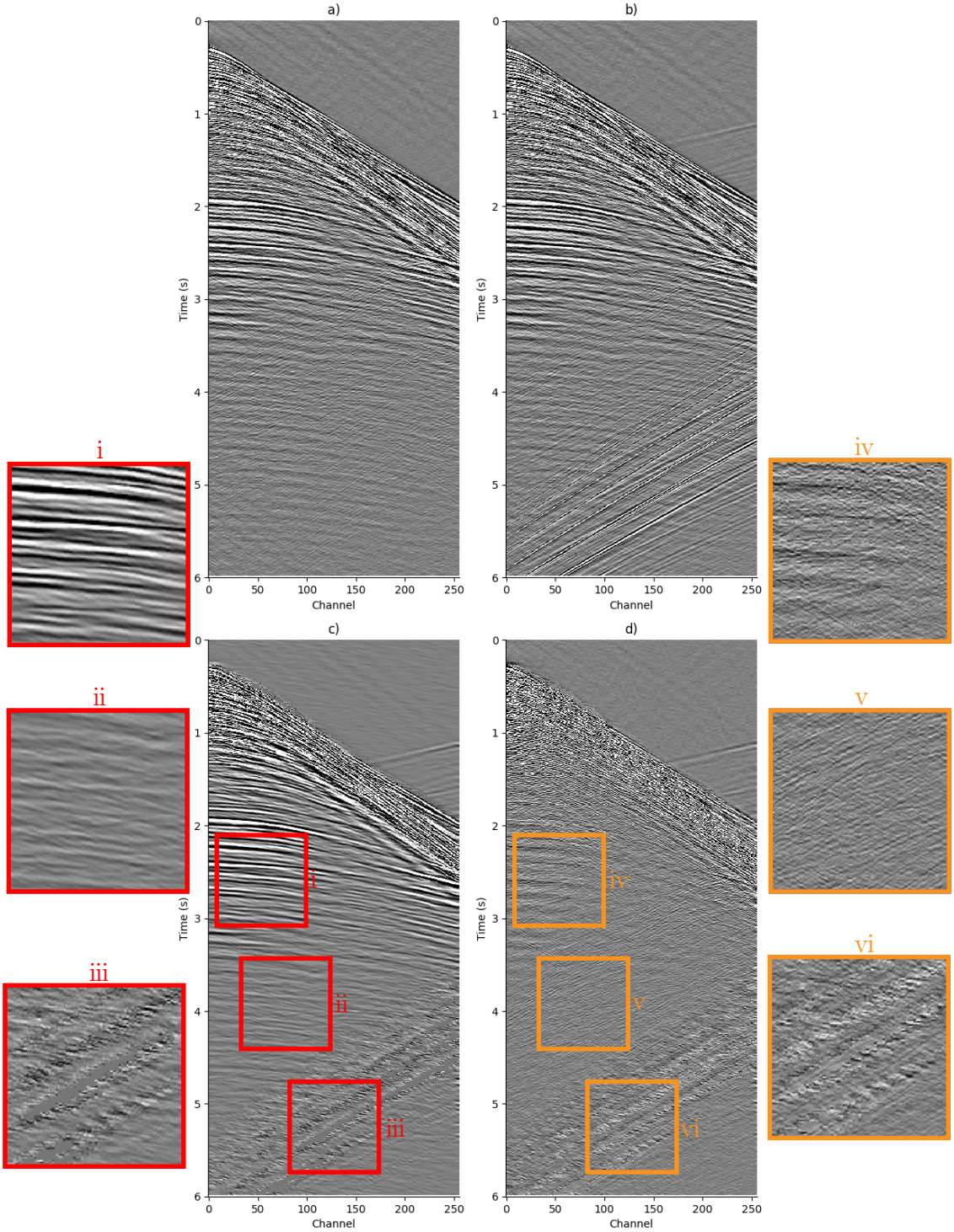
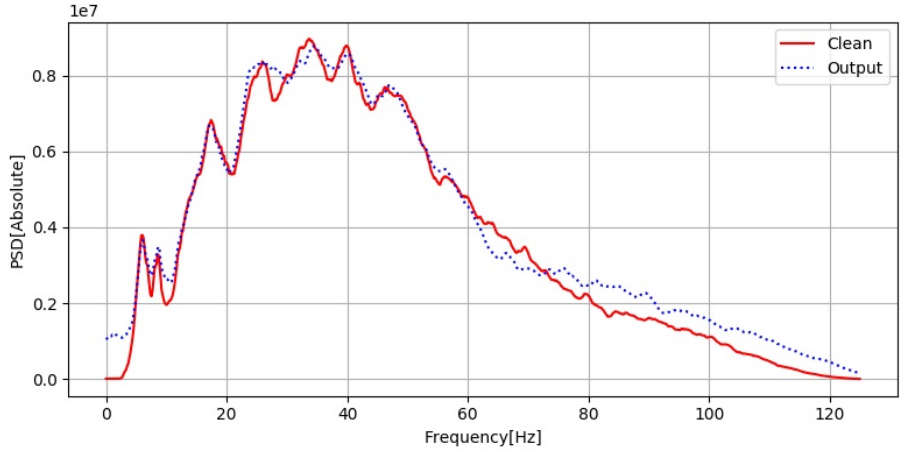
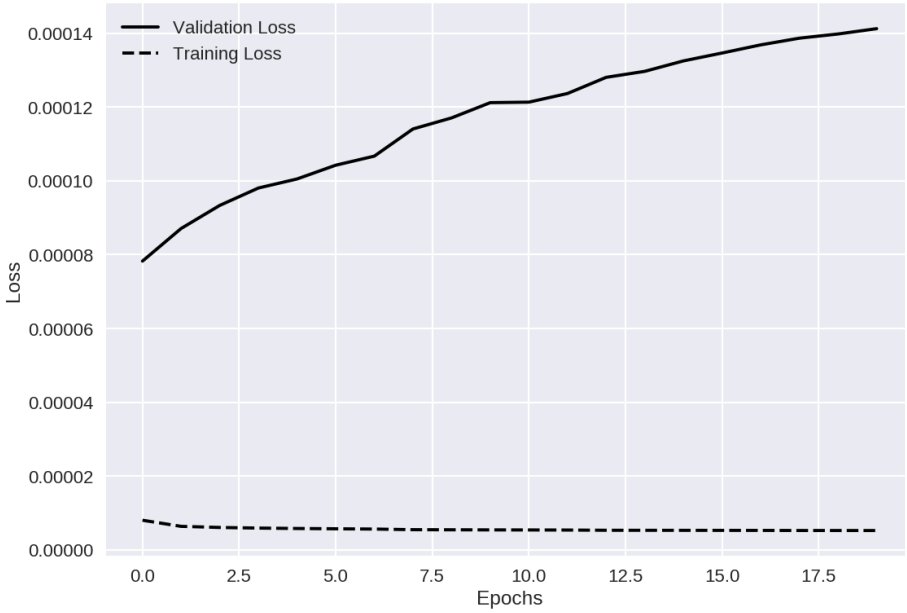


Figure 6.6: *AE2 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*



(a) Frequency spectrum of AE2



(b) Loss plot from AE2

Figure 6.7: *(a) Frequency spectrum and (b) loss plot for AE2*

6.4 No Downscaling CNN

Multiple versions of the No Downscaling CNN (NDCNN) were tested in this thesis. A few selected designs are chosen and analyzed in this section, where the structure of each network is visualized in Table 6.2. NDCNN was implemented as an attempt of counteracting the artifacts introduced by the AE models.

#	NDCNN1	NDCNN2	NDCNN3	NDCNN4
1	Conv 64x7x7 d(3x3)	Conv 4x7x7 d(1x1)	Conv 64x7x7 d(1x1)	Conv 32x7x7 d(3x3)
2	Batch Norm	Batch Norm	Batch Norm	Batch Norm
3	Leaky ReLU	Leaky ReLU	Leaky ReLU	Leaky ReLU
4	Conv 64x6x6 d(3x3)	Conv 8x6x6 d(1x1)	Conv 64x6x6 d(1x1)	Conv 42x6x6 d(3x3)
5	Batch Norm	Batch Norm	Batch Norm	Batch Norm
6	Leaky ReLU	Leaky ReLU	Leaky ReLU	Leaky ReLU
7	Residual (0 + 6)	Residual (0 + 6)	-	Residual (0 + 6)
8	Conv 64x4x4 d(3x3)	Conv 16x4x4 d(1x1)	Conv 64x4x4 d(1x1)	Conv 32x4x4 d(3x3)
9	Batch Norm	Batch Norm	Batch Norm	Batch Norm
10	Leaky ReLU	Leaky ReLU	Leaky ReLU	Leaky ReLU
11	Conv 64x3x3 d(3x3)	Conv 32x3x3 d(1x1)	Conv 1x3x3 d(1x1)	Conv 42x3x3 d(3x3)
12	Batch Norm	Batch Norm		Batch Norm
13	Leaky ReLU	Leaky ReLU		Leaky ReLU
14	Conv 64x3x3 d(2x2)	Conv 16x3x3 d(1x1)		46x3x3 d(2x2)
15	Batch Norm	Batch Norm		Batch Norm
16	Leaky ReLU	Leaky ReLU		Leaky ReLU
17	Conv 64x3x3 d(1x1)	Conv 8x3x3 d(1x1)		Conv 50x3x3 d(2x2)
18	Batch Norm	Batch Norm		Batch Norm
19	Leaky ReLU	Leaky ReLU		Leaky ReLU
20	Conv 32x3x3 d(1x1)	Conv 4x3x3 d(1x1)		Conv 16x3x3 d(1x1)
21	Batch Norm	Batch Norm		Batch Norm
22	Leaky ReLU	Leaky ReLU		Leaky ReLU
23	Conv 1x3x3 d(1x1)	Conv 1x3x3 d(1x1)		Conv 1x3x3 d(1x1)
#P	361,441	14,441	217,473	130,025

Table 6.2: *Table listing the model structure of 4 NDCNN models tested in this thesis, where d denotes the dilation rate of the filters. All models used MAE loss, Leaky ReLU with* $\alpha = 0.3$*, RMSprop optimizer and ran for 40 epochs.*

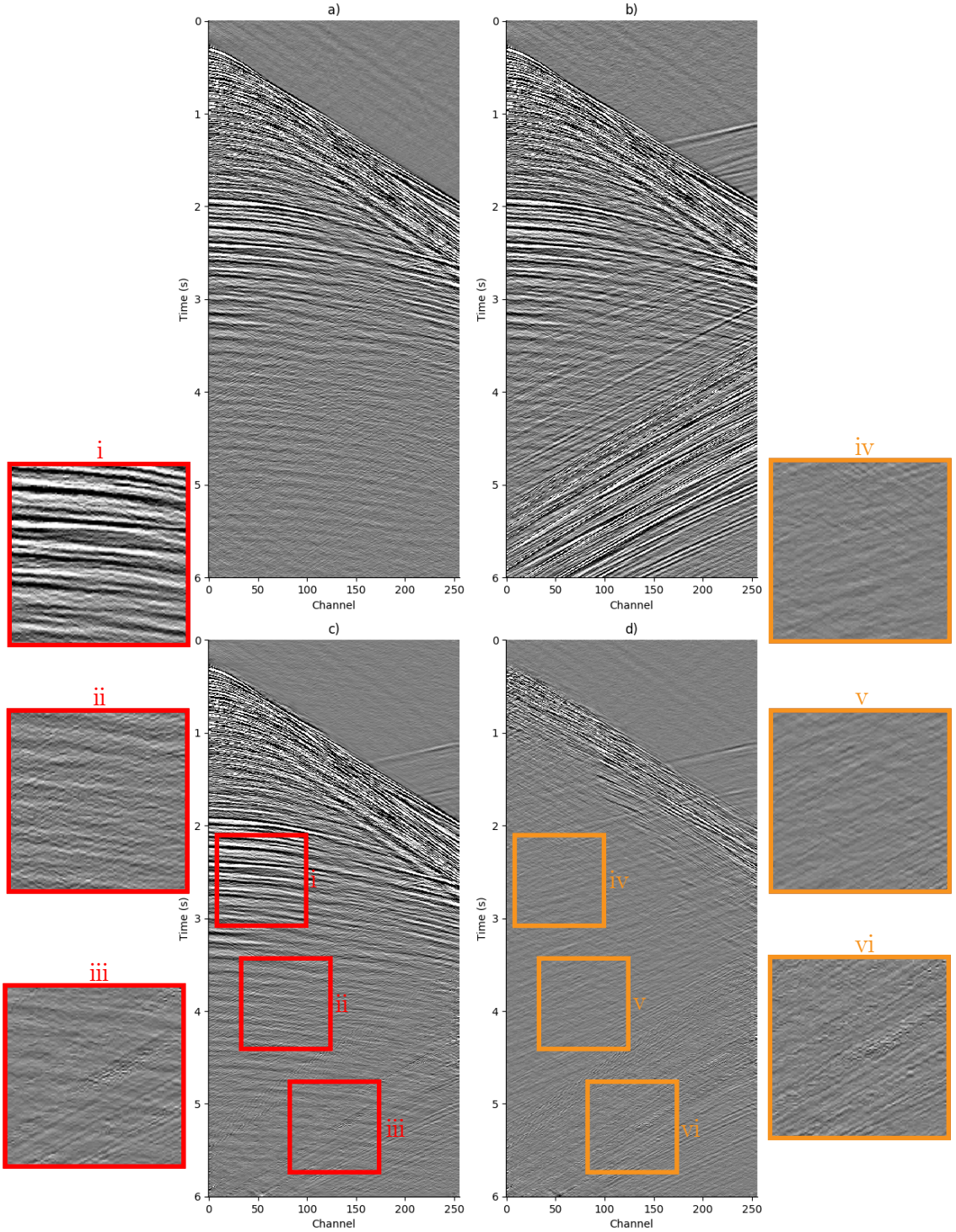
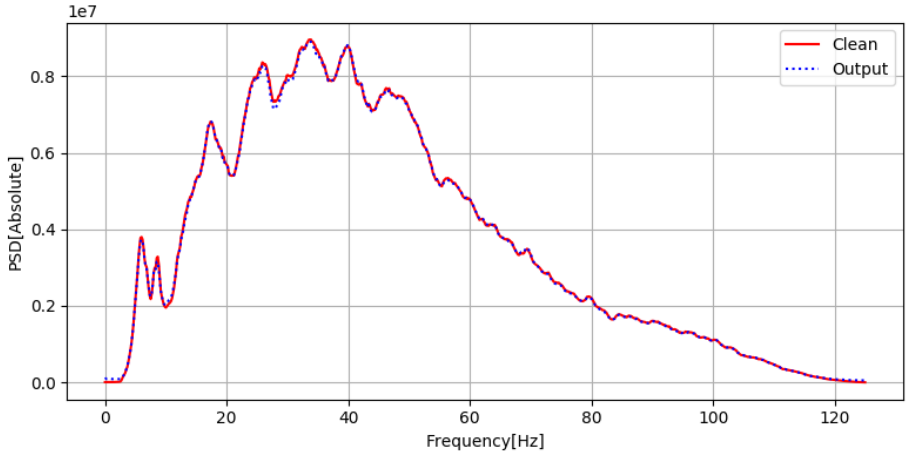
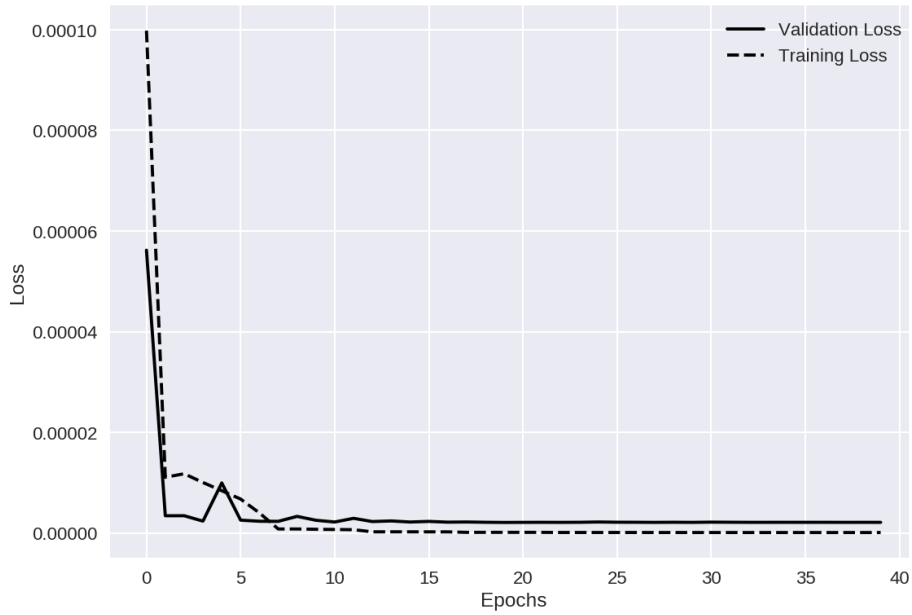


Figure 6.8: *NDCNN1 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*



(a) Frequency spectrum of NDCNN1



(b) Loss from NDCNN1

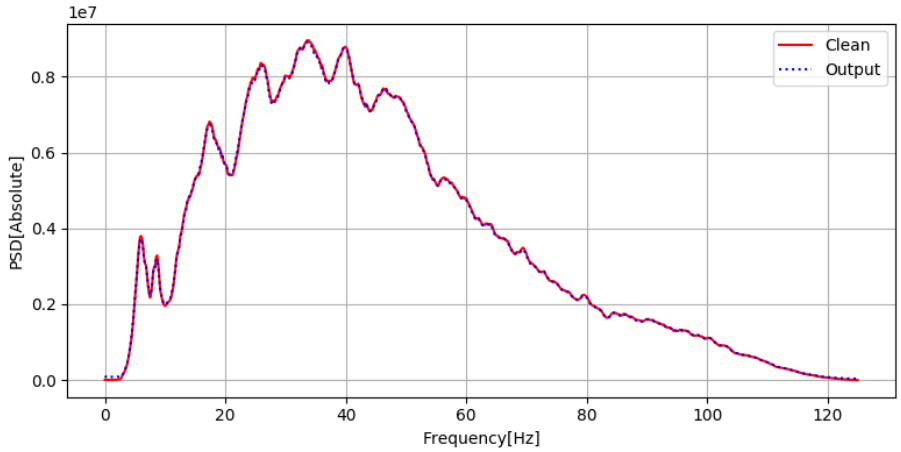
Figure 6.9: *(a) Frequency spectrum and (b) loss plot for NDCNN1*

The denoising results from NDCNN1 can be viewed in Figure 6.8. The output, Figure 6.8c, shows some SI-noise residual, but the amplitudes are weak and almost all noise is removed. The zoomed section (box iii) in Figure 6.8c highlights an area where the SNR in the input data, Figure 6.8b, is very low. There are some residual noise left from the strong events, but almost everything is efficiently removed. The underlying geology is well kept, as there are no apparent geological information in the corresponding area (box iv) in the difference, Figure 6.8d. Boxes ii and v in respectively Figures 6.8c and 6.8d highlight an area with strong SNR, but relatively

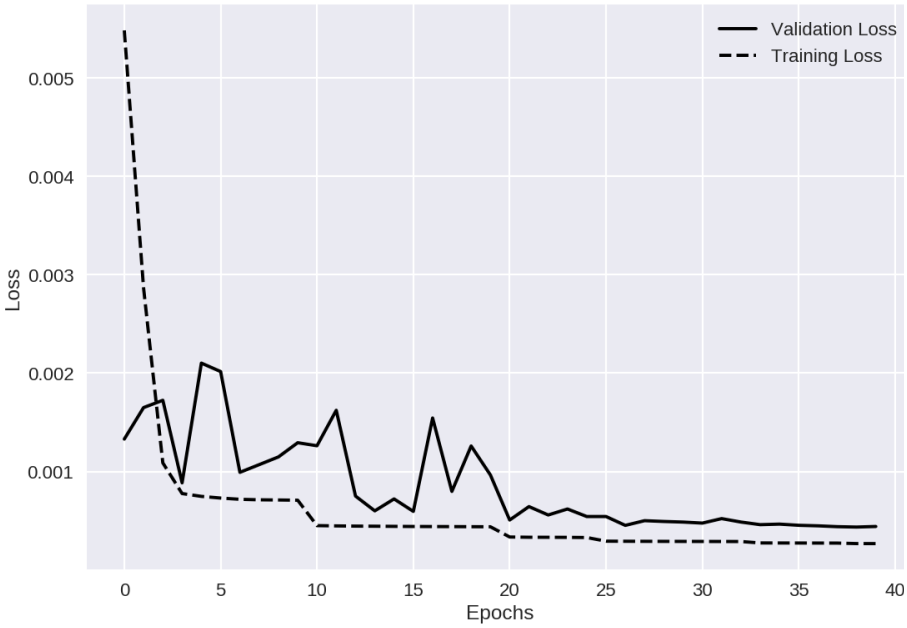
weak signal. The output shows essentially no residual noise and the reflections appear to be well reconstructed. The difference shows insignificant SI-noise residuals, but no geological data. This means that the model performs well for both low and high SNR. Box i in Figure 6.8c shows strong well reconstructed geological reflections and direct comparison with box iv in Figure 6.8d proves that no geological information is left in the difference plot in this area. The difference shows lines intercepting from the top left corner dipping towards higher offset. These lines are tugging noise which are present in the ground truth. The model had not been trained to remove this type of noise, but has learned the characteristic of the noise, thus removing it in the same process. There are substantial amounts of the refracted arrivals left in the difference, Figure 6.8d, which cause some loss in the shallow reflections around 1.5 - 2.5 second TWT. The model recreates the frequency spectrum almost perfectly, as can be seen in Figure 6.9a. The loss is presented in Figure 6.9b, where the training loss stabilizes at 10^{-6} after 7 epochs. The validation loss stabilizes slightly higher, but is still in the same order of magnitude. There are some tugging noise present throughout the ground truth. Some of this noise is removed by the network and will be registered with a higher loss since it exists in the ground truth.

Figure 6.11 visualizes the output from NDCNN2. This network model is a minimized version of NDCNN1 where the number of filters throughout the layers has been substantially reduced to test model redundancy and increase efficiency. The dilation rate has been set to 1 for all layers. NDCNN2 shares similar traits with NDCNN1 where box-plots i and iv (respectively Figures 6.11c and 6.11d) look almost identical to corresponding box plots in Figures 6.8c and 6.8d. The first set of box plots differ as there are visible SI-noise residuals left in the output, as can be seen in the difference. Deep regions, box-plot iii in Figure 6.11c, with low SNR are reconstructed poorer than in NDCNN1, box-plot iii in Figure 6.8c. There are substantial amounts of SI-noise left in the output, suppressing some geological data and creating a washout-effect, as can be seen in box-plot vi in Figure 6.11d. The frequency spectrum, Figure 6.10a is virtually perfectly recreated which also yields for NDCNN1, Figure 6.9a. The loss plot, given in Figure 6.10b, shows an oscillating validation loss and a higher convergence value than shown in Figure 6.9b. Both the training and validation loss converge to a value of approximately $9 \cdot 10^{-4}$ which is orders of magnitudes higher than NDCNN 1. NDCNN2 reconstructs the refracted arrivals noticeably better than NDCNN1, and is likely due to the reduction

in dilation rate in the filters. Although the refracted arrivals are better recreated, the shallow geological reflections around 1.5 - 2 second TWT are still impacted, as shown in the difference, Figure 6.11c.



