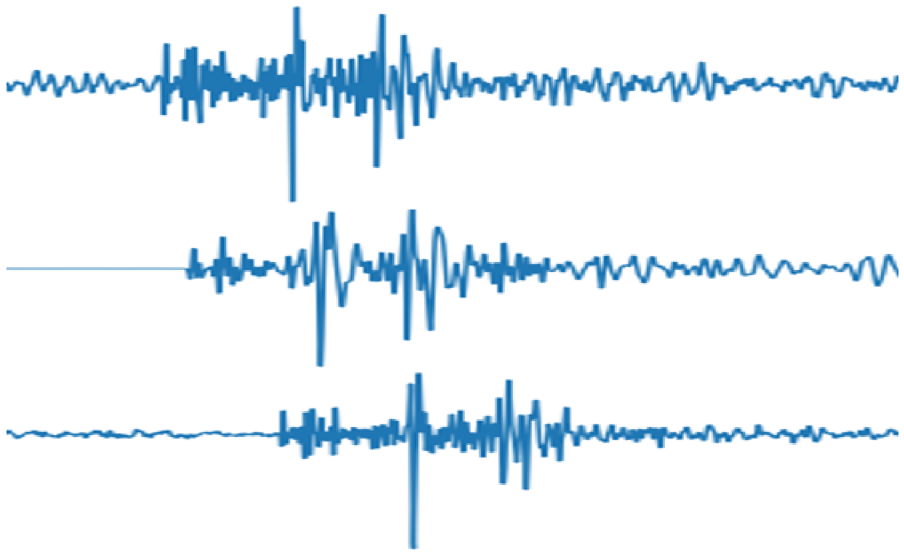
Earthquake Detection in Zeerijp

A Study on the Usage of Template Matching and Neural Networks for Detection of Small Earthquakes in Zeerijp



by
Julia van Deventer - 4393929

in partial fulfilment of the requirements for the degree of
Master of Science in Applied Earth Sciences

Faculty of Civil Engineering & Geosciences,
Delft University of Technology

Contents

Abstract

In this thesis, we discuss two pattern recognition techniques, template matching, and neural networks. We discuss how these techniques have been used for the development of two earthquake detection algorithms. The first algorithm is based on template matching and the second is based on deep learning. The algorithms are designed for the detection of small <0.5M events in the subsurface of Zeerijp, Groningen. These two algorithms have been compared to assess their earthquake detectability and practicality. The systems have been compared using field data from Zeerijp. The algorithm based on deep learning (a neural network) produced too many false positives considering the amount of seismic data we would like to use it for. The algorithm based on template matching did not produce any false positives during testing. The template matching system has been fed six months of continuous seismic data from Zeerijp. This resulted in the detection of at least 22 new events.

Acknowledgements

I am not a sentimental person but for this occasion, I am willing to try. With this thesis, I also conclude my time as a student at TU Delft. Which for me is bittersweet, I am happy that I can move on to something new, but on the other hand, it's a definite end to an era that makes me feel melancholic.

I would like to thank my old student house. I don't think I would have survived the combination of corona lockdown and thesis without some social contact. I want to thank all of my friends which have graduated around the same time as me. It was nice to not have to study alone. Special thanks to the Mijnbouwkunidge Vereniging for providing me with free tea and hot chocolate. I would also like to thank my parents for making all of this possible.

From a more academic point of view, I want to thank Nicolas Vinard for helping me with the forward modelling of earthquakes. I want to thank Guy Drijkoningen for providing feedback at the beginning of my research and for being on my committee. I also want to thank Kees Weemstra for providing feedback on my green light meeting and for being on my commitee. Of course, I would also like to thank my direct supervisors; Femke Vossepoel, Elmer Ruigrok and Hamed Diab Montero for their guidance and for motivating me during the process.

All in all, I have had fun during my studies and during my research but I am also very glad that with this thesis I can conclude this chapter of my life.

1 Introduction

The Netherlands is home to the largest gas field in Europe. The field is located in the province of Groningen and has been in production since 1963 [1]. Since 1991 induced seismicity is observed in Groningen. Until 2003 seismicity was stable and earthquakes had magnitudes around 1.5M. After a production increase in 2003 seismic activity increased and multiple earthquakes with magnitudes >3M have been registered [2]. Even though the earthquakes are relatively weak (maximum 3.6M) they have caused significant structural damages to buildings given that their designs did not consider seismicity. It is still unclear how many houses have been affected. Estimates suggest that at least a hundred thousand homes have been affected [3]. Figure 1.1 shows the location of the gas fields including a buffer zone (grey and yellow) and the areas affected by the two largest earthquakes (green and pink circles).

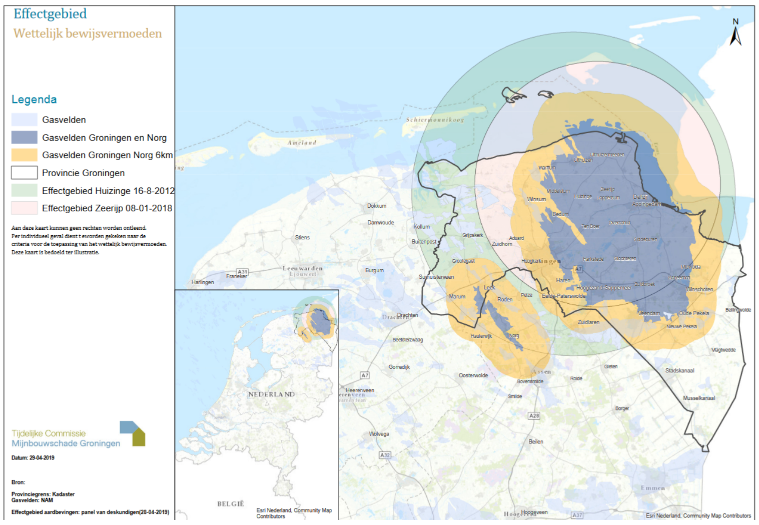

Effectgebied
Wettelijk bewijsvermoeden
Legenda
Gasvelden
Gasvelden Groningen en Norg
Gasvelden Groningen Norg 6km
Provincie Groningen
Effectgebied Huizinge 16-8-2012
Effectgebied Zeerijp 08-01-2018
Tijdelijke Commissie Mijnbouwschade Groningen
Datum: 29-04-2019
Bron:
Provinciegrens: Kadaster
Gasvelden: NAM
NEDERLAND
BELGIË
Esri Nederland, Community Map Contributors

Figure 1.1: Overview of gasfields and area affected by induced seismicity [4]

Because of the increase in seismicity and conflicts about damage payments the Dutch government has decided to slow down and eventually stop production in 2022. It is expected that after gas production has stopped induced seismicity will keep on occurring for some time.

By monitoring the occurrence and characteristics of earthquakes in Groningen the seismic hazard can be estimated accurately as described by Dost et al. [2]. Every year a Seismic Hazard and Risk Assessment (SHRA) on Groningen is published. This document quantifies the seismic risks, analyzes which risks are acceptable and what mitigation strategy should be used. In the case of Groningen, the SHRA determines which houses need to be reinforced and what reservoir pressures should be kept to keep risks at an acceptable level. The latest SHRA of Groningen has been performed by the NAM [5](the company which produces the field).

1.1. Earthquake Detection

To monitor the seismicity the KNMI (the Dutch institution responsible for monitoring seismicity) has set up a dense network of geophones and accelerometers. The network has been expanded since the occurrence of a 3.6M earthquake in 2012. The area is quite noisy due to waves hitting the coastline as well as due to traffic. Therefore the geophones are installed in boreholes to increase signal-to-noise ratios. Figure 1.2 shows an overview of all geophone stations and accelerometers. With this set up all earthquakes above a magnitude of 0.5M are detected. The current detection algorithm (the STA/LTA algorithm) uses the amplitude of the incoming seismic signals to detect earthquakes. Small events (<0.5M) often have the same amplitude as noise and are often missed by this system.

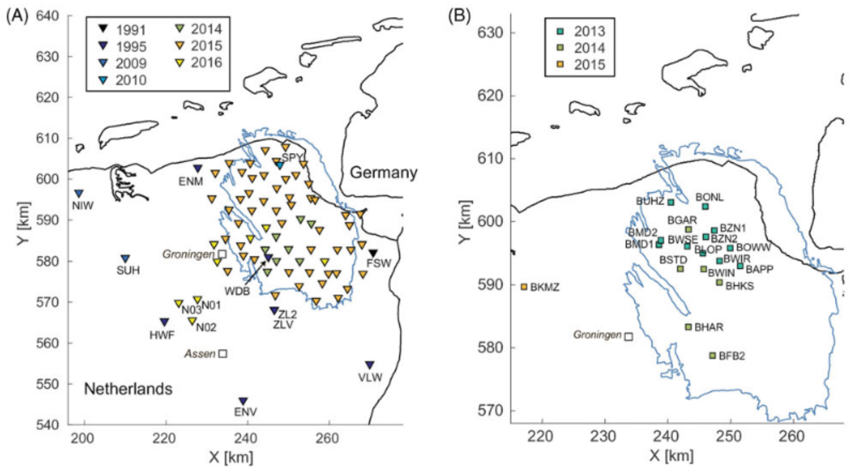


Figure 1.2: Overview of locations seismic stations (A) and location of accelerometers (B) [2]

1.1.1. Detecting Small Earthquakes

Small undetected earthquakes can not be felt at the surface but do contain valuable information. Monitoring the occurrence of even the tiniest earthquakes indicates how seismically active the region is. Low-magnitude events occur significantly more often than high ones. Being able to detect these small earthquakes enables us to monitor the seismicity more accurately. In this way, we can see early on what effect a decrease or increase in production has on the seismicity. Therefore it would be of great value to develop and implement techniques that can detect these low-magnitude events.

1.2. This Thesis

In this thesis, we explore if and how we can detect these small events. The research focuses on an area near the city of Zeerijp (Figure 1.3), one of the seismically most active zones in Groningen.

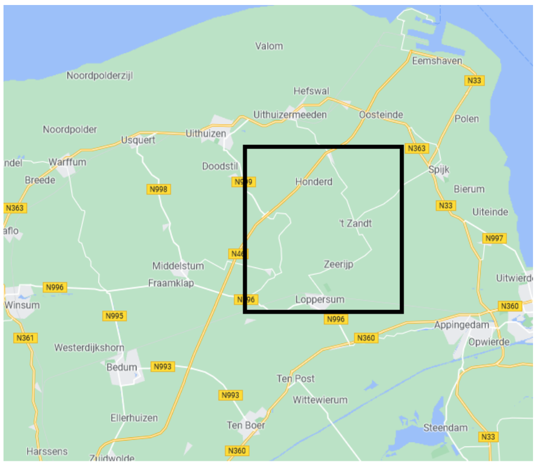


Figure 1.3: Our study area, with an area of 56km^2

The goal is to develop an algorithm that can detect low-magnitude events. Instead of focussing on optimizing one method, it has been chosen to compare two techniques; template matching and neural networks. Template matching and neural networks are both pattern recognition techniques. The research question is
Is it possible to create a practical detection system for low magnitude events(< 0.5M) using template matching or a neural network for the Zeerijp area?
This thesis also looks into how the methods compare. The subquestion is :
How do the developed algorithms compare when used for low magnitude events recorded in the Zeerijp area?
In the end, the best method is used on six months of seismic recordings from Zeerijp to test the algorithm's earthquake detection capability.

1.2.1. Why Template Matching and Neural Networks?

Both template matching and neural networks are promising because they use pattern recognition to make a detection. These techniques don't look at one feature of the seismic signal (like amplitude) but at the complete waveform. Template matching uses previously recorded events and classifies similar waveforms as earthquakes. Neural networks are trained with examples of earthquakes and with examples of seismic noise. By training the network it 'learns' the specifics of an earthquake and classifies signals that match its concept of an event as such.
Jagt et al. [6] have researched clusters of similar earthquakes (using cross-correlation) for events recorded between 2010-2014. The Zeerijp area corresponds to the location of Cluster 2 as shown in Figure 1.4. This indicates that several earthquakes have similar waveforms, making the area suitable for pattern recognition techniques.

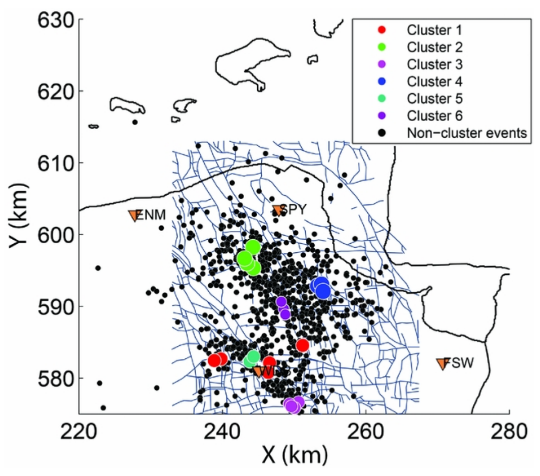


Figure 1.4: Overwiew of different earthquake clusters in Groningen. Our study area is the square given by y 594-602km and x 242-249km [6]

1.3. Similar Research and State of the Art

Template matching is a known technique for earthquake detection. It has been proven an effective tool for detecting and monitoring reoccurring low-magnitude events. Gibbons and Ringdal [7] show how template matching can outperform the LTA/STA algorithm for detecting such events. Vasterling et al. [8] and Goertz-Allmann et al. [9] demonstrate how template matching is used for detection and monitoring of induced seismicity in geothermal and carbon capture and storage projects. Dodge and Harris [10] show how a combination of template matching, cross-correlation, and the STA/LTA algorithm is used to detect and locate earthquakes worldwide. Cross-correlation (the mathematical operation on which template matching is based), has been used in the Groningen region to relocate and identify earthquake clusters based on waveform similarity by Jagt et al. [6]. However, template matching has not yet been used for the detection of small earthquakes.
Methods based on artificial intelligence such as neural networks are a newer technique. Recent studies show that neural networks can be used as an effective earthquake detection and location tool. Zhang et al. [11] discuss how a fully convolutional neural network can be used for accurate epicentre location of small >0.5M earthquakes, for a large area in Oklahoma. Perol et al. [12] have developed a 1D convolutional neural network for earthquake detection and location, for a small area in Oklahoma. The research presented in this thesis shows similarities to the work done by Perol et al. [12]. They also compare their neural network, ConvNetQuake, to template matching. A modified version of ConvNetqauke is one of the networks which earthquake detection potential is investigated in this thesis.
The networks found in Zhang et al. [11] and Perol et al. [12] are both trained with real earthquake data. Even though in this thesis, we also use neural networks, our networks are fully trained on synthetic earthquake data. In addition, we try to detect events with worse signal-to-noise ratios and of lower magnitude. The comparison between template matching and neural networks presented in this thesis differs from Perol et al. [12], in that we used real earthquake data for the comparison. Perol et al. [12] compare the 2 methods based on experiments done with synthetic data.
If we manage to detect any previously undetected earthquakes with either template matching or a neural network these events would be new additions to the earthquake catalogue. In this way, we would contribute new information on the seismicity in Groningen.

1.4. Thesis Outline

We start by discussing the current earthquake detection system and most relevant features of seismicity in the region in chapter 2. Next, we discuss the theory and development of the template matching algorithm in chapter 3 and subsection 3.0.2. Followed by 2 similar chapters on Neural networks, chapter 4 and chapter 5. In chapter 6, a comparison of the 2 systems is presented. The most suitable system is fed 5 months of seismic recordings, the results and set-up are presented chapter 7. The results are followed by the discussion and conclusion (chapter 8 and chapter 9).

2 Induced Seismicity

In this chapter, the current detection system is discussed. We discuss both the setup and the STA/LTA algorithm. We present a Gutenberg-Richter analysis to get an idea of the number of missed earthquakes and their magnitudes. We also show how similar earthquakes in the Zeerijp area are, and whether there are unique events.

2.1. Earthquake Monitoring Network Groningen

This section covers how earthquakes are monitored in the Zeerijp area. The monitoring network in Zeerijp is part of the G network set up by the KNMI between 2014 and 2016. This network consists out of 69 stations, with an average spacing of 4-5km. Figure 1.2 and Figure 2.1 show the monitoring network for the whole of Groningen and the Zeerijp region in particular. The three station in Zeerijp are G09,G14 and G18. These stations have been fully operational since 2015[2].

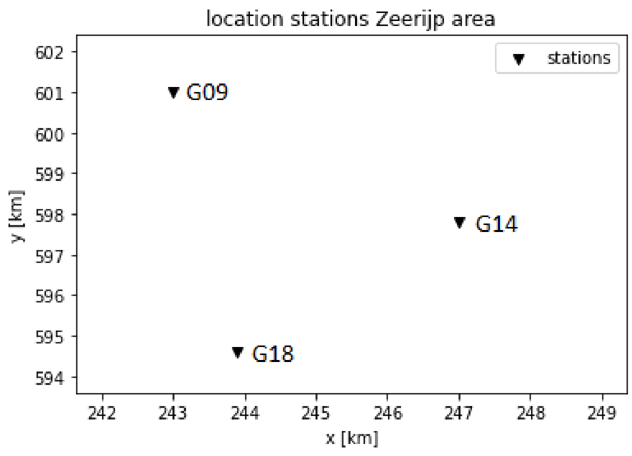


Figure 2.1: Overview of stations in the Zeerijp area

All G-stations are borehole stations. Each station is equipped with 4 geophones at different depths and an accelerometer at the surface. Figure 2.2 shows the set-up of the stations. Borehole stations are used in this region to increase signal-to-noise ratios. Multiple levels of receivers are used to discriminate between P and S arrivals and also for redundancy in case of an instrument failure [2]. The deepest receivers have the best signal-to-noise ratio. These are most suitable for small event detections, therefore only the recordings made by level 4 receivers are used. We refer to these receivers as G094, G144, and G184 to emphasize that we are using the 4th level of each station. Each geophone records seismograms for 3 directions (channels); 1 oriented vertically (HHZ), 2 oriented in the horizontal plane (HH1 and HH2).

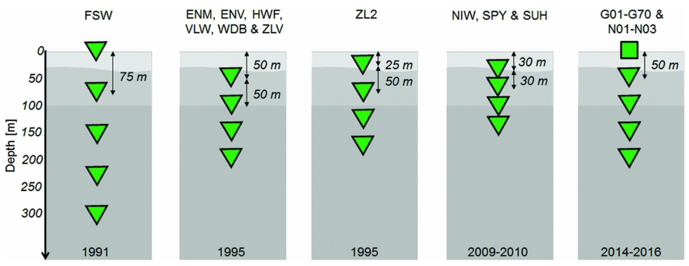


Figure 2.2: Borehole receiver set up Groningen earthquake monitoring system [2]

The current setup in Groningen enables the KNMI to locate all earthquakes up to 0.5M with an accuracy of 0.1-0.3km [2]. The current catalogue is complete up to 0.5M. The smallest earthquake recorded in the Zeerijp area has a magnitude of 0.12M.

2.1.1. STA/LTA Algorithm

In Groningen, the STA/LTA detection algorithm is used. This algorithm is the standard for weak motion seismology [13]. STA/LTA is an abbreviation of short-time-average through long-time-average. The STA/LTA algorithm continuously calculates the average absolute amplitude of the incoming seismic signal. It does this for two-time windows. For a short time window (STA) and for a long time window (LTA). The STA is sensitive to seismic activity while the LTA provides information about the temporal amplitude of local noise. Once the ratio between STA and LTA exceeds a threshold it triggers the system. When multiple seismic stations are triggered within a certain time frame the signal is declared an event and the signal is flagged as such. The signal is flagged as an event until the STA/LTA reaches the de-triggering threshold.
The problem with very small events is that the amplitudes of these events are so low that they often have similar amplitudes as noise. This is especially true for the Groningen area where signal-to-noise ratios are low due to the proximity of the coast and urban activity.

2.2. Estimation of Missed Events using Gutenberg-Richter

The Gutenberg-Richter equation is used to estimate the number of missed earthquakes in Zeerijp. The Gutenberg Richter equation gives the number of events above a certain magnitude that statistically should occur within a specified time frame. For the parameters found in this section, a time frame of a year is used.

2.2.1. Gutenberg-Richter equation

The Gutenberg-Richter equation is given by the equation below [13]:

$$N = 10^{a-bM} \tag{2.2.1}$$

Parameter a indicates how seismic active an area is. The total number of expected events above 0M is equal to 10^a. Parameter b indicates the proportion between the occurrence of small and large earthquakes.

2.2.2. Estimating the a and b Parameters

Information on all recorded events in Groningen is publicly available at www.knmi.nl/kennis-en-datacentrum/dataset/aardbevingscatalogus. For the Gutenberg-Richter analysis, we have used all events between 2015 and 2019 as can be found in Appendix A. From this data, a histogram has been constructed. By using a least-square fit we can estimate the a and b parameters.

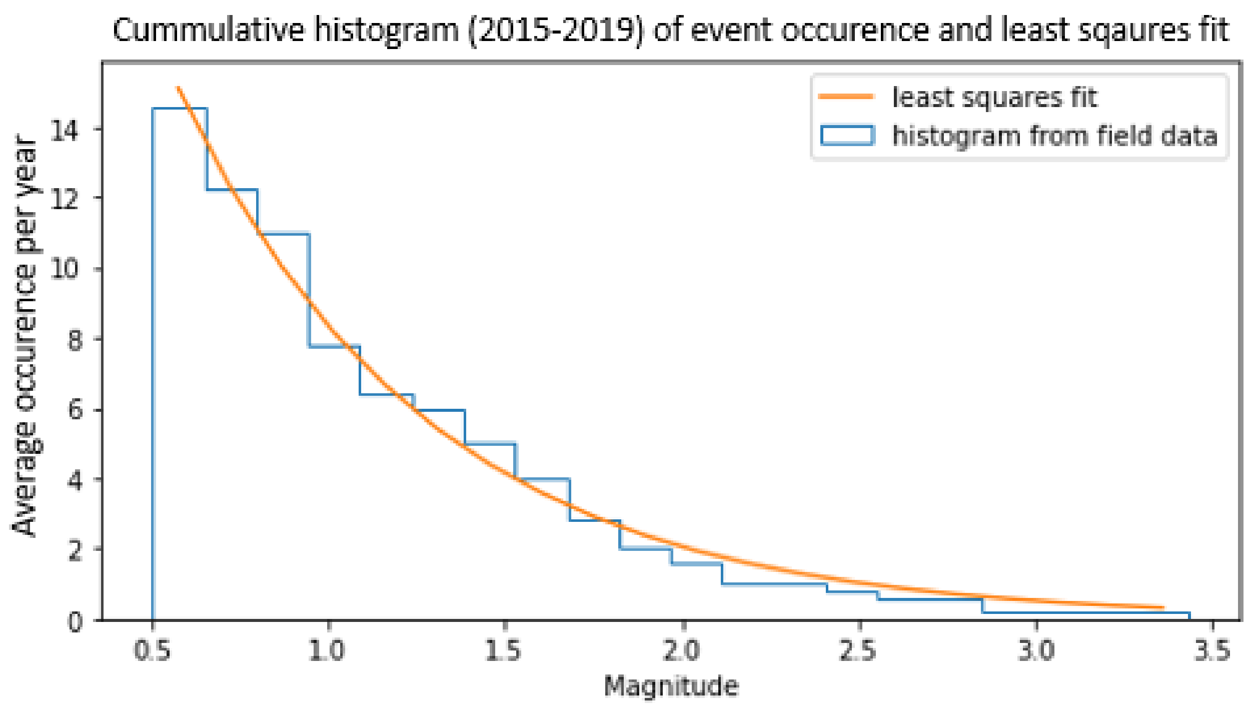


Figure 2.3: Histogram based on events recorded between 2015-2019 and their magnitude,the fit represents the Gutenberg-Richter relation

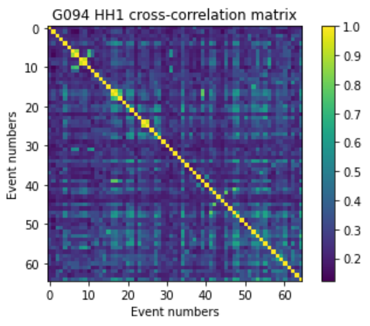
For parameters a and b we found values of 1.53 and 0.61. Based on these values we expect 138.38 events per year above a magnitude of -1M. Currently, 17.2 events above 0M are detected. This means that 121.6 events between -1 and 0.5M go unnoticed per year or 2.3 events per week. The table below shows how many events are expected to have gone unnoticed for different magnitude ranges.

