University of Bergen
Department of informatics

Seismic Event Classification using Machine Learning

Author: Tord Sture Stangeland
Supervisors: Pekka Parviainen and Steffen Mæland

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2021

Abstract

The manual detection of seismic events is a labor intensive task, requiring highly skilled workers continuously analyzing recorded waveforms. Previous work has shown the potential of machine learning methods for aiding in this task, and that deep neural networks are able to learn important patterns in seismic recordings. This study aims to develop a deep neural network to classify earthquake-, explosion and noise events using long beamformed waveform snippets from NORSAR's ARCES array.

The final model was evaluated using an unseen test set and on recordings of the North Korean nuclear weapons tests. I developed custom augmentation methods in order to combat the uneven class distribution, and several preprocessing techniques were deployed in pursuit of performance. Models developed for similar data, state-of-the-art multivariate time series models, as well as self-developed models were experimented with and evaluated.

Analysis of the results demonstrated that the final model can classify noise and explosion events with a high degree of accuracy, while earthquake classifications were less reliable. I conclude that deep neural networks can learn distinguishing features and detect events of interest on long beamformed three-component waveforms.

Acknowledgements

First and foremost, I want to thank my advisors Pekka Parviainen and Steffen Mæland, who guided and assisted me throughout this project. Without your invaluable support, this project would not have been possible. Your unmatched patience, advice and expertise at every step of the way is tremendously appreciated.

I would also like to thank NORSAR for the fascinating thesis topic, making their ARCES dataset available to me, and providing access to analysts. Annie Jerkins, an analyst at NORSAR shared great insight into the manual classification process and provided resources for further reading.

Finally, I would like to thank my family: Gorm, Nina, Nora and Vilde Stangeland, for providing me with unfailing and continuous support throughout my years of study and the writing of this thesis. Your kindness and understanding is inseparable from this accomplishment.

Tord Sture Stangeland

1st of June, 2021

Contents

List of Figures **vi**

List of Tables **xiv**

1 Introduction **1**

1.1 Related work 3

1.2 Background 4

2 Seismology **7**

2.1 Wave Propagation 7

2.2 Array Processing 8

2.3 ARCES Array 9

2.4 Seismic Sensors 10

2.5 Waveforms 10

2.6 Current Solution 12

2.7 Earthquakes 12

2.8 Explosions 14

2.9 Noise 15

3 Machine Learning **17**

3.1 Supervised Learning 17

3.1.1 Generalization 17

3.2 Neural Networks 18

3.2.1 Feed-forward Neural Networks 19

3.2.2 Convolutional Neural Networks 20

3.2.3 Recurrent Neural Networks 24

3.2.4 Objective function 26

3.2.5 Activation functions . . . 27
3.2.6 Gradient Descent . . . 31
3.2.7 Backpropagation . . . 34
3.2.8 Overfitting And How To Prevent It . . . 35
3.2.9 Vanishing And Exploding Gradients . . . 37

4 Data And Methods **39**
4.1 Description Of The Dataset . . . 39
4.1.1 First Batch . . . 39
4.1.2 Second batch . . . 41
4.1.3 Outliers And Problematic Data . . . 44
4.2 Preprocessing . . . 47
4.2.1 Balancing the dataset . . . 48
4.2.2 Time Augmentation . . . 49
4.2.3 Filters . . . 51
4.2.4 Scaling . . . 52
4.2.5 Noise Augmentation . . . 57
4.2.6 Preprocessing dependency outline . . . 58
4.3 Model Selection . . . 58
4.3.1 Two Models In One . . . 60
4.3.2 Random Grid Search . . . 61
4.3.3 Local Search . . . 61

5 Results **63**
5.1 Metrics . . . 64
5.1.1 Accuracy . . . 65
5.1.2 Recall . . . 65
5.1.3 Precision . . . 66
5.1.4 F1 score . . . 66
5.1.5 Fβ score . . . 67
5.1.6 Precision-recall curve and Average Precision . . . 67
5.2 Models . . . 68
5.2.1 Fully Connected Neural Network (FCNN) . . . 70
5.2.2 LSTM . . . 77
5.2.3 CNN . . . 83

5.2.4 Final Model 101

6 Summary **119**

7 Bibliography **123**

A Selected Predictions **127**

A.1 MLSTM-FCN 3N 127

A.1.1 Selected False Positives 127

A.1.2 Selected False Negatives 129

A.2 MLSTM-FCN EE 130

A.2.1 Selected False Positives 130

A.3 Meier inspired CNN 3N 132

A.3.1 Selected False Negatives 132

A.3.2 Selected False Positives 134

A.4 Meier inspired CNN EE 135

A.4.1 Selected False Negatives 135

A.4.2 Selected False Positives 136

A.5 Self-developed CNN 3N 138

A.5.1 Selected False Negatives 138

A.5.2 Selected False Positives 140

A.6 Self-developed CNN EE 143

A.6.1 Selected False Positives 143

A.6.2 Selected False Negatives 144

A.7 InceptionTime 3N 146

A.7.1 Selected False Positives 146

A.7.2 Selected False Negatives 147

A.8 InceptionTime EE 149

A.8.1 Selected False Positives 149

A.8.2 Selected False Negatives 150

A.9 Final Model 151

A.9.1 Noise Events Incorrectly Classified As Earthquakes 151

List of Figures

2.1 Image of a recording station at ARCES. Photo credit: NORSAR/ Jan Petter Hansen 9

2.2 An earthquake waveform from the NORSAR dataset. The recording is from 2nd of January 2009, and is a 4 minute recording starting at 16:53:45. The earthquake has an epicenter 867 km from ARCES and has a magnitude M_L 3.27. 11

2.3 An earthquake waveform from the NORSAR dataset. The recording from 7th of March 2008, is a 4 minute recording starting at 18:18:26, and contains an earthquake of magnitude M_L 2.2 with an epicenter 342 km from ARCES. . . 13

2.4 An explosion waveform with M_L 2.2 with epicenter 629 km from ARCES. . . 15

2.5 16

3.1 Illustration of a basic Artificial Neural Network. Shows the input layer, a single hidden layer, output layer and direction of information flow indicated by the arrows. 19

3.2 Illustration of an artificial neuron, including its inputs, sum or transfer function, activation function and output. 20

3.3 Example of the convolution process. The top image shows the resulting convolved feature from a 3x3 filter convolved over a 5x5 image, with stride = (1,1) and padding = "valid". The bottom picture shows the resulting convolved feature from a 3x3 filter convolved over the same 5x5 image, with stride = (1,1) and padding = "same". 22

3.4 Binary step activation function. 27

3.5 Sigmoid activation function. 28

3.6 ReLU activation function. 29

3.7 TanH activation function. 30

4.1 Magnitude distribution of earthquakes and explosions in the first batch of the ARCES dataset. Both graphs are created with a bin size of 40. 40
4.2 Magnitude and MSRDR distributions of earthquakes and explosions in the ACRES dataset. Both magnitude graphs are created with a bin size of 40, MSRDR graphs uses a bin size of 80. Note that the scales are different in the MSRDR graphs. 42
4.3 46
4.4 Illustration of how the time augmentor shifts the event of interest, and fills the gaps with presumed noise. 50
4.5 Two outlier waveforms labeled earthquake (4.5a) and explosion (4.5b), both with strong continuous expression in the vertical component 53
4.6 Flowchart describing the general preprocessing outline. 59
4.7 Simple flowchart describing the structure of the two-in-one approach. 61

5.1 Graph showing the resulting Fβ as a function of β with the precision and recall from a classifier exclusively predicting earthquakes. The chosen β of 2 is labeled in blue. The horizontal axis shows the β value, while the vertical axis portrays the resulting Fβ score. 69
5.2 The two figures show the outputs of each baseline model. Each model use the same hyperparameters and preprocessing steps. 71
5.3 Confusion matrix showing the predictions made by best performing FCNN 3N model on the validation set. 73
5.4 The two figures show the raw and preprocessed outlier noise waveform incorrectly predicted to be not noise. 74
5.5 The two figures display the performance of the final FCNN EE model. 5.5a shows the predictions made by the model. 5.5b shows the PRC curve of the classifier. 76
5.6 The two figures show the outputs of each baseline model. Each model use the same hyperparameters and preprocessing steps. 78
5.7 Figure showing the architecture of the default MLSTM-FCN published on Majumdar's github: `https://github.com/titu1994/MLSTM-FCN`. Note that the number of units in their LSTM layer differs, depending on the application. 80
5.8 Confusion matrix showing the predictions made by the MLSTM-FCN classifier. 81
5.9 The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve. 83

5.10 The two figures show the outputs of each baseline model. Each model use the same hyperparameters and preprocessing steps. 84
5.11 Confusion matrix showing the predictions made by the best performing Meier inspired 3N model on validation. 88
5.12 Confusion matrix showing the predictions made by the best performing self-developed 3N model on validation. 90
5.13 Illustration of the Inception module used in the InceptionTime architecture, with default hyperparameters. 91
5.14 Illustration of the residual connection in the residual block used in the InceptionTime architecture. 92
5.15 Illustration of the best performing InceptionTime variation on the 3N task. Note that this model does not use residual connections, and that each Inception module is identical. 93
5.16 Confusion matrix showing the predictions made by the selected InceptionTime model 94
5.17 The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve. 96
5.18 The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve. 98
5.19 Illustration of the best performing InceptionTime variation on the EE task. Note that this model does not use bottleneck layers in the Inception modules. 99
5.20 The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve. 100
5.21 Confusion matrix showing the predictions made by the final model on the test set. 103
5.22 The two graphs show the distribution of distances from ARCES for explosions 5.22a and earthquakes 5.22b, overlayed by the distribution of the correctly identified class with respect to distance. Note the broken y-axis of 5.22a. . . 104
5.23 The two graphs show the distribution of magnitudes for explosions 5.23a and earthquakes 5.23b, overlayed by the distribution of the correctly identified class with respect to magnitude. Note the broken y-axis of 5.23a. 105
5.24 The two graphs show the distribution of MSRDR for explosions 5.24a and earthquakes 5.24b, overlayed by the distribution of the correctly identified class with respect to MSRDR. Note the broken y-axis of both figures. 106

5.25 Shows the distribution of the explosions in the test set, and the explosions incorrectly labeled noise, with respect to distance. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of distance to the models' error. 108

5.26 Shows the distribution of the earthquakes in the test set, and the earthquakes incorrectly labeled noise, with respect to distance. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of distance to the models' error. 109

5.27 Shows the distribution of the explosions in the test set, and the explosions incorrectly labeled noise, with respect to magnitude. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of magnitudes to the models' error. . 110

5.28 Shows the distribution of the earthquakes in the test set, and the earthquakes incorrectly labeled noise, with respect to magnitude. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of magnitudes to the models' error. . 111

5.29 Shows the distribution of the explosions in the test set, and the explosions incorrectly labeled noise, with respect to MSRDR. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of MSRDR to the models' error. 112

5.30 Shows the distribution of the earthquakes in the test set, and the earthquakes incorrectly labeled noise, with respect to MSRDR. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of MSRDR to the models' error. 113

5.31 Confusion matrix of the models' classification when given nuclear explosion waveforms as input 115

5.32 The two waveforms are of the first North Korean nuclear test, NK1. This recording by ARCES is 6250 km away from the detonation. The 3N submodel outputs 0.01, while it the correct output is 1. The raw waveform can be seen in 5.32a and the transformed waveform is seen in 5.32b. The transformation consists of normalize scaling, noise augmentation and a manual shortening of the event. 116

5.33 The two waveforms contain the NK6 nulear explosion, where 5.33a shows the ARCES recording (6250 km) and 5.33b shows the MJAR recording (950 km). 117

A.1 Predicted output: 0.59, correct output: 0. Noise. Outlier waveform, likely hardware error. 127
A.2 Predicted output: 0.82, correct output: 0. Noise. Difficult to distinguish without domain expertise, but a notable mistake made with relatively high certainty. 128
A.3 Predicted output: 0.62, correct output: 0. Noise. Appears to be a typical noise waveform, although with a short minor period of particular displacement. Speculated to be the cause for the classification error. 128
A.4 Predicted output: 0.13, correct output: 1. Type: Explosion, magnitude: 2.2, distance: 1106 km. Likely mislabeled start time causes time augmentor to compromise the event of interest. 129
A.5 Predicted output: 0.06, correct output: 1. Type: Explosion, magnitude: 0.9, distance: 119 km. Near but very low magnitude explosion event. 129
A.6 Predicted output: 0.06, correct output: 1. Type: Explosion, magnitude: 2.3, distance: 1148 km. Outlier event, hardware error. Good example of how the normalized scaler transforms the input waveform. 130
A.7 Predicted output: 0.88, correct output: 0. Type: Explosion, magnitude: 2.1, distance: 297 km. This event is also speculated to be of ripple-fired explosions. 130
A.8 Predicted output: 0.71, correct output: 0. Type: Explosion, magnitude: 2.0, distance: 312 km. Speculated to be recording of ripple-fired explosions, which should be relatively distinct from earthquakes. Neither weak nor distant. Notable mistake. 131
A.9 Predicted output: 0.45, correct output: 1. Type: Earthquake, magnitude: 1.75, distance: 134 km. This event is of a near earthquake with good signal to noise ratio. Notable mistake. 131
A.10 Predicted output: 0.33, correct output: 1. Type: Earthquake, magnitude: 1.85, distance: 163 km. This event is of another near earthquake with good signal to noise ratio. 132
A.11 Predicted output: 0, correct output: 1. Type: Explosion, magnitude: 1.5, distance: 293 km. Hardware error. Unprecedented regular displacement in stacked counts. 132
A.12 Predicted output: 0, correct output: 1. Type: Explosion, magnitude: 2.3, distance: 1129 km. Sudden spike, likely unrelated to the event, obfuscating the event of interest. 133

A.13 Predicted output: 0.35, correct output: 1. Type: Explosion, magnitude: 2.4, distance: 1137 km. Either mislabeled, or slow traveling waves, causing time augmentor to cut out the event of interest. 133
A.14 Predicted output: 1, correct output: 0. Noise. Labeled noise, although appears that the recording starts during an ongoing event. 134
A.15 Predicted output: 0.99, correct output: 0. Noise. Atypical expression in stacked counts. 134
A.16 Predicted output: 0.99, correct output: 0. Noise. Irregular noise waveform. . 135
A.17 Predicted output: 0, correct output: 1. Type: Earthquake, magnitude: 1.2, distance: 310 km. Event is eclipsed by noise, except short burst in the transverse channel. 135
A.18 Predicted output: 0, correct output: 1. Type: Earthquake, magnitude: 2.0, distance: 382 km. Inaccurately labeled start time, causing time augmentor to cut out the onset of the event. 136
A.19 Predicted output: 0.89, correct output: 0. Type: Explosion, magnitude: 2.0, distance: 340 km. Recording starts during the end of another event. 136
A.20 Predicted output: 0.99, correct output: 0. Type: Explosion, magnitude: 1.7, distance: 343 km. Striking resemblance to the waveform in A.34. 137
A.21 Predicted output: 1, correct output: 0. Type: Explosion, magnitude: 1.7, distance: 343 km. Inaccurate start time causing time augmentation to cut out the start of the event. 137
A.22 Predicted output: 0.49, correct output: 1. Distance to ARCES: 1142 km. Magnitude: 2.31. Earthquake. 138
A.23 Predicted output: 0.17, correct output: 1. Distance to ARCES: 765 km. Magnitude: 1.7. Explosion 138
A.24 Predicted output: 0.03, correct output: 1. Distance to ARCES: 723 km. Magnitude: 1.8. Explosion. Hardware error. 139
A.25 Predicted output: 0.43, correct output: 1. Distance to ARCES: 1139 km. Magnitude: 2.2. Explosion. Hardware error. 139
A.26 Predicted output: 0.17, correct output: 1. Distance to ARCES: 205 km. Magnitude: 1.6. Explosion. Notable mistake. 140
A.27 Predicted output: 0.96, correct output: 0. Noise. Irregular noise waveform. Same error made in A.16, although here with more uncertainty. 140
A.28 Predicted output: 0.62, correct output: 0. Noise. Hardware error. 141

A.29 Predicted output: 0.63, correct output: 0. Noise. Seems to contain seismic event. Likely error stemming from the way the noise data is sampled. Seems to disregard everything prior to the labeled event start. 141

A.30 Predicted output: 0.93, correct output: 0. Noise. This waveform contains a seismic event, but is mislabeled. Contains the classic P and S wave structure. 142

A.31 Predicted output: 0.51, correct output: 0. Noise. Outlier noise waveform with very strong displacement in all channels' stacked counts. Not a problem due to normalize scaler, but incorrect prediction nonetheless. 142

A.32 Predicted output: 0.87, correct output: 0. Type: Explosion, magnitude: 2.1, distance: 570 km. Outlier waveform, likely due to hardware error. 143

A.33 Predicted output: 0.60, correct output: 0. Type: Explosion, magnitude: 2.2, distance: 921 km. Incorrectly labeled start time causes time augmentor to cut out significant portions of the event of interest 143

A.34 Predicted output: 0.93, correct output: 0. Type: Explosion, magnitude: 3.0, distance: 392 km. Noisy waveform near the end of the raw waveform has been inserted right before the actual event. Additionally, the labeled time is inaccurate, causing time augmentor to cut out the start of the event. 144

A.35 Predicted output: 0.18, correct output: 1. Type: Earthquake, magnitude: 2.1, distance: 404 km. The labeled start time is incorrect, causing time augmentor to cut out the start of the waveform. 144

A.36 Predicted output: 0.32, correct output: 1. Type: Earthquake, magnitude: 2.6, distance: 426 km. The labeled start time is incorrect, causing time augmentor to cut out the start of the waveform. 145

A.37 Predicted output: 0.33, correct output: 1. Type: Earthquake, magnitude: 2.2, distance: 393 km. The labeled start time is incorrect, causing time augmentor to cut out the start of the waveform. 145

A.38 Predicted output: 0.95, correct output: 0. Type: Noise. The recording starts during an ongoing (presumably) seismic event, confusing the classifier. Shows insensitivity to location of event. 146

A.39 Predicted output: 0.99, correct output: 0. Type: Noise. Mislabled event. Same as A.30. 146

A.40 Predicted output: 0.99, correct output: 0. Type: Noise. Short, but significant amplitude spike. 147

A.41 Predicted output: 0.34, correct output: 1. Type: Explosion, magnitude: 1.7, distance: 759 km. Very short but significant displacement in stacked counts in the vertical channel. 147
A.42 Predicted output: 0.22, correct output: 1. Type: Explosion, magnitude: 2.2, distance: 1106 km. Appears that the event of interest occur prior to the labeled start time, causing time augmentor to cut the event out of the input waveform. 148
A.43 Predicted output: 0.44, correct output: 1. Type: Explosion, magnitude: 1.6, distance: 383 km. Incorrectly labeled start time. Still, the event is, at least visually, clearly a seismic event. Notable mistake. 148
A.44 Predicted output: 0.67, correct output: 0. Type: Explosion, magnitude: 2.4, distance: 1140 km. Hardware error. 149
A.45 Predicted output: 0.99, correct output: 0. Type: Explosion, magnitude: 1.6, distance: 409 km. The start time of this event is mislabeled, causing time augmentor to cut out the start of the event. 149
A.46 Predicted output: 3e-8, correct output: 1. Type: Earthquake, magnitude: 2.4, distance: 404 km. The start time of this event is mislabeled, causing time augmentor to cut out the start of the event. 150
A.47 Predicted output: 2.7e-5, correct output: 1. Type: Earthquake, magnitude: 2.5, distance: 432 km. Relatively clear event, with high signal to noise ratio, not negatively affected by augmentation. Notable mistake. 150
A.48 3N prediction: 0.97, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. The waveform deviates from the typical noise recording. 151
A.49 3N prediction: 0.97, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. The waveform deviates from the typical noise recording. 151
A.50 3N prediction: 0.99, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. This event has much stronger displacement in stacked counts than typical noise recordings. It also appears to contain a clear P and S wave. 152
A.51 3N prediction: 0.97, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. Short, but distinct segment deviating from the norm. 152
A.52 3N prediction: 0.99, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. Another event predicted to not be noise with very high certainty and an earthquake event with very high certainty. 153

List of Tables

4.1 Distribution of the distance to the epicenters of non-noise events 43

5.1 Confusion Matrix for binary classification. 64
5.2 Architecture of the FCNN baseline model. 71
5.3 Architecture of the best performing FCNN 3N model. 73
5.4 Architecture of the best performing FCNN EE model. K/A is short for Kernel and Activity. 75
5.5 Architecture of the LSTM baseline model. 77
5.6 Architecture of the CNN baseline model. 84
5.7 Architecture of the best performing CNN 3N inferred from Meier et al. . . . 87
5.8 Architecture of the self-developed CNN 3N model 89
5.9 Architecture of the best performing CNN 3N inspired by Meier et al. Note that this architecture is inferred from the paper, and futher optimized by experimenting with intermediate layers, and does not necessarily reflect the architecture used to achieve their results. 95
5.10 Architecture of the best performing self-developed CNN EE model 97
5.11 Summary of the performance of all of the models on both tasks. Note that the hyperparameters for the models differ between the two tasks. A randomly guessing 3N model would produce validation accuracy of ∼50%, and the EE equivalent would produce an F2 score of 0.292. 101
5.12 Performance metrics of the final classifier on the test set. 102

Chapter 1

Introduction

Earthquakes are an unpreventable and inevitable danger for life on Earth. The violent shaking of the ground has claimed an average of 27,000 lives a year since 1990 [30], and the cause of considerable financial loss. The study of this phenomenon help scientists and engineers take measures to reduce the harm and simultaneously gain insight to the inner workings of the planet.

Over time, the technology used for studying this natural occurrence has evolved in tandem with the technological revolution. There are now thousands of stations across the globe constantly recording the movement of earths mantle. Historically, these recordings have been manually analyzed by scientists. This labor-intensive task may be able to utilize classification algorithms in order to reduce the workload, allowing resources to focus on research rather than detection.

Unfortunately, automatic detection and labeling of interesting events is challenging. Seismological recordings are littered with non-tectonic earthquake events such as blasts related to mining activity and mining induced earthquakes all camouflaged by ambient noise. Low magnitude earthquakes are very difficult to detect among the noise and make up the majority of tectonic earthquakes. Regardless, the resilient scientists have been able to detect millions of earthquakes throughout time, and have produced large databases with labeled recordings.For every earthquake detected, hours, if not days, of uninteresting noise recordings are accumulated.

The noise is made up by a mix of human and natural sources, discussed later in the thesis, whose presence complicate the automatic detection of events. The growing collection of dormant and underutilized data has fueled the developments in automatic analytical model building. These models are a branch of artificial intelligence based on the idea that systems can learn and identify patterns in data. Referred to as Machine Learning (ML), this field has developed algorithms which excel at classification with noisy data. The application of these tools may realize value in otherwise discounted data.

The goal of this project will be to implement an ML-based method for distinguishing noise, explosion and earthquake events, ideally operating on minimally processed data (i.e. directly on multivariate time series), which can run in real-time. It should be optimized, but not limited, for events up to 1000 km distance from the seismic station. To test how well it generalizes to events at longer distances, we can also employ it on the six known nuclear weapons tests performed by North Korea between 2006-2017. The project will use 150 second beamformed waveform snippets processed by the ARCES array. The project will experiment with the Deep Learning models utilized in the previous works, experimental architectures, as well as state-of-the-art models such as InceptionTime, by Fawaz et al. [8]

Several machine learning techniques will be employed in order to aid the task. Multiple baseline models will be developed in order to validate the results of attempted improvements. One model will train specifically to distinguish noise events from non-noise events. The non-noise events will then be fed into another model, which will classify between earthquake and explosion events. A description of the data sets will be found in section 4.1. The heavily unbalanced dataset will require the development of augmentation techniques to allow for upsampling. This will ideally prevent complications of overfitting and help the model generalize to new data.

The best performing model in each subtask will be selected, combined into a single pipeline and evaluated using a dataset of events previously unseen by any of the models. The evaluation of the final model will function as a test of generalization, and will simulate how the model would perform during deployement. Finally, recordings from nuclear explosion tests will be input into the model to evaluate its sensitivity to very distant events.

1.1 Related work

The utilization of machine learning methods has increased tremendously in many areas of research, including seismology. Meier et al. in their article Reliable Real-Time Seismic Signal/Noise Discrimination With Machine Learning [7], successfully show how machine learning can make Earthquake Early Warning systems more reliable and faster. Their deep learning classifiers can reliably discriminate between earthquakes and noise signals in real time with an accuracy of 99.5%. The dataset used for training consisted of 375k three-component quake records from 8432 different earthquakes, as well as 946k three-component noise records.

Tibi et al. [33] did a comparative study, comparing the performance of amplitude ratio(AR) methods and trained Convolutional Neural Network(CNN) models in order to classify events, distinguishing between Tectonic Earthquakes, Mining Induced Earthquakes and Mining Blasts. Using a subset of high signal quality spectrogram samples, their AR models had success rates ranging from 80% to 90%. The CNN reported accuracies ranging from 91% to 98% on the same subset. On the entire event catalog the CNN achieved accuracies ranging from 94% to 100%. The team used a dataset containing 1040 Tectonic Earthquakes, 6286 Mining Induced Earthquakes, 51 Mining Blasts from four different quarries. Their CNN model had 4 Convolutional Layers with 2x2 filters (with filter counts: 18, 36, 54, 54) followed by a three-node SoftMax activated output layer.

In the article by Meier et al., they used 3 second waveform snippets, on earthquakes of magnitudes ranging from 3 to 9.1. Their data is limited to events that occur with an epicenter within 1000 km, and uses an outlier filtration process which discards records with peak ground velocities outside of 6 standard deviations from their predicted value. Tibi et al.s article, limited their events to those within an epicenter 25-150 km from the recording station. For their CNN model, they used 90 second spectrograms, from three-component and single-component stations. In recordings from single-component stations, they filled the horizontal channels with zero values.

Several different ML techniques have been applied to detect seismic signals [22, 3, 7], but classification of the source mechanism has received less attention. Further, most recent work on ML in seismology (also including [23]) has focused on data from single stations, or from regional networks of single stations. Seismic arrays, such as ARCES, adds important

information about the incoming wave, but how to include this in an ML context, has so far not been comprehensively studied. This project will attempt to classify considerably longer three-component waveform snippets, with less restriction on distance and filtering of data. Two models, specialized to their individual task, will be developed, joined to produce the final classifier. The models will be trained on waveforms collected by the ARCES array.

1.2 Background

In the Nordics, where NORSAR (Norwegian Seismic Array) operates several seismometer installations, the vast majority of detected events originate from mining explosions. Large mines like the ones in Kiruna and Malmberget typically blasts multiple times each day, while naturally triggered earthquakes typically occur once a day or less, in the entire Nordic region. Monitoring seismic data is a highly skilled and labor-intensive task. The detection of interesting seismological events, such as earthquakes, is challenging due to anthropogenic activity.

When NORSAR prepares its public earthquake bulletin, all detected events are manually analyzed, and the events of suspected non-natural origin are removed. Given the significant mining activity, much of the analysist's time is spent reviewing instrument data for uninteresting events. Traditional approaches for identifying explosion events are based on waveform template matching, i.e. computing the correlation between an incoming signal and a predefined template event [11]. This method is generally very sensitive, but has two major drawbacks: 1) Due to the specificity of the template it does not generalize to other mines (or even parts of the same mine) other than it was constructed for, and 2) It cannot detect new mines or even events for which no template already exist. Therefore, NORSAR would like to deploy an automated system to flag explosion events, as a fast and precise analysis support tool.

The best situated station for monitoring northern Fennoscandia is named ARCES, and is located just outside of Karasjok, Finnmark. It consists of a total of 25 seismometers, placed in concentric rings. This seismic array allows for measuring both the angle and velocity of the incoming waves. The input data are ground-motion time series recorded at 80 Hz, at multiple stations. Using data from ARCES, NORSAR wishes to develop a machine learning

method to effectively identify explosion events regardless of source location. By nature, earthquakes produce a different seismic wave compared to explosions since explosions are pure compression events. Earthquakes, meanwhile, are typically caused by fault slips that create both compressional and tensional pressure. At the seismometer, two main distinct wave phases are typically observed; the primary (P) wave, which propagates longitudinally, and the secondary (S) wave, which propagates transversely. The main characteristic between explosions and earthquakes lies in the ratio of the power of the two waves – earthquakes generally have stronger S-wave than P-wave, which tends not to be the case for explosion events. This fact was used to train an SVM, which identified explosions based on P- and S-wave average power as input [23].

Chapter 2

Seismology

2.1 Wave Propagation

Wave propagation is any of the ways in which waves travel through a medium. The propagation can be distinguished with respect to the direction of oscillation by longitudinal waves and transverse waves. Non-electromagnetic waves can only propagate through a transmission medium. Seismic waves are waves of energy that travel through Earth's layers, as a consequence of seismic activity. These activities include earthquakes, volcanic eruptions, magma movements, landslides etc.

There are two main types of seismic waves: body waves and surface waves. There are two types of body waves: P-wave and S-wave, and two types of surface waves: Love-wave and Rayleigh waves. P-wave (primary/pressure/compression wave) propagates through the interior of the earth at the greatest velocity of the three body waves. P-waves travel through air at the speed of sound, which is significantly slower than when propagating through the earth. The velocity of any wave depends on the medium, and its properties, in which it propagates through. Due to the P-wave having the highest velocity, this is the wave that is first recorded by a seismograph. The wave is caused by alternating compression and rarefaction (the opposite of compression), creating P-waves in the axis of the direction in which the ground moves. When felt, the P-wave is experienced as a small jolt.

S-waves, also called secondary waves, are transverse waves oscillating perpendicular to the direction of wave propagation. S-waves are the second wave to be recorded by a seismograph, after P-waves. This wave is experienced as strong shaking, relative to the P-wave, and causes significant displacement in the stacked counts axis in the recordings.

Love and Rayleigh waves are surface waves, and travel much slower than both P and S waves. Surface waves are limited to traveling through solid media. Love waves have a horizontal motion oscillating in the direction perpendicular to the direction of travel. Rayleigh waves cause the ground to shake in an elliptical pattern, reminiscent of ocean waves. Rayleigh waves are the most destructive wave resulting from a seismic event, but also the slower traveling of the two surface waves.

2.2 Array Processing

Using an array rather than individual stations provide significant benefits to the detection and analysis of seismic events. An array can provide estimates for the epicenter of an event and the apparent velocity of a signal. Epicenter is the term used to describe the location of a seismic event at ground level, directly above the hypocenter (referring to the initial point of the event). Apparent velocity in this context refers to the velocity of the wave in the horizontal plane, and can be used to derive the true velocity of the wave using the angle of incidence. The slowness of a wave , given by $\frac{1}{\text{velocity}}$, can be extrapolated with the medium density of the travel path to calculate the total travel time of the wave. The array can use the azimuth of the event combined with the derived travel time to provide a good estimates of the location of the epicenter. A network of arrays and stations improve the accuracy of the estimates, and may provide a more holistic view of earthquakes due to their directional effects.

Arrays also combat the local noise of individual stations. Through a process called beam forming, the array can improve signal to noise ratio of a seismic signal by summing the signals of each station in the array using the delay for a given slowness vector. Utilizing the effects of wave interference, the chaotic noise will cancel out, and the structured seismic event will be enhanced [16]. Theoretically able reduce the noise by a factor of $\sqrt{\text{Number of stations}}$, this signal clarifying technique allow detection of earthquakes otherwise undetectable by a single station. Every event in the NORSAR dataset has been preprocessed using beam forming, prior to the use in this project.

2.3 ARCES Array

The ARCES array in Karasjok consists of 25 seismometres placed in concentric rings with a diameter of 3 km. The 3-component broadband seismometres are sensitive instruments which excel at detecting and localizing events up to 300 km. However, the station is also used to record teleseismic events, earthquakes at distances greater than 1000 km. These devices are used to detect earthquakes, but also to monitor nuclear weapon testing by detecting the infrasound signature of nuclear explosions which can propagate for very long distances. The individual stations are connected to a central facility which supplies power and fiberoptic cables. The central station uses 4 GPS clocks, controlling an atomic clock which ensures the timestamps of each station are accurate up to a millisecond.

Figure 2.1: Image of a recording station at ARCES. Photo credit: NORSAR/ Jan Petter Hansen

Each station, as seen in 2.1, independently record ground motion and transmits to NORSARs headquarters through the central station. By using an array rather than individual stations, NORSAR is able to collect information such as the slowness of the beams, better localization estimates through triangulation, and improve the signal to noise ratio by utilizing wave interference [12].

2.4 Seismic Sensors

Seismic sensors measure motion in the ground and translates it into a useful output. Traditionally, these instruments consisted of a mass suspended by a spring translating the motion of the mass using a stylus onto a rotating drum. These days, there are two types of sensors: passive and active. Passive seismometers record the motion of the ground by suspending a magnetic mass surrounded by coil outputting a voltage linearly proportional to the ground velocity at frequencies above its natural frequency. These passive sensors are limited to record frequencies above 0.03 Hz. Active sensors, which are most widely used, also consist of a swinging system, but can record a wider range of frequencies: typically 0.01-100 Hz [15]. The output of the sensors is called counts and is the raw number read off the instrument. This project will use the term "stacked counts" which is the resulting output after the beam forming process combines the counts of the individual stations in the ARCES array.

2.5 Waveforms

The dataset from NORSAR comes in the form of several labeled three-component waveform snippets and information about the recording. These snippets are 4 minutes long in their raw state, and their information varies by the event type. For non-noise events, the information includes the coordinates of the epicenter of the event, the distance to the event from ARCES, the magnitude of the event and more. An epicenter is defined by the point on the earth's surface vertically above the focus of an earthquake. A focus is the point at which an earthquake rupture starts. For the explosion events, this focus is where the explosion takes place.

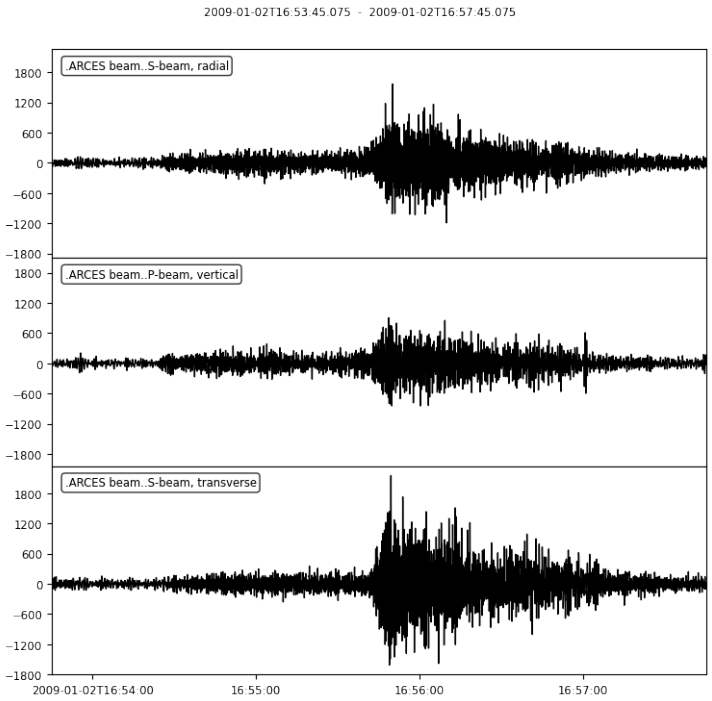


Figure 2.2: An earthquake waveform from the NORSAR dataset. The recording is from 2nd of January 2009, and is a 4 minute recording starting at 16:53:45. The earthquake has an epicenter 867 km from ARCES and has a magnitude M_L 3.27.

The three-components which make up the waveforms, an example of which can be seen in 2.2, are distinguishable by the directional axis in which they record movement of the earth. Three components are necessary because each individual component cannot capture a complete picture of wave motions from other directions. In the waveform, two of these components are horizontal, radial and transverse, and one is vertical. The radial component is oriented along the source-receiver azimuth, the transverse component is oriented in the direction perpendicular to the direction of the radial. The positive radial direction is the direction to the receiver from the source, and the transverse component is positive in the direction 90 degrees clockwise from the positive radial direction. Combined, these components provide a more comprehensive image of the waves and their directional displacements than any single component does by itself. At ARCES the original direction of the seismometers are north/south, east/west and vertical. After a recorded event, the epicenter estimate is used to translate the channels to the radial, transverse and vertical directions followed by beam forming the waveforms.

2.6 Current Solution

Accurate detection and distinction of seismic events is a fundamental and challenging task in seismology. NORSAR operate 5 seismic arrays of different sizes, 3 infrasound arrays, and 3 single stations in the Fennoscandia region. These arrays and stations are constantly recording and transmitting data to NORSAR. All of the data is manually analyzed, using a variety of techniques, instruments and variables. The weaker the event is the more challenging it is to classify as the waveforms are so similar. For such weak events, information such as time of the event, its location and its depth can be significant in determining what is going on. For example, seismic events recorded at night are more likely to be earthquakes. Events located near a known quarry or at a very shallow depth are more likely to be explosions. Characteristics, such as explosions typically having higher P-wave energy content and the frequency content of the wave, are explored using several types of digital filters with different parameters on different bands of the wave. In addition to the seismic arrays, infrasound instruments are used in conjunction with the seismometers by the analysts.

Continuous analysis of the data produces thousands of manually classified events each year. The classification of these events are uncertain, and events may be incorrectly labeled. Events of low magnitude (less than $\sim$1.3 magnitude) are more likely to be misclassified than stronger events, but external effects could increase the chance of misclassification of higher magnitude events as well.

2.7 Earthquakes

An earthquake is the shaking of the surface of the Earth as a result of movements within Earth's crust, volcanic activity, landslides, mine blasts, nuclear blasts and glacial activity. The term earthquake is a general term to describe any seismic event that generate seismic waves.

The most common type of earthquake is tectonic earthquakes. These earthquakes occur anywhere in the earth where stored elastic strain energy drives fracture propagation along a fault plane. Faults are described as a fracture or zone of fractures between two blocks of rocks. Faults are zones where blocks of rock move relative to each other. These movements

vary from a few millimeters to thousands of kilometers. In the case of earthquakes, these movements may occur rapidly, causing significant seismic waves, or slowly which is referred to as a creep. There are different types of faults, which may be further explored by the reader's interest elsewhere. A fault plane is the plane that represents the fracture surface of a fault. This plane describes the plane in which there is a slip along a fault.

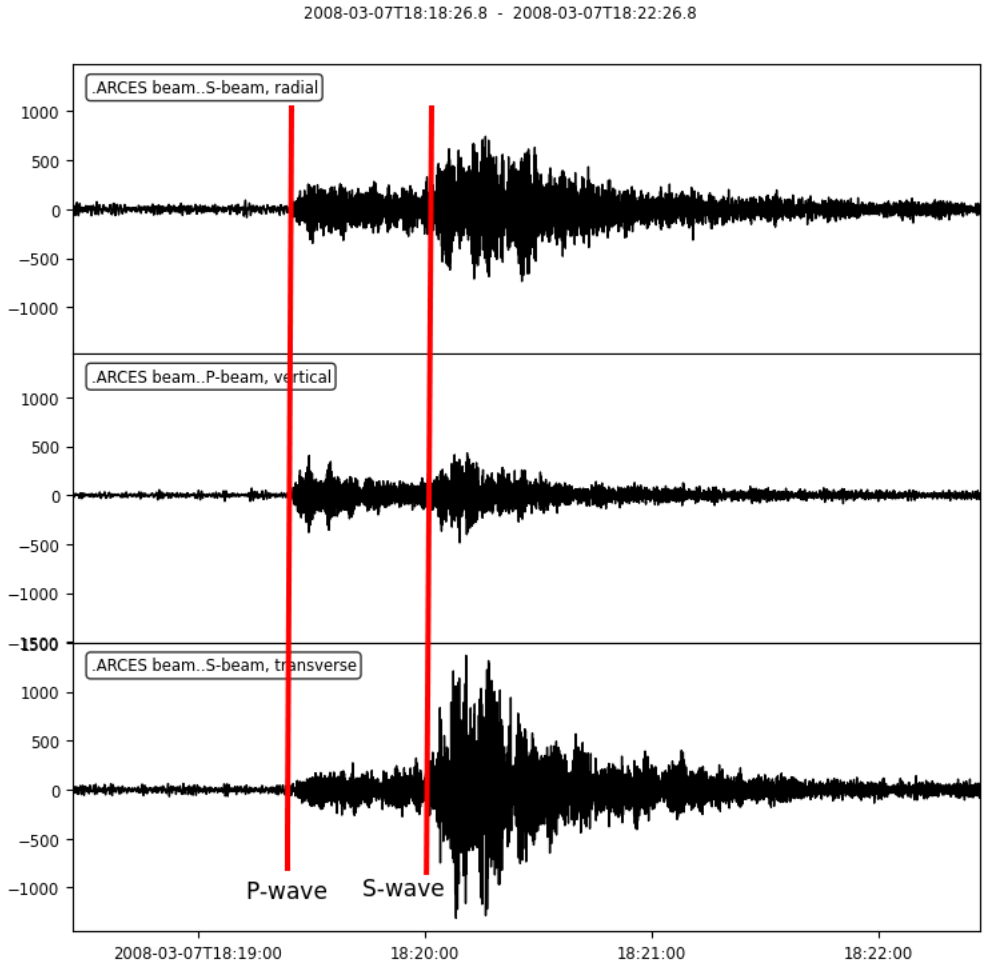


Figure 2.3: An earthquake waveform from the NORSAR dataset. The recording from 7th of March 2008, is a 4 minute recording starting at 18:18:26, and contains an earthquake of magnitude M_L 2.2 with an epicenter 342 km from ARCES.

Using the waveforms recorded by the seismometres, such as the one in 2.3, analysts can extract information about the events. Common metrics of earthquakes include its magnitude and its intensity. Magnitude is a measure of the size of the earthquake source, and is independent of what the earthquake feels like. There are different types of magnitude metrics,

and the information below is from *Routine Data Processing in Earthquake Seismology*, by J. Haskov and L. Ottemöller [16]. The book provides detailed explanation about each type. More magnitude types exist, but the variations of type as well as their use is not considered important for this project.

- M_L: Local magnitude, for events with magnitude less than 6-7 and distances less than 1500 km. Frequency band 1-20 Hz.
- M_c: Coda magnitude, for events with magnitude less than 5 and distances less than 1500 km.
- M_b: Body wave magnitude, for teleseismic events of magnitudes less than 7 and distances 20°- 100°. Frequency around 1 Hz and >0.3 Hz.
- M_s: Surface wave magnitude, for teleseismic events of magnitude up to 9 and distances 20°- 160 °. Period 18-22s.
- M_W: Moment magnitude, for any earthquake at any distance.

The local magnitude of the event is used when making reference to magnitudes of events in this project. Earthquakes differ in how they visually appear on the seismograph, depending on their magnitude and distance from the array. The P and S wave radiation patterns are strongly dependent on rupture directivity. The signals received to two stations at equal epicentral distance but at different azimuths may differ. Their energy is evenly distributed over the whole recorded frequency band and have a wide frequency content [23]. In fractions, the energy released by an earthquake is more prominent in S waves than in P waves.

2.8 Explosions

The explosion events in the ARCES data are generally explosions related to quarry and mining activity. Explosions are compression point events, and does not display the directional effects experienced by earthquakes. Ripple-firing is a common practice for economic blasts, and constitutes the majorty of explosion events. This practice uses spaced out small explosives detonated individually with a time delay. This technique enhances some frequencies while dampening others [23]. The resulting waveform is affected by these time-invariant variation of energy minima and maximas. In comparison to earthquakes, explosions have

lower energy. An earthquake event of the same magnitude will have a waveform with a high energy in a broader band of frequencies.