(a) Frequency spectrum of NDCNN2



(b) Loss from NDCNN2

Figure 6.10: *(a) Frequency spectrum and (b) loss plot for NDCNN2*

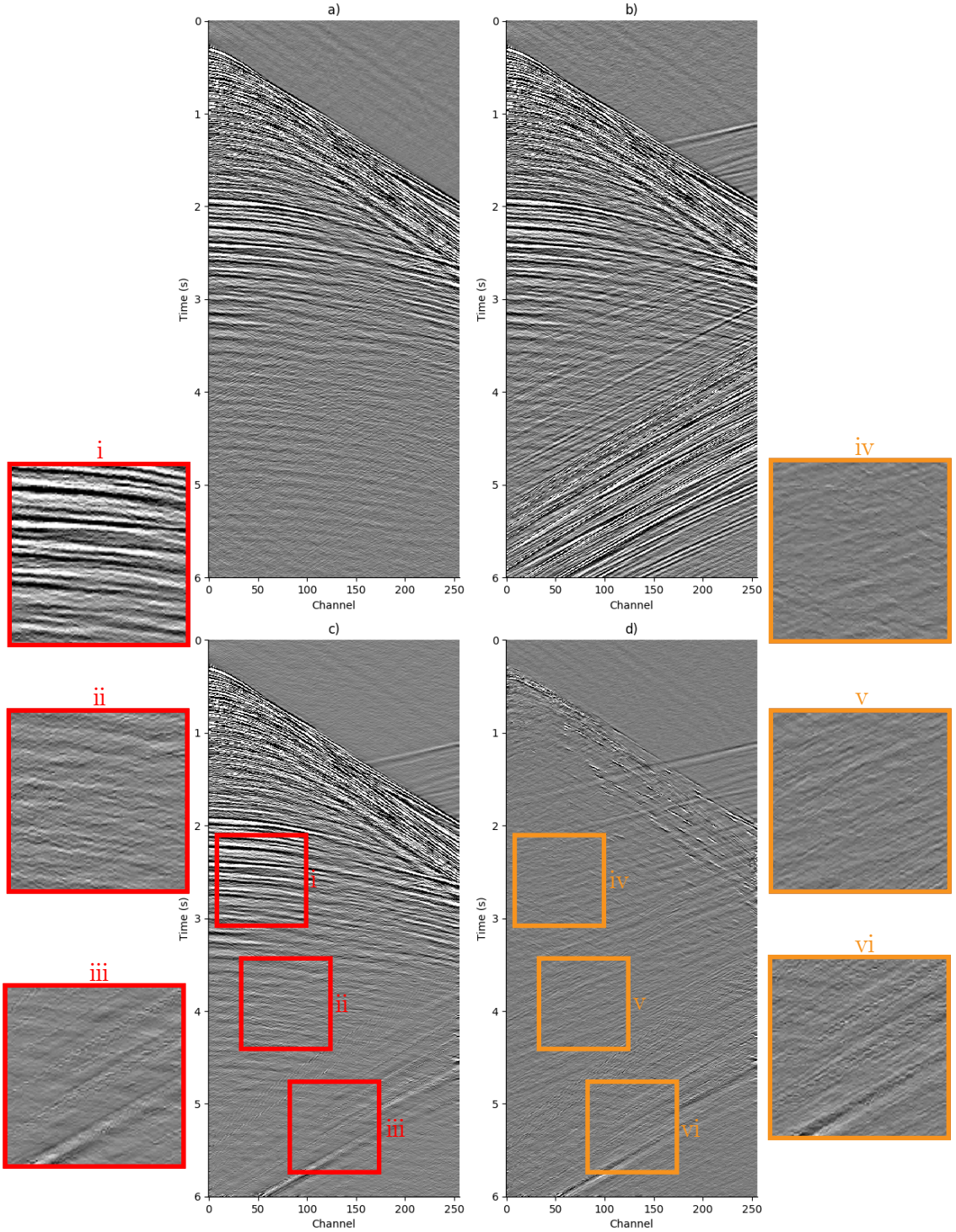


Figure 6.11: *NDCNN2 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*

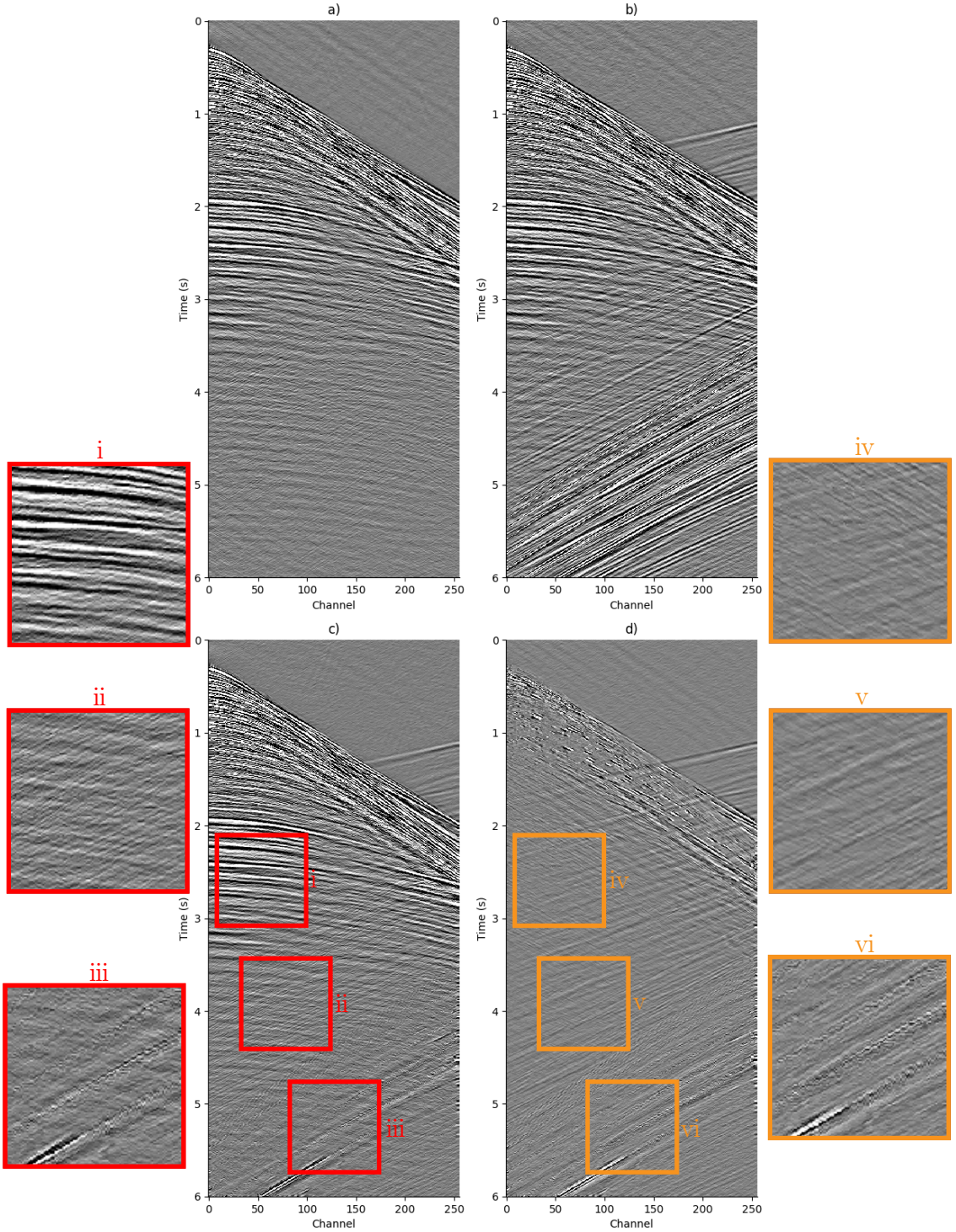
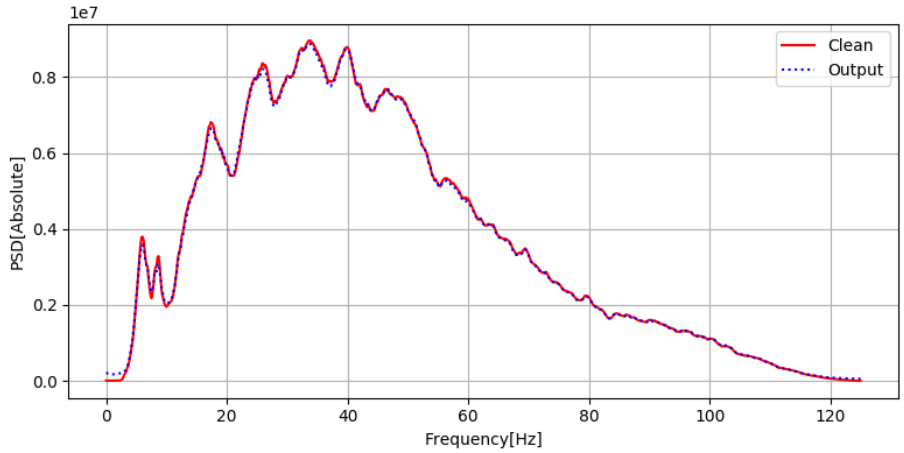
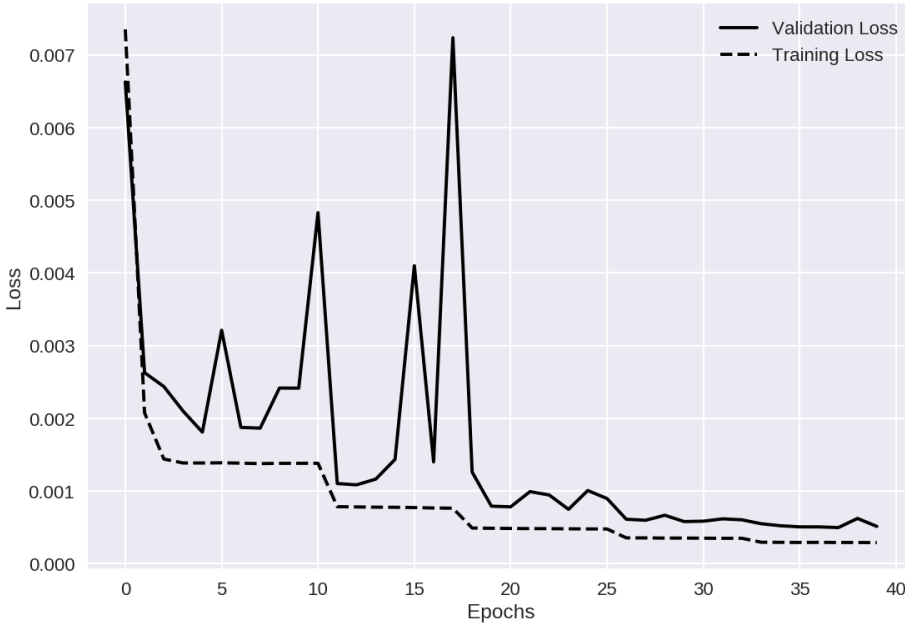


Figure 6.12: *NDCNN3 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*



(a) *Frequency spectrum of NDCNN3*



(b) *Loss from NDCNN3*

Figure 6.13: *(a) Frequency spectrum and (b) loss plot for NDCNN3*

The performance of NDCNN3 can be viewed in Figure 6.12. NDCNN3 is a compressed version of NDCNN1 where the same amount of filters has been kept, but both dilation and multiple layers have been removed. The main trend is that the output, Figure 6.12c, shows more residual SI-noise than NDCNN1, Figure 6.8c. There are apparent SI-noise residuals left in the deeper parts of the output, box-plot iii in Figure 6.12c, creating a washout-effect in the data. This characteristic is shared with NDCNN2, box-plot iii in Figure 6.11c, but is less prominent. The residual SI-noise is more distributed over the entire output, Figure 6.12c, compared

to NDCNN1, Figure 6.8c, and NDCNN2, Figure 6.11c. This is apparent in the difference where the residual noise in the deeper parts, box-plot vi in Figure 6.12d, show rather strong residual noise amplitudes. The underlying geology appear to be intact as there are no visible geology in the difference. The strong reflections visible in box-plot i in Figure 6.12c are recreated without loss, as can be seen in box-plot iv in Figure 6.12d. Although not important, it does remove more tugging noise than the two other models. The frequency spectrum, visible in Figure 6.13a, shows an almost perfectly reconstructed frequency spectrum, similar to those from NDCNN1 and NDCNN2, Figures 6.9a and 6.10a. The loss plot from NDCNN 3 in Figure 6.13b shows that both training and validation loss converge towards the same value as NDCNN 2, Figure 6.10b, namely $9 \cdot 10^{-4}$. The validation loss differs much from both NDCNN 1 and 2 where it oscillates significantly during the epochs with a substantial spike around 17 epochs.

NDCNN4 is an attempted optimized version of NDCNN1. It shares the same characteristichs as NDCNN1 but has a reduced number of filters in most layers. The output from NDCNN4 is visualized in Figure 6.14c. It is almost identical to the output from NDCNN1, Figure 6.8c, as suspected. It has a bit more residual SI-noise left in the image, but shows less washout-effects than NDCNN2 and NDCNN3, Figures 6.11 and 6.12. Box-plots i, ii (Figures 6.14c) and iv and v (Figure 6.14d) show no apparent difference with the corresponding subplots in NDCNN1 Figure 6.8. The only difference between the two network models is connected the strong SI-noise at 4 - 6 second TWT. Comparing box-plot v from both Figures 6.14d and 6.8d shows the similar residual noise spikes, but with varying amplitude. This is more apparent in Figures 6.14 and 6.8, showing a slightly higher amplitude in the SI-noise residual in the difference from NDCNN4. The frequency spectrum of NDCNN4, Figure 6.15a is similar to all the other spectra. The loss plot from NDCNN4, Figure 6.15b, shows a similar curve shape as NDCNN1, Figure 6.9b, but both validation and training loss in NDCNN4 are one order of magnitude higher than NDCNN1. The loss differs from all the other models in that the validation and training loss converge towards the exact same value compared to the other models where the validation loss always converges toward a slightly higher number. These results are further discussed in the next chapter.

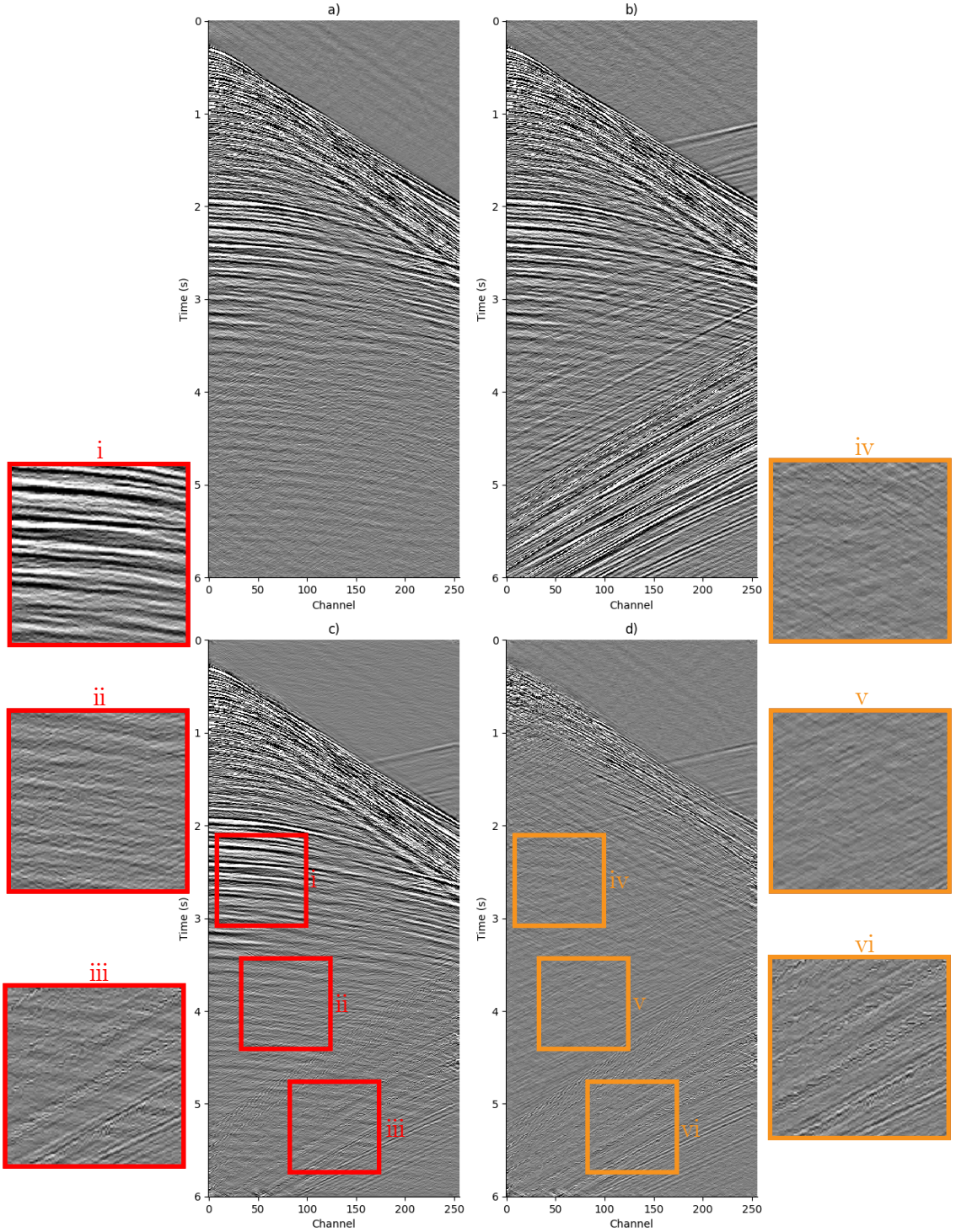
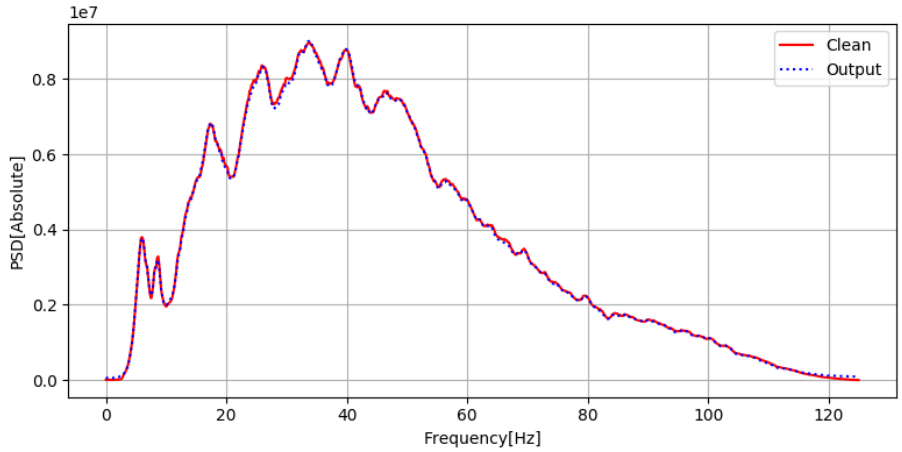
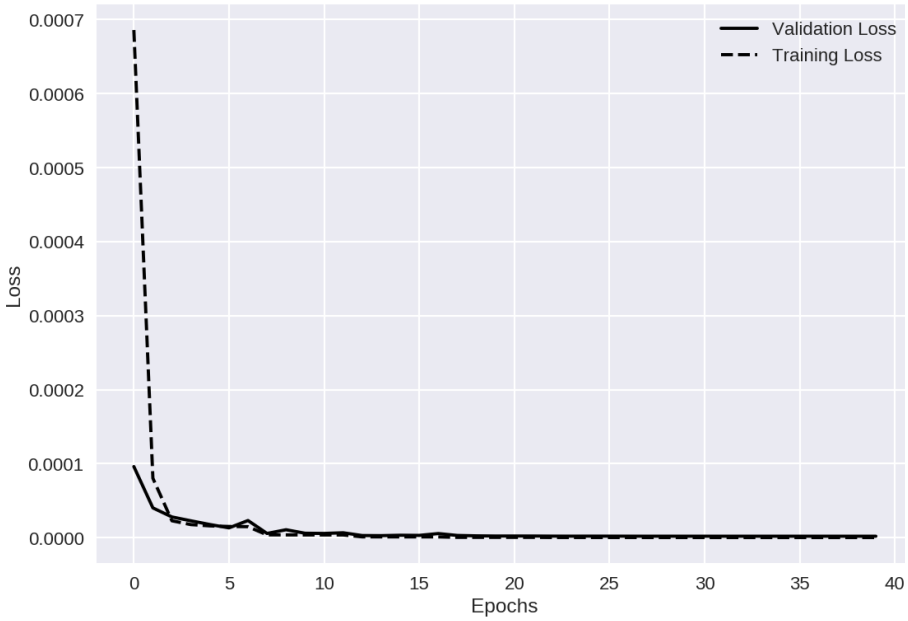


Figure 6.14: *NDCNN4 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*



(a) Frequency spectrum of NDCNN4



(b) Loss from NDCNN4

Figure 6.15: *(a) Frequency spectrum and (b) loss plot for NDCNN4*

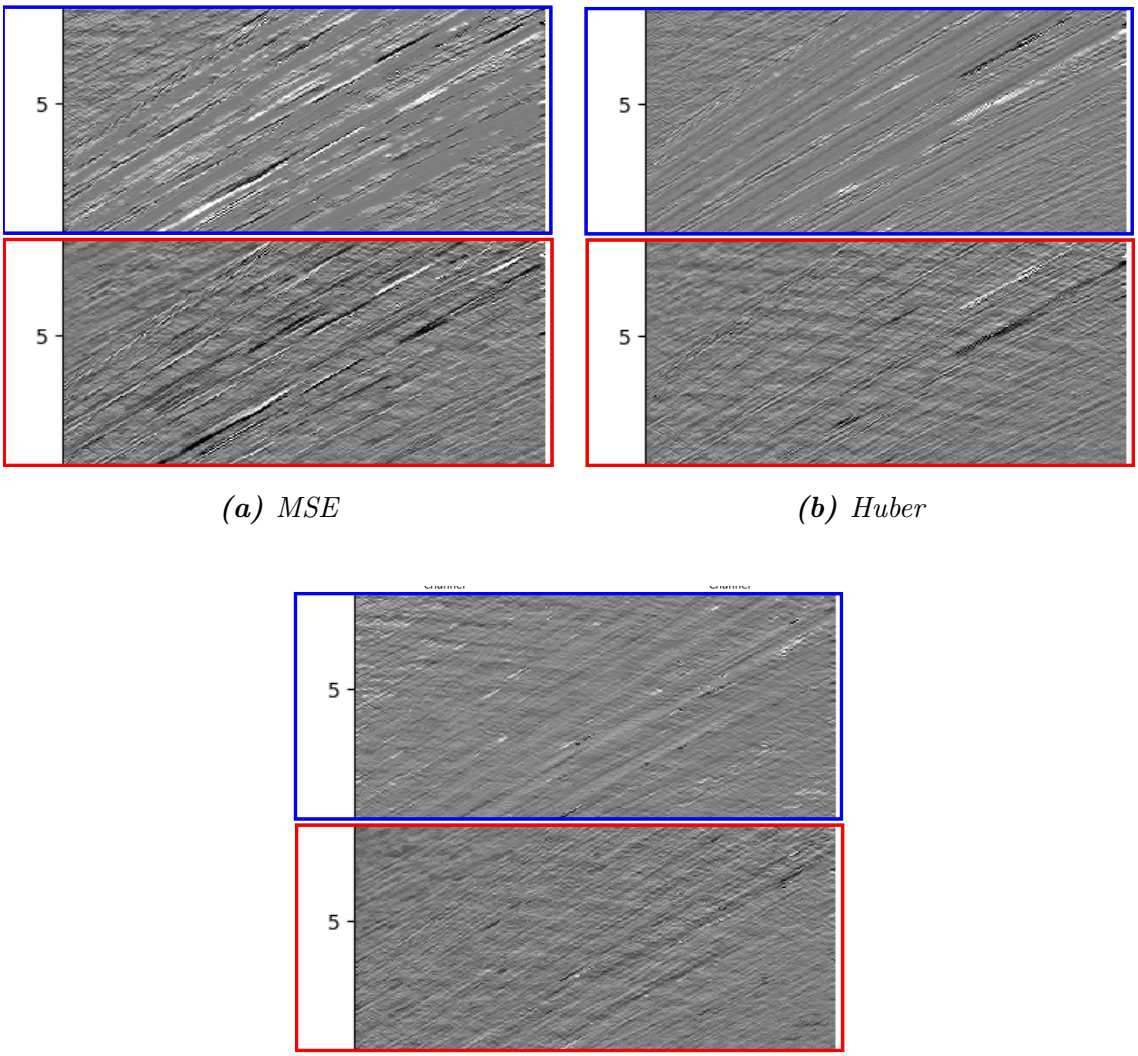
6.4.1 Loss function

The loss function in machine learning is a key part of the network. Four different loss functions have been mentioned in this thesis, and three of them have been tested to investigate their difference in performance. These three functions are: MSE (Figures 6.16a and A.5), Huber (Figures 6.16b and A.6) and MAE (Figures 6.16c and A.7). The fourth one mentioned earlier in this thesis is Binary cross-entropy which has been omitted since it is restricted to binary problems only (see section

3.2.3). In Figure 6.16 the output is marked blue, and the difference between input and output is marked with red. The network model might be sub-optimal, but this was purposely done to see how the loss functions performed in such conditions.

By comparison of Figures 6.16a, 6.16b and 6.16c it is evident that MAE perform best overall. The zoomed sections in the figures show a notable difference in the SI-noise removal in deep regions, where both MSE and Huber suffer from washout and underlying geology loss. MAE has substantial amounts of SI-noise left in the output, but less washout than the other functions with a better recreation of the underlying geology. Comparing the general performance of all functions, Figures A.5, A.6 and A.7 shows a significant difference in recreations of main geological events. There are more residual geology left in both MSE and Huber, 2-3.5 second TWT, compared to MAE. MSE performs slightly better than Huber, but still sup-par compared to MAE.

MAE, Figure A.7, recreates the shallow events, such as water bottom reflection and refracted waves, better than both MSE and Huber, Figures A.5 and A.6. It shows geological spikes, which are not present in the two other models, but with relative low amplitude. MAE has a slightly different overall amplitude in the output, compared to MSE and Huber. Although this might cause concern due to artifacts in the data, MAE still perform best and is therefore the desirable loss function. The results will be discussed more in detail in the next chapter.



(a) MSE *(b) Huber*

(c) MAE

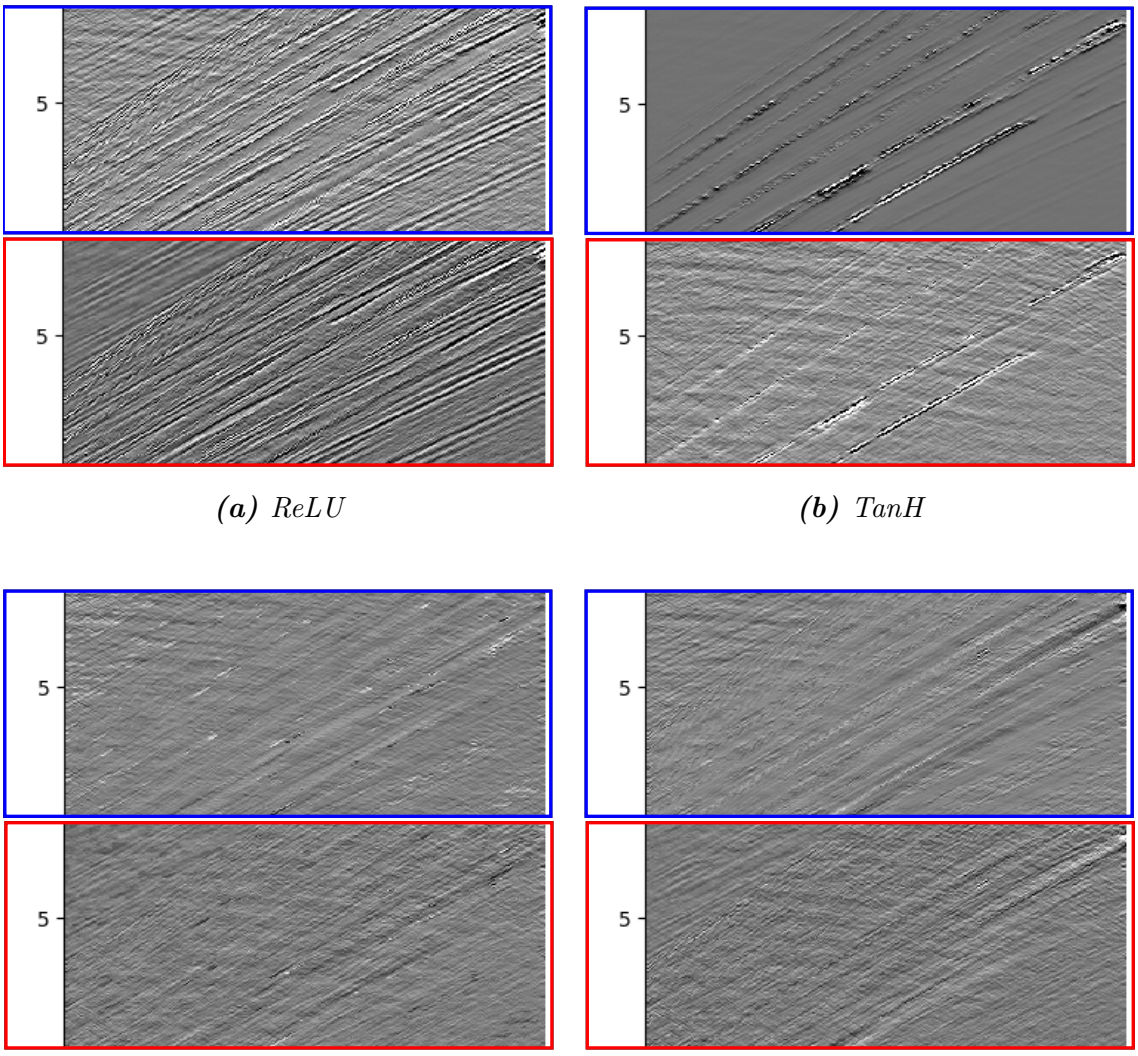
Figure 6.16: *Figure visualizing the impact of the four different loss functions used in similar models. Figures (a), (b), and (c) respectively visualize MSE, Huber and MAE. The blue and red boxes visualize output and difference (ground truth - output). The figures are cutouts from larger sections found in the Appendix A (Figures A.5 - A.7).*

6.4.2 Activation function

The activation function in Machine learning introduces non-linearity to the network, as mentioned in Chapter 3. Four different activation functions were tested employing MAE loss to investigate their performance. These four are: ReLU (Figures 6.17a and A.8), TanH (Figures 6.17b and A.9), Leaky ReLU with $\alpha = 0.01$ (Figures 6.17c and A.10) and Leaky ReLU with $\alpha = 0.3$ (Figures 6.17d and A.11). In Figure 6.17

the output is marked blue, and the difference between input and output marked with red.

Comparing Figures 6.17a, 6.17b, 6.17c and 6.17d show an apparent poor performance of TanH compared to the three other functions. It is evident that the function could not reproduce the image and completely failed. The entire output, Figure A.9, shows almost no geology and the only reconstructed parts are the refracted arrivals and parts of the noise. ReLU, Leaky ReLU with $\alpha = 0.01$ and Leaky ReLU with $\alpha = 0.3$, Figures 6.17a, 6.17c and 6.17d show more resemblance. ReLU has substantially more residual noise left in the output, and more geology in the difference, Figure A.8. The residual noise overshadows the underlying geology causing a bad recreation of the deeper events. Both version of Leaky ReLU perform better than the latter functions, where $\alpha = 0.01$ cause a patchy reconstruction, with substantial geological information left in the difference, Figure A.10. Leaky ReLU with $\alpha = 0.03$ has a better overall amplitude with better reconstruction of deep geology, Figure A.11. It might have slightly more geological residual at 2.5 - 4 second TWT left in the difference, but recreates the amplitude well and performs better overall. These results will be further discussed in Chapter 7.



(a) *ReLU* ***(b)*** *TanH*

(c) *Leaky ReLU* $\alpha = 0.01$ ***(d)*** *Leaky ReLU* $\alpha = 0.3$

Figure 6.17: *Figure visualizing the impact of four different activation functions used in a model. Figures (a), (b), (c) and (d) respectively visualize ReLU, TanH, Leaky ReLU with* $\alpha = 0.01$ *and Leaky ReLU with* $\alpha = 0.3$*. The blue and red boxes visualize output and difference (ground truth - output). The figures are cutouts from larger sections found in the Appendix A (Figures A.8 - A.11).*

6.4.3 Feature maps

As already mentioned, NDCNN4 is an attempt to design an optimized version of NDCNN1. This optimization is based on a systematic analysis of the feature maps output from a given layer, where empty or duplicate filters were removed to increase efficiency. Figure 6.18 shows such output from layer 3 in NDCNN1. It is apparent that both duplicate and blank filters exist. Multiple filters capturing the same

features will likely not give an improvement to the model and might in the worst case scenario affect the model negatively. Figure 6.19 shows the feature maps from layer 3 in NDCNN4. The number of filters has been reduced by a factor of 2 which removes some of the duplicates visible in figure 6.18. There are certain features which NDCNN4 does not seem to capture in this layer and there are other features which seems to be introduced. The overall output from layer 3 in NDCNN4 seems to show that there are fewer duplicates and no empty filters.