	Undetected events expected per year	Undetected events expected per 6 months	Undetected events expected per week
0 - 0.5M	16.60	8.30	0.32
-0.5 - 0M	34.51	17.16	0.66
-1 - -0.5M	69.65	34.82	1.34

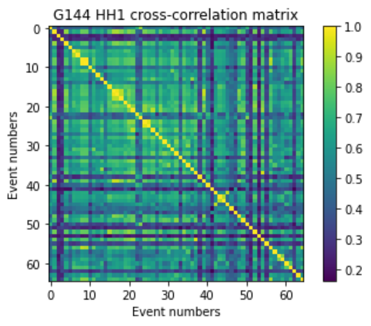
Table 2.1: Undetected events expected based on Gutenberg Richter for different magnitude ranges

2.3. Earthquake Similarity

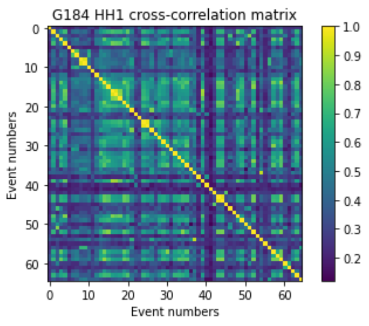
If we want to use a pattern recognition system we first need to establish there exists a recognizable pattern. Especially for template matching, the undetected earthquakes must be similar to previously recorded events. Since it can only detect patterns/wave-forms which are similar to its templates.
In chapter 1 we already discussed that the KNMI has established that a cluster is present using data from 2010 and 2014 before the network was updated(Figure 1.4).
We perform the same type of analysis as done by Jagt et al. [6] for all earthquakes recorded in Zeerijp between 2015-2019. In order to assess how similar the earthquakes in the region are, all earthquakes are cross-correlated with each other. This is done for all stations and all channels separately. The result of this operation is a symmetric cross-correlation matrix. In this matrix, every value corresponds to the cross-correlation coefficient (ccc) of two events[6]. See chapter 3 for a detailed explanation on how cross-correlation works. The length of the events are set to 10 seconds and are filtered with a 3-22Hz bandpass filter. Figure 2.4 shows the cross-correlation matrices of channel HH1 for all 3 stations. For this channel the cross-correlation coefficients are highest.



(a) Cross-correlation matrix G094 HH1



(b) Cross-correlation matrix G144 HH1



(c) Cross-correlation matrix G184 HH1

Figure 2.4: Cross-correlation matrices HH1

When we use a ccc of 0.6 we don't find any unique events. A unique event would be an event that wouldn't have a cross-correlation coefficient with at least one other event above the ccc threshold. Since we have found no unique events we can have established that the earthquakes in this region produce recognizable signals. Figure 2.5 shows that there are two main clusters of earthquakes present in the Zeerijp area. From this, we can conclude that there are two main source areas. Within these source areas, the events likely have the same focal mechanisms since we observe very high ccc's. When earthquakes have similar focal mechanisms (movement along the fault), they produce similar waveforms [13]. Undetected events are therefore most likely related to one of the clusters. This makes the area suitable for template matching. The question remains whether or not template matching will be able to recognize small earthquake signals when they are buried in noise. This is explored in chapter 6.

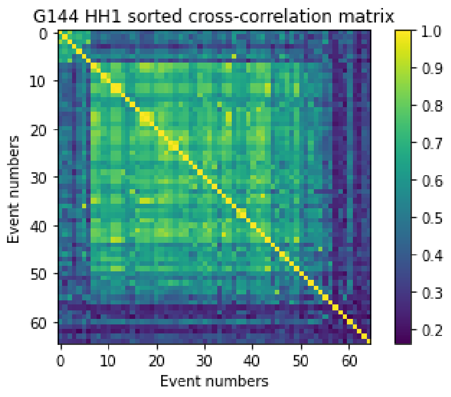


Figure 2.5: Cross-correlation matrix sorted for G144 HH1

3
Template Matching Theory

In this chapter, we discuss the theory behind template matching. Template matching is a technique used for pattern recognition. Template matching uses cross-correlation to find patterns similar to its templates. In the case of this thesis, we use template matching to find patterns in time-series data, but it is most frequently used for pattern recognition in images.

3.0.1. Cross-correlation

Cross-correlation is defined as the summation of the dot product of 2 functions, where one function 'slides' over the other. We illustrate this with functions $f(t)$ and $g(t)$. At the start $f(0)$ is multiplied with $g(0)$ afterward, $f(1)$ is multiplied with $g(1)$. This process is repeated for n steps. After n steps, all multiplications are summed together. Then $g(t)$ is shifted with time step τ and the process is repeated. The result of this operation is a function of τ. Equation 3.0.1 shows the definition of cross-correlation [13].

$$c(\tau) = \sum f(t)g(t+\tau) \tag{3.0.1}$$

The highest cross-correlation coefficient (ccc) is observed for the shift where the signals are most similar. Cross-correlation is often used for determining the time shift between time series. However, for template matching the time-shift is irrelevant. We want to find similar patterns in the data. When using Equation 3.0.1 the cross-correlation coefficient is dependent on the amplitudes of the correlated signals. For template matching, we want a measure of similarity which is universal. By introducing normalization, the function becomes independent of amplitudes. Equation 3.0.2 shows the normalized cross-correlation function[13].

$$c_n(\tau) = \frac{c(\tau)}{\sum f(t)^2 \sum g(t)^2} \tag{3.0.2}$$

Using Equation 3.0.2 the ccc can vary between 1 and -1. A ccc of 1 indicates that the signals are precisely the same and a ccc of -1 indicates perfect antithesis. The ccc is a measure of similarity. It indicates to what extent two signals go up and down in the same pattern, independent of amplitude.

3.0.2. Templates

Template matching works by cross-correlating templates with target data. The templates consist out of patterns one wants to locate in the target data. By setting a certain ccc as threshold, one can find the location of the similar pattern in the data stream. In the case of this thesis the templates are parts of event recordings and the target data is seismic data recorded in Zeerijp. chapterDevelopment of Template Matching for Earthquake Detection Template matching uses cross-correlation to find patterns similar to its templates. In the case of this thesis, we use high energy intervals of selected previously detected events as templates. The templates are cross-correlated with seismic data. Once an interval with a ccc above a threshold value is found, an event is detected. The intervals with a ccc under the threshold are classified as noise.
To develop template matching we first choose which events to use as templates. This is described in section 3.2.To increase the signal-to-noise ratio the data is filtered.section 3.3 explains how the filter setting is determined. Once the templates and the filter have been chosen. The templates are fine-tuned. The interval

for the templates is specified, the template length is optimized and the detection threshold value is determined. During fine-tuning, we test how well different settings perform on earthquake and noise data. All of this is described in section 3.4. In Appendix A the noise and events used in this section can be found. Please note that the unit of amplitude for all seismic recordings shown in this chapter should be m/s instead of m.

3.1. How Does Template Matching Make a Detection?

For most of this chapter (except for section 3.2), the seismic input data is presented in the form of screens. These screens are 10-20 second intervals of seismic data. Every event is recorded by three stations, and each station records three channels. Therefore every event is recorded on 9 screens. When a template has a ccc value above the threshold with one of the screens, an event has been detected. The more confident the template is about a detection the more screens it classifies as event. If the template is very confident it classifies all screens 9/9 as an event. If it's not confident it classifies 1/9 screens. If none of the screens is classified as an event, the time interval is classified as noise. **??** shows a visualization of this process with three screens. In practice, nine screens are used.

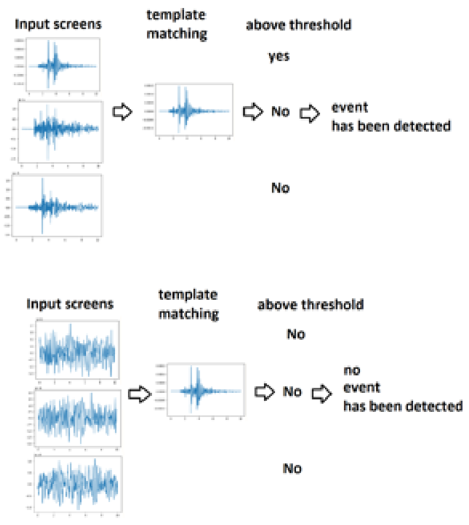


Figure 3.1: Visualisation of how template matching makes an event detection

In section 3.2 we use whole days of data instead of screens. In that case, the system returns indexes of intervals with a ccc above the threshold.

3.2. Determining Which Events to Use as Templates

Low magnitude events have bad signal-to-noise ratios. This can cause high ccc values with noise. In this section, we explore what magnitudes are suitable. Ten-second intervals of all events are cross-correlated with seismic recordings. The recordings are from the 7th, 8th, and 9th of January 2018. These recordings contain two event recordings (events 47 and 48). All events have been cross-correlated with these recordings. We have looked at the behaviour of events above 1M, between 0.5-1M, and below 0.5M. Events <1M generate too many false positives and are therefore not used as templates.

	Event detection	false positives
Templates >1M	2/2	4
Templates 0.5-1M	2/2	100000+
Templates <0.5M	2/2	1000000+

Table 3.1: Test templates of different M

3.3. Determining Filter

We are only interested in the frequencies which are dominant during events. The rest of the frequencies can be filtered out, increasing the signal-to-noise ratio. To determine the dominant frequency we have looked at all events >1M in the frequency domain. Events <1M are left out because they contain too much noise. The interval we look at corresponds to the 6-11 second interval of event 19. During this interval, the signal is strongest. Figure 3.2 shows the recording of event 19. To properly analyze the frequency spectrum of the events, a high-pass filter of 1.5Hz is used. These lower frequencies belong to noise, so filtering them out won't cause trouble.
From here on if we refer to recordings of events we simply refer to it by using its number followed by event. On the y axis, we find the amplitude of the recording which has as unit m/s. Seismic activity is measured by a sensor containing a magnet in a coil. Due to seismic activity, the magnet moves and creates a current. The strength of this current is linear to the velocity of the magnet relative to the coil, hence the unit of amplitude is m/s. The x-axis shows time, so the recording shows how amplitude changes over time.

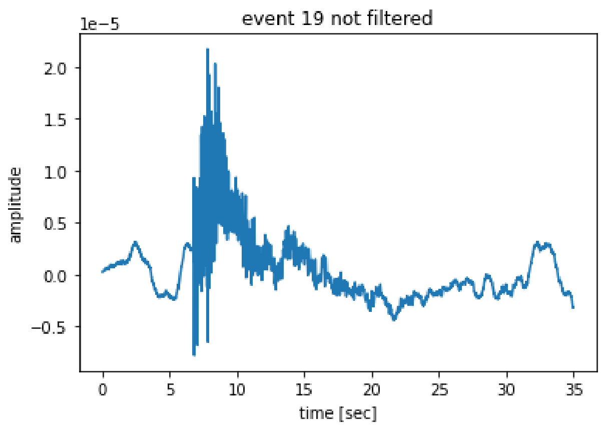


Figure 3.2: Event 19 G184 HHZ not filtered

The lowest dominant frequency is 3Hz (event 56)(Figure 3.3) and the highest dominant frequency is 22Hz (event 76) (Figure 3.5). Therefore a 3-22Hz bandpass filter is used for template matching.

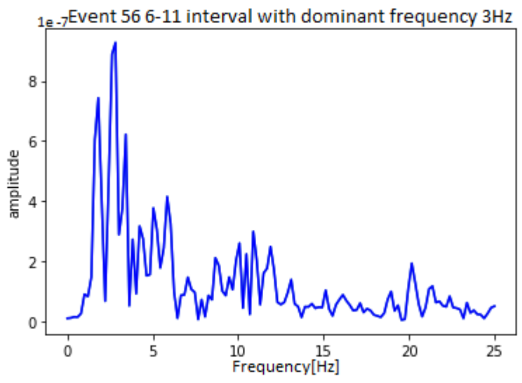


Figure 3.3: Frequency spectrum event 56

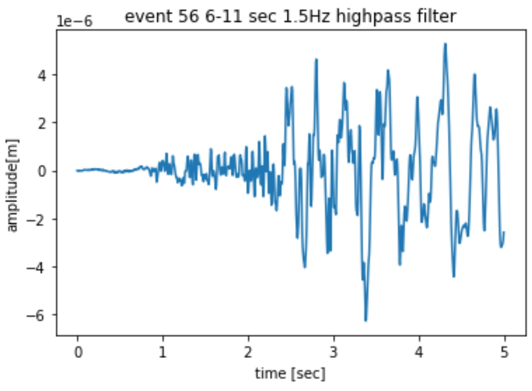


Figure 3.4: Event 56 interval filtered

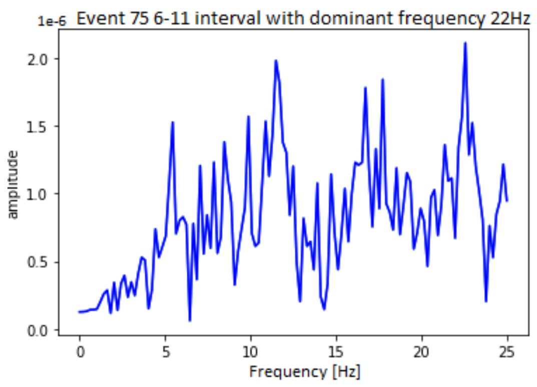


Figure 3.5: Frequency spectrum event 75

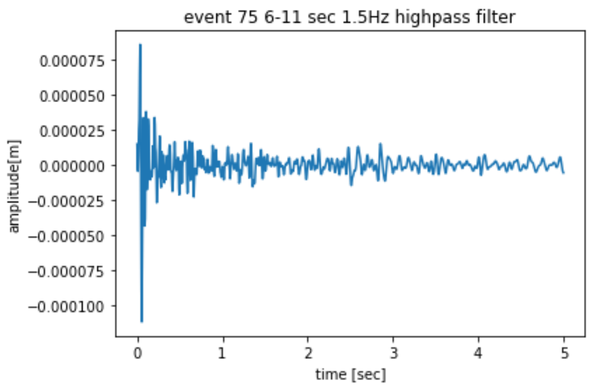


Figure 3.6: Event 75 interval filtered

3.4. Determining Template Length, Interval and Threshold

The template length, interval, and detection threshold values are determined with the same test. Several lengths and intervals are tested with events <1M and noise for different threshold values. Two intervals are tested. The first interval only contains the first part of an event. The second interval starts with a second of noise. The idea is that with introducing an extra second of noise, we make the template more random and reduce the chance of it being similar to noise. Figure 3.7 and Figure 3.8 show examples of the 2 different intervals .

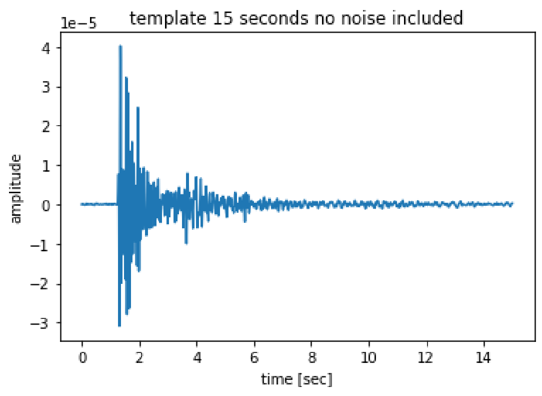


Figure 3.7: Example of a template with a length of 15 seconds without noise

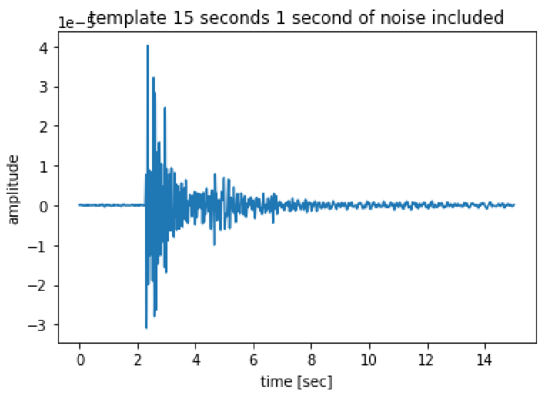


Figure 3.8: Example of a template with a length of 15 seconds including 1 second of noise included

The templates are cross-correlated with 18 events (162 screens) <1M. These events have not been used as templates. The templates are also cross-correlated with 700 instances of noise (6300 screens). We look at the number of correct detections for threshold values between 0.4 and 0.9. We are looking for the template interval and length combination which can detect all events while producing a minimum amount of false positives. For noise instead of the number of false-positive instances, we have chosen to show the number of false-positive screens. This is done to be able to see the difference in performance for the lower thresholds. If instances would be used here instead parameters settings would seem to perform the same but in reality, one produces significantly more false-positive screens than the other.
We only discuss the results of the settings which showed the most promise, the full results can be found in Appendix C. We observe that above a threshold of 0.6 not a single false-positive detection has been made by any of the tested settings. Therefore 0.6 is chosen as the threshold value. A template length of 12.5 seconds including 1 second of noise gave the best result averaged over all the thresholds. It detects most of the events and produces almost no false positives. It isn't the best noise classifier. That would be a template length of 15 seconds with one second of noise, but above a threshold of 0.6, both settings perform the same noise-wise. In Figure 3.9 a graph is seen plotting the data from the tables as percentages to better visualize the differences between the settings.

Templat 12.5 seconds with 1 second of noise included	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	17/18	13/18	3/18	**81%**
Screens of noise classified as events	204/6300	4/6300	0/6300	0/6300	0/6300	0/6300	**99%**

Table 3.2: Results using a templates of 12.5 seconds with one second of noise included

Template 15 seconds with 1 second of noise included	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	10/18	1/18	0/18	**60%**
Screens of noise classified as events	828/6300	11/6300	0/6300	0/6300	0/6300	0/6300	**99.6%**

Table 3.3: Results using a templates of 15 seconds with one second of noise included

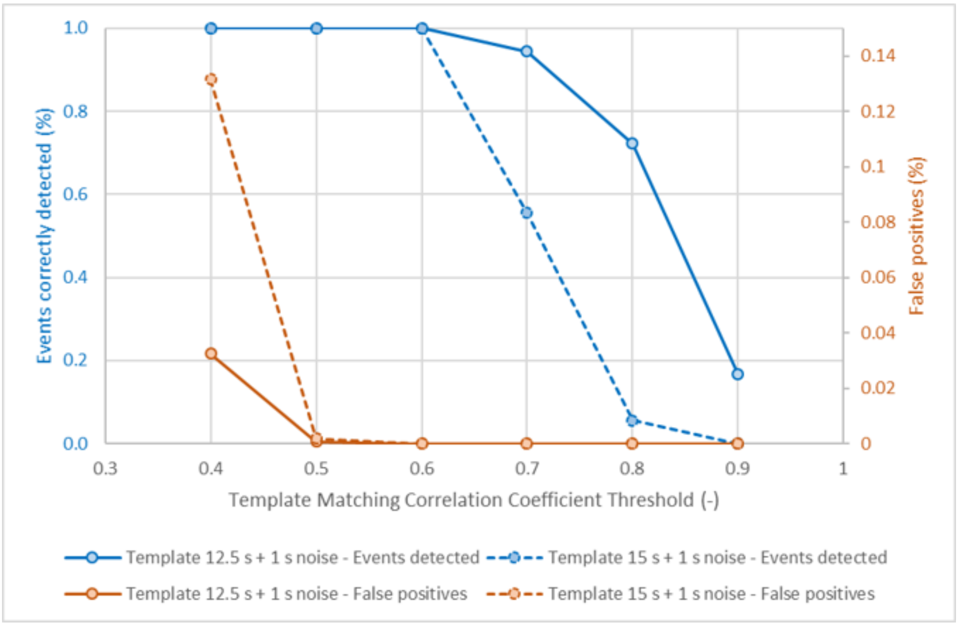


Figure 3.9: The two best template matching systems

4
Theory Neural Networks

In this chapter, the inner workings of neural networks are explained. We begin with the easiest type of network, the multi-layer perceptron network or Feedforward Neural Network (FNN). Later the workings of the convolutional neural network (CNN) are also discussed.

4.1. Introduction to Neural Networks

Artificial neural networks (from now on referred to as neural networks) are computing systems vaguely inspired by biological neural networks that make up brains. Computers struggle with tasks such as recognizing handwritten text which humans have little difficulties with. This is where neural networks come in.
Neural networks aim to 'learn' a computer to recognize certain ideas like cats, for example. Most people are familiar with the idea of a cat. Therefore we recognize cats in pictures. We can even recognize abstracted images of cats, like cartoons or paintings. By feeding neural network examples of both cats and none cats we can 'teach' the concept of a cat to a neural net. There is a neural network publicly available that is trained just for this purpose. On www.isthisacat.com you can upload images that a neural network then classifies.