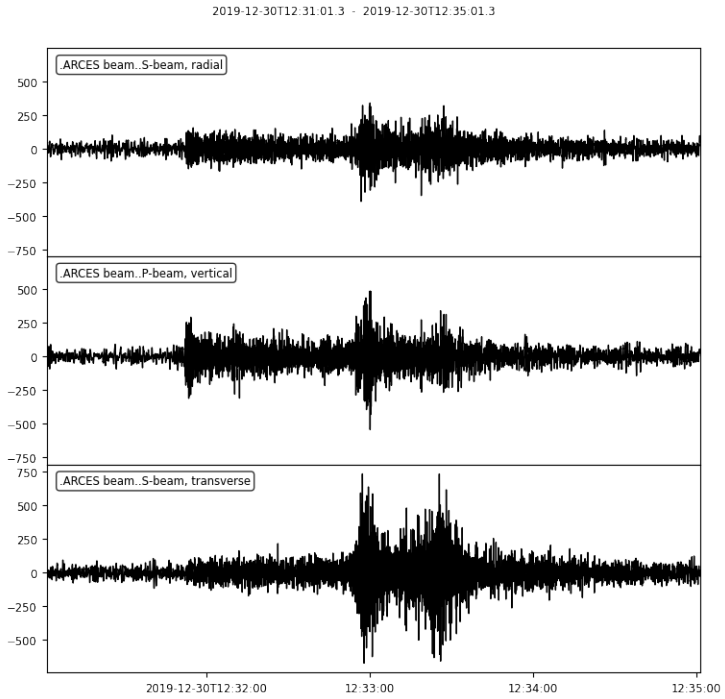


Figure 2.4: An explosion waveform with M_L 2.2 with epicenter 629 km from ARCES.

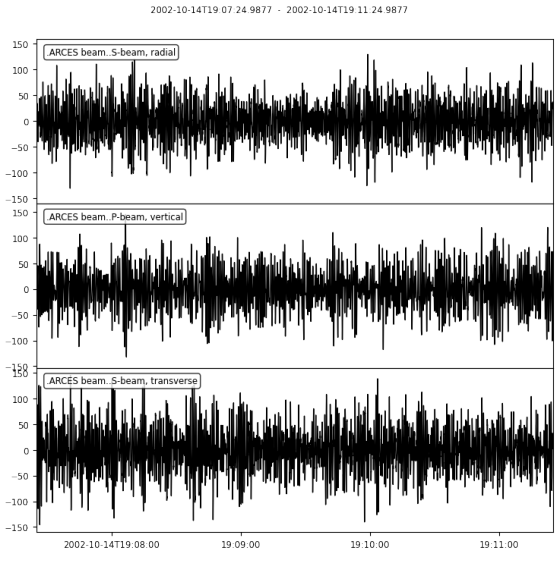
Figure 2.4 portrays a waveform of an event labeled explosion. Comparing explosion waveform to earthquakes such as the ones in 2.3 and 2.2, their raw, unprocessed, form does not display any immediate distinguishing features from each other.

2.9 Noise

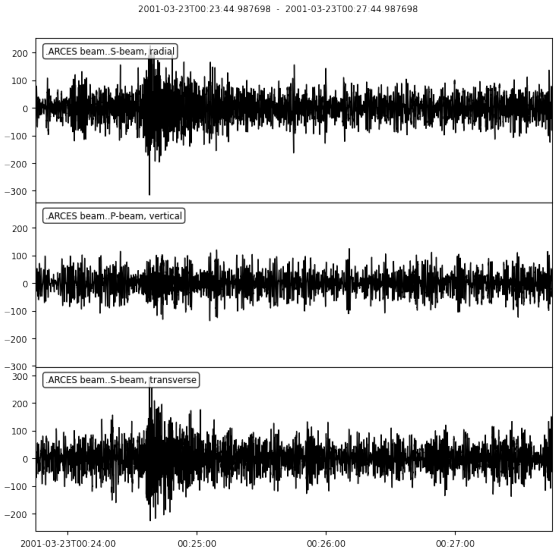
The ARCES array is constantly recording and transmitting to the headquarters at Kjeller, outside of Oslo. As a result, there is a tremendous amount of data containing data of little to no interest. The instrumentation required to record seismic events at teleseismic distances, and detect nuclear explosions is very sensitive, so microscopic disturbances will be picked up. Potential sources for the noise captured in the waveforms include:

- Various human activities: car and plane traffic, digging and maintenance work near the array.
- Resonance from seismic activity
- Effects of frost and ice
- Earth slides and avalanches
- Ocean waves crashing into the coastline
- Strong wind and other weather phenomena

These sources differ in their affect on the waveforms, but the most general way to differentiate them is by their duration. Particularly long lasting noisy conditions poses additional challenge to correctly discriminating important events. Strong winds and rough seas can cloak distant and or weak earthquakes, leaving them undetected. In general, earthquakes of magnitudes less than 1.3 are difficult to detect, worsened by the camouflaging noise.



(a) A waveform labeled noise. Note that the stacked counts axis is scaled to fit each waveform.



(b) A waveform labeled noise with a period of stronger stacked count displacement.

Figure 2.5

The noise waveform in 2.5a is a random waveform labeled noise in the NORSAR dataset. The chaotic waveform from 2002 does not display any clearly distinguishable features. Some noise waveforms do contain short sections of stronger displacement in the stacked count axis, such as in 2.5b, which upon immediate inspection can be mistaken for an seismic event.

Chapter 3

Machine Learning

3.1 Supervised Learning

Supervised learning is a type of learning task which involves mapping an input to an output and comparing the inferred output to the true output. This task requires a labeled dataset to be used for training, in which the model creates an inferred function, which can be used to infer new data. The performance of such a model is measured by its ability to infer correctly on unseen data.

3.1.1 Generalization

The goal of a supervised machine learning model is to perform well on unseen data, that is, the models' ability to generalize. Unseen data is data which the model has not been exposed to during the training phase. This is important because a model could report metrics indicating great performance on the data which it has used to train, while being completely useless when exposed to unseen data. This phenomena is called overfitting, and occurs when a model learns the noise and random fluctuations in the training data as important features used during inference. Learning these features is detrimental to its ability to infer on new data. Many machine learning models have parameters the developer can adjust and there are techniques available to reduce the risk of this occurring. Conversely,

the term underfitting is used to describe a model which reports poor performance even on the training set.

In order to gauge a models' ability to generalize, it is common to split the dataset into segments. There are different ways to split the dataset, but the technique used in this project is a standard train-validation-test split. This involves segmenting the dataset into three unique sets, each serving a different purpose. The training set is used for training the models. The validation set is a partition of the dataset which is indicative of the models ability to generalize, and is used to select models. The one model which performs the best on validation data is finally evaluated on the test data. The test data is a segment of the data which is only used once. This evaluation simulates how the model performs when deployed. It is important to have both a validation set and a test set as the validation set is used for selection. This selection introduces bias, as the developer selects the model which performs the best on this set. The test set has not been selected for and is therefore unbiased.

3.2 Neural Networks

Neural Networks (abbreviated NN) are biologically inspired learning models, which tries to "learn" underlying structures and relationships in data, in order to perform a task. The design of the neural network is based loosely on how biological brains function. This type of model had a boom in the 1980s, with the rediscovery of backpropagation, but due to limited data sets and computing power, they were soon outperformed by other machine learning models. With the enormous continuous growth of information and technological development, NNs have made a comback in the 2010s. The growing computational power allowed for the development of "deep" neural networks. The improvement in hardware combined with more efficient methods for training deep neural networks with non-linear hidden-units and a very large output layer [6].

A "deep" neural network means that it contains multiple layers between the input and output layers. Deeper networks have significantly more weights than a "shallow" network, and allow for a combination of differing layers. A deeper network improves the networks ability to learn non-linear features. With this explosion in combinations of layers possible, so too does the number of hyperparameters increase. Hyperparameters is a parameter the

creator can use to control the learning process. Hyperparameters can be visualized as a switchboard of nobs and levers that can be tuned resulting in different results of the model. Finding the best possible combination of these hyperparameters is key to optimizing the performance of the neural network, the exhaustion of which is impossible.

3.2.1 Feed-forward Neural Networks

There are several types of neural networks. The feed-forward neural network is the typical example of a neural network. This type of neural network attempts to approximate some function. In essence a feed-forward neural network attempts to map input to its output based on parameters. The network attempts to learn these parameters in order to produce the most accurate approximation of the output. The feed-forward term is used as the model takes some input and feeds it forward throughout intermediate layers before finally to the output.

Feed-forward Neural Networks consists of a set of layers disposed linearly. The layers are often categorized by three labels: input layer, hidden layer and output layer. The input layer consists of a number of input neurons corresponding to the shape of input data. Each input

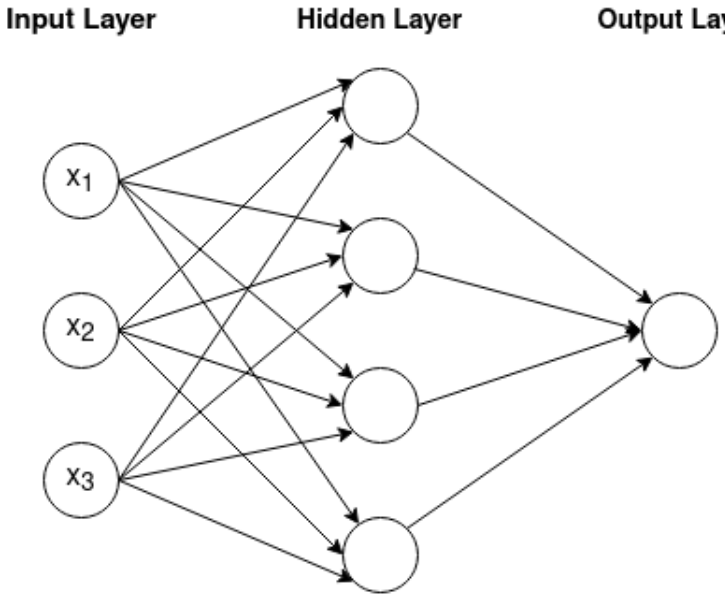


Figure 3.1: Illustration of a basic Artificial Neural Network. Shows the input layer, a single hidden layer, output layer and direction of information flow indicated by the arrows.

is multiplied by a weight corresponding to the "importance" of the input fed forward to the hidden layer(s). One can think of a hidden layer as a collection of neurons. The hidden layer(s) computes the weighted input of the previous layer based on its activation function. This output is then used as input in the subsequent layer. The final layer is called the output layer. This layer will make the final approximation of the network using the weighted inputs of the previous layer and its activation function. Each layer is made up by artificial neurons. Each neuron in a layer is independent of all other neurons in that layer.

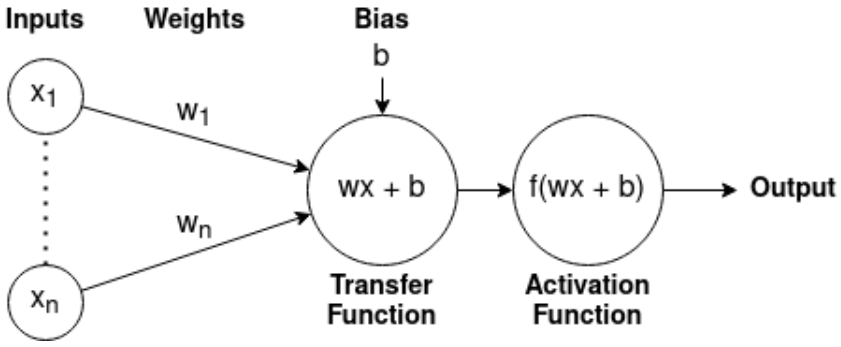


Figure 3.2: Illustration of an artificial neuron, including its inputs, sum or transfer function, activation function and output.

As seen in 3.2, an artificial neuron is made up by inputs, a weighted sum of the inputs, an activation function and a single output. The inputs of a neuron are made up by each output of the all the neurons in the previous layer, multiplied by a weight corresponding to the "importance" of that input. These weighted inputs are then added together with a bias in the transfer function, before arriving in the activation function. The activation function produces the output, which will be input to all the neurons in the subsequent layer. The bias is a constant introduced in the transfer function and is similar to the intercept added to a linear equation.

3.2.2 Convolutional Neural Networks

Convolutional Neural Network (abbreviated CNN) is an implementation of a neural network which processes array data, such as images. These types of networks were developed for computer vision, but excel at other tasks like natural language processing as well. As Neural Networks become deeper, the number of parameters to calculate grows quickly, which in turn

increases the training time. CNNs utilize dimensionality reduction to reduce the number of parameters to tune, which diminishes the increased training time as a result of depth.

For a computer, an image is an array of pixel values. Depending on the size and resolution of the image, the shape of an array of an image can be expressed by this its dimensions:

$$\text{Number of pixels wide} * \text{Number of pixels high} * \text{Number of channels(e.g. RBG)},$$

where each element in the array contain an integer between 0 and 255. Even for small images, this quickly becomes a large input, and thus parameters to learn by the network. In order to handle this type of data, convolutional layers are employed, giving the network its name.

Convolutional Layer

The convolutional layer is one of the main building blocks used in CNNs. This layer performs the convolution which involves a dot product of a set of weights with the input. This set of weights, called a filter or a kernel, is smaller than the input data. Since the operation is a dot product, the operation results in a single value. Due to the filter being smaller than the input data, this allows for it to be applied multiple times to different parts of the input image. In the convolution step, this filter is systematically applied left to right, top to bottom, of the image. How much the filter is moved for each operation is specified by the developer, by an integer or tuple, called a stride. The filter moves left to right, moving right at each operation by the specified stride value and after computing the rightmost multiplication, the filter the moves down by the specified stride value until the entire image has been traversed.

The ideal outcome of the training process is that the network is able to learn high level features from the data. In order to do this, several layers are likely necessary. The first convolutional layer extracts low level features such as edges, gradient orientation etc. Later layers use the previously extracted lower level features to extract higher level features. In order to accomplish this, the convolutional process may either reduce, retain or increase the dimensionality of the output, as compared to the input. One of the hyperparameters used to control this is how the convolutional step uses padding.

If the image is convolved with no padding (called "valid" padding), the resulting convolved feature is of smaller dimension than the original image. The resulting output height and output width can be calculated using:

- W: Input size
- P: Padding size
- F: Filter size
- S: Stride
- O: Output size

in the expressions:

$$O = ((W - F + 2P)/S) + 1 \tag{3.1}$$

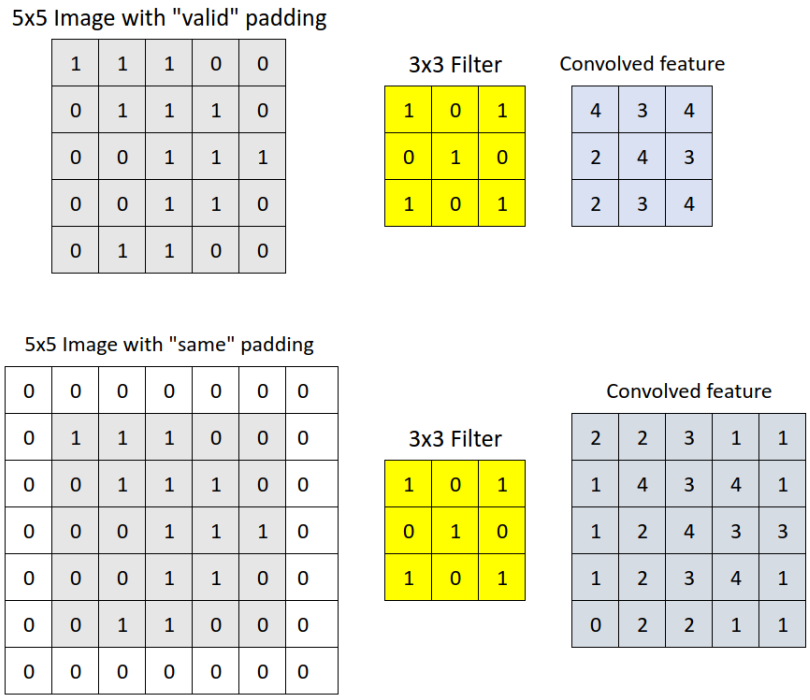


Figure 3.3: Example of the convolution process. The top image shows the resulting convolved feature from a 3x3 filter convolved over a 5x5 image, with stride = (1,1) and padding = "valid". The bottom picture shows the resulting convolved feature from a 3x3 filter convolved over the same 5x5 image, with stride = (1,1) and padding = "same".

If padding is applied in order to retain the same width and height in both the input and output, this padding is called "same". As seen in figure 3.3, when using valid padding, the resulting convolved feature is smaller than the input image. This is a consequence of the interaction of the filter with the borders of the image. In order to apply the filter so that

the border pixels can be in the center of the filter, "same" padding is used. Same padding creates a padding around the border of the image containing zeros.

The set of weights of each filter is learned through the training process and determines what kind of features it will detect. As a result, in order to capture several features, multiple filters are necessary. The user, therefore, also needs to determine how many filter is to be used by the convolutional layer. Finally, the resulting convolutional features are the output of the convolutional layer, which summarize the presence of features in an input image.

Pooling Layer

The summarized presence of features which convolutional layers output are sensitive to the location and rotation of the features in the input. In addition, depending on the padding and stride used, the size of the convolved features may be the same or greater than the input. On top of this, when using multiple filters, the spatial scale of the image may also be increased. In order to reduce this location and rotation sensitivity and to reduce the spatial size, pooling layers are often employed. Two common types of pooling layers, Average Pooling and Max Pooling summarize the average presence of a feature and the most activated presence of a feature respectively. In both methods, the user defines the stride, padding and pool size. The stride and padding behave the same as in the convolutional layer, but the pool size acts a little differently. The pool size defines a segment which will iteratively move, defined by the stride, from left to right, top to bottom, like the convolutional filter of the input image. The difference is that this segment will not calculate the dot product. For Max Pooling, this segment will return the highest value in each segment. The Average Pooling segment returns the average value of each segment. This operation is performed on every feature map from the Convolutional Layer.

The pool size is smaller than the size of the feature map, and is typically set to 2x2 pixels applied with a stride of 2 pixels on image data. In effect this means that it will half the size of each dimension of each feature map by a factor of 2, reducing the number of pixels of all feature maps to a quarter. This downsizing of the feature maps means that a feature that appears in a slightly different location in the feature map, will still have the same location in the pooled feature map, making it less sensitive to variance rotation and location of the detected features. In the case of Max Pooling, the pooling process will select for the most

prominent feature, acting like a denoiser of the feature map as less important features will be ignored. In either case the pooling operation is specified by the user, this process does not need to be learned, and thus does not add to the number of parameters the model needs to learn.

Global pooling layers are global operations used to reduce the dimensionality of the feature maps. These are often used to replace the flattening operation and sometimes fully connected layers, reducing the number of trainable parameters significantly. Common global pooling operations include Global Max Pooling and Global Average Pooling. In the context of CNNs, these reduce each input feature map to a single value. The Global Max Pooling layer reduces the feature map to its highest value. The Global Average Pooling layer reduces the feature map to its average value.

Fully Connected Layer

Finally, when the input has been processed and a final pooled feature map has been created, it is common to use a Fully Connected (FC) layer. A FC layer is a global operation, where each neuron is connected to every element in the previous layer and following layer. Prior to the use of such a layer in a CNN, the feature maps are flattened, that is the output is converted to a long vector in order to facilitate this transition. It is common to use one or several FC layers near the end of CNNs, as these layers learn a (possibly non-linear) function from the high level feature map extracted by the convolution layers. This is important as feature maps from convolution layers (typically) depend only on a subset of the input dimensions. In other words these feature maps are generated by a sequence of local operations. If one were to build a classifier consisting entirely of stacked convolutions, so that one ends up with the correct number of output neurons, each output neuron would depend only on a subset of the input. Global operations such as FC facilitate the flow of information such that the prediction depends on the entire input.

3.2.3 Recurrent Neural Networks

Recurrent Neural Networks(RNN) is a class of Artificial Neural Networks. This type of network attempts to allow neural networks to reason with contextual information. That is,

the output of the network is informed by previous (and future) input. While Feed-forward Neural Networks(FNNs) allow only information to travel one way: input to output, RNNs can have information traveling in both directions. Unlike standard Feed-forward neural networks, RNNs uses a single input, such as a word, or slice of a temporal series to produce an output. "Recurrent networks ... have an internal state that can represent context information. ... [they] keep information about past inputs for an amount of time that is not fixed a priori, but rather depends on its weights and on the input data... A recurrent network whose inputs are not fixed but rather constitute an input sequence can be used to transform an input sequence into an output sequence while taking into account contextual information in a flexible way."[2]. This ability to store context information allows the network to "remember" previous computations.

Due to this "memory", RNNs excel at modeling series data as it exhibits temporal dynamic behavior. Applications vary, but RNNs are often used for Natural Language Processing, Speech Recognition, Machine Translation, Image Recognition and many more. For example, in this project, an input into an RNN is a single timestep slice consisting of the 3 channels. RNNs is used with serial data, where the order of data is important. In series data, a single input will likely have an impact on other elements of the series. For example, the incomplete sentence "The people of Norway speak ...", the word "Norway" and the word "speak" affects the likelihood of the next word.

The network's ability to "remember" long term dependencies in sequences is, however, limited. When utilizing back-propagation, the network class suffers from vanishing and exploding gradients. This phenomena is particularly prominent in Recurrent Neural Networks, as a result of their undefined input shape. Vanishing/exploding gradients is an issue in deep neural networks, where the gradients are propagated through several layers. The depth of a Recurrent Neural Networks is dependent on its input as a result of its cyclic behavior, and thus, the vanishing/exploding gradient phenomena is dependent particularly on the length of the sequences.

Long Short-Term Memory

Long Short-Term Memory (LSTM) is a type of RNN architecture, whose development reignited hope in the RNNs ability to handle context in data. Prior to the development

of the LSTM, the standard RNNs ability to handle long term dependencies were particularly limited by exploding and vanishing gradients. Unable to learn with time lags greater than 5-10, LSTMs can learn to bridge time intervals in excess of 1000 discrete time steps. This feat is achieved by the LSTM "cell" constant error flow through internal states special units [10]. This constant error flow prevents gradients exploding or vanishing, and is today widely used in a variety of RNN applications.

The key idea behind LSTMs is the cell state. It is very easy for information to "flow" through the node unchanged. LSTM nodes contain three "forget gates", which dictates how much information the node lets through. Gates are made up by a sigmoid activation functions each outputting a number between 0 and 1. If the value is zero, the LSTM will let nothing through, while a value of one will let everything through unchanged.

3.2.4 Objective function

An objective function is a numerical expression for the objective that is to be achieved by the model. For example, a goal is to descend from a mountain to sea-level, an objective function may be expressed by how the spatial coordinates translates to height above sea-level. This objective is often what one wants to either maximize or minimize. For example, the goal of most companies is to maximize profit. The objective function describes how the given inputs translates realization of ones goals numerically. When running machine learning or optimization algorithms, the objective function is used to evaluate the performance of the model. The maximization or minimization of this function is the goal of the algorithms.

Neural Networks attempt to minimize the error of the model. The objective function capturing the error is broadly referred to as the loss function. The form of the loss function differs depending on the task. The loss function captures all the computations of an input of the model down to a single number. A decrease in the loss function of a model is a sign of a better performing model. The cost function is the average of the loss function for all the training samples. Intuitively, the cost function calculates the error between the ground truths and the predictions made by the model. There are several types of objective functions, the choice of which significantly impacts the behavior of the model.

3.2.5 Activation functions

Activation functions are a crucial component of neural networks. These functions take values from the single neuron summing block, and determines to what extent the neuron should be activated. The activation of a neuron is expressed as a single number, with a range depending on the function in use. Without activation functions, the output of a layer can be anywhere in the range $[-\infty, \infty]$. Restricting the output of a layer, to a range of values is important to maintain numerical stability, particularly in deeper networks. These functions takes the weighted sum of the output of the previous layer as input, adding bias, which will introduce non-linearity to the outputs, and produces a new output [9]. The ability to add non-linearity to a network is one of the most important characteristic of an activation function. The choice of the activation function has a significant impact on the performance of the neuron, and even the network in general. There are several different activation functions, some of which will be discussed below:

- **Binary step**:

$$y(x) = \begin{cases} 1 & \text{if x } \geq \ 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

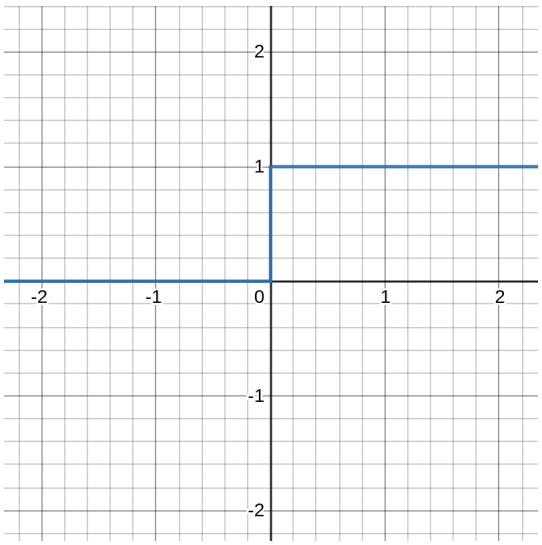


Figure 3.4: Binary step activation function.

This very simple activation function, is problematic due to only two possible values as output and can activate different labels to 1. This problem is solved by using smoother activation functions that can output a range of values.

- **The sigmoid activation function** σ:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

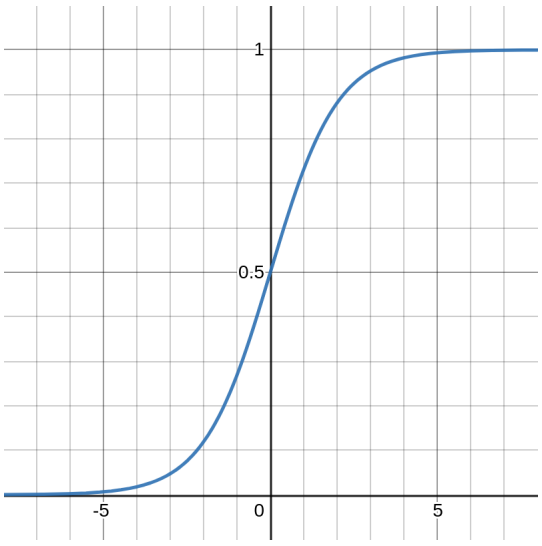


Figure 3.5: Sigmoid activation function.

This activation function is much smoother than the binary step function. The function has an range of (0,1), where $x \in \mathbb{R}$. The continuous nature of the function allows the network to learn non-binary classification tasks. Values of x near 0 are very steep, meaning that small changes in x will have a big impact on $\sigma(x)$. This tends to produce values of $\sigma(x)$ to either end of the curve. This is a good property for classification as it makes clear distinctions among predictions. The problem of this activation function is what happens when x is either very large or very small. Here, a small change of x will have a near zero impact on $\sigma(x)$, leading to very slow or no learning. This is what is called a vanishing gradient, which will be discussed in greater detail in the next section.

- **ReLU activation function**:

$$\text{ReLU}(x) = \max(0, x) \tag{3.4}$$

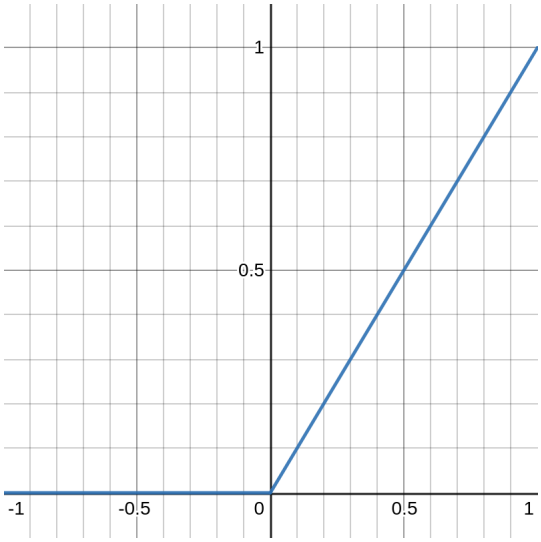


Figure 3.6: ReLU activation function.

The ReLU (Rectified Linear Unit) is a simple and effective activation function. It is a combination of an identity activation function and a threshold function. This activation function in its linear component does not suffer from the vanishing gradient problem that other activation functions, such as sigmoid, does, as ReLU does not have a maximum value. However, the unbounded maximum may cause an exploding activation. Another problem arise when the network has a substantial amount of 0-activations. When there a 0-activation, there is no gradient, and this neuron does not learn. This can lead to a large part of the network not learning, and only using a few "active" neruons. Variations of the ReLU attempt to mitigate the downsides of this activation function to varying degrees of success.

- **TanH activation function**:

$$\text{TanH}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.5}$$

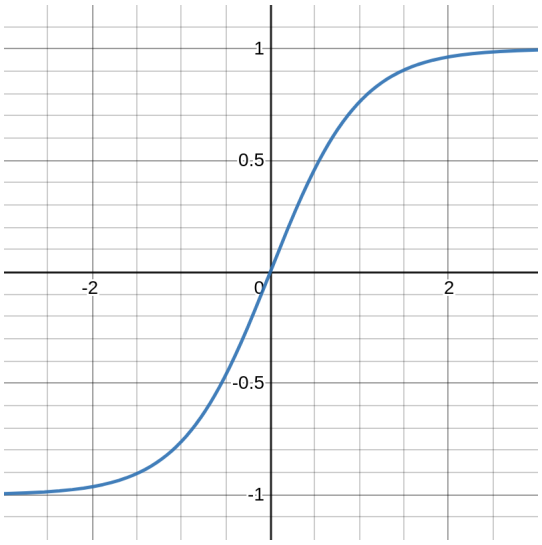


Figure 3.7: TanH activation function.

TanH (Hyperbolic tangent Activation Function) is a similar activation function to the sigmoid. TanH is monotonic has range (-1,1), has an S-shape and is differentiable. This activation function has a more pronounced gradient than sigmoid, but still suffers from the same vanishing gradient problem. This activation function is commonly used in binary classification.

- **Softmax activation function**:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{for} i = 1, ..., K \text{ and} z = (z_1, ..., z_k) \in \mathbb{R}^k \tag{3.6}$$

This activation function is the generalization of the sigmoid function for multi-class classification. This function takes a given vector z of $K \in \mathbb{R}$, and normalizes it into a probability distribution so that the sum of the probability distribution adds up to 1. This activation function is often used in the output layer of a neural network to normalize the output into a probability distribution. This allows the model to utilize argmax to pick the most likely class in the distribution.

There are many other activation functions available for neural networks. The choice of

which has a major impact on the network's ability to learn. It is common for all hidden layers to utilize the same kind of activation function, and for the output layer to have an activation function appropriate for the task. It is important to note that if one uses linear activation functions, the entire network will be linear. A linear activation function outputs a linear transformation of the input. This would be propagated throughout the network until the output layer. The output layer would simply become a linear transformation of the input in the first layer.

3.2.6 Gradient Descent

Gradient descent is an optimization algorithm used to optimize numerous parameters of a neural network. The algorithm uses the first-order derivative of a differentiable objective function iteratively optimizing the parameters to find a local minimum. This is achieved by taking small "steps" in the direction of the steepest descent. The size of these steps are defined by the non negative constant called "learning rate", denoted in 2.2 by α. In other words, gradient descent attempts to tune the weights, Θ, of the model, so that loss decreases. Let the parameters of a model $\Theta = \{\theta_0, ...\theta_j\}$ and let the loss function be defined by $L(y, \hat{y}(\Theta))$, where y is a vector of ground truth of the training data x, and $\hat{y}(\Theta)$ is the output of the model given the parameters Θ. The goal of gradient descent is to get

$$\min_{\Theta} L(y, \hat{y}(\Theta)) \tag{3.7}$$

In order to accomplish this goal, one needs to compute for each $\theta_j \in \Theta$

$$\theta_j^t = \theta_j^{t-1} - \alpha \frac{\partial}{\partial \theta_j^{t-1}} L(y, \hat{y}(\Theta^{t-1})) \tag{3.8}$$

repeatedly until convergence. It is important to compute these parameters simultaneously, that is, not let the updated value of θ_l^t affect the computation of θ_j^t, $l \neq j$.

One can interpret this algorithm by imagining a mountain (cost function). With the goal of descending this mountain, one can use gradient descent to take steps each of which in the direction that leads to the greatest decrease in height (loss). Depending on length of the step one takes and where on the mountain one starts their journey, one may find themselves stuck in a valley unable to get out as each step will increase height. Conversely, if the length of

ones step is too large, one may find oneself on a whole different mountain. Admittedly, here the cracks in the metaphor starts to show, but this is a possible outcome in a neural network context. This restriction to local extremes, means that the starting values of the parameters, as well as the learning rate, may cause the algorithm to converge to a sub-optimal extreme, or may "step" over optimal values of the parameters. Therefore, gradient descent only finds the global extreme if the loss function C is strictly convex, and thus does not guarantee to find the global minimum.

The choice of the learning rate will significantly impact the efficiency and capabilities of the model. A learning rate too large may prevent the convergence of the algorithm or step over the global extreme. A learning rate too small will be very time consuming as well as could cause convergence to an undesirable local extreme. The decision of this learning rate is an iterative process searching over multiple values of the learning rate.

Types Of Gradient Descent

There are two main types of Gradient Descent: Batch Gradient Descent and Stochastic Gradient Descent. Each of these have their own advantages and disadvantages, but none of them are guaranteed to find a global minimum.

Batch Gradient Descent

Batch Gradient Descent updates the parameters of the model once every epoch. An epoch is defined by a cycle through the training set. It calculates the error for each training sample, and updates the parameters once all of the samples have been evaluated. Let the size of the training set be represented by N, an epoch by $s \in S$ and the learning rate be a user defined constant α. An update of the parameter $\theta_j \in \Theta$ is calculated by the following expression:

$$\theta_j^s = \theta_j^{s-1} - \frac{\alpha}{N} \sum_{i=1}^{N} \nabla L(y^i, \hat{y}^i(\Theta^{s-1})) \tag{3.9}$$

Using all of the training set has the benefit of calculating its true error, and ensures the update is in the direction of steepest descent. The few updates to the model makes it quite

efficient, but since large datasets cannot always be held in RAM, vectorization is much less efficient. The stable error gradient may cause a premature convergence to a suboptimal local minima which the model is unable to get out of. In addition, in cases where the training set is very large, training speed may become an issue due to the slow model updates. Batch Gradient Descent is often believed to be excellent with smooth and convex error manifolds, but be less useful when the manifold contains several local extremes.

Stochastic Gradient Descent

Stochastic Gradient Descent (abbreviated SGD) is an optimization algorithm, very commonly used in neural network applications, regarded as a stochastic approximation of gradient descent. The key difference from Batch Gradient Descent is how frequently it updates the parameters, Θ, of the model. Unlike Batch Gradient Descent, SGD updates the parameters several times each epoch using batches. A batch is a subsample of the full training set, the size of which is defined by the user. The number of updates, or steps, K, performed each epoch for a batch size b, and a number of training points N is given by this equation: $\frac{b}{N}$. The process starts with a random initialization of the parameters and then each parameter $\Theta_j \in \Theta$ are updated once every step $k \in K$ for each epoch, using the following formula:

$$\Theta_j^k = \Theta_j^{k-1} - \frac{\alpha}{b} \sum_{i=k}^{i+n} \nabla L(y^i, \hat{y}^i(\Theta^{k-1})) \tag{3.10}$$

When the batch size is small enough to fit in RAM, the expression to the right of α in 3.10 can be vectorized which allows for paralellization of the gradient computation. Small batch sizes are often noisy, and is not necessarily representative of the true error gradient. Using such a noisy representation, the frequent updates may cause the model error to jump around. If the batch size is very large, the better the approximation is to the true gradient. As the batch size grows, the limitations expressed in Batch Gradient Descent become more prominent. The upside of this approach is that the model immediately provides insight into its performance and rate of improvement. The noisy samples may also help the model avoid local minima, but conversely may also prevent the model from converging to an optimum minimum. The frequent updates are computationally expensive, and may lead to long training times for large datasets. Experimentation is required to find the best possible batch size, in order to balance the tradeoff. Note: Some authors make the distinction between SGD and

Mini-Batch Gradient Descent, but according to Deep Learning by Goodfellow et al. [13], these are now commonly referred to simply as SGD.

Adam and RMSprop

Adam and RMSprop are two variations on gradient descent used in this project. These *adaptive* optimizers use ideas such as momentum and adaptive learning rate to overcome pathological error manifolds or to speed up training. Momentum is a technique which uses the gradients of past updates to inform the current. This helps build speed, moving faster towards a minimum, and dampens oscillations. RMSprop uses per-parameter learning rates, dividing the appropriate learning rate by an exponentially weighted average of squared gradients. This has the effect of automatically adjusting the parameter updates so they match the scale of the gradients. This decreases the scale of updates to parameters with large gradients, and increases the scale of updates to parameters with small gradients. Normalizing the parameter updates based on their magnitude prevents gradients from exploding as well as vanishing. The Adam (Adaptive Moment Estimation) algorithm uses momentum as well as an adaptive learning rate.

3.2.7 Backpropagation

In order to calculate the gradients of the weights of a feed forward neural network the backpropagation algorithm is used. Generalizations of the algorithm exist for other types of ANNs as well. This algorithm computes the gradient of the loss function with respect to the weights and biases of the network. These are the gradients used by gradient descent to update the model. In simple terms, the algorithm uses the chain rule to compute the gradient one layer at a time, iterating backward from the last layer to the first. Due to algorithm iterating backwards, the subsequent layer can use the previously calculated partial derivatives which is efficient in comparison to the naive approach of calculating the gradient of each layer separately. Knowing the gradient of each weight and bias of the model is the same as knowing how sensitive the loss function is to these variables. As a result, one also knows which variables to adjust to have the most impact.

Training a neural network from scratch is started by a random initialization of the weights and biases of the network. Each training sample processed by the network returns an output which is the weighted accumulation of activations in the network. The network will at first perform poorly, returning undesirable outputs. Knowing what the desired output to be, using backpropagation the network will be able to determine which weights and biases which contributed to the poor performance. Backpropagation lets the network calculate the gradient of these variables, and using gradient descent, adjust them so that their detrimental effect on the output is diminished. Popularized by Rumelhart et al., this algorithm is central to the fundamentals of neural networks and its introduction revolutionized how neural networks learn because "the ability to create useful new features distinguishes back-propagation from eariler, simpler methods..." [29].

3.2.8 Overfitting And How To Prevent It

With the development of deeper and wider neural networks, the chance of overfitting to the training data becomes more likely. When a network is trained for too long or the model is too complex, the model can start to learn unimportant features of the training data. These leared features prevent the model from generalizing on new data. In order to mitigate the chance of overfitting, several techniques are at disposal. The techniques deployed in this project are explained below:

Early Stopping

As models can overfit from training for too long, this is a technique which terminates training after the model has "peaked" in performance. This technique works by the user defining parameters such as the metric to monitor, how many epochs after the best recorded metric the model should wait before stopping and often whether or not the parameters of the model should be returned to the state in which they produced the best metrics.

Data Augmentation

This involves inflating the dataset with artificial samples, injected with noise or changed in some minor way. Although this should be used sparingly, can be a powerful substitute to additional authentic data.

L1 and L2 regularization

Regularizing the layers of a model with L1 and L2 regularization are widely used in supervised learning models. These techniques alter the objective function so that the model attempts to not only minimize loss, but also the complexity of the model. Two of the main reasons that cause a model to be too complex are the total number of features and the weights of the features. L1 regularization attempts to shrink the coefficients of unimportant features to zero, by adding the sum of absolute values of the parameters, multiplied by a user defined scaler term, λ, to the original loss function:

$$L(y, \hat{y}(\Theta))_{\text{new}} = L(y, \hat{y}(\Theta)) + \lambda \sum |\Theta| \tag{3.11}$$

As this technique forces weights of uninformative features to be zero, it doubles as built-in feature reduction tool, reducing the total number of features of a model. As this technique utilizes absolute sum of weights, it is robust to outliers.

L2 regularization penalizes large weights more harshly than small weights. This has the effect of encouraging the use of small but non-sparse weights. The technique removes a small percentage of weights at each iteration, making them smaller, but never equal to zero. This decreases the complexity of the model by reducing the weights, but does not have the feature reduction effect of L1. L2 regularization is controlled by a user defined λ term, which functions as a scaler for the sum of squared weights term:

$$L(y, \hat{y}(\Theta))_{\text{new}} = L(y, \hat{y}(\Theta)) + \lambda \sum \Theta^2 \tag{3.12}$$

The squared term does not make this regularization technique robust to outliers, but does help the model be able to learn complex data patterns.

Dropout

Whereas L1 and L2 modify the loss function in order to regularize, dropout changes the network by randomly dropping neurons each iteration of training. Dropping neurons means that the network will not use the weights and activation of the dropped neurons during training, essentially training a different (thinned) neural network for each iteration. This

technique can be interpreted as adding noise to the networks' hidden units. "Backpropagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout breaks up these co-adaptions by making the presence of any particular hidden unit unreliable." [31].

3.2.9 Vanishing And Exploding Gradients

The back propagation algorithm, although effective, has drawbacks which become more prominent depending on the depth (number of hidden layers) of the network. The error gradient of a networks is computed using the chain rule, and thus in a network of n layers, n derivatives are multiplied together. If these derivatives are either small or large, the continuous multiplication will cause the gradients to exponentially become smaller or larger causing the gradients to either vanish or explode.

In cases of exploding gradients, this causes the model to be unstable and unable to effectively learn. Symptoms of this problem are seen through continuous and large increases in the loss of the model, eventually resulting in NaN (Not a Number) values, as a consequence of very large weights. When the weights/loss contain NaN values, they can no longer be updated, and the network fails.

In cases of vanishing gradients, this causes the model to be incapable to learn significant features from the data. When the gradients of the weights and biases of the model are near zero, they will not be updated meaningfully, and the network will no longer learn.

The phenomena of vanishing and exploding gradients is widespread, and several techniques have been developed in order to mitigate the problem. Common remedies include: reducing the depth of the network, gradient clipping (limiting the magnitude of the gradients, to prevent exploding gradients), carefully initializing the weights and batch normalization. Explaining these remedies in further detail are beyond the scope of this project.