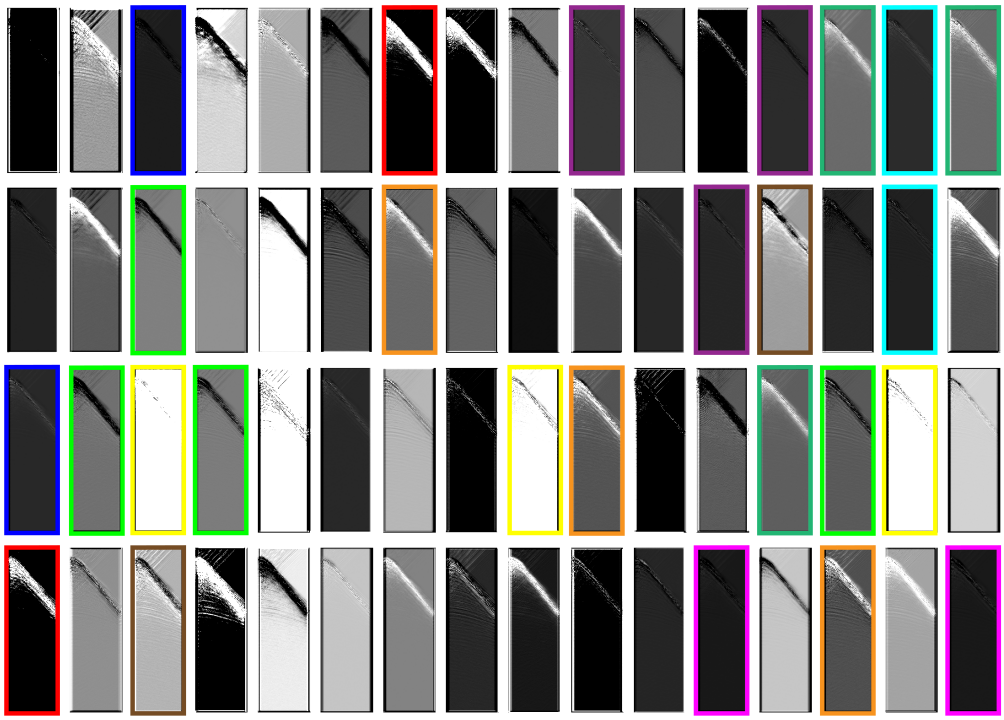


Figure 6.18: *Feature maps from NDCNN1 layer 3. Examples of duplicate (or empty) filters are marked with borders of similar color, grouping them together. This layer consists of 64 filters, explaining the high amount of feature maps.*

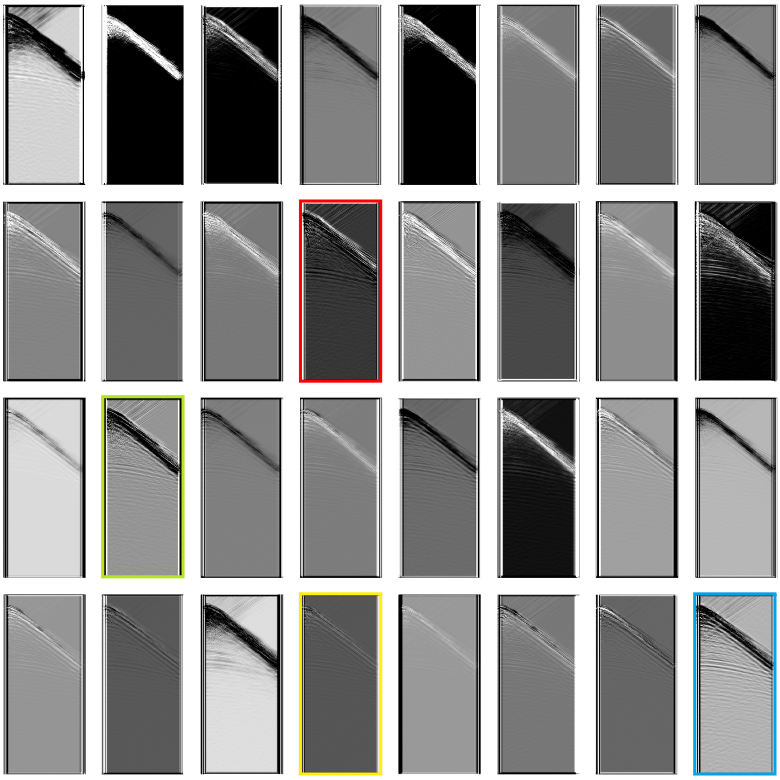
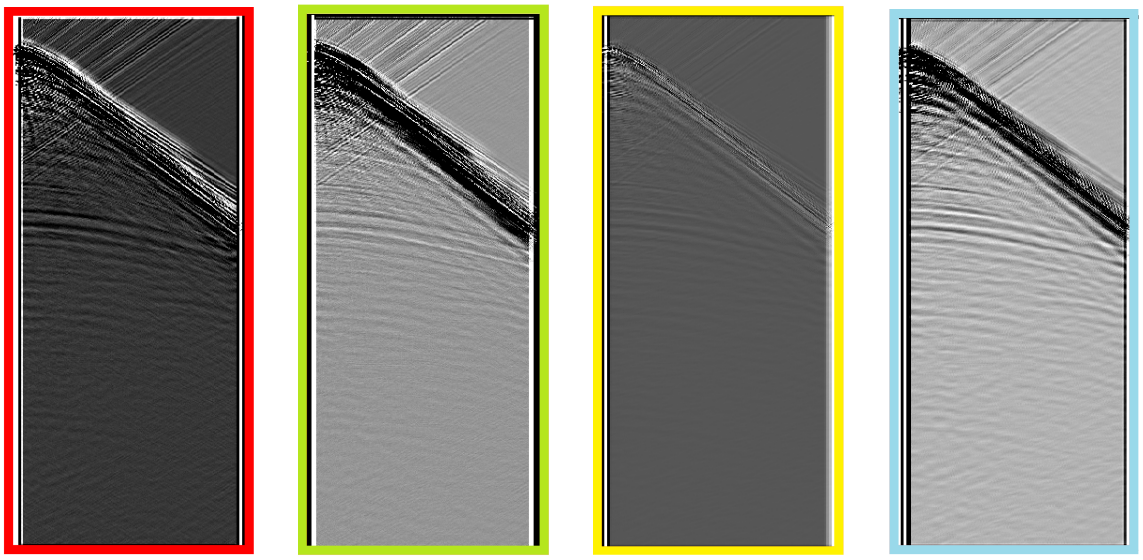


Figure 6.19: *Feature maps from NDCNN4 layer 3. Four feature maps are highlighted and visualized below.*



(a) ***(b)*** ***(c)*** ***(d)***

Figure 6.20: *Figure highlighting a section of the feature maps viewed in Figure 6.19 where the figures are color coded with the same colors.*

Although there are fewer duplicates, most features seems to be kept intact in the filter reduction. There is a similar variance in features in the feature maps from NDCNN1, Figure 6.18, than there are for NDCNN4, Figure 6.19. The feature maps from NDCNN4 also show multiple duplicates. Figure 6.20 shows a selection of 4 feature maps from the layer. Figures 6.20a - c are different and thus fetching different features from the data. However, Figure 6.20d seems very similar to Figure 6.20b, demonstrating that there still are duplicates in this layer. The performance of NDCNN4 was quite similar to NDCNN1, but with slightly poorer performance at large traveltimes. This is likely due to variance loss in the feature maps for certain layers, giving the network a more narrow feature representation and it therefore recognizes less features. Another negative observation from both, Figures 6.18 and 6.19, is that the network seem to focus too much on the water bottom reflection and refracted events. They have a substantially higher amplitude and seem to draw too much attention from the network, leaving the geology and noise less in focus.

6.5 U-NET

Two versions of U-NET were tested in this thesis and the structure of these networks are visualized in Table 6.3. U-NET was implemented as a way of combining AE and NDCNN, utilizing features from both network structures to create a robust and efficient model.

#	U-NET1	U-NET2
1	Conv 16x6x6	Conv 16x3x3
2	Leaky ReLU	Leaky ReLU
3	Max Pool	Max Pool
4	Conv 32x6x6	Conv 32x3x3
5	Leaky ReLU	Leaky ReLU
6	Max Pool	Max Pool
7	Conv 32x4x4	Conv 32x3x3
8	Leaky ReLU	Leaky ReLU
9	Conv 32x3x3	Conv 32x3x3
10	Leaky ReLU	Leaky ReLU
11	Up Sampling	Up Sampling
12	Residual (11 + 5)	Residual (11 + 5)
13	Conv 16x3x3	Conv 16x3x3
14	Leaky ReLU	Leaky ReLU
15	Up Sampling	Up Sampling
16	Residual (15 + 2)	Residual (15 + 2)
17	Conv 8x3x3	Conv 8x3x3
18	Leaky ReLU	Leaky ReLU
19	Conv 1x3x3	Conv 1x3x3
#P	50,577	29,153

Table 6.3: *Table listing the structure of the U-NET models used in this thesis: U-NET1 and U-NET2. Both models used Leaky ReLU with $\alpha = 0.3$, MAE loss and RMSprop optimizer. Both models had dilation rates of 1 in all layers.*

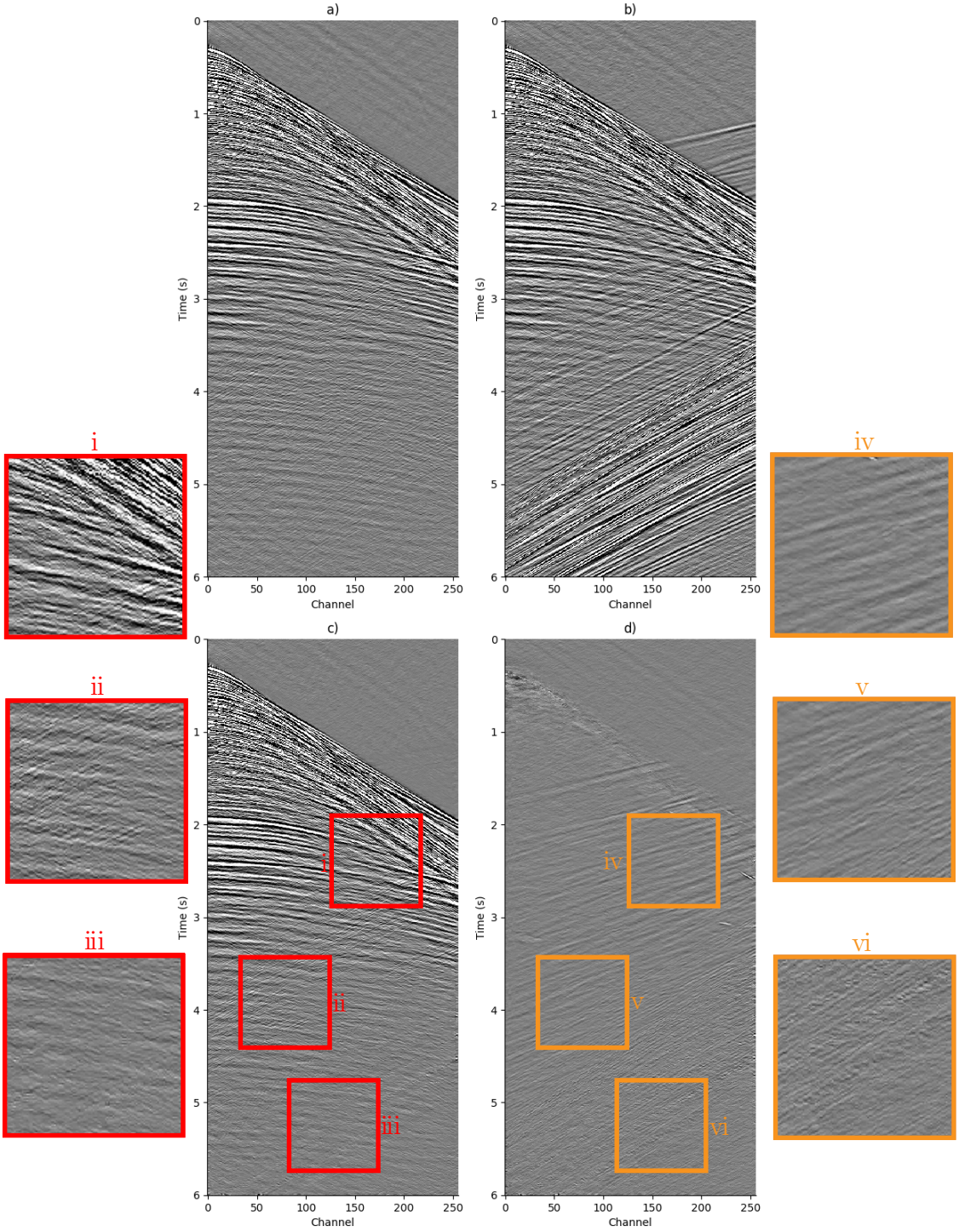
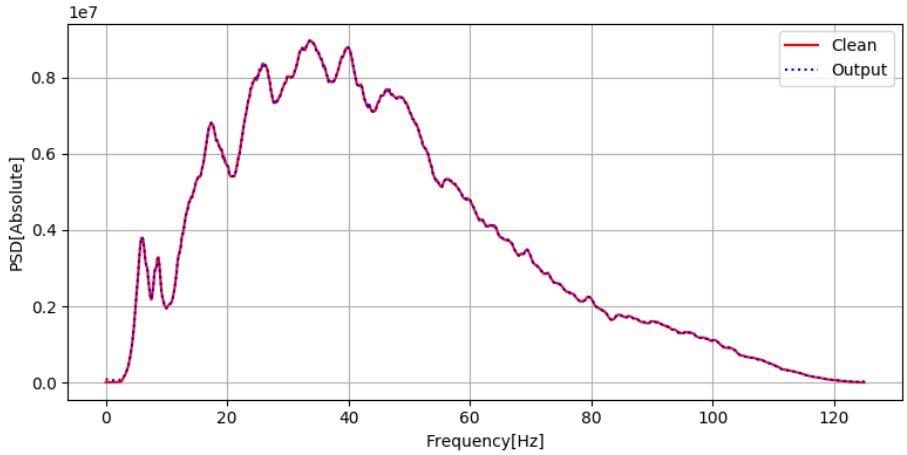
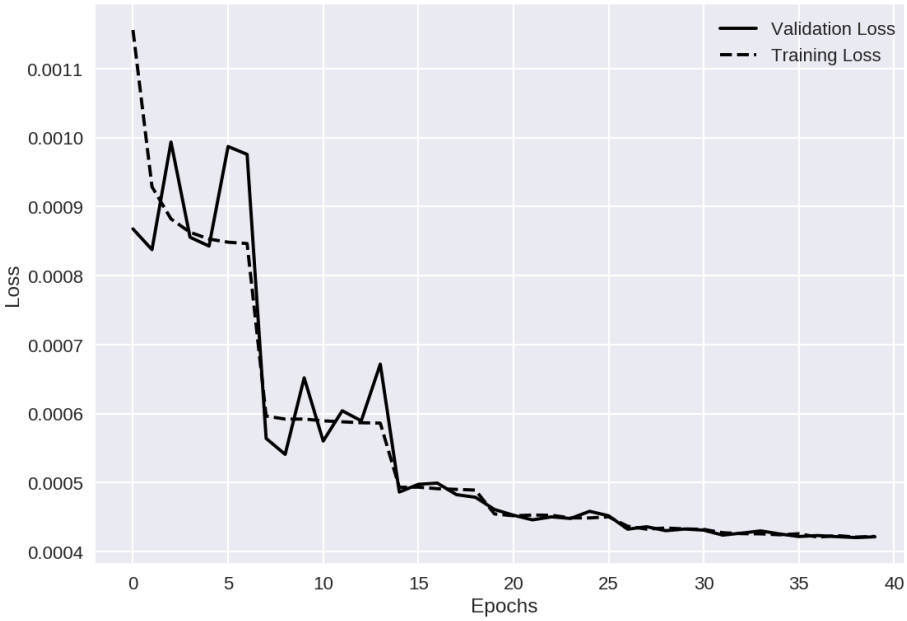


Figure 6.21: *UNET-1 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*



(a) Frequency spectrum of U-NET1



(b) Loss from U-NET1

Figure 6.22: *(a) Frequency spectrum and (b) loss plot for U-NET1*

The denoising results from U-NET1 can be viewed in Figure 6.21. The output, Figure 6.21c, shows some residual SI-noise, but it has weak amplitudes and with almost everything being attenuated. Box-plot iii in Figure 6.21c highlights a section where the high amplitude SI-noise blends with the input data. Almost no residuals are left, as can be further illustrated in box-plot vi in Figure 6.21d containing the noise removed. The underlying geology is well recreated as there is essentially no geological information left in the difference. Comparing sections with less noise, i.e. box-plot ii in Figure 6.21c and box-plot v in Figure 6.21d, show low-amplitude

residual noise, an virtually no geological loss. The network model does a good job at recreating the shallow events, i.e. box-plot i in Figure 6.21c and box-plot iv in Figure 6.21d, where almost no geological information is left in the difference. The frequency spectrum of U-NET1, 6.22a, is well reconstructed with apparently no deviation at any frequency. The loss plot, Figure 6.22b shows a step-wise training loss which stabilizes at 10^{-4} after approximately 25 - 30 epochs. The validation loss is oscillating much until 15 epochs, but stabilizes at the same value as the training loss.

Results from U-NET are plotted on Figure 6.23. The output, Figure 6.23c, shows some weak residual SI-noise. Direct comparison of box-plots iii and iv in Figures 6.23c and d and 6.21c and d shows more SI-noise residuals in output from U-NET2, but with low amplitude. Both models recreates the underlying geology well, as there are essentially no geology visible in the difference. Box-plots ii and v from Figures 6.23c and 6.23d show low-amplitude noise residuals, with apparent difference from the corresponding boxes in Figures 6.21c and 6.21d. There is a small difference in in the recreation of shallow events between U-NET1 and U-NET2. U-NET2 shows small spikes of geology, box-plots i and iv in Figures 6.23c and 6.23d, which are not present in corresponding box-plots in Figures 6.21c and 6.21d. These spikes are minor and sparse, and there is little significant difference in the performance of the models. The frequency spectra of U-NET2, 6.24a, is well recreated without apparent loss and show significant resemblance with the spectrum of U-NET1 6.22a. The loss plot, Figure 6.24b shows the same trend as U-NET1, but with a significant larger saddle point at $8 \cdot 10^{-4}$ before approaching similar values. The model appears to not have converged fully, and might improve slightly with more training.

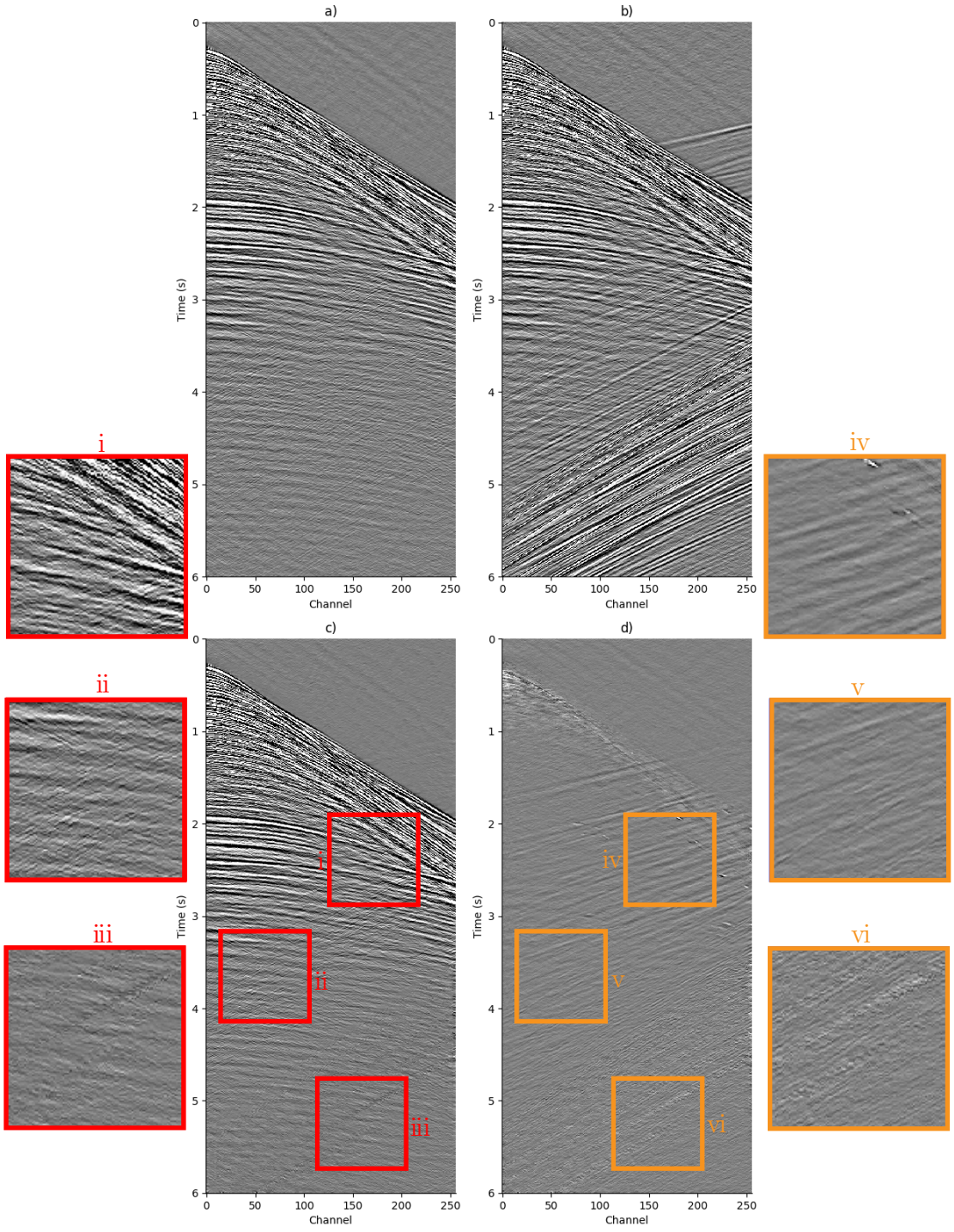
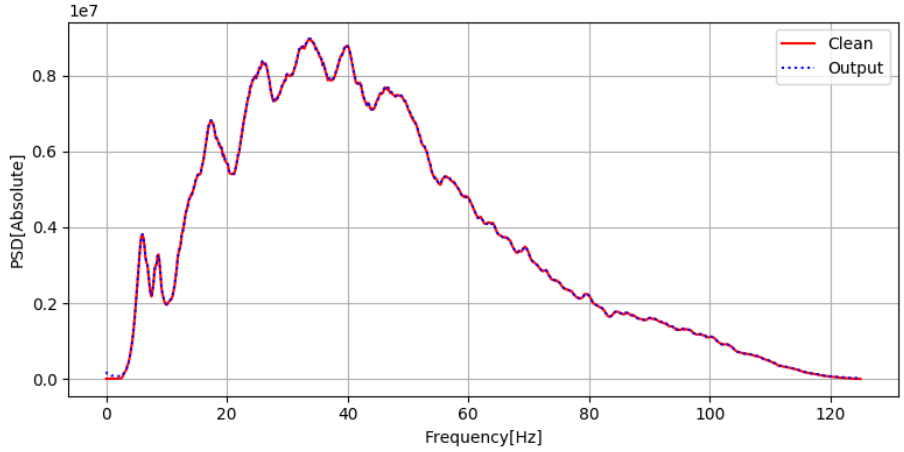
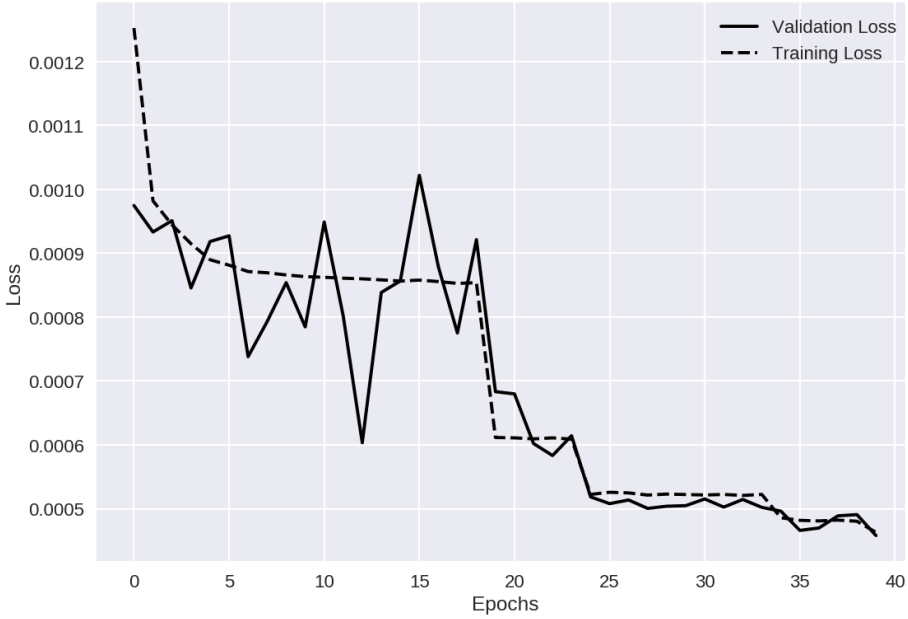


Figure 6.23: *U-NET2 where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c). Figures i,ii, ... ,vi visualize zoomed sections of interest. The red sections mark locations in the output c), while the orange sections mark corresponding areas in the difference d).*



(a) Frequency spectrum of U-NET2



(b) Loss from U-NET2

Figure 6.24: *(a) Frequency spectrum and (b) loss plot for U-NET2*

6.6 Execution time

The execution times of the 4 NDCNN models and 2 U-NET models compared are listed in Table 6.4. The most striking difference in this table is the training time required for the different NDCNN models compared to the U-NET models. While two of the NDCNN models need more than 100 hours, the U-NET models require only 7-8 hours training time. Another clearly visible result is how training time increases with number of filters. NDCNN2 has the same amount of layers as ND-

CNN1, but still has a lower training time than NDCNN3 which has less than half as many layers. The number of filters in NDCNN1 and NDCNN3 are high, while the number of filters in NDCNN 2 is greatly reduced. This is directly connected to the number of parameters in a model, Table 6.2. The more parameters in a model, the higher the computational demand. The optimized verison, NDCNN4, required 33% less training only due to filter reduction.

There is quite a large difference in number of parameters between the two U-NET models, yet there is little difference in training time. This is likely due to the training time reaching optimal conditions and being bottlenecked by reading and writing to files. All network models yielded efficient results for real time denoising, once training was completed. The most demaning model used 0.16s to denoise 1 shot, while the most efficient model required only 0.02s per shot. A key note from this section is that the U-NETS ran significantly faster than all NDCNN models, even though the NDCNN2 has less parameters than both the U-NETS.

	NDCNN1	NDCNN2	NDCNN3	NDCNN4	U-NET1	U-NET2
Train	157:58:19	35:44:28	53:06:37	101:40:03	7:26:26	7:3:28
R.T.	0.16s	0.03s	0.05s	0.1s	0.02s	0.02s

Table 6.4: *Execution time for the different models on a marine seismic shot gather with 1500 time samples and 256 traces per shot where R.T. denotes real time denoising. The training times for the networks are given on the format HH:MM:SS and Real time denoising is measured in seconds.*

7 | Discussion

7.1 Overview of the results

In this section the results will be reviewed and discussed in more depth. The challenges faced in this thesis can be divided into two aspects. The first one is to perform quality denoising of marine seismic data, and the second one is to not damage the data in the process. Four different architectures were tested, namely: Classification CNN, Denoising Autoencoder, NDCNN and UNET. The Classification CNN does not perform denoising and was implemented to prove whether a neural network could distinguish between noise contaminated data and clean data. The model reached almost perfect results after a few epochs, which indicates that the network can separate clean and noisy data with high certainty, which was later demonstrated in the denoising results.