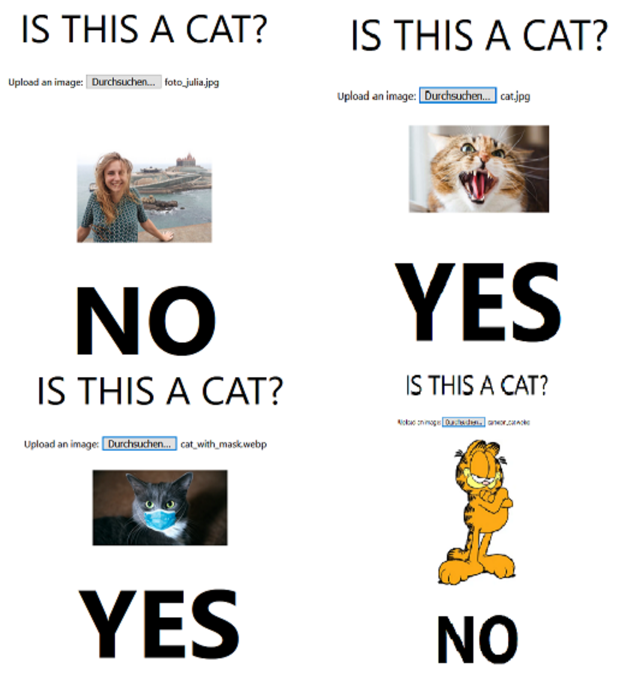

IS THIS A CAT?
Upload an image: Durchsuchen... foto_julia.jpg
NO
IS THIS A CAT?
Upload an image: Durchsuchen... cat.jpg
YES
IS THIS A CAT?
Upload an image: Durchsuchen... cat_with_mask.webp
YES
IS THIS A CAT?
NO

Figure 4.1: Apparently I am not a cat and neither is Garfield

Neural networks can be used for a big range of tasks. There are 2 types of problems they are good at solving; classification problems, and non-linear regression problems. The premise of www.isthisacat.com is a good example of a classification problem. Making financial forecasts is a good example of a non-linear regression problem. Neural networks could in theory solve linear regression problems too, but there exist more efficient mathematical methods for these problems.

4.2. The Multilayer Perceptron Network or Feedforward Neural Network

The multilayer perceptron network is often referred to as Feedforward Neural Networks (FNN). We first look into how neurons are assigned values and how activation functions work. Then we look at how we can minimize the cost function by using gradient descent. Finally, we discuss how backpropagation is used to calculate the gradient of all the neurons in the network.

4.2.1. Structure

Neural networks of this type consist out of layers of neurons (perceptrons). Neurons are entities within the network that hold values. In some networks, for example, each neuron can only hold values between 0 and 1. The layered structure enables the network to find patterns. If certain neurons in the first layer have a high value this triggers neurons in the next layer to have a high value until the output layer is reached and an outcome is produced. These networks always have an input layer, a number of hidden layers, and an output layer. Figure 4.2 shows an example of a typical multilayer perceptron network.

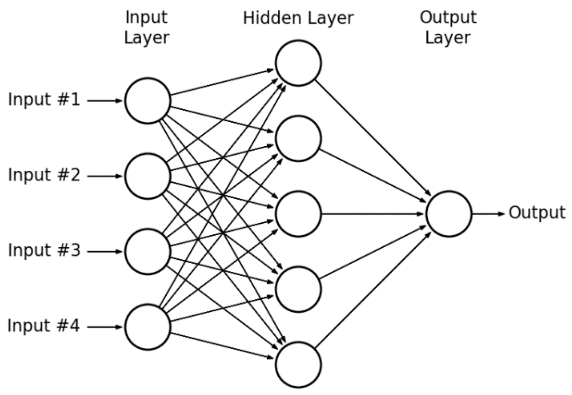


Figure 4.2: A hypothetical example of a multi-layer perceptron network [14]

The first layer is the input layer and has as many neurons as input variables. If the input data consists out of black and white photographs, the number of input neurons is equal to the number of pixels. Each input neuron would contain the greyscale value of a pixel.
All of the learning takes place in the hidden layers. In subsection 4.2.3 and Equation 4.2.15, the workings of hidden layers is clarified. A neural network with more than two hidden layers is called a deep neural network. How the number of hidden layers is chosen differs per problem but is partly a trial and error procedure.
The last layer is the output layer and answers the question which the network was supposed to solve. For a regression problem involving a financial forecast, the answer will be a single number such as a price. In those cases, the output layer contains only one neuron. Binary classification problems also have one output neuron. A single neuron suffices because the answer is either a 1 (yes) or 0 (no). For non-binary classification problems, one needs as many neurons as categories in the output layer.

4.2.2. Activation

Neurons are assigned values (activations) by activation functions. Every neuron, except for the neurons in the input layer, has an activation function. The activation functions used for the input and hidden layers are non-linear and enable the network to solve for non-linearity. The activation function used for the last layer can be linear depending on the problem. The range of values the neuron can contain depends on the type of activation function. Activations are calculated using weights and biases. Every connection within the network has a weight. And every neuron, except for the neurons in the input layer, has a bias. Weights indicate how much a certain activation from a previous layer should count. Biases are used as a threshold value.

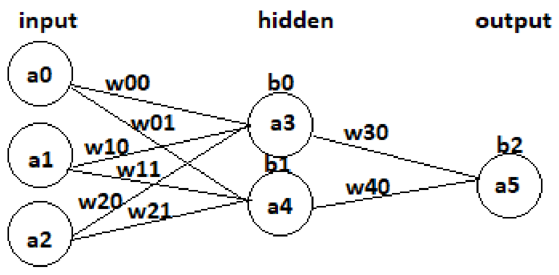


Figure 4.3: A representation of the activations (a) weights (w) and biases (b) within a simple multilayer perceptron network

The input of the activation function is given by a variable denoted as z. z is given by Equation 4.2.12[14]. For the neuron with activation $a3$(Figure 4.3) z is thus given by Equation 4.2.2

$$z = \sum w * a - b. \tag{4.2.1}$$

$$\sum_{n=0}^{2} w_n * a_n - b_0 \tag{4.2.2}$$

The activation functions in hidden layers are non-linear. The 2 most commonly used activation functions are the Sigmoid and ReLu (rectified linear unit) functions. The Sigmoid function 'squeezes' whatever input it gets between zero and one (Figure 4.4, Equation 4.2.3[14]). Relu is partly linear, for z>0 the function is linear but equals 0 for z<0 (Figure 4.5, Equation 4.2.4[14]).
For the output layer, different activation functions can be used depending on the type of problem. For regression problems that output number(s) a linear activation function is used. For classification problems, a Sigmoid function is used. For these problems, the output value of the neuron corresponds to how certain the network is about its decision. An output activation of 0.93 means that the network is very confident about its classification.

$$\sigma = \frac{1}{1 - e^{-z}} \tag{4.2.3}$$

$$R(z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z <= 0 \end{cases} \tag{4.2.4}$$

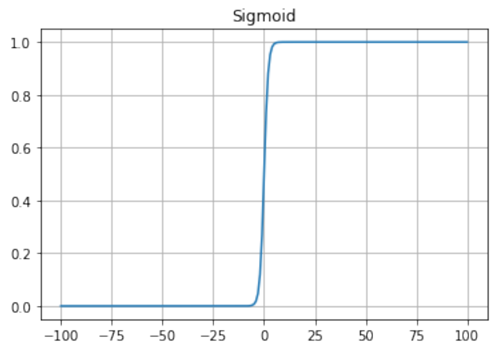


Figure 4.4: Sigmoid function for z between -100 and 100

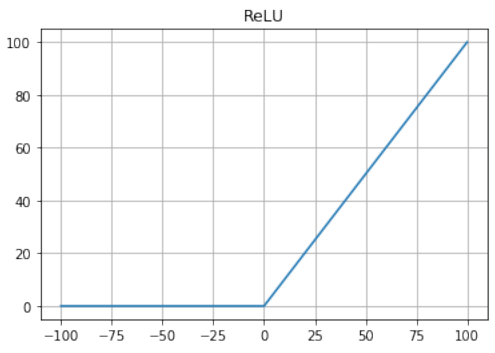


Figure 4.5: ReLU function for z between -100 and 100

4.2.3. Cost Functions and Gradient Descent

A neural net learns by adjusting its weights and biases based on training input. The network learns via training. We feed the network training inputs and we want the network to adjust its weights and biases in such a way that the output of the network corresponds to the answer belonging to the training input. We measure the performance of the network using a cost function. With performance, we mean how well the network's output corresponds to the actual (desired) answer belonging to the training input. Multiple types of cost functions can be used depending on the problem. For this explanation, we use the simplest one, the mean squared error Equation 4.2.5 [14].In Equation 4.2.5 w and b denote all the weights and biases of the network, n is the total number of training examples, a is the vector of outputs (activation(s) from the last layer) from the network when x is given as training input, and y is the desired output for training input x. The output a is dependent on x, w and b, but to keep the notation simple this has not been indicated explicitly [14].

$$C(w,b) = \frac{1}{2n}\sum_x (y(x) - a)^2 \tag{4.2.5}$$

By using a cost function we obtain a single number that represents how well the network performs. If the outcome of the cost function is low it means that the network often finds an output value that matches the desired outcome for the training input. The lower the outcome of the cost function the better the network works. Figure 4.6 and Figure 4.7 show plots of cost function dependable on 1 and 2 variables respectively. For these functions, it is easy to find the minimum. The cost function of a neural network depends on all the weights and biases. Neural networks can obtain millions of these, creating very complex cost functions. We can't plot them or calculate their minimum directly. Instead, we use an algorithm called gradient descent. This algorithm is often referred to as optimizer.

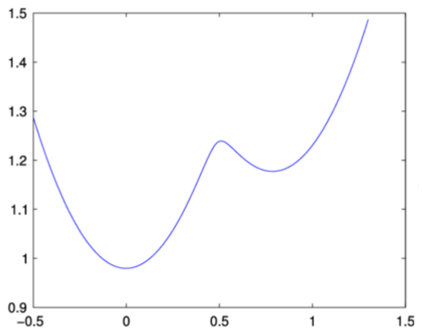


Figure 4.6: Cost function with 1 variable

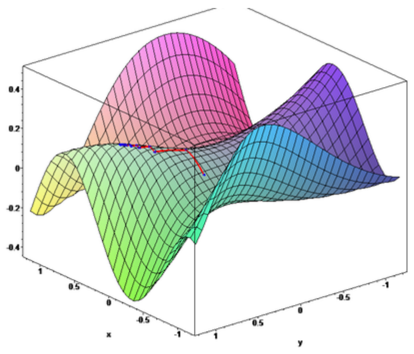


Figure 4.7: Cost function with 2 variables

Gradient descent uses the gradient of the cost function to descent down the slope of the cost function towards a (local) minimum. We want to find how we should change **v**, a vector consisting out of all weights and biases, to lower C(**v**), the cost function. The relation between **v** and (**v**) is given by Equation 4.2.6[14]. The gradient of C(**v**)[14] is defined as Equation 4.2.7.

$$\Delta C(\mathbf{v}) = \nabla C(\mathbf{v}) * \mathbf{v} \tag{4.2.6}$$

$$\nabla C(\mathbf{v}) = (\frac{\partial C(\mathbf{v})}{\partial w1}, \frac{\partial C(\mathbf{v})}{\partial b1})^T \tag{4.2.7}$$

If $\Delta C(\mathbf{v})$ is negative we descent down the function onto a (local) minimum. By choosing (**v**) as the gradient of C(**v**) times some negative constant (Equation 4.2.8[14]), $\Delta C(\mathbf{v})$ will always be negative. By combining Equation 4.2.6 and Equation 4.2.8 we get Equation 4.2.9 [14]. Equation 4.2.9 is the most basic form of gradient descent. The constant epsilon is called the learning rate. It acts as the step size by which the function descends towards the (local) minimum. By setting the learning rate too low, the process will take a lot of time. But if the step size is chosen too large you risk missing the local minimum (overshoot).

$$\mathbf{v} = -\epsilon * \nabla C(\mathbf{v}) \tag{4.2.8}$$

$$\Delta C(\mathbf{v}) = -\epsilon * \nabla C(\mathbf{v})^2 \tag{4.2.9}$$

Computing the gradient for every training example is computationally heavy. To speed up the process mini-batch stochastic gradient descent is often used [14]. This means that the cost function is not calculated sepa-

rately for every training example. The average value of the cost function for a small group of training examples is taken instead. This small group is called the mini-batch and is chosen at random.
Every time the weights and biases are updated for one batch is called an iteration. Once the parameters have been updated for all training examples it's called an epoch. When using mini-batches one epoch consists out of multiple iterations.
Another method to speed up gradient descent is to have an adjustable learning rate. In the beginning, the learning rate can be higher without the risk of overshooting since the cost function is far from a minimum. Near a minimum, the learning rate should be low to prevent overshooting. Some optimizers use an adjustable learning rate and increase or decrease the learning rate based on recent gradient values.
Popular optimizers such as the ADAM algorithm combine both of the aforementioned techniques[15].

4.2.4. Backpropegation

The algorithm which calculates the gradient is called backpropagation. The algorithm finds how sensitive the cost function is to the changes of each particular weight and bias. The backpropagation algorithm consists out of 5 steps.

- Step 1: Set the activations for the input layer
- Step 2: Calculate all the activations, using the initial weights and biases
- Step 3: Calculate the error for the final layer
- Step 4: Calculate the error back for all layers
- Step 5: Calculate the gradient of the cost function using the error

In the following paragraphs we explain how steps 3 to 5 are calculated. In subsection 4.2.2 and subsection 4.2.1 we have discussed how steps 1 and 2 are performed.
Calculating Step 3
Initially we have no idea how a single weight or bias influences the cost function. Instead, we start with the error, a partial derivative described by Equation 4.2.10[14].Equation 4.2.11[14] is a vectorized form of Equation 4.2.10, and ∘ represents the Hadamard product.

$$\delta_j^L = \frac{\partial C_x}{\partial z_j^L} = \frac{\partial C_x}{\partial a_j^L}\sigma'(z_j^l) \tag{4.2.10}$$

$$\delta^L = \nabla_a C_x \circ \sigma'(z^L) = (a^L - y) \circ \sigma'(z^L) \tag{4.2.11}$$

$$z_j^L = \sum_{n=k} w_{j\,k}^{\,L} * a_k^{L-1} - b_j^L \tag{4.2.12}$$

In the equations above the superscript L refers to the layer. The subscript j and k stand for the neuron in layers L and L-1 (fig.4.8).

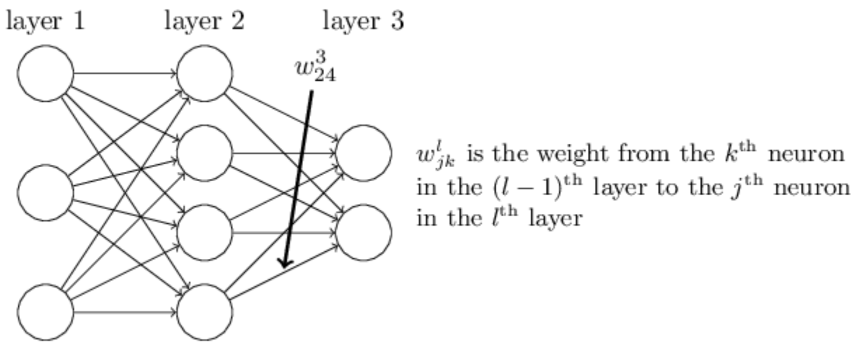

layer 1
layer 2
layer 3
w_{24}^3
w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

Figure 4.8: Visual representation of the sub and superscripts [14]

The quadratic cost function is given by Equation 4.2.13[14]. Since $a = \sigma(z)$ and $\sigma = \frac{1}{1-e^{-z}}$ the error δ as defined in 4.2.10 can be calculated easily. This also holds true for other activation and cost functions.

$$= \frac{1}{2n} * \sum_n (a-y)^2 \tag{4.2.13}$$

$$\sigma = \frac{1}{1-e^{-z}} \tag{4.2.14}$$

Calculating Step 4
Using the chain rule Equation 4.2.10 can be rewritten as Equation 4.2.15[14].Equation 4.2.15 shows how the error δ^L can be calculated back, starting from the output layer.

$$\delta^L = ((w^{L+1})^T \delta^{L+1}) \circ \sigma'(z^L) \tag{4.2.15}$$

Calculating Step 5
After Step 4, the error of each layer is known. By combining the chain-rule, some clever algebra, and Equation 4.2.10 we find expressions for the partial derivative of the cost function. (Equation 4.2.16 and Equation 4.2.17 [14]).The error is known, and thus we can easily solve for the gradient.

$$\frac{\partial C}{\partial w_j k^L} = \delta_j^L a_j^{L-1} \tag{4.2.16}$$

$$\frac{\partial C}{\partial b_j^L} = \delta_j^L \tag{4.2.17}$$

4.3. Convolutional Neural Networks

CNNs work a little differently than FNNs. CNNs make use of convolutional layers in addition to fully connected layers (the type of layers used in an FNN). The name convolutional layer is a bit misleading. Most machine learning libraries use cross-correlation rather than convolution. The results are practically the same. Performing a convolution is the same as performing a cross-correlation with a flipped filter. Within machine learning, the operation is called convolution and we do the same in this section.
Convolutional neural networks in general consist out of convolutional layer(s), pooling layer(s), and fully connected layer(s).Figure 4.9 shows an example of an CNN architecture.

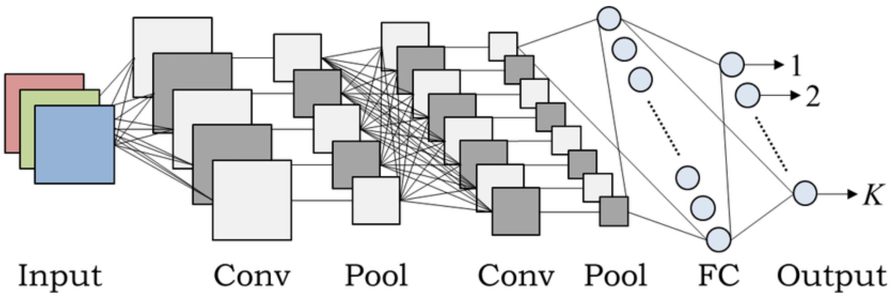


Figure 4.9: an example of the architecture of a CNN [16]) [15]

4.3.1. How Do Convolutional Networks Work?

In contrast to an FNN, the first layer of a convolutional network takes a tensor as input (1,2 or 3D). A convolutional layer then convolves the input with filters. These filters are also referred to as kernels. Figure 4.10 shows an example of a 2D convolution. In the case of Figure 4.10, the output is restricted to positions where the filter lies entirely within the matrix. In Figure 4.10 the filter moves over the matrix one block at a time, meaning the stride is one. The output of the convolution is called a feature map. All of the values in this feature map are put through an activation function. A single convolutional layer generally has more than one filter, and thus produces several of these feature maps.

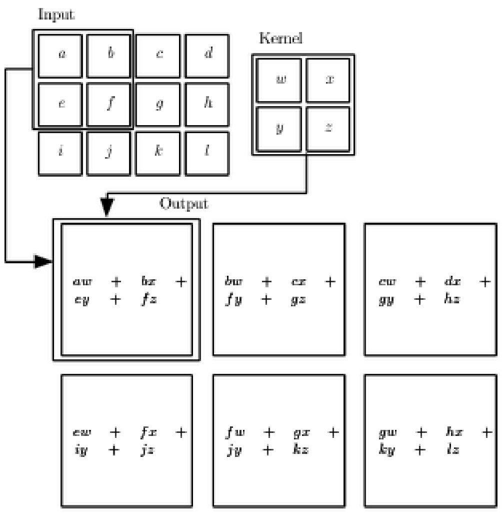


Figure 4.10: An example of how cross-correlation is performed using a filter(kernel) [15]

Convolutional layers learn filters. Every value in the filter acts as a weight and each filter also has one bias. Every value of the feature map has the same weights and the same bias. Since every feature map shares the same weights and bias, it is computationally efficient. The weights and biases of convolutional layers are learned through backpropagation and gradient descent.
A convolutional layer is often followed by a pooling layer. A pooling layer compresses the feature map. It does this by replacing a region of values with one number. For example, this number can be the highest or the average of all the values in the region. The size of this pooled region is denoted as the filter size.Figure 4.11 shows an example of both max and average pooling. Pooling makes the network more efficient and helps prevent overfitting by discarding values.

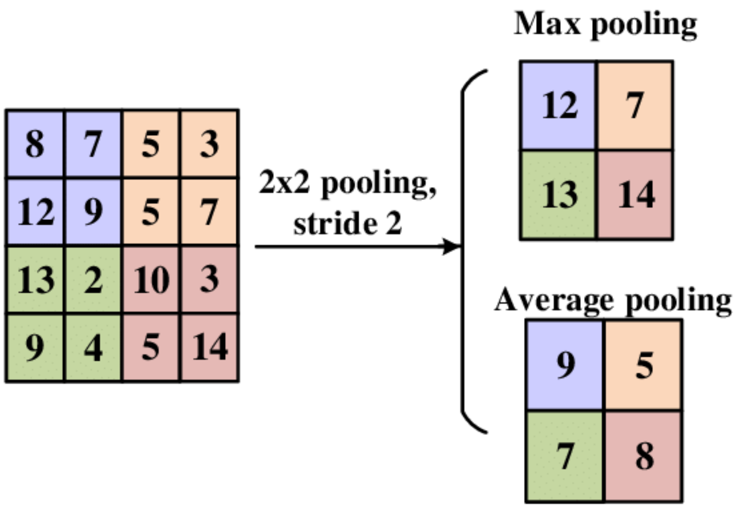


Figure 4.11: Example of max and average pooling, note the stride is 2 so the filter moves 2 blocks at a time [17]

To transition from the convolutional or pooling layers into the fully connected layers it is necessary to perform an operation called flattening. Flattening reshapes the feature maps into a 1D shape to make the transition possible. From here on the CNN works the same as an FNN.

4.4. How to Train a Neural Network?

For this thesis, we train our network with examples of earthquakes and seismic noise. These examples need to be labelled and normalized. The next step would be to divide this data into three subsets: training, validation, and test. We use the training data to train the network. The validation data is used to check if the performance achieved with the training data is reproducible. Learning curves show the accuracy and/or loss of the network per epoch. The accuracy corresponds to how accurate the outcomes of the network are compared to the desired output of the training input. The loss refers to how much the value of the cost function has dropped. the Learning curves show both the training and validation data.Figure 4.12 Shows the learning curve of a properly functioning network; the training and validation data show the same behaviour across the epochs.

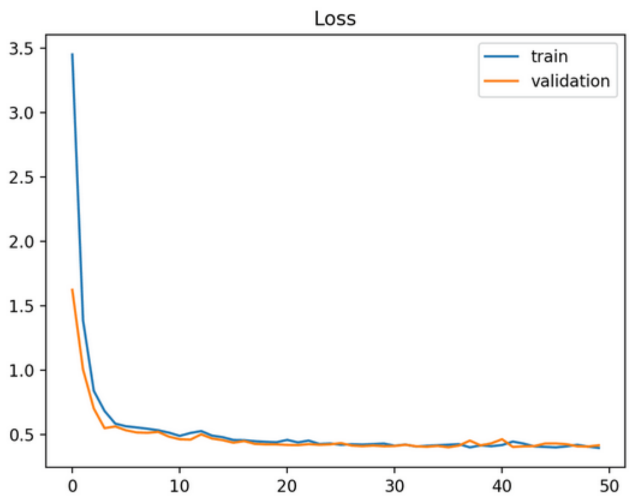


Figure 4.12: Example of a learning curve, the validation and test data show the same behaviour indicating that the network is trained well[18]

The training and validation data are used during fine-tuning of the hyperparameters. Hyperparameters are all the network parameters that can not be learned but are chosen by a person instead. Such as the number of layers and neurons in the network. To prevent choosing settings that prefer the training and validation data over new data a final check is done using the test set. After the network has been fully trained, we ask the network to classify the test set. When the network works properly the test set yields similar results.