Chapter 4

Data And Methods

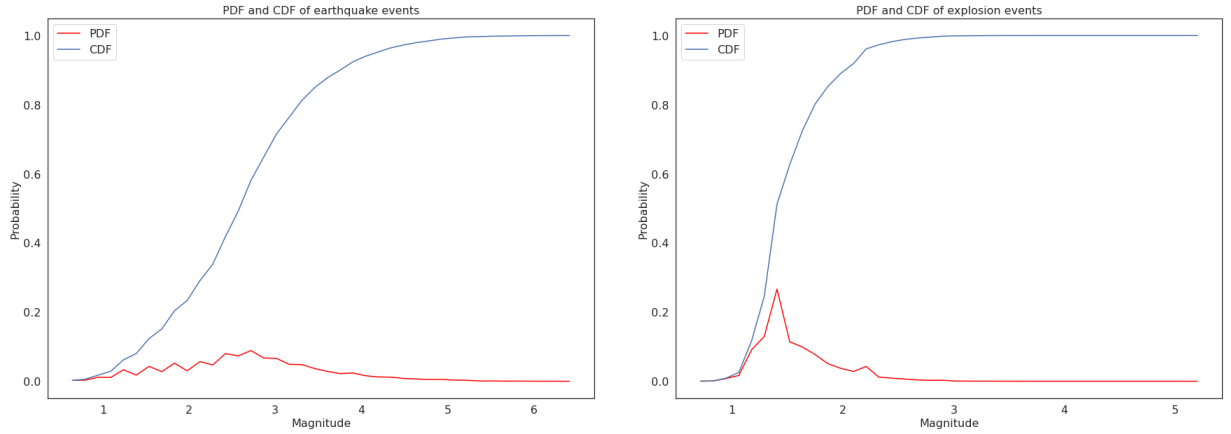
4.1 Description Of The Dataset

The NORSAR dataset is collected by the ARCES array outside of Karasjok, Finland. These events are manually analyzed using all of NORSARs' stations and arrays, but the waveforms used in this project were recorded at ARCES. In addition, these events are suplemented with Institute of Seismology of the University of Helisinki's bulletin. NORSARs' own bulletin is publicly available [27], although this information is less complete than what is available for this project. Throughout the development of this project, the data set was updated to contain the most recent events, the chronological order of which is referred to by batches. The data set is made up of several samples of waveform data, as well as information about each event. The information contained in the information section of each sample differs between each class of event, and some of the type of information differs between the "batches". The earliest recordings in the dataset are from January, 1991, while the most recent is from November, 2020.

4.1.1 First Batch

The first batch of the dataset contains 206k three-component events. The class distribution of the dataset is 6852 earthquake events, 107786 mining explosion events, 606 mining induced

earthquake events, 91612 recorded noise events. Each waveform contain 6001 timesteps from a three-component recording of 150 seconds. Every recording has been beamformed (see seismology section) prior to use in this project, which reduces some of the noise in the data. The data has been manually labeled by NORSAR analysts, and regardless of the label, the majority of the timesteps for each sample consists of noise. For the earthquake and explosion events, the timesteps of interest are not labeled, and may start at any point within the interval. An earthquake/explosion event may occur in the last few timesteps, while still being labeled an earthquake/explosion. The beginning of the events may also not be included in the recording whilst still being labeled an earthquake or explosion.



(a) PDF and CDF of magnitudes of earthquakes in the first batch.

(b) PDF and CDF of magnitudes of explosions in the first batch.

Figure 4.1: Magnitude distribution of earthquakes and explosions in the first batch of the ARCES dataset. Both graphs are created with a bin size of 40.

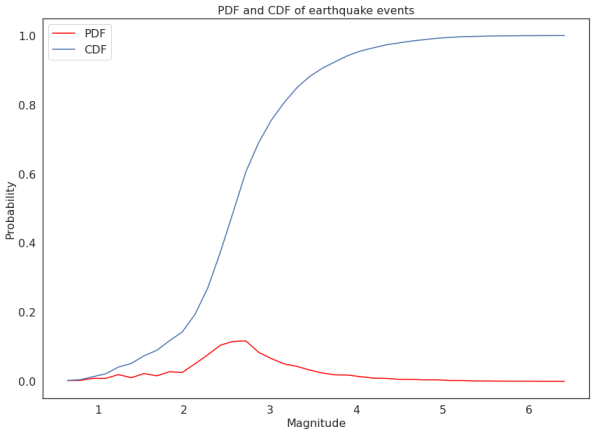
This first batch of data was smaller than the second batch, due to several events in the first batch containing NaN values. The plots in 4.1 show the pdf and cdf of earthquake and explosion events of this dataset. As seen in 4.1a, the magnitudes of the earthquakes in the data set are normally distributed with a positive skew. This skew is less prominent in the second batch, as this first data set contained many NaN values, all of which were excluded from these plots. This first batch of data was not used as much for modeling, but rather to develop a large portion of the pipeline in preparation for the second, and more intact batch of data.

4.1.2 Second batch

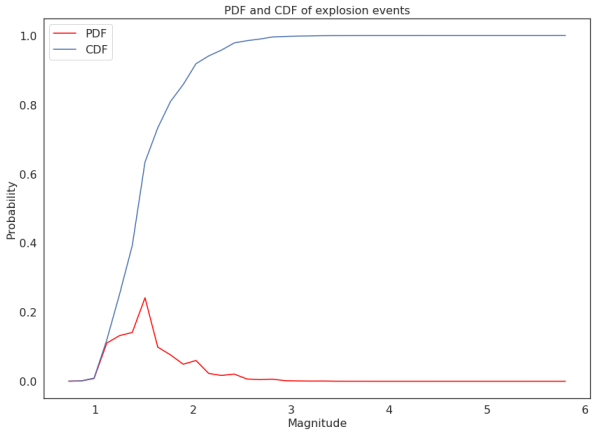
The second batch of the data set contains 235k three-component events. These events are distributed by 9464 earthquake events, 111173 mining explosion events and 124048 recorded noise events. Similarly to the previous batch, all events are beamformed prior to the use in this project. Unlike the first batch of data, the waveforms in this batch contained 9460 timesteps rather than 6001. With a sample rate of 40 Hz, these samples are 240 seconds long. The increase of 2612 earthquakes in the second batch is due to several samples containing NaN values in the first batch being omitted as well as recently recorded earthquakes being included.

The graphs in 4.2 show the distribution of magnitudes and Magnitude Square Root Distance Ratio (MSRDR) of earthquakes and explosions in the second batch of data. Explosions events are more likely to be of a lower magnitude event than earthquakes, but still contain events measured at a magnitude of 5.8. The smallest explosion is recorded to have a mangitude of 0.6 . For the earthquakes, the event of highest magnitude is measured at 6.4, while the lowest is at 0.5. The earthquake events appear to have a distribution similar to the normal distribution, although with longer tails. The explosion events appear more as a beta distribution, where events with magnitude less than 2 make up the vast majority of the dataset. The MSRDR is a metric can be interpreted as an indicator for the signal to noise ratio. This ratio gives an indication of the level of the desired signal to the level of background noise. A high value of MSRDR indicates that the event is more distinct, while a low value indicates that the interesting signal is less prominent. This property is important as waves transfer energy in the form of heat to the environment as it travels. A distant and low magnitude event is harder to detect, than a high magnitude near event. The distributions show that the vast majority of the samples are on the lower end of the MSRDR spectrum. The explosion events are more evenly distributed in the lower end of this scale, while the earthquake events follow a distribution similar to the beta distribution, but with a higher range of MSRDR than explosions. Looking at the magnitude distribution of earthquakes in figure 4.2a, and table 4.1, the mode of earthquakes is higher than explosions, and the epicentral distances are more evenly distributed. Explosions tend to be of lower magnitudes as well as nearer to ARCES.

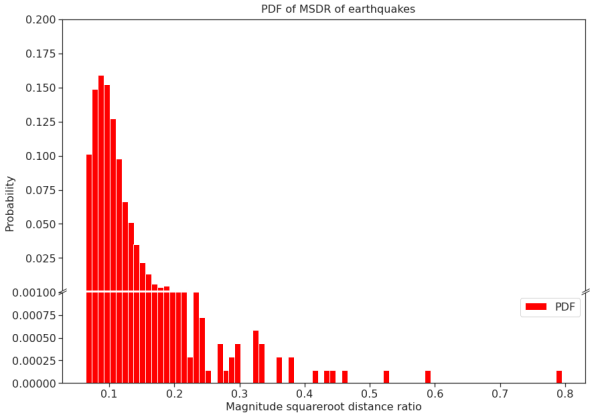
Since all recordings in the dataset contain noise, low magnitude earthquakes and low MSRDR earthquakes will be more challenging to classify. In addition, there is significantly



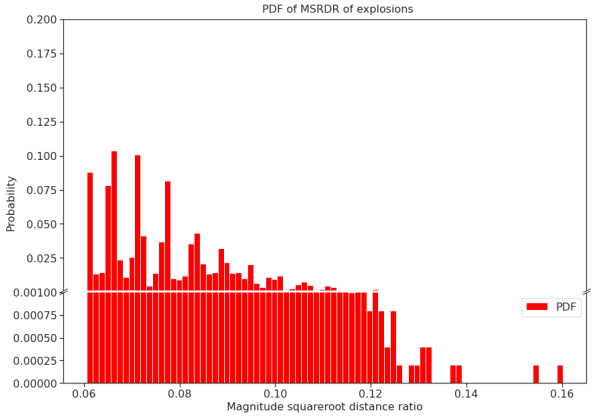
(a) PDF and CDF of magnitudes of earthquakes in the second batch.



(b) PDF and CDF of magnitudes of explosions in the second batch.



(c) PDF of MSRDR of earthquakes in the second batch.



(d) PDF of MSRDR of earthquakes in the second batch.

Figure 4.2: Magnitude and MSRDR distributions of earthquakes and explosions in the ACRES dataset. Both magnitude graphs are created with a bin size of 40, MSRDR graphs uses a bin size of 80. Note that the scales are different in the MSRDR graphs.

fewer earthquake samples than noise or explosion events, which requires up/down sampling combined with augmentation methods to overcome. The process of artificially inflating the data set and other methodology employed in this project will be outlined in detail in section 4.2.

Epicentral distance interval (km)	**Number of explosion events**	**Number of earthquake events**	**Total number of events**
[0, 200)	4972	432	5404
[200, 400)	84521	2238	86759
[400, 600)	11251	1266	12517
[600, 800)	3628	809	4437
[800, 1000)	2607	1804	4411
[1000, 1200)	3721	1467	5188
[1200, 1400)	252	820	1072
[1400, 1600)	102	359	461
[1600, 1800)	65	183	248
[1800, 2000)	24	58	82
[2000, 2200)	21	17	38
[2200, 2400)	8	8	16
[2400, 2600	0	2	2
[2600, 2800)	1	1	2

Table 4.1: Distribution of the distance to the epicenters of non-noise events

The events in the second batch range from 10 km to 2700 km, where the vast majority of recorded events occur in the range 200 to 400 km from ARCES. Relatively few events have an epicenter further away than 1200 km, as seen in table 4.1. The decision was made to include teleseismic events, as removing them would reduce the number of earthquakes disproportionately. In addition, these events are not completely dominated by noise, providing value to the learning process. In addition these events mimic the signatures of weaker, but nearer earthquakes, helping the network classify weaker events. The distribution of the distance of explosion events at shorter intervals can be seen clustering around their quarry/mine as expected. As the networks are not using this information, this does not directly affect the classification of the models in this project. The location of the epicenter combined with the time of day plays an important role when classifying these events manually, but this information is unavailable to the models in this project.

This batch of data contained more detailed information about the events. Included in this information, but not in the first batch, is a combination of manually and automatically

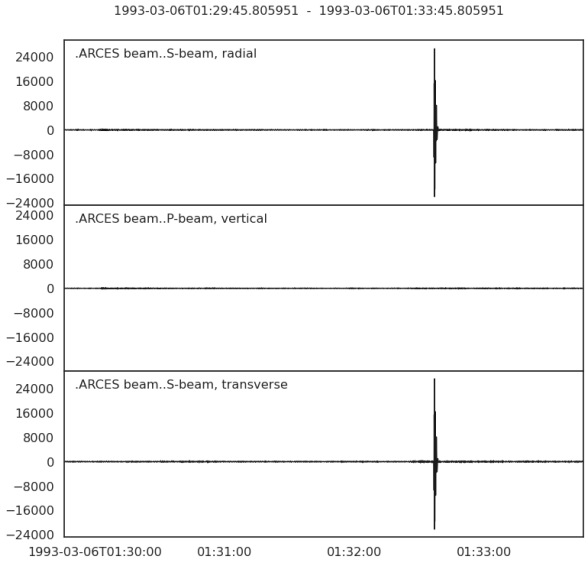
labeled timesteps of interest. These timesteps inform where the event starts to occur. These timesteps are always after at least 60 seconds of information (presumed to be) irrelevant to the event of interest. With this labeled timestep, a time uncertainty is included which in combination provides an interval in where the start of the event occurs. These longer recordings, combined with labeled time of interest with a lower limit, allow for augmentation of the recordings to a greater degree than the first batch. That is, the training set may be artificially inflated with unique samples, which maintains the important relationships of the waveforms where interesting events occur. Simultaneously, this allows for a reduction in the amount of noise which previously made up a large amount of the 4 minute recordings. This is a key tool used for offsetting the limitations of the relatively few earthquake samples in the NORSAR dataset. Unfortunately, like the first batch, the recordings do not necessarily include the entirety of the event. Certain recordings may only contain a few seconds of the interesting event labeled while still being considered an earthquake/explosion, however the onset of the event is always included.

4.1.3 Outliers And Problematic Data

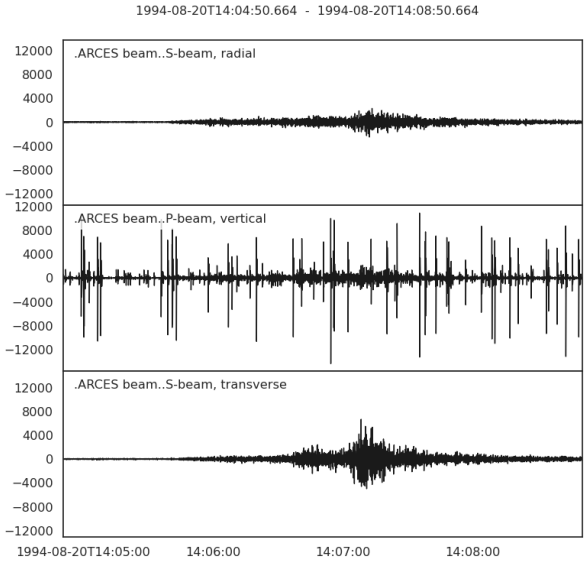
The majority of the recordings received from NORSAR were varied, but reasonable datapoints. These events, although nearly impossible to distinguish visually, appeared as recordings of natural events. However, a not insignificant number of events did not appear to contain natural recordings. These outliers expressed themselves in different ways, but likely had a notable negative effect on the performance of the models in this project. Some outliers had, perceived, large random spike(s) greatly trumping the otherwise normal looking recording in scale, see 4.3a and 4.3b. Others contained very strong expression in one, 4.3d, or several channels, 4.3e. Two outliers, occuring in 1993,4.3c, and 1994, 4.3f, look very similar, although one with twice the expression in the y-axis.

The examples shown in figure 4.3 illustrate the presence of outliers in the dataset, but is by no means a complete list. Due to the size of the dataset, it is infeasible to manually inspect every single event, and thus an accurate number of how prevalent these outliers are is hard to quantify. However, a noteworthy quantity was found while inspecting a subset of the data to consider the outliers' presence significant.

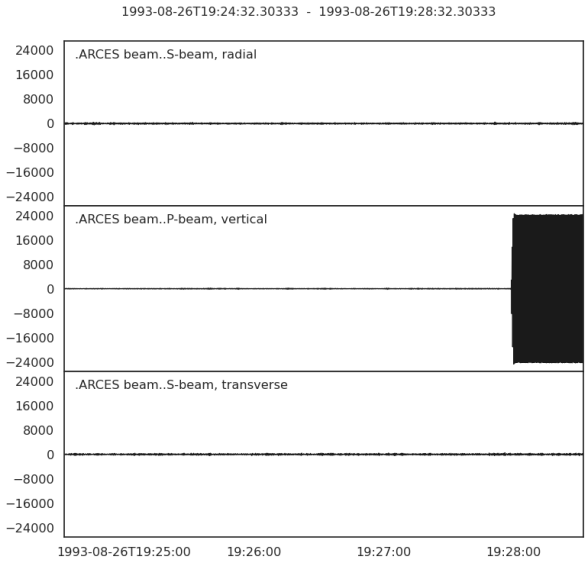
The problem of incorrectly labeled events poses further difficulty to the classification task. Distinguishing between explosions and earthquakes is challenging, and within the



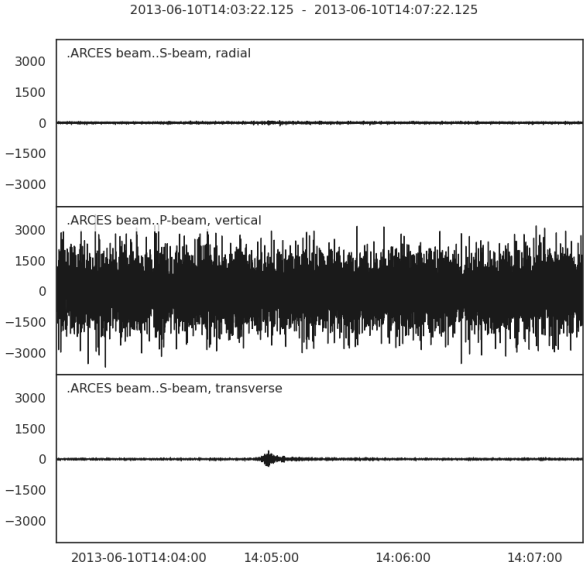
(a) Outlier waveform, labled earthquake of magnitude 3.3 epicenter 1561 km from ARCES, with large spike in both S-beams. The P-beam behaves similar to the normal waveforms.



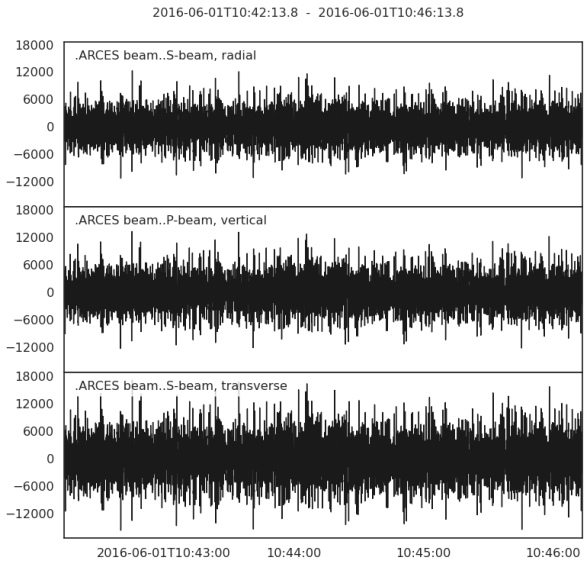
(b) Outlier waveform, labled earthquake of magnitude 2.8 epicenter 583 km from ARCES, with large spikes in the P-Beam. Both S-beams appear normal.



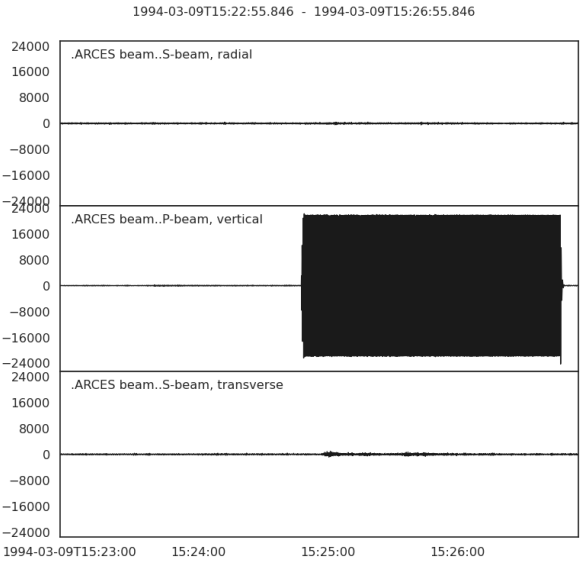
(c) Outlier waveform, labled earthquake of magnitude 2.8 epicenter 1358 km from ARCES. Constant and severe expression in the P-beam.



(d) Outlier waveform, labled explosion of magnitude 1.3 epicenter 282 km from ARCES. Uncharacteristically strong expression in the P-beam. Difficult to visually detect any event of interest in the P-beam as it looks similar to the noise recordings, although much stronger displacement. The transverse S-beam, however, appears to behave as expected.



(e) Outlier waveform, labled explosion of magnitude 1.9 epicenter 718 km from ARCES. Uncharacteristically strong expression in all beams, considering the distance and magnitude. Indistinguishable from noise in all channels, were it not for the extreme expression.



(f) Outlier waveform, labled explosion of magnitude 2.3 epicenter 792 km from ARCES. Constant and severe expression in the P-beam.

Figure 4.3

information section of each event is the label preceded with words such as "probably" and "likely", leaving room for error. According to NORSAR, classifying events of magnitudes less than 1.3 is particularly difficult, and there is a higher likelihood of misclassified events in this range. Incorrectly labeled events are detrimental to the learning process of a neural network. Yet, there are techniques available to mitigate their effects. Label smoothing is one of these techniques [26], but were not implemented in this project. This decision was made in order to focus optimization on other hyperparameters, as the number of incorrectly labeled events is unknown.

Another issue, although more general to this type of data, is that more than one interesting event may occur on a single recording, while only one type being labeled. A recording labeled earthquake may contain several earthquakes and/or explosions in the same waveform. The labeled event may be insignificant in its expression compared to other events. This does not appear to be a significant issue in the events labeled noise, but is problematic when training a model to distinguish between explosions and earthquakes. Outliers can also be found in the noise recordings, distinguished by atypical displacement in stacked counts or

features clearly deviating from the norm. As these events are not labeled differently by the analysts, these are perceived as neither earthquakes nor explosions thus considered noise.

When attempting to augment events of interest using the time augmentor, described in section 4.2, the presence of several, irrelevant, and potentially more expressive events, becomes a considerable drawback to the augmentation effort. The cutting and moving around different segments of recording containing several interesting events is likely often detrimental to the structural integrity of the waveform.

4.2 Preprocessing

Preprocessing is an important aspect to the machine learning process. Preprocessing refers to all the transformations of the raw data prior to being fed to the machine learning algorithm. Raw data often contains non-relevant information detrimental to the algorithms, the inclusion of means the model needs to learn their irrelevance. Including irrelevant data unnecessarily increases the dimension, which is problematic due to the curse of dimensionality. High dimensional data means that the volume of the space is very large, and a significant amount of data is necessary to have a representative sample. A model which struggles to fit on data with many input features is generally known as the curse of dimensionality. Preprocessing consists of reducing the data to its most useful form, removing unnecessary information and transforming the data to help maintain numerical stability.

The decision of how to transform the data into its most valuable form to the models is a key element in this project. The ARCES dataset in its raw form is not suitable for Neural Networks. The dataset is significantly unbalanced, with the number of earthquake events making up a fraction of the number of noise/explosion events. The length of the recordings results in the majority of the events consisting of pure noise, further complicating the classification task. In addition, analysts use a variety of filters during signal processing in order to enhance the intervals of interest, simplifying the manual classification task. Throughout the development of this project, these factors became important to account for in order to improve performance of the models.

In this chapter, the various methods used to handle these challenges is described in each subsection. In subsection 4.2.6 the entire preprocessing pipeline is outlined, describing the order of operations necessary to handle dependency issues resulting from the transformations.

4.2.1 Balancing the dataset

The class distribution of the dataset is crucial to consider while doing machine learning. Depending on the task, this statistic has far-reaching downstream effects in the performance of the model, as well as how to interpret the resulting metrics. For example, a two-class dataset, with a 95% population of class 0, will yield a binary accuracy of 95% for a model which always guesses class 0. This deceiving performance metric, depending on the objective function, may stump the learning process, and result in a useless model. During the development of this project, the distribution of the data during the training phase has played a central role in the preprocessing pipeline.

There are several ways to handle an unbalanced dataset, two of which being used in this project were upsampling and downsampling. Downsampling consists of randomly selecting samples from the dataset such that the resulting dataset contains less events than the original. Similarly, upsampling which consists of randomly selecting (with replacement) from the original dataset such that the resulting dataset contains more events than the original.

The vast majority of the events in the NORSAR dataset is noise and explosion events, 124k and 111k respectively. Earthquake events make up 3.89% of the events in the second batch of data. Each recording contains thousands of datapoints, meaning that downsampling the other two classes will lead to a very sparse dataset. On the other hand, upsampling, especially utilized to such an extent in this project, leads to a lot of redundancy with little variance in the data, meaning that the risk of overfitting is increased. Further preprocessing was therefore necessary to counteract these effects, which are described in the later subsections.

The upsampling and downsampling methods have been used in several different ways in this project, depending on which type of model is being trained, and how many classes were to be included. The initial balancing consist of downsampling the number of the most numerous class to match the second most numerous class, and to upsample the least numerous class to match the second most numerous class. This way each of the 3 classes have the same amount of events. Following this balancing, during the model selection and optimization stage of this project, a subsample of the balanced dataset is selected in order to reduce: computational demand, effect of redundant upsampled events, and reduce training time.

In addition, for the two-in-one model described in subsection 4.3.1, downsampling was also used in order to get even distribution between the noise and the non noise events. The general distribution of the data looks after balancing follows this formula: $n_i = \frac{\text{Total number of events}}{\text{Number of classes}}$ where n_i is the number of events of class i. For the first model, explosions and earthquakes were treated as having the same label: not noise. The explosions and earthquakes making up the non-noise class is populated with equal number of earthquakes to explosions.

4.2.2 Time Augmentation

The disproportionate amount of earthquake events in the dataset, and the necessitated upsampling produces a significant amount of redundant samples in the dataset. Individual earthquake events may be duplicated several times, which leads to overfitting issues. In order to combat these effects, two augmentation methods were developed: "Time Augmentation" and "Noise Augmentation". Noise Augmentation will be described later in this chapter.

Data augmentation are techniques used to artificially inflate the amount of data by adding modified copies of existing data. These methods are often used as a regularizer to reduce the effects of overfitting, in this case as a result of upsampling an underrepresented class.

In the second batch of data received from NORSAR, the start time of the event of interest were included for each of the non-noise events. Ahead of the event of interest in these recordings were at least 60 seconds of information not relevant to the event. For a large portion of the data, this buffer consisted of pure noise which could be shifted without affecting the structural integrity of the event of interest. The buffer also meant that the recordings were longer than those of the previous batch. In addition this meant that each recording contained at least the onset of the event, although still were not guaranteed to include the entire event. The increased length, of the events meant that the proportion of noise to actual event were increased, and contained the information necessary to reduce the length of the recording, while maintaining the important elements. The combination of the 60 second buffer and labeled start time of the event allowed for an augmentor to be written that takes advantage of this information. This augmentor, referred to as Time Augmentor, became a powerful tool to allow for upsampling, and mitigated the strain of the limited dataset.

The augmentor reduces the length of the recording to match the same 2 minute 30 second duration of the first batch, but shifts the event of interest around and fills the surrounding space with information, in most cases, not relevant to the event of interest. The first step of the augmentor is to map a redundancy index to each of the events in the dataset. This index is used to distinguish between redundant events, allowing for each event to be augmented the same way every time it is used.

After this index has been mapped and each event has a unique identifier, the augmentor is fitted. The fitting process iterates through every event in the dataset, shifting the part of the waveform containing the event of interest. This shift is limited to start between 0 seconds and 50 seconds, in order to preserve the integrity of the area of interest. The space surrounding the shifted event of interest is filled with noise from areas with a high probability of containing noise.

The empty space prior to the onset, is filled with the information from the buffer, and if necessary, with a slice, with duration equal to the space required, from the end of the original waveform. In the manually inspected waveforms, all events of interest were labeled to start near the 60 second mark, but the augmentor is able to handle cases where the event of interest occurs near the end of the original waveform. These recordings contain a long stretch of irrelevant information preceding the event of interest. If events such as these are shifted so that they occur early in the augmented event, the end of the augmentation is left empty. In these cases where the span of the event of interest is short of the period needed to fill the augmented waveform, the end is filled with a slice of the recording preceding the onset of the labeled event.

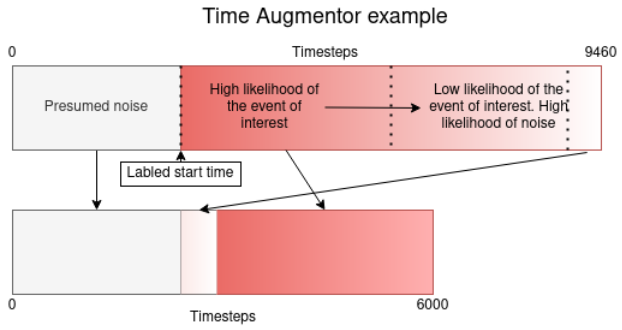


Figure 4.4: Illustration of how the time augmentor shifts the event of interest, and fills the gaps with presumed noise.

This augmentor has shown to be very effective in its goal to reduce overfitting, but has certain downsides which is detrimental to the learning process, and poses a challenge to its ability to infer onto unseen data. There is no guarantee that a recording contains a single isolated event. Manual inspection of the waveforms show that some recordings contain the end of a previous event in its 60 second buffer. The augmentor operates with the assumption that the buffer can be shifted without affecting the integrity of the labeled event. For cases were this assumption is violated, the structural integrity of the waveform is compromised, and is a shortcoming of the augmentation process.

Other events, where the earthquake or explosion is far away, the labeled start time show when the initial wave arrives, but the visually clear event arrives several minutes later. Reducing the length of the recording risks cutting out information of significance. If the event is shifted so that it occurs very early, this reduces the chance of this problem, but does not eliminate it. An argument could be made that this is a shortcoming of the dataset in general as a result of its fixed length, but worsened by additional shortening.

A few recordings contain its labeled event, but also include subsequent unlabeled earthquakes and explosions. The augmentation process will generally leave the labeled event intact, but the unlabeled event may be cut and shifted, creating irregular and unnatural structures in the waveform. Some events may contain both earthquakes and explosions, but has only single event labeled. This is a problem with or without the augmentation, however, the impact of multiple events on a single waveform may be worsened by the augmentor.

Despite these shortcomings, the problem of duplicate recordings is serious, stemming from upsampling necessitated by the unbalanced dataset, whose effects need mitigation. Overall, the dataset contain recordings suited for such a solution, and shows promise in deployment. As with anything in machine learning, there are advantages and disadvantages to any solution, and the advantages of this tool appears to outweigh its disadvantages.

4.2.3 Filters

The raw data from NORSAR is noisy. The waveforms contain significant amounts of uninteresting and unimportant information which analysts typically apply a variety of techniques to reduce and overcome. Filtering is a commonly used tool utilized in order to enhance the

signal of interest in the data. Filtering is the practice of modifying a signal in order to remove undesirable aspects of the signal prior to its use in calculation or classification.

The success found from filters in the manual approach in seismic classification, indicated that it could yield positive results when applied to the preprocessing pipeline of this machine learning application. A few filters, such as highpass and bandpass, were experimented on at different frequencies, with varying success in the two classification tasks depending on the model. Further experimentation on filters could be very impactful on further research, but considering the momentous search space, its discouraging performance and time limitations, more fine-grained experimentation were considered out of scope.

In general, the use of filters yielded varying results depending on the classification task. When classifying noise and not noise, the use of filters had a positive impact, often improving the performance during model selection. In contrast, when classifying between earthquakes and explosions, the use of a filter had detrimental effects, significantly stumping the model selection process, largely producing useless models.

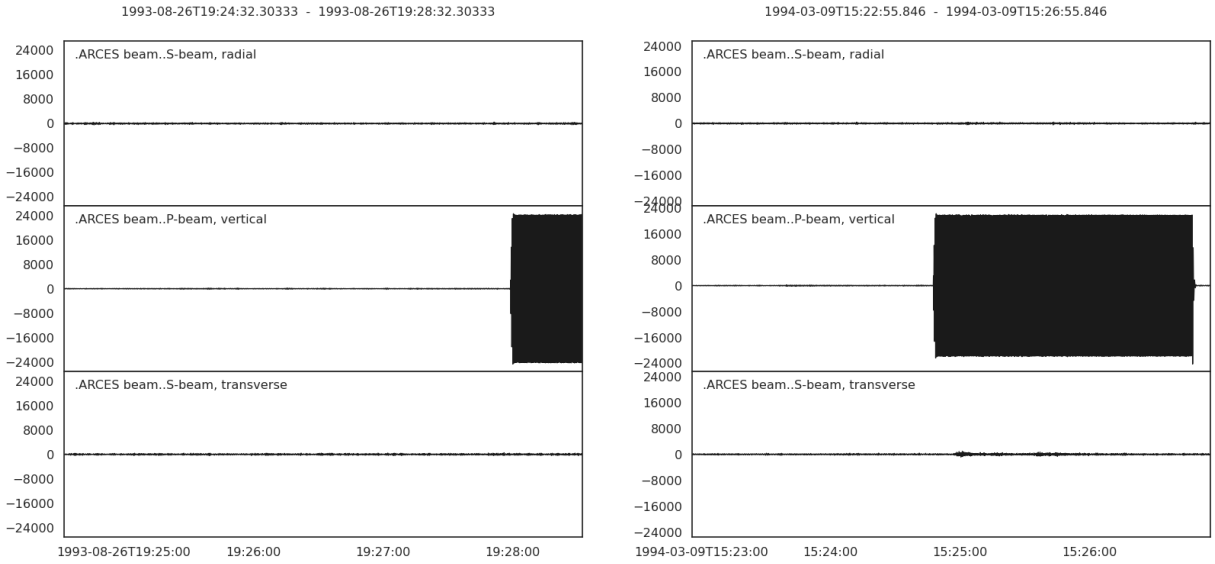
There are two main drawbacks to using filters in this project. The first and most important is that distinguishing features appear to be less prominent when applying filters. This is seen with the general low performance during the model selection phase of the earthquake/-explosion classifiers. The second drawback is that some recordings become indistinguishable from noise when filters are applied. On the other hand, other recordings are indistinguishable from noise when no filters are applied. These two drawbacks may, at least in part, explain the general low performance using filters in the earthquake/explosion classifiers. Further, it is important to note that when performing manual classification of these events, analysts use different filters depending on the distance to the event as well as the magnitude. Through a process of trial and error the use of filters is a very important tool for enhancing the signal of interest, and improving the signal to noise ratio. However, since one specific filter is not applicable to all events, it is not surprising that filters do not yield good results for this project.

4.2.4 Scaling

Scaling or normalization are techniques used in many machine learning applications. For models which use Gradient Descent based algorithms, having scaled data aids the descent

to converge the the local minimum faster, and aids numerical stability. This is a result of the gradient descent formula, where the scale of the feature has an impact on the step size of the gradient descent. In effect, features with a larger range, will have a larger effect on the step, while features with a smaller range will be less important.

In practice, the range of input values in data can vary widely from one dimension to another. If one is to calculate the euclidean distance between a set multidimensional points, with one dimension with a much wider range of possible values, this one dimension will govern the final distance. Different scaling techniques attempt to transform the data, such that each feature contributes proportionally to the final distance. The choice of scaler can have a significant impact on a neural networks ability to learn from data, and picking one that transforms the data in a productive way can be key to developing useful models.



(a) *(b)*

Figure 4.5: Two outlier waveforms labeled earthquake (4.5a) and explosion (4.5b), both with strong continuous expression in the vertical component

The various scaling techniques experimented upon in this project has largely been chosen based on how they handle outliers in the data, and how their transformation impacts the model performance. There is a small (but not insignificant) amount of faulty recordings in the dataset, as discussed in the dataset subsection of the first chapter. These outliers often distinguished themselves by unnaturally strong expression in the stacked counts of the

waveform. In some cases this strong displacement were continuous, see 4.5 for examples of this. Other outliers have short, but strong, regular expressions in the displacement axis, which negatively impact the scaling process.

Fitting is a term used to describe the initialization phase of scalers who depend on information about more than a single waveform. This phase looks at all of the data in the training set and stores information necessary for the transformation. For all of the scalers used and discussed, there are a few ways in which one can fit and transform a scaler to this type of data. For example, one can fit a scaler so that it scales each component independently, one can transpose the data so that each timestep acts as a feature, or one can fit a scaler such that all of the features are scaled the same. Only the first two were experimented on for this project. Transposing the data so that each timestep acts as a feature did yield some positive results, particularly in the noise-not-noise model, but had unintended effects with scalers such as standard scaler and minmax. Transforming each component independently seemingly yielded the best results. Further experimentation on scalers could potentially be very impactful in future research.

Min-Max Scaling

Min-Max scaling is a normalization technique which transforms each component/feature so that the value at each time step is in the range [0,1]. Let x represent a waveform and X represent all waveforms in the set in which the scaler was fitted, then the formulation of the MinMax scalers transformation is this:

$$x_{scaled} = \frac{x - \min(X)}{\max(X) - \min X} \tag{4.1}$$

The Min-Max scaler, prior to transformation, is fit to the entire training set. From the training set, it finds the maximum and minimum value for each feature. As a result, this technique is very sensitive to outliers in the data. In particular in cases where there are features with absolute values that far exceed what is normally present in the data. In addition, there is a great range of values in the stacked counts for all of the channels in the data, and so waveforms tend to be overly squished. This scaler is therefore deemed to not be useful for this project.

Standard Scaling

Standard scaling transforms each feature to be centered around the mean of the fitting set, with a unit standard deviation. This helps each feature to have a proportional impact on the gradient descent, but does not bound the range of values and so the standardization process is not as affected by outliers in the data. Again, let x represent a waveform and μ and σ represent the mean and standard deviation respectively of all waveforms in the set in which the scaler was fitted. The formulation for the transformation done by the standard scaler is this:

$$x_{scaled} = \frac{x - \mu}{\sigma} \tag{4.2}$$

As with the Min-Max scaler, this technique fits to the training set prior to transformation. The issue with this scaler is that the magnitude of the displacement in the stacked count axis is not the only distinguishing factor among the three classes. However, the network will quickly learn that a strong displacement in stacked count typically means that there is an interesting event occuring, and will overfit to this concept. As this displacement in stacked counts is a function of distance and magnitude, some events of interest will be squished to look like what the network interprets as noise. What is more important is to maintain the structural integrity of the waveform. For this data, the standard scaler tends to overly squish events with a weak magnitude and/or with a distant epicenter, which may explain the less than optimal performance when used in certain models. So despite its robustness to outliers, the difference in magnitude in the stacked count axis among the waveforms makes the use of this scaler problematic. However, with a more selective filtering process of the dataset, the downsides of this scaler may be mitigated.

Normalize Scaling

Normalize scaling is used to scale each component to its unit norm. There are three different norms which can be used, where L2 norm is the one used in this project. Further experimentation with different norms may yield positive results, but were deemed out of scope for this project. The transformation of this scaler on a waveform x is given by the formula:

$$x_{scaled} = \frac{x}{\|x\|_2} \tag{4.3}$$

Unlike the previous scalers, this one does not have a fitting phase. Each sample is independently transformed from other events which causes them to have a similar range of amplitudes. As a result, the outliers do not have a spillover effect on the other transformations. Each channel in the waveform is transformed independently from the others, which removes the relative difference among the channels. In addition, the effect of distance and magnitude is lessened, and the network cannot learn the concept of strong stacked count displacement means that something interesting is going on. On the contrary, events with a strong displacement in the stacked count axis, have their noise scaled down relative to the signal of interest. The network could learn that in transformed waveforms with areas of little displacement, are likely to not be pure noise waveforms. Despite this drawback, this scaler has consistently performed well.

Robust Scaling

Robust scaling transforms the waveform by removing its median and scales the data according to the user defined quantile range. This quantile range is by often 75%-25%, which is what was used in this project. This scaler does not have a limit on the range of values for each channel, and transforms all waveforms to have similar magnitudes in terms of the stacked count axis. The transformation applied to a waveform x by the robust scaler can be calculated using this formula, where $Q_1(X)$ and $Q_3(X)$ represents the first and third quantile respectively of the waveforms in the fitting set X:

$$x_{scaled} = \frac{x - \text{Median(X)}(X)}{Q_3(X) - Q_1(X)} \tag{4.4}$$

Like with Standard and MinMax scaling, this scaler is fitted to the training set where it calculates the component-wise quantiles necessary for the transformation. As outliers can often influence the mean and variance in a negative way, the median and interquartile range can often give better results. This proved to be the case for this project, and had performance comparable to the Normalized scaling technique. The two differ in that the Normalized scaler transforms waveforms to be much more similar in magnitude than the Robust scaler, but the Robust scaler does appears to maintain the magnitude of displacement in the stacked counts axis of noise.

4.2.5 Noise Augmentation

As an additional measure to limit the risk of overfitting as a result of the redundancy caused by upsampling earthquake events, another augmentation method was developed for this project. This augmentation technique applies a small amount of gaussian noise to each batch of data in the generator. Unlike the other preprocessing methods, this noise is not applied prior to the training process, but rather applied batch wise. This means that the same event is slightly different every time it is fed to the network, and thus, making it more difficult for the network "memorize" the training set.

This augmentor is initialized differently dependent on which classification task the network is to perform. If the network is to classify between noise and non-noise events, the augmentor uses the noise samples in the training set during initialization. If the classification task is to distinguish between earthquake and explosion events, a subsample of noise events in the dataset is used.

The initialization process consists of iterating through all the preprocessed noise events, calculating the average standard deviation. The augmentation occurs batch wise, through the generator, adding Gaussian noise with zero mean and a standard deviation equal to 1/15 of the average standard deviation of the sampled noise events. When using normalize scalar, the fraction of average standard deviation is 1/20. This is because the noise samples when transformed with normalize scaler have a similar magnitude to the events of interest, and so the augmentation has more of an impact. Further experimentation of these fractions could yield positive results.

This augmentation process has minor impact on the structural integrity of the recording, but still has a downside. For events of either low magnitude and or events that occur very far away from ARCES applying even a minor amount of noise even further obscure the labeled event.

This augmentation technique, developed during the early stages of the project, had immediate impact on the current overfitting issue, and continued to prove a valuable tool in all future model development. This augmentation tool were also useful as it allowed for a reduction in "time augmentation" shift interval (0 to 50 seconds) in order to reduce potential impact on structural integrity.

4.2.6 Preprocessing dependency outline

The development of this preprocessing pipeline required careful consideration of order of operations. The key goal for the preprocessing stage is to retain the structural integrity of the events of interest, while squeezing out as much value as possible from the limited and unbalanced dataset without compromising the data or models. Performing the various steps in the correct order is critical in the pursuit of this mission.

The augmentation of noise, for example, depends on every other preprocessing step in order to produce the desired outcome. In order to get an accurate mean standard deviation, each event needs to be transformed to its otherwise final form, or the added noise will obfuscate the events of interest.

Similarly, the scaling of the data depends on largely on the use of filters as well as the time augmentation. The filtering process impacts the range of values in the features, smoothing out the signal. The time augmentation shifts the different segments of the recording, affecting the range of possible values of each time step. If these steps are not considered during the scaling of the data, the scaling technique will compromise the structural integrity of the data, and stagger the models' ability to learn patterns in the data.

The chart in figure 4.6 provides insight into the order of operations of the preprocessing steps. The exact process differs depending on whether or not the data is to be loaded directly to RAM or if the generator used by the model loads the data batch-wise, but the preprocessing is the same for either. The splitting of the dataset into training, validation and test occurs prior to this flowchart, and the treatment differs among each. The scaler, if required to be fit to the data, is only fitted to the training set. This scaler is used to transform all the data in every dataset. The noise augmentor is only fitted with events in the traning set, or what would be if in the training set if noise samples were to be used.