7.1.1 AE

It is evident that the downscaling process in the AE models affect the seismic signal negatively by removing important information. The difference image in both AE models show considerably high-frequency residual while this is much less prominent in the NDCNN and UNET models. Downscaling in neural networks, as explained in Chapter 3, is a way of removing information which appears as incoherent between the different input images. The fact remains that compression removes information and in the case of the AE models, appear to have removed too much information, leaving the outputs looking slightly synthetic. The pixelated artifacts in the intercepting SI-noise as well as the refracted events are also likely a result of the compression, where the high frequencies are poorly recreated. There is less pixelation in AE2 than AE1 which might be due to the extra layer added at latent level in AE2. The extra layer makes the model focus more on data which lacks high frequencies, thus omitting more content from the direct arrival, causing the output to look slightly more synthetic. This is likely due to the lower frequencies being more consistent between the shots, while the high frequency content vary more. The noise is poorly denoised, as there are much residuals left in both AE models.

7.1.2 NDCNN

The NDCNN models perform significantly better than the AE models. The output from each NDCNN model, Figures 6.8c, 6.11c, 6.12c and 6.14c, look rather clean with mostly minor noise artifacts present. There is a clear difference in the noise attenuation between the four models where NDCNN1 and NDCNN4 attenuate most of the noise, only leaving small residuals, and NDCNN2 and NDCNN3 show higher amplitudes and more residuals. The most coherent difference between these network models is the dilation rates, where NDCNN1 and NDCNN4 have increased dilation, while NDCNN2 and NDCNN3 have a constant dilation of 1 throughout the networks. NDCNN1 is the computationally most demanding model, but also yields the best results. All network network models perform well, but struggle with the shallow events, e.g. refracted events and water bottom reflection. The high frequency content and amplitudes of these events probably make it hard for the network to recreate such features and an ideal input dataset would have had these events muted.

There is an apparent difference between the NDCNN models with respect to the shallow reflections and the refracted events. The network models with sparse filters, a dilation rate higher than one, show a poorer reconstruction of these events. This can be seen where NDCNN1 and NDCNN4, Figures 6.8 and 6.14, have more residuals from the refracted events, causing a poorer recreation of the shallow reflections compared to NDCNN2, Figure 6.11. NDCNN3, Figure 6.12, performs poorly, even though no dilation is present, but this might be explained by the reduction in number of layers. It is likely that the network models struggle due to the high frequency content in these events. The highest frequency in the shot gather is $125Hz$ representing Nyquist. If the sparsity in the higher diluted filters becomes larger, the filter will most likely not be able to recreate the higher frequencies. The first layer in NDCNN1 and NDCNN4 has a 3x3 dilation rate. This means there are two "empty" pixels between each sample mapped by the filters, making the filter sample every 4 pixels. This implies that it might struggle to recover the higher frequencies, explaining why there is much residuals left of the refracted and shallow events in these models.

The frequency spectra of all NDCNN models, Figures 6.9a, 6.10a, 6.12 and 6.14, show that each model perform well for the whole band. This implies that even if some high-frequency content is lost in the very shallow parts, the overall impact is

negligible due to small amplitudes in the difference images. The main information in the difference plots for the NDCNN, Figures 6.8 and 6.12, are carried by noise, proving that the network models do recreate most parts, even in the shallow regions.

7.1.3 U-NET

It is evident that the U-NET models performed best of all network models considered in this thesis, Figures 6.21 and 6.23. They recreated the entire seismic image with good precision, even at the high frequency shallow events where the other models were sub-optimal. Residual noise is present in both models, but with low amplitudes and little to no washout effects present. U-NET2, Figure 6.23d, shows more residual SI-noise left compared to U-NET1, Figure 6.21d, but it still performs better than all the NDCNN network models. It seems that the combination of downscaling, while adding the output from previous layers create a robust model for working with seismic. This is likely due to an increased resolution gained in the combination of upscaling and inheritance from previous layers (Ronneberger et al., 2015). Furthermore, the U-NET models are much more efficient as they only require approximately 10% of the training time needed for the NDCNN network models.

7.1.4 NDCNN1 vs U-NET1

As U-NET1 performed best, and NDCNN1 performed best of the NDCNN models, these two networks have been compared in the case of different types of SI-noise. The results are included in the Appendix, Section A.4, visualising three different types of SI. The U-NET appear to be more robust than the NDCNN, as can be seen from Figures A.12 and A.13. Both network models struggle with SI-noise with similar moveout as the refracted events. Both models leave residual noise in case the noise crosses the direct arrivals, but removes a descent amount of the noise intercepting at larger arrival times. NDCNN1 left higher amplitude noise and struggled more with the refracted events, while the U-NET showed slightly more washout while removing more noise. Both network models fail in the case of strong SI-noise from the side, Figures A.14 and A.15. SI-noise intercepting from astern is removed well by both models, Figures A.16 and A.17. The kinematic of the SI-noise is substantially different from the geological data in this case, making the noise easier to remove. The U-NET models were trained with a more optimal generator function, as there

were a slight issue with the function used to train the NDCNN models. This means that the U-NET is trained with a higher variance of data, compared to the NDCNN models. The NDCNN could therefore potentially give slightly better results, but the difference would likely be small.

7.1.5 U-NET1 vs Industry Standard denoising

Figure A.18 shows a noise contaminated stack from a different area. The denoising result of U-NET1 on this stack can be seen in Figure A.19 with the difference shown in Figure A.21. The U-NET removes a significant amount of noise, while keeping almost all geology intact. There are patches with residual noise present in the time range 800-1200s and 1700-2200s, but the remainder of the stack is well denoised. These ranges contain respectively strong SI-noise from ahead and from the side. The difference, Figure A.21, shows low amplitude geology removed in the denoising process. Figure A.20 visualizes the denoising result employing an industry standard procedure. It is evident that the industry standard denoising performs better than the U-NET. The difference for the industry standard, Figure A.22, shows no apparent geology and more noise removed than from the U-NET.

An important aspect to mention in the comparison industry denoising and the U-NET is the differences in data handling of the two. The U-NET is reading one shot at a time and denoising in real time. The industry standard is using a multi-shot approach to break the coherency of the data (Shen et al., 2019), thus giving the industry denoising an advantage over the neural network. This could in principle be done for the network as well, but will not be optimal as the network is restricted to small batch sizes.

7.2 The different parameters of the models

When building a neural network model, there are numerous hyper parameters which have to be fine tuned obtain the optimal results.

7.2.1 Filter size

The filter size greatly increases the total number of parameters and thus has a direct impact on the execution time. Comparing NDCNN2, Figure 6.11, and NDCNN3,

Figure 6.12, show that these two network models mainly differ in the recreation of the shallow events. They share the same filter sizes in the first layers, but NDCNN3 does not have any smaller filters and rely entirely on features extracted from larger filters. This might explain differences observed in ouput data, but there is also major differences in the number of layers between the models. One noticeable difference between all the network models is that NDCNN2 and NDCNN3 show slightly more washout and high-amplitude spikes where the high amplitude SI-noise intercepted, while NDCNN1 and NDCNN4 attenuate more of the noise. This is likely caused by the dilation rate, as both NDCNN2 and NDCNN3 only has dilation rates of 1 throughout the networks.

U-NET1, Figure 6.21, and U-NET2, Figure 6.23, only differ in filter sizes, where U-NET1 has larger filter sizes in the first layers, while U-NET2 has small filters throughout the network, Table 6.3. There is close to no visible difference between the U-NET models. The only visible performance difference in the models are with respect to the shallow events and slightly higher amplitudes of the residual noise. U-NET1 removes slightly more noise, box-plot iii in Figure 6.21c, while U-NET2 shows small noise spikes, box-plot iii in Figure 6.23c, not present in U-NET1. It is likely that the compression in the U-NET reduces the impact of the filter change, given the small difference between the two network models. It might also be the case that the filter size has less impact on the data than initially assumed, thus favouring smaller filters due to lower run time.

7.2.2 Number of filters

The filters in a neural network capture characteristics of the data, where more filters per layer will capture a broader variation. One key finding in this thesis is that the number of filters did impact the data, but did not impact it as much as first suspected.

NDCNN1 and NDCNN2 are two network models with a significant difference in the number of filters. Figure 6.11c shows a relatively good recreation of the geological data. There is almost no geological information lost in the image, as can be seen in Figure 6.11d, except for the refracted waves which every model struggled to recreate. The main difference between these two network models relates to noise attenuation. Comparing Figure 6.11d with Figure 6.8d shows considerably more noise residuals left in the output from NDCNN2. It appears as if the model

captures the characteristics of the noise: the noise has been attenuated, but not removed. The output from NDCNN1 clearly shows less noise residuals left in the output although far from perfect. NDCNN2 shows washout zones where the main SI-noise intercepts. NDCNN2 shows loss of underlying signals in this area, while NDCNN1 performs much better. It is evident that the number of filters directly correlates with the denoising performance.

Comparing NDCNN1 and NDCNN4 strengthens this assumption. These two network models are almost identical, only differing in the change of number of filters per layer. NDCNN4 was designed in an attempt to optimize NDCNN1 without any performance loss. This was almost accomplished, but more noise residuals are left in the output from NDCNN4, Figure 6.14c, compared to NDCNN1, Figure 6.8c. Since the only difference between these models are the number of filters, the reduction in filter size has to be the cause for the extra residual noise. Adding more filters will likely reduce the noise, but this has to be done in a controlled manner. If too many filters exist per layer, model update will slow down. There is a limitation in the amount of features a network can identify from an image, thus introducing too many filters will only lead to overlapping features and non-active filters. If too many filters are introduced, this might impact the output negatively as certain features become weighted more in favour of others. As can be seen from Table 6.4, the more filters a network model has, the higher run time and the more memory needed. A deep model with many filters therefore requires much more processing power and requires longer training time before it converges. This is why NDCNN4 was an attempted optimization of NDCNN1. It increased computational efficiency by approximately 30%, but suffers from more residual noise.

7.2.3 Number of Layers

Establishing the necessary number of layers in machine learning is not an exact science. There are certain rules of thumbs stating that a linearly separable problem does not require a hidden layer. A more pragmatic approach is to simply assume that more layers means more complexity in the data. The more layers a network contains, the more complexity is introduced. Theoretically, a network can map close to any map-able function containing three layers, and the function is not necessarily map-able if any more layers are needed (Tamura and Tateishi, 1997). Marine seismic data consists of a large amount of samples per shotgather and in

this thesis a size of $1500x256$ was used, resulting in $3.84 \cdot 10^6$ samples per shot. It might not theoretically be possible to perfectly map a denoising function, but it does not have to be lossless. As long as the loss is kept relatively low without significant residuals present in the data, the result is acceptable. Introducing more layers might therefore prove reasonable when working with marine seismic data, as the data might appear relatively complex for the computer.

Model NDCNN3 is a compressed version of NDCNN1. It consists of three convololutional layers, but keeps the same number of filters as NDCNN1, see Table 6.2. Comparing the results from the use of these two network models demonstrate that a certain number of layers are needed for a network model to yield optimal results. NDCNN3, Figure 6.12, was impacted less by the layer reduction than assumed, but still performs notably worse than NDCNN1, Figure 6.8. More washout zones exist where the noise intercepted data, although the underlying geology appear to be well recovered. It seems likely that the shallow events (with more high frequency features) require more layers for the network to recreate, as they introduce complex and overlapping structures. The remainder of the data is less complex and more easily recreated. It has to be mentioned that this only applies for linear data noise, as SI-noise appear more curved when originating from within close proximity to the towing vessel.

7.2.4 Activation Functions

Activation functions are introduced to the network to create non-linearity and thus solving problems that are governed by a higher-order equation. The three different activation functions tested were: TanH, ReLU and Leaky-ReLU. It is evident that TanH failed and gave poor results. This is likely due to vanishing gradients as TanH is known to have this problem, see section 3.2.2. Once the value from TanH approach -1 or 1, the gradients are so small that it might stop the network from learning properly. ReLU has the opposite problem as it may cause exploding gradients. It has no boundaries and the gradients can become increasingly large. ReLU performed well in areas where no noise was present, but struggled once the high amplitude SI-noise intercepted. As ReLU is 0 for $x < 0$, the gradients can approach 0 and stop the model from enhancing. It seems likely that either or both of these special cases of ReLU caused the poor performance. If the amplitude values of the noise were negative, the error was lower than zero. If the amplitudes were positive, it might

have caused exploding gradients.

Leaky ReLU adjusts ReLU for the zero-gradient problem, adding a slant for negative values of ReLU. Two different versions of α were tested, 0.01 and 0.3. The results suggests that $\alpha = 0.3$ performs best, as the output, Figure A.11, is attenuated more with less artifacts. Although seismic data may seem complex, they contain a high degree of linearity (superimposed linear wavefields). Increasing the slope of Leaky ReLU pushes the activation function towards becoming linear. As this yields better results, it is likely that less complexity and more linearity in the model makes the model perform better.

7.2.5 Loss

The loss function is an important aspect of neural networks. Each time a batch is passed through the network, the loss is calculated and the parameters are updated accordingly. If the loss function is sub-optimal, the model might break down. The three loss models tested were MSE, Figure A.5, Huber, Figure A.6, and MAE, Figure A.7. It is evident that Huber performed worst of the three functions. MSE yielded notably better results, while MAE performed best. MSE is sensitive to outliers given the quadratic operation in the loss function, Equation 3.10. The shallow events, such as water bottom reflection and refracted events, have amplitudes order of magnitude higher than the deeper geological reflections. The SI-noise used in this thesis was abnormally strong, therefore outliers were created, especially in the deeper regions. The performance of the loss function, Figure A.5, supports this observation especially for the intercepting SI-noise crossing the deep reflections at 5 - 6 second TWT. The entire area is washed out and with no underlying geology reconstructed.

The MAE loss function performs significantly better. Both the shallow events and the noisy parts are better reconstructed as well as the geological information in the entire image, figure A.7. This is likely due to the fact that MAE weights differences equally, as can be seen from Equation 3.11. Intercepting high amplitude SI-noise at late arrival times (5-6 second TWT) will not be weighted differently, than SI-noise crossing the shallow events. The noisy parts are better attenuated than those of the MSE, A.5, which is likely due to this characteristic. Although the MSE is sensitive to outliers, the shape of the function yields lower gradients when the loss approaches zero in contrast to MAE which has constant gradients for all

errors. A constant gradient when the loss is approaching zero causes oscillations in the loss as the optimization might shift parameters too much in the direction of the gradient. The Huber loss is a combination of both MSE and MAE, combining the strengths of both functions, Equation 3.12.

The Huber loss, though it is supposed to be an optimized loss function, performed poorly and in certain areas worse than MSE, Figure A.6. It performed a better denoising than MSE, but at a cost of poor recreation of the geological reflections throughout the image. Huber should in theory work better then both MSE and MAE, but this was not the case. Since Huber relies on a user defined parameter, δ, the only reasonable explanation of this poor performance is a poorly chosen value. Higher and lower values will only shift the model towards either MAE or MSE, thus the weak performance is of unknown origin.

Neural networks are typically rated for the loss they produce. If there are no visible way of determining the result of the model, the entire performance is based on loss. MAE gave higher loss values than MSE, but performed better. The loss for all NDCNN models converged after 10-20 epochs and did not improve after that, although the network models had to train for a lot longer to gain optimal results. The U-NET behaved differently, as it did not converge before 30-35 epochs. This loss behaviour fits better with the actual performance of the networks, and might be cause by the large images in NDCNN. If substantial parts of a large image are relatively well recreated, these loss functions will give a rather good result, even though the remaining parts are poorly recreated. This is because of the number of samples in the image. The U-NET performed better since it compresses the image. Error in a pixel in a smaller image will have a larger impact, thus explaining why the loss performance was different. The performance test of the loss functions could have been more thorough, testing more cases for more networks, but was omitted due to the high demand of running multiple models. Instead, all functions was tested on the same network model where loss was the only varying factor.

7.3 Model restrictions

The model restrictions in this thesis will vary depending on which network model is in focus. Since U-NET1 gave the overall best results and NDCNN1 gave the best results of the NDCNN models, these will be the two network the models considered

in this section.

7.3.1 Data

The data tends to be a restriction of every network model, as the entire job of a neural network is to process data. Each shot gather represents a large amount of data, which is the reason for why it was downscaled and cut before fed to the network. The downsized images are still to be considered large for a neural network, limiting the possible batch size due to high memory demand. This is a restrictions of the network model as the network optimizes weights between each batch. Since the batch size was limited to 2 shot gathers per batch, the network will update parameters based on a small variance and might therefore adjust weights in a wrong direction or with a too high magnitude. This can, to some extent, be countered with decreasing the learning rate, but this will then increase training time needed for reaching optimal results.

Seismic data contains many fine details which have to be kept intact. A convolutional layer with no padding will shrink the size of an image. Thus, at the end zero padding was applied, but it caused some edge effects. To counter these, mirror padding was applied before zero padding, Figure 5.9. The edge effects would therefore be introduced in the padded rows/columns of the data which were to be cut off. Mirror padding may not be optimal, as it pulls some stronger events into the image at high offsets. A case with deeper strong intercepting SI-noise might pull some features into the output due to the mirroring along the last trace in the shot gather, as can be seen in Figure 6.12c.

The way the data is fed to a model might impact the running time considerably. Reading and writing data take time and might cause a queue where the nodes on the GPU are waiting for information, thus remaining idle. It is preferable to use all nodes on the GPU to their full extent during training, to reduce the run time. One way of doing this would be to load all data into a stack, which implies all reading is completed before the network is launched, narrowing the waiting time for the network. The dataset used in this thesis is abnormally large for machine learning standards, and loading all data into memory at once would require Terra-bytes of memory. Such amounts of memory is not easily accessible and would likely cost more than beneficial to use.

The seismic data has a very large dynamic range. Typically amplitudes of interest

vary from more than 10^5 near the water bottom and for refracted events to less than 0.01 after 4-5 second TWT. Many of the network models struggled with recreating these high amplitude shallow events, and it was shown that the feature maps in NDCNN1 and NDCNN4, Figures 6.18 and 6.19, omitted important characteristics in favour of focusing on these shallow events. This caused the models to yield sub-optimal results as datasets without such events would likely perform better. Four different scaling methods were implemented as an attempted solution to this problem. All scaling methods, except for normalization, adjusted the dynamic range, thus reducing the difference between the low and high amplitudes. Normalization was the scaling method working best, and also the only scaling method not adjusting the dynamic range. It is evident from the results that the scaling methods did not work optimally, but reducing the dynamic range still seems like a feasible solution. A much used trick in seismic processing is to use a gain function (T2 or maybe T1.6) to reduce the dynamic range in the data. In our case this was not really an option since a gain would boost the (already high amplitude) noise disproportionate to the underlying geology.

The intercepting SI-noise is well attenuated in the results discussed in Chapter 6. This particular SI-noise is only one of a range of different types of SI-noise used for training. The denoising of SI-noise with different dips and moveout are presented in Appendix A, Section A.4. The network models perform well for SI-noise coming from astern, Figures A.16 and A.17, but struggles more when the noise intercept with the same dip as the underlying geology, Figures A.12 and A.13. High amplitude SI-noise coming from the side makes the network models fail, as there are no underlying geology to map, Figures A.14 and A.15. These shots are presented in A.23. The underlying geology is entirely covered with high amplitude SI-noise. This noise is shot to shot coherent, and the networks therefore has no features to map from the underlying geology. It is likely that this is the reason for why the networks struggled in this area.

7.3.2 Network structure

The network structure is an important part where restrictions may lie. Comparing the NDCNN models with the U-NET models shows a major difference in performance of the network structure. NDCNN2 has a significantly longer run time than U-NET1, even though U-NET1 has four times as many parameters. It seems evi-

dent that even though the number of parameters affect the run time, the size of the image is much more important. The U-NET models have a substantial difference in parameters, but this does not show in the run time. This is likely because the model structure has become optimal enough to begin to be affected by the bottlenecking of reading and writing to files.

The loss function used in these models might not be optimal. The results presented in Chapter 6 support that MAE is the optimal loss function, but from a theoretical point of view Huber should give a better fit. There might be an error in the implementation of the loss function, or eventually that the example presented was not representative enough to conclude with regards to the best performing loss function. Although the loss seems to converge after 10-15 epochs for almost every network model, the results still improve until 35-40 epochs. This should in theory not be the case, and is likely explained by a loss function which is not optimal enough to view the large seismic files and take the full dynamic range into account.

Each network structure involves many hyper parameters to be set. There are rules of thumb as to how one should move forward when designing a model. However, a nertwork model will never be optimal without trial and error, and there will always be room for improvements.

7.4 Industry aspect

The UNET1 network gave the best results similar to the performance of industry standard denoising a few years ago (Gulunay et al., 2005). Certain SI-cases exist where the network model fails, Section A.4, but the overall denoising capability is rather good, as can be seen in the denoised stack, Figure A.19. It is evident that the industry steadily improves the processing software and use current standard denoising, as shown in Figure A.20. The results shown in this thesis can not compete with such industry standard for now, but neural networks still perform better than industry standard denoising with respect to time.

Real time denoising, opens up for the possibility to test whether data has to be re-shot during acquisition. The data flows in continuously and real time denoising by using neural network makes it easier to efficiently create rough estimates to whether the data is too contaminated or not. It is less common for contractor companies to actually re-shoot now a days, but it may still prove a valuable tool for

QC geophysisists on board.

Neural network opens up for more efficient and robust quality control of seismic processing possibly in real time. It may then be possible to check whether any interesting features exist in the data and if it is worth spending time and resources on processing. If data with no important information can be omitted easily, the contractor companies might increase their efficiency and spend more time on important data.

7.4.1 Execution time

The efficiency of denoising algorithms are important to the industry. The less computing time needed for denoising, the more time (and thus money) saved. Neural networks are, as presented in Table 6.4, very efficient when fully trained. Direct comparison with standard industry denoising is not an exact science as the computing time needed varies a lot based on the severity of the SI-noise as well as the survey. We therefore assume that denoising of a single shotgather takes x seconds. A typical seismic survey may cover an area of $3000km^2$ which requires approximately 10^6 shots where the towing vessel tows 12 reciever cables. The total computer time required for this will be:

$$\frac{10^6 \cdot 12 \cdot x}{3600 \cdot 24} \approx 139 \cdot x \text{ days} \tag{7.1}$$

This will of course be ran in parallel, but give an estimation of how much computing time is needed for denoising a marine seismic survey.

Assume that 10% of the data is denoised, taking approximately $14 \cdot x$ days. If this data was used to train a neural network, where the training might take multiple days (or even weeks), the processing of the remaining 90% could then be done in real time. Another example would be to train on vintage data and then denoise in real time when new data (from the same area) comes in. It is, as mentioned, not possible to give an exact estimation of the value x, but the U-NET (Table 6.4) used a mean of 0.02 seconds for each shot gather. Some of the NDCNN models required more computation time, where the most demanding model required 0.16 seconds. This is, according to processors at CGG, substantially lower than conventional SI-denoising.

8 | Conclusion and further work

8.1 Conclusion

Neural networks have become increasingly more popular during the last 5 to 10 years. They have been tested in various fields of application, but with less studies carried out in the field of seismic data processing. It is common for the North Sea to have rough weather during the winter months. This results in high seismic activity during summer, as the weather conditions tend to be better. When multiple surveys are carried out simultaneously, SI-noise becomes a problem. If the SI-noise could be removed more automatically on board, it would potentially increase overall efficiency for the vessels and strengthen the QC on board. It may also be deployed onshore, with a potential to speed up processing.

Using convolutional neural networks for SI-noise removal has been proven to work in this thesis. The results obtained demonstrate that a high percentage of the noise is attenuated, while mostly all geological information is kept intact. The SI-noise used in this thesis is unusually strong which therefore creates a more complex problem than what normally encountered during seismic acquisition. Thus the performance of each of the network models considered is therefore determined for a highly realistic acquisition case.

All data was processed in the shot domain, which may not be the optimal domain to work in. A resorting or transformation of input data will likely enhance the results, but will also remove the real time denoising aspect of the network models. Never the less, the network models performed well in the shot domain, with U-NET being the best both with respect to output results and computational efficiency. It required only 7.5 hours to train, and used 0.02 seconds per seismic shot gather once finished training. This is significantly faster than any existing industry denoising. The results does not compete completely with state-of-the art denoising, but proves that denoising with neural networks is possible.

Furthermore, it is worth to mention that since we clearly can use a neural network to efficiently attenuate seismic interference noise, the same network can probably also be trained to tackle other types of noise. It should in if fact also be applicable

for processes like deghosting and demultiple. As such, neural network seismic data-processing could become a very valuable approach in the not too distant future.

8.2 Further work

There are multiple aspects of both the dataset and the network models which this thesis does not test. One of the most evident improvements would be to identify a way for the network model to handle a significantly increased batch size. This will increase the variance of the models and might make it possible to better remove SI-noise with conflicting dip, as well as a generalized improved result. There are numerous ways of solving this problem. One of the simpler ways would be to upgrade the hardware, thus increasing the available GPU memory. This is an easy fix, given the available funds, but will not increase efficiency of the model. As demand increases, this will only postpone the problem, rather than fixing it.

Instead of increasing hardware capabilities, the way the data is represented and fed to the network can be optimized. Moving to another domain, such as common channel, will both break the coherency of the SI-noise, but will also enable for larger batch sizes, as a common channel gather can easily be cut into smaller pieces (Sun et al., 2019). The downside is that the network looses real time capabilities, as preprosessing of a full line is needed to create common channel gathers. Alternative ways of sorting and/or transforming data should be further investigated.

Seismic data contains a large dynamic range which causes problems for many of the network models presented. The shallow events are high in amplitudes and take too much "focus" from the network. Muting the high-amplitude reflections and refractions is one way to solve this problem, unless used for FWI. It is an irreversible process, but the data is commonly muted at a later processing step anyways. Another way of solving this problem would be to employ a scaling method which balances the dynamic range without breaking the model. Both cubic and square root amplitude balancing were implemented, but gave poor results. Further work should include alternate scaling methods.

The U-NET gave significantly better results than the other network models tested, and it seems to be a good network structure for seismic applications. The network model has not been tested thoroughly enough yet and there are likely room for improvements in the model structure.

SI-denoising of marine seismic data in the shot domain is a complex task. Since the results presented here proves that it is feasible, there is a likelihood that other types of denoising might work, such as: demultiple, deswell and deghost.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from https://www.tensorflow.org/.

Agostinelli, F., Hoffman, M., Sadowski, P., and Baldi, P. (2014). Learning activation functions to improve deep neural networks. *CoRR*, abs/1412.6830. arXiv:1412.6830.

Akbulut, K., Saeland, O., Farmer, P., and Curtis, T. (2005). Suppression of seismic interference noise on gulf of mexico data. *SEG Technical Program Expanded Abstracts 1984*, pages 527–529. doi: https://10.1190/1.1894083.

Anaconda (2019). Anaconda Software Distribution. https://www.anaconda.com/distribution/ [Read 27.02.19].

Apple (2019). Macbook pro. https://www.apple.com/no/shop/buy-mac/macbook-pro/15-tommer [Read 05.05.19].

Baardman, R., . T. C. (2019). Classification and suppression of blending noise using convolutional neural networks. Society of Petroleum Engineers. doi: 10.2118/194731-MS.

CGG (2018). Seismic acquisition [FIGURE]. https://www.cgg.com/imgs/overview-images/4_full.jpg [Downloaded 26.08.18].

Chaitanya, C. R. A., Kaplanyan, A. S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., and Aila, T. (2017). Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Trans. Graph.*, 36(4):98:1–98:12.

Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. The MIT Press, Cambridge MA. ISBN: 978-0-262-03358-9.

Chen, K. and Sacchi, M. D. (2015). Robust reduced-rank filtering for erratic seismic noise attenuation. *GEOPHYSICS*, 80(1):V1–V11. doi: https://doi.org/10.1190/geo2014-0116.1.

Chollet, F. et al. (2015). Keras. https://keras.io.

Cisco (2018). [Internet] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html#_Toc484813970 [Read 28.08.18].

Dondurur, D. and Karslı, H. (2012). Swell noise suppression by wiener prediction filter. *Journal of Applied Geophysics*, 80:91–100. doi: https://doi.org/10.1016/j.jappgeo.2012.02.001.

Elboth, T., Geoteam, F., and Hermansen, D. (2009a). Attenuation of noise in marine seismic data. *SEG Technical Program Expanded Abstracts 2009*, pages 3312–3316. doi: https://library.seg.org/doi/pdf/10.1190/1.3255547.