4.4.1. What Can Go Wrong During Training?

It's odd that neural networks perform well in the first place, and there isn't a definitive answer to why this is either [15]. Since neural networks can have up to millions of parameters, it is no wonder that they can fit any function. But neural networks surprisingly often find the fit it's supposed to. Based only on the training data the network often finds a fit representing the concept we wanted the network to learn. The concept of finding a fit that represents the concept we want the network to learn is called generalization.

4.4.2. Overfitting

Sometimes a network does not generalize well because it has learned the training data too well. This is called overfitting, meaning that the neural net fitted the training data in a very specific manner. This fit only fits the training data well. Causes of overfitting can be that one has run too many epochs. This can encourage the network to learn the particularities of the training data (like noise) instead of learning the general idea. Overfitting can be prevented by increasing the number of training examples. Using fewer parameters in the network like reducing the number of neurons, layers, filters, etc. also prevents overfitting. However, sometimes it is hard/impossible to gather more training data, and bigger neural networks are more powerful. There are other techniques to prevent the network from overfitting. These are called regularization techniques.

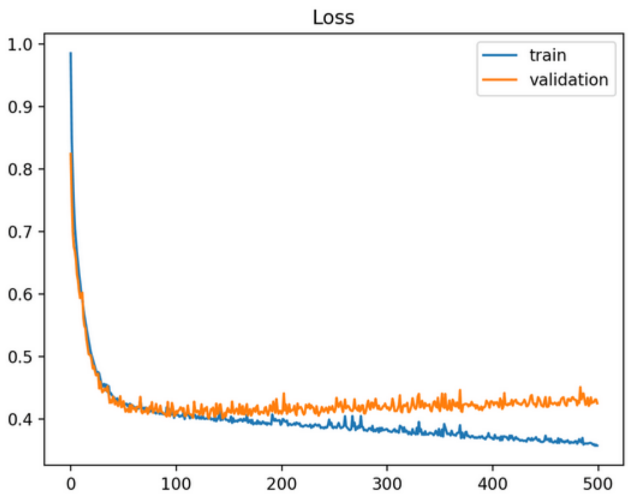


Figure 4.13: Example of what overfitting looks like, the loss on the training data keeps decreasing while the loss on the validation data stays the same after epoch 50 [18]

4.4.3. Regularization: Drop-out Layers

Regularization techniques are techniques that prevent overfitting. Making use of drop-out layers is a popular regularization technique that is also used for this thesis. When using a drop-out layer a random part of the neurons in the layer is not trained for an iteration. Every iteration a different subset of neurons is left out. Using drop-out layers you essentially train multiple networks. These networks are not likely to over-fit in the same way since they have been trained with different data. Therefore the network as a whole won't overfit easily.

4.4.4. Underfitting

The opposite of overfitting is underfitting. When a network is underfitted it has not been able to learn the training data properly. Solutions for underfitting can be either more training (train for more epochs) or by using less regularization.

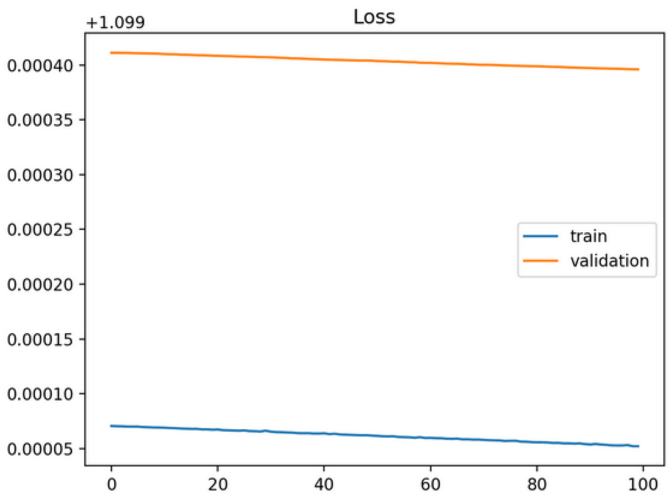


Figure 4.14: Example of what under fitting looks like, the loss on the training data seems to be flat[18]

5 Development of Neural Networks for Earthquake Detection

In this chapter, we discuss the development of a neural network for earthquake detection. We first need to find a suitable architecture. Several architectures have been tried, section 5.1 describes these architectures. Secondly, we need training data. There is not enough field data available to train the networks. Therefore synthetics are used instead. Using the ERZSOL3 software we have forward modelled synthetic events. This process is explained in section 5.2. The architectures have been fully trained with synthetics using different synthetic datasets. The real events have been kept apart for later testing. Finally, we compare the different architectures and training sets by using real events and seismic noise. In Appendix A an overview of all seismic noise and events used in this chapter can be found. Please note that the unit of amplitude for all seismic recordings shown in this chapter should be m/s instead of m.

5.1. Architectures Used

As explained in chapter 4 designing a properly working neural network from scratch, especially a deep neural network, can take up a lot of time and involves trial and error. To speed up the process it was chosen to start with existing networks and modify if needed. These networks were originally designed to solve problems similar to earthquake detection. Three architectures have been tried.

5.1.1. Network 1

The first network is an FNN, originally designed to find a signal buried in different levels of noise. This makes it also suitable for finding earthquakes. Since we are looking for a signal produced by an earthquake buried in seismic noise.Figure 5.1 shows a visualisation of the network.

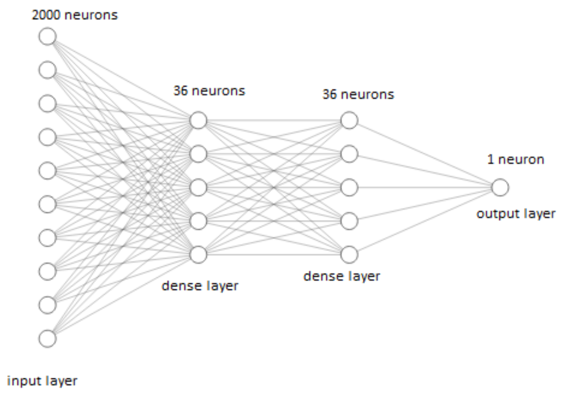


Figure 5.1: Visualisation of architecture network 1

This FNN consists out of three layers. Every layer is followed by a drop-out layer. The activation function used for the first two dense layers is ReLu and Sigmoid for the last. As gradient optimizer, Adam has been used, with a mini-batch size of 50. Figure 5.2 shows the details of the network.

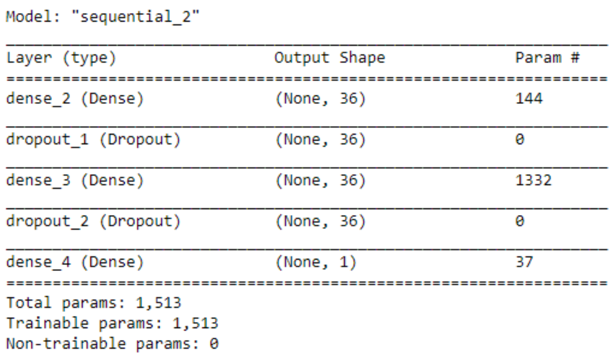

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 36)                144
_________________________________________________________________
dropout_1 (Dropout)          (None, 36)                0
_________________________________________________________________
dense_3 (Dense)              (None, 36)                1332
_________________________________________________________________
dropout_2 (Dropout)          (None, 36)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 37
=================================================================
Total params: 1,513
Trainable params: 1,513
Non-trainable params: 0
```

Figure 5.2: Details on architecture network 1 FNN

5.1.2. Network 2

The second network is a 1D convolutional neural network (1D CNN), Figure 5.3 is a visualization of the architecture. Network 2 is a modified version of a network created to recognize human activity (walking, running, playing soccer, etc.) based on 1D time series. Test subjects were given a bracelet that recorded their movement in the form of 1D time-series data while performing different physical activities. This study is described in [19]. This network was considered suitable for earthquake detection since it showed to be good at differentiating between different types of 1D time series. For earthquake detection we want the network to be able to differentiate between seismic noise and earthquakes. The network has been altered to make it suitable for a binary classification problem. Originally the convolutional layers contained 64 filters. However, this led to overfitting. To reduce overfitting the number of filters was set to 10.

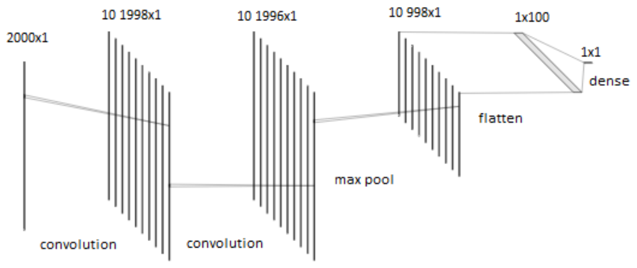
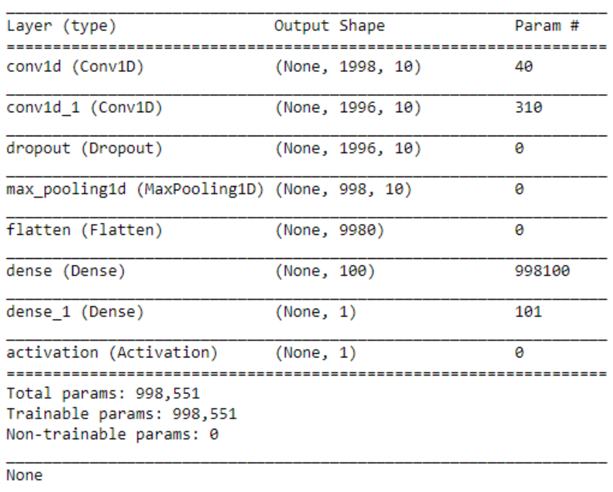


Figure 5.3: Visualization of architecture network 2

This network consists out of two convolutional layers, one max-pooling layer, and two dense layers. Each convolutional layer contains 10 filters of size 3, with stride 1. The last convolutional layer is followed by a drop-out layer. The max-pooling layer has filter size 2. The first dense layer contains 100 neurons and the second contains one neuron. Relu is used as activation throughout the network except for the final layer where Sigmoid is used. Adam is used as an optimizer, with a mini-batch size of 50. Figure 5.4 shows the details of the architecture.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 1998, 10)          40
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1996, 10)          310
_________________________________________________________________
dropout (Dropout)            (None, 1996, 10)          0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 998, 10)           0
_________________________________________________________________
flatten (Flatten)            (None, 9980)              0
_________________________________________________________________
dense (Dense)                (None, 100)               998100
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
_________________________________________________________________
activation (Activation)      (None, 1)                 0
=================================================================
Total params: 998,551
Trainable params: 998,551
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 5.4: Details on architecture network 2 1D CNN

5.1.3. Network 3

The third network is also a 1D CNN. This network is an altered version of the ConvetNetquake network as described in [12]. This network is designed to detect and locate earthquakes. The network has been modified to fit a binary classification problem. The original network contained eight convolutional layers and required input data with a length of 1000 time steps. For the recordings made in Zeerijp, this would correspond to 5-second intervals. This interval was too small to do much learning. In the end, a 10-second interval has been used. Adding an extra layer, so that we end up with the same number of neurons in the dense part as the original network, gave the best results. Figure 5.5 shows the network architecture.

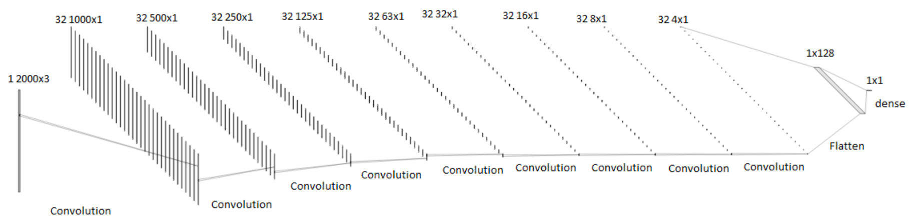
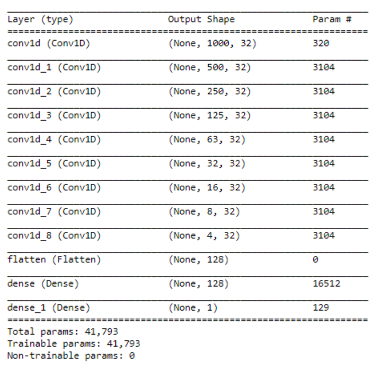


Figure 5.5: Visualization of architecture network 3 1D CNN

The required input shape of the data differs from the previous CNN. This CNN requires all three channels from one station together as input. The input is a 2D tensor, with shape 3 X 2000. This network has 9 convolutional layers with 32 filters each. These filters have size 3, and stride 2. Meaning that after every convolution the size of the data is halved. The first fully connected layer uses a ReLU activation function, and the output layer a Sigmoid function. The optimizer used was ADAM, with a mini-batch size of 20.Figure 5.6 shows an overview of the details of this architecture.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 1000, 32)          320
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 500, 32)           3104
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 250, 32)           3104
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 125, 32)           3104
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 63, 32)            3104
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 32, 32)            3104
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 16, 32)            3104
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 8, 32)             3104
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 4, 32)             3104
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 128)               16512
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 41,793
Trainable params: 41,793
Non-trainable params: 0
```

Figure 5.6: Details on architecture network 3 1D CNN

5.2. Generating Synthetic Events

A neural network needs a lot of train examples to learn a concept. However, there have only been 83 earthquakes in Zeerijp between 2015 and 2019. This is too little data to train a network with. Therfor we use synthetic data instead. We have used software to forward model events. We used the software routine ERZSOL3. ERZSOL3 only models clean traces without noise. To make the synthetics look more like real events, seismic noise was added on top of these traces.

5.2.1. ERZSOL3

The software ERZSOL3 is written by Brian Kennettin Fortran. The algorithm makes use of the reflectivity method. The formulas given by Equation 5.2.1 [20] and Equation 5.2.2[20] form the basis of the reflectivity method. In these equations, ρ_1 represent the density of the underlying layer and ρ_2 the density of the upper layer. v_1 and v_2 represent the velocity of the underlying and upper layer. $w(t)$ is the wavelet generated by the earthquake and $n(t)$ represents seismic noise. ERZSOL3 only models a trace without noise. The noise is added afterwards.

$$T(t) = R_0 * w(t) + n(t) \tag{5.2.1}$$

$$R_0 = \frac{(\rho_2 v_2 - \rho_1 v_1)}{(\rho_2 v_2 + \rho_1 v_1)} \tag{5.2.2}$$

ERZSOL3 is designed to calculate the seismic response from a set of stratified uniform layers generated by a point source moment tensor. A detailed explanation of the inner workings of the ERZSOL 3 routine can be found in[20].
ERZSOL3 requires three input files; a model file, a cmd file, and a dst file. Examples of these files can be found in Appendix B. The model file consists out of a 1D velocity and density model. The cmd file contains information about the source mechanism; moment tensor components, dominant frequency, type of wavelet, and depth of the source. The cmd files also contain information on the slowness, desired sampling frequency, and length of the output signal. The dst file contains information on the number of receivers and the location of the receivers relative to the source.

5.2.2. Velocity Model

There is a detailed 3D velocity and density model available for Groningen. We have used a 1D section of this model cut at RDx=253km and RDy=569km (Figure 5.7). However ERZSOL3 requires only 1 value per layer. Therefore the average values for each layer have been used. Figure 5.8.

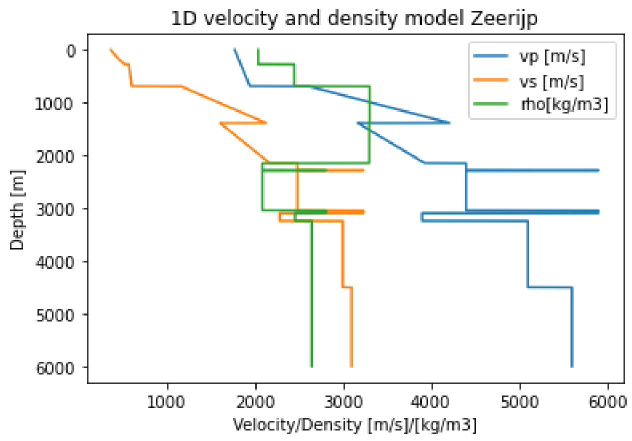


Figure 5.7: 1D velocity and density model for the Zeerijp area

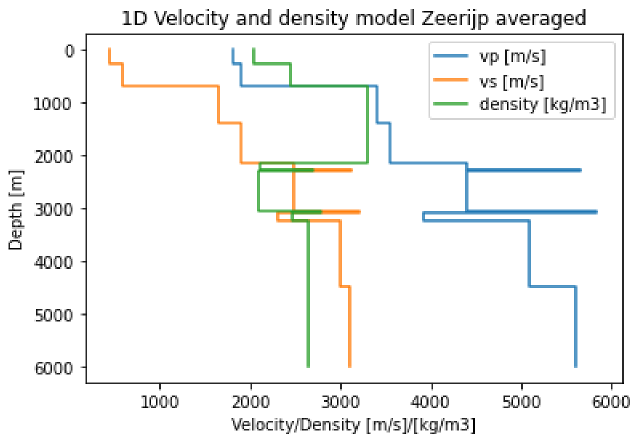


Figure 5.8: 1D velocity and density model for the Zeerijp are with averaged values for vp and vs

5.2.3. Source Mechanism and Wavelet

[21] Describes the results of probabilistic moment tensor inversion for earthquakes in Groningen. The events indicated by numbers 4,7,8,9 and 11 in Figure 5.9 are located in Zeerijp.

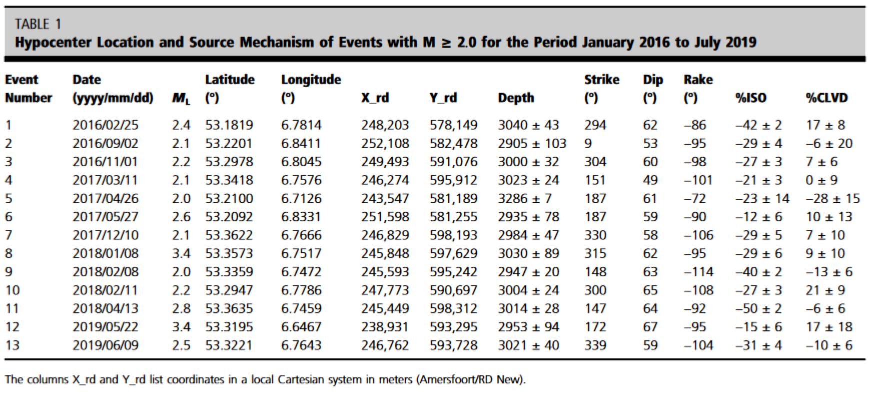


TABLE 1
Hypocenter Location and Source Mechanism of Events with M ≥ 2.0 for the Period January 2016 to July 2019

Event Number	Date (yyyy/mm/dd)	M_L	Latitude (°)	Longitude (°)	X_rd	Y_rd	Depth	Strike (°)	Dip (°)	Rake (°)	%ISO	%CLVD
1	2016/02/25	2.4	53.1819	6.7814	248,203	578,149	3040 ± 43	294	62	−86	−42 ± 2	17 ± 8
2	2016/09/02	2.1	53.2201	6.8411	252,108	582,478	2905 ± 103	9	53	−95	−29 ± 4	−6 ± 20
3	2016/11/01	2.2	53.2978	6.8045	249,493	591,076	3000 ± 32	304	60	−98	−27 ± 3	7 ± 6
4	2017/03/11	2.1	53.3418	6.7576	246,274	595,912	3023 ± 24	151	49	−101	−21 ± 3	0 ± 9
5	2017/04/26	2.0	53.2100	6.7126	243,547	581,189	3286 ± 7	187	61	−72	−23 ± 14	−28 ± 15
6	2017/05/27	2.6	53.2092	6.8331	251,598	581,255	2935 ± 78	187	59	−90	−12 ± 6	10 ± 13
7	2017/12/10	2.1	53.3622	6.7666	246,829	598,193	2984 ± 47	330	58	−106	−29 ± 5	7 ± 10
8	2018/01/08	3.4	53.3573	6.7517	245,848	597,629	3030 ± 89	315	62	−95	−29 ± 6	9 ± 10
9	2018/02/08	2.0	53.3359	6.7472	245,593	595,242	2947 ± 20	148	63	−114	−40 ± 2	−13 ± 6
10	2018/02/11	2.2	53.2947	6.7786	247,773	590,697	3004 ± 24	300	65	−108	−27 ± 3	21 ± 9
11	2018/04/13	2.8	53.3635	6.7459	245,449	598,312	3014 ± 28	147	64	−92	−50 ± 2	−6 ± 6
12	2019/05/22	3.4	53.3195	6.6467	238,931	593,295	2953 ± 94	172	67	−95	−15 ± 6	17 ± 18
13	2019/06/09	2.5	53.3221	6.7643	246,762	593,728	3021 ± 40	339	59	−104	−31 ± 4	−10 ± 6

The columns X_rd and Y_rd list coordinates in a local Cartesian system in meters (Amersfoort/RD New).

Figure 5.9: Table containing moment tensor parameters for earthquake in Groningen[21]

Using Figure 5.9, the values found in the table below have been used to describe the moment tensor. The source mechanism is modelled as only having a double coupled component. The isotropic and compensated linear vector dipole components have not been taken into account.

Moment tensor parameters	
Strike	140° - 340°
Dip	45° - 65°
Rake	-115° - -90°

Table 5.1: Moment tensor parameters

A Ricker wavelet has been. The wavelets are generated with a frequency between and 3 and 22Hz, the same as the dominant frequencies of real events. The events are generated for magnitudes between 0 and 4M. The sampling frequency is 200Hz, the same as the sampling frequency used in Zeerijp. The length of the produced wavelet is 10 seconds.

5.2.4. Source and Receiver Locations

Most of the earthquakes in the Groningen area happen at depths between 2.8 and 3km [21]. The Zeerijp area is 7 by 8km. The source locations are chosen as random points within a cube based on these dimensions (Figure 5.10). There are many faults in the area, therefore it is realistic to pick source locations as random points within this volume. The stations used as receivers are G184, G144, and G094.Figure 5.11 shows the locations of these stations.