4.3 Model Selection

Central to the machine learning process is trial and error. The No Free Lunch Theorem has exhaustively proven itself throughout this project. There is no model that works as is for any

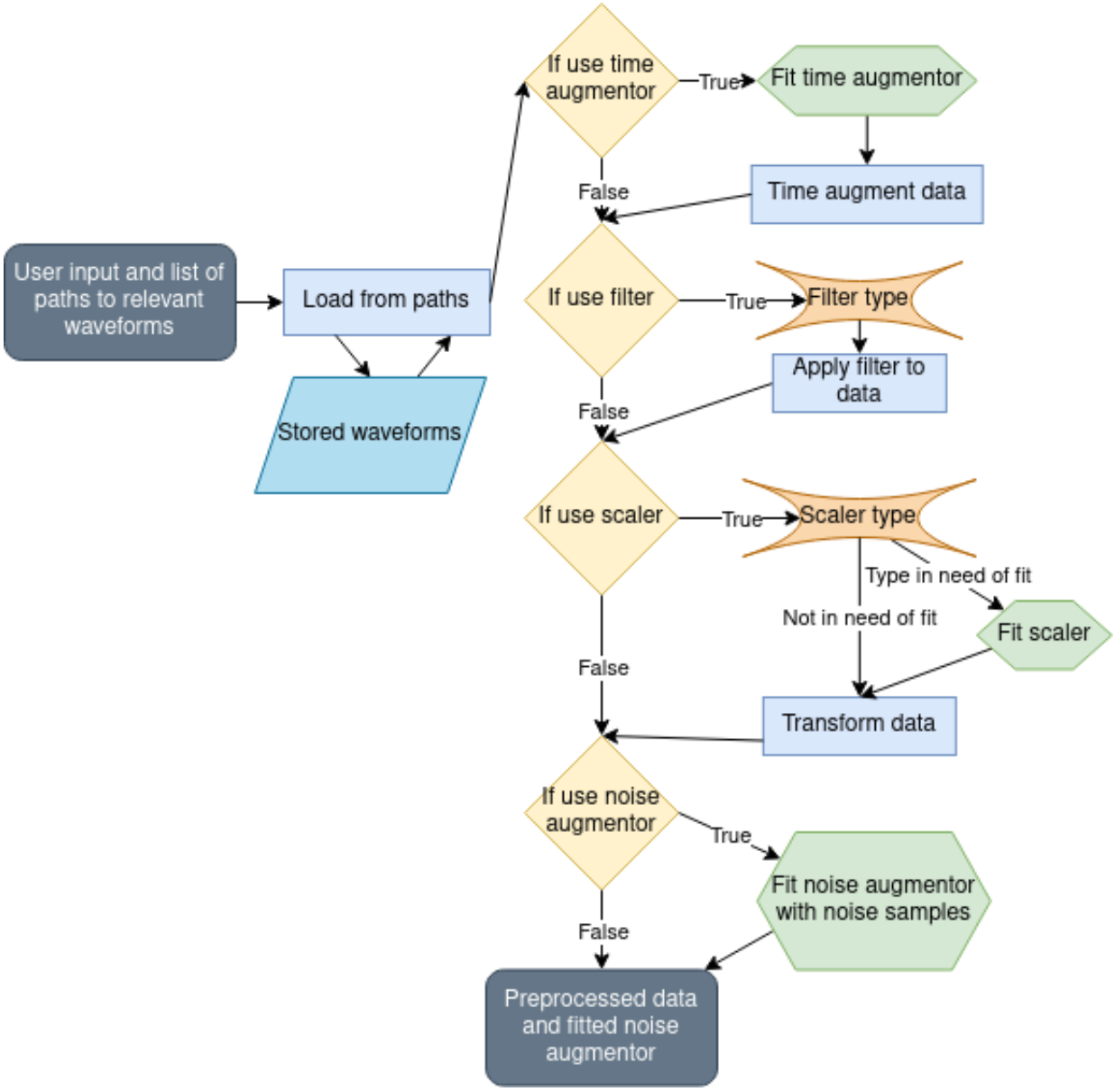


Figure 4.6: Flowchart describing the general preprocessing outline.

problem, and extensive experimentation is required in order to determine its usefulness. For most problems in deep learning one could spend lifetimes attempting to find the best models and their optimal combination of hyperparameters for a single problem and still never find it. It is an end which only exists in the abstract, one that can never be achieved. Yet, some models are better than others, and less than perfect models can still be extremely valuable. Finding useful model is the name of the game, and several tools are at disposal to this end.

Every type of model required a model selection phase for each classification task. The model selection comprised of two main steps: random grid search and a local search of significantly reduced search space, referred to as a local search. In order to reduce the computational strain of training thousands of models, a subsample of the dataset was used in this process. The number of events in this subsample were sufficient to prevent issues from overfitting, and allowed for a greater number of models evaluated. All models were trained with the same events in each of the datasets (training and validation), depending on the classification task, so that the performance metrics of each could be compared objectively.

4.3.1 Two Models In One

Early in the development, the models tended to be better at classifying noise correctly, but struggled to differentiate between earthquakes and explosions. Upon inspection of the waveforms, it quickly became clear that the difference between the noise events and the earthquakes and explosions were greater than the difference between the earthquakes and explosions. As a result, models performing well at distinguishing noise from the other classes reported better metrics, and were thus selected for.

It became evident that a separation of the classification tasks were in order. This separation allowed for one model to be optimized for differentiating between pure noise events and non-noise events, and another were to be optimized for distinguishing between explosion and earthquake events. The flexibility allowed the two tasks to have their own tailored solution, without compromising each other. On the other hand, the two models would potentially require different preprocessing steps, which complicates the pipeline and expanded the search space tremendously.

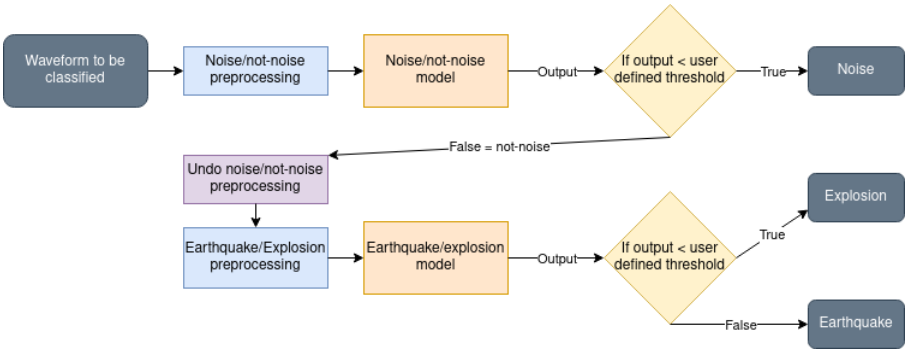


Figure 4.7: Simple flowchart describing the structure of the two-in-one approach.

A high level flow chart of the pipeline can be seen in 4.7. The noise not-noise model works as a filter, reducing the presence of noise samples, allowing the earthquake/explosion model to be optimized separately. This flowchart is a simplified, but accurate description of the process.

4.3.2 Random Grid Search

The random grid search works by selecting a number of combinations of hyperparameters defined prior to training, and then iteratively training the model with these hyperparameters, and finally saving their performance metrics. The search space of hyperparameters is massive and could not reasonably be exhausted. A random sampling of this search space would still yield some insight into the potential of the model. Combinations of hyperparameters which showed particular promise, would then be further optimized in a local search. The models in this project were trained and evaluated on a small subset of their hyperparameter space and their performance metrics were stored and compared in order to find the better performing models.

4.3.3 Local Search

The local search holds all but one hyperparameter constant during optimization. This hyperparameter is changed slightly for a few iterations in order to give an indication of whether altering this hyperparameter had a positive impact on performance. After the nearby space of

the one hyperparameter has been sampled, the algorithm explore a different hyperparameter, restoring the previously explored parameter to its original value. Finally, when the nearby space of all hyperparameters were explored, if a new best hyperparameter combination is found, the process restarts with these hyperparameters. If not, the process terminates.

This local search optimization algorithm yielded positive results but had notable drawbacks. The algorithm makes minor adjustments to a particular hyperparameter, and holds every other hyperparameter constant. This means that if a combination of hyperparameters is near a minimum of the loss function, the algorithm will be useful to reach the bottom of this minimum. However, the algorithm will be stuck in this here, and will fail to make adjustments sufficient to reach a different, hypothetically better, minimum. In addition, the algorithm is oblivious to the performance impact of each hyperparameter, which hyperparameter optimization algorithms such as Bayesian Optimization is not. Bayesian optimization was deemed out of scope for this project.

Chapter 5

Results

The goal of machine learning models is to perform well on unseen data. As explained in the Machine Learning chapter, this is evaluated using a train-validation-test split approach. For this project, a pseudo-test set is also used. A portion of the data was removed from the rest, which has not been used until the final model. This true test set makes 7.5% of the entire dataset. The remaining 92.5% of the data is then further split into 80%-12%-8% (train-validation-pseudo test) segments. The pseudo test set is then available to evaluate generalization without introducing bias to the true test set.

Several models were created and experimented on throughout this project. From one layered networks to multi-layered networks to state of the art modular networks were attempted to achieve the highest performance possible. A broad selection of models were experimented upon, and were selected for primarily based on their precision on validation data. The models used evenly balanced data during training, as correctly identifying earthquakes is more important than mistaking an explosion for an earthquake.

The majority of the time spent developing this project has been done on a laptop without a GPU. In the last few months the models were trained on an NVIDIA RTX 3090. Access to a GPU had tremendous impact on training time, but its late arrival was a limiting factor to experimentation. The reported models in this section were all trained on the GPU.

In the pursuit of quickly eliminating poorly performing models, searching as many combinations of hyperparameters as possible without excessive training time, and reduce potential

overfitting of excessive training [28], Early Stopping has been used. Early Stopping monitors a user defined performance metric, and ends the training process early if the learning of the model stagnates. What constitutes a stagnant model is controlled by the user, and at termination the parameters of the model producing the best performance may be restored. The learning rate during the training process is also gradually decreased to help convergence to the local extreme. Similarly to Early Stopping, the user defines some patience interval after which the learning rate is reduced by a given fraction, and is referred to as Reduce Learning Rate On Plateau(RLROP). Both of these tools serve obvious benefits, but their use may prematurely terminate the training of models who could otherwise perform very well. In this project Early Stop was set to monitor the relevant performance metric, restore the best weights upon termination set to occur if no improvement is made in 7 epochs. RLROP is set to monitor the same metric as Early Stop, and reduce the learning rate by half if no improvement is made in 3 epochs, to a minimum learning rate of 5e-5.

The limited access to a GPU combined with the general time constraints of this thesis meant that the experiments and model optimization performed were restricted. The hyperparameter combinations attempted were a tiny fraction of the search space, the time spent training was short in terms of epochs and Early Stopping and RLROP were likely too impatient and aggressive. The potential for improvement by further experimentation on these models (and others) is clear, but this is hardly unique to this thesis.

5.1 Metrics

		Predicted	
		No	**Yes**
Actual	**No**	True Negative (NP)	False Positive (FP)
	Yes	False Negative (FN)	True Positive (TP)

Table 5.1: Confusion Matrix for binary classification.

A confusion matrix, such as the one seen in table 5.1, display the complete output of the model. In practice, the cells are replaced by a number representing the number of times a prediction of that type is made in place of the explanatory labels which make up the sample table above. That is, for example, where it says True Positive, would be a number

representing how many predictions were both predicted true and were actually true. This table is very useful for insight into how the model performs, and several other metrics can be derived from it.

5.1.1 Accuracy

Selecting the right performance metric for the task and distribution of the data is a key aspect of model selection. A widely used and simple to understand metric is accuracy:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{5.1}$$

In the confusion matrix, one can derive accuracy by the summation of the diagonal divided by the summation of all the cells in the table. This metric is useful when the classes in the dataset are evenly distributed, but can be misleading otherwise. For example, the validation set of the earthquake/explosion model consists of 92.38% explosions. If accuracy is used to evaluate this model, the model could exclusively predict explosion and report an accuracy of 92.38%. The metric appears to indicate a fairly decent performance, but the model is in reality completely useless.

5.1.2 Recall

As classifying all events in the earthquake/explosion dataset as explosions might lead to great accuracy, it is not a useful measurement of performance. Instead it is more indicative of performance to gauge how many of the earthquakes(for example) were actually predicted as earthquakes. This metric is called recall and is given by this formula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.2}$$

However, similarly to accuracy, the model could exclusively predict earthquakes (positive class) and report a recall of 1. Recall is, therefore, a step in the right direction, but is not by itself a good metric to optimize.

5.1.3 Precision

It is important that a model when classifying an event as an earthquake, is correct. Precision is a metric which measures the models ability to only identify the relevant points. Precision is the number of true positive predictions divided by the number of times the model predicted the positive class:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.3}$$

Intuitively, the precision of the earthquake/explosion model, where earthquake is the positive class, is the number of times the model correctly classified earthquakes divided by correctly classified earthquakes and explosions incorrectly classified as an earthquake. As explosions appear to be much more common than earthquakes, and a good model would be able to correctly predict earthquakes it makes sense to use precision as a metric of the model. As precision does not account for incorrectly labeling an earthquake as an explosion, but rather is a measurement of how good a model is at correctly classifying when predicting earthquakes, a combination of recall and precision is often good to maximize.

5.1.4 F1 score

F1 score is the harmonic mean of precision and recall.

$$\text{F1 score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

Using a harmonic mean is better than using their average as it punishes extreme values. For example, if the model exclusively predicts earthquakes, the average of recall and precision would be 0.5. In contrast, the F1 score would be $2 * \frac{0.0762 * 1.0}{0.0762 + 1.0} \approx 0.14$. The F1 metric allow for optimization where the balance of recall and precision is the goal.

5.1.5 Fβ score

In cases of unbalanced data where the cost of a False Negative is higher than a False Positive, it may be advantageous to assign more weight to the recall metric. While maximizing recall by itself may lead to only predicting the positive class, a weighted version of the F score may be used. The user defines some non-negative β in which to weight the F score. The higher the β above 1, the more Recall will impact the resulting F score. The smaller the β is below 1, the more weight precision has. F β is calculated using this formula:

$$\mathrm{F}\beta = \frac{(1+\beta^2)\mathrm{Precision} * \mathrm{Recall}}{\beta^2\mathrm{Precision} + \mathrm{Recall}} \tag{5.5}$$

When classifying earthquakes and explosions it is more important to correctly identify all the earthquakes than it is to mistake an explosion for an earthquake. In other words, recall is more important than precision (where earthquakes is the positive class). In situations such as this the Fβ score metric may be useful.

5.1.6 Precision-recall curve and Average Precision

The output of a binary classifier is a single value ranging from 0 to 1. The user then defines some cutoff threshold which determines which class the output value belongs to. This threshold is for binary classifiers commonly set to 0.5, but may be changed in situations where it is more important to correctly identify samples belonging to particular class, usually at the cost of precision. For example if a classifier is made to predict whether or not a patient has some terminal disease, it is more important to identify all the real cases, even though this means an increase in false positives. In order to visualize the tradeoff of different thresholds on precision and recall a precision-recall curve is commonly used. Further, this graph can be reduced to a single value by calculating the area under the precision recall curve (AUC-PR). The way used in this project to get this value is using Average Precision (AP) which summarizes the curve as a weighted mean of precision, P, achieved at each threshold,$t \in T$, weighted by the increase in recall, R, from the previous threshold, $t-1$:

$$\mathrm{AP} = \sum_{t}^{T} (R_t - R_{t-1}) P_t \tag{5.6}$$

The AP metric gives insight into the performance of the model with the consideration of several thresholds. This curve gives insight into the performance of a model, and can help select the best threshold for a specific problem. A perfect classifier will hug the upper right corner of the graph, and a random classifier will be near baseline. The baseline used in this project is the curve created if the model exclusively predicted the positive class with certainty (meaning integer output).

5.2 Models

Noise-not-Noise vs Earthquake-Explosion

The decision to initially discriminate between noise and not-noise(earthquakes and explosions) was made to allow for individual optimization of two models, rather than a single optimized model. For the noise-not-noise(3N) models, both the training and validation sets were distributed evenly. During model selection a 25% subsample of the dataset (excluding the true test set) was used splitted into training-validation-pseudo sets. The training set were evenly distributed with the not-noise class consisting of ∼51% earthquake events and ∼49% explosion events. The validation set consisted of evenly represented classes, but the not-noise class was made up by 7.19% earthquakes and 92.81% explosions. The Earthquake-Explosion(EE) models were trained on an evenly balanced training set, but not a balanced validation set. The validation set consisted of 7.62% earthquakes and 92.38% explosions.

Due to the evenly balanced 3N validation set, accuracy was chosen as the performance metric for optimization and selection. This makes the selection process simpler and is justified by the distribution, however, in a deployed version, it may be more beneficial to select for a model with the least amount of false negatives (not-noise being the positive class). In the EE problem, the explosion class is overly represented in the validation set. For this problem it is more important to correctly identify as many of the earthquakes as possible, so a high recall is important. As discussed in the metric section, optimizing for recall alone is problematic, so a trade-off between precision and recall is necessary, where recall is more important. Consulting NORSAR, a standard for a model to incorrectly classify 5 explosions as earthquakes per correctly labeled earthquake was deemed acceptable. Fβ was thus selected as the optimization metric, with a β of 2 where earthquake is the positive class. This

β selection is somewhat arbitrary, but strikes a reasonable balance between the two metrics, although with some drawbacks.

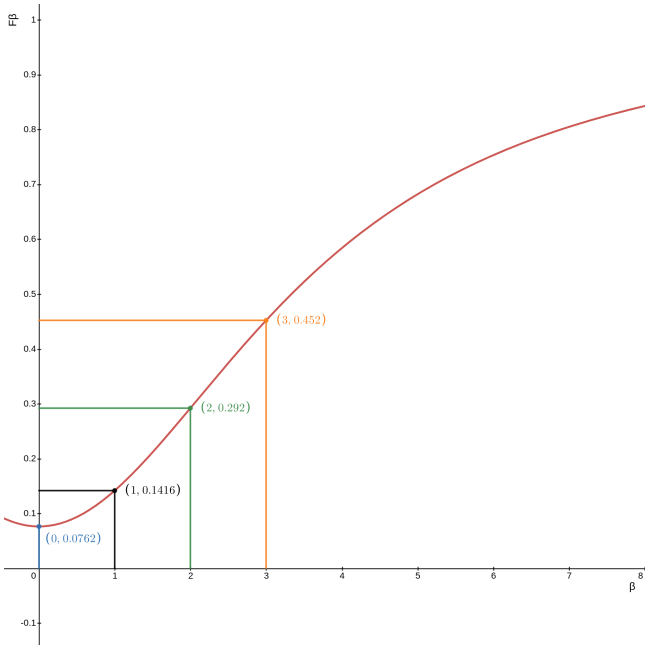


Figure 5.1: Graph showing the resulting Fβ as a function of β with the precision and recall from a classifier exclusively predicting earthquakes. The chosen β of 2 is labeled in blue. The horizontal axis shows the β value, while the vertical axis portrays the resulting Fβ score.

When performing grid search, the vast majority of the models perform poorly, and end up exclusively predicting a single class. Those which only predict the positive class will have a high recall, but a poor precision. If the beta is too high, the performance measure will deceptively indicate that these perform better much better than those who exclusively predict the negative class. In figure 5.1, the resulting F score for some given β is shown, portraying the possible values of Fβ when exclusively predicting earthquakes. Models which start off predicting only the positive class but over time start predicting the negative class, this will indicate a drop in performance as this is often correlated with a drop in recall. Admittedly, this is not exclusive to a high value of β, but a higher value worsens this. This affects techniques such as Early Stopping and RLROP, because their application depends on improvements in performance. With less constraints on available experimentation time, not using these could mitigate the drawbacks of a higher β, but not would not remove the deceptive element of the metric. An argument could be made that a β of 3 would be better for evaluating these models. As explosions outnumber earthquakes by a magnitude of 12, a

model with a high recall in explosions as well as earthquakes would still have a significant impact on the precision of earthquakes. As a result, it is to be expected that there will be a high number of false positives (where earthquakes is the positive class) relative to true positives. In performance terms: there will be big difference between recall and precision. A change in either metric will have a notable impact on the Fβ score. Experimentation during this project has shown that an increase in precision often comes at the cost of recall, and vice versa, and so a higher β will select for models which can improve in precision with less of a cost to recall.

All the models created and evaluated in this project uses the Tensorflow library (`tf-nightly-gpu version 2.5.0.dev20210126`). The architectures given by the tables in their respective section will only contain parameters and their values which differ from the default setting in this version of Tensorflow.

5.2.1 Fully Connected Neural Network (FCNN)

Multi Layer Perceptron, or FCNNs, are commonly used in classification tasks. These types of networks are made up of one of the simplest and traditional layers for Deep Learning models. These types of models treat elements as independent from each other. As a result the temporal information in the timesteps are lost, leading to poor performance where these relationships are significant. As a result, these were suspected to perform worse on EE than 3N. This model was included in this project as a sanity check of the importance of temporal relationships while classifying this type of data.

Baseline

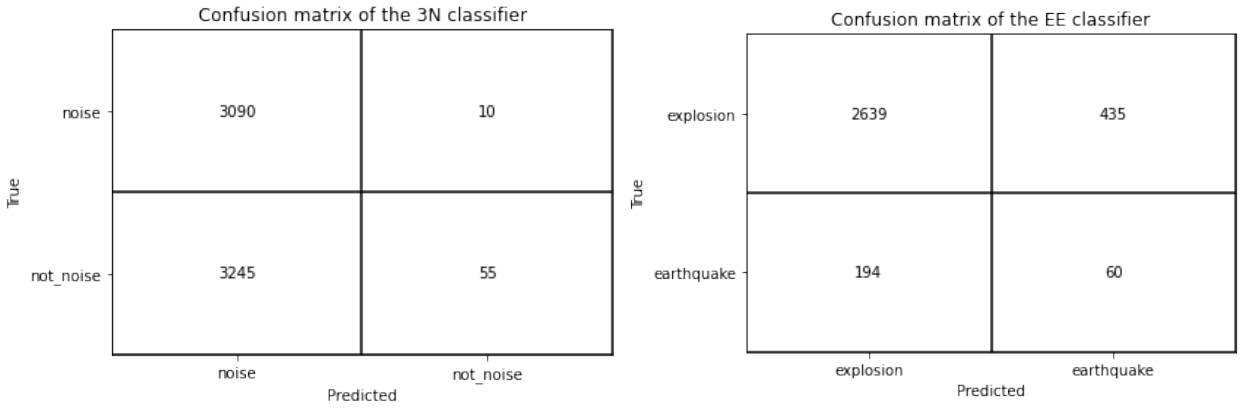
Baseline FCNN architecture was created in order to have points of comparison when evaluating the effects of different preprocessing as well as experimenting with more complex FCNNs. The baseline model for FCNN, decribed in table 5.2, is a very simple network with only a single hidden layer with 50 units and a ReLU activation. The Flattening layer in Layer 1 "flattens" the input into a single array, disregarding temporal information, but necessary for subsequent Dense layers. Time augmentation has been used to shorten the waveform snippets to 150 seconds. Standard scaling has also been used in order to enable to network

to learn. Using no scaling on this dataset causes immediate numerical stability issues, and so a scalar is necessary. Standard scaling was chosen because simply due to being very commonly used, and often recommended when scaling data without domain knowledge. During training the baseline model neither Early Stopping nor RLROP was used.

Optimizer: SGD
Loss: Binary cross-entropy
Learning rate: 0.01
Epochs: 50
Batch size: 128

Layer 1	**Layer 2**	**Output Layer**
Flattening layer	Dense Neurons: 50 Activation: ReLU	Dense Neurons: 1 Activation: Sigmoid

Table 5.2: Architecture of the FCNN baseline model.



(a) Confusion matrix of 3N baseline

(b) Confusion matrix of EE baseline

Figure 5.2: The two figures show the outputs of each baseline model. Each model use the same hyperparameters and preprocessing steps.

As seen in figure 5.2a the model almost exclusively predicts noise, leaving it with an validation accuracy of ∼49%. The training accuracy matches the validation, meaning the model is severely underfitting. This result is as expected functioning as a benchmark in which to compare more advanced models in while doing model selection. Figure 5.2b shows that the EE baseline typically predicts explosions, but not exclusively. The resulting performance shows a recall of ∼23%, and a precision of ∼12%, resulting in a F2 score of ∼0.2.

Model Selection

In an attempt to improve the performance of the model on both tasks, random grid searches were performed. The searches experimented with different combinations of hyperparameters:

- Batch-size: 64, 128, 256
- Learning rate: 0.1, 0.01, 0.001, 0.0001
- Type of optimizer: Adam, SGD, RMSprop
- Number of hidden layers, L: 1-5
- Number of neurons in each layer, defined by a sequence of integers $F = [f_1...f_L]$ and some integer n. The number of neurons in some layer $i \in L$ is given by $f_i * n$.
- L2 regularization rate: 0.3, 0.2, 0.1, 0.01, 0.001, 0.0001, 0
- L1 regularization rate: 0.3, 0.2, 0.1, 0.01, 0.001, 0.0001, 0
- Activation function used in the hidden layer(s): ReLU, TanH, Softmax, Sigmoid
- Whether or not to use batch normalization, dropout, or neither between each hidden layer.

It is important to note that the list above contains the potential hyperparameters that a model could be trained with during model selection, but does not mean that each of the hyperparameters were attempted. The generally poor performance of FCNN on this dataset also lead to less time invested in exploring the grid. The results from this experimentation is likely not representative of FCNN performance on this type of data.

Several of these searches were completed with different preprocessing steps used, particularly different scaling techniques. Again, the number of possible combinations of hyperparameters is very large, so the results only show a small fraction of the possible models. During training both Early Stopping and RLROP was also used, with fairly aggressive and impatient settings, necessitated by resource constraints. All models were trained for a maximum of 50 epochs, but the vast majority ended training prior to reaching this point due to Early Stopping. 50 epochs were selected to reduce the amount of time spent on each model, but could result in models being severely undertrained. Generally, on either classification problems, the models relatively quickly converged within 25 epochs. This does not mean that nothing could be gained by training them further, resulting in selecting for models which learn quickly.

Noise vs not-Noise

Optimizer: RMSprop
Loss: Binary cross-entropy
Learning rate: 0.001
Epochs: 50
Batch size: 64

Layer 1	Layer 2	Output Layer
Flattening layer	Dense Neurons: 200 Activation: ReLU	Dense Neurons: 1 Activation: Sigmoid

Table 5.3: Architecture of the best performing FCNN 3N model.

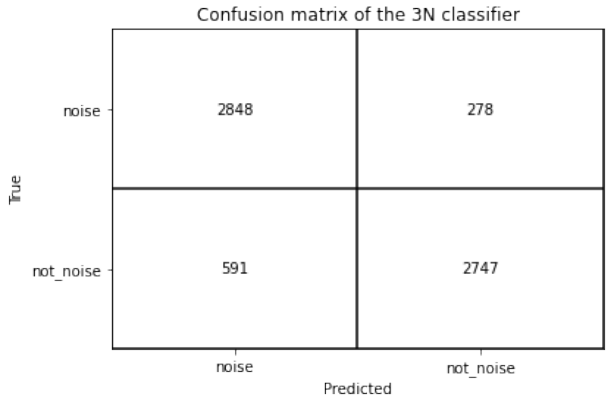
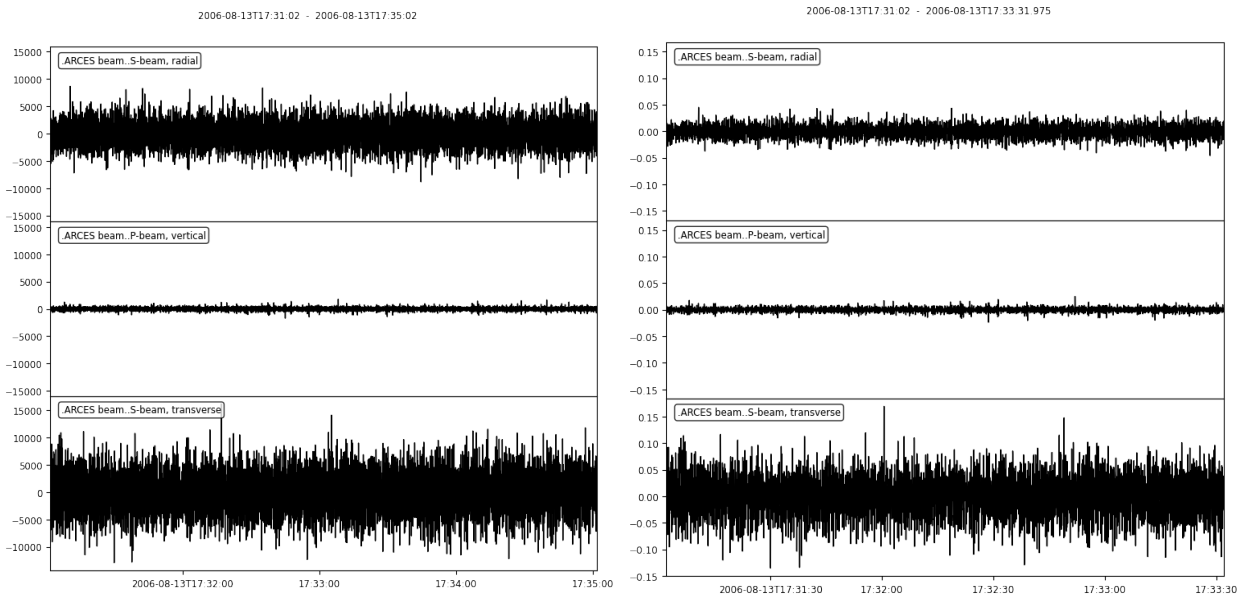


Figure 5.3: Confusion matrix showing the predictions made by best performing FCNN 3N model on the validation set.

The model described in table 5.3 was the best performing FCNN model on the 3N task. The model was trained with time augmentation, noise augmentation and standard scaling. The model also utilized RLROP with a patience of 3 epochs with a factor of 0.5, and Early Stopping with a patience of 7 epochs. The FCNN model had a validation accuracy of 86.6%, but despite its simple architecture struggled with overfitting, reporting a training accuracy of 96%. The model identified 82% of the non-noise events with a precision of 91%. It identified 91% of the noise events, with a precision of 83%. Of the few FCNNs able to learn, they were prone to overfit even with a simple network. Unsurprisingly, adding complexity through additional layers and or a high amount of neurons did not necessarily improve the networks' ability to learn. Regularization techniques such as dropout, batch L1 and L2 also appeared

to have detrimental impact on the model. The best performing model was found using grid search and was improved by several iterations of local search by a 3% gain in validation accuracy.



(a) Raw outlier noise waveform.

(b) Input transformed waveform.

Figure 5.4: The two figures show the raw and preprocessed outlier noise waveform incorrectly predicted to be not noise.

The model predicts with certainty most events with distinct differences from noise reasonably well. That is, events which can easily be visually identified as an event of interests. The event in 5.4 is an outlier noise event with significant continuous displacement in the stacked counts axis, falsely predicted to be a non-noise event with certainty. Other waveforms of similar features were also predicted to be non-noise events incorrectly. Exactly breaking down the reasoning used by a neural network during inference is a futile process. Regardless, some speculation into high level trends may provide insight into shortcomings of particular models. This network appears to have learned periods of strong displacement in stacked counts is an feature of non-noise events. At the same time, there is a significant portion of non-noise events with short periods of displacement in stacked counts incorrectly labeled to be noise events. The shorter these events are, the more certain the classifier is that the event is noise. Distinguishing the events based on this feature is effective to a point, but very limited. This hypothesis would also explain why FCNNs struggle with waveforms

transformed by the normalized scaler. This type of transformation is independent of other events, and so relative differences among waveforms are lost. Still, with the normalized scaled data, the network could learn that if there are portions of the waveform with very little displacement in the stacked count, this would be an indicator of a not-noise event. However, as channels differ in their average displacement for each waveform and the model initially flattens the data, learning such a pattern by a FCNN is unlikely.

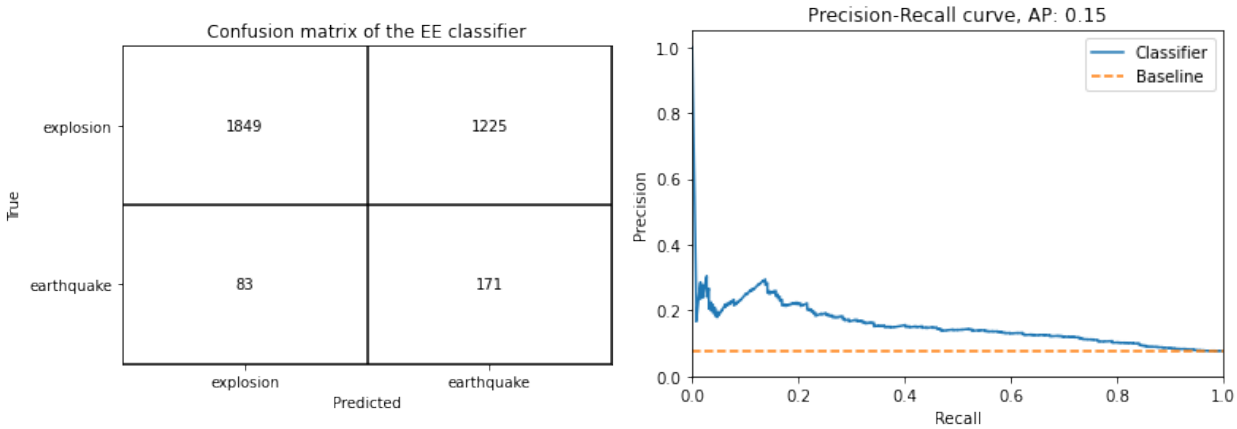
Earthquakes vs Explosions

This model, even when shallow in depth and width was very sensitive to the activation function in the hidden layers while distinguishing between earthquakes and explosions. Simply using a ReLU activation in the baseline architecture instead of Softmax gave immediate improvements in performance (precision = 0.16, recall = 0.23, F2 = 0.21), while using standard scaling. Improvements beyond this proved to be challenging, indicating the hypothesis made in the start of this section to be true. After several grid searches with different scalers and digital filters, a few models reported F2 scores higher than exclusively predicting earthquakes (F2 = 0.292).

Optimizer: SGD
Loss: Binary cross-entropy
Learning rate: 0.01
Epochs: 50
Batch size: 256

Layer 1	**Layer 2**	**Layer 3**
Flattening layer	Dense Neurons: 320 Activation: ReLU K/A regularizer: L2 = 0.001	Batch Normalization
Layer 4	**Layer 5**	**Layer 6**
Dense Neurons: 640 Activation: ReLU K/A regularizer: L2 = 0.001	Batch Normalization	Dense Neurons: 1 Activation: Sigmoid

Table 5.4: Architecture of the best performing FCNN EE model. K/A is short for Kernel and Activity.



(a) Confusion matrix

(b) PRC and AP

Figure 5.5: The two figures display the performance of the final FCNN EE model. 5.5a shows the predictions made by the model. 5.5b shows the PRC curve of the classifier.

The best FCNN model on the EE dataset is shown in table 5.4, and its performance on validation in figure 5.5. As with the 3N FCNN model, this was trained with standard scaled data, noise augmentation, time augmentaion, RLROP and Early stopping, with the same patience settings as 3N. Expectantly, this model did not perform well (5.5), with an F2 of 0.354. A default threshold of 0.5 has been used for predictions by all models. It identifies 67% of the earthquake events with a precision of 12%. It identifies 60% of the explosion events with a precision of 96%. If a model exclusively predicted earthquakes it would report an F2 score of 0.292, meaning this result is only a minor improvement from random. The increased complexity in this model compared to 3N, contributes mostly to its ability to overfit with a training accuracy of 88% on less distinguishable classes. As expected FCNN performs better on 3N than on EE, and the results appear to confirm the importance of capturing the temporal dimension. Looking at figure 5.5b, the model performs close to what would be expected from a classifier predicting randomly. With the poor F2 performance combined with the PRC indicating close to random predictions, further investigation into the type of prediction errors appears fruitless. Unlike the 3N problem, these waveforms do not have differentiable differences by their average expression in the stacked count axis. It is reasonable to speculate that the correct predictions made on the validation set have counterparts of very similar expression in the training set. More general distinguishing features are not learned, and so the model struggles on unseen data.

5.2.2 LSTM

LSTMs have shown promising results in time series classification tasks [19, 21]. Although generally used for comparatively shorter snippets, LSTMs "... learns to extract features from sequences of observations and how to map the internal features to different activity types. The benefit of using LSTMs for sequence classification is that they can learn from the time series directly, and in turn do not require domain expertise to manually engineer input features." [4]. A downside of LSTMs is that they are notoriously slow to train, and their performance is worse on long sequences [1]. Due to promising results in other domains and performance independent from domain expertise, LSTMs were experimented with in this project. The waveforms from NORSAR, even when reduced in duration by augmentation, are very long (6000+ timesteps) which is problematic with "pure" LSTM networks. Consequently, the experimentation with self-developed LSTM architectures was limited in comparison to models such as CNN. Rather, proven State-Of-The-Art models such as MLSTM-FCN were considered a more viable candidate.

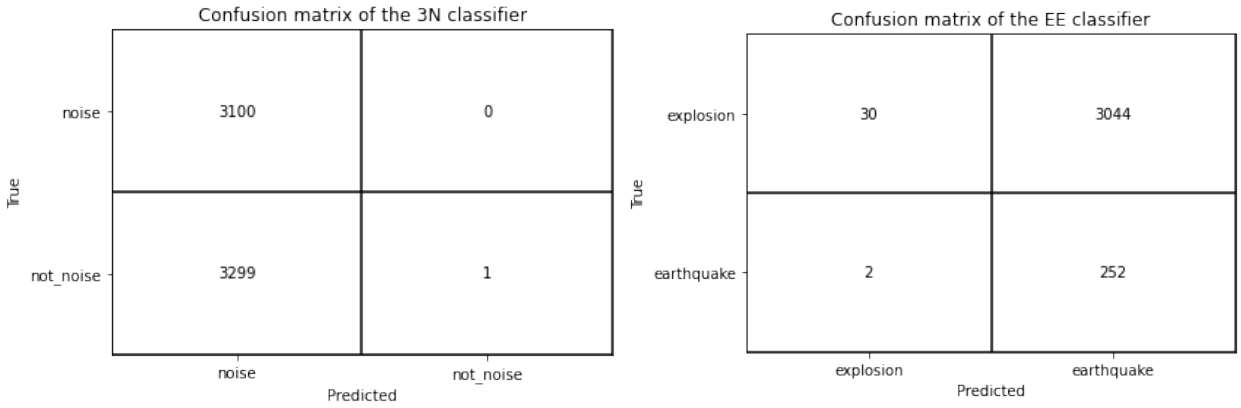
Baseline

A very simple LSTM model was created in order to create a point of comparison to the effect of preprocessing as well as experimentation with the hyperparameters and different architectures of LSTMs. This baseline model consists of an input layer, a single LSTM TanH activated layer with 1 unit and a sigmoid activated classification layer.

Optimizer: SGD
Loss: Binary cross-entropy
Learning rate: 0.01
Epochs: 50
Batch size: 128

Layer 1	**Output Layer**
LSTM	Dense
Units: 1	Neurons: 1
Activation: TanH	Activation: Sigmoid

Table 5.5: Architecture of the LSTM baseline model.



(a) Confusion matrix of 3N baseline

(b) Confusion matrix of EE baseline

Figure 5.6: The two figures show the outputs of each baseline model. Each model use the same hyperparameters and preprocessing steps.

As expected this bare bones architecture fails to fit to the data, and proved to be trend with pure LSTM models on this dataset. Experimentation on variations of this type of model yielded few models performing better than completely random. The only model which stood out from the rest was a 3N model achieving validation accuracy of 67%. As LSTMs struggle with long sequences, pure LSTM models are not suitable for this type of data. The results from experimentation with this model are uninteresting, and do not provide valuable insight beyond its incompatibility worth reporting in this project.

Noise vs not-noise

MLSTM-FCN LSTM-FCN is an architecture developed by Karim et al. and has performed well on the University of California Riverside (UCR) Benchmark datasets [5, 19], as well as on real life data classifying animal sounds [21]. These results were on univariate time series, but the architecture was further developed to handle multivariate time series, with a similar architecture with a new name: MLSTM-FCN (Multivariate LSTM-FCN) [20]. In the LSTM-FCN paper by Karim et al. they propose a model which utilizes attention [19] in the first layer to counteract the effects of longer sequences called ALSTM-FCN. An architecture was also developed to handle multivariate data, MALSTM-FCN. Attention could potentially improve the performance of this model, and likely other architectures, on the ARCES dataset, but is left for future research. Evaluating both the MLSTM-FCN and MALSTM-FCN was

deemed out of scope, as these are very expensive to train. The MLSTM-FCN architecture was chosen over the attention version due to its shorter training time, while producing similar performance. The significant training time of this model limited the architectural search space evaluated in this project to focus primarily on the number of units in the LSTM layer. Different types of scalers, filters and noise augmentation parameters were still included in the search.

The MLSTM-FCN architecture, as shown in figure 5.7, consists of two main design features. The first feature is the use of a *squeeze-and-excite* block, proposed by Hu et al. [18]. This block is designed to "... improve the representational power of a network by enabling it to perform dynamic channel-wise feature recalibration" [18], which is considered by Karim et al. as a form of learned self-attention on the output feature maps of prior layers. Exploration of attention is out of scope for this project, but papers such as *Attention Is All You Need*[34] explore this topic in depth. In short, attention enhances the important parts of the input data and fades out the rest, focusing the computational resources on the influencial elements of the input. The *squeeze-and-excite* block in the MLSTM-FCN model allows the network to use self-attention to incorporate inter-correlations between the variables at each timestep [20].

The second feature of one LSTM block and one FCN block is shared with its univariate version. The LSTM block in the multivariate model starts with a dimension shuffle, transposing the input waveform. The waveform is converted from the original waveform of 6000 timesteps, each with 3 features, into a waveform of 3 timesteps, each with 6000 features. This dimensionality shuffle provides the downstream LSTM layer with the global temporal information of each channel at once. The process reduces training and inference time, without losing accuracy [20]. The dimensionality shuffle is followed by a masking layer, with default mask value. This masks the input waveform such that timesteps which contain the mask value are skipped in the downstream layers. The default mask value of 0 means that any timestep containing all zeros are skipped in the subsequent layers. A masking layer is typically used to handle sequences of varying length and so this layer does not have any effect on this particular dataset. The block is completed by an LSTM layer and a dropout layer with a rate of 0.8 by default.