Elboth, T., Reif, B. A., and Øyvind Andreassen (2009b). Flow and swell noise in marine seismic data. *GEOPHYSICS*, 74(2):Q17–Q25. doi: https://doi.org10.1190/1.3078403.

Ellacott, Stephen W., M. J. C. A. I. J. (1997). *Mathematics of Neural Networks*. Springer Scinence + Business Media, New York. ISBN: 987-1-4613-7794-8.

Gershenson, C. (2003). Artificial neural networks for beginners. *CoRR*, cs.NE/0308031.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Gondara, L. (2016). Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246. doi: 10.1109/ICDMW.2016.0041.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Gulunay, N., Magesan, M., and Baldock, S. (2005). Seismic interference noise attenuation. *SEG Technical Program Expanded Abstracts 2004*, pages 1973–1976. doi: https://library.seg.org/doi/abs/10.1190/1.1843302.

Han, J. and Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. In *From Natural to Artificial Neural Computation*, pages 195–201, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN: 978-3-540-49288-7.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Lecture 6e. rmsprop: Divide the gradient by a running average of its recent magnitude. Lecture in CSC321. University of Toronto. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf [Downloaded 04.03.19].

Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *Journal of physiology*, 148(3):574–591. PMCID: PMC1363130.

Huber, P. J. (1964). Robust estimation of a location parameter. *Annals of Statistics*, 53(1):73–101. doi: 10.1214/aoms/1177703732.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167. arXiv:1502.03167.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications*. Springer. ISBN 978-1461471370.

Jansen, S. (2013). Two marine seismic interference attenuation methods. Master's thesis, University of Oslo. DUO: http://urn.nb.no/URN:NBN:no-34309.

Jin, Y., Wu, X., Chen, J., Han, Z., and Hu, W. (2018). *Seismic data denoising by deep-residual networks*, pages 4593–4597. doi: 10.1190/segam2018-2998619.1.

Karlik, B. and Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence And Expert Systems*, 1(4):111–122.

Karpathy, A. (N.D.). Convolutional neural networks. http://cs231n.github.io/convolutional-networks/ [Read 21.01.19].

Krizhevsky, A. (2009). CIFAR-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

Kuffler, S. W. (1953). Discharge patterns and functional organization of mammalian retina. *Journal of Neurophysiology*, 16(1):37–68. doi: https://doi.org/10.1152/jn.1953.16.1.37.

Kumar, D. and Ahmed, I. (2011). Seismic noise. *Encyclopedia of Solid Earth Geophysics*, pages 1157–1161. doi: https://doi.org/10.1007/978-90-481-8702-7_146.

Kundu, P. K. (1977). *Fluid mechanics*. Academic Press: New York. 1st edition.

Lahiri, A. (2016). Chapter 6 - fourier optics. In Lahiri, A., editor, *Basic Optics*, pages 539 – 603. Elsevier, Amsterdam.

Larner, K., Chambers, R., Yang, M., Lynn, W., and Wai, W. (1983). Coherent noise in marine seismic data. *GEOPHYSICS*, 48(7):854–886. doi: https://doi.org/10.1190/1.1441516.

Laurain, R., Ruiz-Lopez, F., and Eidsvig, S. (2015). Managing and modeling the seismic interference. *In 77th EAGE Conference and Exhibition 2015*. doi: https://doi.org/10.3997/2214-4609.201412597.

Li, F.-F., Johnson, J., and Young, S. (2017). Lecture 11: Detection and segmentation. CS231n: Convolutional Neural Networks for Visual Recognition, Stanford, http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf.

Li, H., Yang, W., and Yong, X. (2018). *Deep learning for ground-roll noise attenuation*, pages 1981–1985. doi: 10.1190/segam2018-2981295.1.

Liner, C. L. (2000). On the history and culture of geophysics, and science in general. *The Leading Edge*, 19(5):502–504. doi: https://doi.org/10.1190/1.1438642.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *ICML*, 30.

Minsky, M. and Papert, S. A. (1969). *Perceptrons: An introduction to Computational Geometry*. The MIT Press, Cambridge MA, 1st edition. ISBN: 0-262-63022-2.

Mitchell, T. (1997). *Machine learning*. McGraw-Hill Pub. Co. (ISE Editions): London.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *The Foundations of Machine Learning*. MIT Press. http://mitpress.mit.edu/books/foundations-machine-learning-0.

Murphy, K. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press, New York. ISBN 978-0262018029.

Musset, A. E. and Khan, M. A. (2009). *Looking Into the Earth*. Cambridge University Press, New York. ISBN: 978-0-521-78574-7.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Nvidia (2019a). CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit [Read 26.02.19].

Nvidia (2019b). Nvidia cuDNN. https://developer.nvidia.com/cudnn [Read 26.02.19].

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378.

Oliphant, T. (2006). NumPy: A guide to NumPy. http://www.numpy.org/.

Pocatilu, P., Alecu, F., and Vetrici, M. (2010). Measuring the efficiency of cloud computing for e-learning systems. *WSEAS Transactions on Computers*, pages 42–51.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv e-prints*, page arXiv:1505.04597. arXiv:1505.04597.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408. doi: http://dx.doi.org/10.1037/h0042519.

Rosenblatt, F. (1961). *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Cornell Aeronautical Laboratory.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747. arXiv:1609.04747.

Sanchis, C. and Hanssen, A. (2011). Multiple-input adaptive seismic noise canceller for the attenuation of nonstationary coherent noise. *GEOPHYSICS*, 76(6):V139–V150. doi: https://doi.org/10.1190/geo2010-0367.1.

Santa-Cruz, D. and Ebrahimi, T. (2000). An analytical study of jpeg 2000 functionalities. In *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, volume 2, pages 49–52 vol.2. doi: 10.1109/ICIP.2000.899222.

Schlumberger (2017). Seismic Acquisition. *Schlumberger Oilfield Glossary* [Internett] Schlumberger. https://www.glossary.oilfield.slb.com/Terms/s/seismic_acquisition.aspx [Read 27.08.18].

Shen, H., Elboth, T., Tao, C., Tian, G., Wang, H., Qiu, L., and Zhou, J. (2019). Using data regrouping methods to attenuate shot-to-shot coherent interference noise in marine seismic data. *Earth and Space Science*. doi: https://doi.org/10.1029/2018EA000485.

Si, X. and Yuan, Y. (2018). *Random noise attenuation based on residual learning of deep convolutional neural network*, pages 1986–1990. doi: 10.1190/segam2018-2985176.1.

Smith, S. L., Kindermans, P., and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *CoRR*, abs/1711.00489. http://arxiv.org/abs/1711.00489.

Smith, S. W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Technical Publishing, 1st edition.

Sood, D. (2018). Backpropagation concept explained in 5 levels of difficulty. https://medium.com/coinmonks/backpropagation-concept-explained-in-5-levels-of-difficulty-8b220a939db5 [Read 19.02.19].

Sun, J., Slang, S., Greiner, T., Elboth, T., McDonald, S., and Geluis, L. J. (2019). Deblending of seismic data via deep convolutional neural networks. *Submitted to GEOPHYSICS*.

Tamura, S. and Tateishi, M. (1997). Capabilities of a four-layered feedforward neural network: four layers versus three. *IEEE Transactions on Neural Networks*, 8(2):251–255.

Taylor, L. and Nitschke, G. (2017). Improving deep learning using generic data augmentation. *CoRR*, abs/1708.06020.

Tensorflow (2019). convolutional.py. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/layers/convolutional.py [Read 05.03.19].

van Rossum, G. (1995). Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

Widrow, B. and Lehr, M. A. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442. doi: 10.1109/5.58323.

Willmott, C. J. and Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30(1). doi: 10.3354/cr030079.

Xie, J., Xu, L., and Chen, E. (2012). Image denoising and inpainting with deep neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 341–349. Curran Associates, Inc.

Zhao, H., Gallo, O., Frosio, I., and Kautz, J. (2017). Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57. doi: 10.1109/TCI.2016.2644865.

A | Appendix

This appendix presents full size images of the scaling methods (Figures A.1) - A.4, loss functions (Figures A.5 - A.7) and activation functions (Figures A.8 - A.11) which were only included as cutout in the main body of the thesis. These figures visualize the output from the network marked blue, and the difference between ground truth and output marked red. Ground truth and input can be seen respectively in Figures 6.8a and 6.8b. It also includes denoising results from NDCNN1 and U-NET1 with different types of SI-noise. The denoising results of a noise contaminated stack is appended where both industry standard SI-denoising and the U-NET1 denoising are included.

A.1 Scale

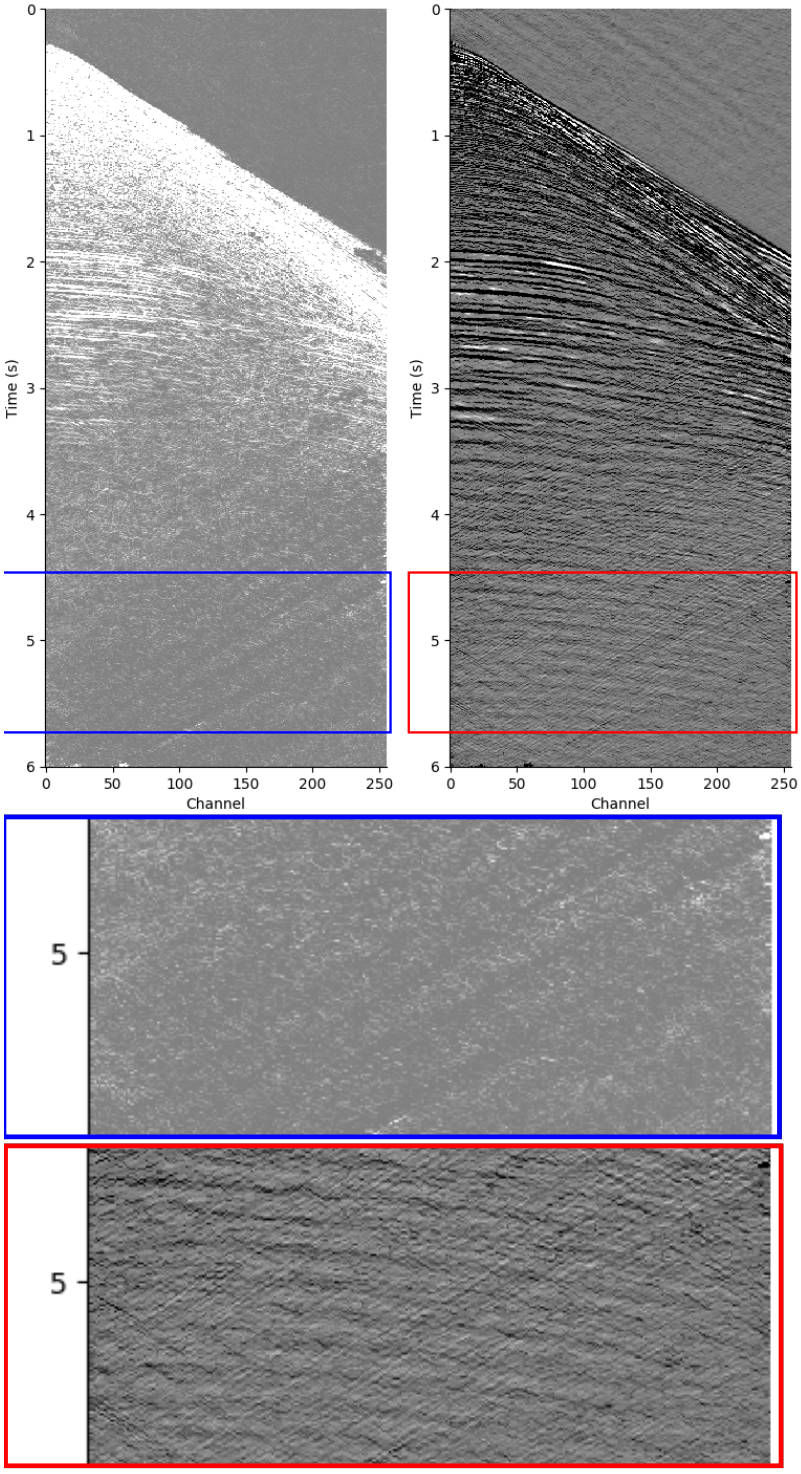


Figure A.1: *Figure visualizing the effect of cubic root scaling of the data. The blue and red sections visualize the output and the difference of the model.*

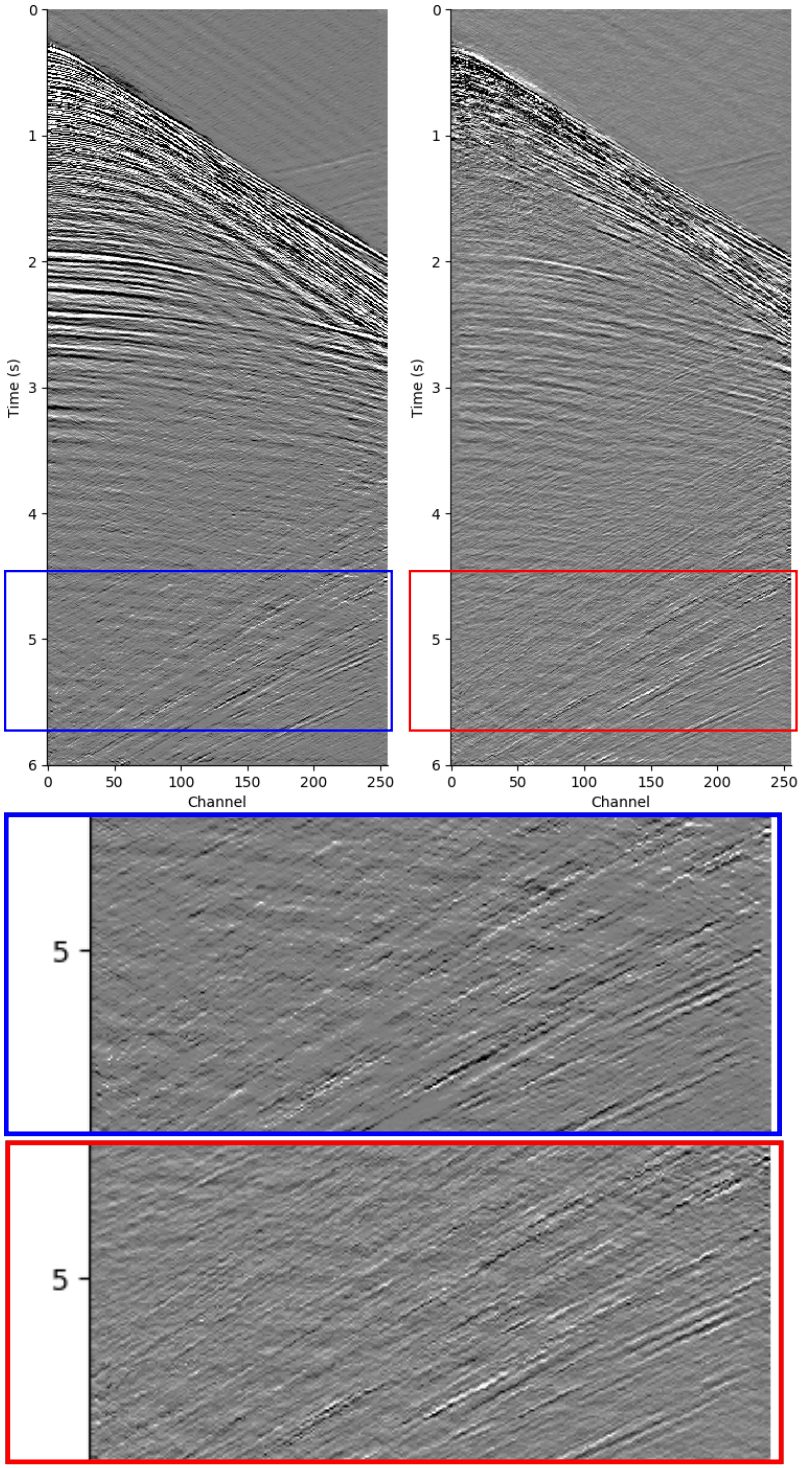


Figure A.2: *Figure visualizing the effect of square root scaling of the data. The blue and red sections visualize the output and the difference of the model.*

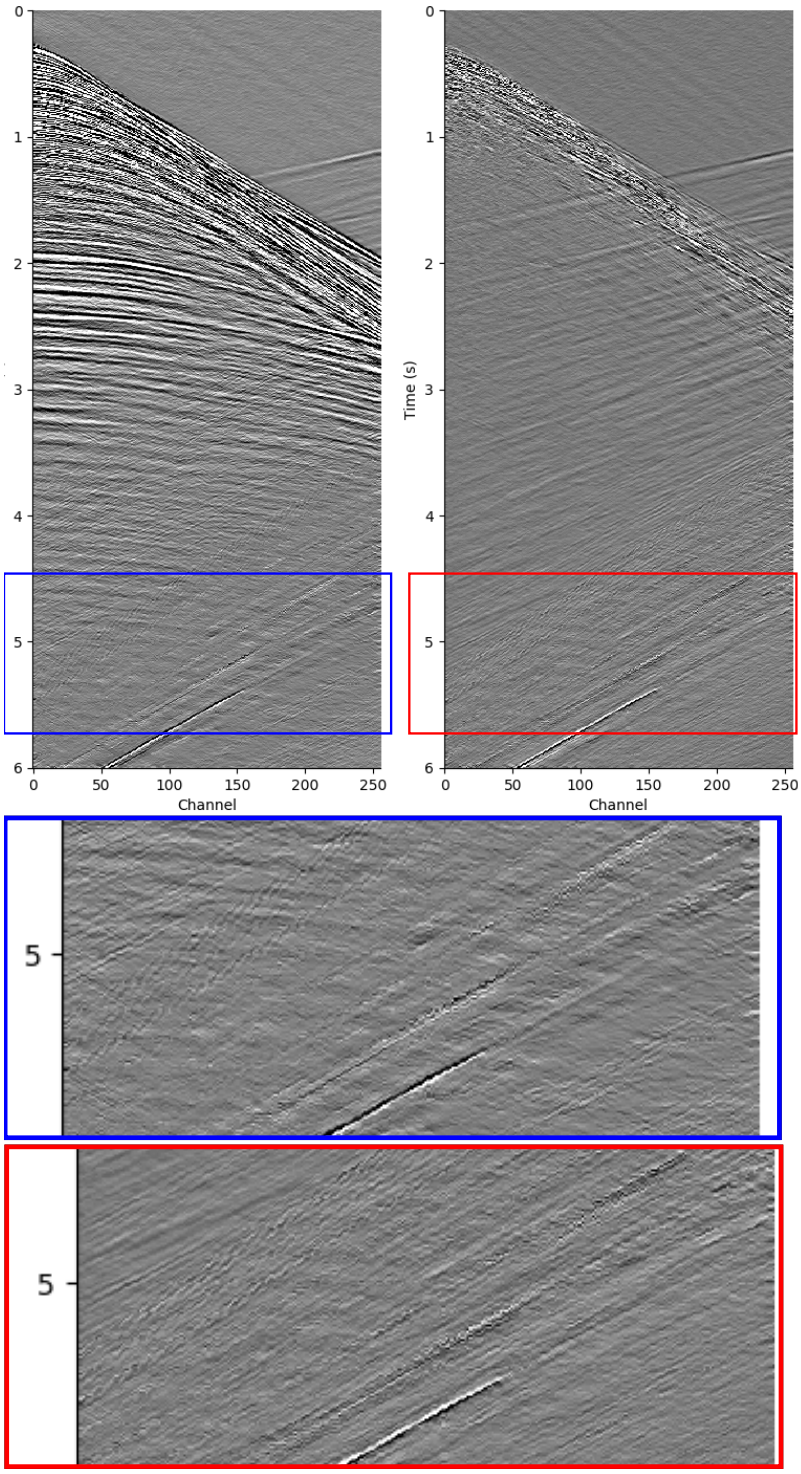


Figure A.3: *Figure visualizing the effect of thresholding of the data. The blue and red sections visualize the output and the difference of the model*

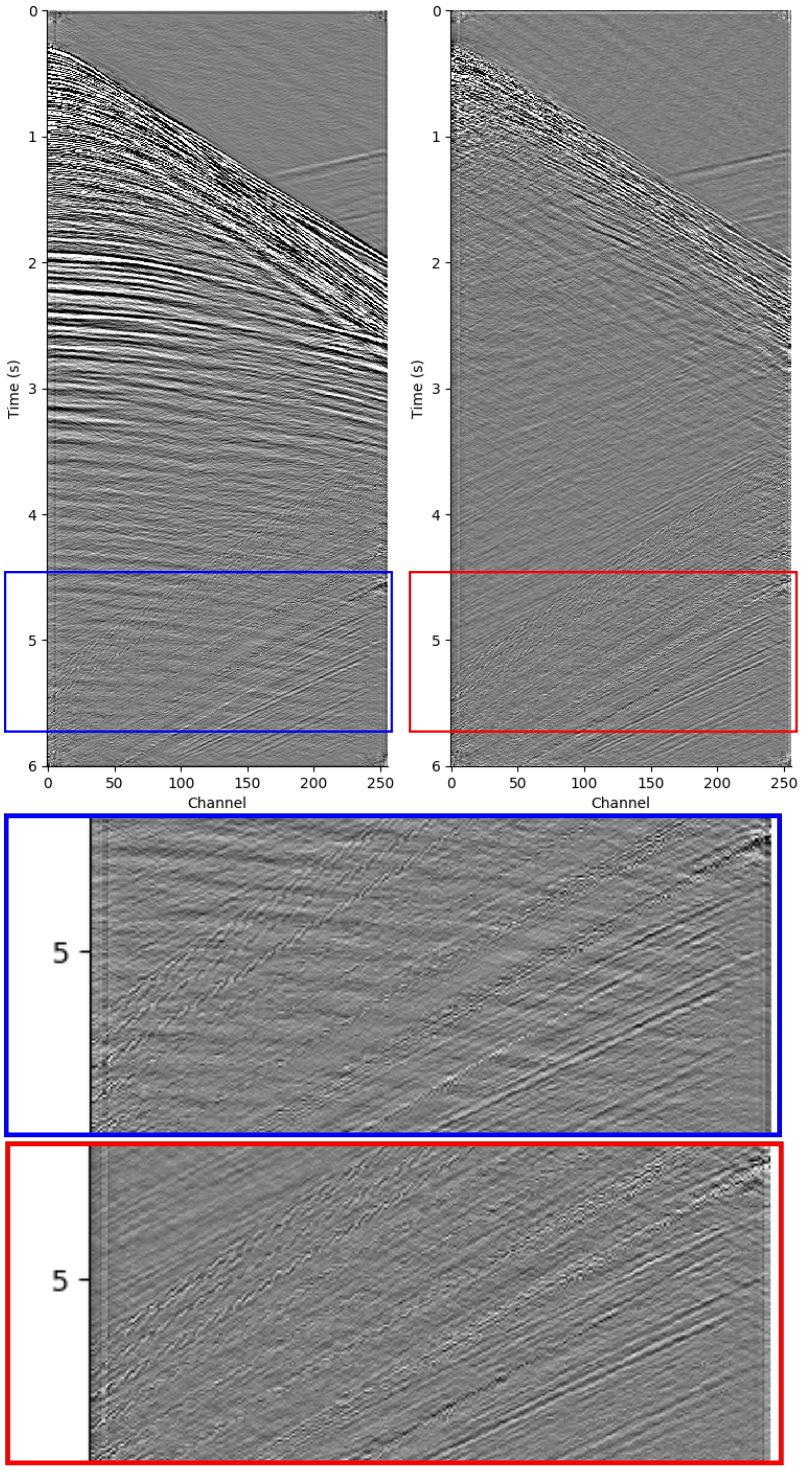


Figure A.4: *Figure visualizing the effect of normalization of the data. The blue and red sections visualize the output and the difference of the model.*

A.2 Loss

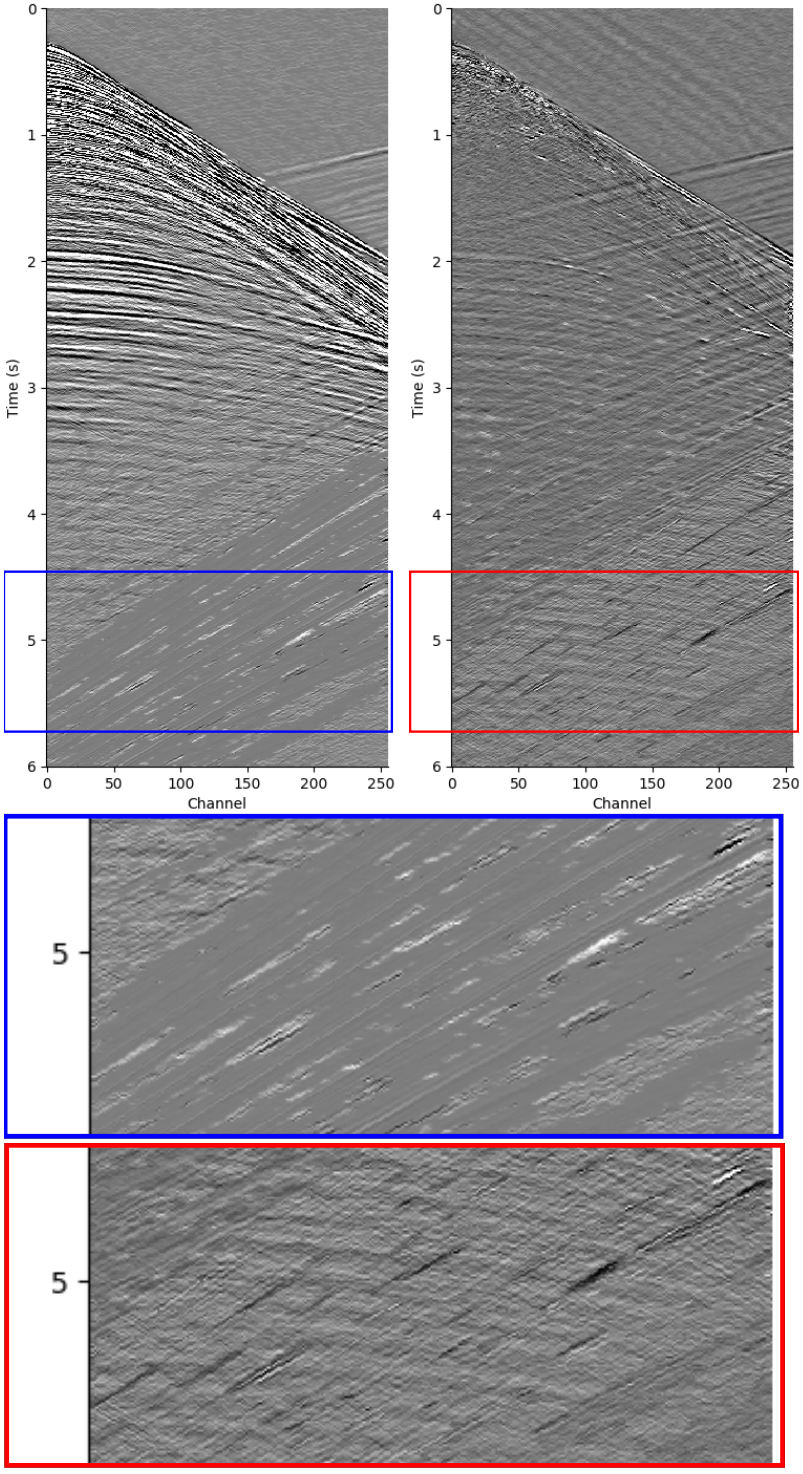


Figure A.5: *Figure visualizing the effect of using MSE loss function. The blue and red sections visualize the output and the difference of the model*