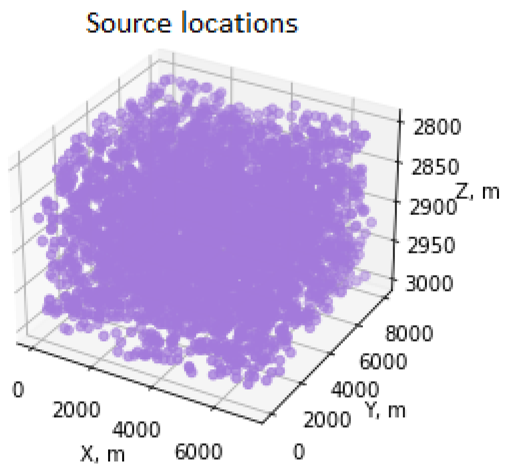


Figure 5.10: Example of random distribution of source locations

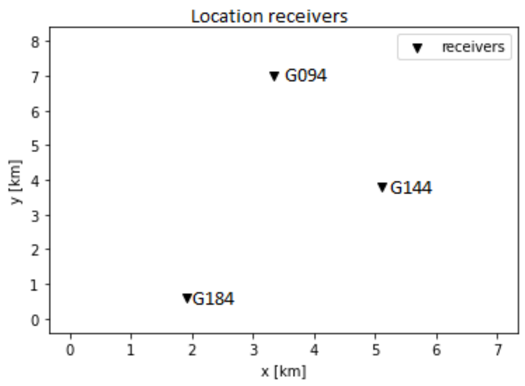


Figure 5.11: Location of receivers

5.2.5. Adding Noise to Synthetics

To make the synthetics more realistic, noise is added on top of the generated events. The noise added on top of the synthetic was from a different recording, then the noise used to train the neural network. On Figure 5.13 you can see the result of the forward modeling, the final result with added noise can be seen in Figure 5.14. Figure 5.14 and Figure 5.15 show how the synthetic looks next to a real event.

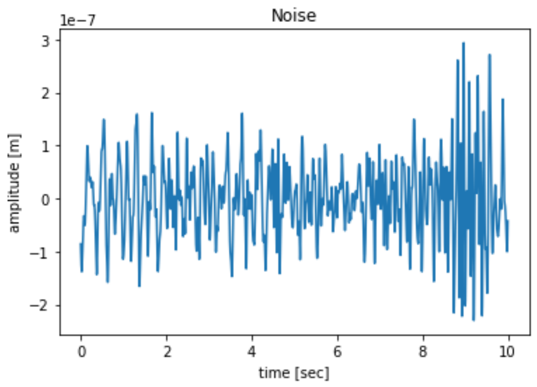


Figure 5.12: Noise

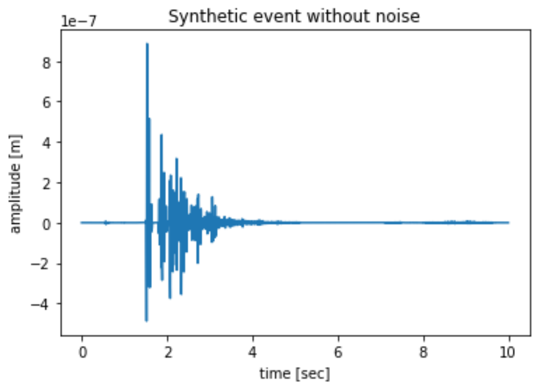


Figure 5.13: Synthetic seismogram without noise HHZ component.

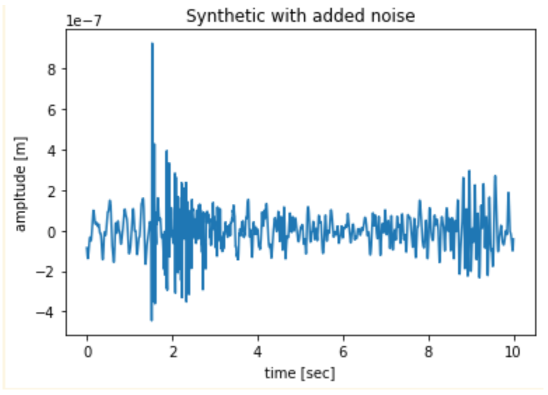


Figure 5.14: Synthetic seismogram with noise.

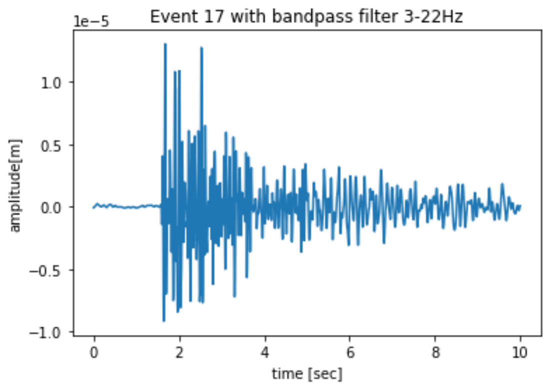


Figure 5.15: Event 17 HHZ component

5.3. Training Sets

We use three different training sets. These sets contain screens; 10-second intervals of seismic data. The training sets contain 36000 or 72000 screens. Half of the screens belong to synthetic events and half of the screens belong to seismic noise. We use different sizes to investigate if increasing the amount of data can improve the results. We use 2 dominant frequency ranges 3-22Hz and 3-15Hz. The 3-22Hz range is based on the dominant frequency observed in real events. However high frequencies are very sensitive to small-scale heterogeneities, small disturbances in ray path and source location[22] [6]. Therefore a frequency range of 3-15Hz is also considered. The table below shows the specifics of each set.

	Size	Dominant frequnecies
Set 1	36000 screens	3-22 Hz
Set 2	72000 screens	3-22 Hz
Set 3	36000 screens	3-15 Hz

Table 5.2: Different training sets

5.4. Comparing Different Architectures and Training Sets

The trained networks are tested with 47 real events (423 screens) and 4000 instances of noise (36000 screens). The best network is found by comparing the number of correct detections made by each setup. We are looking for an architecture and training set combination that produces a high number of true positives and a low number of false positives.Figure 5.16 and Figure 5.17 show flowcharts illustrating the difference between training and testing of the networks.

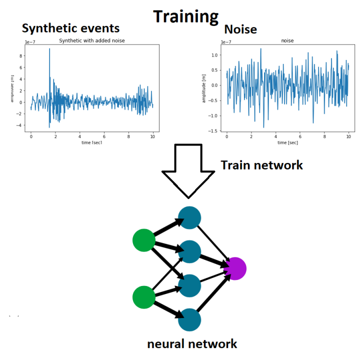


Figure 5.16: Flowchart training network

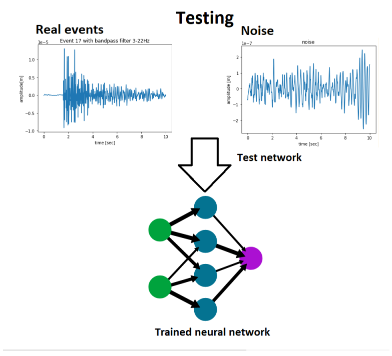


Figure 5.17: Flowchart testing network

5.4.1. How Does a Neural Network Make a Detection?

Each event is recorded on 3 stations, and these stations record 3 channels. This means that every event has 9 screens. Once the network recognizes one of the screens as an event, the event is detected. The more screens per event it recognizes the more confident the network is about its detection. On average the network only classifies 2/9 screens as an event. The number of correctly classified events can thus differ from the number of screens that have been correctly classified. If none of the screens is classified as an event, the time interval is classified as noise. Figure 5.18 shows a visualisation of this process with 3 screens, in reality, 9 screens are used.

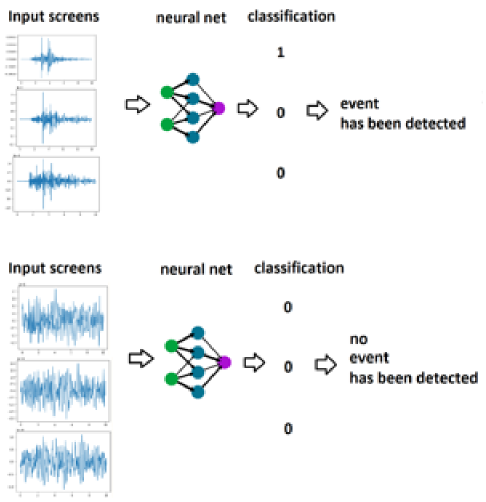


Figure 5.18: Visualisation of detection

5.4.2. Result Comparison

We observe significant differences in classification accuracy between the different architectures. Training set 2 produces the least amount of false positives for all the architectures. The best performing network is network 2, trained with set 2. Network 2 shows the best combination between the amount of true and false positives. Network 2 correctly classified 45/47 events and 3885/4000 instances of noise. We use a confusion matrix to visualise the performance of the networks. Figure 5.19 illustrates how a confusion matrix works; FN stances for False Negative, TN for True Negative etc.

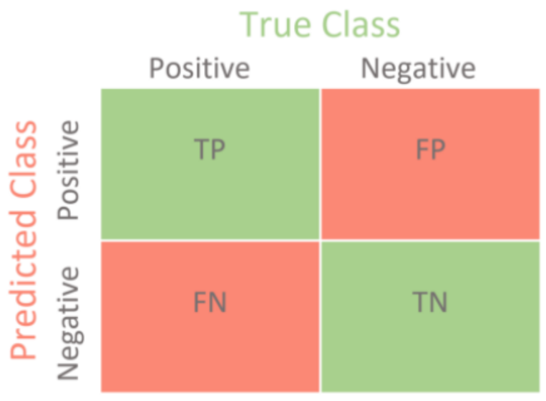


Figure 5.19: Confusion matrix explained [23]

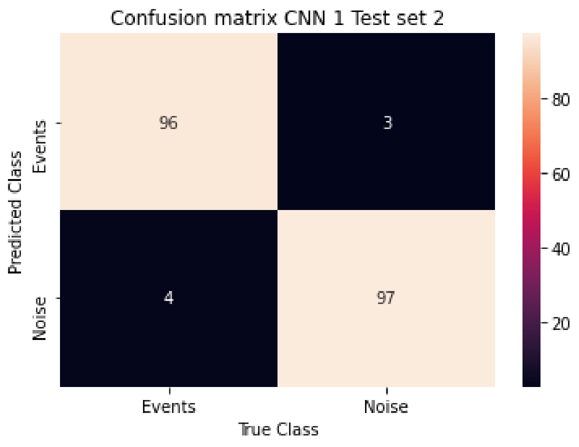


Figure 5.20: Confusion matrix CNN 1 training set 2

Figure 5.21,Figure 5.22 and Figure 5.23 shows how network 2 performed compared to the other networks. Network 3 produced less false-positives. However, this network missed most of the events. The confusion matrixes and training curves for all tested architectures can be found in Appendix D.

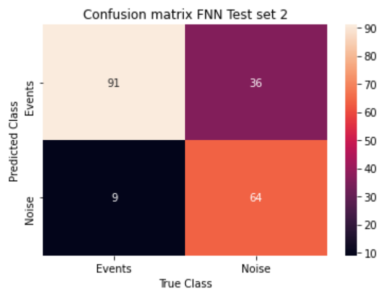


Figure 5.21: Confusion matrix FNN training set 2

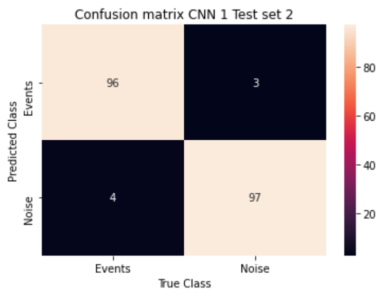


Figure 5.22: Confusion matrix CNN 1 training set 2

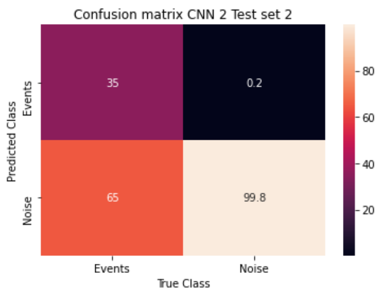


Figure 5.23: Confusion matrix CNN 2 training set 2

6 Comparison Template Matching and Neural Network

In this section,the setup and results of the comparison between template matching and the neural network are presented. Both systems are tested with 18 events <1M and 700 instances of noise. These 18 events have been buried in 4 different noise levels. The systems are compared based on the number of correct detections. An overview of seismic noise and events used in this chapter can be found in Appendix A. Please note that the unit of amplitude for all seismic recordings shown in this chapter should be m/s instead of m.

6.0.1. Noise Levels

Undetected events have bad signal-to-noise ratios. To mimic these ratios, the events are buried in different levels of noise. For the purpose of adding different levels of noise, the events recordings are regarded as consisting out of 100 percent signal. In reality, part of the event recording consists out of noise. However we can't filter the noise out since it has the same frequencies as the earthquake signal. The following definition of signal to noise ratio (SNR) is used [13]:

$$SNR = P_{signal}^2 / P_{noise}^2 \tag{6.0.1}$$

The following signal to noise ratios are used 0.75,0.5,0.25 and 0.125. In order to illustrate the effect of adding noise event 16 (M=0.68), Figure 6.1, is used to illustrate this process.

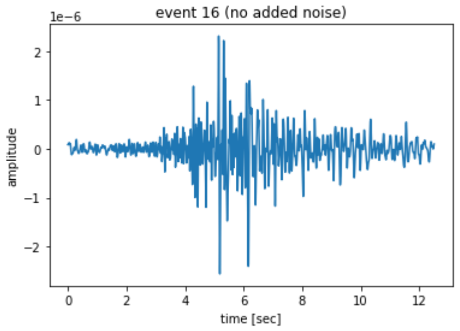


Figure 6.1: Event 16 before additional noise is added

From 6.2 we can observe how event 16 is slowly buried in noise.

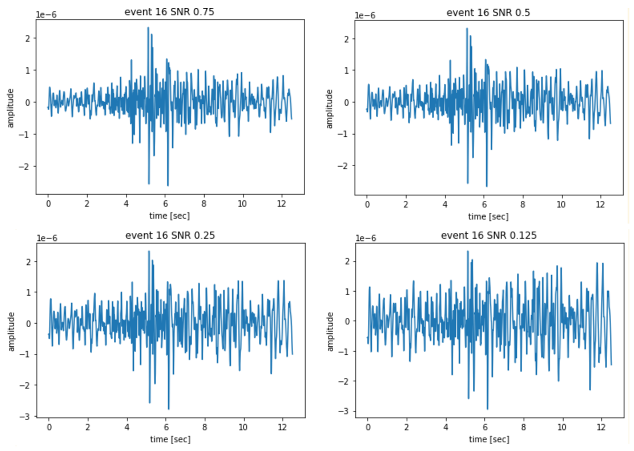


Figure 6.2: Event 16 buried in increasing amounts of noise

6.1. Comparison Results

The system have been tested with 18 events <1M, buried in different noise levels and with 700 instances of noise. The template matching detects more events and has a higher number of average detected screens per event for low levels of noise. But when more noise is added both the number of total detections and the average number of detected screens decreases quickly. The neural network on the other hand is more consistent. As noise levels increase it does get worse but not much. Template matching doesn't produce any false-positives, but the neural network produces 2.

Template matching	No added noise	SNR 0.75	SNR 0.5	SNR 0.25	SNR 0.125
Number of detected events	18/18	15/18	15/18	13/18	13/18
Average number of screens per detection	4.8	3.2	2.4	2	1.7

Table 6.1: Results on events template matching

Neural network	No added noise	SNR 0.75	SNR 0.5	SNR 0.25	SNR 0.125
Number of detected events	16/18	16/18	16/18	16/18	15/18
Average number of screens per detection	2.1	2	1.9	1.6	1.5

Table 6.2: Results on events neural network

	Number of false positives
Neural network	2/700
Template matching	0/700

Table 6.3: Results on noise classification

6.2. Conclusion Comparison

We see that the performance of the neural network is less influenced by adding extra noise. It outperformed template matching on the detection of events buried in noise. However, the neural network performed worse than template matching when classifying noise. Meaning that we expect the neural network to find more

events, but also more false positives. Considering the underlying principles of template matching and neural networks, these results are not surprising. Template matching can only look for very specific patterns (its templates) while the neural network uses a broader definition. Therefore The neural network is more versatile/flexible than the rigid template matching. This flexibility is a double-edged sword. When it comes to noise classification this flexibility causes the neural network to make mistakes.
In this chapter, we find a false-positive rate of 0.29 percent for the neural network. This rate is too high from a practical point of view. This low rate is still problematic because of the size of the seismic data. A single day of seismic recordings would consist out of 17.279.998 instances. When we use the found false-positive rate this would result in 50112 false-positive instances. Even one day would produce an unworkable amount of false positives, let alone six months of data. Template matching didn't produce a single false-positive and is the technique we use on six months of data from Zeerijp.
The neural network did show a lot of potential. Apart from more event detection, another advantage of the neural network is its computational efficiency. It's significantly more efficient than template matching. Some techniques could reduce the number of false positives of the neural net. We could use template matching with a low ccc threshold or make use of other cross-correlation techniques. However, exploring whether these techniques can reduce the false positive rate (enough) is out of scope for this thesis. In chapter 8 we discuss several techniques which could improve the false positive rate of the network.

7 Detecting Events in Continuous Seismic Data

In the previous chapter, it has been concluded that template matching is the best option to use for the detection of missed earthquakes. In this chapter, the setup and results of the experiment on real seismic recordings from Zeerijp are discussed. The results are presented in four categories: known events, strong new detections, weak new detections, and false positives. First, a summary of all detections is given, and some examples for each of the categories is presented. We also compare the magnitudes of strong new event detections to our Gutenberg-Richter analysis. In this way, we can say something about the completeness of the detections. This chapter closes with a detailed overview of all detections and their estimated magnitudes. Please note that the unit of amplitude for all seismic recordings shown in this chapter should be m/s instead of m.

7.1. Set-up

To increase computational efficiency two changes have been made to the set-up of template matching. There have been no changes to the templates themselves or the threshold value.
In the previous experiments, all templates were used on every screen presented to the system. For the test on continuous data, we have changed this. Now only the templates which share the same station and channel as the presented seismic data are used. This speeds up the process significantly and takes up less working memory.
During the past experiments, small screens were used to compare template matching with the neural network. We now use complete days instead of small screens to increase computational efficiency. The new system no longer returns screen numbers but returns all indexes which have a ccc above 0.6 instead. These indexes are later translated into the date and time of the detection.
The data consists out of 6 months of continuous seismic data, recorded by stations G184, G144, and G094. We used recordings of all channels (HH1, HH2, and HHZ). The data has been recorded between January 1st and June 30th, 2018. The system is fed complete days, with no overlap between the days. The seismic data has been filtered with a 3-22Hz bandpass filter.

7.2. Results

The results are presented in four categories: known events, strong new detections, weak new detections, and false positives. Known events are the events that are already in the catalogue. All of these events have been detected indicating that our set-up works properly. The new event detections are split into two categories, strong and weak. The strong detections resemble previous events well (ccc<0.65). These events are visible on at least two stations. In total, we detected 22 strong new event detections. The weak category shows detections that don't resemble previous events well (ccc>0.65). These detections are only visible on 1 station. Some of the weak detections require more investigation to determine if they are actual detections. We have found 18 weak event detections False positives are the detections generated by the system which after inspection don't resemble events at all. In total, we have found 5 false positives. In the sections below we show examples of each category. In Figure 7.3.1 a complete overview of all detection can be found. Plots of all detections can be found in Appendix E.

Known detections	12
Strong event detections	22
Weak event detections	18
False positives	5
Total number of detections	57

Table 7.1: Summary of detections

7.2.1. Known Events

Events 47-58 occurred between January and June 2018 and all of them have been detected. As an example, we show events 47 and 56. Both of these events had very clear pre-shocks hidden in the seismic data. These pre shocks are found in the strong event section. Event 47 and 56 are also the largest events in the set, with magnitudes of 3.40 and 2.81 respectively.

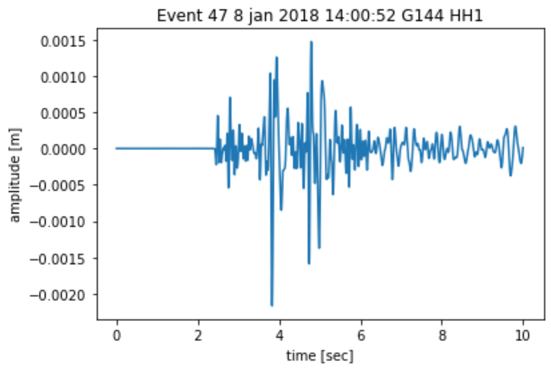


Figure 7.1: Event 47

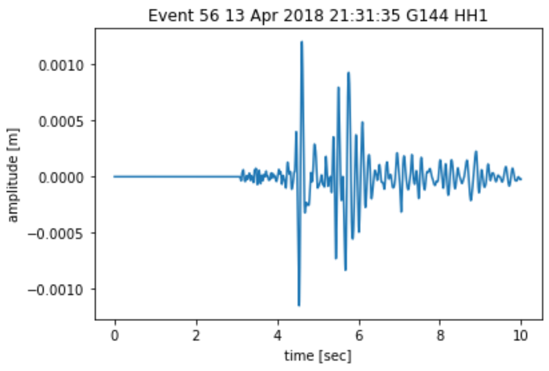


Figure 7.2: Event 56

7.2.2. Strong New Events

Looking at these two detections, one can see the resemblance to events 47 and 56. The pre-shock belonging to event 56 occurred only six hours before event 56.

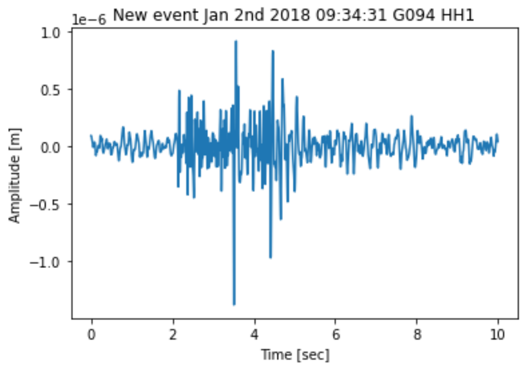


Figure 7.3: Pre shock event 47

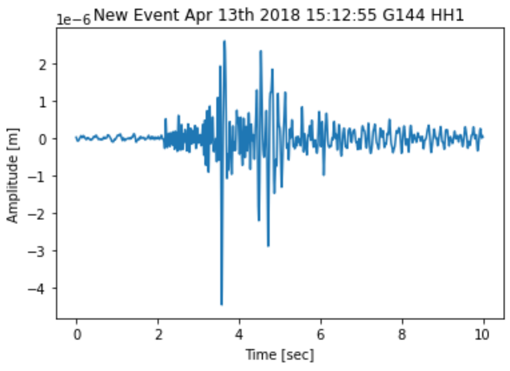


Figure 7.4: Pre shock event 56

7.2.3. Weak New Events

For weak detections, it’s sometimes hard to determine whether or not something is an event. For example, two weak detections are shown.