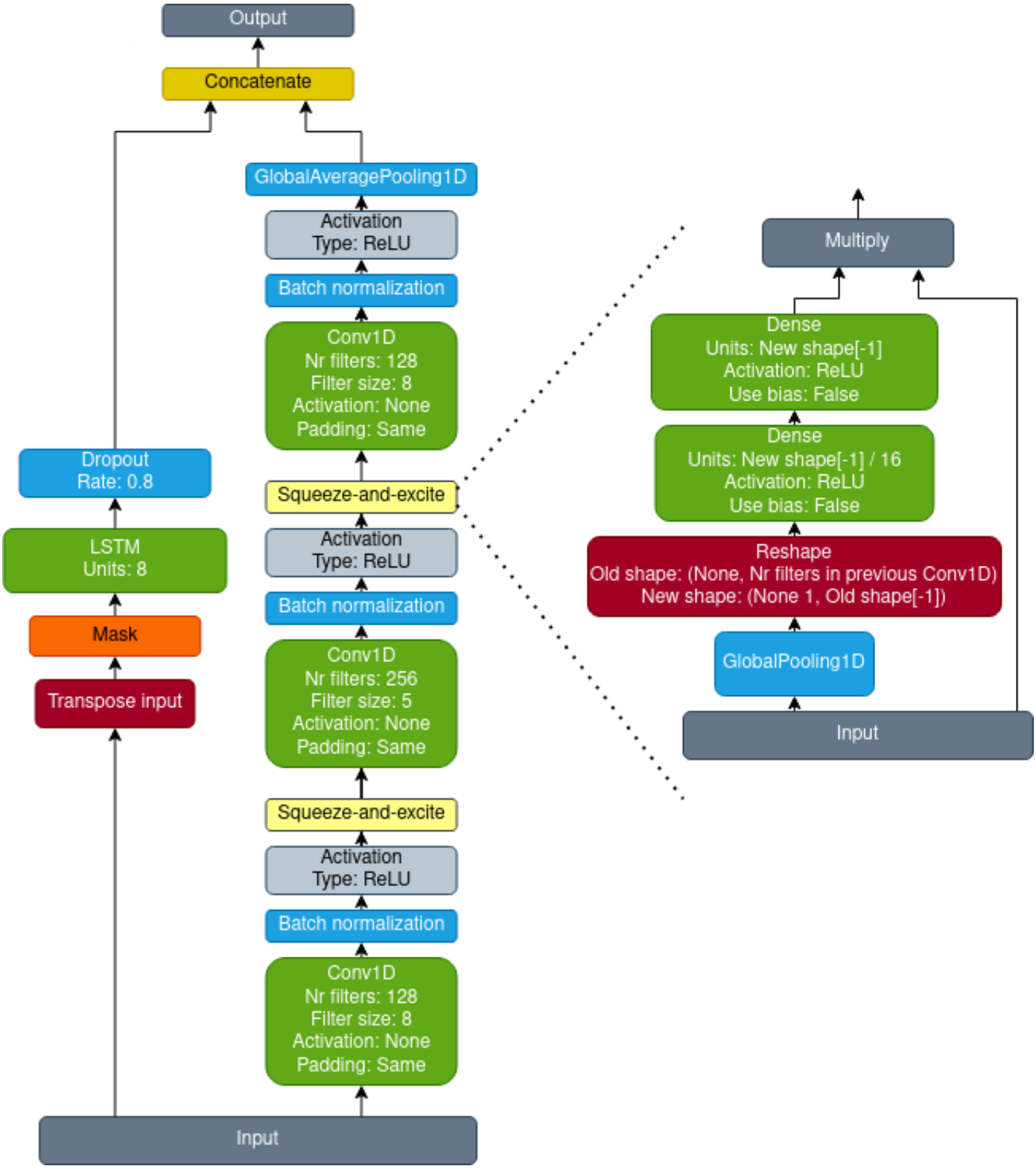


Figure 5.7: Figure showing the architecture of the default MLSTM-FCN published on Majumdar's github: `https://github.com/titu1994/MLSTM-FCN`. *Note that the number of units in their LSTM layer differs, depending on the application.*

The FCN block consists of three stacked convolutional layers, separated by batch normalization prior to activation for each convolutional layer. In between the activation of the first and second activation layers, are the squeeze-and-excite blocks. The stacked convolutional layers are identical to the convolution block in the CNN architecture by Wang et al., and "... has shown compelling quality and efficiency for semantic segmentation on images [25]"[35]. The FCN block is completed by a global average pooling layer before being concatenated with the output of the LSTM block. For the purposes of this project, the output layer is a Dense layer of one unit and a sigmoid activation function.

A limited grid search was applied to this model, evaluating different scalers, units in the LSTM layer, type of optimizers, learning rates, and batch sizes. The best performing model has an identical architecture to figure 5.7, using a batch size of 128, the Adam optimizer and a learning rate of 0.001. Normalize scaler yielded the best results, together with time augmentation, noise augmentation, early stopping and RLROP.

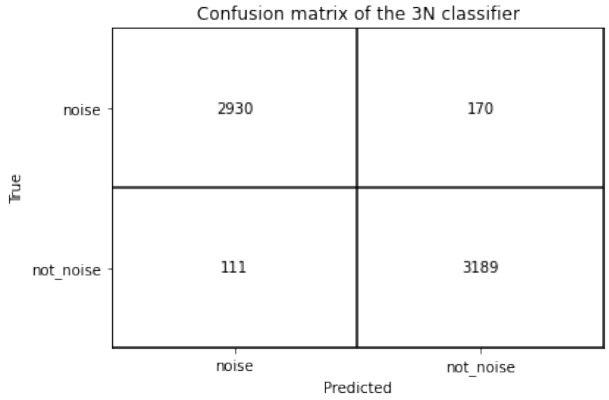


Figure 5.8: Confusion matrix showing the predictions made by the MLSTM-FCN classifier.

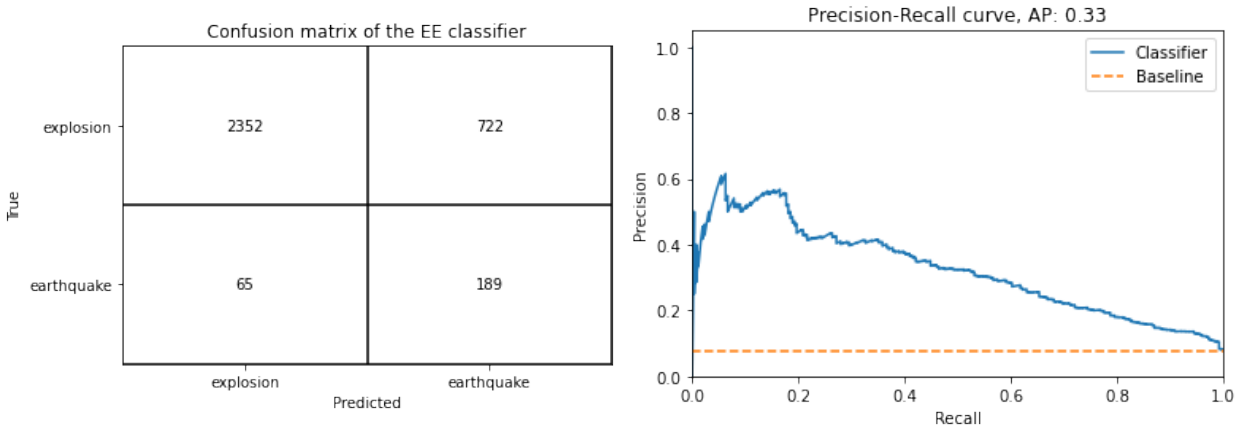
The model reports an accuracy on validation of 95.6%. It identifies 96.6 % of the not-noise events correctly, with a precision of 94.9 %. It identifies 94.5% of the noise events correctly with a precision of 96.3%. A selection of prediction errors made by this model can be found in the appendix A.1. Like many of the other models in this project, this model too struggles with incorrectly labeled events and mislabeled start times. This model does appear to have greater difficulty distinguishing noise-like not-noise events, than many of the CNN models. Given the performance of the pure LSTM models, it would be interesting to see how much each of the different blocks contribute to the final output of the classifier. This was deemed out of scope, but could yield interesting insight into the decision process of this model. Due

to the dimensionality shuffle in the LSTM block, it would be reasonable to speculate that this model assigns relative importance on particular channels for prediction. In theory, the model could learn that, for example, the radial channel is less important than the other channels when classifying noise vs not-noise. Investigating the LSTM block combined with domain expertise of the waveforms could provide insight into why this particular model has a greater difficulty than pure CNN models on less clear events.

Earthquakes vs Explosions

MLSTM-FCN Another version of the MLSTM-FCN model was experimented with, optimized to distinguish between explosion and earthquake events. This model had an identical architecture to the one seen in figure5.7 except for 64 units in the LSTM layer, rather than 8. The model was trained with an RMSprop optimizer with a learning rate of 1e-4, with a batch size of 256 for 50 epochs. Early stopping, RLROP, noise augmentation, time augmentation and normalize scaler was also used. The grid search of the MLSTM-FCN architecture for EE was also limited, due to general poor performance on this dataset. The LSTM block is speculated to contribute little to the output, and the CNN block has very small filter sizes. Experimentation on the CNN architectures indicate that a larger filter size is needed to adequately capture relevant features of for classification. The speculation of little contribution of the LSTM block combined with the belief of larger filter sizes required, lead to more experimentation on CNNs than on this type of architecture. Still, the grid search lead to the selection of a best performer, whose results are illustrated in figure 5.9.

As seen in figure 5.9b, the model performs much better than random, but still does not produce an F2 score higher than 0.49. The classifier identifies 74.4% of the earthquake events with a precision of 20.7%. It performs better on explosions, identifying 77% with a precision of 97%. Selected predictions can be found in the appendix A.2. This model has a lot of uncertainty in its correct classifications as well as its errors. The model underfit in both validation and training, with a training accuracy of 80%. Gaining insight into the errors of this model is particularly difficult due to this, but in general the model struggles much more with events that other models classify with ease. Unlike the better performing models in this project, the errors of this model is populated by events with high signal to noise ratio. Even clear, relatively strong events occurring within a short distance are a challenge to classify for this model. Still, with further experimentation into the individual blocks of this model, the



(a) Confusion matrix

(b) PRC

Figure 5.9: The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve.

author believes this model may yield significantly better results with a more comprehensive hyperparameter search.

5.2.3 CNN

The selection of which models to experiment with was based on previous work with multivariate time series. CNN models have already been applied to similar data by works such as Meier et al. and Tibi et al. as discussed in the introduction. State-of-the-art architectures such as InceptionTime [8], reports highly competitive performance on benchmark univariate time series datasets. Although InceptionTime is made for univariate data, it is still possible to apply to this dataset with good results, albeit with drawbacks relating to training time. CNN architectures have shown to work well with data where spatial relationships are relevant to the analysis of the data as seen by their application to image data [14]. The success of this type of model on data where spatial relationships are important combined with its success on similar makes this type of model a highly viable candidate.

CNN Baseline

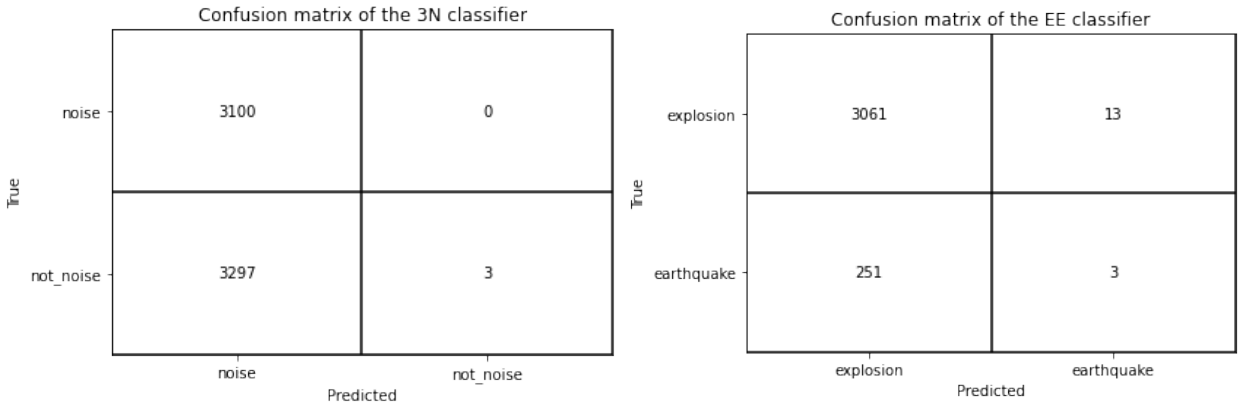
In order to be able to compare the performance of the preprocessing techniques as well as the usefulness of more complicated models, a very simple model was trained used as a baseline.

The input for the baseline on either problem is standard scaled and utilizes standard scaling. Neither Early Stopping nor RLROP is used for this model.

Optimizer: SGD
Loss: Binary cross-entropy
Learning rate: 0.01
Epochs: 50
Batch size: 128

Layer 1	**Layer 2**	**Layer 3**
Conv1D Nr filters: 16 Filter size: 8 Padding: Same Activation: ReLU	GlobalAveragePooling1D	Dense Neurons: 1 Activation: Sigmoid

Table 5.6: Architecture of the CNN baseline model.



(a) Confusion matrix of 3N baseline

(b) Confusion matrix of EE baseline

Figure 5.10: The two figures show the outputs of each baseline model. Each model use the same hyperparameters and preprocessing steps.

Model Selection

This project spent considerable time experimenting with various self-developed CNN architectures as well as state-of-the-art models and models used on similar data. The convolution aspect of the network worked very well capturing what is interpreted as the unique signature of events. Stacked convolutional layers initially capture low level features and layer-wise

translate this to higher level features. Initially, this was mistakenly believed to mean that the stacked convolution layers should decrease in the number of filters for each layer, but the performance of the models improved drastically with the inverse of this relationship. The subsequent layer(s) to the convolutional block, was also experimented with. Models such as InceptionTime, MLSTM-FCN and MALSTM-FCN [8][20] use a Global Average Pooling layer prior to the output layer instead of a fully connected layer in order to reduce the parameters of the models. This drastically reduces training time, but the simpler architectures created for this project did not benefit from this. The models which performed the best on this dataset, was inspired by the CNN used by Meier et al. A single or several stacked convolutional layers, with increasing number of filters in each, followed by two fully connected layers prior to the output layer. The grid used to search the different architectures differ among each model, but a general outline of hyperparameters tried is listed below:

- Batch size
- Optimizer
- Learning rate
- Activation in the convolutional layer(s)
- Number of convolutional layers
- Number of filters in each convolutional layer: given by a sequence of numbers multiplied by the number of filters used in the first layer
- Filter size
- L1 and L2 regularization rates.
- Activation in the fully connected layers
- Units in each dense layer.
- Whether to use dropout, batch normalization or neither between each convolutional layer. Batch normalization would be placed prior to the activation.
- Dropout rate, if appropriate.

This list captures the hyperparameters experimented with for the self developed models. The CNN models frequently performed well with randomly selected hyperparameters, and as a consequence were experimented with more than the other models in this project. Depending on the hyperparameters used, the training time differed significantly, and some optimization were limited by this as well as models exceeding 24 GB which was the capacity

of the GPU. Results from optimization of the EE models indicated that the network size should be increased, but the excessive training time, network size and time constraints prevented this. Note that hyperparameters related to preprocessing are not included in this list.

Noise vs not-noise

Meier et al.'s model The model used for signal/noise discrimination in the paper by Meier et al. was evaluated for use in this project. Their model performed very well distinguishing noise/signal short waveform snippets of 4 seconds, which is significantly shorter than the 150 second snippets used in this project. As seismic stations record a variety of nuisance signals, some of which may initially look very similar to earthquake signals, earthquake early warning algorithms sometimes send out false alerts. Their model excels at classifying short 4 second snippets of these waveforms. This differs from the goals and shape of the data for this project, but distinguishing between noise and not noise is a part of this project as well. Their CNN model architecture, described in the table below, is what has been evaluated in this project. However, their exact architecture is not discussed in full detail in their article. "After each convolution layer, the filter outputs are downsampled and activated" [7] does not describe how the output is downsampled. Both MaxPooling and Average Pooling between the convolution layers have been explored. How each convolution layer is padded is also not described. A several grid searches were completed trying out different combinations of pooling and batch normalization, with different preprocessing techniques. The architecture which produced the best results on 3N is described in table 5.7.

As their data and the data used in this project is of such significant different duration, the result of this model on this dataset is not comparable to their own. The duration of the waveforms, the inferred architecture and the necessary preprocessing steps produces a different model than the original. The evaluated model is inspired by, but not the same as the one by Meier et al. The final model included both average pooling as well as batch normalization prior to the activation of the feature map. The pooling layer does indeed reduce the dimensionality of the output, but still the model has 7.8 million trainable parameters. This particular model is trained with normalized data, but produced nearly as good results with the robust scaler. Using a filter such as in their paper had a negative impact on this

Optimizer: Adam
Loss: Categorical cross-entropy
Learning rate: 1e-3
Batch size: 48
Epochs: 40

Layer 1	**Layer 2**	**Layer 3**	**Layer 4**
Conv1D Nr filters: 32 Filter size: 16 Padding: same	AveragePool1D	Batch Normalization	Activation Type: ReLU
Layer 5	**Layer 6**	**Layer 7**	**Layer 8**
Conv1D Nr filters: 64 Filter size: 16 Padding: Same	AveragePool1D	Batch Normalization	Activation Type: ReLU
Layer 9	**Layer 10**	**Layer 11**	**Layer 12**
Conv1D Nr filters: 128 Filter size: 16 Padding: Same	AveragePool1D	Batch Normalization	Activation Type: ReLU
Layer 13	**Layer 14**	**Layer 15**	**Layer 16**
Flatten	Dense Units: 80 Activation: ReLU	Dense Units: 80 Activation: ReLU	Dense Units: 2 Activation: Softmax

Table 5.7: Architecture of the best performing CNN 3N inferred from Meier et al.

dataset, and so the final model used unfiltered data. Time augmentation, noise augmentation, early stopping and RLROP were used during training. The grid search results using normalized scaler showed that any combination of the hyperparameters performed similarly. As neural networks are sensitive to the initialization of the parameters, the selected best model is not dissimilar enough in performance to rule out any of the other combinations as viable candidates.

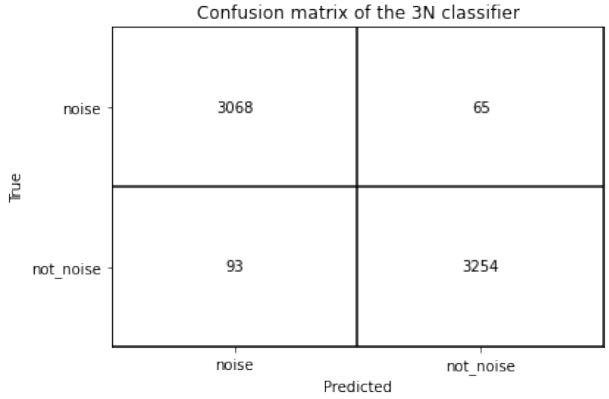


Figure 5.11: Confusion matrix showing the predictions made by the best performing Meier inspired 3N model on validation.

The confusion matrix in 5.11 shows the predictions made by the best performing model. The model performs surprisingly good, classifying 97.6% of the events correctly. It correctly identifies 97.2% of the non-noise events with a precision of 98%. In terms of noise events it correctly identifies 97.9%, with a precision of 97.1%. Looking at the errors of the model, this model, like the others, struggle with outlier events, especially those related to hardware errors. Events with mislabeled start times or teleseismic events where the secondary arrives very late in the waveform, are problematic due to the time augmentation. The waveforms affected by these conditions are difficult for the model to label correctly. A few of the false positive errors are caused by the waveform starting during the end of an ongoing event, where the event is labeled noise. A selection of prediction errors can be found in the appendix A.3. Speculation into the errors of the model, is that it struggles with events where there is a short but significant displacement in stacked counts. These errors are highly represented in the false positives and false negatives. This could be prominent in this model due to the short filter size, where not enough context is contained in the feature maps.

Best Performing Self-Developed Model The best performing CNN model on the 3N dataset performed surprisingly well, despite the long waveforms in its dataset. This model used the sequence of convolutional layers with increasing number of filters inspired by Meier et al. The number of filters as well as the filter size is large compared to most State-of-the-Art models, although these are typically not developed for such long waveforms. The model uses two fully connected hidden layers after the convolutional layers, which introduces a lot of parameters to the model. The model has ∼57 M trainable parameters, 54.4 M in the first fully connected layer. This model is inspired by the model inferred from the article by Meier et al., but the large difference in width, filter size and intermediate layers is significant enough to warrant independent evaluation.

Optimizer: Adam
Loss: Binary cross-entropy
Learning rate: 1e-4
Batch size: 256
Epochs: 50

Layer 1	**Layer 2**	**Layer 3**	**Layer 4**
Conv1D Activation: ReLU Nr filters: 72 Filter size: 56 Padding: same L1 reg: 0.0001 L2 reg: 0.01	MaxPool1D	Conv1D Activation: ReLU Nr filters: 144 Filter size: 56 Padding: same L1 reg: 0.0001 L2 reg: 0.01	MaxPool1D
Layer 5	**Layer 6**	**Layer 7**	**Layer 8**
Conv1D Activation: ReLU Nr filters: 288 Filter size: 56 Padding: same L1 reg: 0.0001 L2 reg: 0.01	MaxPool1D	Flatten	Dense Activation: ReLU Units: 252
	Layer 9	**Layer 10**	
	Dense Activation: ReLU Units: 214	Dense Activation: Sigmoid Units: 1	

Table 5.8: Architecture of the self-developed CNN 3N model

The model is trained with normalized scaled data and is using time augmentor as well as

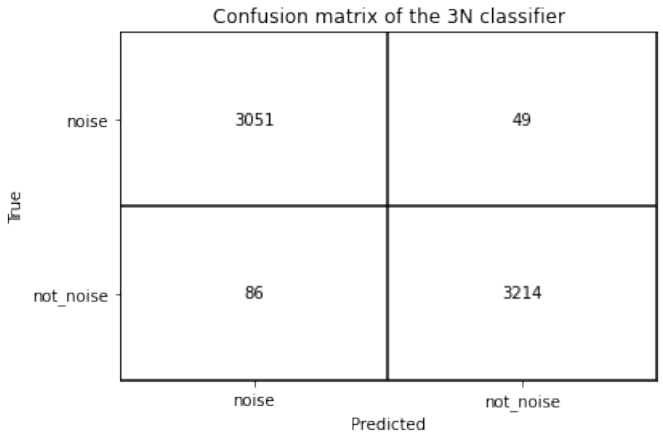


Figure 5.12: Confusion matrix showing the predictions made by the best performing self-developed 3N model on validation.

noise augmentation. During training Early stopping and RLROP is used, although the model does not converge by the end of the 50 epochs. Some of the mistakes the model makes could be a mitigated by further training. The model performs very well, with a validation accuracy of 97.8%. The model performs equally well on both classes, reporting precision and recall of ∼98% for both classes. A selection of false positives and false positives can be found in the appendix A.5. The reader should note that the noise augmentation is applied randomly by the generator, and does affect how an individual waveform appears to the model every time it is seen. This also applies to predictions. Looking at the false positive predictions made by the classifier, these are largely made up of outlier events, likely mislabeled events, and noise events with segments which could be mistaken for an event of interest. The term outlier is used here as events with moments of very strong displacement, likely as a result of hardware error, of a very rare event. It should be noted that the false positives, which arent mislabeled data, the model makes this prediction with uncertainty. In other words, a different threshold would filter out these false positives. The scaler used transforms the outlier noise events of continuous strong displacement, so that they appear as any other noise event. Still, a few mistakes are made on these events.

InceptionTime The current best state-of-the-art model for timeseries classification is called InceptionTime, by Fawaz et al. [8]. This model outperforms its predecessor HIVE-COTE [24] on the UCR benchmark datasets [5], in both results and speed. This architecture was inspired by recent success of Inception-based networks in computer vision tasks. The

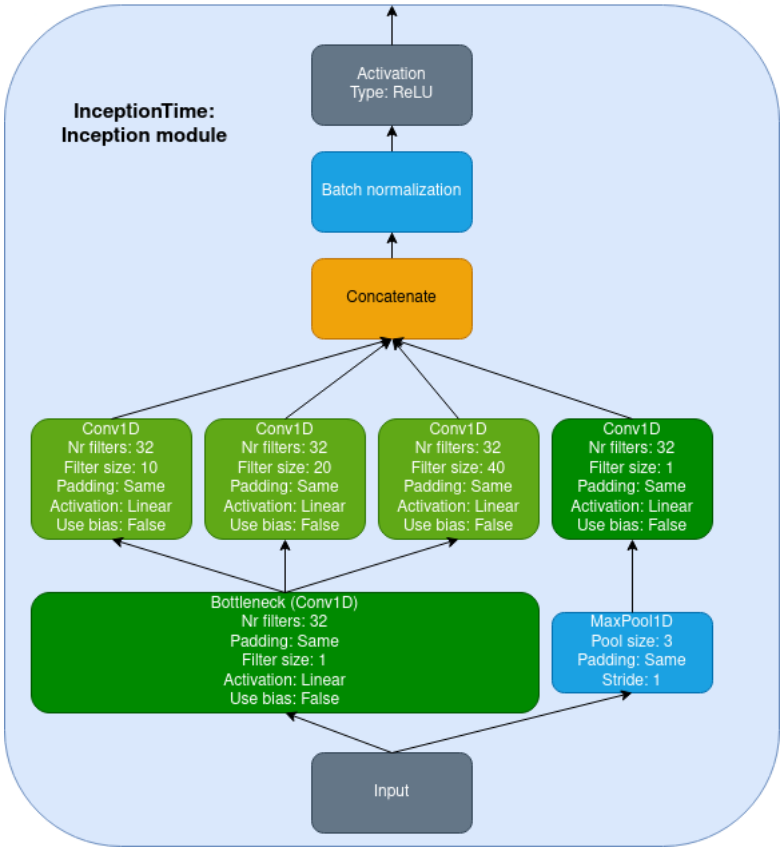


Figure 5.13: Illustration of the Inception module used in the InceptionTime architecture, with default hyperparameters.

model consists of Inception modules, which allow the network to use multiple convolution and pooling layers of different parameters in parallel. The network can then learn which layer (or combination) is most valuable to the output, without the engineer needing to deal with the trade-off of the different parameters and layer types. "By stacking multiple Inception modules and training the weights (filters' values) via backpropegation, the network is able to extract latent hierarchical features of multiple resolutions thanks to the use of filters with various lengths." [8]. An obstacle to this approach is the increased number of learn-able parameters in the network. To deal with this, it is common to use a bottleneck layer at the start of the Inception module in order to reduce the dimensionality of the input, the complexity of the model and reduces its chance of overfitting on small datasets. An illustration of the Inception module used in Inception time can be seen in figure 5.13. The use of Inception modules in neural network architectures is growing, initially proposed by Szegedy et al. in their GoogLeNet architecture [32].

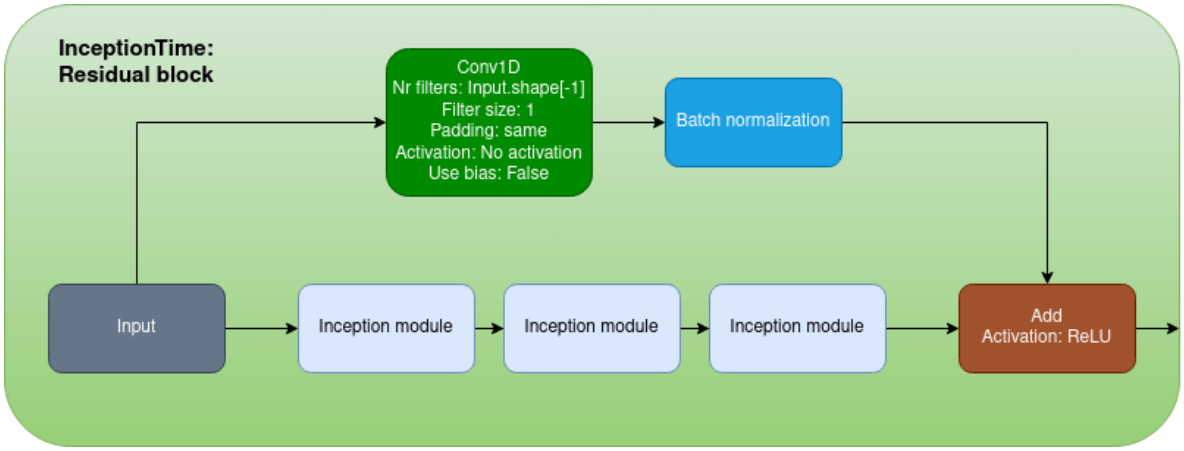


Figure 5.14: Illustration of the residual connection in the residual block used in the InceptionTime architecture.

As deeper networks can learn more complex features more easily, it has become a trend to keep adding layers to neural networks to make them better. The issue with this approach is that the input and the output is too spaced away and the network may suffer from vanishing gradient problem. The network will be unable to learn important features at the deeper layers. The use of stacked Inception modules increases the depth of the network of thus the risk of vanishing gradients as well as the Degradation problem. The degradation problem is a phenomena which occurs when adding more layers to a suitably deep model leads to higher training error [17]. In order to combat this, Fawaz et al. uses residual connections every third Inception module to maintain access to the original input of the model.

This residual connection can be seen in figure 5.14, and is an architectural design popularized by the ResNet architecture [17]. The combination of Inception modules, residual connections and larger filter sizes than previously proposed TSC models, aided InceptionTime to reach state-of-the-art performance. However, the NORSAR waveforms contain considerably more timesteps, than the waveforms in the UCR dataset. The longest waveform in the UCR dataset is 1882 timesteps, compared to the 6000 timesteps of the NORSAR waveforms. The InceptionTime reports significantly lower training times than its predecessor, but the training time still became an issue optimizing this model for the project as it depends on the shape of the input. Increasing the depth of the model by adding Inception modules had significant impact on the training time as well, so depth was a restrictive factor during optimization.

Random grid search was used for this model, evaluating different combinations of:

- Number of Inception modules
- Whether or not to use bottleneck layers
- Whether or not to use residual connections
- Number and size of filters in the inception modules.
- L1 and L2 regularization inside and or outside Inception modules.
- Inception module output activation type.
- Batch size, optimizer and learning rate.

Performing a search of such a large space lead to widely differing performance between each model, and this search space was over time reduced to values near the default InceptionTime hyperparameters. Due to the ambitious search space and the long training times, the results of this project does not reflect a representative evaluation of this model on this dataset in a broad sense. Yet, the model performs quite well, despite the limited search.

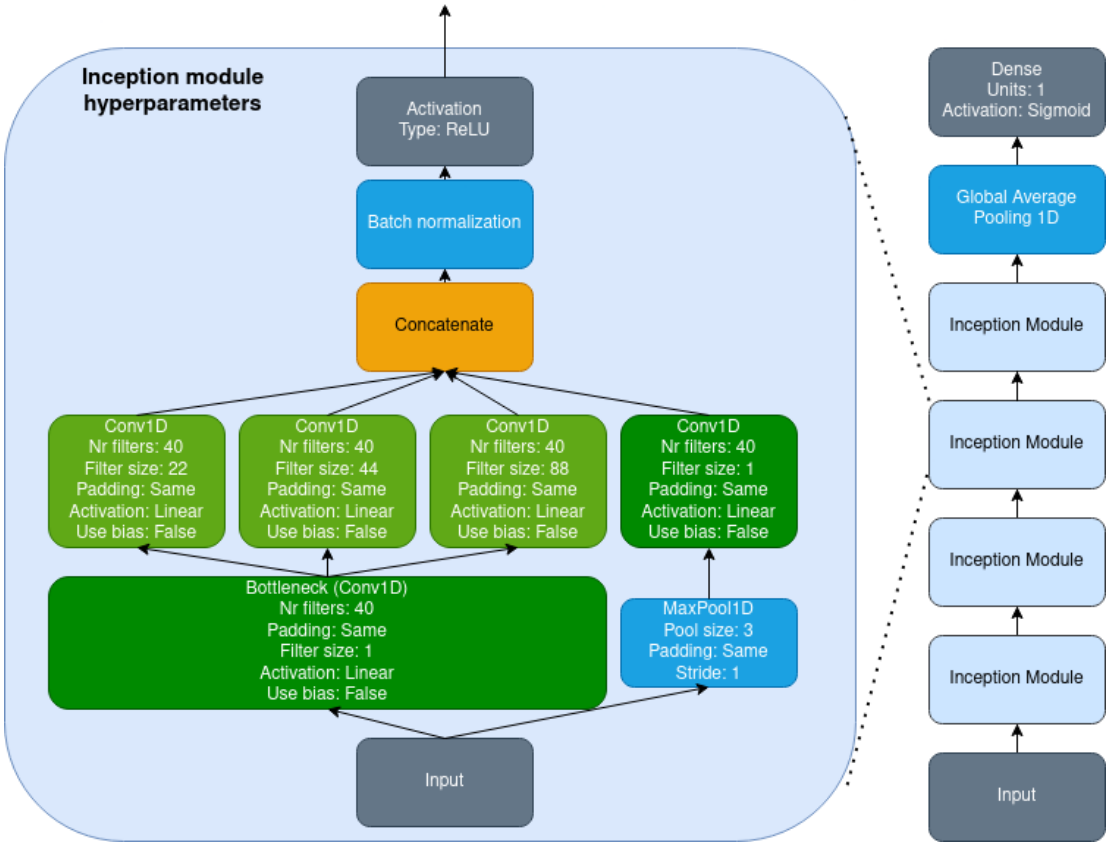


Figure 5.15: Illustration of the best performing InceptionTime variation on the 3N task. Note that this model does not use residual connections, and that each Inception module is identical.

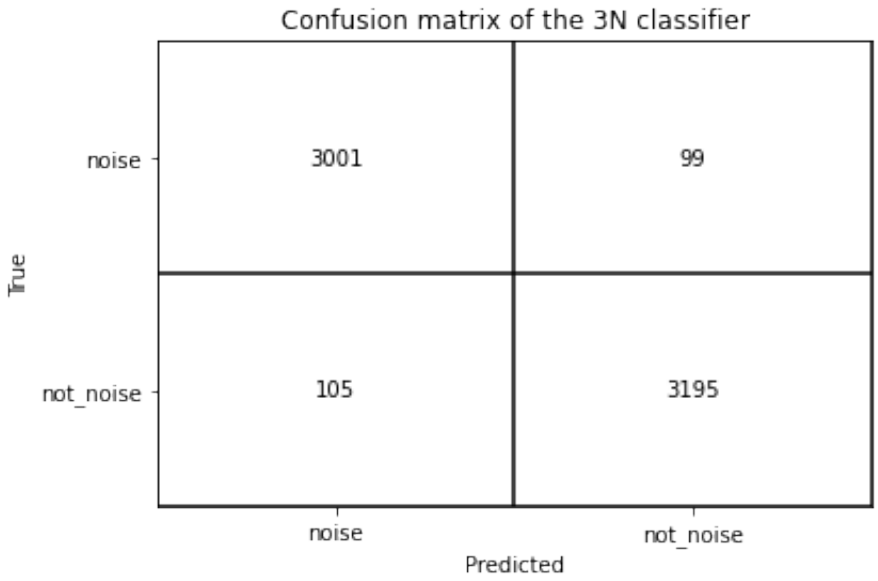


Figure 5.16: Confusion matrix showing the predictions made by the selected InceptionTime model

The final InceptionTime model was trained with normalized scaled data, time augmentation, noise augmentation, Early Stop and RLROP. Despite its many layers, the model only has 1 million trainable parameters, largely due to using Global Average Pooling instead of Dense layer(s) at the end of the Inception modules. Still, the model has a very long training time, worsened by the enlarged filter sizes. The model reports a validation accuracy of 96.8%, with precision and recall for both classes at ∼97%. Examples of prediction errors can be found in the appendix A.7. The model has similar predictions error as the other CNN models, many due to problems stemming from incorrectly labeled start times used by time augmentor. This model also struggles with outlier events of short intervals of strong displacement, including those not due to hardware problems. Figure A.44 shows a waveform which has been labeled noise, although the recording appears to start during an ongoing seismic event. It should be noted that this portion is likely caused by transient noise. The model predicts not-noise, which is reasonable, although technically incorrect with the given label. This shows that the model is able to handle waveforms where the event is not centered. The model does make at least one obvious mistake, seen in figure A.43. Although the event is negatively impacted by the mislabeled start time, enough of the waveform is present to clearly display a not-noise event. Still, there is a lot of potential for optimizing this model in future research.

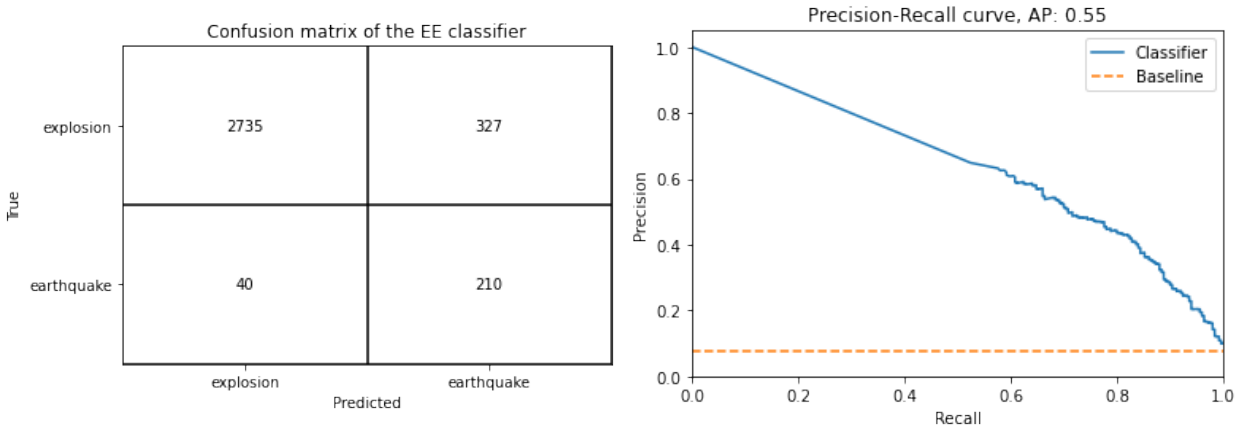
Earthquakes vs Explosions

Meier's Model The original model by Meier et al. was designed to distinguish short waveforms between signal and noise. However, the element of a sequence of convolution layers each with a higher filter count than the previous, proved to be a powerful architectural feature for this type of data. As figure 5.2.3 show, a similar design yielded good results, and so Meier's model was experimented on for the earthquake vs explosion section as well. The EE version differs from the 3N by using Max Pooling rather than Average Pooling prior to the Batch Normalization layer. These selection of these intermediate layers were made through grid searches and were not specified in the paper [7].

Optimizer: Adam
Loss: Categorical cross-entropy
Learning rate: 1e-3
Batch size: 48
Epochs: 40

Layer 1	**Layer 2**	**Layer 3**	**Layer 4**
Conv1D Nr filters: 32 Filter size: 16 Padding: same	MaxPool1D	Batch Normalization	Activation Type: ReLU
Layer 5	**Layer 6**	**Layer 7**	**Layer 8**
Conv1D Nr filters: 64 Filter size: 16 Padding: Same	MaxPool1D	Batch Normalization	Activation Type: ReLU
Layer 9	**Layer 10**	**Layer 11**	**Layer 12**
Conv1D Nr filters: 128 Filter size: 16 Padding: Same	MaxPool1D	Batch Normalization	Activation Type: ReLU
Layer 13	**Layer 14**	**Layer 15**	**Layer 16**
Flatten	Dense Units: 80 Activation: ReLU	Dense Units: 80 Activation: ReLU	Dense Units: 2 Activation: Softmax

Table 5.9: Architecture of the best performing CNN 3N inspired by Meier et al. Note that this architecture is inferred from the paper, and futher optimized by experimenting with intermediate layers, and does not necessarily reflect the architecture used to achieve their results.



(a) Confusion matrix

(b) PRC

Figure 5.17: The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve.

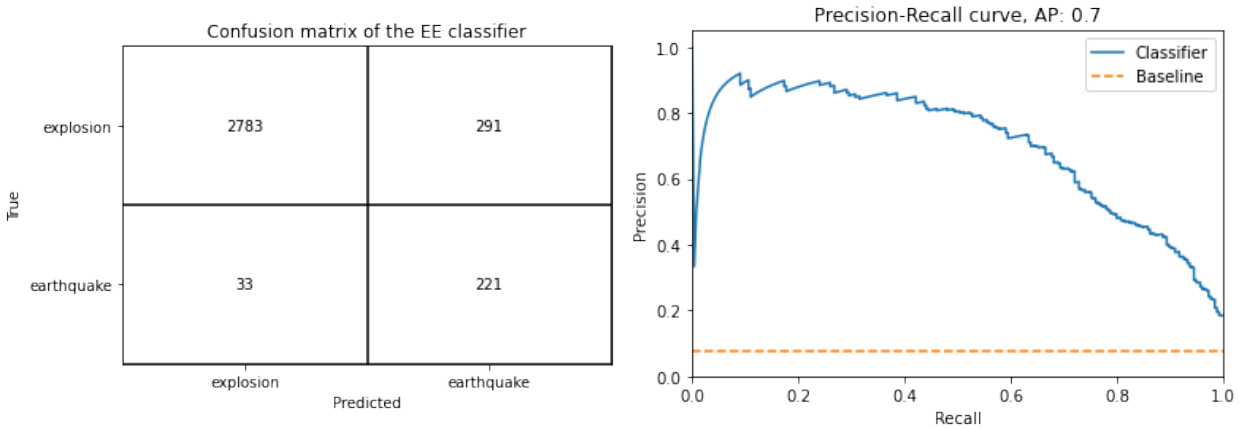
Similarily to its 3N counterpart, this model was trained with data preprocessed with normalized scaler, time augmentation and noise augmentation. The training sequence was influenced by Early Stopping and RLROP. The model achieves an F2 score of 0.68, which is significantly better than ∼0.29 (exclusively predicting earthquakes). The model correctly identifies 84% of the earthquakes with a precision of 39%. The more numerous earthquake class is correctly identified with a recall of 89% and a precision of 99%. The success of this model is surprising considering its small filter size. It appears that a useful feature maps of the waveforms may be possible with small filters. Analysis of the errors of the model should be left to domain experts, and so this paper will not attempt to speculate on errors unrelated to preprocessing for the EE models. Several of the mistakes made stem from inaccurately labeled start times, causing the time augmentor to cut out the start of events. In addition, some events have significantly less noise at the start of a waveform than the end, which is problematic for this augmentation technique. This will obfuscate the start of the event with disproportionate amounts of noise, which may be interpreted as related to the event by the model. Experimenting with a user defined buffer could be successful at mitigating this problem, although this would not be an issue on real data. A selection of prediction errors of this model can be found in the appendix A.4. The improved performance of Max Pooling layer over Average Pooling layer is speculated to be due to Max Pooling extracting the most prominent features. While classifying such similar classes focusing the model on the extremes of the feature map may help extrapolate the features key to distinguishing them.

Best Performing Self-Developed Model The architecture of the self developed model for the EE problem, initially looks very similar to the 3N version. They both share a the element of growing width between each convolutional layer, as well as Max Pooling between each of these layers. However, this model has a noticeably more and larger filters. This also uses dropout layers in between the convolutional layers, although at a very small rate. While performing optimization of this model, the results indicated that the model should be wider and deeper. The model already contains ∼135 M trainable parameters, so due to the increased training time and the larger size of the network, this was not done.

Optimizer: Adam
Loss: Binary cross-entropy
Learning rate: 1e-4
Batch size: 256
Epochs: 50

Layer 1	**Layer 2**	**Layer 3**	**Layer 4**
Conv1D Activation: ReLU Nr filters: 78 Filter size: 72 Padding: same L1 reg: 0 L2 reg: 0.001	Dropout Rate: 0.001	MaxPool1D	Conv1D Activation: ReLU Nr filters: 312 Filter size: 72 Padding: same L1 reg: 0 L2 reg: 0.001
Layer 5	**Layer 6**	**Layer 7**	**Layer 8**
Dropout Rate: 0.001	MaxPool1D	Conv1D Activation: ReLU Nr filters: 624 Filter size: 72 Padding: same L1 reg: 0 L2 reg: 0.001	Dropout Rate: 0.001
Layer 9	**Layer 10**	**Layer 11**	**Layer 12**
MaxPool1D	Flatten	Dense Activation: ReLU Units: 254	Dense Activation: ReLU Units: 234
	Layer 13		
	Dense Activation: Sigmoid Units: 1		

Table 5.10: Architecture of the best performing self-developed CNN EE model



(a) Confusion matrix

(b) PRC

Figure 5.18: The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve.