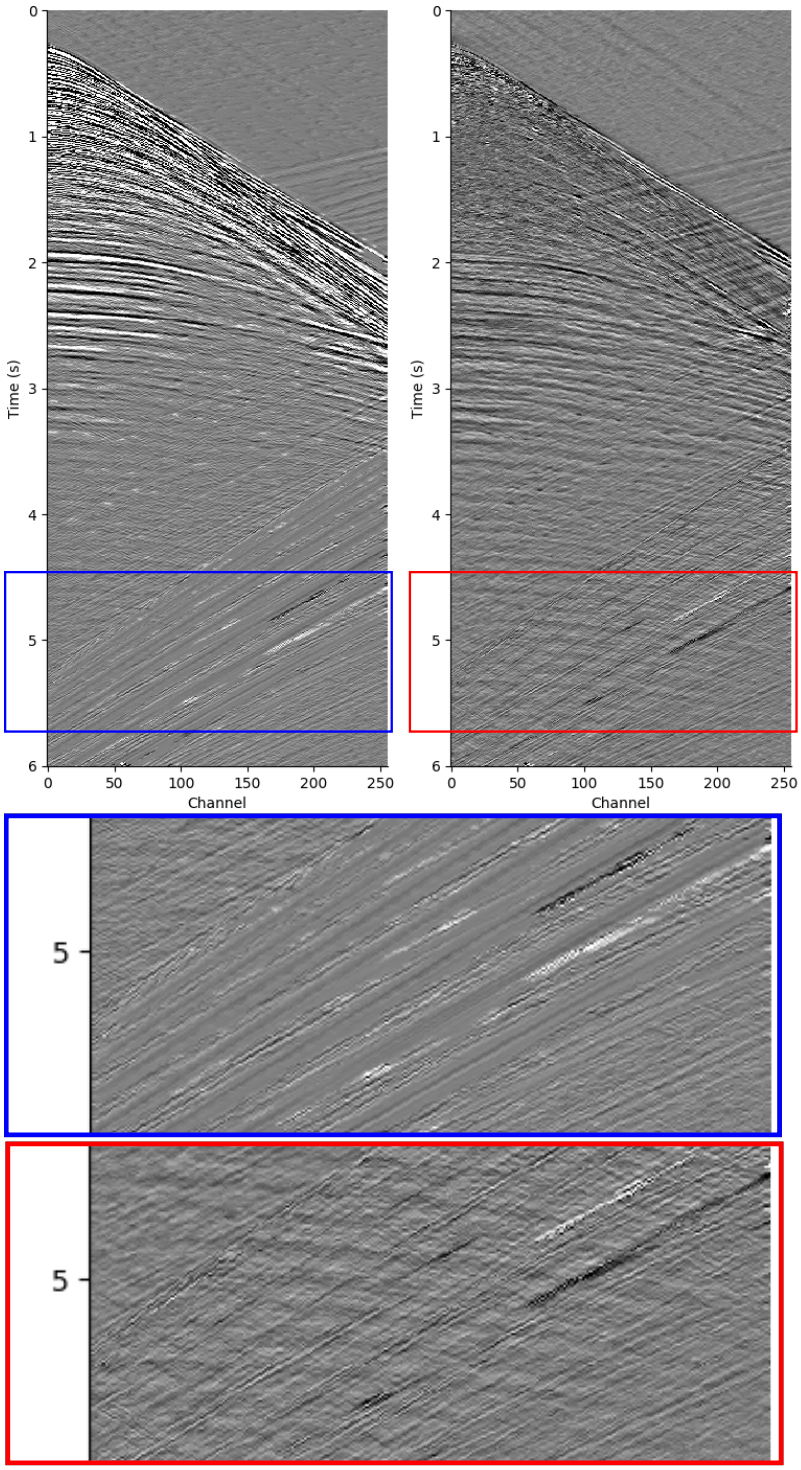


Figure A.6: *Figure visualizing the effect of using Huber loss function. The blue and red sections visualize the output and the difference of the model*

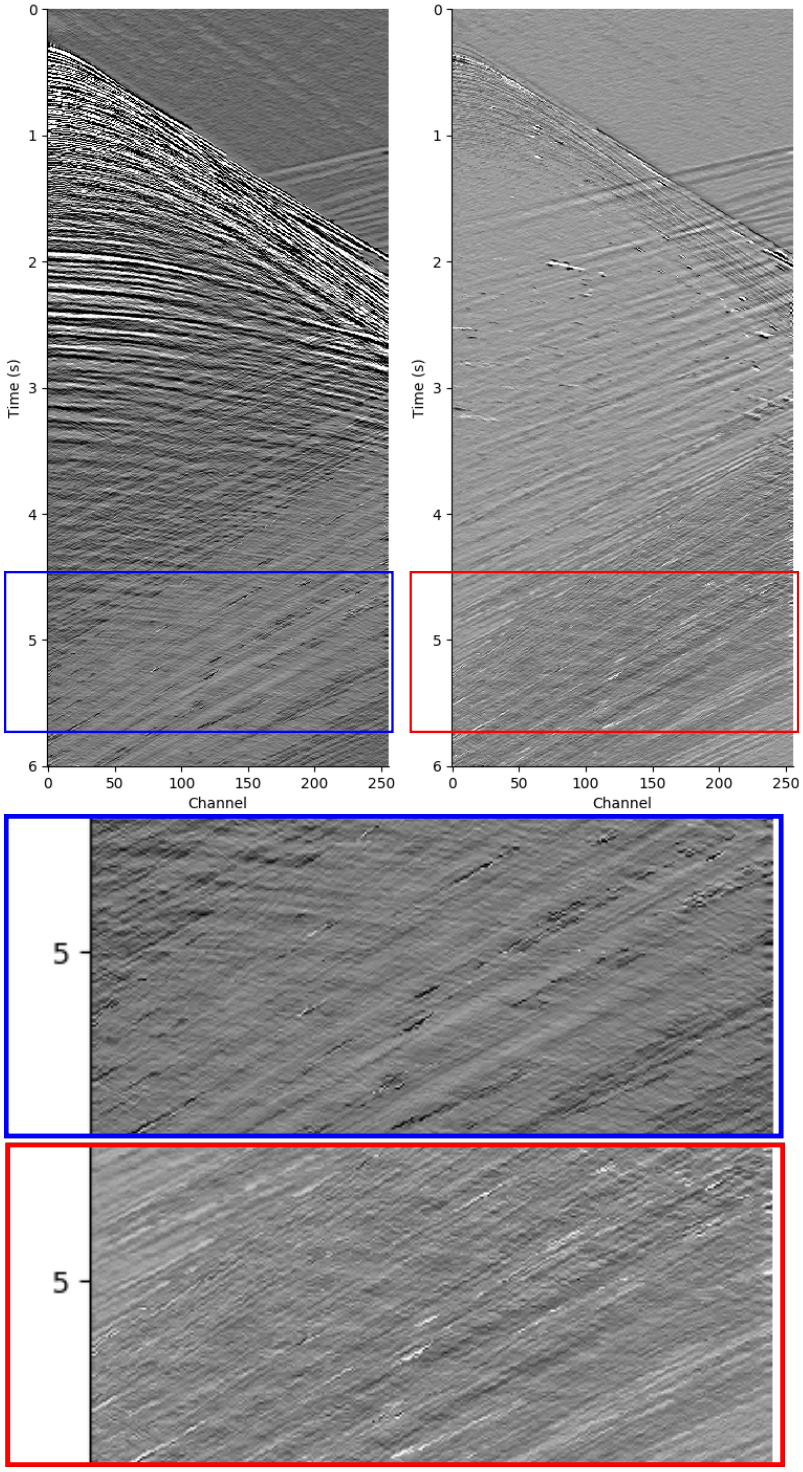


Figure A.7: *Figure visualizing the effect of using MAE loss function. The blue and red sections visualize the output and the difference of the model*

A.3 Activation

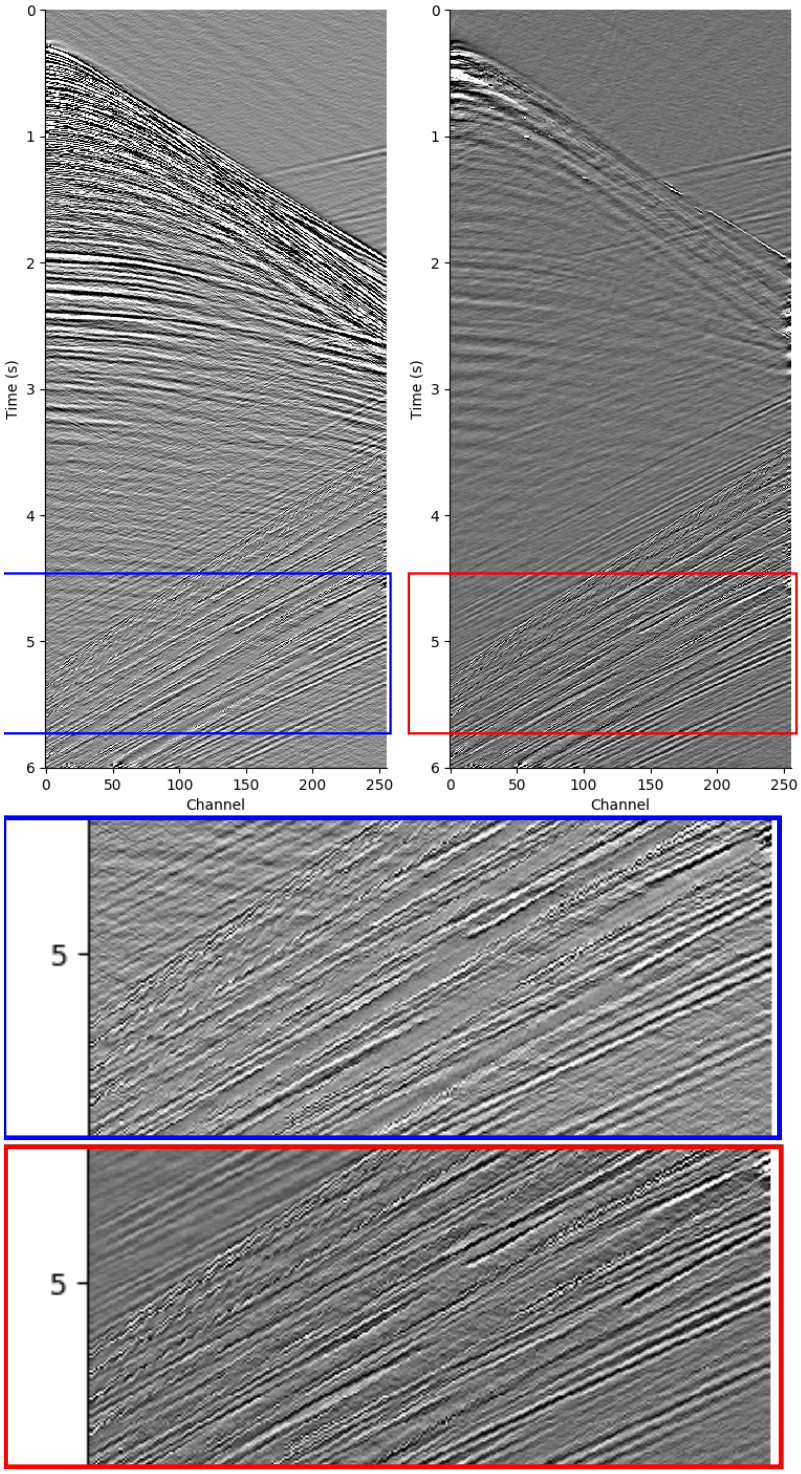


Figure A.8: *Figure visualizing the effect of using ReLU activation function. The blue and red sections visualize the output and the difference of the model*

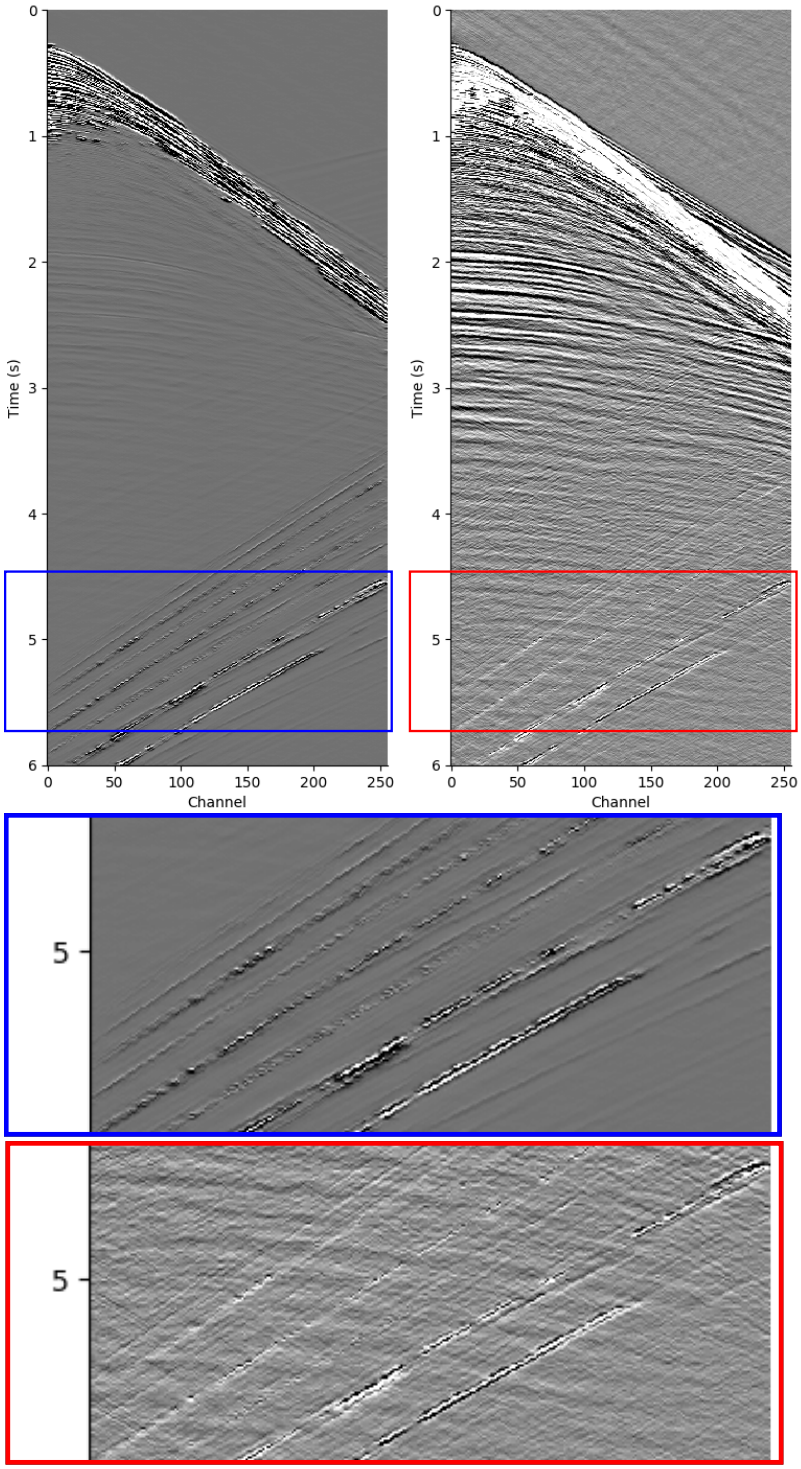


Figure A.9: *Figure visualizing the effect of using TanH activation function. The blue and red sections visualize the output and the difference of the model*

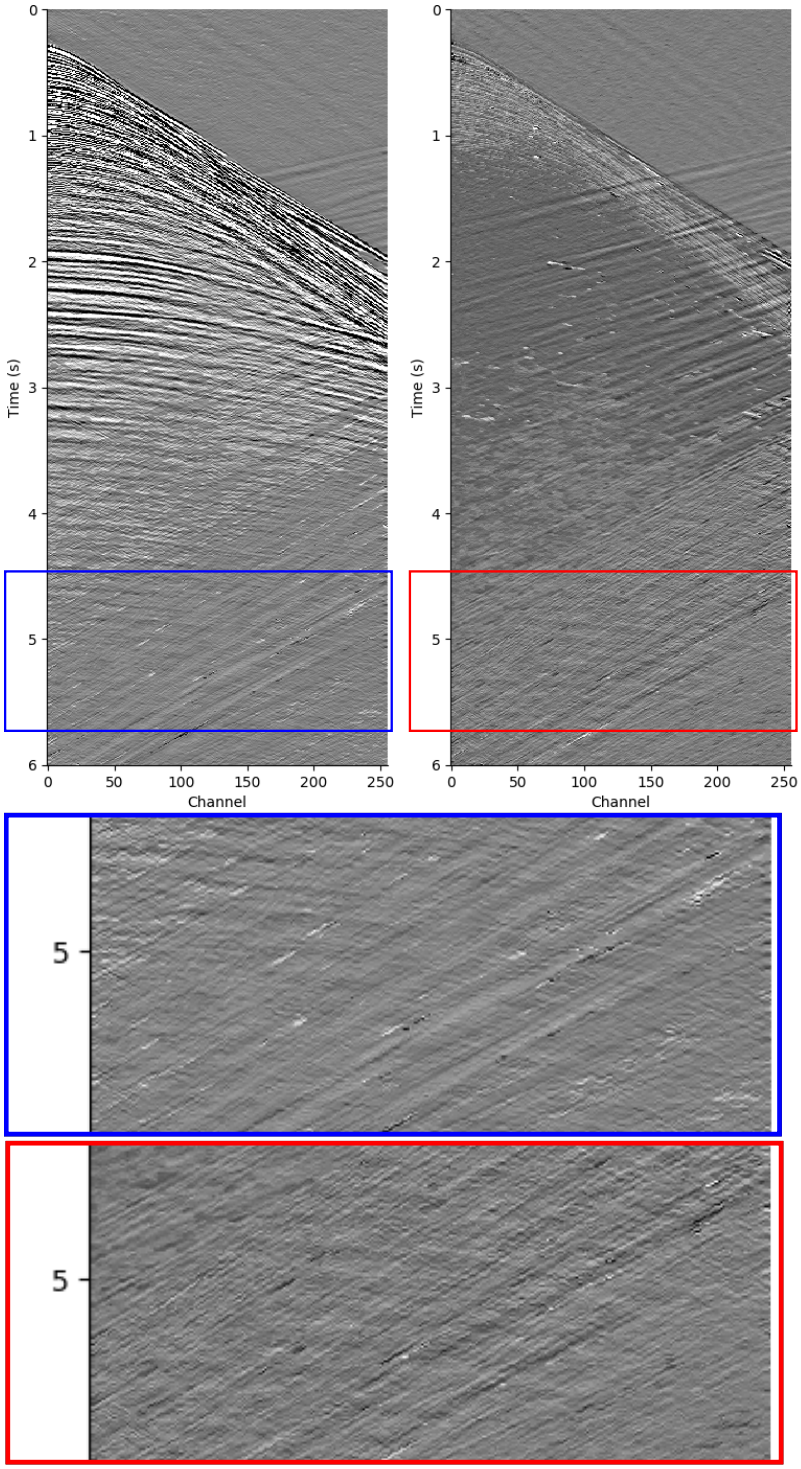


Figure A.10: *Figure visualizing the effect of using Leaky ReLU activation function with* $\alpha = 0.01$. *The blue and red sections visualize the output and the difference of the model*

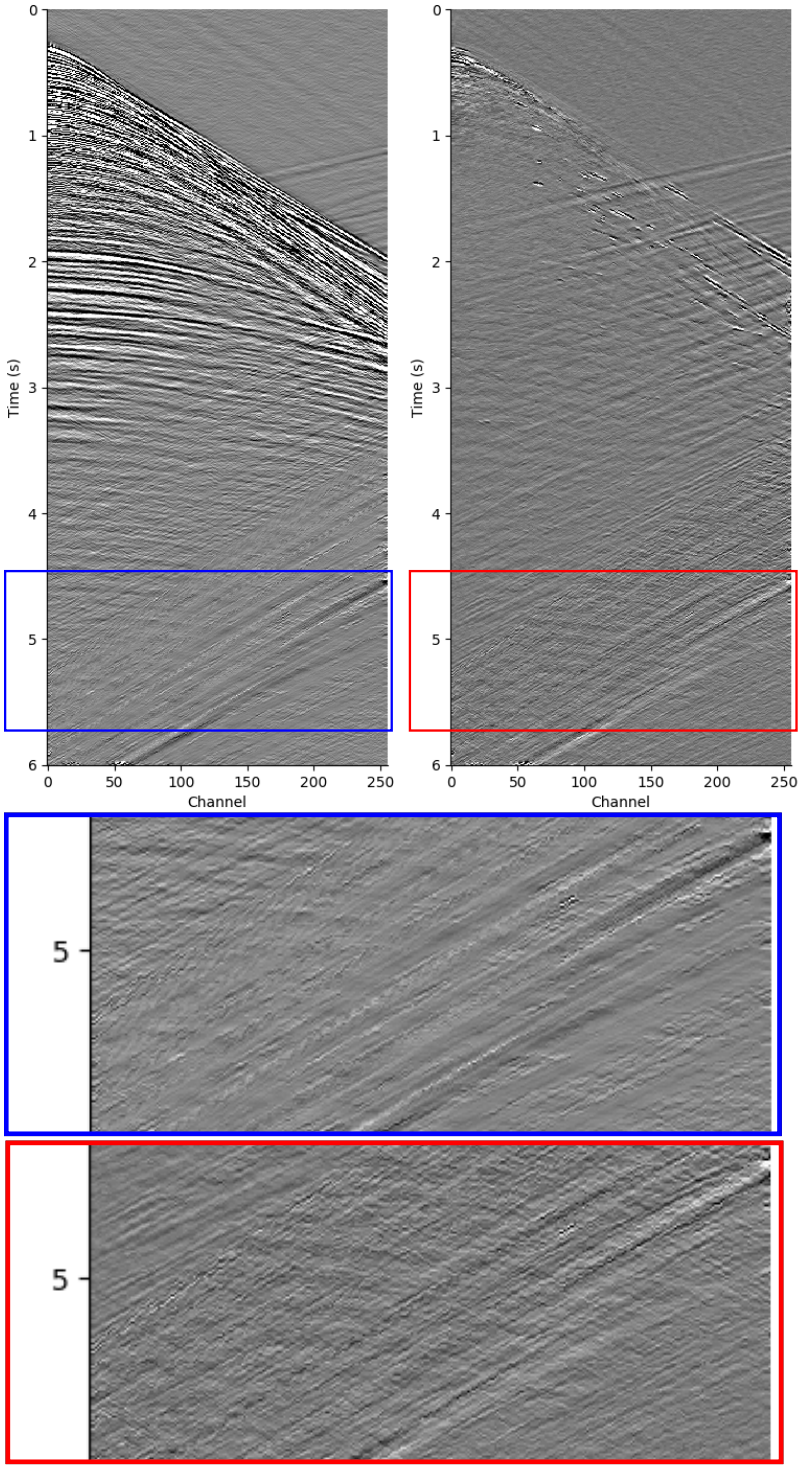


Figure A.11: *Figure visualizing the effect of using Leaky ReLU activation function with* $\alpha = 0.3$. *The blue and red sections visualize the output and the difference of the model*

A.4 Denoising results

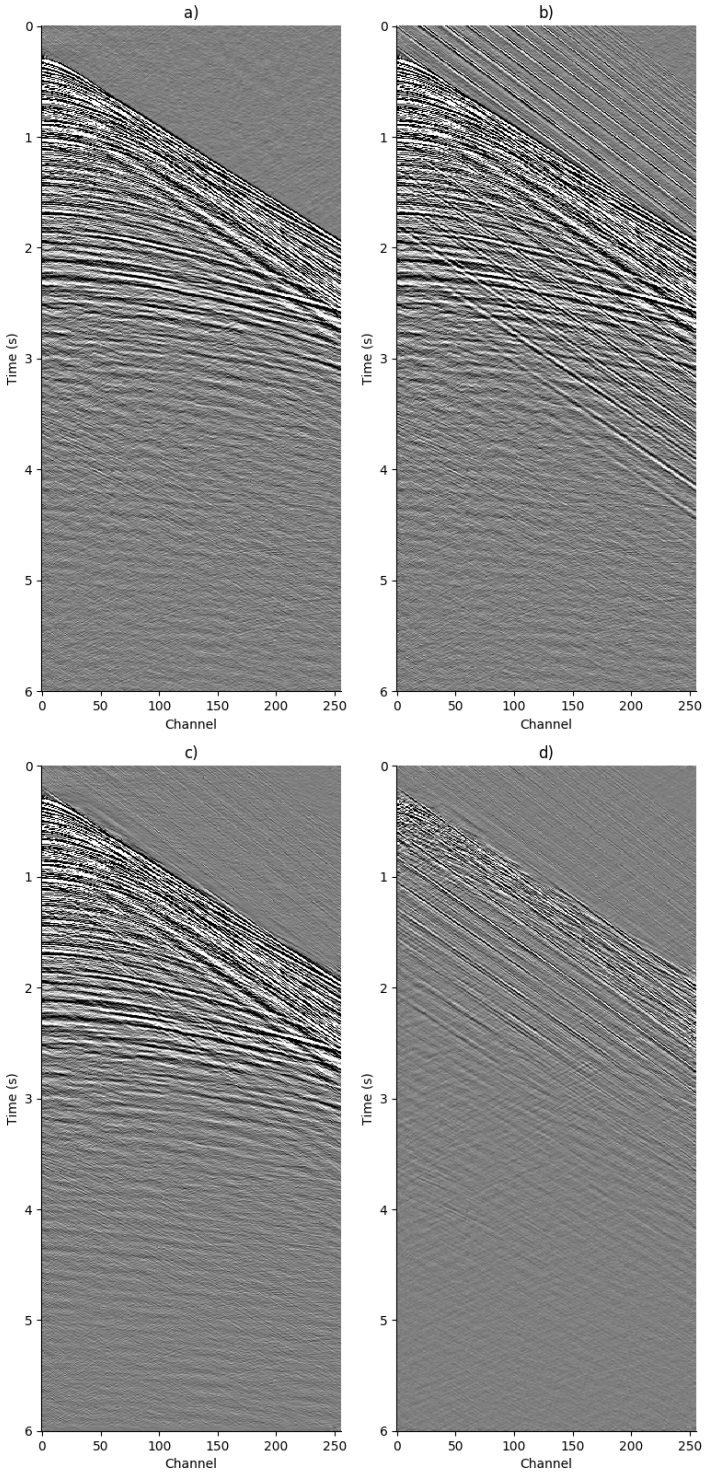


Figure A.12: *Denoising result from NDCNN1 with SI-noise coming from abow where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c)*

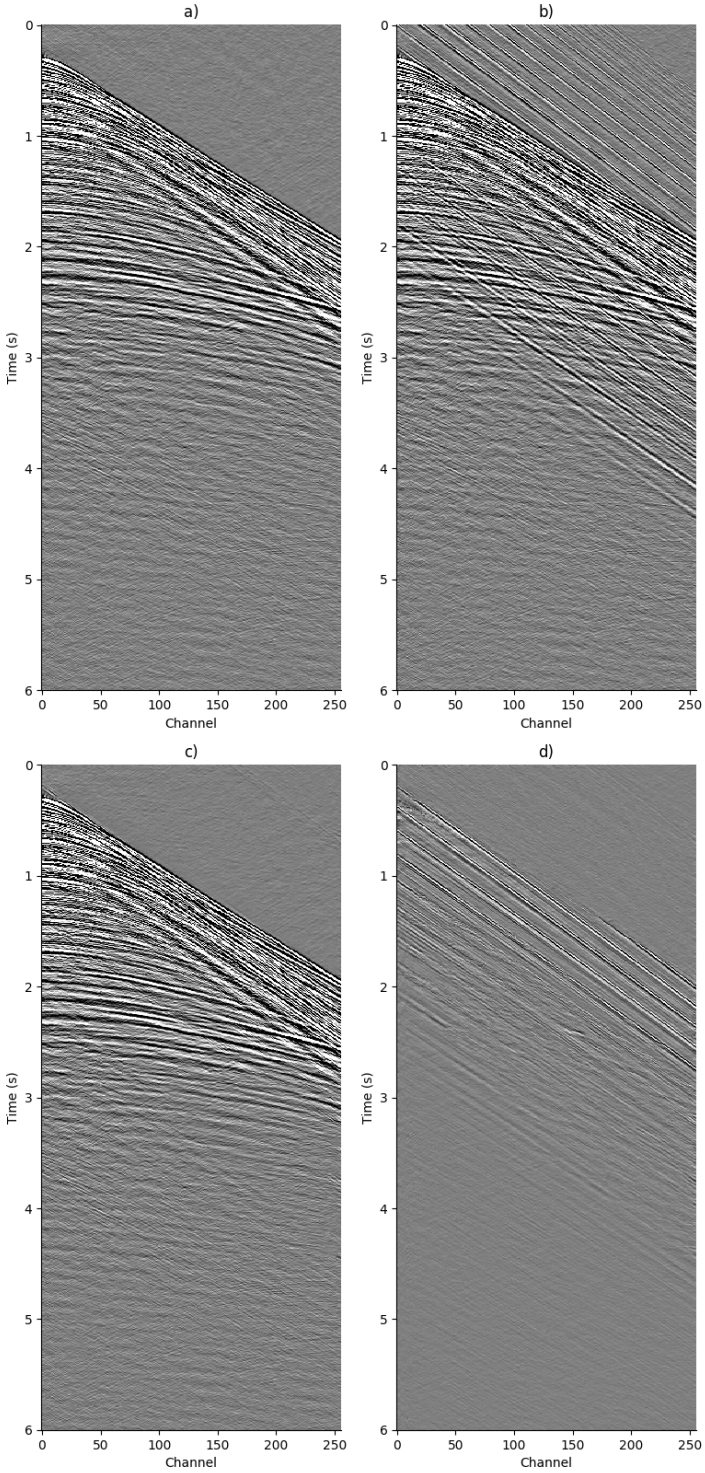


Figure A.13: *Denoising result from U-NET1 with SI-noise coming from abow where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c)*