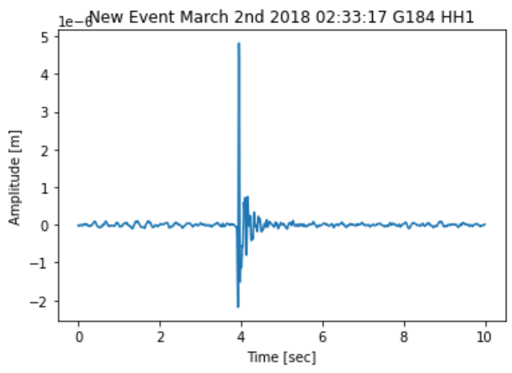


Figure 7.5: Weak event detection March 2nd

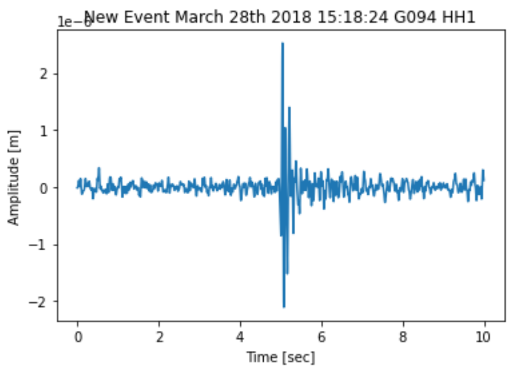


Figure 7.6: Weak event detection March 28th

7.2.4. False Positives

Very few false positives have been found. Two examples can be found here.

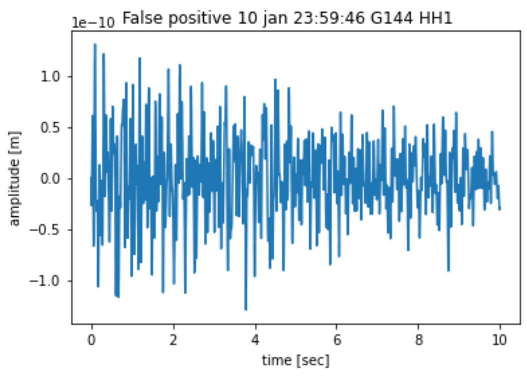


Figure 7.7: False positive January 10th

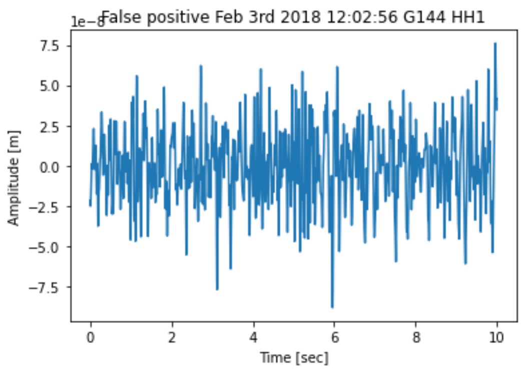


Figure 7.8: False positive February 3rd

7.2.5. How Do these Results Compare to the Gutenberg-Richter Analysis?

Based on the Gutenberg-Richter analysis we expect 60 undetected events between -1 and 0.5M in six months of seismic recordings. We have found 22 strong detections with estimated magnitudes between 0.56 and -0.74M. In the table below, the number of new strong detections for different magnitude ranges and the number of expected events based on Gutenberg-Richter can be found. The catalogue should be complete up to 0.5M, so it seems like the estimated magnitudes are too high. It looks like the estimated magnitudes are too high for the 0-0.5M range as well. We assume that the observed surplus belongs to lower magnitude tranches. This leads to the following conclusions: all events expected between 0 and 0.5M have been found, we found around half of the events between -0.5 and 0M (between 7/17 and 11/17) and for events under -0.5M we only found between 10 and 20 percent(ranging between 3/35 and 7/35).

Magnitude range	Number of new strong event detections	Number of expected events based on Gutenberg Richter
>0.5M	3	0
0 - 0.5	12	8
-0.5 - 0	4	17
-1 - -0.5	3	35

Table 7.2: Comparison new strong event detections and Gutenberg-Richter

7.3. Overview of All Detections and Estimated Magnitudes

At the end of this section, a large table listing all detections in order of occurrence is presented. We also provided the new detections with an estimated magnitude based on the magnitudes of known events. We also indicated how many windows have been detected, and how many windows the earthquake was visible.

7.3.1. Magnitude Estimation

We have estimated the magnitude of the detected events by looking at the maximum amplitudes of known events. We used the maximum amplitude of all events above 1M for the HH1 component. We made a scatter-plot of the relation between maximum amplitude and magnitude. We fitted this scatter plot to find the relation between magnitude and maximum amplitude. We fit Equation 7.3.1 where m represents the magnitude and a amplitude. We solved for b using least squares, b was found to be $8.23 * 10^{-7}$. Using this relation and the amplitudes of the new detections it was possible to estimate their magnitude.

$$a = b * 10^{m} \tag{7.3.1}$$

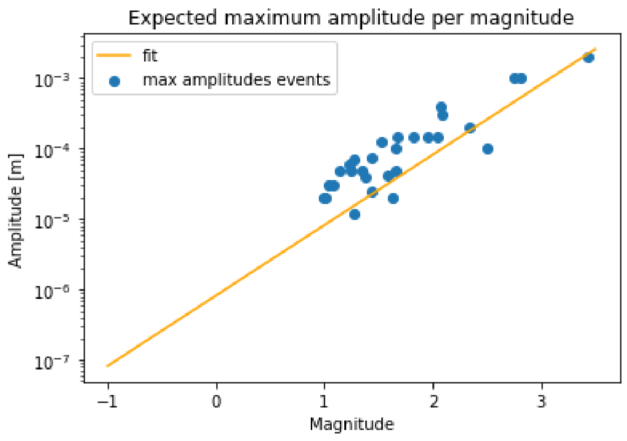


Figure 7.9: Fit for maximum amplitudes

Detection #	Date	Time	Type	Magnitude	Estimated magnitude	Screens detected	Screens visible	Max ccc
1	1/2/18	9:34:31	New strong event		0.23	6	9	0.827
2	1/8/18	14:00:52	Event 47	3.4		9	9	1
3	1/8/18	16:58:16	New strong event		0.56	7	9	0.795
4	1/9/18	15:46:49	Event 48	0.68		8	9	0.848
5	1/10/18	23:59:46	False positive			1		0.921
6	1/17/18	0:38:44	False positive			1		0.626
7	1/21/18	6:06:27	False positive			1		0.712
8	2/3/18	12:02:56	False positive			1		0.619
9	2/5/18	23:49:59	Event 49	0.44		3	9	0.835
10	2/8/18	15:25:30	Event 50	2.04		9	9	1
11	2/12/18	0:05:19	New strong event		0.56	3	9	0.889
12	2/12/18	1:15:22	New strong event		-0.31	1	6	0.651
13	2/12/18	10:44:53	Event 51	0.92		5	9	0.782
14	2/16/18	14:56:49	New strong event		0.39	6	6	0.962
15	2/17/18	15:30:23	New weak event		0.73	1	3	0.601
16	2/18/18	1:26:54	Event 52	1		9	9	1
17	2/20/18	21:58:25	New strong event		0.39	2	3	0.752
18	2/22/18	11:20:23	New weak event		-0.91	1	3	0.627
19	2/25/18	7:13:07	New strong event		-0.61	2	6	0.725
20	2/28/18	5:14:17	New strong event		0.08	1	6	0.672
21	2/28/18	5:47:23	New weak event		0.39	1	3	0.657
22	2/28/18	16:22:59	New weak event		0.69	1	3	0.636
23	2/28/18	18:03:49	New weak event		0.23	1	3	0.646
24	3/1/18	18:16:33	New weak event		0.26	1	3	0.603
25	3/1/18	21:34:26	Event 53	0.92		2	9	0.702
26	3/2/18	0:02:12	New weak event		-1.52	1	3	0.621
27	3/2/18	2:33:17	New weak event		0.78	1	3	0.619
28	3/2/18	21:33:07	New weak event		0.18	1	3	0.631
29	3/3/18	2:38:23	New weak event		0.26	1	3	0.603
30	3/3/18	19:57:57	New strong event		0.26	1	6	0.655
31	3/18/18	17:02:08	New strong event		-0.012	2	6	0.662
32	3/25/18	14:29:28	New strong event		0.26	1	7	0.623
33	3/26/18	8:31:21	New weak event		0.48	1	1	0.634
34	3/28/18	15:18:24	New weak event		0.39	1	3	0.618
35	3/30/18	4:18:06	New weak event		0.26	1	3	0.634
36	3/31/18	0:12:05	New weak event		-0.61	1	2	0.602
37	4/1/18	3:13:08	Event 54	0.61		4	9	0.856
38	4/3/18	18:46:00	New strong event		0.18	2	6	0.804
39	4/4/18	3:35:41	Event 55	0.9		6	9	0.874
40	4/13/18	15:12:55	New strong event		0.69	9	9	0.834
41	4/13/18	21:31:35	Event 56	2.81		9	9	1
42	4/26/18	0:04:08	New weak event		-1.43	2	3	0.628
43	4/27/18	6:02:54	New strong event		-0.012	2	3	0.789
44	5/12/18	16:06:47	Event 57	0.72		5	9	0.948
45	5/13/18	17:35:12	New strong event		-0.012	3	6	0.918
46	5/17/18	6:11:24	New strong event		0.52	3	9	0.724
47	5/18/18	16:29:54	New weak event		-0.43	1	2	0.616
48	5/21/18	23:27:49	Event 58	1.63		9	9	1
49	5/22/18	0:19:44	New strong event		-0.14	1	4	0.701
50	5/25/18	12:10:46	New weak event		0.39	1	3	0.614
51	5/25/18	23:02:55	New strong event		0.39	2	6	0.668
52	5/26/19	4:22:42	New weak event		-0.61	1	3	0.625
53	5/27/19	3:01:48	New strong event		-0.56	1	6	0.745
54	5/29/18	20:36:54	New strong event		0.39	2	9	0.804
55	6/4/18	23:36:11	New strong event		-0.74	3	6	0.795
56	6/19/18	4:46:44	New strong event		0.18	2	6	0.75
57	6/28/18	18:44:30	False positive			1		0.626

Table 7.3: Overview all detections in chronically order

8 Discussion and Outlook

In this section, we discuss some of the simplifications and uncertainties surrounding parts of the work presented in this thesis. We also discuss how the results can be improved.

8.1. Forward Modelling Events

During forward modelling, two big simplifications were made. First of all, it was chosen to use average values for the velocity model. It would have been more realistic to divide layers without constant velocities into small layers with constant velocity. It was expected that this simplification would not have a big influence on the performance of the neural network, since it does not learn the training data by heart.
The second simplification is that we modelled the earthquakes as if their moment tensor only consisted out of a Double Coupled (DC) component. In reality, they also had Isotropic and Compensated Linear Vector Dipole (CLVD) components. These were omitted to make it easier to model the synthetic events. Again we justified this simplification because the network only has to 'learn' the general concept of an event. The double coupled component was the strongest component for the Zeerijp earthquakes. We have seen that the main weakness of the neural network is the number of false positives it produces. We might be able to reduce this by using more realistic synthetics so that the network can 'learn' more features that set events apart from seismic noise.

8.2. Training Sets Neural Network

We have used three different training sets for the neural networks. These sets either differed in what dominant frequency was used for generating synthetic events or in size. However, we haven't looked at optimal screen length. We started with a length of 10 seconds, and this worked well. These 10 seconds were based on template matching. There we saw that 10 seconds provided a high cross-correlation coefficient without having too many false positives. It would be interesting to see what would happen when using different interval lengths, so we can determine which part of the earthquake best describes its essence.

8.3. Used Neural Network Architectures

We have tried more architectures than presented in this thesis, however, we haven't tried every single architecture that showed potential. For example, we haven't tried a recurrent neural network which on paper should have been a good option for this type of problem.The network presented in this thesis as the best network for earthquake detection is by no means the absolute best network for this problem. It is the best relative to the other networks presented in this thesis.

8.4. Size of Noise Test Set Template Matching

Because of memory problems the noise sets on which template matching was tested only contained 6300 screens of noise. These memory problems were caused by using all templates on every input screen. Instead of using only the templates belonging to the same station and channel as the input screen. In a later stage, this was changed.

8.5. Categorizing Detections Made by Template Matching

The way the detections made by template matching are categorized is partly subjective, since we look at visibility. The goal of presenting them like this is for the reader to get a clear overview of what kind of detections the algorithm can make.

8.6. Improving False Positive Rate Neural Network

We could improve the false positive rate by using cross-correlation. We could use the template matching algorithm with a lower threshold value on all of the detections made by the neural network. By first screening the detections with template matching, we hopefully can reduce the number of false positives. We could do the same as Perol et al. [12] and cross-correlate all the detections made by the neural network with each other. The idea is that the undetected events are similar to each other because they are from the same area and probably have the same focal mechanism. We then would need to find a suitable threshold value, which produces a selection with few false positives.

A different approach would be to train two networks instead of one. We could have one network for the horizontal components (HH1 and HH2) and one for the horizontal component(HHZ). Due to the nature of P and S waves, the recordings made for the horizontal and vertical components of the same event are quite different. By combining both components in one network we might have confused the network with these different representations. The idea is when we have separate networks the concept will be easier to grasp. In this way, we can reduce the number of false positives.

Finally one could try and improve the training data. We could try using a bigger set. We have seen that using a bigger training set reduced the number of false positives. In this chapter, we already mentioned that improving the forward modelling process could improve results and changing the screen length might also help. These proposals could also be used in combination which each other.

9 Conclusion

We started this thesis with the following questions:

- Is it possible to create a practical detection system for low magnitude events(< 0.5M) using template matching or a neural network for the Zeerijp area?
- How do the developed algorithms compare when used for low magnitude events recorded in the Zeerijp area?

We have tried to answer these questions by developing algorithms based on template matching and neural networks. We assessed these algorithms by testing the ability of each system to differentiate between events of different noise levels and actual seismic noise. It was concluded that template matching was the most practical method to look for earthquakes in continuous seismic data. Finally, six months of seismic recordings were fed to the template matching algorithm.

Is it possible to create a practical detection system for low magnitude events(< 0.5M) using template matching or a neural network for the Zeerijp area?
The short answer is yes, it was possible to create such a system using template matching. The developed algorithm was able to detect at least 22 previously undetected events. We have found that template-matching has detected all expected events for magnitudes between 0 and 0.5M. For magnitudes <-0.5M template matching did not detect all expected events. For events between -0.5 and 0M template matching found around half of the expected events. For magnitudes between -1 and -0.5M, the algorithm only detected between 10 and 20 percent of expected events. We have used a Gutenberg-Richter analysis to calculate the number of expected events.
Whether it is possible to create a practical detection system using Neural networks is more complicated. As far as this thesis goes the answer is no. The neural network presented in this thesis produces too many false positives from a practical point of view. The false-positive rate is only 0.29 percent but is still too high for using it on months of seismic recordings. We have proposed several ways to improve the network. One could combine the neural network with template matching or other cross-correlation techniques. You could also create two different neural networks one trained for the horizontal channels (HH1 and HH2) and another trained for the vertical channel (HHZ). Or one could try to create better training data by improving the realism of the data or by experimenting with input length.

How do the developed algorithms compare when used for low magnitude events recorded in the Zeerijp area? Template matching was found to be affected more by noise than the neural network. However, it does make very few false positives. On the other hand it is very computational heavy. The Neural-network is less sensitive to noise. This indicates that it can detect lower magnitude events than template matching. It is also more computationally efficient than template matching. Its only drawback is that it produces too many false positives.
It's worth trying to reduce the number of false positives if one is interested in detecting events below -0.5M or wants a method that is less computational heavy. If you are only interested in detecting events up to -0.5M and don't care about computational costs the template matching algorithm discussed in this thesis will do nicely.

Bibliography

[1] Groningen gasfield:https://www.nlog.nl/en/groningen-gasfield, Nlog.

[2] B. Dost, E. Ruigrok, J. Spetzler, Development of seismicity and probabilistic hazard assessment for the Groningen gas field, volume 96, Cambridge University Press, 2017.

[3] R. Selles, Meer dan 100.000 schademeldingen afgehandeld, ???? URL: `https://www.schadedoormijnbouw.nl/nieuws/meer-dan-100000-schademeldingen-afgehandeld`.

[4] R. Selles, Schade en geografische afstand: https://www.schadedoormijnbouw.nl/schade-gebouwen-objecten/schade-beoordelen/schade-en-geografische-afstand, Insituut Mijnbouwschade Groningen.

[5] NAM, Seismic hazard and risk assessment groningen field update for production profile, 2020.

[6] L. Jagt, E. Ruigrok, H. Paulssen, Relocation of clustered earthquakes in the groningen gas field, Netherlands Journal of Geosciences 96 (2017).

[7] S. J. Gibbons, F. Ringdal, The detection of low magnitude seismic events using array-based waveform correlation, Geophysical Journal International 165 (2006) 149–166.

[8] M. Vasterling, U. Wegler, J. Becker, A. Brüstle, M. Bischoff, Real-time envelope cross-correlation detector: application to induced seismicity in the Insheim and Landau deep geothermal reservoirs, volume 21, 2016.

[9] B. P. Goertz-Allmann, D. Kühn, V. Oye, B. Bohloli, E. Aker, Combining microseismic and geomechanical observations to interpret storage integrity at the In Salah CCS site, volume 198, 2014. doi:`10.1093/gji/ggu010`.

[10] D. A. Dodge, D. B. Harris, Large-scale test of dynamic correlation processors: Implications for correlation-based seismic pipelines, Bulletin of the Seismological Society of America 106 (2016) 435–452.

[11] X. Zhang, J. Zhang, C. Yuan, S. Liu, Z. Chen, W. Li, Locating induced earthquakes with a network of seismic stations in oklahoma via a deep learning method, Scientific Reports 10 (2020).

[12] T. Perol, M. Gharbi, M. Denolle, Convolutional neural network for earthquake detection and location, Science Advances 4 (2018).

[13] J. Havskov, O. L., Routine data processing in earthquake seismology: With sample data, exercises and software, Springer, 2010.

[14] M. A.Nielsen, Neural networks and deep learning, Determination press, 2015. URL: `http://neuralnetworksanddeeplearning.com`.

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

[16] A. Hidaka, T. Kurita, Consecutive Dimensionality Reduction by Canonical Correlation Analysis for Visualization of Convolutional Neural Networks, volume 2017, 2017.

[17] H. Yingge, I. Ali, K.-Y. Lee, Deep Neural Networks on Chip - A Survey, 2020.

[18] J. Brownlee, How to identify overfitting machine learning models in scikit-learn, 2020. URL: `https://machinelearningmastery.com/overfitting-machine-learning-models/`.

[19] J. Brownlee, 1d convolutional neural network models for human activity recognition, Machine Learning Mastery (2020). URL: `https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/`.

[20] B. L. N. Kennett, Seismic Wave Propagation in Stratified Media, ANU Press, 2009.

[21] B. Dost, A. van Stiphout, D. Kühn, M. Kortekaas, E. Ruigrok, S. Heimann, Probabilistic Moment Tensor Inversion for Hydrocarbon-Induced Seismicity in the Groningen Gas Field, the Netherlands, Part 2: Application, volume 110, 2020.

[22] R. J. Geller, C. S. Mueller, Four similar earthquakes in central california, Geophysical Research Letters 7 (1980) 821–824.

[23] J. Mohajon, Confusion matrix for your multi-class machine learning model, 2021. URL: `https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model`.

I

Appendices

A

Appendix A

In this appendix an overview of seismic noise and events used for the development of the algorithms is shown. Please note that the dates are American (mm/dd/yy).