The model is trained using normalized scaler, time augmentation, noise augmentation, Early Stop and RLROP. This model is more flexible to other types of scalers, but the best results were obtained using normalize scaler. This model reports an F2 score of 0.72, which is significantly better than the baseline model. The validation set consist mostly of explosion events, 91% of which were correctly identified with a precision of 99%. Since the explosion events are so numerous, the relatively few misidentified explosions has a significant effect on the precision of earthquakes. The model correctly identifies 87% of the earthquakes, with a precision of 43%. Looking at 5.18b, one can see that the model performs reasonably well, although there seems to be a fairly consistent trade off in precision and recall at any threshold. Earthquake and explosion waveforms are visually nearly indistinguishable, and insights into the type of errors made by the model should be left to the domain experts. Yet, some errors can be inferred by visual inspection. Some of the false positives are due to time augmentation issues. A few of the waveforms have inaccurately labeled start times, which when processed by the time augmentor, causes the structural integrity of the waveform to be compromised. In addition, some of the waveforms contain very little noise prior to an event, compared to what is presumed to be noise, by the time augmentor, at the end of the waveform. When reorganizing slices of the waveform, the noisy end of the raw waveform is shifted to be between the event of interest and the relatively little noise at the start of the waveform. The model struggles on events where this is the case. Label smoothing could mitigate the effect of this during training, but it is not likely a problem if deployed on raw

data. Adjusting the uncertainty of the start time during augmentation could help mitigiate this as well, but seeing as how the model appears to struggle disproportionately on these errors, it would appear the majority of the data is structurally intact. Further investigation into the errors of intact waveforms is beyond the capabilities of the author. A selection predictions made by the model are displayed in the appendix A.6.

InceptionTime An EE version of InceptionTime was also created. This model differs from its 3N counterpart by being 2 Inception modules deeper, using residual connections and not using bottleneck layer in the modules. This model, although not as memory intensive as some of the other candidates, is notoriously slow to train, completing a step of size 64 events in 560 ms. Performing a grid search using a single GPU is expensive, being a limiting factor in the optimization of this model. The final model was trained with a batch size of 64, RMSprop optimizer with learning rate 1e-3 for 50 epochs. Time augmentation, noise augmentation, early stop and RLROP was used.

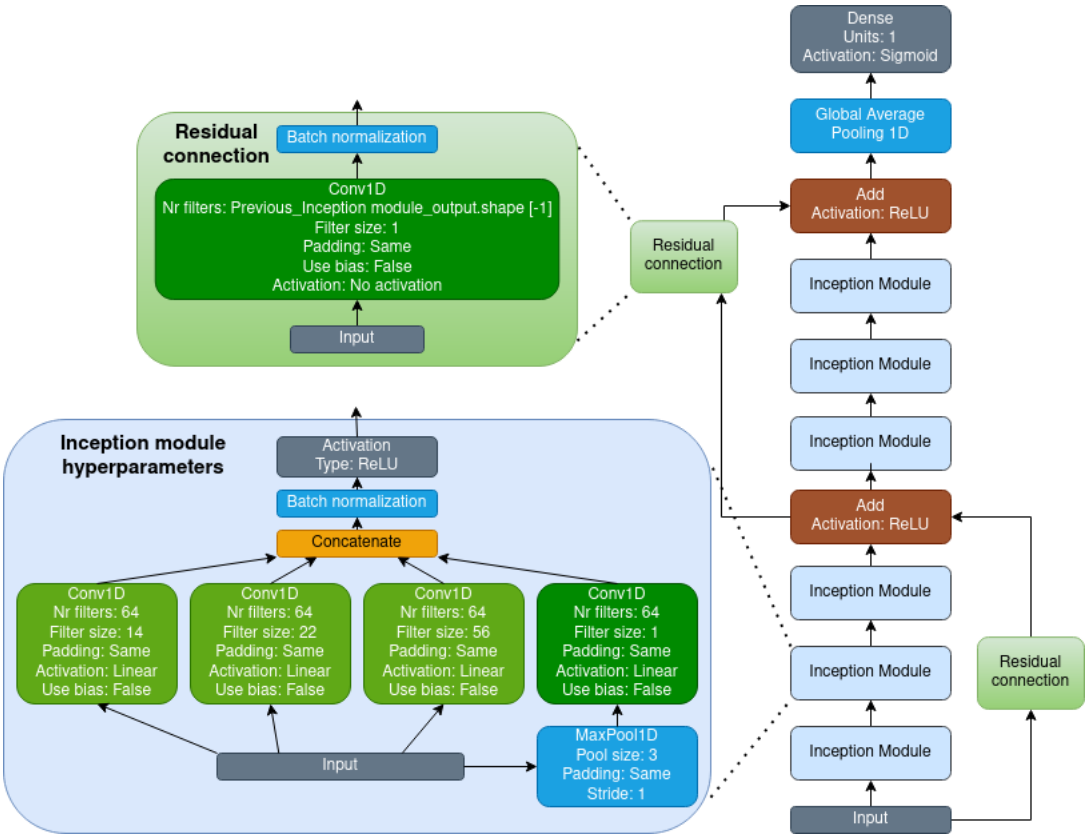
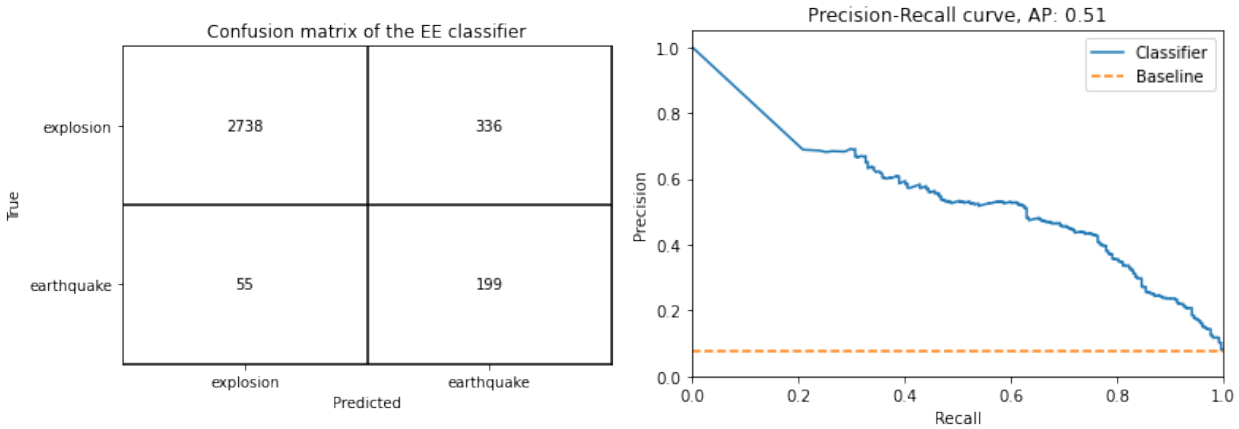


Figure 5.19: Illustration of the best performing InceptionTime variation on the EE task. Note that this model does not use bottleneck layers in the Inception modules.



(a) Confusion matrix

(b) PRC

Figure 5.20: The two figures illustrate the performance of the model in terms of its confusion matrix and Precision-Recall curve.

The model performs quite well on EE, achieving an F2 score of 0.64. The model indentifies 78.3% of the earthquake events, with a precision of 37.2%. For explosions, the model identifies 89% of them, with a precision of 98%. As seen on the PRC 5.20b, the model performs much better than random, although far from perfect. The model does overfit significantly, with a training accuracy of ∼100%. Regularization techniques such as L1 and L2 were attempted both inside the Inception modules and the residual connection, to no avail. As with the 3N version, this model too started off with too wide of a search space initially, only to show that hyperparameters near the default were more likely to yield better models. Examples of prediction errors can be found in the appendix, A.8. Speculation into the errors of EE models is difficult without domain expertise, but some mistakes are clearly due to issues stemming from preprocessing. As with many of the other models in this project, this too is negatively affected by mislabled start times, causing time augmentation to cut out the start of the event of interest. This model too struggles with waveforms which stem from hardware issues. This model does appear to make some notable mistakes, where events of high signal to noise ratio, not too distant and relatively high magnitude, are incorrectly labeled. All in all this model performs well, despite being negatively impacted by time augmentation. It is unclear to what extent the waveforms in this dataset contain incorrectly labeled start times, but this clearly negatively impacts the inference of the models, and likely also the kinds of features it learns during training.

5.2.4 Final Model

The final model is made up by the best performing model on 3N and the best performing model on EE. The table 5.11 below summarizes the performance of the best model in each category for each task.

Model	Noise vs Not-Noise Validation accuracy	Earthquake vs Explosion F2 score
FCNN	86.6%	0.35
MLSTM-FCN	95.6%	0.49
Meier et al.	97.6%	0.68
Self-developed	**97.9%**	**0.72**
InceptionTime	96.8%	0.64

Table 5.11: Summary of the performance of all of the models on both tasks. Note that the hyperparameters for the models differ between the two tasks. A randomly guessing 3N model would produce validation accuracy of ∼50%, and the EE equivalent would produce an F2 score of 0.292.

As table 5.11 shows, the self-developed models outperformed the other models in both tasks. This architecture uses features from Meier et al.'s model, such as three stacked convolutional layers with the increasing number of filters in each layer. The growth sequence of the filters is also identical, where the second layer has twice as many filters as the first, and third with 4 times the filters of the first. The differences come in the form of significantly larger filters, and significantly more units in the dense layers. The dense layers also use different number of units. As stated previously, the exact architecture of Meier et al.'s model is not described, but has been inferred and subsequently optimized.

Finally, now that the self-developed models have been selected as the best performers, they are combined into what effectively becomes the final model. The pipeline illustrating this merger is described in the Two Models In One 4.3.1 subsection. In order to gauge the final model's performance on new data, the previously untouched test set is used. The test set consists of 17642 events: 698 (∼3.96%) earthquakes, 8365(∼47.42%) explosions and 8579(∼48.63%) noise events. This test set is used only once for this final evaluation. These events in the test set are in their raw state, 9460 timesteps, which is incompatible with the input shape in which the two models were trained. Consequently, time augmentation is

needed in order to produce reshape the waveform. This means that any incorrectly labeled start times or other features in which makes this augmentation technique problematic, are also applied to the test set. Thus, the resulting evaluation does not reflect exactly how the model performs when deployed. Additionally, the waveforms in the NORSAR dataset have start times centered around the 1 minute mark. Through augmentation this temporal invariance is mitigated, but could be adversely affected by efforts to maintain the entirety of the event of interest in the augmented waveform. In short, the time augmentor has strict limits in which it can shift the event of interest. The downstream effects of the specifics of this cutoff are not studied, but is speculated to possibly create regions of the waveform the model has learned to ignore. For example: the very beginning and the end of the waveform. If this model is to be deployed, there are considerations, such as this, is necessitated by the design choices made for this project.

The selected model is made up the self-developed 3N model and the self-developed EE model. These models have not been retrained since producing the results described in this section. The choice to not train the models on the entire dataset, nor a larger subsample was made because of the use of upsampling. Since the non-earthquakes event make up the vast majority of the data, the earthquake events would need to be upsampled several times in order to balance the training set. The test data also needed to be preprocessed, in order to reflect the features in which the models were trained upon. As both of these models use normalize scaler, the scaler did not require fitting. The noise augmentor did, however, and depends on all previous preprocessing steps. The preprocessing used on the test set was fitted using the combined training, validation and psuedo-test set used for model selection. The test set was then transformed using the fitted preprocessing steps prior to being input into the model.

Class	**Precision**	**Recall**	**F2 score**
Noise	97.291%	97.995%	0.97854
Explosions	98.001%	89.085%	0.90736
Earthquakes	39.585%	79.226%	0.66006
Average	78.292%	88.769%	0.84865
Weighted average	95.352%	93.036%	0.93490

Table 5.12: Performance metrics of the final classifier on the test set.

Figure 5.21 shows the predictions made by the final model on the test set. The model performs as expected from the performance of its individual parts on validation. Each

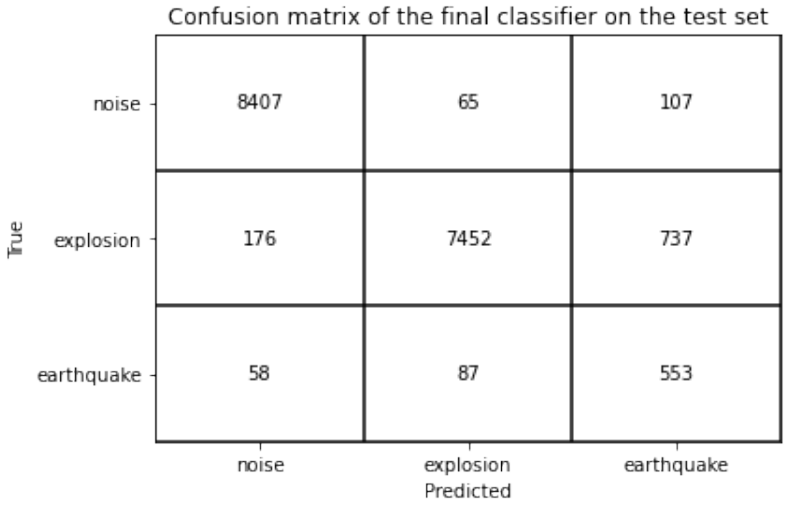
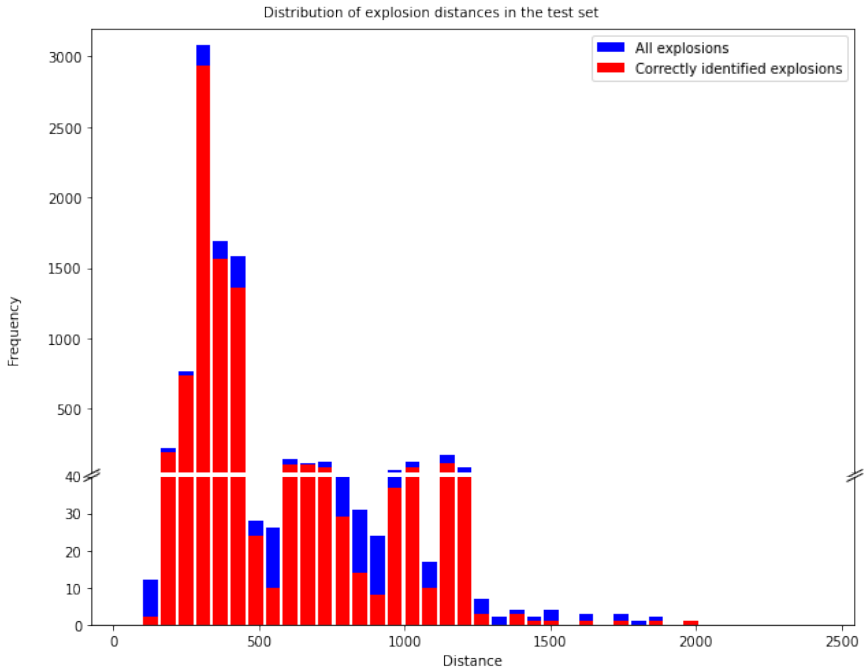
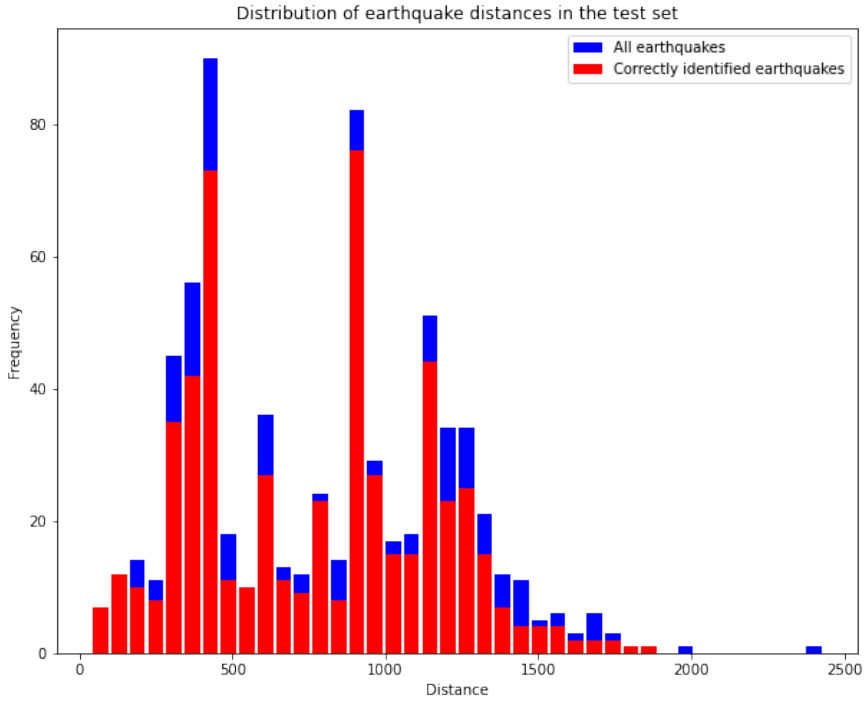


Figure 5.21: Confusion matrix showing the predictions made by the final model on the test set.

submodel makes one classification per millisecond (± 0.2 ms), meaning that the maximum inference time is ∼ 2 ms on an RTX 3090. The model identifies 97.99% of the noise events, with a precision of 97.3%. As the 3N component of this model works as a filter for the noise events, being able to removing the noise effectively is key. In real life, the vast majority of the recordings at NORSAR are made up by noise events. Being able to remove them effectively with minor misclassifications may be very useful. The final model has an F2 score of ∼0.98, where noise is the positive class. The model identifies 89.1% of the explosion events, with a precision of 98%. The majority of these unidentified explosions are classified to be earthquakes. This result is reasonable, as earthquakes and explosions are much more difficult to distinguish than explosions and noise. A similar result can be found the other way around, where most of the misclassified earthquakes are classified as explosions. The earthquake class performs the worst, where the model identifies ∼79%. This is done with a precision of ∼40% which is quite good considering how infrequent earthquake samples are in comparison to the two other classes. The F2 score of earthquakes is 0.66, a relatively small drop from the individual models' 0.74. All in all, the final model correctly classifies 16412 of the events, making 1230 mistakes. The accuracy of the classifier is 93%, although the reader should note the severe class imbalance. 8.4% of earthquake events are incorrectly labeled as noise, compared to 2.1% of explosion events. This suggests the 3N component is more effective at detecting explosion events rather than earthquake events.

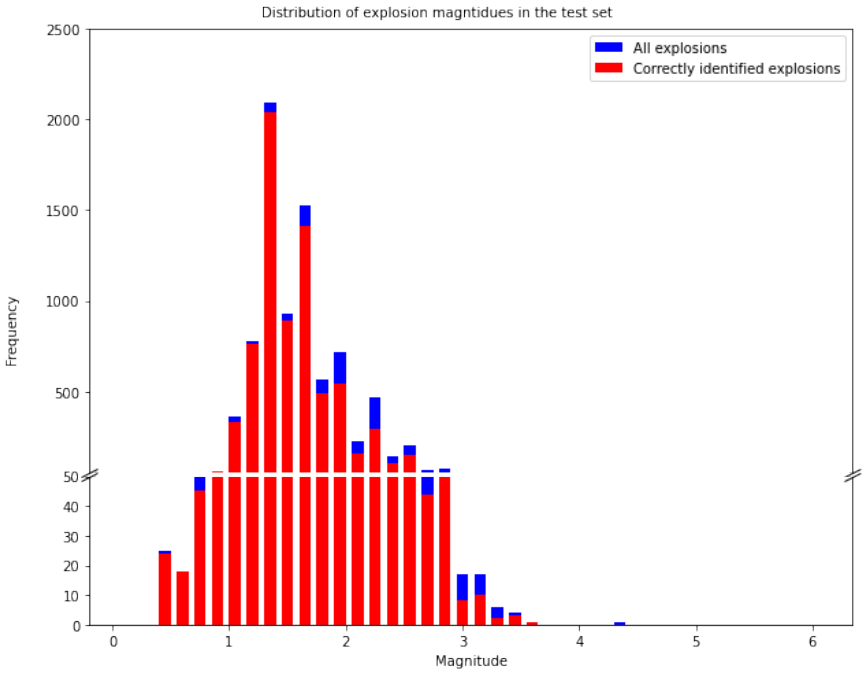


(a)

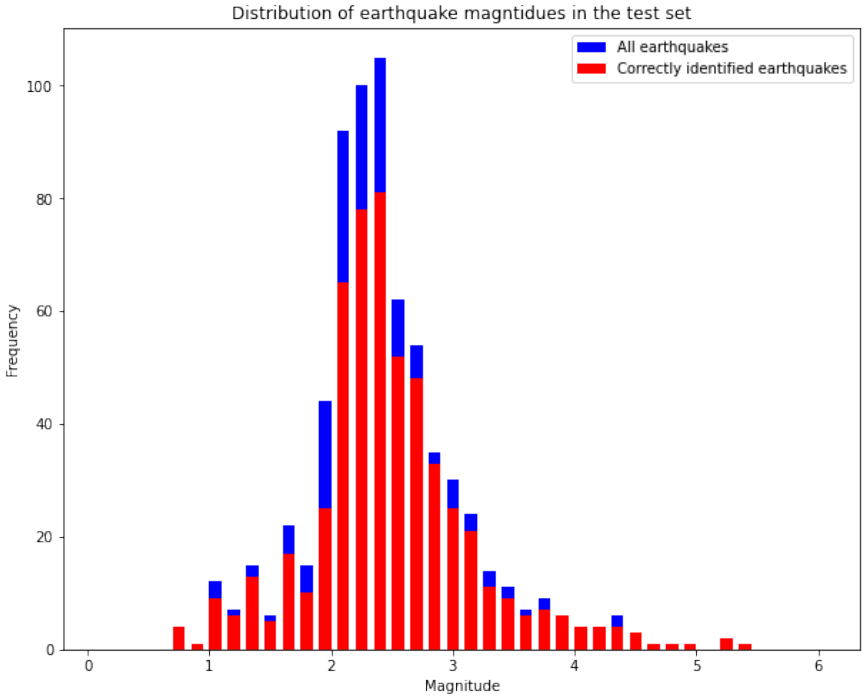


(b)

Figure 5.22: The two graphs show the distribution of distances from ARCES for explosions 5.22a and earthquakes 5.22b, overlayed by the distribution of the correctly identified class with respect to distance. Note the broken y-axis of 5.22a.

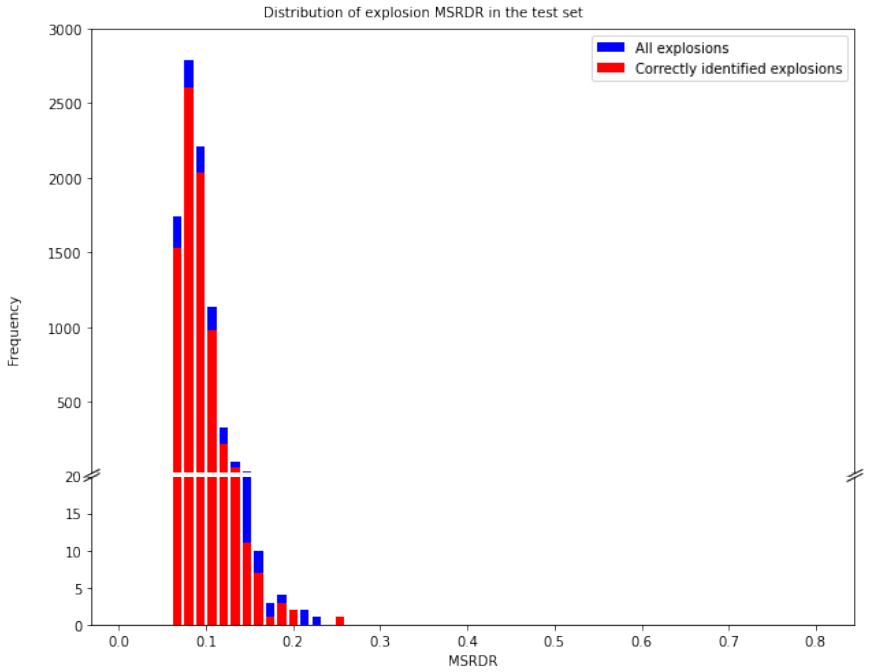


(a)

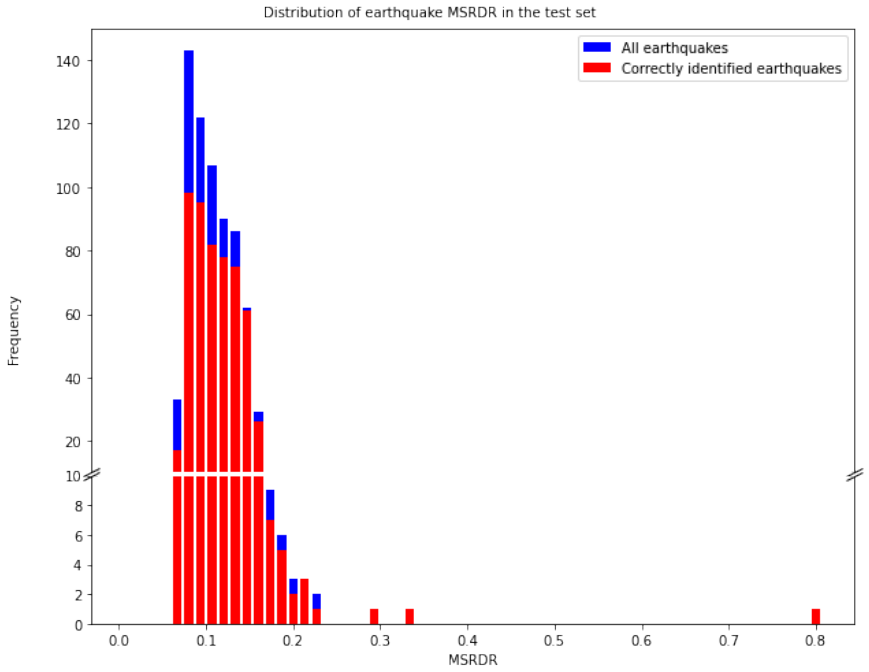


(b)

Figure 5.23: The two graphs show the distribution of magnitudes for explosions 5.23a and earthquakes 5.23b, overlayed by the distribution of the correctly identified class with respect to magnitude. Note the broken y-axis of 5.23a.



(a)



(b)

Figure 5.24: The two graphs show the distribution of MSRDR for explosions 5.24a and earthquakes 5.24b, overlayed by the distribution of the correctly identified class with respect to MSRDR. Note the broken y-axis of both figures.

The two graphs in figure 5.22 show the true epicentral distance distribution of the explosion and earthquake events. This graph illustrates how prediction errors relate the events distance from ARCES. It is immediately clear that the model is better at identifying explosions, but there is no significant direct relationship between the distance of an event alone and the models' ability to predict correctly. Looking at the teleseismic events in the explosion category, the model appears to struggle more with events occuring at distances greater than 1250 km, although there are not enough events above this threshold to make this claim with certainty. For earthquakes, all of them are predicted correctly for the first two bins, but the errors do not appear to display any significant relationship for earthquakes either. The lack of relationship to distance could be due to distance playing a more distinct role in the 3N submodel than the EE submodel.

Figure 5.23 show the distribution of magnitudes for the explosion events and the earthquake events in the test set, and how magnitude relates to the errors of the model. It appears that the model is better at classifying low magnitude explosions rather than high magnitude ones. This is likely due to the explosions in the ARCES dataset are typically of lower magnitudes. For earthquakes, the model makes less mistakes with high magnitude events rather than low. It appears the model has learned that features representative of a low magnitude event are more likely to be explosions and vice versa. It is interesting to see that the highest magnitude explosion event is not identified.

The graphs in figure 5.24 show how the model performs relative to MSRDR. The explosion graph indicates that the explosion events of low MSRDR are correctly identified at a rate higher than high MSRDR events. Considering this relationship with the distance and magnitude graphs, this relationship is attributed to the sensitivity to magnitudes rather than distance. The MSRDR is used analogously to signal-to-noise ratio, meaning that a higher MSRDR means high signal quality and vice versa. One would expect that both of these classes would display some relationship to the effect of: higher MSRDR, less errors. This appears to only be the case for earthquakes.

The figures (5.22, 5.23, 5.24) suggests that the model is more sensitive to the magnitude of the event rather than its distance, when distinguishing between earthquakes and explosions. The median explosion magnitude is much lower than the median earthquake magnitude in the ARCES dataset. If a model uses this feature as its sole distinction criteria, the model would perform better than random, but would not produce dependable classifications.

However, due to the data being scaled in each channel independently by normalize scaler, the information necessary to derive magnitude is lost, and so the model cannot depend on this feature alone.

As noise events do not have epicentral distance, magnitude nor MSRDR, no plots were made for these events. However, looking at these statistics when the model incorrectly predicts noise may provide insight. For the following graphs, for both explosions and earthquakes, the earthquake/explosion predicted in err to be noise, is put under the microscope.

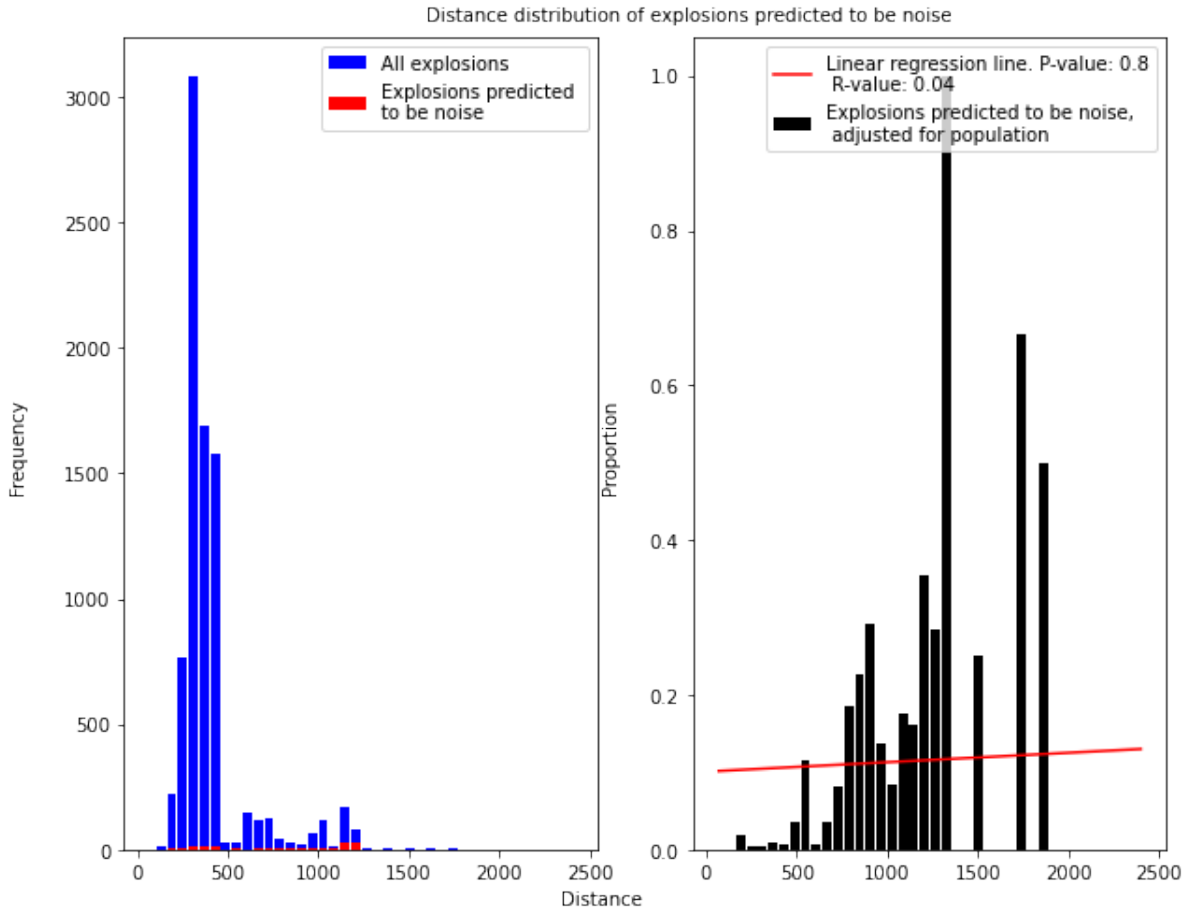


Figure 5.25: Shows the distribution of the explosions in the test set, and the explosions incorrectly labeled noise, with respect to distance. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of distance to the models' error.

The graphs in figure 5.25 shows how the distance of the event affects the prediction of noise for explosion events. The model does appear to predict noise more frequently as

distance increases, although this relationship is not statistically significant. It is clear that the model rarely makes noise predictions for the near explosions, but with the given sample, no significant relationship can be found. It should be noted that explosion with an epicentral distance shorter than 500 km make up the vast majority of the test set.

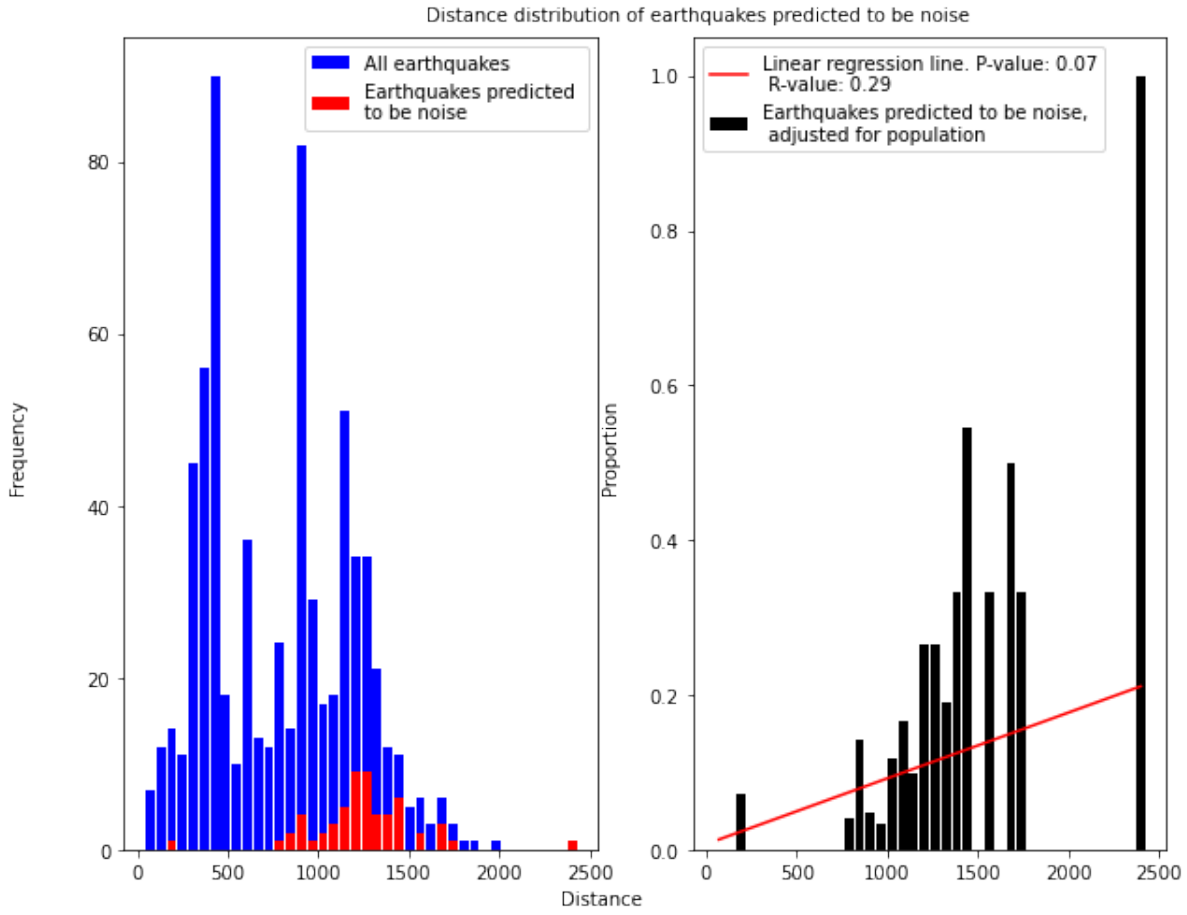


Figure 5.26: Shows the distribution of the earthquakes in the test set, and the earthquakes incorrectly labeled noise, with respect to distance. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of distance to the models' error.

Figure 5.26 shows the same relationship as 5.25, but for earthquake events. Here, there is a steeper positive correlation between distance and predicting noise, although statistically insignificant with a p-value of 0.07. The model only predicts noise for a few events less than 750km. Looking at the confusion matrix 5.21, figure 5.25 as well as this, one can see that the model is more likely to predict noise for earthquake events than explosion events. This plot

illustrates that the models' errors on earthquakes related to distance is more or less confined to the 3N submodel. This is likely the case for explosions as well but the insufficient number of distant explosions makes this relationship harder to justify.

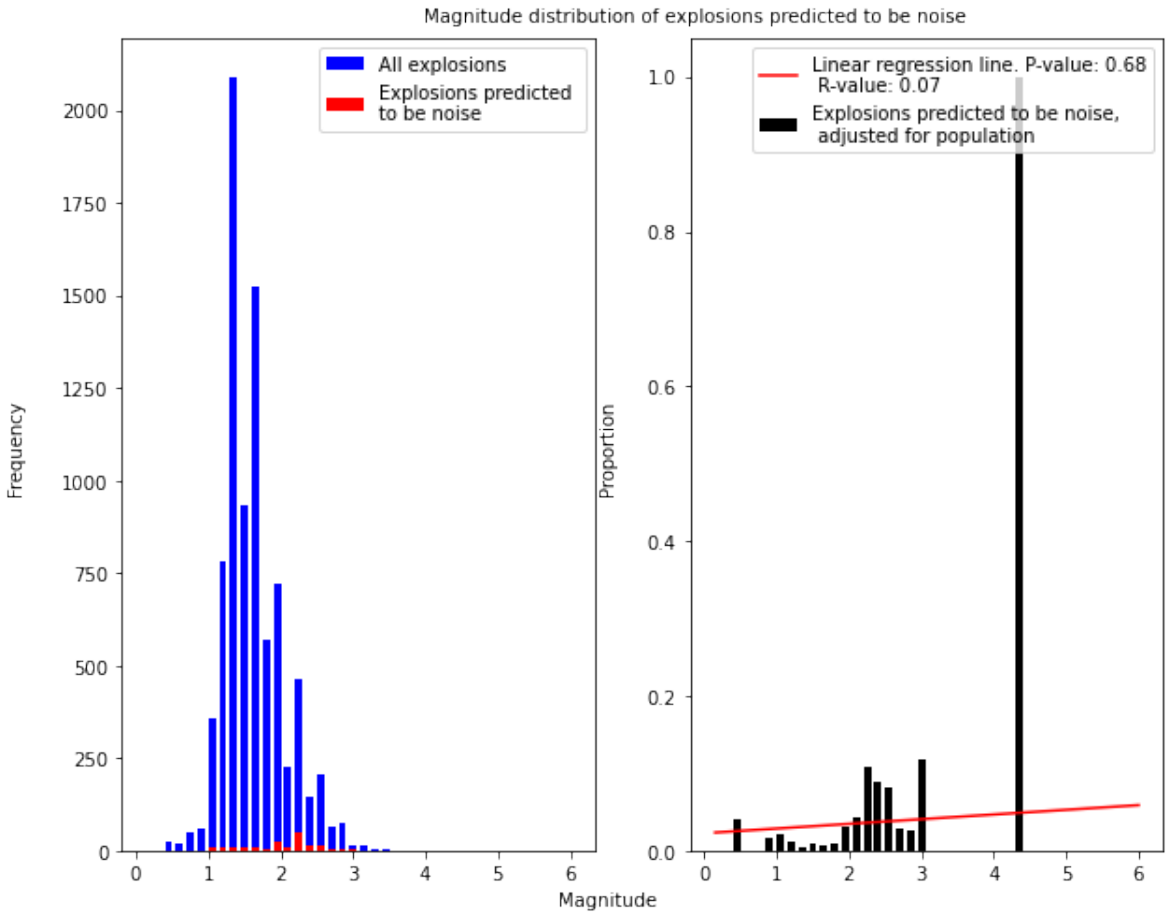


Figure 5.27: Shows the distribution of the explosions in the test set, and the explosions incorrectly labeled noise, with respect to magnitude. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of magnitudes to the models' error.

In figure 5.27 one can see the distribution of the magnitudes of explosions in the test set, and the distribution of the magnitudes of explosions classified as noise. There is a, statistically insignificant, small positive relationship between magnitude and the models' prediction of noise. Looking at this relationship without considering the distance of these events yields limited insight, but it is still interesting to see that the model is able to identify low magnitude events at a high rate.

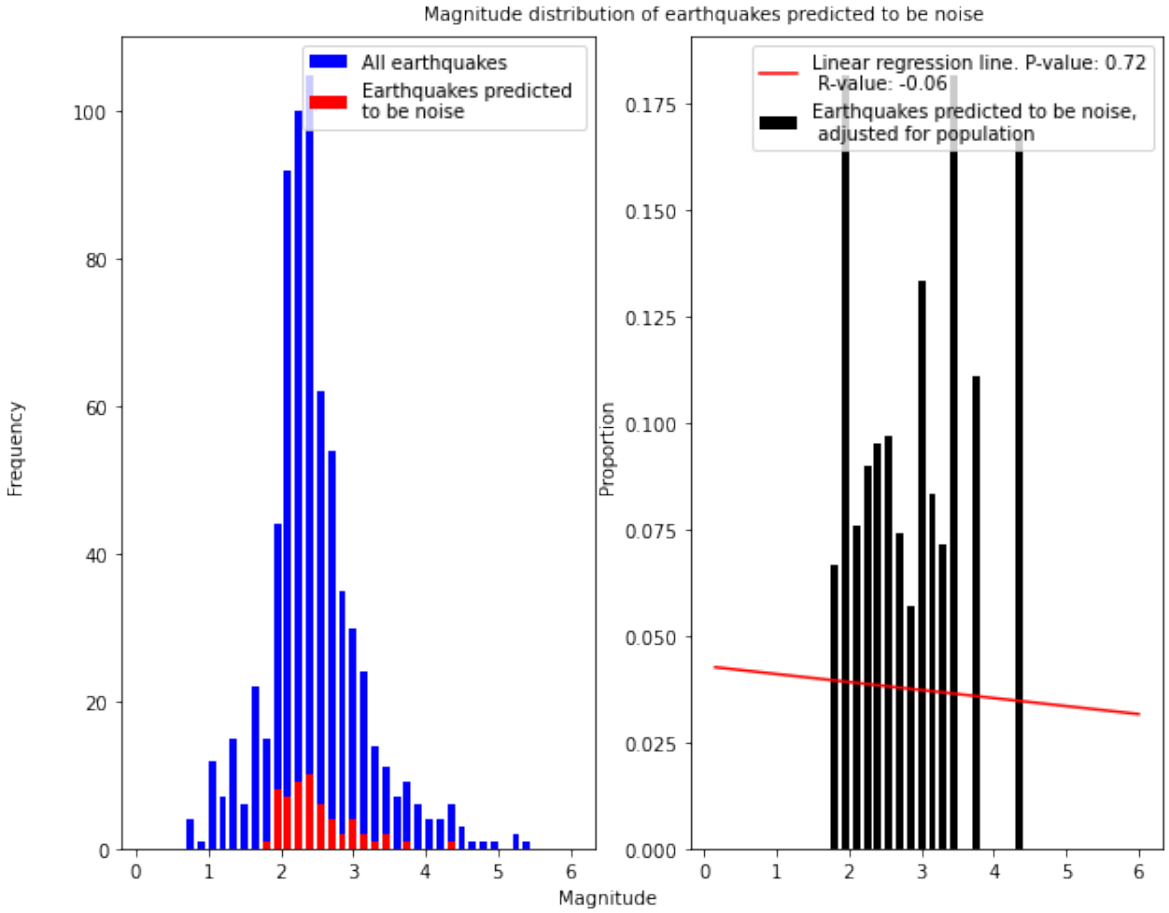


Figure 5.28: Shows the distribution of the earthquakes in the test set, and the earthquakes incorrectly labeled noise, with respect to magnitude. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of magnitudes to the models' error.