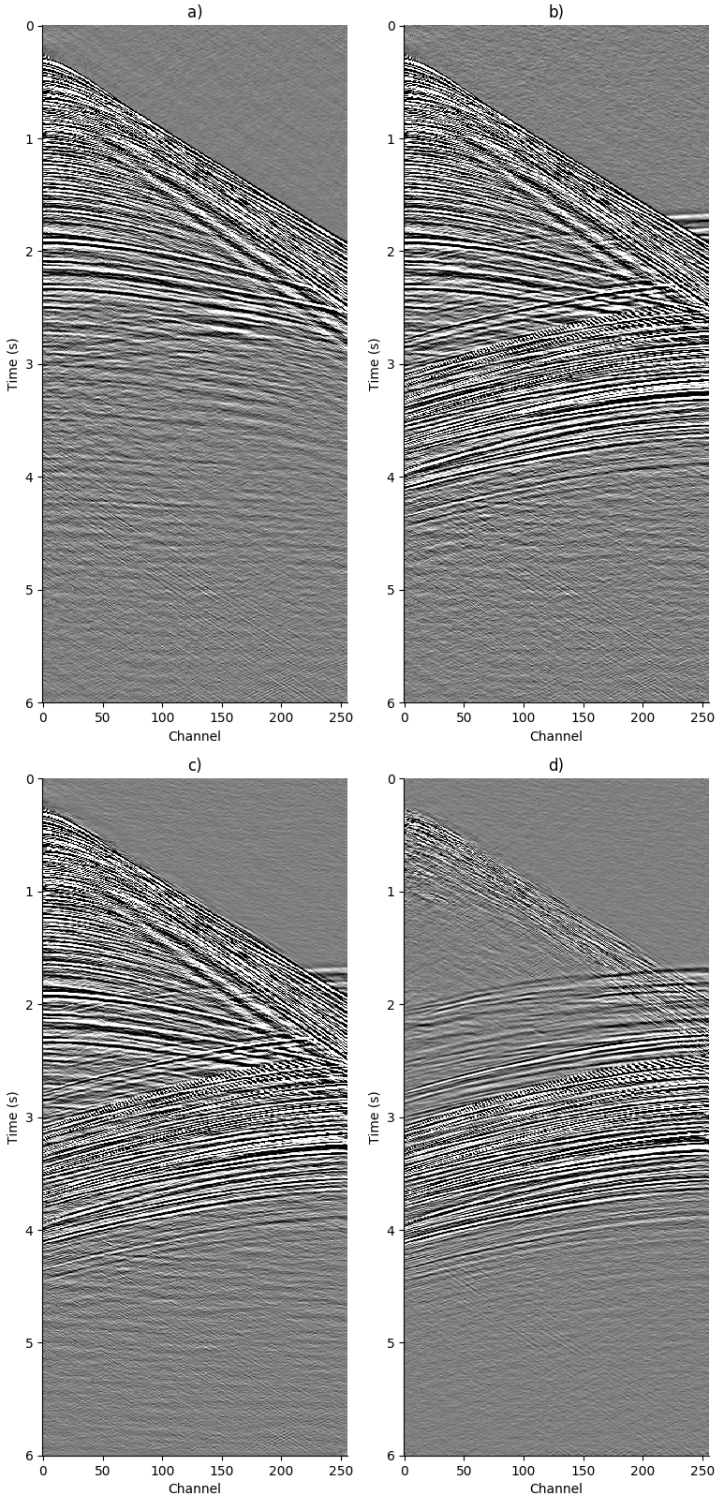


Figure A.14: *Denoising result from NDCNN1 with SI-noise coming from the side where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c)*

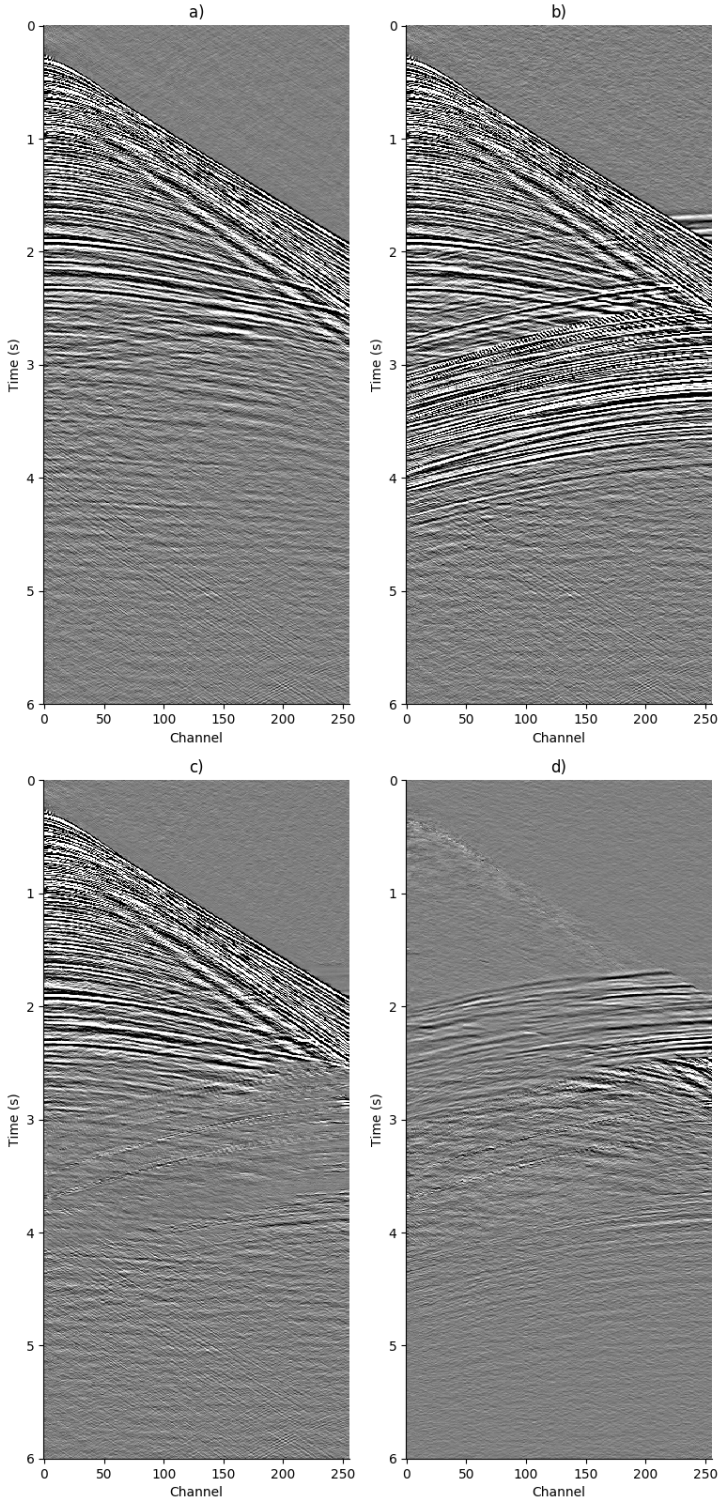


Figure A.15: *Denoising result from U-NET1 with SI-noise coming from the side where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c)*

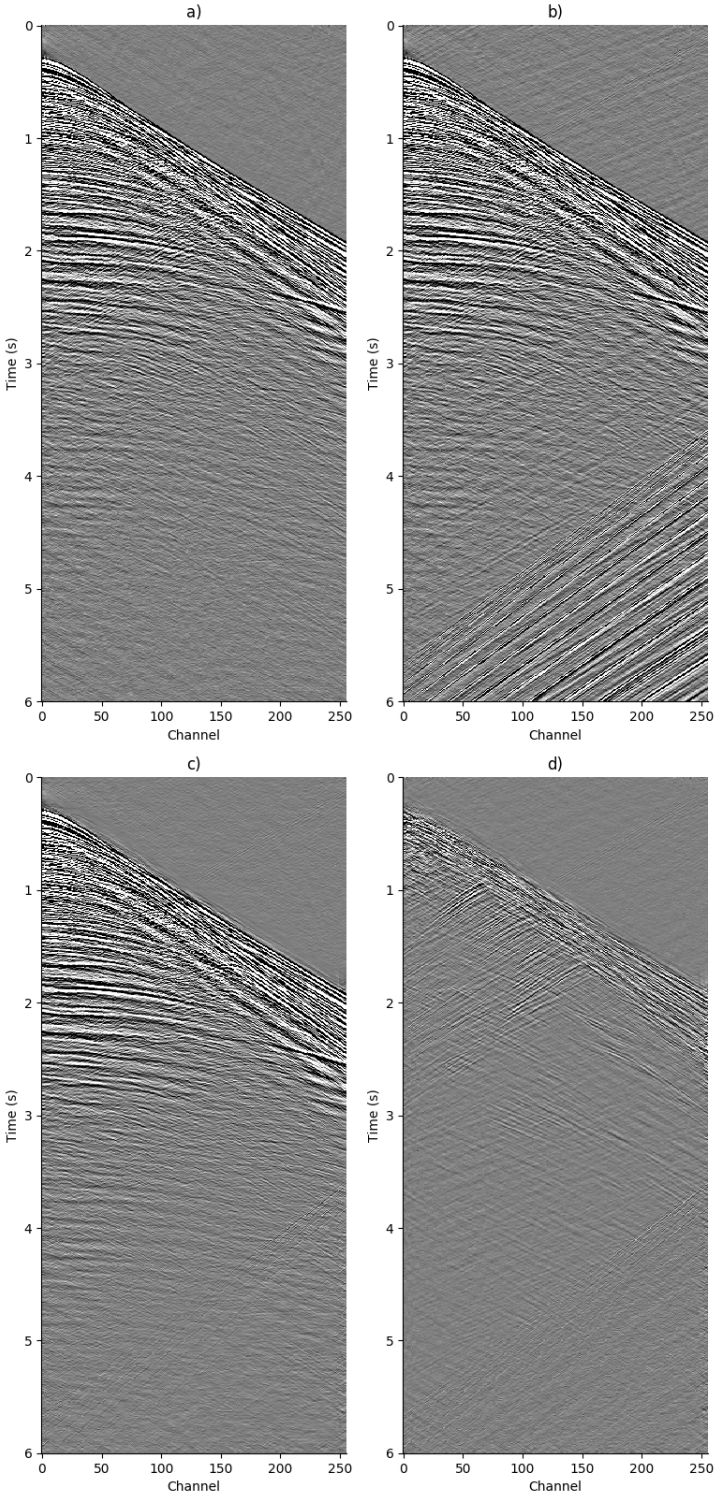


Figure A.16: *Denoising result from NDCNN1 with SI-noise coming from astern where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c)*

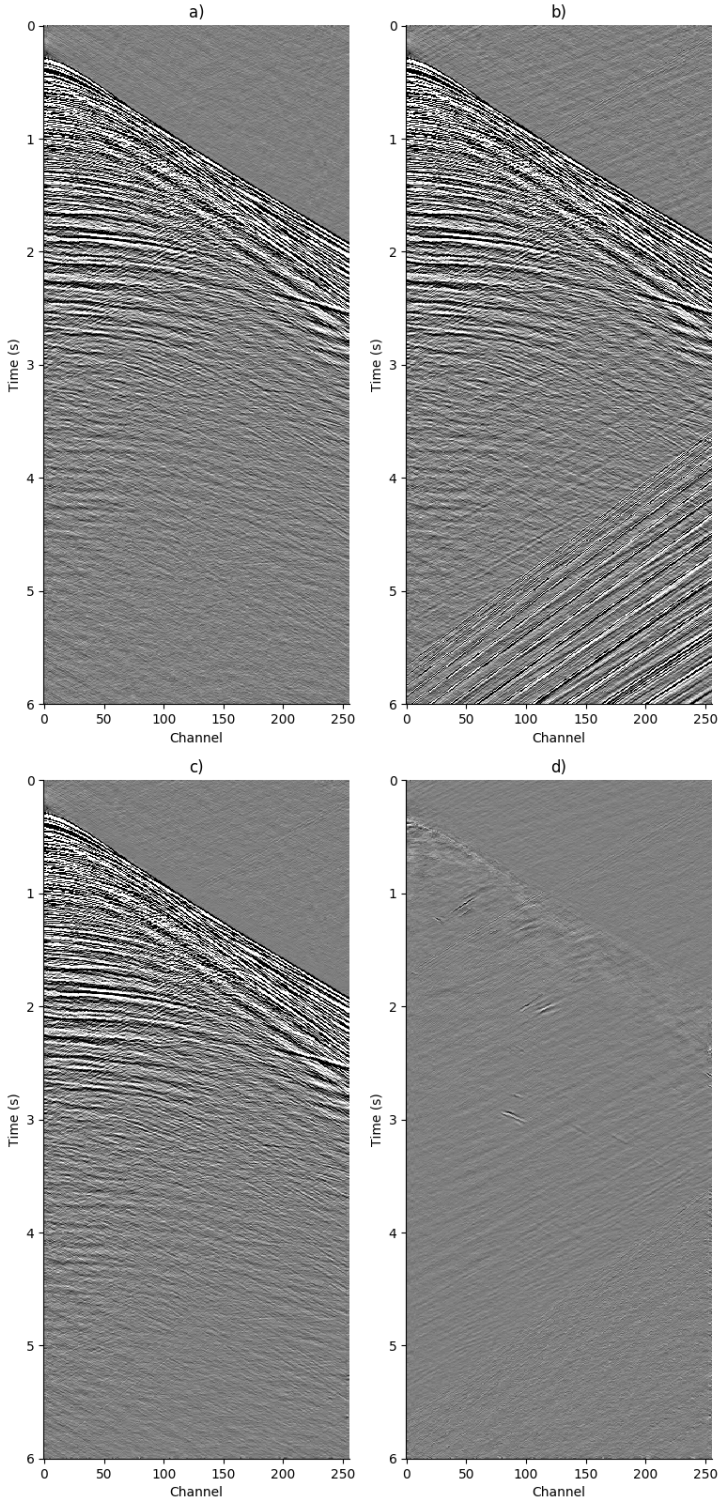


Figure A.17: *Denoising result from U-NET1 with SI-noise coming from astern where a), b), c) and d) visualize respectively clean image, input, output and difference (a-c)*