Usage	Recorded on
Noise used for fine tuning templates	9/2/17
Noise added to syntehtic seismics	1/7/19
Noise used for training architecures	8/6/17
Noise used for testing architecures and training sets	9/6/19
Noise used for comparing template matching and neural net	11/20/17

Table A.1: Noise usage

A.1. Overview noise

A.2. Overview of events recorded in Zeerijp between 2015-2019

eventno	eventID	originTime(UTC)	latitude	longitude	depth	magnitude	description
1	event_1	2/10/15	53.3487	6.72833	3	0.911213	Loppersum
2	event_2	5/7/15	53.3468	6.74333	3	1.01505	Zeerijp
3	event_3	6/7/15	53.34	6.75	3	1.91405	Zeerijp
4	event_4	6/10/15	53.344	6.753	3	1.75534	Zeerijp
5	event_5	6/10/15	53.343	6.713	3	0.41634	Westeremden
6	event_6	6/10/15	53.345	6.713	3	0.800359	Westeremden
7	event_7	7/4/15	53.351	6.694	3	1.07698	Westeremden
8	event_8	7/16/15	53.341	6.715	3	0.428682	Westeremden
9	event_9	8/7/15	53.339	6.759	3	0.802166	Loppersum
10	event_10	8/17/15	53.35	6.761	3	0.600928	Zeerijp
11	event_11	9/10/15	53.372	6.713	3	0.844236	Startenhuizen Lopp
12	event_12	11/16/15	53.357	6.743	3	0.856557	Zeerijp
13	event_13	1/19/16	53.362	6.77	3	0.611921	’t Zandt
14	event_14	2/21/16	53.358	6.739	3	0.197729	Zeerijp
15	event_15	3/1/16	53.328	6.714	3	0.334155	Loppersum
16	event_16	5/6/16	53.342	6.694	3	0.68065	Westeremden
17	event_17	6/1/16	53.361	6.75	3	1.24217	Zeerijp
18	event_18	6/18/16	53.345	6.799	3	0.127466	Leermens
19	event_19	7/18/16	53.378	6.709	3	1.6623	Eppenhuizen
20	event_20	8/24/16	53.372	6.724	3	0.597877	Garsthuizen
21	event_21	10/2/16	53.328	6.779	3	0.32784	Wirdum
22	event_22	11/8/16	53.331	6.795	3	1.44056	Eenum
23	event_23	11/8/16	53.329	6.794	3	0.853776	Eenum
24	event_24	12/7/16	53.333	6.774	3	1.79145	Loppersum
25	event_25	1/19/17	53.327	6.774	3	0.92482	Wirdum
26	event_26	2/4/17	53.371	6.739	3	0.90378	Garsthuizen
27	event_27	2/4/17	53.371	6.736	3	0.603355	Garsthuizen
28	event_28	2/12/17	53.381	6.713	3	1.26969	Startenhuizen
29	event_29	2/14/17	53.371	6.718	3	0.825201	Garsthuizen
30	event_30	2/15/17	53.377	6.714	3	1.58658	Startenhuizen
31	event_31	2/19/17	53.393	6.758	3	1.41208	Zijldijk
32	event_32	2/26/17	53.328	6.778	3	1.44401	Wirdum
33	event_33	3/11/17	53.35	6.761	3	2.08334	Zeerijp
34	event_34	7/10/17	53.332	6.741	3	0.666265	Loppersum
35	event_35	7/25/17	53.352	6.737	3	0.960307	Loppersum
36	event_36	8/13/17	53.358	6.743	3	0.465839	Zeerijp
37	event_37	8/14/17	53.367	6.742	3	1.23084	Garsthuizen
38	event_38	8/14/17	53.36	6.742	3	0.274475	Zeerijp
39	event_39	9/14/17	53.344	6.741	3	0.549283	Zeerijp
40	event_40	12/1/17	53.36	6.736	3	1.67847	Zeerijp
41	event_41	12/1/17	53.357	6.757	3	1.27785	Zeerijp
42	event_42	12/7/17	53.386	6.784	3	1.82144	’t Zandt
43	event_43	12/10/17	53.37	6.765	3	2.06669	’t Zandt
44	event_44	12/22/17	53.366	6.752	3	1.66058	Zeerijp
45	event_45	12/22/17	53.357	6.75	3	0.358491	Zeerijp
46	event_46	12/30/17	53.357	6.756	3	1.43183	Zeerijp
47	event_47	1/8/18	53.363	6.751	3	3.43469	Zeerijp
48	event_48	1/9/18	53.355	6.752	3	0.680083	Zeerijp
49	event_49	2/6/18	53.336	6.762	3	0.437586	Loppersum

Table A.2: Overview of events 2015-2019

eventno	eventID	originTime(UTC)	latitude	longitude	depth	magnitude	description
50	event_50	2/8/18	53.335	6.751	3	2.03768	Loppersum
51	event_51	2/12/18	53.371	6.714	3	0.922629	Garsthuizen
52	event_52	2/18/18	53.335	6.763	3	1.00086	Loppersum
53	event_53	3/1/18	53.326	6.76	3	0.916779	Loppersum
54	event_54	4/1/18	53.35	6.741	3	0.619502	Zeerijp
55	event_55	4/4/18	53.334	6.75	3	0.901809	Loppersum
56	event_56	4/13/18	53.371	6.75	3	2.81124	Garsthuizen
57	event_57	5/12/18	53.355	6.764	3	0.7164	Zeerijp
58	event_58	5/22/18	53.374	6.795	3	1.63178	’t Zandt
59	event_59	7/14/18	53.361	6.744	3	0.272782	Zeerijp
60	event_60	7/29/18	53.373	6.793	3	0.883042	’t Zandt
61	event_61	8/22/18	53.378	6.754	3	0.502138	’t Zandt
62	event_62	9/17/18	53.381	6.713	3	0.59096	Startenhuizen
63	event_63	10/10/18	53.347	6.734	3	0.53691	Loppersum
64	event_64	10/10/18	53.347	6.732	3	0.538023	Loppersum
65	event_65	11/25/18	53.385	6.716	3	1.57888	Eppenhuizen
66	event_66	2/7/19	53.35	6.714	3	0.698895	Westeremden
67	event_67	2/9/19	53.38	6.754	3	0.882142	’t Zandt
68	event_68	2/16/19	53.36	6.743	3	1.37385	Zeerijp
69	event_69	2/24/19	53.339	6.761	3	0.396396	Loppersum
70	event_70	3/12/19	53.367	6.765	3	0.390296	’t Zandt
71	event_71	4/21/19	53.341	6.757	3	0.901397	Zeerijp
72	event_72	5/3/19	53.33	6.711	3	1.00545	Loppersum
73	event_73	5/28/19	53.351	6.738	3	1.14094	Zeerijp
74	event_74	9/9/19	53.343	6.754	3	1.52814	Zeerijp
75	event_75	3/17/20	53.371	6.73	3	1.35541	Garsthuizen
76	event_76	4/28/20	53.356	6.762	3	1.533	Zeerijp
77	event_77	5/2/20	53.388	6.762	3	2.50787	Zijldijk
78	event_78	5/31/20	53.372	6.734	3	0.643497	Garsthuizen
79	event_79	7/3/20	53.335	6.749	3	0.675347	Loppersum
80	event_80	7/14/20	53.342	6.729	3	2.74944	Loppersum
81	event_81	7/15/20	53.342	6.731	3	0.559852	Loppersum
82	event_82	7/19/20	53.376	6.713	3	2.33607	Startenhuizen
83	event_83	7/19/20	53.376	6.712	3	0.901401	Startenhuizen
84	event_84	9/3/20	53.338	6.703	3	1.96237	Westeremden
85	event_85	10/16/20	53.328	6.767	3	1.03638	Loppersum
86	event_86	11/16/20	53.371	6.731	3	1.08549	Garsthuizen

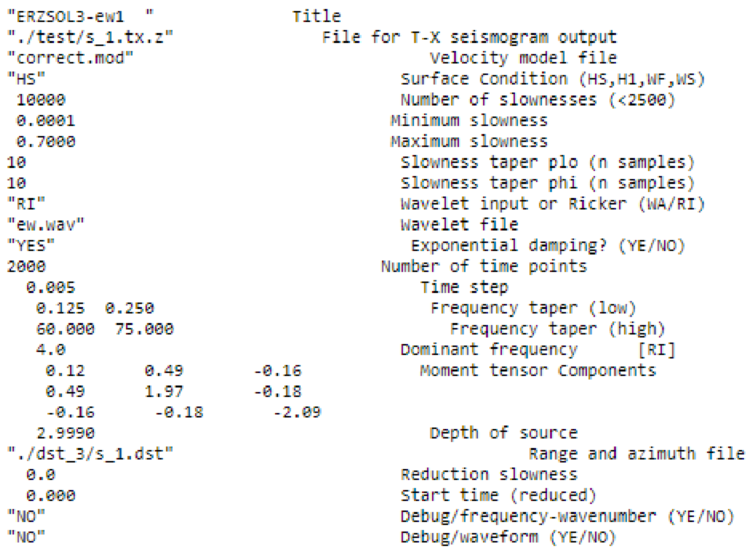
Table A.3: Overview events 2015-2019

B

Appendix B

In this appendix one can find an exmaple of a cmd and dst file. These files contain the input information for the ERZSOL3 software routine.

B.1. ERZSOL 3 Input Files



```
"ERZSOL3-ew1  "             Title
"./test/s_1.tx.z"              File for T-X seismogram output
"correct.mod"                             Velocity model file
"HS"                                   Surface Condition (HS,H1,WF,WS)
 10000                                 Number of slownesses (<2500)
 0.0001                               Minimum slowness
 0.7000                               Maximum slowness
10                                     Slowness taper plo (n samples)
10                                     Slowness taper phi (n samples)
"RI"                                   Wavelet input or Ricker (WA/RI)
"ew.wav"                               Wavelet file
"YES"                                   Exponential damping? (YE/NO)
2000                                  Number of time points
  0.005                                  Time step
   0.125  0.250                           Frequency taper (low)
   60.000  75.000                           Frequency taper (high)
   4.0                                 Dominant frequency      [RI]
    0.12      0.49      -0.16            Moment tensor Components
    0.49      1.97      -0.18
    -0.16      -0.18      -2.09
   2.9990                                 Depth of source
"./dst_3/s_1.dst"                                   Range and azimuth file
  0.0                                  Reduction slowness
  0.000                                Start time (reduced)
"NO"                                   Debug/frequency-wavenumber (YE/NO)
"NO"                                   Debug/waveform (YE/NO)
```

Figure B.1: An example of a .cmd file.

```
 3                             # of distances /distances
3.86      -86.25
4.51      12.19
7.27      -9.16
```

Figure B.2: An example of a .dst file.

C
Appendix C

C.1. Results Test Template Lenght and Threshold

In this appendix, all of the test results of different parameter settings for template matching can be found. Please note that we use false-positive screens instead of false-positive instances to better show the differences in performance for the different settings.

Template 10 seconds	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	17/18	13/18	3/18	**81%**
Screens of noise classified as events	1071/6300	12/6300	0/6300	0/6300	0/6300	0/6300	**97%**

Table C.1: Results templates 10 seconds

Template 10 seconds with 1 second of noise included	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	16/18	12/18	3/18	**79%**
Screens of noise classified as events	720/6300	8/6300	0/6300	0/6300	0/6300	0/6300	**98%**

Table C.2: Results templates 10 seconds including 1 sec of noise

Template 12.5 seconds	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	15/18	12/18	3/18	**78%**
Screens of noise classified as events	859/6300	8/6300	0/6300	0/6300	0/6300	0/6300	**98%**

Table C.3: Results templates 12.5 seconds

Templat 12.5 seconds with 1 second of noise included	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	17/18	13/18	3/18	**81%**
Screens of noise classified as events	204/6300	4/6300	0/6300	0/6300	0/6300	0/6300	**99%**

Table C.4: Results using a templates of 12.5 seconds with one second of noise included

Template 15 seconds	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	17/18	13/18	3/18	**81%**
Screens of noise classified as events	828/6300	11/6300	0/6300	0/6300	0/6300	0/6300	**98%**

Table C.5: Results templates 15 seconds

Template 15 seconds with 1 second of noise included	0.4	0.5	0.6	0.7	0.8	0.9	Average correct percentage
number of events detected	18/18	18/18	18/18	10/18	1/18	0/18	**60%**
Screens of noise classified as events	828/6300	11/6300	0/6300	0/6300	0/6300	0/6300	**99.6%**

Table C.6: Results using a templates of 15 seconds with one second of noise included

D

Appendix D

D.1. Training Curves and Confusion Matrixes of Tested Architectures

In this section, all confusion matrixes and training curves can be found. Please note that the networks have been trained on synthetics, for which they have high accuracy scores. The networks have been trained using screens but in the confusion matrixes, we show the performance on instances.