Looking at figure 5.28, there is no relationship between magnitude and the models prediction of noise. The distribution of earthquakes predicted to be noise appears to follow the distribution of magnitudes fairly evenly, suggesting that the magnitude of the event matters little when distinguishing between noise and earthquakes. In fact, at both extremes of magnitude, no noise prediction is made. The insight gained from this graph is also limited given the lack of consideration of distance.

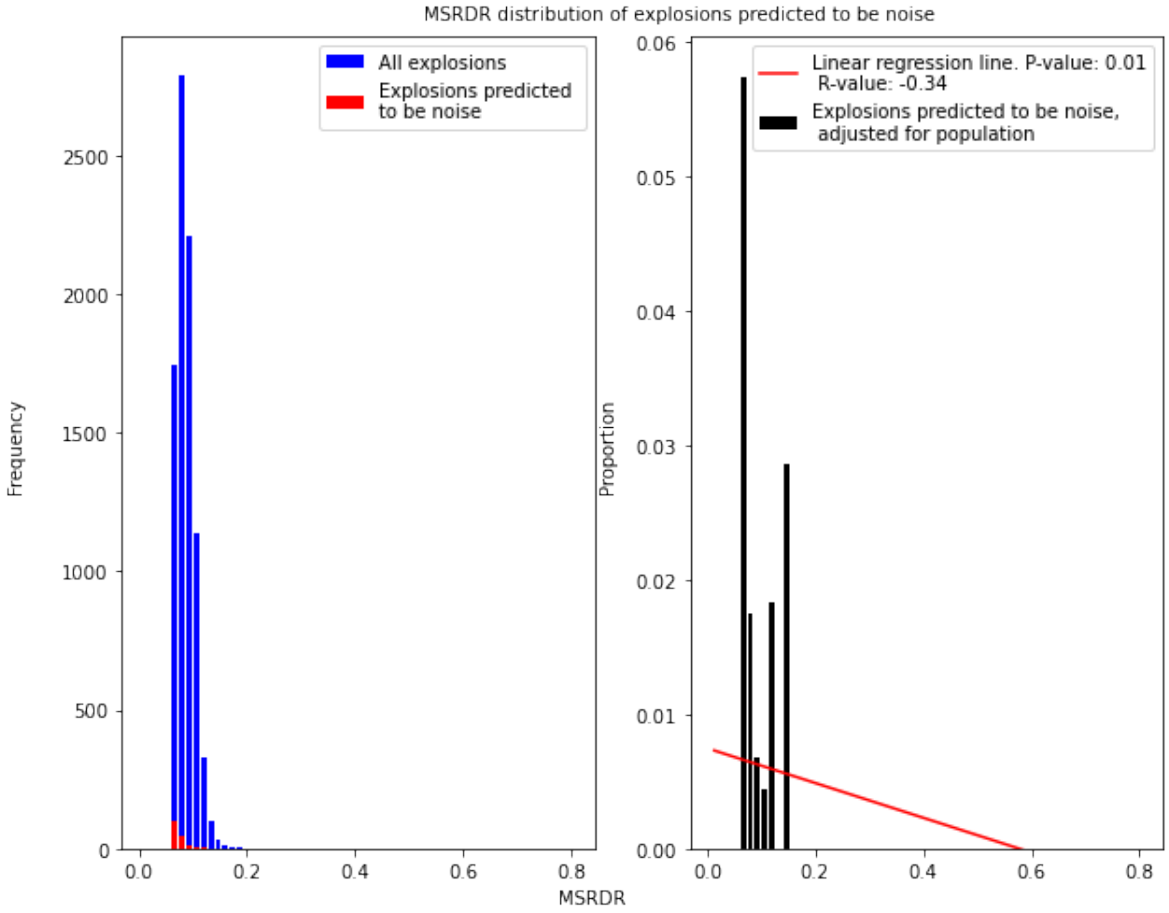


Figure 5.29: Shows the distribution of the explosions in the test set, and the explosions incorrectly labeled noise, with respect to MSRDR. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of MSRDR to the models' error.

The figure in 5.29 shows the relationship between MSRDR and the models prediciton of noise for explosion events. Here one can see a significant negative relationship between MSRDR and the prediction of noise, confirming that MSRDR functions as an indicator of signal-to-noise ratio. All of the incorrect noise predictions for explosion events are found when the MSRDR is low. There is a significant relationship indicating that the greater MSRDR is, the less likely the explosion is to incorrectly be labeled as noise. This relationship is not surprising, given that this model is trained to detect not-noise like activity in the waveforms, but rather confirms that the model is sensitive to these factors.

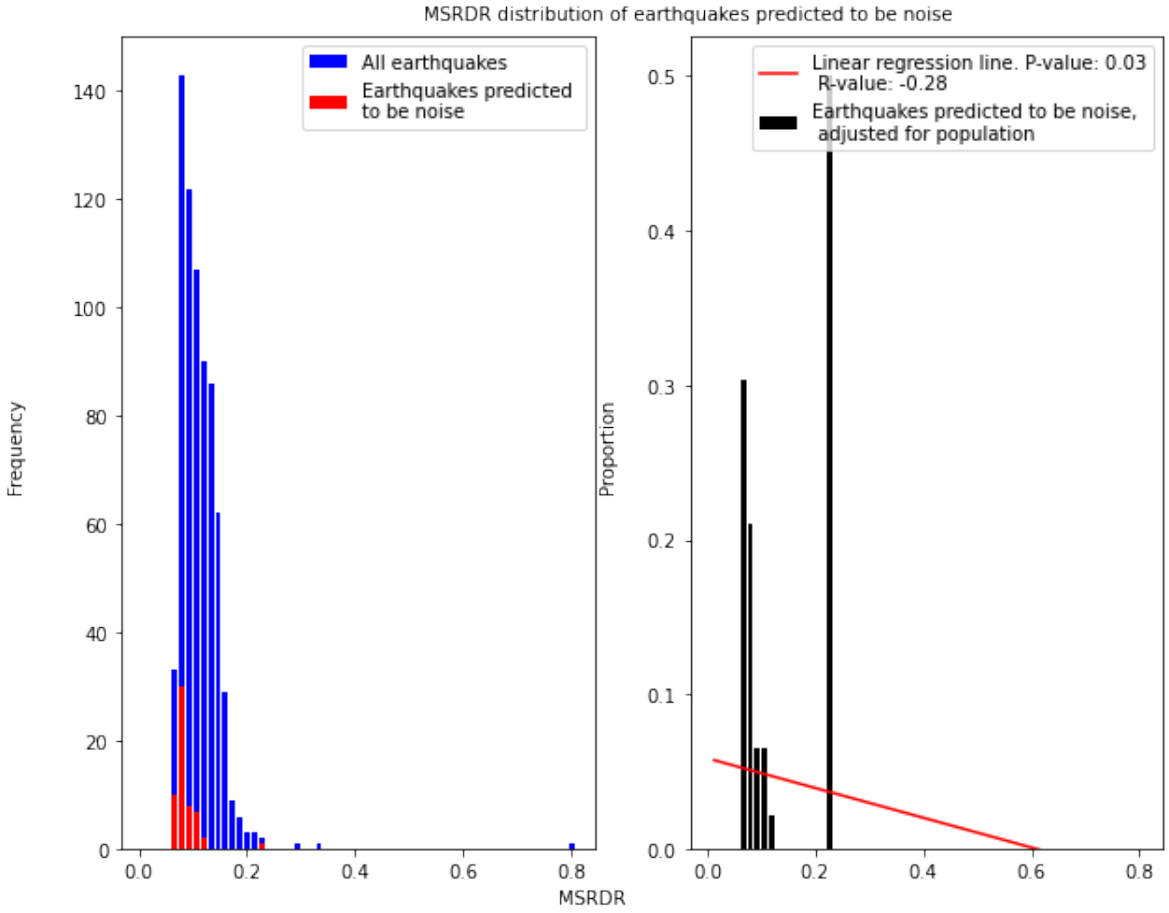


Figure 5.30: Shows the distribution of the earthquakes in the test set, and the earthquakes incorrectly labeled noise, with respect to MSRDR. The right graph shows these errors adjusted by the bin frequency. Finally, a linear regression line has been plotted to display the relationship of MSRDR to the models' error.

Similarily to its explosion equivalent, the figure 5.30 also finds a significant relationship between MSRDR and the incorrect classification of noise. This relationship shows that there is a negative correlation between an earthquake being classified as noise and its MSRDR. Unlike explosions, the model does make a noise classification of an earthquake event with MSRDR higher than 0.2, but in general the significant majority of the incorrect noise lables are found in the lower values of MSRDR. The insight gained from these plots are the same as its explosion equivalent, but also highlights that the 3N model is less sensitive to earthquake waveforms than explosions. This is speculated to be due to the earthquake waveforms in the training set being upsampled, unlike the explosion events. The upsampled events vary in

the temporal location of the earthquake due to time augmentation, but does not introduce new features of earthquakes to be learned.

The model performs quite well on the test set, comparable to what it achieved on validation. The results of the model show imprecision when identifying earthquakes, but can fairly reliably classify noise and explosion events. The noise-not-noise filter functions well, identifying all but 2% of the events. The harder task of distinguishing earthquakes from explosions does not achieve the same performance, but correctly classifies enough of the explosions and identifies a reasonable amount of the earthquakes to be considered useful. Looking at the models' earthquake predictions, whose true label is noise, one can find several waveforms which do not conform to the typical noise structure. The majority of these events are likely a result of transient noise whose expression mimics that of explosion or earthquakes. Still, it is not impossible for the model to identify previously undetected events in the test data. Examples of these events can be found in the appendix A.9.1. Statistically, it is unlikely that these events are infact earthquakes, but their interesting features are noted by the model.

Nuclear Explosions

NORSAR is the designated Norwegian National Data Center for the Comprehensive Nuclear Test Ban Treaty (CTBT). This international effort monitoring nuclear explosions, means that several stations and arrays across the globe have recorded the 6 nuclear weapons tests in North Korea. It would be interesting to see how the final model handles these waveforms, which clearly contain seismic events, although the type of which it has not been exposed to yet:

- "NK1": 2006-10-09T01:35:27
- "NK2": 2009-05-25T00:54:43
- "NK3": 2013-02-12T02:57:51
- "NK4": 2016-01-06T01:30:01
- "NK5": 2016-09-09T00:30:01
- "NK6": 2017-09-03T03:30:02

The list above contain the name of the event followed by the UTC date and time. The nuclear explosion dataset is made up of 23 waveforms, from 4 different arrays. USRK in south-east Russia (built in 2008, so the first event is not recorded), 400 km from the explosions. USRK only has vertical seismometers, and so the other two channels are filled with zeros. MJAR in Japan, 950 km away, but the waves need to travel through the seabed/sea prior to arriving at the array. KSRS in South Korea, 450 km distance from the epicenter. Both MJAR and KSRS both contain only one non-vertical seisometer. The recordings from these two arrays are not beamformed, but do contain three non-zero components. The last array is ARCES, 6250 km away. As this project has previously only worked with beamformed waveforms from the ARCES array, applying the model to data from several different arrays with and without beamforming, with and without three non-zero channels, may provide insight into the behavior of the model, and how it handles different types of data.

The nuclear explosion seismograms are originally 5 minutes long, centered to start at around the 1 minute mark. With a sample rate of 40 Hz, this makes the events 12 thousand timesteps. These events do not have labeled start times, so a manual cutting of the waveform was performed to make the shape compatible with the model. Normalized scaler and noise augmentation was used to transform the data prior to being input into the model. The model is the same as what was used for evaluating the final model, and is not retrained for this evaluation.

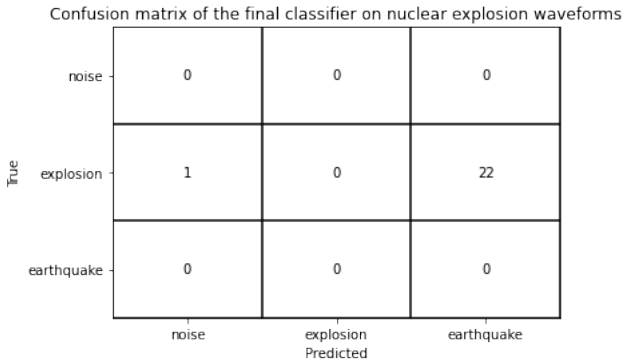
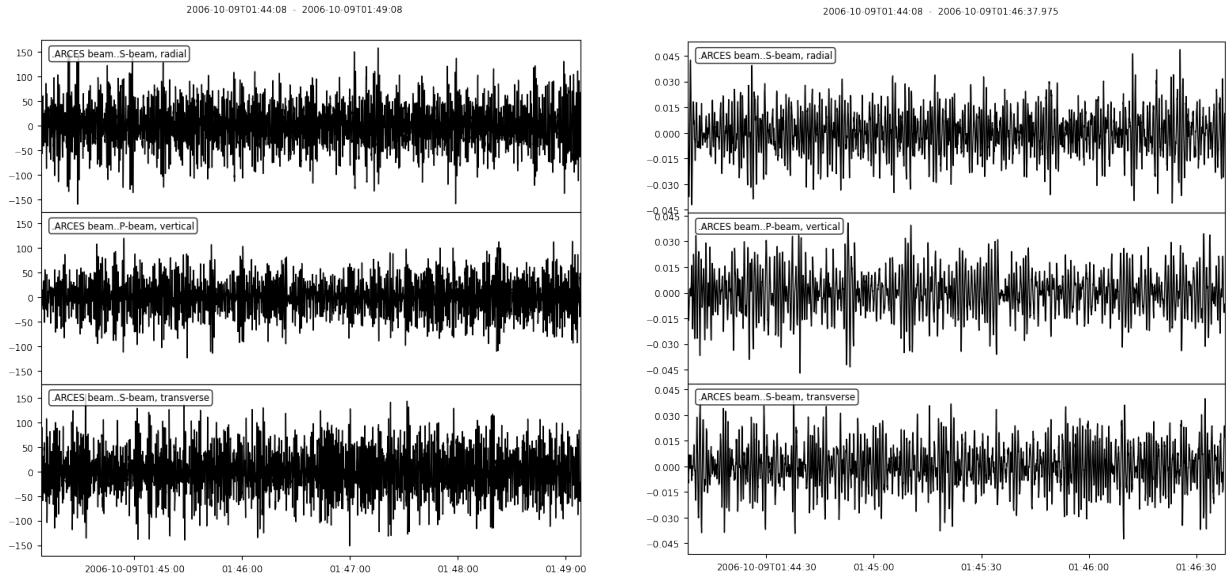


Figure 5.31: Confusion matrix of the models' classification when given nuclear explosion waveforms as input

Figure 5.31 shows the confusion matrix of the nuclear explosions recorded by all the stations. The nuclear explosion waveforms are marked with explosion as the true label. The model nearly exclusively predicts earthquakes for this data, except for one noise prediction.

The model predicts, with a high degree of certainty, that the events are at least not noise events. Further, for all of the waveforms, but two, the model outputs values higher than 0.99, by the EE model. The two waveforms of lower certainty have outputs 0.54 and 0.55. The 0.55 output is the recording of NK1 and the 0.54 is of NK2, both from the MJAR array. From the 3N submodel the NK1 MJAR reording has an output of 0.63, while the NK2 has output 0.69.

The nuclear explosions are labeled by the author to be considered explosions. It would appear that nuclear explosions either share few features with the explosions in the training set or many with the earthquakes in the training set.

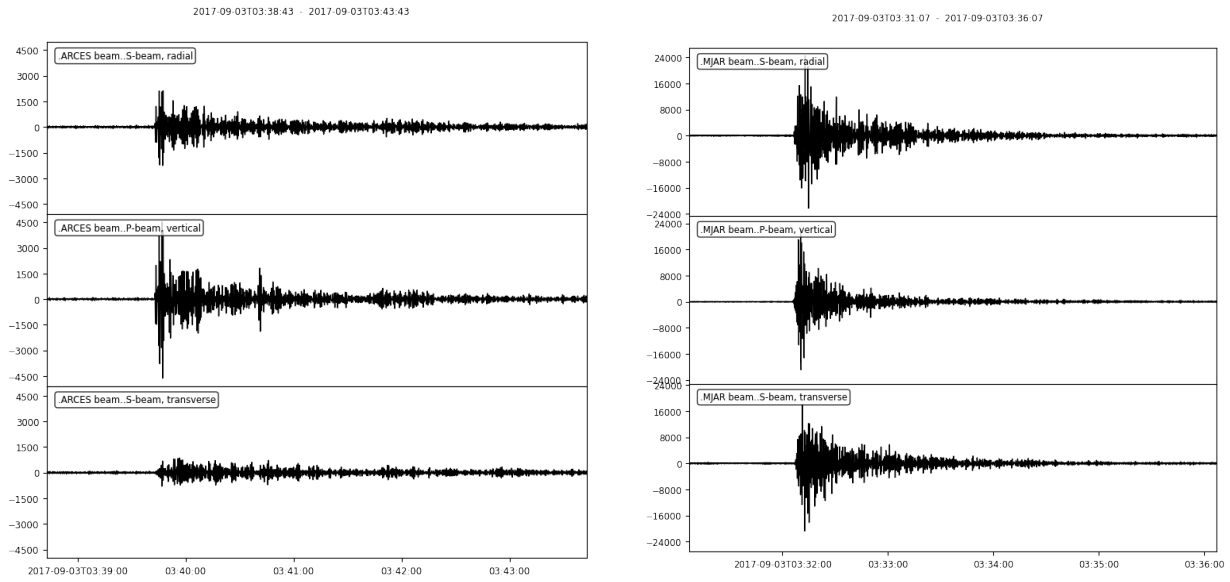


(a) *(b)*

Figure 5.32: The two waveforms are of the first North Korean nuclear test, NK1. This recording by ARCES is 6250 km away from the detonation. The 3N submodel outputs 0.01, while it the correct output is 1. The raw waveform can be seen in 5.32a and the transformed waveform is seen in 5.32b. The transformation consists of normalize scaling, noise augmentation and a manual shortening of the event.

The noise classification is ARCES recording of NK1. NK1 is thought to be a "fizzle", which means the nuclear explosion grossly fails to meet its expected yield. The model outputs uncertainty with the same recording, but from 950 km away, and so it is not surprising that a fizzle event 6250km away is indistinguishable from noise by the model. The model

consistently detects that the recorded event is at least not noise, indicating that it is sensitive to events occurring at a greater distance than the data in its training data.



(a) *(b)*

Figure 5.33: The two waveforms contain the NK6 nulear explosion, where 5.33a shows the ARCES recording (6250 km) and 5.33b shows the MJAR recording (950 km).

The two raw waveforms in figure 5.33 are of the most recent nuclear explosion in the nuclear explosion dataset. These examples are included to illustrate how nuclear explosions appear on seismographs, and show how the waveform is impacted by distance as well as the propagation medium. The ARCES recording has significantly less displacement in stacked counts compared to the Japanese recording, however both were correctly identified to not be noise by the model. It should be noted that the model is not trained to classify these types of events, but still identifies them to be of interest, despite their very distant origin.

Chapter 6

Summary

The goal of this project was to see if deep neural networks could classify beamformed seismic data from an array into three distinct categories. When evaluating the success of this goal at a binary level, one would only need to see if the models perform better than random. The results of the final model clearly show that it performs significantly better than random. The more relevant question to answer is to what degree did the final model accomplish its goal. This is harder to answer, and should arguably be answered by relevant stakeholders. I interpret the results of the final model like this: when the model makes a noise or explosion prediction, it is very likely correct. It still makes mistakes, so in practice any classification of these classes should be manually confirmed by analysts, especially when the model output indicates uncertainty. The events in which the model outputs earthquake should definitely be manually checked. The model will be correct $\sim$40% of the time when predicting earthquake, so blindly relying on the model will produce a lot of false positives.

An important reason for developing models which automate the classification process is to reduce the workload of analysts. In practice, I think the model can help highlight interesting segments of the continuous recording, which warrant further investigation. Further, the model can make an initial classification, which may speedup the analysis process. A useful feature of neural networks is that the output can be interpreted as the probability of the input belonging to that particular class. An analyst can use the probability output to have an idea of how certain the model is about the classification. Thus, I think the value of this model comes as a first-glance analysis of any given waveform. It is correct more often

than its not, but it still makes mistakes. How much this reduces the workload of analysts is beyond what I can answer, but I think the model can be a useful tool if used appropriately.

The labeled start times have been key to mitigating the effects of the unbalanced dataset. The time augmentor allowed for significant upsampling, while offsetting the negative consequences. Unfortunately, this type of augmentor does not introduce new earthquake features for the model to learn, but makes it so the model is more flexible to where in the waveform the event occurs. Shifting the event of interest also means shifting the noise, helping the model learn to ignore these uninteresting segments. Combining this with the noise augmentation and this sensitivity to noise is further mitigated. The automatically labeled start times did turn out to be somtimes imprecise and sometimes completely off, leading to the time augmentor compromising the structural integrity of some waveforms. Tuning the buffer between the labeled start time and the cut off time would reduce the impact of this shortcoming. It is futile to manually oversee this tuning holistically, so doing this through an additional hyperparameter during model selection is likely a good way to do this.

Themes for this project include complicated models with long training times, a practically infinite potential search space for each model, limited access to adequate computational resources, and a restrictive deadline. Therefore, the results reported in this project are far from the "best" possible for each model. I speculate that with a more fine grained filtration of "bad" recordings, optimally tuned time and noise augmentors, label smoothing, long sequences of iterative improvements on models found during continuous grid searches will have significant impact on the results on this dataset. It could also be interesting to convert the waveforms input of the EE submodel to spectrograms instead, such as in the work by Tibi et al. [33]. Manual analysis of seismograms include looking at individual frequency bands; training the EE submodel on spectrograms could yield more reliable results.

It is difficult to gauge the success of using two individually optimized models instead of one. The decision to split the classification task in two immediately improved the results, but without comparison to a representative model I cannot make a claim one way or the other. Having one model strictly filtering out non-interesting events is very useful, and reduces the workload of the subsequent model tremendously. Fewer nodes in the output layer also reduces the complexity of each model. Optimizing them individually is very useful when dealing with classes which are very similar in features. The downside of this design choice is that it doubles the amount of models which need to be optimized, which, in effect, halves the

attention each model receives in this project. This lead to aggressive strategies in Early Stop and RLROP, which more than likely discarded models which otherwise would yield promising results. A relatively low number of epochs were used in order to cover a larger section of the search space. The overly aggressive training techniques and likely under-training of models leaves a lot of potential just from their reduction.

The final (combined) model makes a classification every ∼2 ms, which allows an implementation of the model to make ∼12 classifications per sample, as the sample rate is 40 per second. There is some variance in the output of a single waveform (due to noise augmentation), in which a voting system could be developed to reduce the impact of this variability. A sliding window could prove useful to offset the effects of short strong displacement in stacked counts on the normalized scaler. In general, I think the model would perform better on live (but beamformed) data, as a lot of the errors made by the model is due to either a poorly tuned time augmentor or inaccurately labeled start times, depending on your perspective.

There is a lot of potential in improving the results on this dataset. Manual classification of low magnitude events is difficult, and is prone to errors when the signal is noisy. As a result there are likely several incorrectly labeled low magnitude events, which are counterproductive to the training process. In addition, there are some seemingly incorrect labels (at least to someone who is not a domain expert), such as A.30 in the appendix. Manually relabeling all of the events is clearly infeasible, so other techniques such as label smoothing could be useful in future research. The dataset also contains quite a lot of outlier events stemming from hardware issues. How much these events impact the training process is unclear, but it certainly limits the types of scalers that can be used productively. The performance of the any supervised model, is limited by the integrity of the data. For this project, the best possible model would achieve performance equal to that of analysts. As with most real datasets, this integrity is less than optimal and should be considered when interpreting the results of the final model. The 3N component achieves high accuracy despite these discrepancies, producing results comparable to NORSAR existing event detection method. However, neither model is perfect, leaving potential improvement for future research and development.

Chapter 7

Bibliography

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016. arXiv:1409.0473.

[2] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

[3] K. Bergen, T. Chen, and Z. Li. Preface to the focus section on machine learning in seismology. *Seismological Research Letters*, 90:477–480, 2019.

[4] J. Brownlee. *Deep Learning for Time Series Forecasting - Predict the Future with MLPs, CNNs and LSTMs in Python.* None, 2018.

[5] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[6] Geoffrey Hinton et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29, 2012.

[7] Meier et al. Reliable real-time seismic signal/noisediscrimination with machine learning. *Journal of Geophysical Research: Solid Earth*, 124, 2019.

[8] Hassan Ismail Fawaz, Benjamin Lucasand Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Min Knowl Disc*, 34, 2020.

[9] J. Feng and S. Lu. Performance analysis of various activation functions in artificial neural networks. *Journal of Physics: Conference Series*, 2019.

[10] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 10 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015.
URL: `https://doi.org/10.1162/089976600300015015`.

[11] S. Gibbons and F. Ringdal. The detection of low magnitude seismic events using array-based waveform correlation. *Geophysical Journal International*, 165:149–166, 2006.

[12] S. Gibbons, J. Schweitzer, T. Kværna, and M. Roth. Enhanced detection and estimation of regional s-phases using the 3-component arces array. *Journal of Seismology*, 23(2): 341–355, 2018.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Mandar Haldekar, Ashwinkumar Ganesan, and Tim Oates. Identifying spatial relations in images using convolutional neural networks. *International Joint Conference on Neural Networks (IJCNN)*, 06 2017.

[15] J. Havskov and G. Alguacil. *Instrumentation in Earthquake Seismology*. Springer, 2 edition, 2016.

[16] J. Havskov and L Ottemoller. *Routine Data Processing in Earthquake Seismology*. Springer Publishing, 2010. doi: https://doi.org/10.1007/978-90-481-8697-6.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. arXiv:1512.03385.

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[19] F. Karim, S. Majumdar, and H. Darabi. Insights into lstm fully convolutional networks for time series classification. *IEEE Access*, 7:67718–67725, 2019. doi: 0.1109/access.2019.2916828.

[20] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237–245, Aug 2019. ISSN 0893-6080. doi: 10.1016/j.neunet.2019.04.014.
URL: `http://dx.doi.org/10.1016/j.neunet.2019.04.014`.

[21] Y. Kim, J. Sa, D. Park, and S. Lee. Resource-efficient pet dog sound events using lstm-fcn based on time-series data. *Sensors*, 18(11):4019, 2018.

[22] Q. et al. Kong. Machine learning in seismology: Turning data into insights. *Seismological Research Letters*, 90:3–14, 2019.

[23] J. Kortström, M. Uski, and T. Tiira. Automatic classification of seismic events within a regional seismograph network. *Computers and Geosciences*, 87:22–30, 2016.

[24] Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with hivecote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, 12:1–35, 07 2018. doi: 10.1145/3182382.

[25] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015. arXiv:1411.4038.

[26] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help?, 2020. arXiv:1906.02629.

[27] NORSAR. Norsar seismic bulletins, 1971.
URL: `https://doi.org/10.21348/b.0001`.

[28] L. Prechelt. Early stopping - but when? *Lecture Notes in Computer Science*, pages 55–69, 1998. doi: https://doi.org/10.1007/3-540-49430-8_3.

[29] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, pages 533–536, 1986. doi: https://doi.org/10.1038/323533a0.

[30] R. Spence, E. So, and C. Scawthorn. *Human Casualties in Earthquakes: Progress in Modelling and Mitigation (Advances in Natural and Technological Hazards Research Book 29)*. Springer, 2011.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.
URL: `http://jmlr.org/papers/v15/srivastava14a.html`.

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. arXiv:1409.4842.

[33] R. Tibi, L. Linville, C. Young, and R. Brogan. Classification of local seismic events in the utah region: A comparison of amplitude ratio methods with a spectrogram-based machine learning approach. *Bulletin of the Seismological Society of America*, 109(6), 2019.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[35] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline, 2016. arXiv:1611.06455.

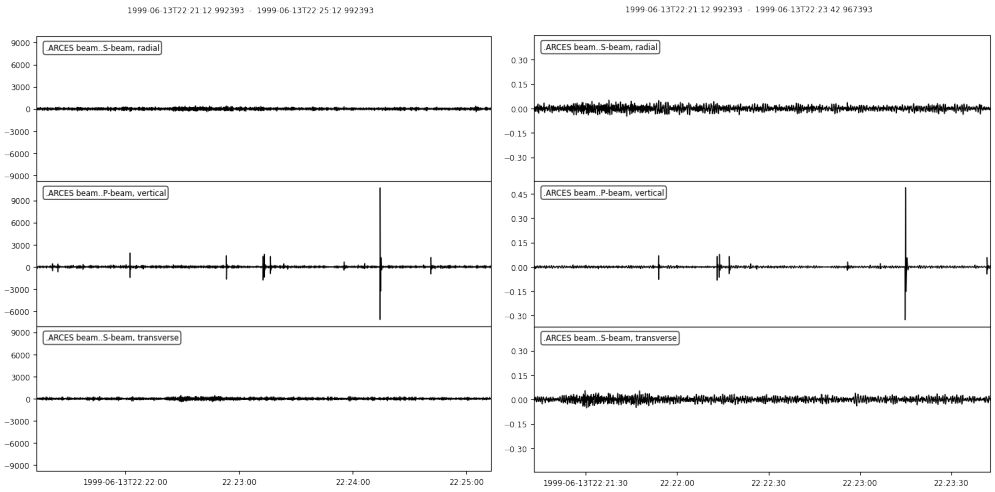
Appendix A

Selected Predictions

In the following figures the raw waveform is in the left column, and will have labeled (a). The transformed waveform will be captioned (b). Each of the transformed waveforms in the appendix has been normalize scaled, time augmented and noise augmented.

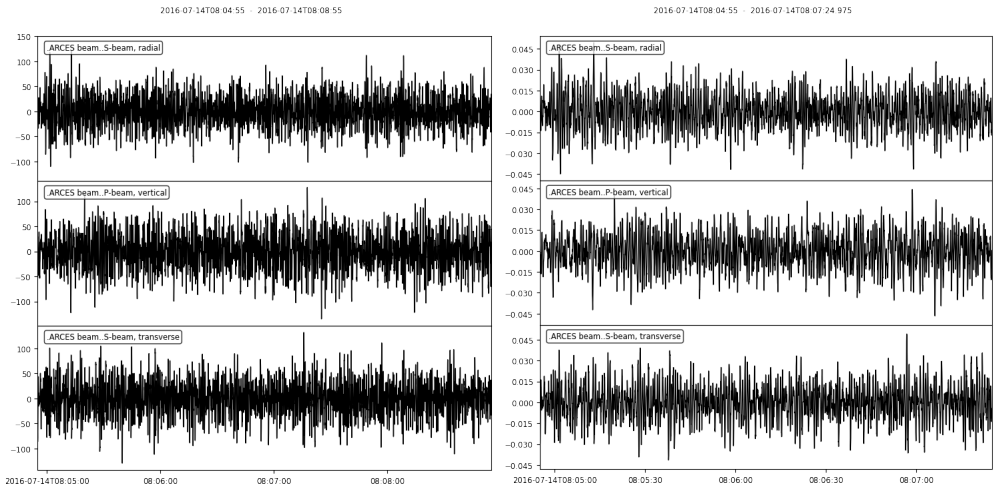
A.1 MLSTM-FCN 3N

A.1.1 Selected False Positives



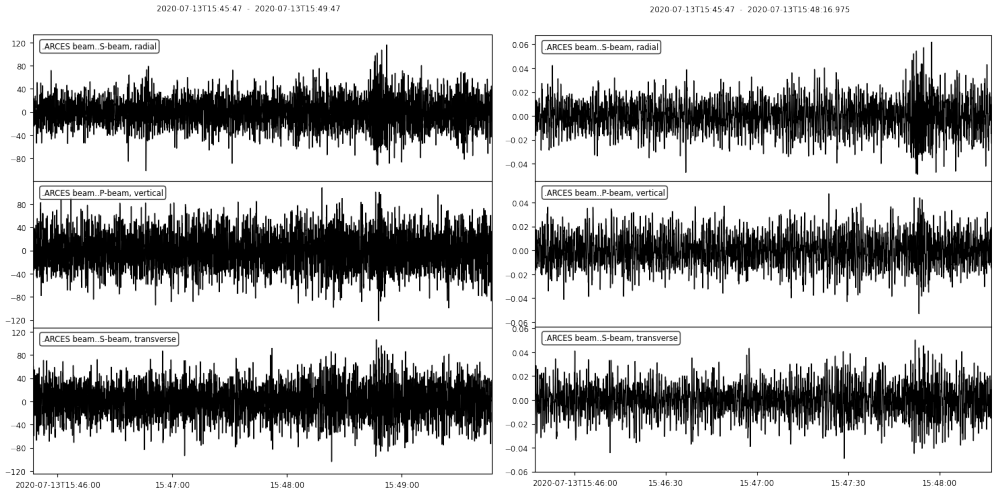
(a) *(b)*

Figure A.1: Predicted output: 0.59, correct output: 0. Noise. Outlier waveform, likely hardware error.



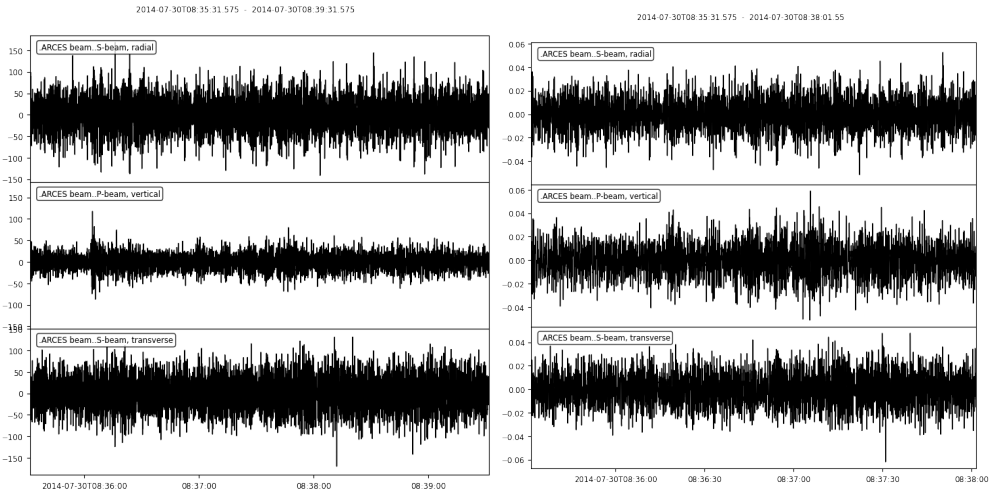
(a) *(b)*

Figure A.2: Predicted output: 0.82, correct output: 0. Noise. Difficult to distinguish without domain expertise, but a notable mistake made with relatively high certainty.



(a) *(b)*

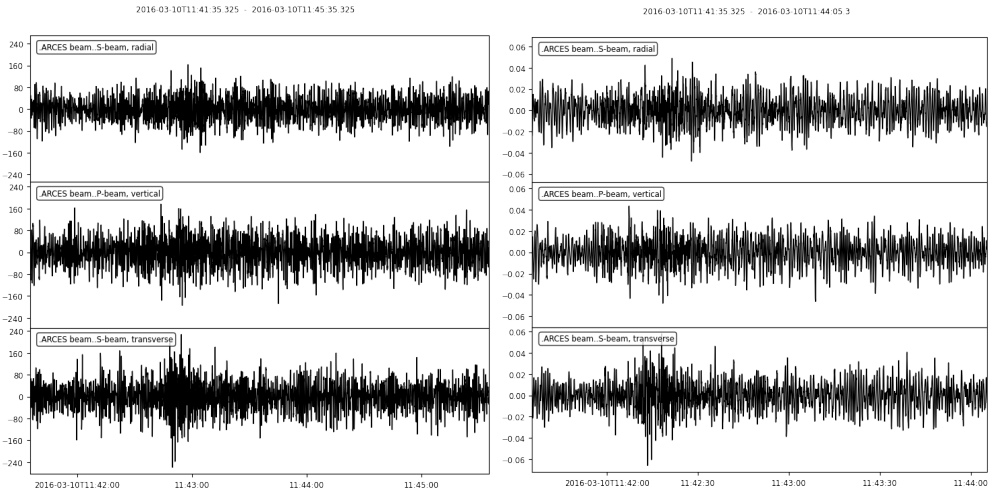
Figure A.3: Predicted output: 0.62, correct output: 0. Noise. Appears to be a typical noise waveform, although with a short minor period of particular displacement. Speculated to be the cause for the classification error.



(a) *(b)*

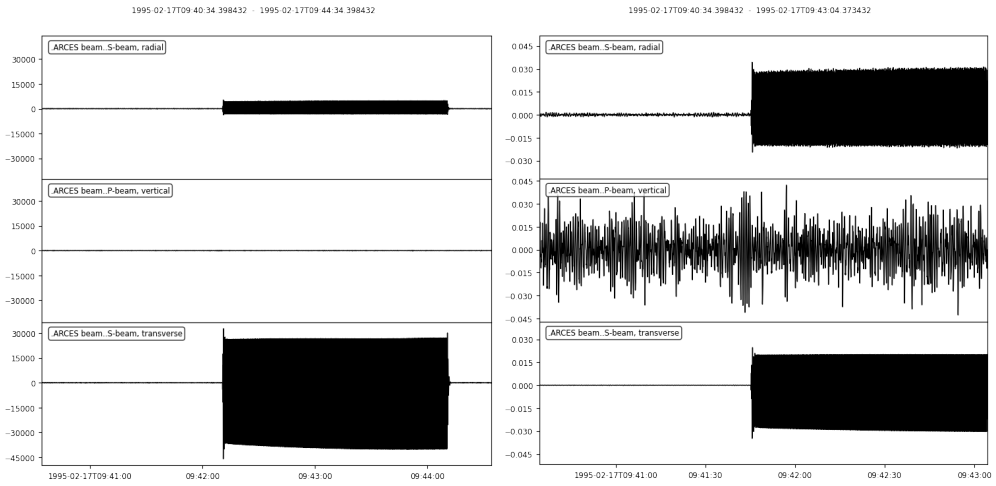
Figure A.4: Predicted output: 0.13, correct output: 1. Type: Explosion, magnitude: 2.2, distance: 1106 km. Likely mislabeled start time causes time augmentor to compromise the event of interest.

A.1.2 Selected False Negatives



(a) *(b)*

Figure A.5: Predicted output: 0.06, correct output: 1. Type: Explosion, magnitude: 0.9, distance: 119 km. Near but very low magnitude explosion event.

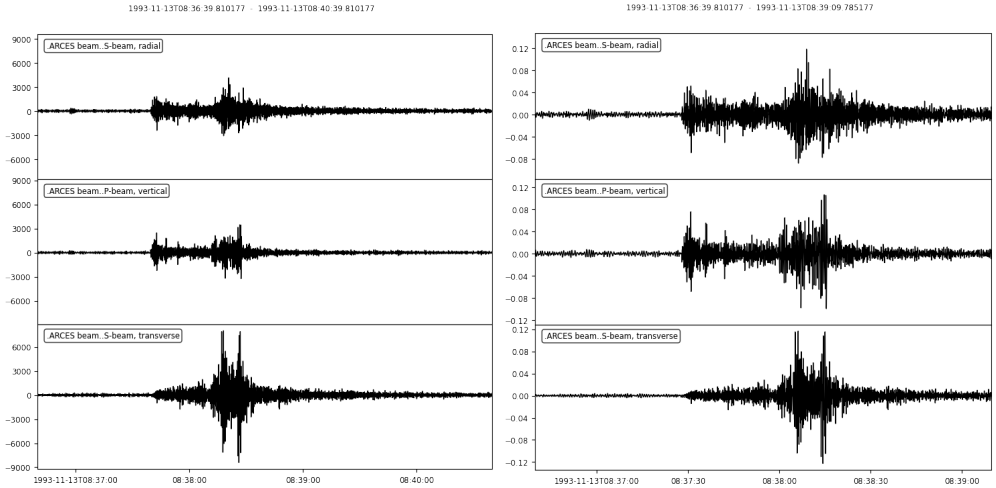


(a) *(b)*

Figure A.6: Predicted output: 0.06, correct output: 1. Type: Explosion, magnitude: 2.3, distance: 1148 km. Outlier event, hardware error. Good example of how the normalized scaler transforms the input waveform.

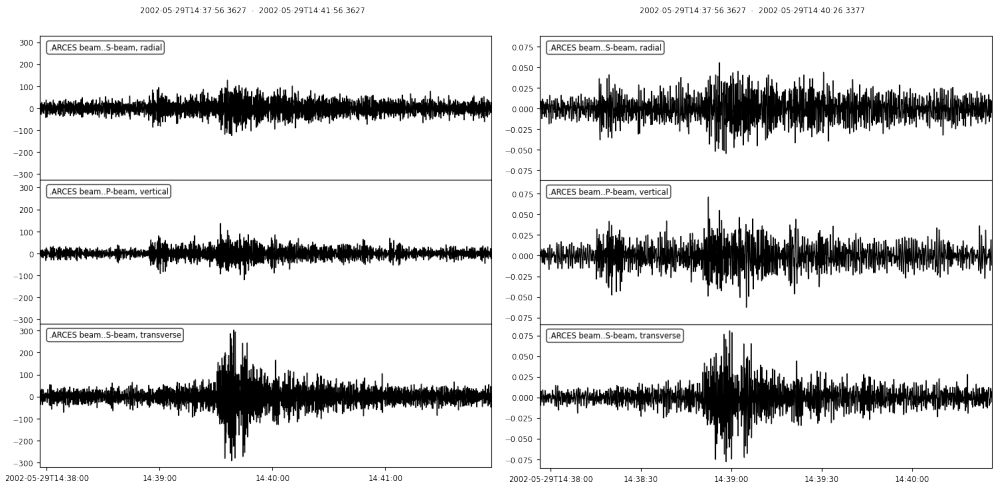
A.2 MLSTM-FCN EE

A.2.1 Selected False Positives



(a) *(b)*

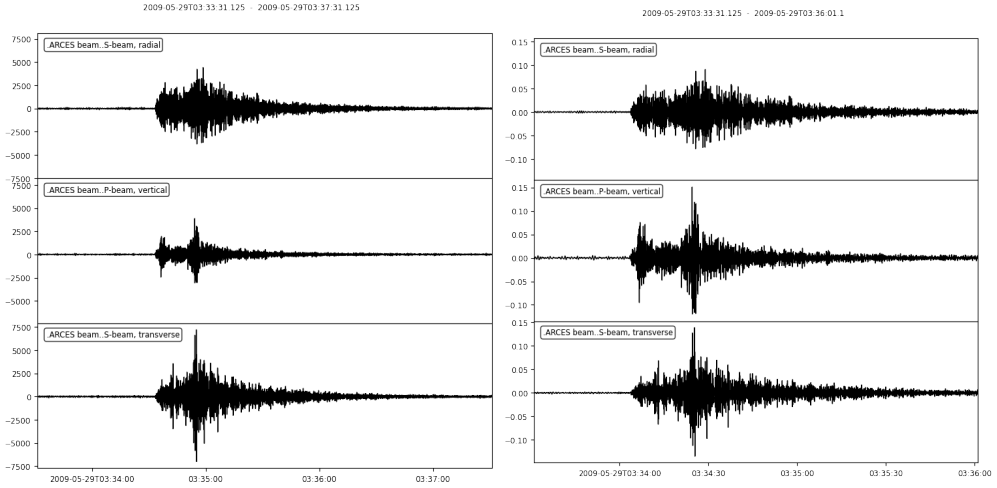
Figure A.7: Predicted output: 0.88, correct output: 0. Type: Explosion, magnitude: 2.1, distance: 297 km. This event is also speculated to be of ripple-fired explosions.