A.5 Stack

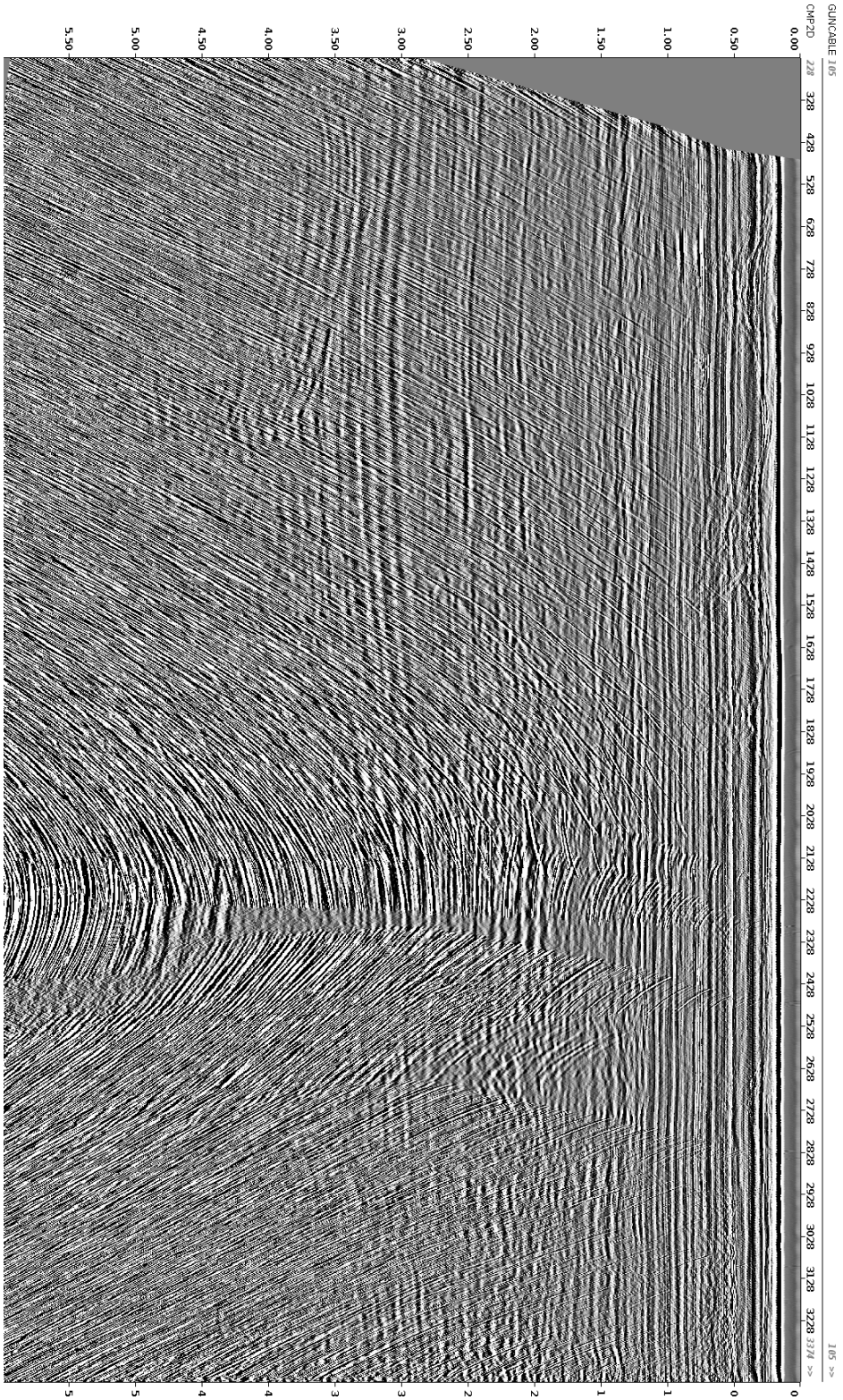


Figure A.18: *Noise contaminated stack*

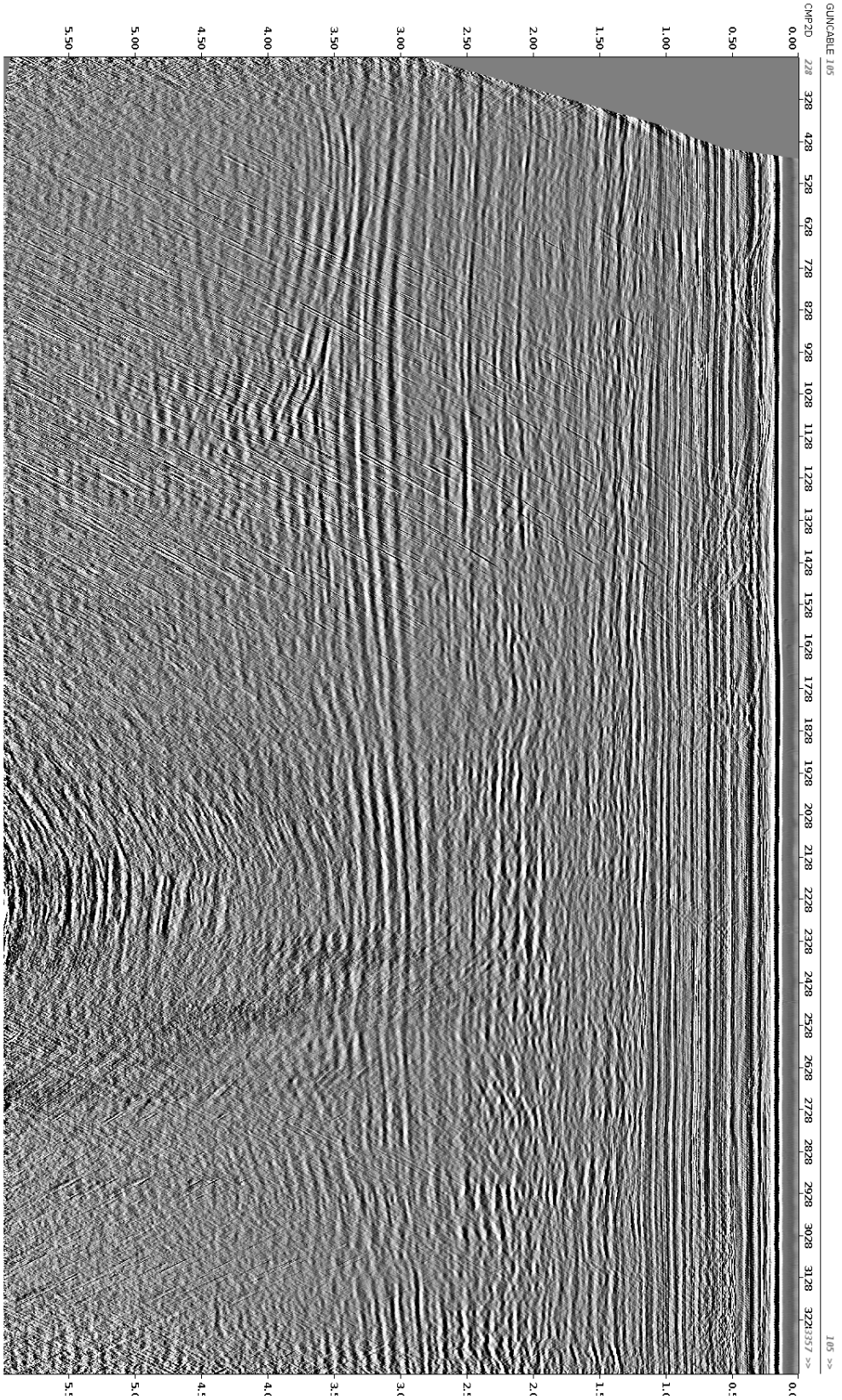


Figure A.19: *Denoised stack by U-NET1*

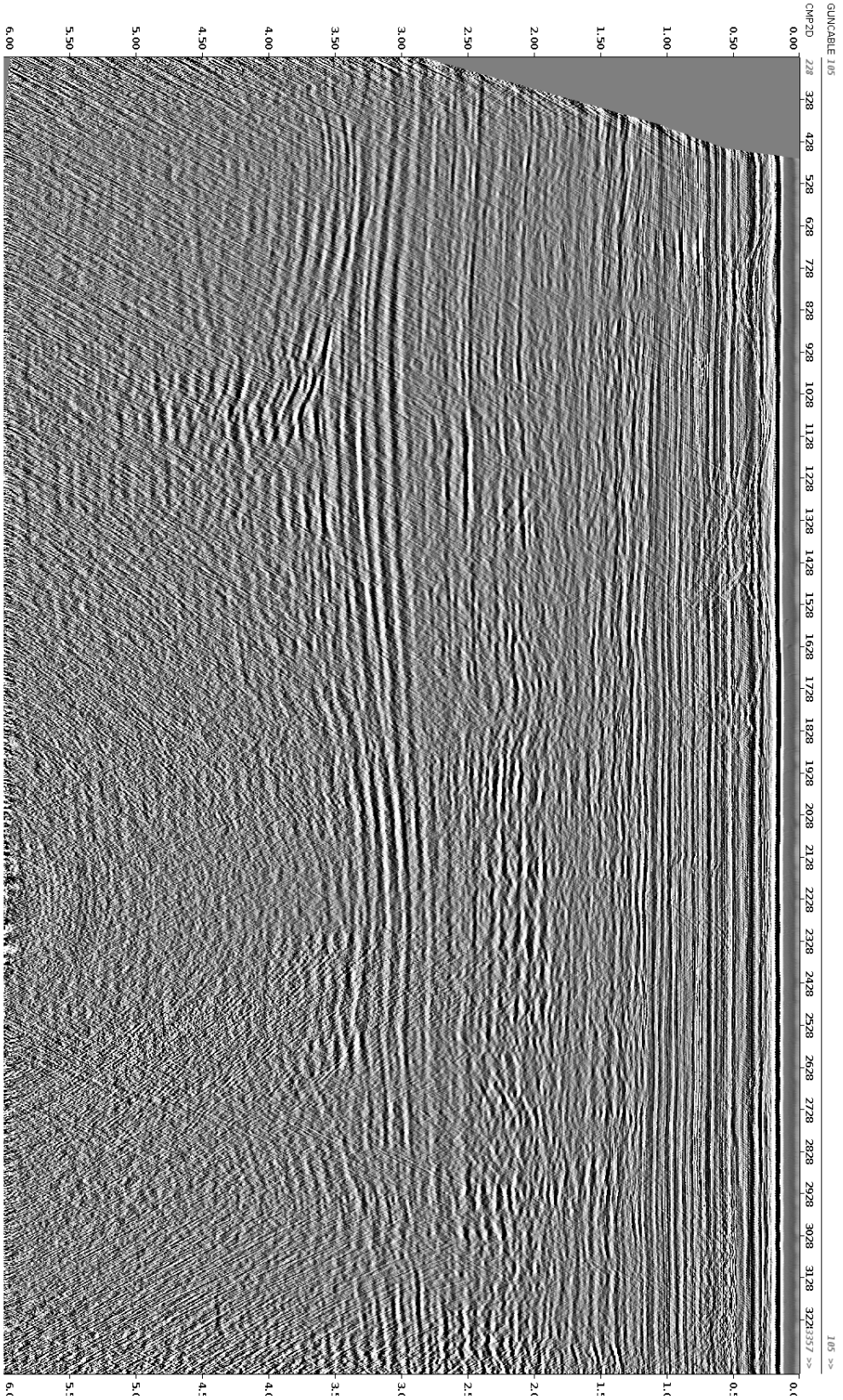


Figure A.20: *Denoised stack by industry standard SI-denoising*

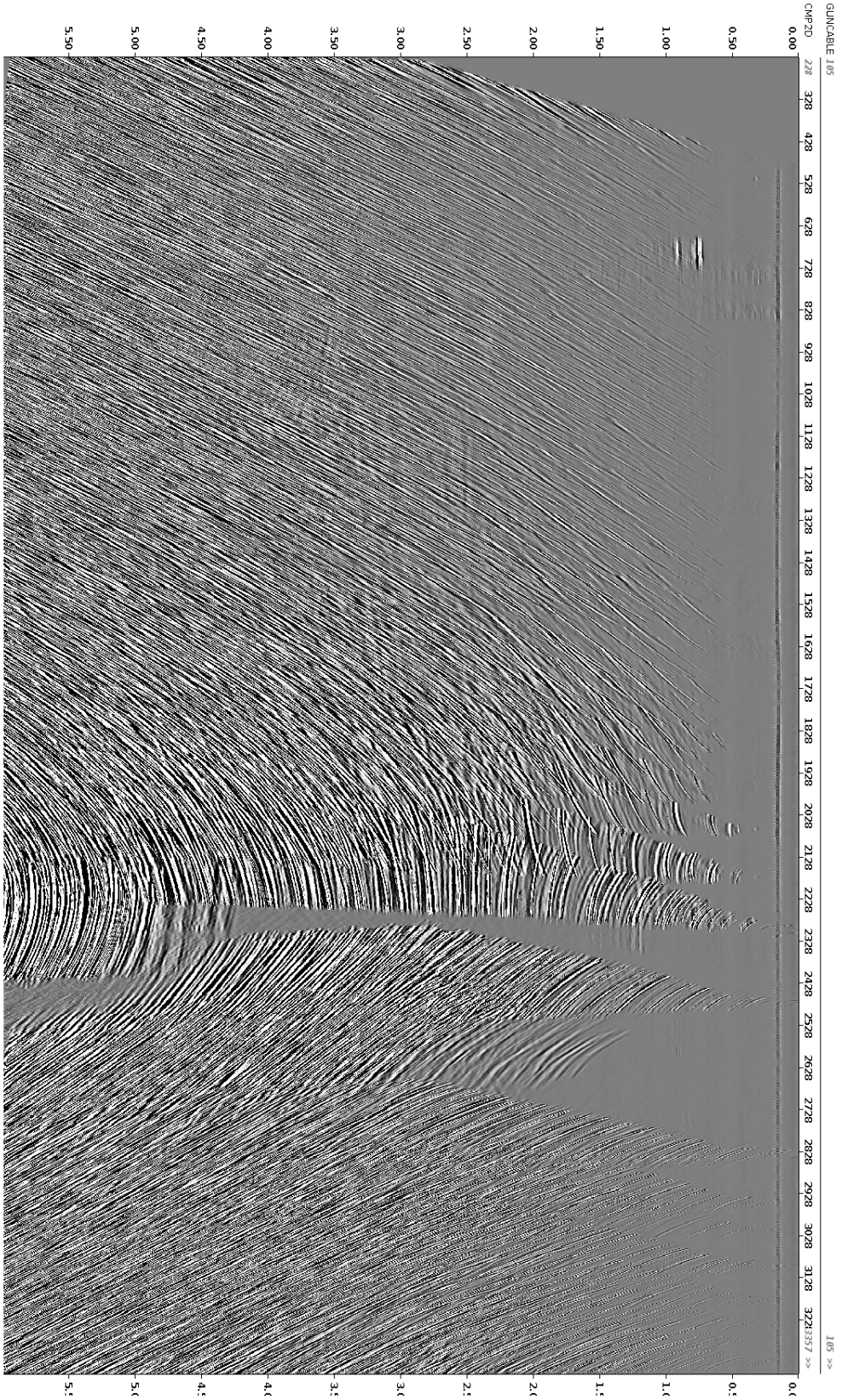


Figure A.21: *Difference U-NET1*

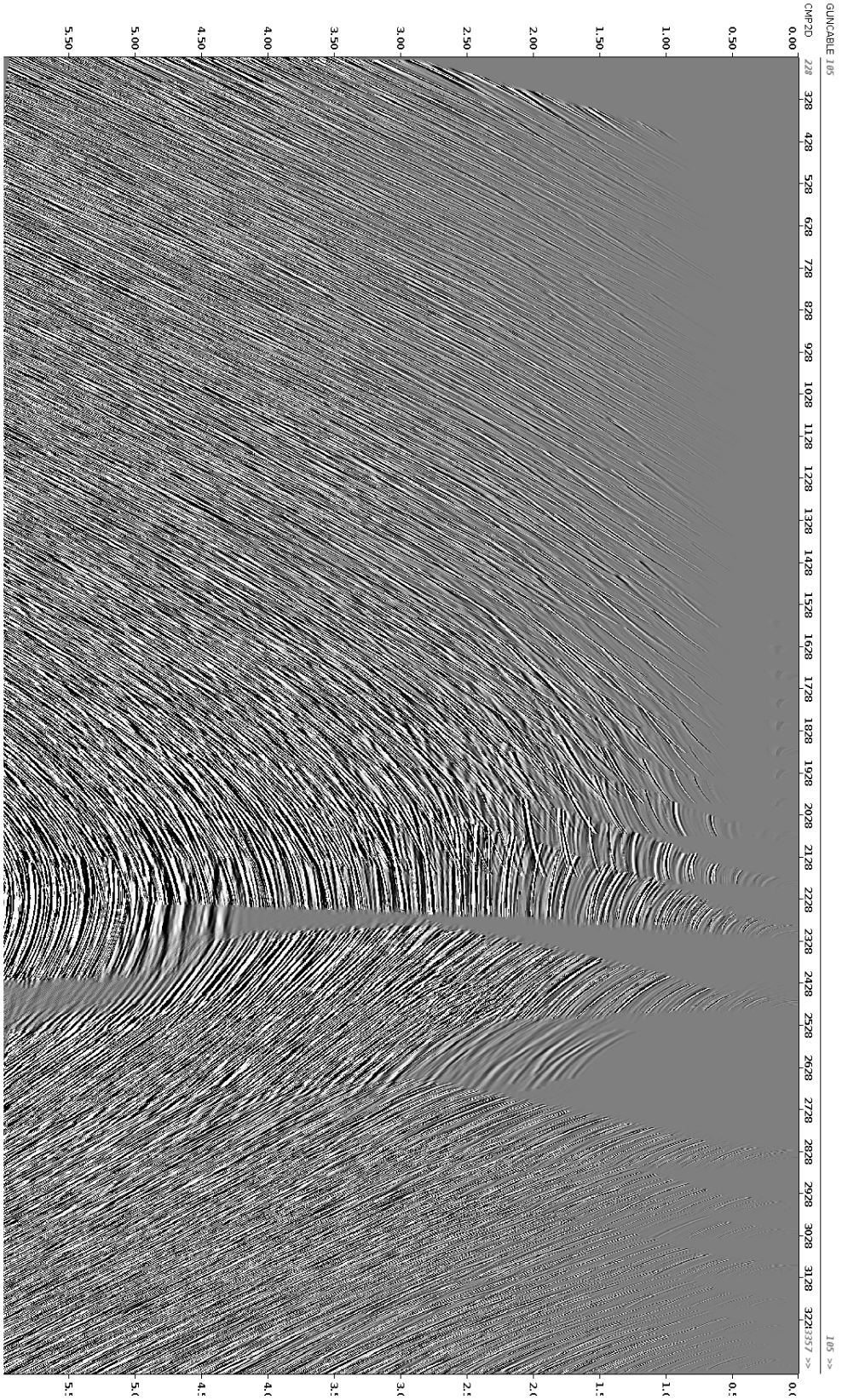


Figure A.22: *Difference industry standard SI-denoising*

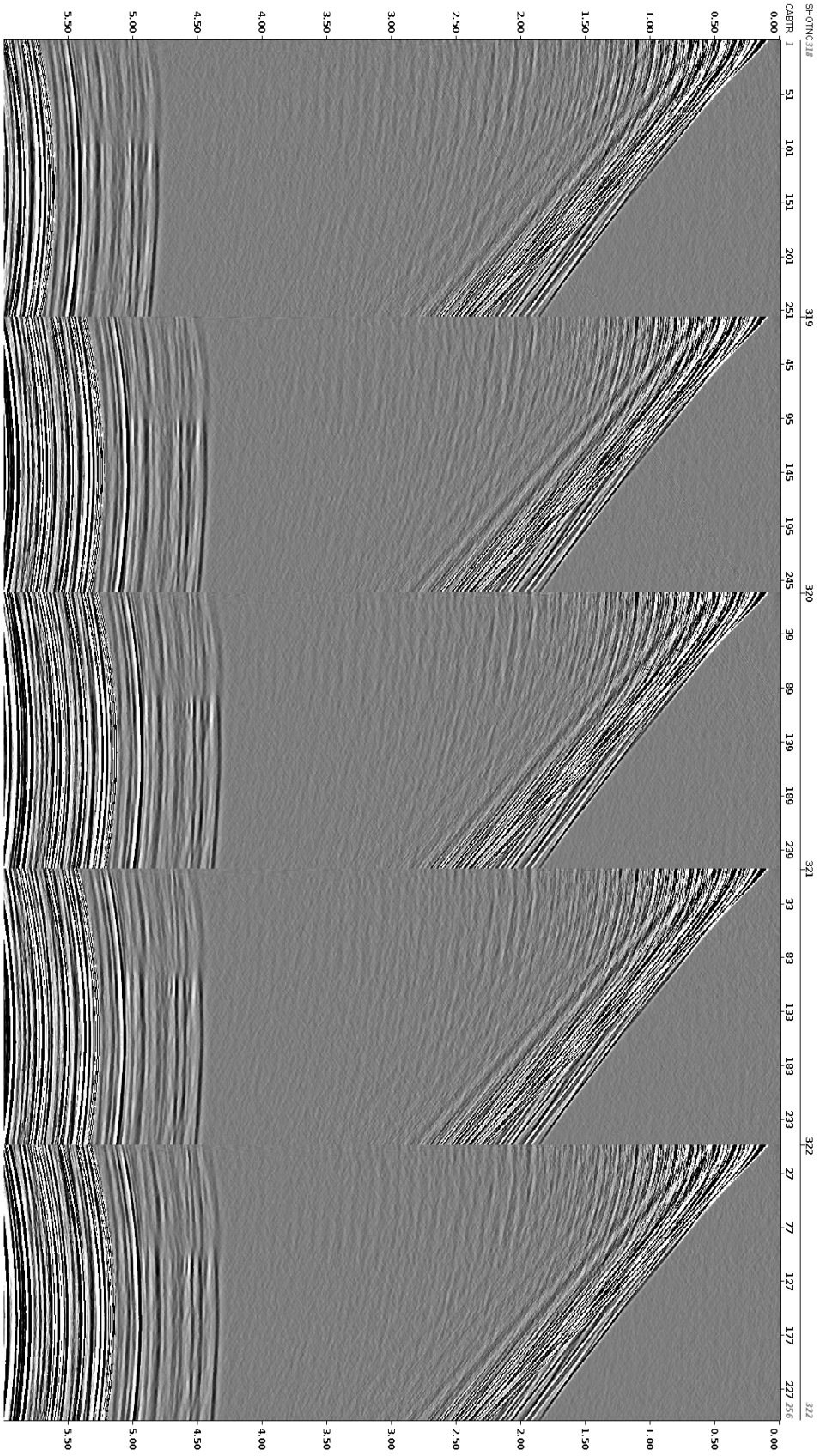


Figure A.23: *5 consecutive shotgathers highlighting area where the U-NET struggles to denoise.*

B | Appendix

This appendix presents the Expanded Abstract, based on results from this thesis, accepted for oral presentation at the EAGE Annual 2019.

2019 EAGE ANNUAL
81ST CONFERENCE + EXHIBITION
LONDON | THE UNITED KINGDOM

Leave this section empty

Using Convolutional Neural Networks for Denoising and Deblending of Marine Seismic Data

S. Slang[1,2*], J. Sun[1,2], T. Elboth[2], S. McDonald[2] and L. Gelius[1]

[1]University of Oslo, Norway, [2]CGG

Summary

Processing marine seismic data is computationally demanding and consists of multiple time-consuming steps. Neural network based processing can, in theory, significantly reduce processing time and has the potential to change the way seismic processing is done. In this paper we are using deep convolutional neural networks (CNNs) to remove seismic interference noise and to deblend seismic data. To train such networks, a significant amount of computational memory is needed since a single shot gather consists of more than 10^6 data samples. Preliminary results are promising both for denoising and deblending. However, we also observed that the results are affected by the signal-to-noise ratio (SnR). Moving to common channel domain is a way of breaking the coherency of the noise while also reducing the input volume size. This makes it easier for the network to distinguish between signal and noise. It also increases the efficiency of the GPU memory usage by enabling better utilization of multi core processing. Deblending in common channel domain with the use of a CNN yields relatively good results and is an improvement compared to shot domain.

Introduction

The recent availability of powerful GPUs and open source software have enabled artificial neural networks (ANNs) to be applied to a number of practical and industrial scale problems. The level of adoption of this technology within the field of O&G exploration is well illustrated by the number of abstracts related to ANNs that are submitted to the annual EAGE and SEG conferences. Since 2001, there have typically been one or two papers per year discussing ANNs. In 2018 the level rose significantly to between 50 and 100 papers.

In seismic processing, ANNs have the potential to be applied to many of the key processing steps (swell noise attenuation, seismic interference attenuation, deblending, deghosting etc.) which today involve significant testing time and computational power. Once trained, ANNs are computationally very light and potentially adaptable to varied datasets. Their use could therefore significantly save processing times and, in the long term, impact the whole business sector.

A natural first step in this direction is to look at various forms of seismic data denoising. This is not an entirely new concept, and is clearly inspired by work done on natural picture denoising, where we refer to Zhang et al. (2017) for a recent overview. However, this field is still immature and, as indicated by Xie et al. (2018), a lot of work is needed before ANNs can be effectively applied to seismic data denoising.

A common limitation in recent papers using ANNs for seismic data denoising (see e.g. Ma (2018), Baardman (2018), Si and Yang (2018), Li et al. (2018), Jin et al. (2018), and Zhang et al. (2018)) is that they only use synthetic data or noise on datasets with limited dynamic range and/or frequency content. As proof of concept, this has value. However, we have not yet seen convincing examples that compare against existing state-of-the-art denoising results.

In conventional processing, it is a common practice to sort and/or transform the seismic data into domains wherein it is easier to separate the noise from the desired signal. We have not yet seen this approach utilized in ANN denoising, and we believe that this could potentially improve results significantly.

This paper is structured as follows: in the theory section, we will introduce our network architecture and outline the design approach. We will then present two examples of denoising done on real marine seismic data, before pointing towards how we believe this work could be taken further.

Theory – CNNs for seismic denoising

A common type of layer in ANNs is the Fully Connected (FC) layer. They tend to give good results, but they are computationally heavy since they have one parameter for each sample in the input data. Seismic datasets tend to be large (~10^6 samples per shot gather), making the use of FC layers a challenging undertaking given the large memory requirements and computational demand. This leads us to another common ANN type, which is the Convolutional Neural Networks (CNNs).

CNNs are neural networks consisting of at least one convolutional layer. Convolutional layers differ from other types of layers in that they employ convolutions over subsets of the data, rather than a general matrix multiplication. According to Goodfellow et al. (2016), this makes CNNs well suited for 2D images where neighboring pixels are connected in a larger pattern. It should therefore be possible to denoise seismic data with localized and 'random' noise either in the shot domain or when sorted to, for example, the channel domain. We assume that CNNs will be able to handle seismic data, given its continuous nature and 2D structure. This will greatly reduce the computational cost and memory requirements compared to FC layers. Although it is much more efficient to use CNNs with respect to computational power, seismic images are large and still push the limits of hardware available today.

Seismic noise varies a lot in amplitude and might be orders of magnitude larger than the underlying signal, making it hard for the network to recreate the underlying signal structure. Given the difficulties raised by the large input volumes and the sometimes low SnR that can occur in recorded seismic data, it is understandable that previous attempts have been made with synthetic data or data subsets with limited size and dynamic range. When using synthetic data, the user has full control over the dataset. In this work, we apply CNNs to real life, full-scale marine seismic gathers and investigate how well this works for two types of commonly encountered seismic noise attenuation problems.

Example 1: Seismic interference noise attenuation

The first example investigated is the attenuation of seismic interference (SI) noise. This is dispersive coherent acoustic energy originating from other seismic crews operating nearby. The energy is mostly propagating in the water column, and is typically recorded with amplitudes similar to or larger than that of the seismic reflection signal. As such, it is common practice to try to attenuate this noise early on in the processing flow.

CNNs require both noisy images and clean images, which are regarded as ground truth, in image denoising. The network calculates the error between the denoised image and ground truth to update the weights. Our marine seismic training data consists of 800 records containing almost pure SI-noise recorded from different directions, and 482 (nearly) noise-free seismic shot gathers from the North Sea. It is a straightforward task to blend clean shots with varying SI-noise creating a dataset for the network to train on where ground truth is known.

The network architecture is based on common models used in image analysis with CNNs. It consists of convolutional layers with batch normalization and Rectified Linear Unit (ReLU) activation function. To handle the full seismic range, residual learning is applied, making sure all layers learn from the full frequency range in the original input image. The main difference compared to other attempts is that no downscaling is applied. The image is full-size throughout the network to reduce potential blurring and precision loss. This results in a large model requiring significant computational power and about 12GB GPU memory to train on a single image. The overall training process to achieve this level of denoising involved around 10^4 shot gathers, and took nearly two weeks on a modern GPU (6 GB x2). However, we note that once the network is trained, the actual denoising of a single shot gather is done in less than a second.

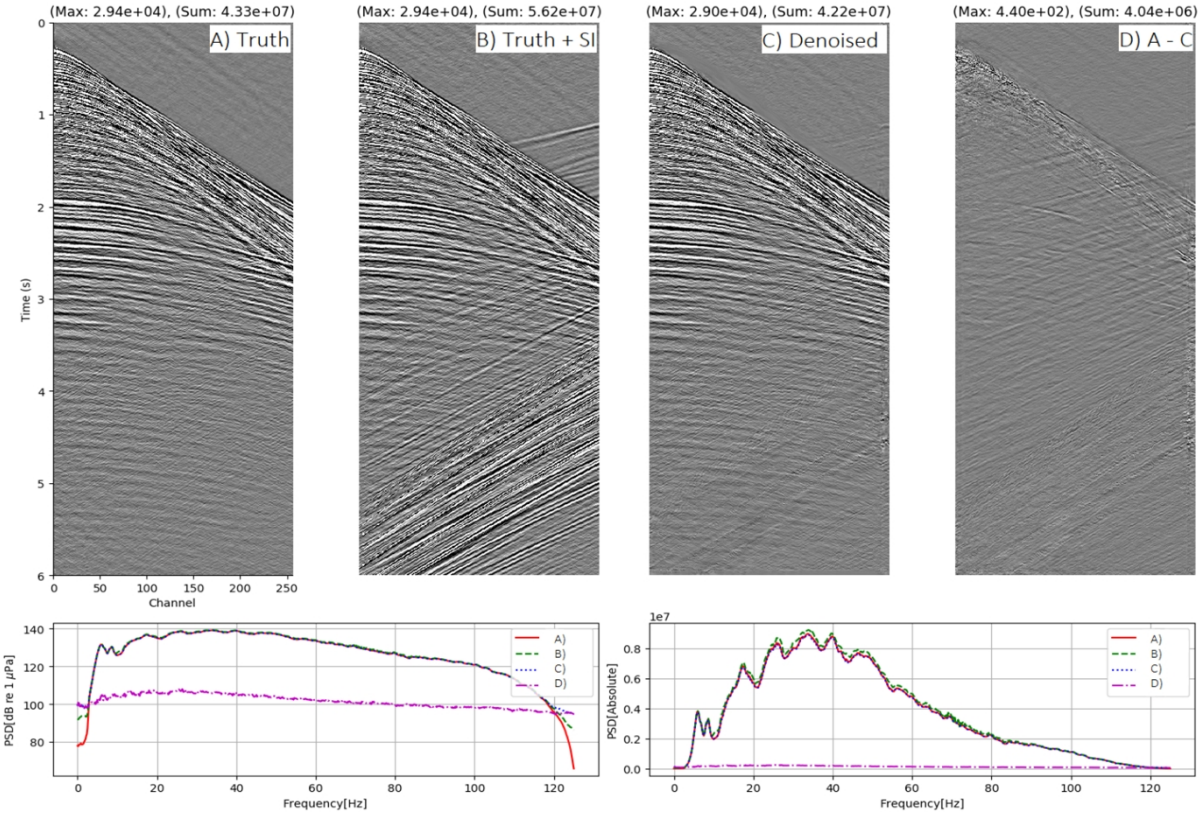


Figure 1: Figure illustrating the removal of SI noise from shot gathers where the different images from left to right are: A) clean shot, B) SI noise contaminated shot, C) denoised shot, D) difference between clean and denoised shot. The numbers above each shot are picture values showing the maximum and overall energy of the image. Fourier spectra (logarithmic and absolute) are appended at the bottom of the image to show the frequency content of each image.

Figure 1 illustrates that the network is removing SI rather well for cases where the SnR is relatively high. This is visible in the spectra showing that the network has a good recovery for all frequencies.

There is some leakage apparent in D), but it appears more dominant due to the high gain applied to the images. However, in cases with strong SI with very similar direction to the underlying reflection data, the results are not as good. Moving to common channel domain, where the noise is less coherent, is a way to overcome this problem. This allows us to use smaller images which still contain coherent signal but contain less coherent noise. Smaller images require less computing memory, which opens the possibility of using FC-layers. The downside of such a domain change is that 'real time' denoising during data acquisition is no longer possible since multiple shot records are needed to construct common channel images.

Example 2: Seismic data deblending

The second example is the deblending of N+1 shot data to extend the useful record length in the seismic data. Deblending (separation) of overlapping seismic records is important since it is a key technology enabling improved sampling and/or more efficient acquisition. The basic idea was probably first introduced almost 50 years ago by Barbier (1971), but wide scale industrial adoption has only been achieved in the last few years.

Our dataset consists of 1300 towed unblended marine split spread gathers. The shot gathers are artificially blended with the N+1 shot with a fixed delay of 1.2 second and a specially designed dither. Based on our study, we have found that it is difficult for the neural network to learn how to deblend seismic data in the shot domain. The events of two different shots in the blended data share similar characteristics in both amplitudes and dip, thus there is no clear characteristic features for the network to learn. Sorting the data into common channel domain, as mentioned in Example 1, gives better results. The N+1 shot in common channel exhibits randomness, while shot N has a continuous nature. Thus, deblending of overlapping seismic records in common channel domain is similar to removing random noise in image processing. The characteristics of CNNs, as mentioned in the theory section, is suitable for this case. The unblended data serves as ground truth for the network and is used to update the weights of the artificial neurons.

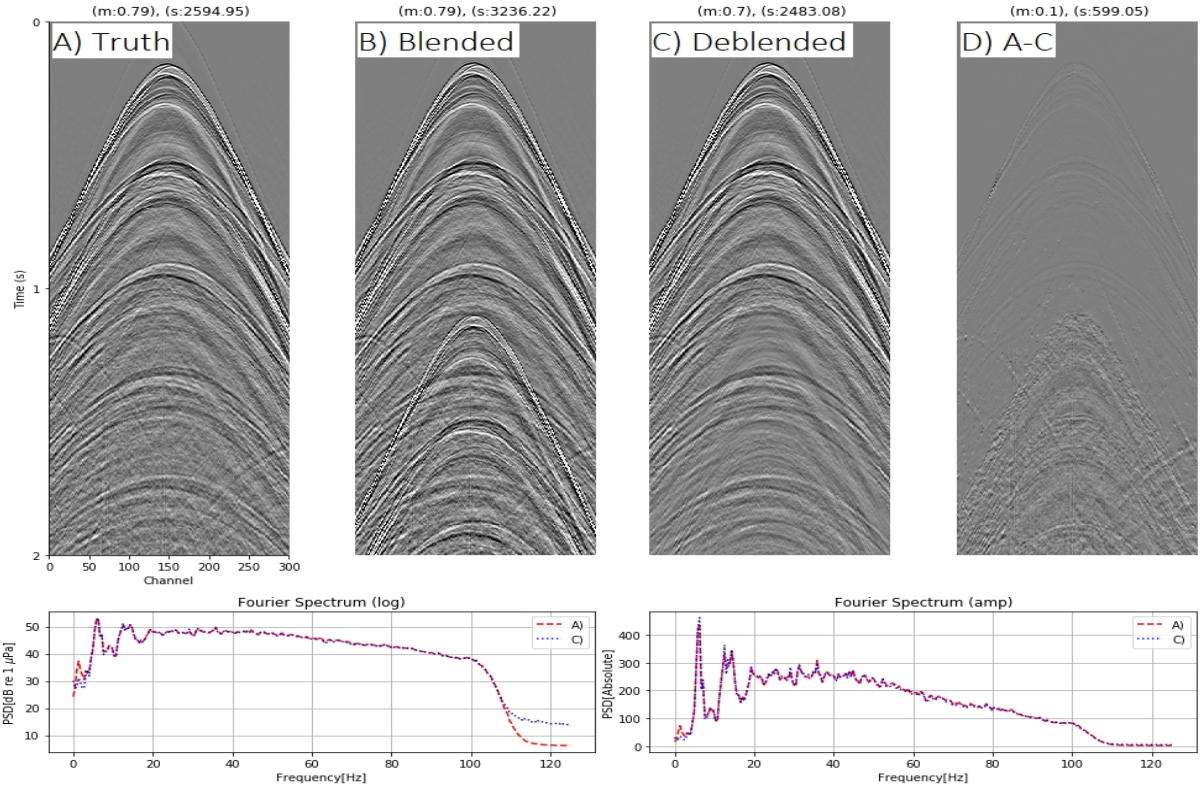


Figure 2: Figure illustrating the deblending result from shot gathers where noise amplitude is 80% of signal amplitude. The images from left to right are: A) unblended shot, B) blended shot, C) deblended shot, D) difference between unblended shot and deblended shot. The numbers above each shot are picture values showing the maximum and overall energy. Fourier spectra (logarithmic and absolute) are appended at the bottom of the image to show the frequency content of unblended shot and deblended shot.

Figure 2 illustrates that the network has the ability to learn deblending, and works well when we only introduce 80% of shot N+1, thus artificially enhancing the SnR. The recreated plot visible in C) has a quality that is close to current state-of-the art commercial processing, but would compare less favorably without artificial SnR enhancement. The training, validation, and test sets consist of 14400, 2700, 900 images respectively with 60 channels per image to reduce memory usage. This particular test took approximately 2.5 days to run, however, once the network was trained, the computational cost of deblending a single image was approximately the same as Example 1 – less than a second. All the processing is done in common channel domain. The records with no overlapping noise are almost perfectly recreated.

The network architecture is similar to Example 1, with a network consisting of convolutional layers with applied batch normalization. The largest difference is the usage of Leaky ReLU activation function to reduce the risk of "dead" neurons. Compared to common CNNs, both our mentioned network models differ in terms of static image size and thus no max pooling. This reduces the risk of blurring and loss of geological data. Even with a robust network, deblending in common channel domain remains a challenging task. This is because the SnR tends to always be low over a large area compared to other types of noise removal. Although this issue may cause some residual noise in the output image, the network is removing a significant amount of the N+1 shot and keeping detailed underlying signal intact. As mentioned in Example 1, there are multiple ways to potentially enhance the result. One way could be preprocessing the data to improve the SnR or moving the processing to a sparser domain. These novel approaches will be the focus of our future work.

Conclusions

Convolutional neural networks show promising results for interference noise attenuation and N+1 deblending of marine seismic data. When applied alone, the results are below the level achieved by state-of-the-art commercial processing. However, we have shown that applying machine learning to seismic data processing can still produce encouraging results and is an approach worth exploring further. For the problems shown, sorting the data into common channel domain seems to give better results than shot domain, due to the random nature of the noise in said domain. The higher the signal to noise ratio, the better the results become. We finally mention the importance of having high quality training datasets. The ground truth should ideally be without any noise. This is difficult to achieve when we work with real data, which inevitably will have some noise contamination.

Acknowledgements

The authors want to thank CGG MCNV for providing the field data used in this paper.

References

Baardman, R [2018] Classification And Suppression Of Blending Noise Using CNN. First EAGE/PESGB Workshop Machine Learning 2018

Barbier, M. [1971] Seismic Exploration. US patent 3956730A

Goodfellow, I., Bengio, Y. and Courville, A. [2016] *Deep Learning*. MIT: MIT Press. http://www.deeplearningbook.org

Jin, Y., Wu, X., Chen, J., Han, Z., and Hu, W. [2018] Seismic data denoising by deep-residual networks. SEG Technical Program Expanded Abstracts 2018: pp. 4593-4597.

Li, H., Yang, W. and Yong, X. [2018] Deep learning for ground-roll noise attenuation. SEG Technical Program Expanded Abstracts 2018: pp. 1981-1985.

Liu, D., Wang, W., Chen, W., Wang, X., Zhou, Y. and Shi, Z. [2018] Random-noise suppression in seismic data: What can deep learning do?. SEG Technical Program Expanded Abstracts 2018: pp. 2016-2020.

Mikhailiuk, A. and Faul, A. [2018] Deep Learning Applied to Seismic Data Interpolation. 80th EAGE Conference and Exhibitions 2018.

Si, X. and Yuan, Y. [2018] Random noise attenuation based on residual learning of deep convolutional neural network. SEG Technical Program Expanded Abstracts 2018: pp. 1986-1990.

Xie, P., Boelle, J. and Puntous, H. [2018] Generative-adversarial network-based fast-noise removal on land-seismic data. SEG Technical Program Expanded Abstracts 2018: pp. 2171-2175.

Ma, J. [2018] Deep learning for attenuating random and coherence noise simultaneously. 80th Annual International Conference and Exhibition, EAGE, Expanded abstracts

Zhang, Y., Lin, H., and Li, Y. [2018] Noise attenuation for seismic image using a deep-residual learning. SEG Technical Program Expanded Abstracts 2018: pp. 2176-2180.