D.1.1. Network 1

Test set 1

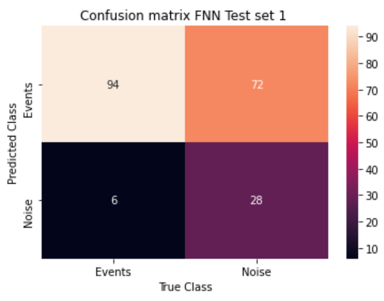


Figure D.1: Confusion matrix FNN training set 1

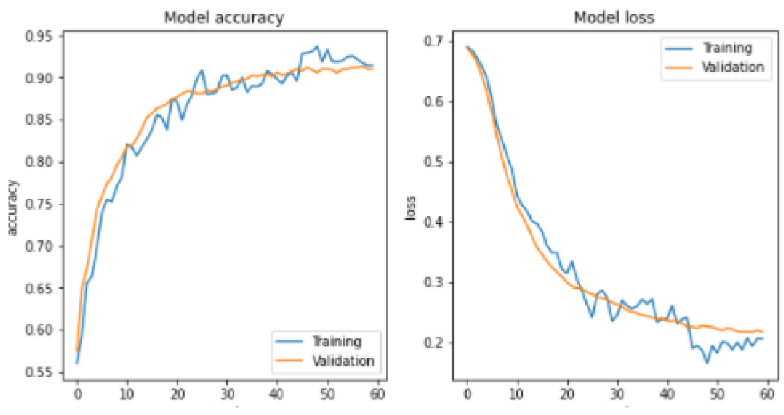


Figure D.2: Training curve FNN training set 1

Test set 2

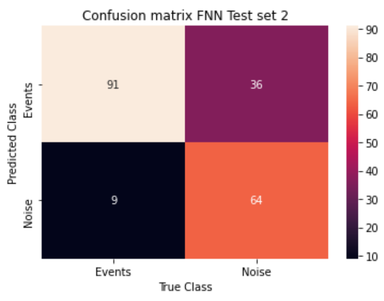


Figure D.3: Confusion matrix FNN training set 2

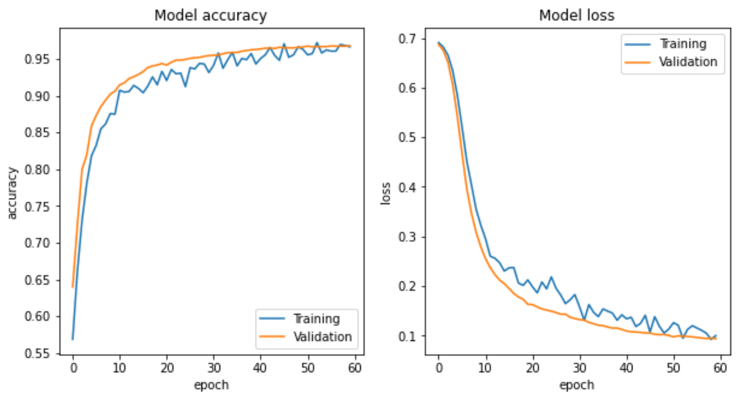


Figure D.4: Learning curve FNN training set 2

Test set 3

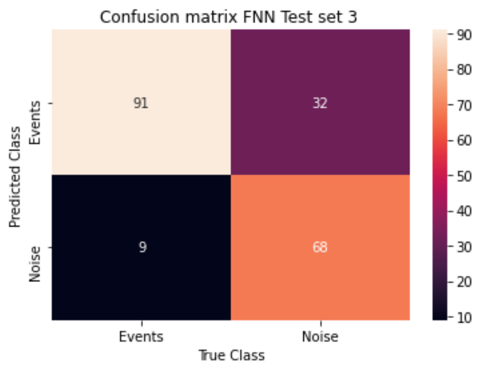


Figure D.5: Confusion matrix FNN training set 3

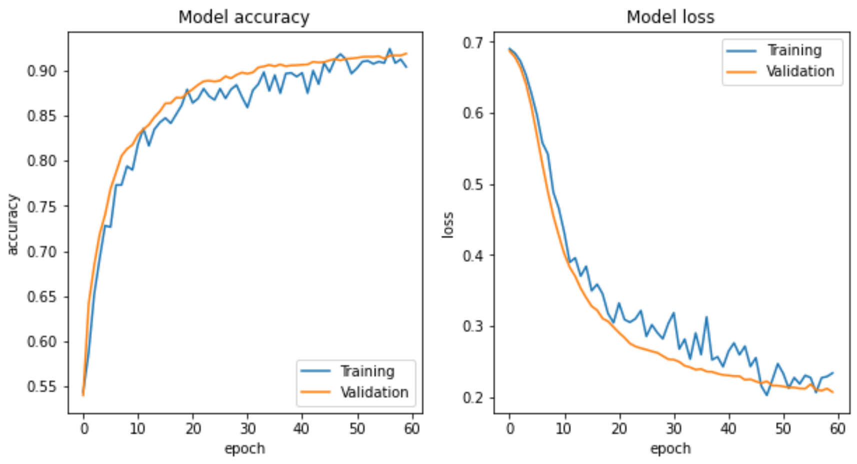


Figure D.6: Learning curve FNN training set 3

D.1.2. Network 2

Test set 1

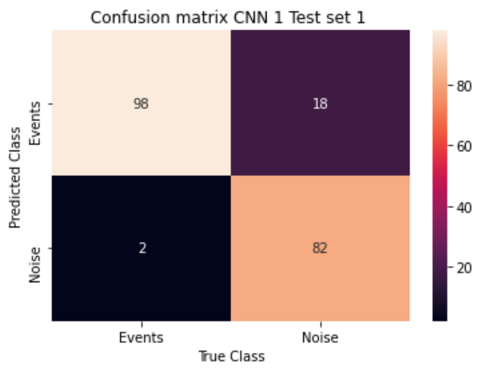


Figure D.7: Confusion matrix network 2 training set 1

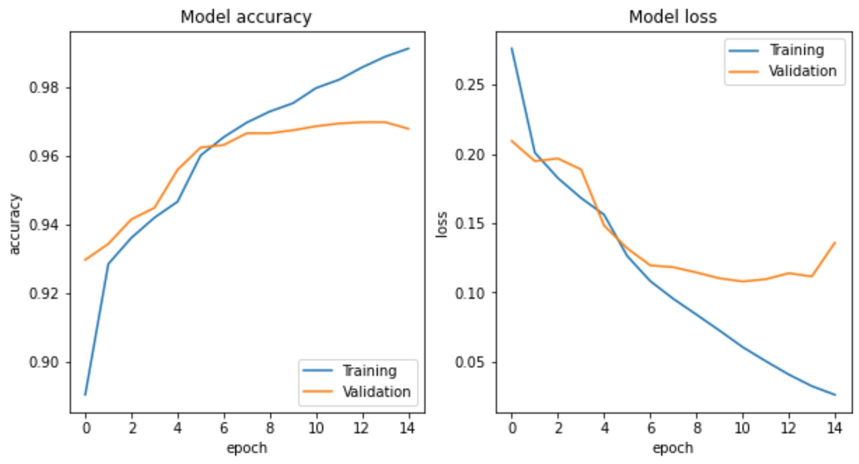


Figure D.8: Learning curve network 2 training set 1

Test set 2

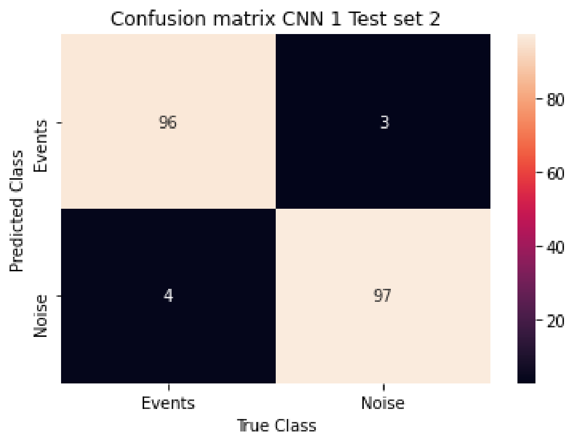


Figure D.9: Confusion matrix CNN 1 training set 2

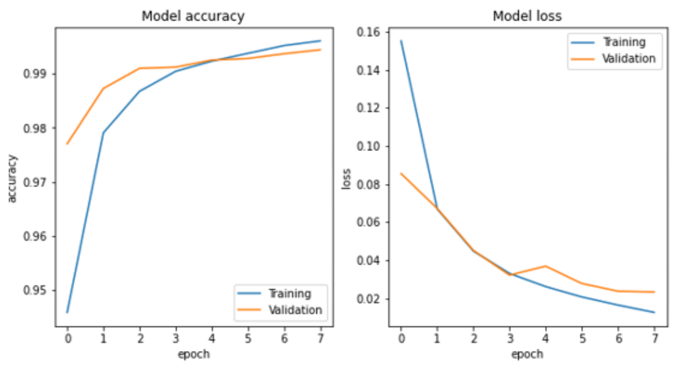


Figure D.10: Learning curve CNN 1 training set 2

Test set 3

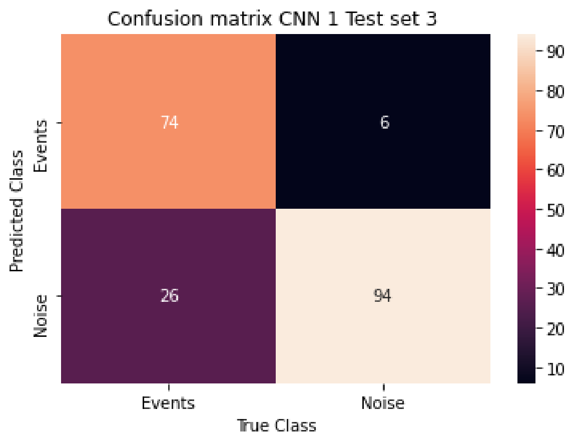


Figure D.11: Confusion matrix CNN 1 training set 3

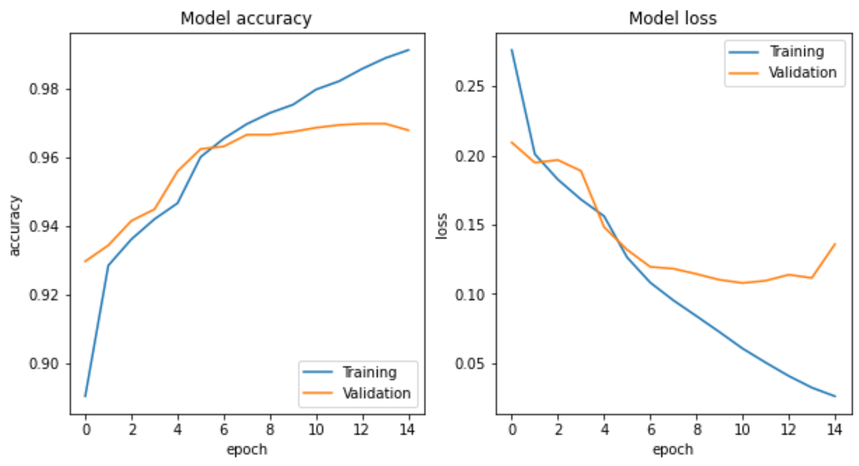


Figure D.12: Learning curve CNN 1 training set 3

D.1.3. Network 3

Test set 1

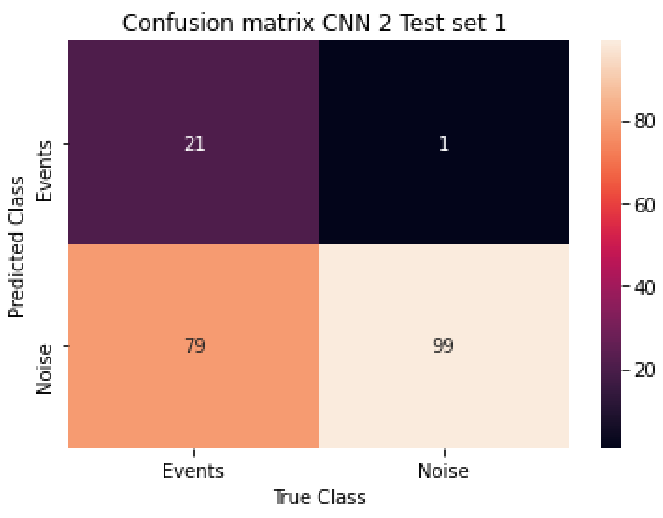


Figure D.13: Confusion matrix CNN 2 training set 1

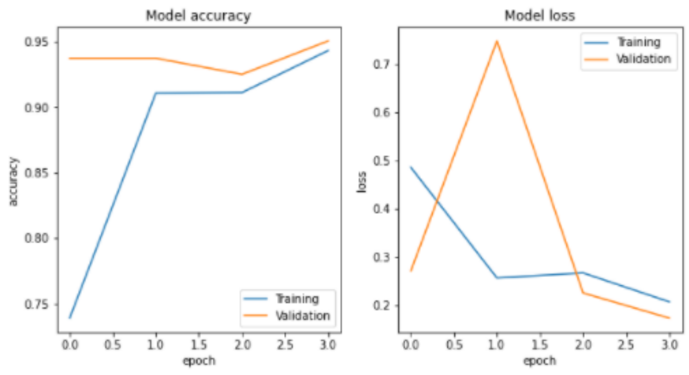


Figure D.14: Learning curve CNN 2 training set 1

Test set 2

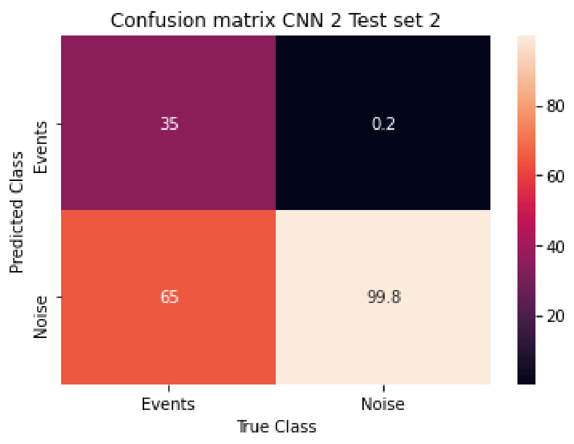


Figure D.15: Confusion matrix CNN 2 training set 2

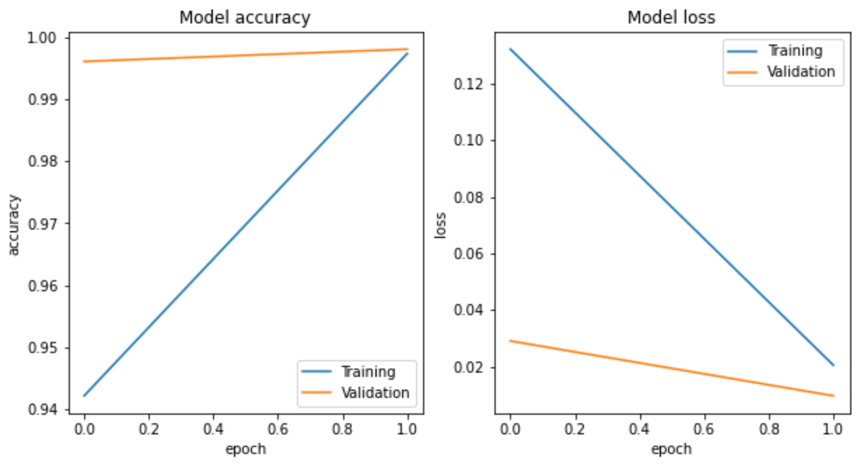


Figure D.16: Learning curve CNN 2 training set 2

Test set 3

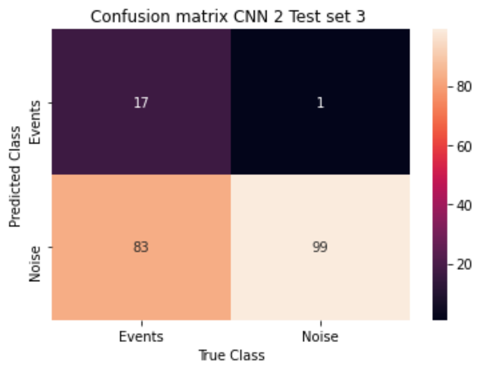


Figure D.17: Confusion matrix CNN 2 training set 2

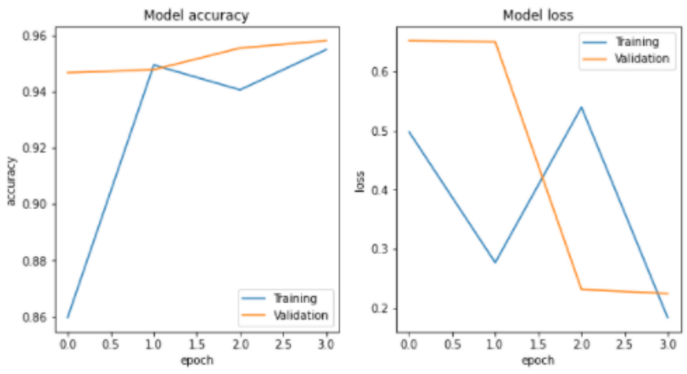


Figure D.18: Learning curve CNN 2 training set 2

E

Appendix E

E.1. Results Template Matching on Continous Seismic Recordings

Most of the detections are plotted for the same station channel combination. In this way, one can easily spot similarities. The default is station G144 channel HH1. We chose this pare based on the observations made in chapter 2. For some detections, this channel station combination doesn't show clear depictions of events. For those cases, another station channel combination has been used to show the detection. Please note that the unit of amplitude should be m/s and not m.

E.1.1. Strong Event Detections

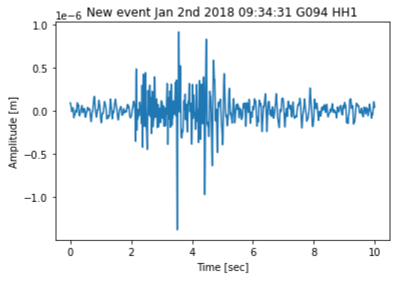


Figure E.1: New event January 2nd

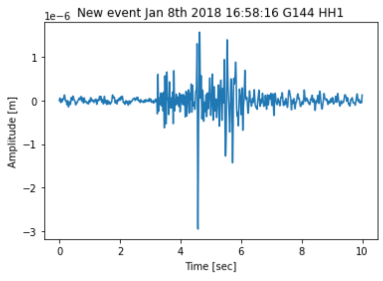


Figure E.2: New event January 8th

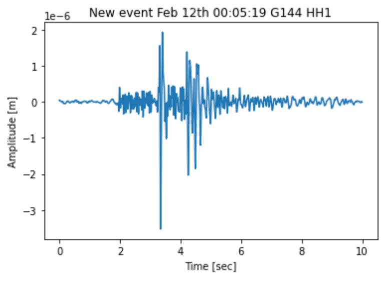


Figure E.3: New event February 12th

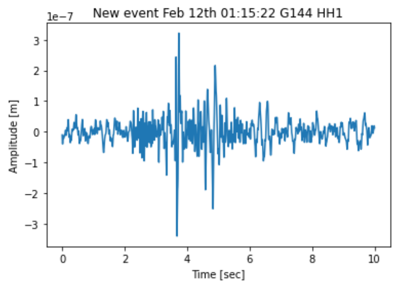


Figure E.4: New event February 12th 2

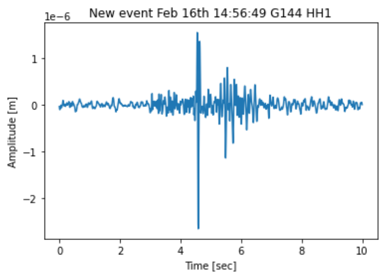


Figure E.5: New event February 16th

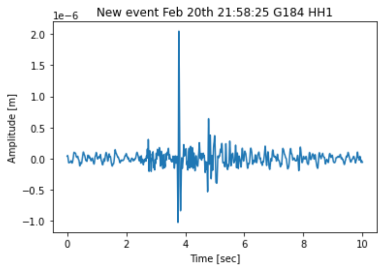


Figure E.6: New event February 20th

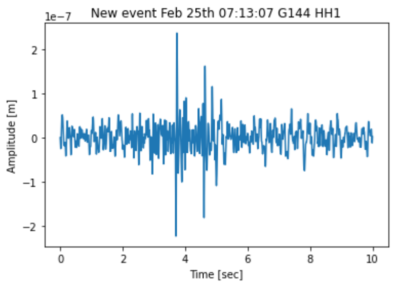


Figure E.7: New event February 25th

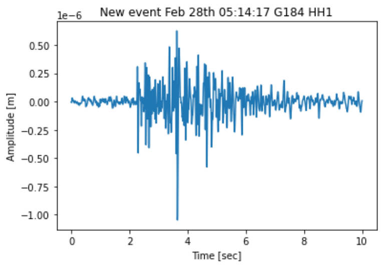


Figure E.8: New event February 28th

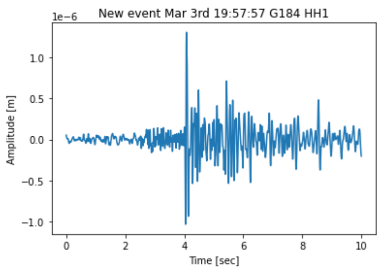


Figure E.9: New event March 3rd

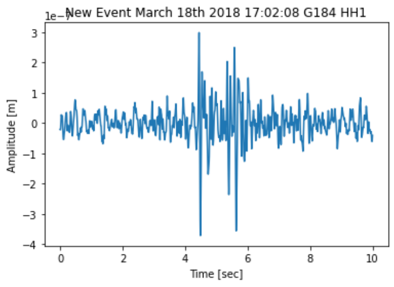


Figure E.10: New event March 18th

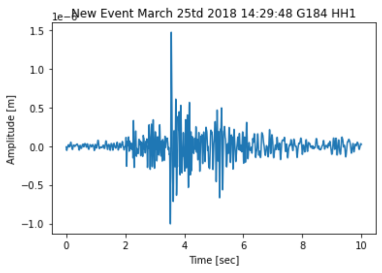


Figure E.11: New event March 25th

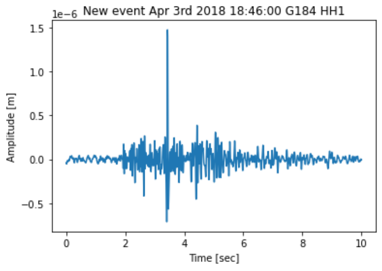


Figure E.12: New event April 3th

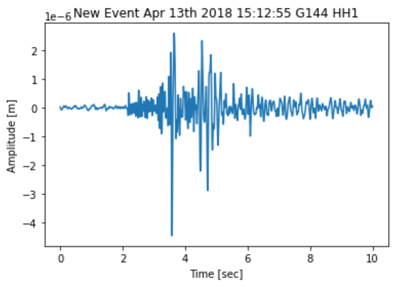


Figure E.13: New event April 13th

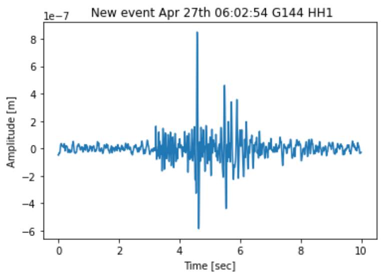


Figure E.14: New event April 27th

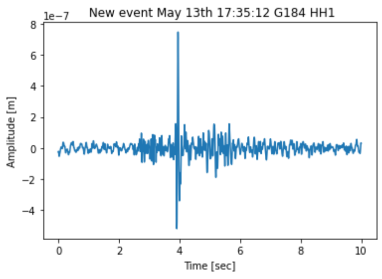


Figure E.15: New event May 13th

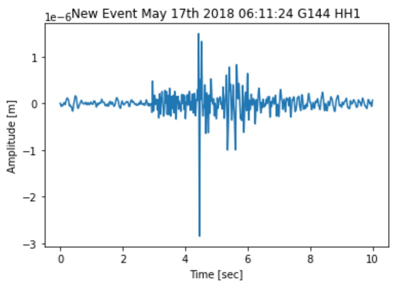


Figure E.16: New event May 17th

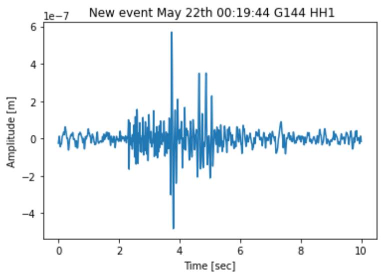


Figure E.17: New event May 22nd

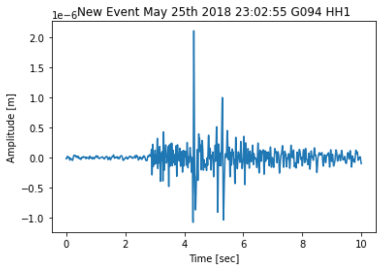


Figure E.18: New event May 25th

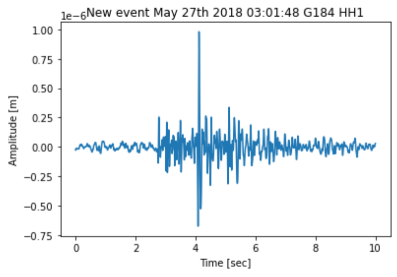


Figure E.19: New event May 27th

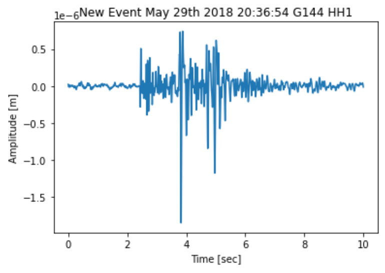


Figure E.20: New event May 29th

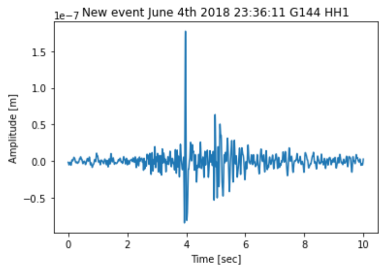


Figure E.21: New event June 4th

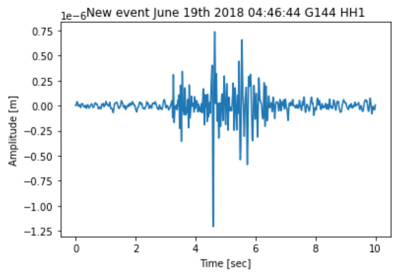


Figure E.22: New event June 29th

E.1.2. Weak event detections

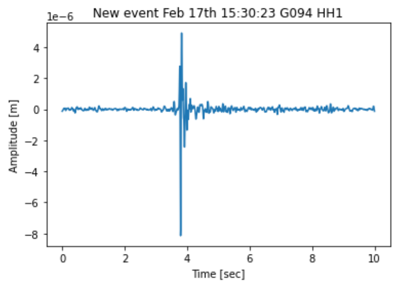


Figure E.23: New event February 17th

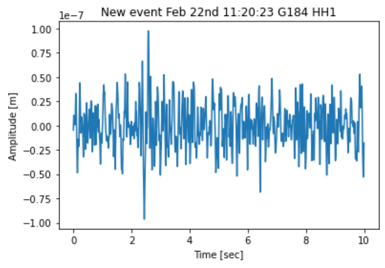


Figure E.24: New event February 22th

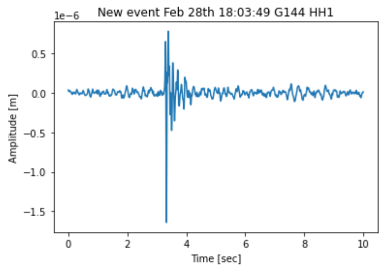


Figure E.25: New event February 28th (1)

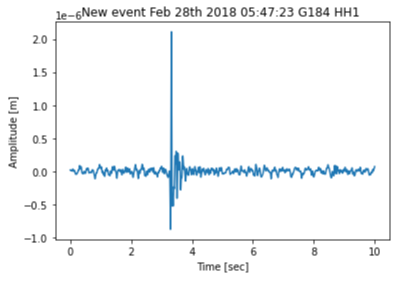


Figure E.26: New event February 28th (2)

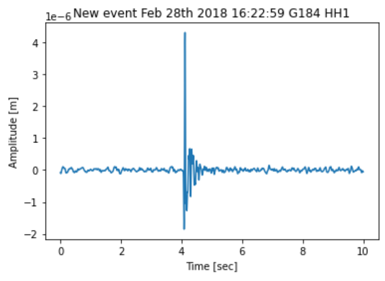


Figure E.27: New event February 28th(3)

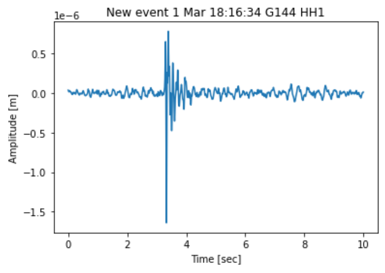


Figure E.28: New event March 1st

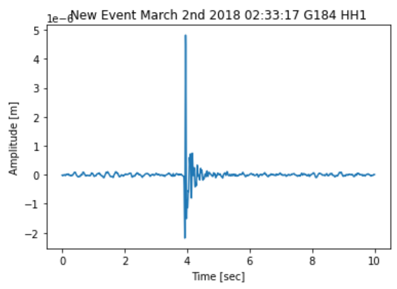


Figure E.29: New event March 2nd (2)

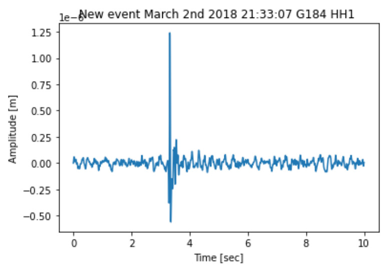


Figure E.30: New event March 2nd(3)

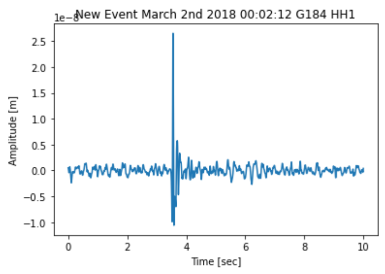


Figure E.31: New event March 2nd

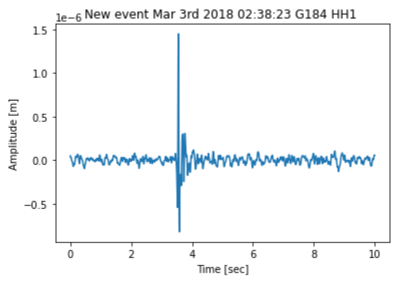


Figure E.32: New event March 3th

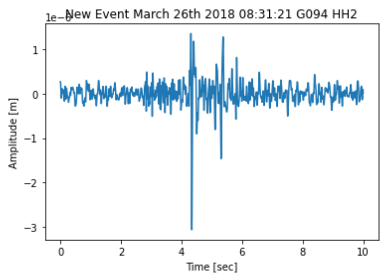


Figure E.33: New event March 26th

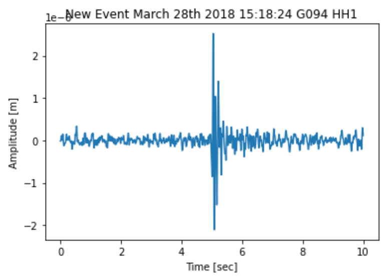


Figure E.34: New event March 28th

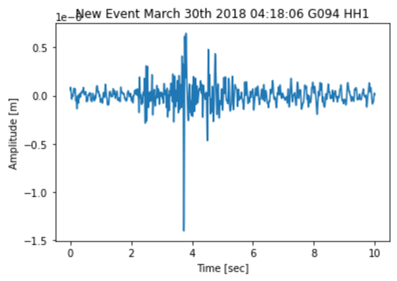


Figure E.35: New event March 30th

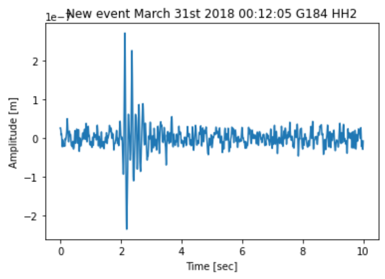


Figure E.36: New event March 31st

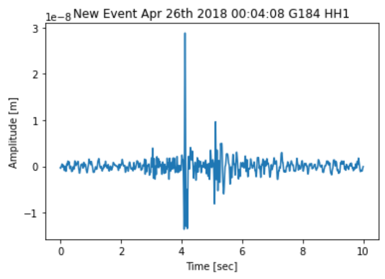


Figure E.37: New event April 26th

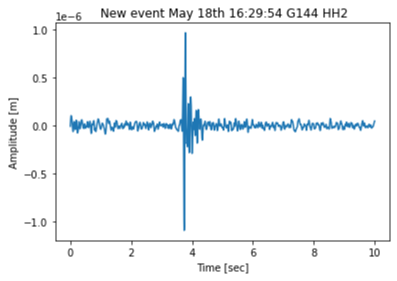


Figure E.38: New event May 18th

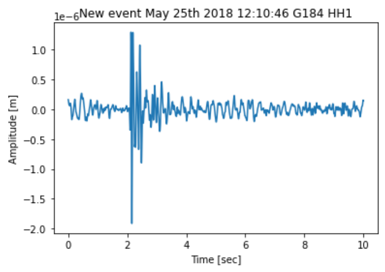


Figure E.39: New event May 25th

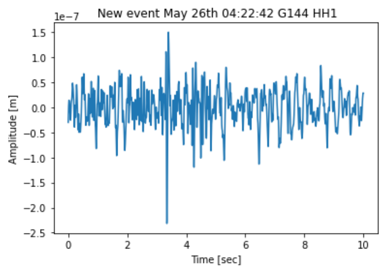


Figure E.40: New event May 26th

E.1.3. False positives

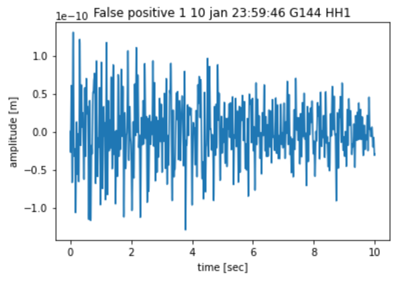


Figure E.41: False positive January 10th

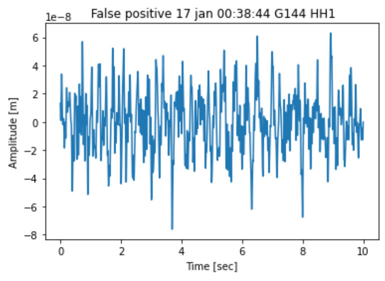


Figure E.42: False positive January 17th

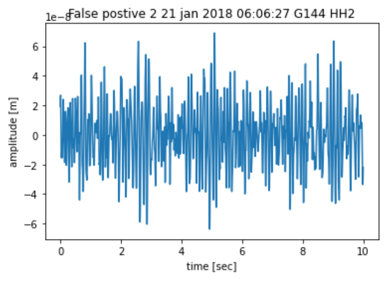


Figure E.43: False positive January 21st

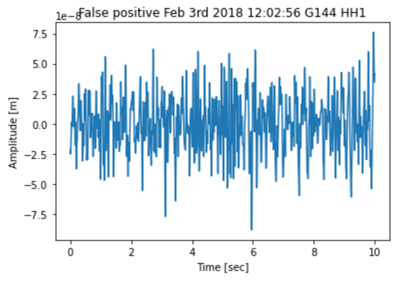


Figure E.44: False positive February 3rd

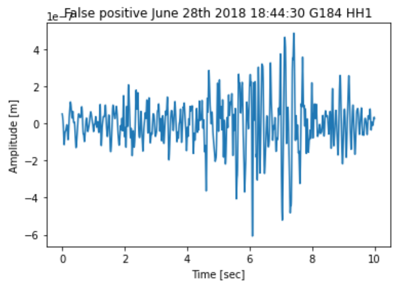


Figure E.45: False positive June 28th