(a) *(b)*

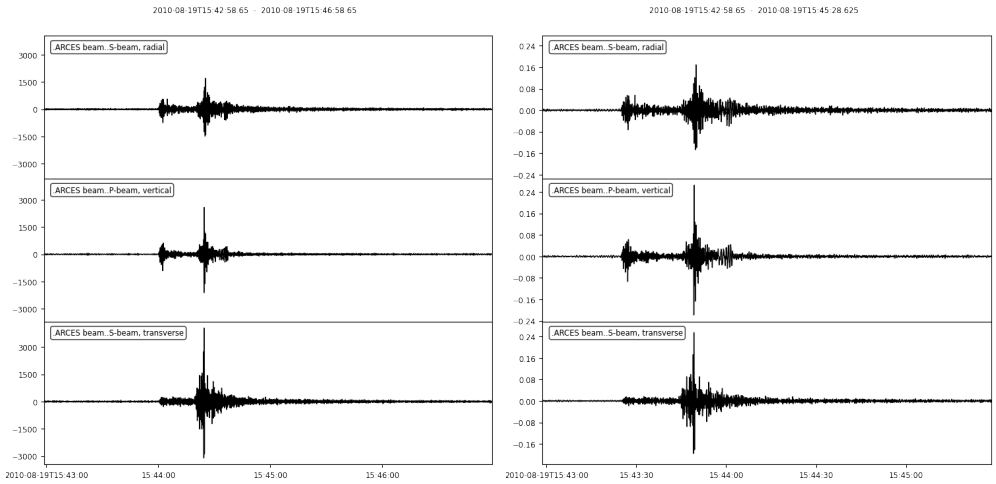
Figure A.8: Predicted output: 0.71, correct output: 0. Type: Explosion, magnitude: 2.0, distance: 312 km. Speculated to be recording of ripple-fired explosions, which should be relatively distinct from earthquakes. Neither weak nor distant. Notable mistake.

Selected False Negatives



(a) *(b)*

Figure A.9: Predicted output: 0.45, correct output: 1. Type: Earthquake, magnitude: 1.75, distance: 134 km. This event is of a near earthquake with good signal to noise ratio. Notable mistake.

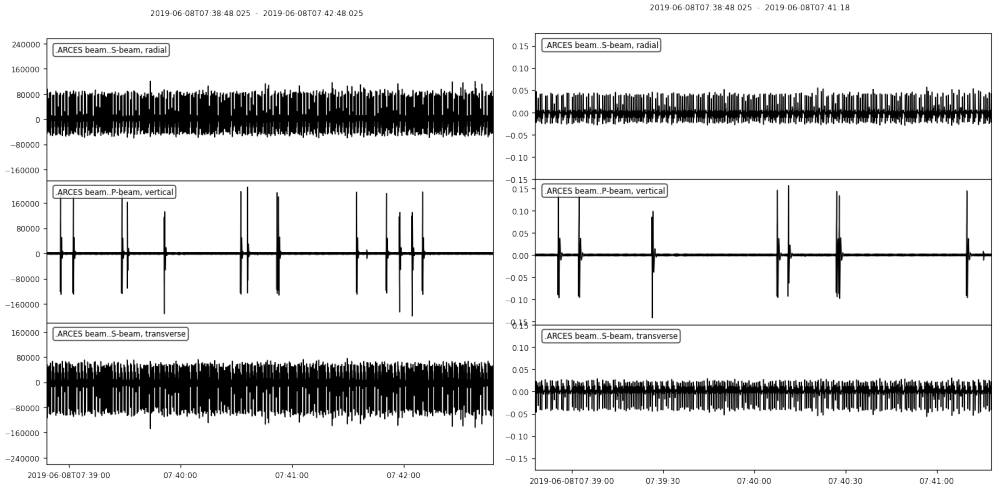


(a) *(b)*

Figure A.10: Predicted output: 0.33, correct output: 1. Type: Earthquake, magnitude: 1.85, distance: 163 km. This event is of another near earthquake with good signal to noise ratio.

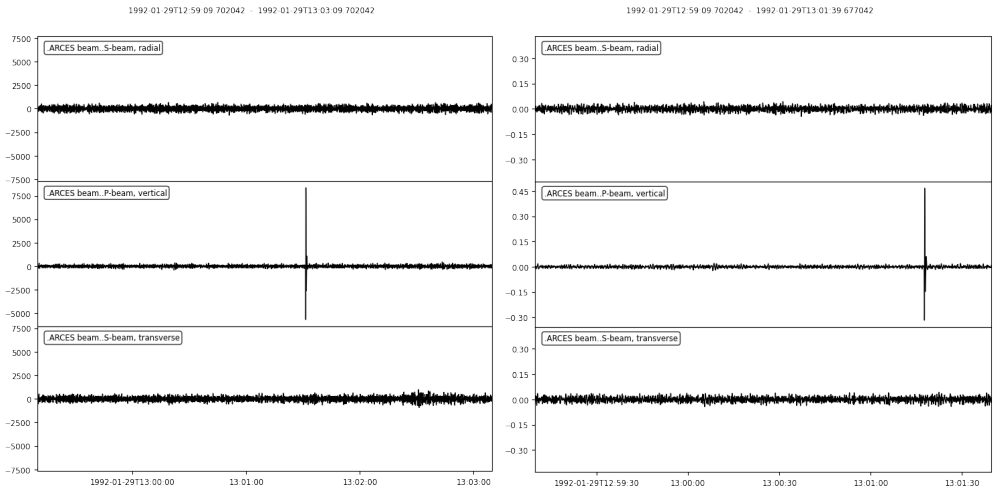
A.3 Meier inspired CNN 3N

A.3.1 Selected False Negatives



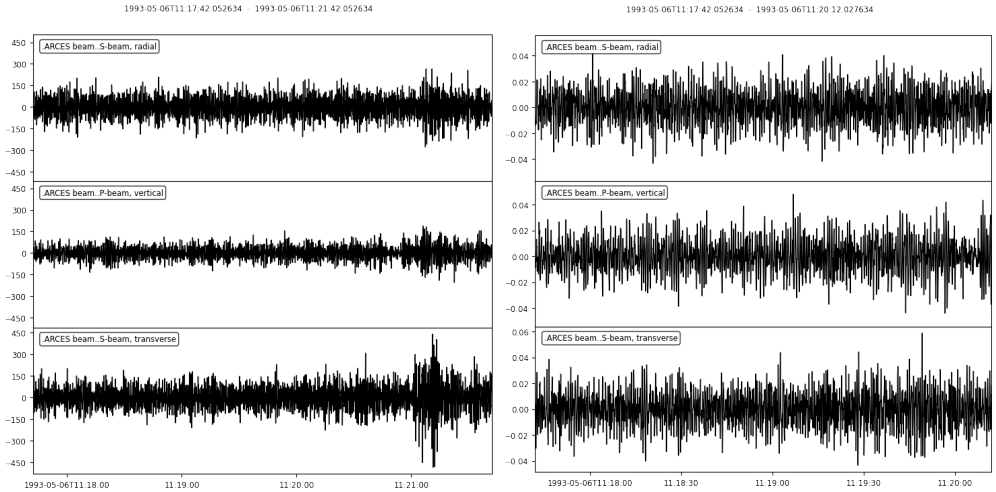
(a) *(b)*

Figure A.11: Predicted output: 0, correct output: 1. Type: Explosion, magnitude: 1.5, distance: 293 km. Hardware error. Unprecedented regular displacement in stacked counts.



(a) *(b)*

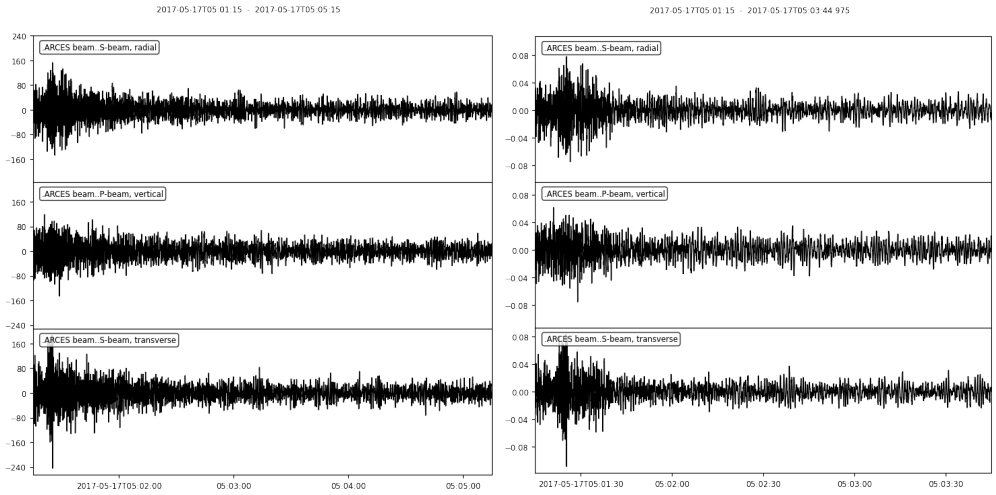
Figure A.12: Predicted output: 0, correct output: 1. Type: Explosion, magnitude: 2.3, distance: 1129 km. Sudden spike, likely unrelated to the event, obfuscating the event of interest.



(a) *(b)*

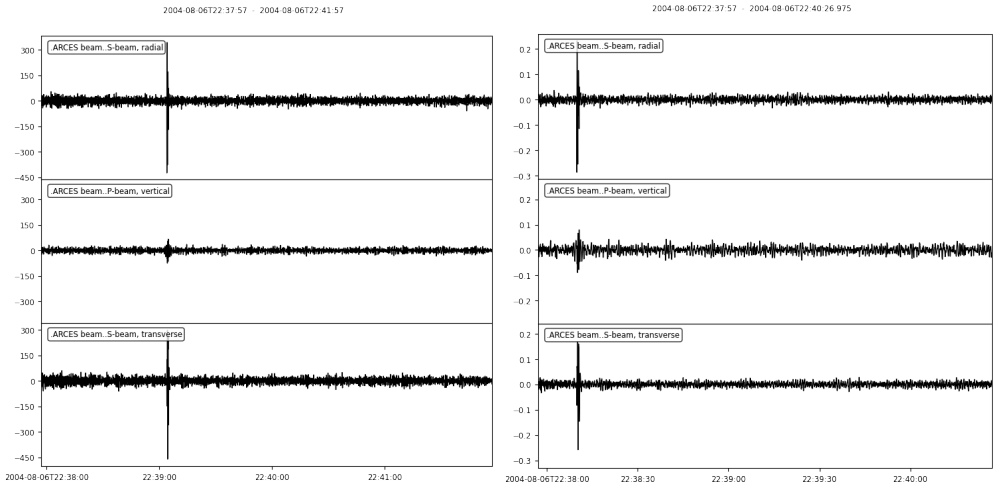
Figure A.13: Predicted output: 0.35, correct output: 1. Type: Explosion, magnitude: 2.4, distance: 1137 km. Either mislabeled, or slow traveling waves, causing time augmentor to cut out the event of interest.

A.3.2 Selected False Positives



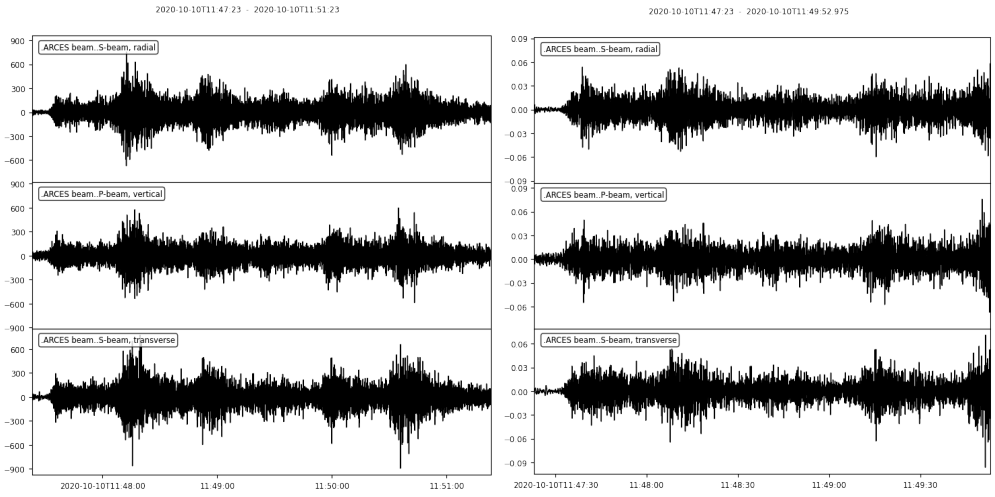
(a) *(b)*

Figure A.14: Predicted output: 1, correct output: 0. Noise. Labeled noise, although appears that the recording starts during an ongoing event.



(a) *(b)*

Figure A.15: Predicted output: 0.99, correct output: 0. Noise. Atypical expression in stacked counts.

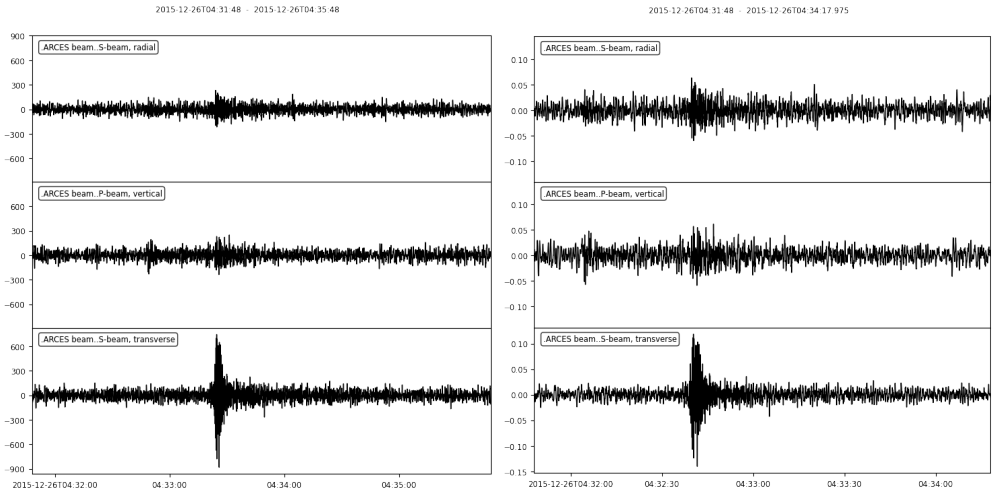


(a) *(b)*

Figure A.16: Predicted output: 0.99, correct output: 0. Noise. Irregular noise waveform.

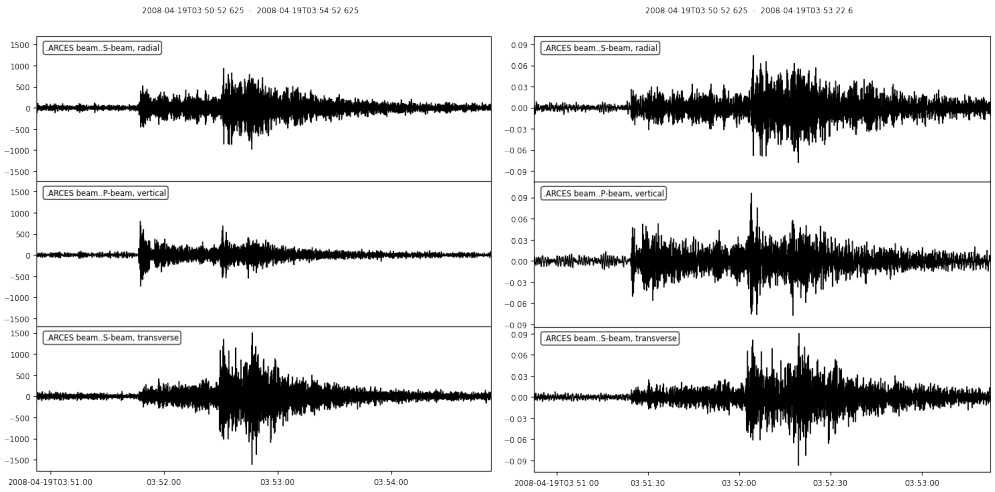
A.4 Meier inspired CNN EE

A.4.1 Selected False Negatives



(a) *(b)*

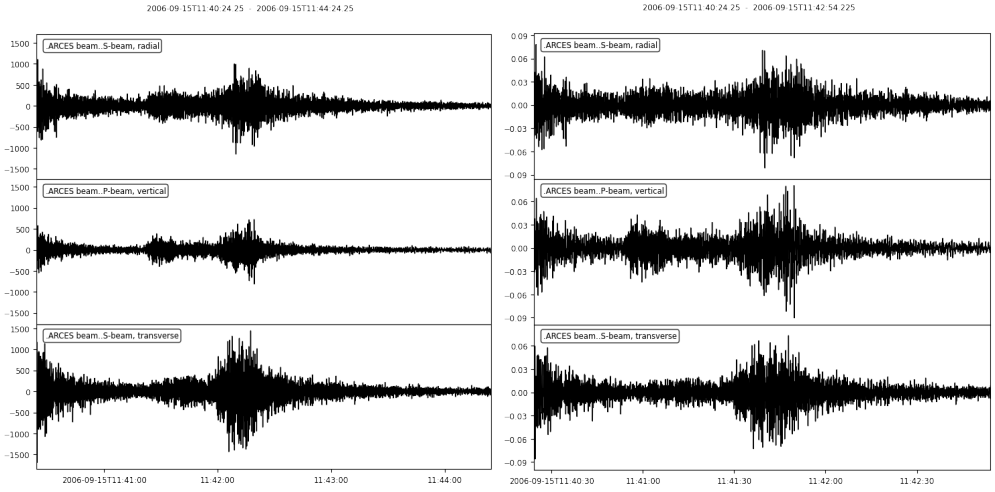
Figure A.17: Predicted output: 0, correct output: 1. Type: Earthquake, magnitude: 1.2, distance: 310 km. Event is eclipsed by noise, except short burst in the transverse channel.



(a) *(b)*

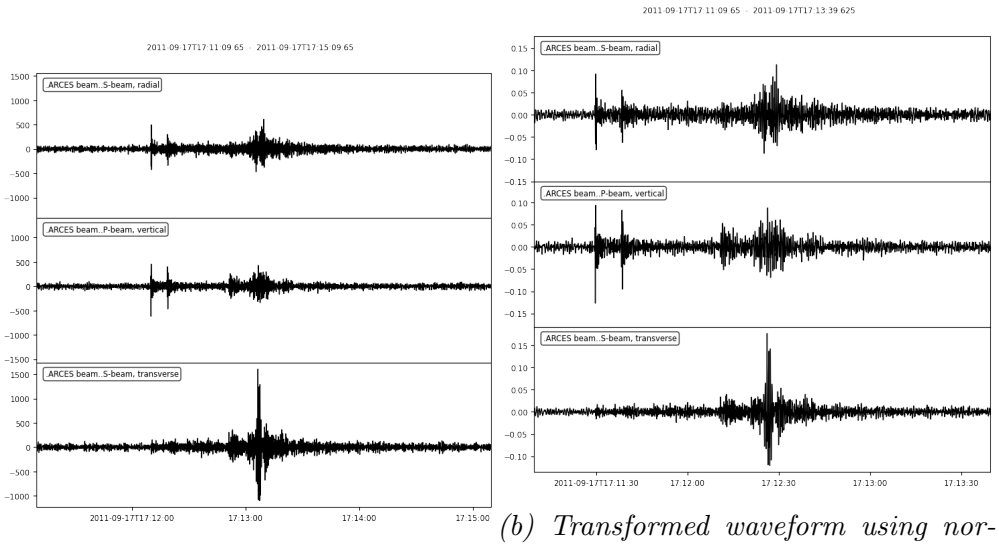
Figure A.18: Predicted output: 0, correct output: 1. Type: Earthquake, magnitude: 2.0, distance: 382 km. Inaccurately labeled start time, causing time augmentor to cut out the onset of the event.

A.4.2 Selected False Positives



(a) *(b)*

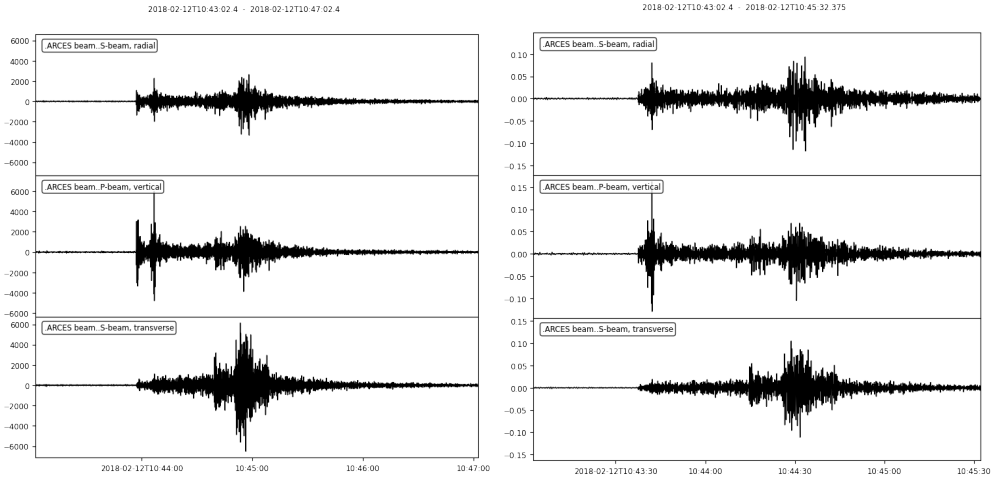
Figure A.19: Predicted output: 0.89, correct output: 0. Type: Explosion, magnitude: 2.0, distance: 340 km. Recording starts during the end of another event.



(a)

(b) Transformed waveform using normalize scaler, time augmentor and noise augmentor

Figure A.20: Predicted output: 0.99, correct output: 0. Type: Explosion, magnitude: 1.7, distance: 343 km. Striking resemblance to the waveform in A.34.

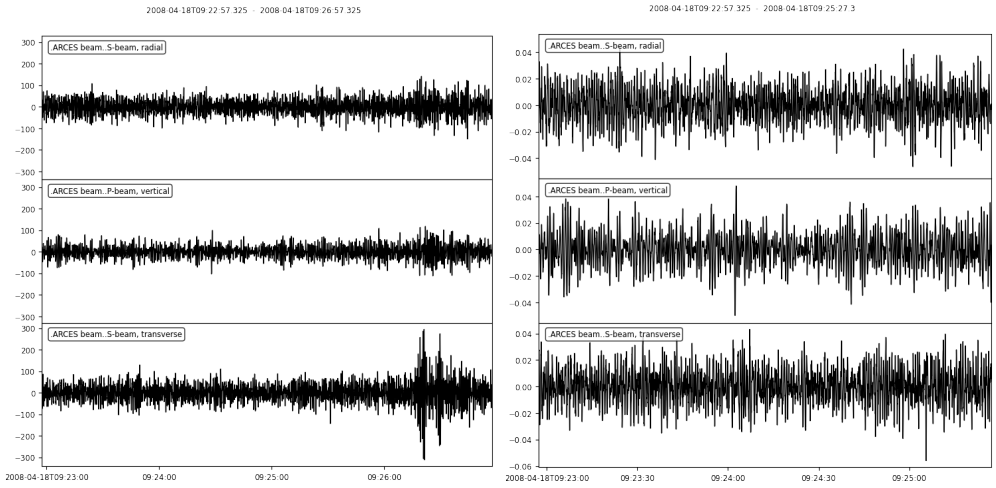


(a) *(b)*

Figure A.21: Predicted output: 1, correct output: 0. Type: Explosion, magnitude: 1.7, distance: 343 km. Inaccurate start time causing time augmentation to cut out the start of the event.

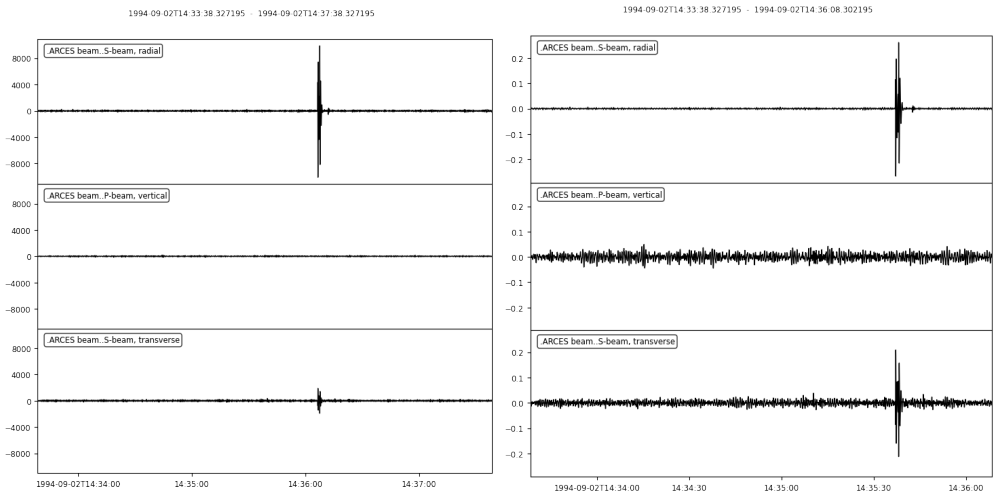
A.5 Self-developed CNN 3N

A.5.1 Selected False Negatives



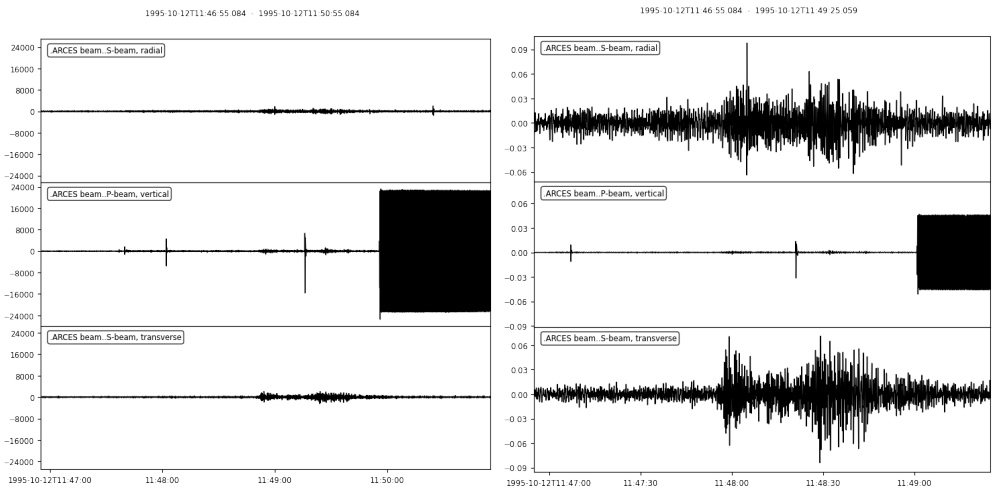
(a) *(b)*

Figure A.22: Predicted output: 0.49, correct output: 1. Distance to ARCES: 1142 km. Magnitude: 2.31. Earthquake.



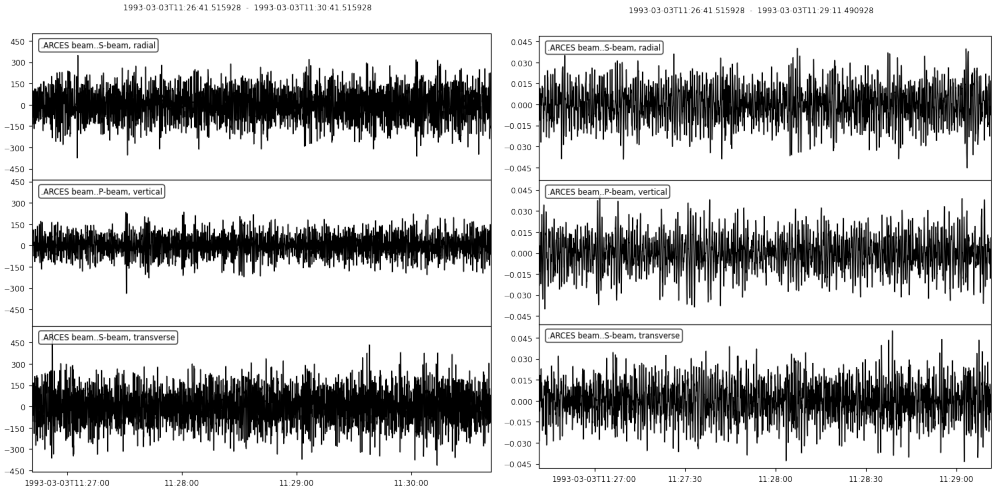
(a) *(b)*

Figure A.23: Predicted output: 0.17, correct output: 1. Distance to ARCES: 765 km. Magnitude: 1.7. Explosion



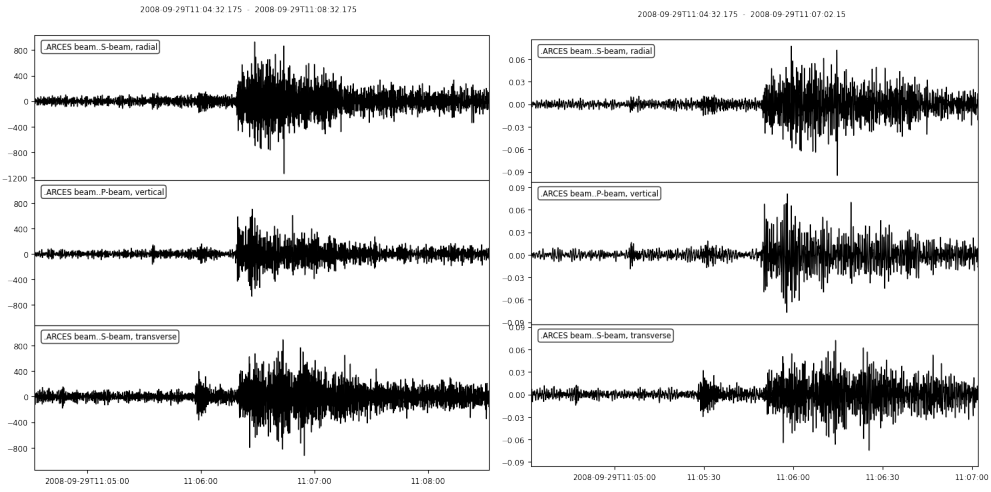
(a) *(b)*

Figure A.24: Predicted output: 0.03, correct output: 1. Distance to ARCES: 723 km. Magnitude: 1.8. Explosion. Hardware error.



(a) *(b)*

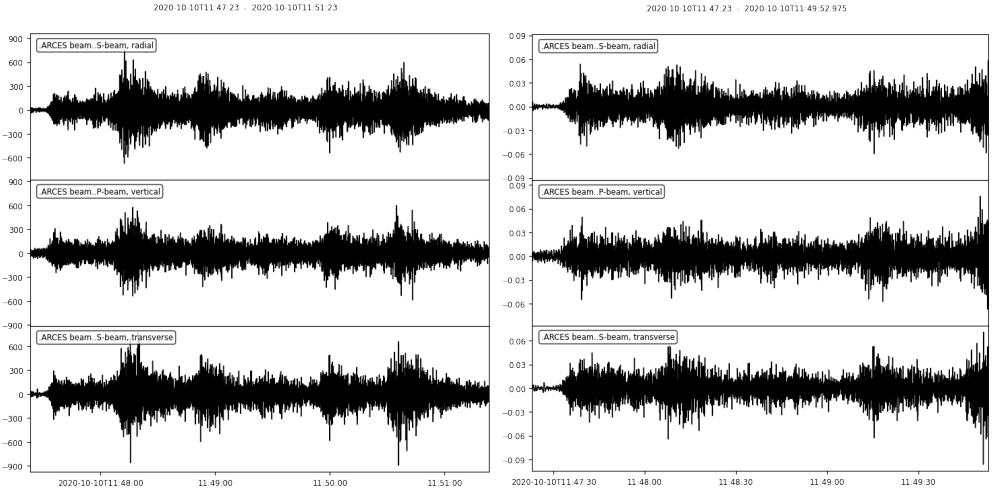
Figure A.25: Predicted output: 0.43, correct output: 1. Distance to ARCES: 1139 km. Magnitude: 2.2. Explosion. Hardware error.



(a) *(b)*

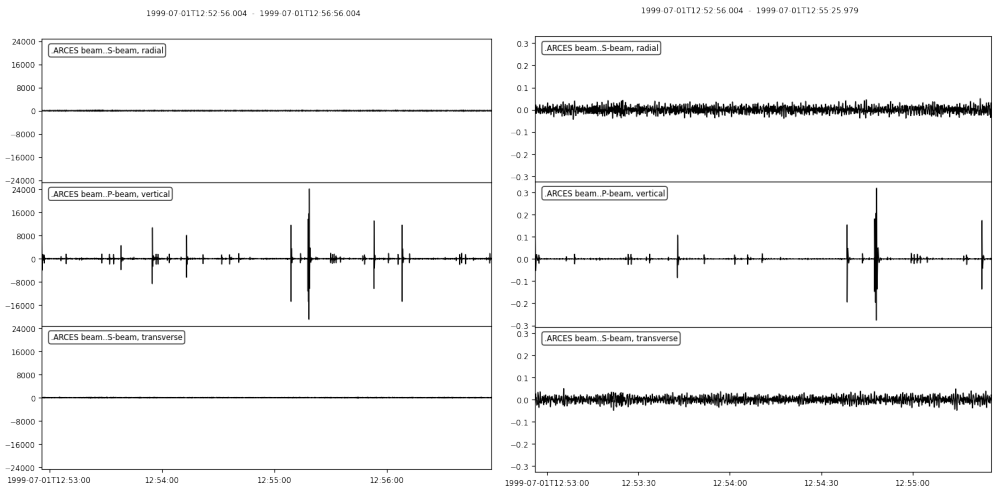
Figure A.26: Predicted output: 0.17, correct output: 1. Distance to ARCES: 205 km. Magnitude: 1.6. Explosion. Notable mistake.

A.5.2 Selected False Positives



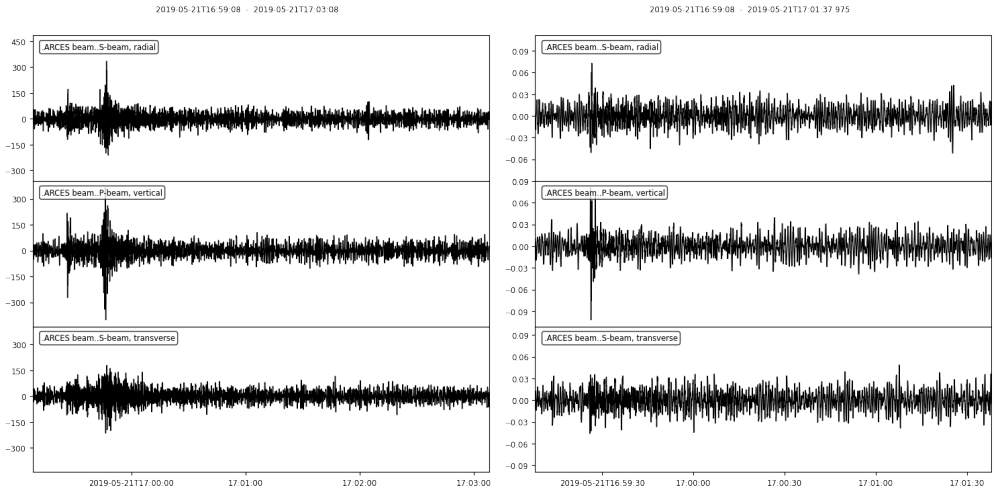
(a) *(b)*

Figure A.27: Predicted output: 0.96, correct output: 0. Noise. Irregular noise waveform. Same error made in A.16, although here with more uncertainty.



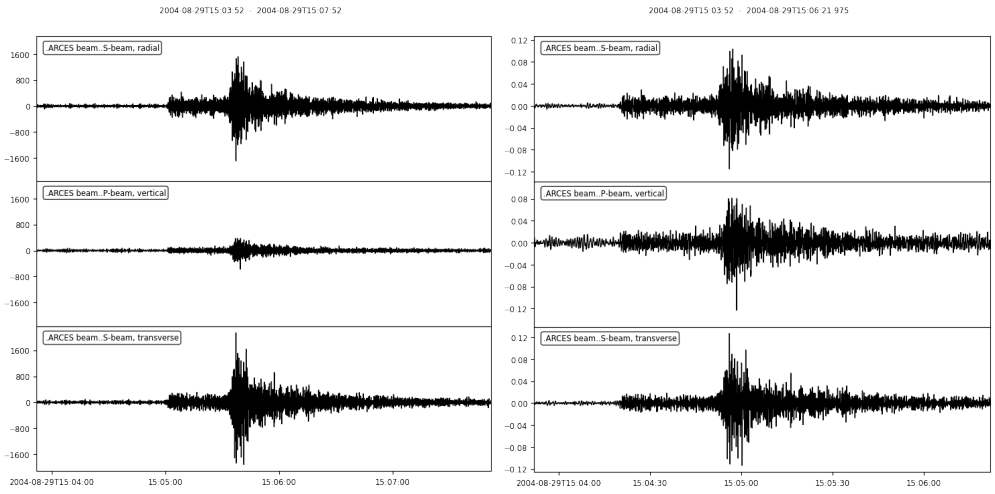
(a) *(b)*

Figure A.28: Predicted output: 0.62, correct output: 0. Noise. Hardware error.



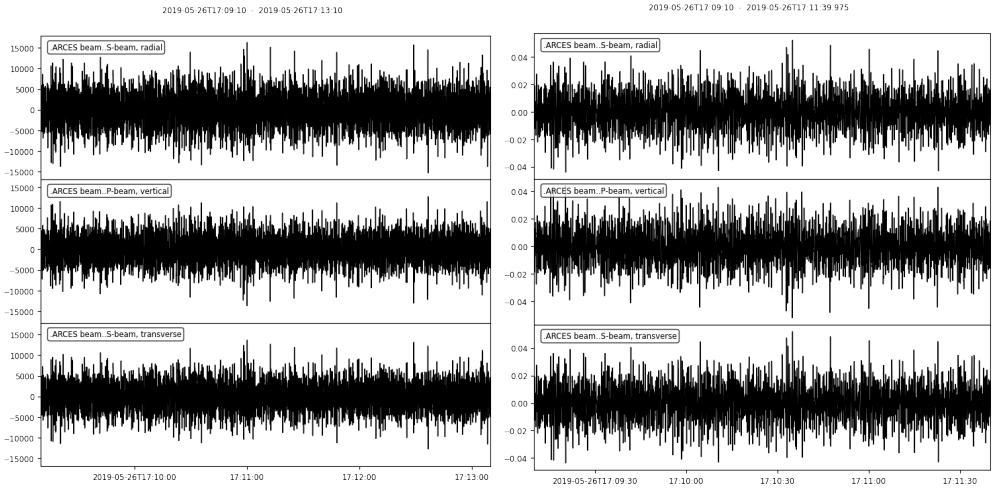
(a) *(b)*

Figure A.29: Predicted output: 0.63, correct output: 0. Noise. Seems to contain seismic event. Likely error stemming from the way the noise data is sampled. Seems to disregard everything prior to the labeled event start.



(a) *(b)*

Figure A.30: Predicted output: 0.93, correct output: 0. Noise. This waveform contains a seismic event, but is mislabeled. Contains the classic P and S wave structure.

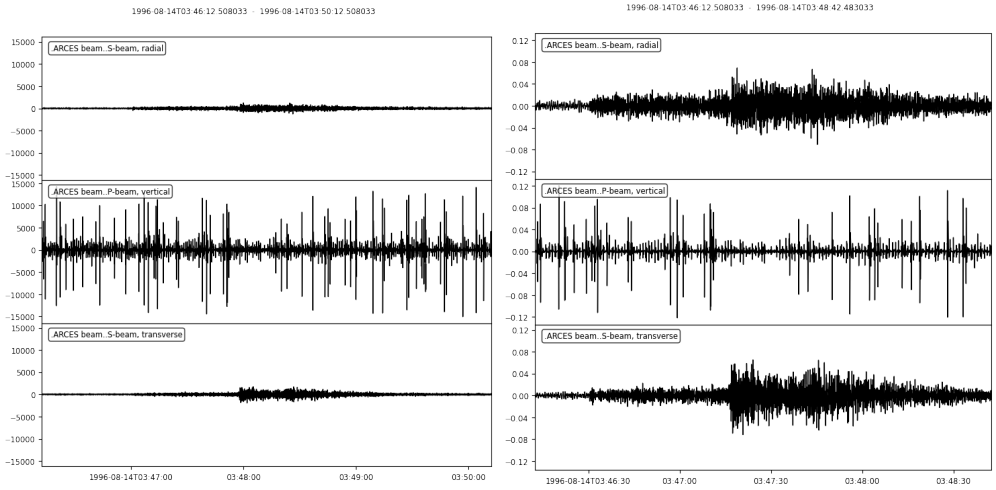


(a) *(b)*

Figure A.31: Predicted output: 0.51, correct output: 0. Noise. Outlier noise waveform with very strong displacement in all channels' stacked counts. Not a problem due to normalize scaler, but incorrect prediction nonetheless.

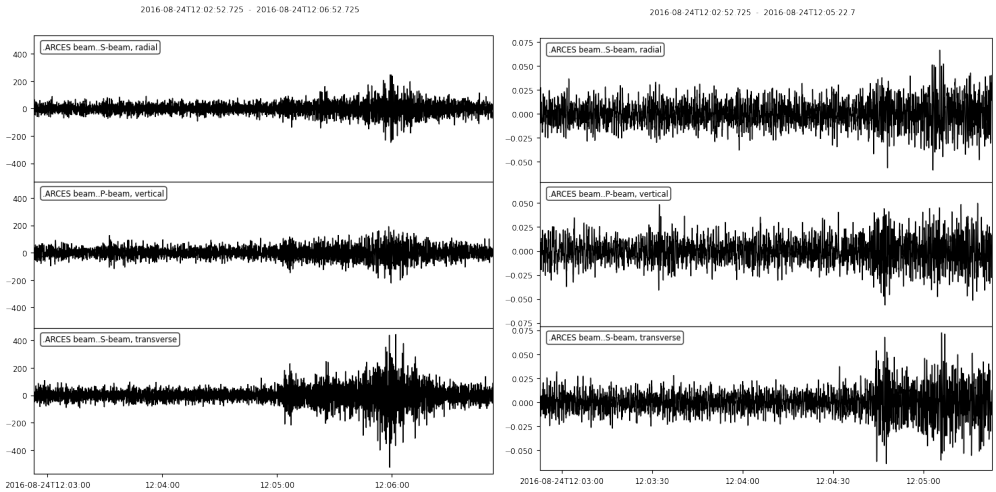
A.6 Self-developed CNN EE

A.6.1 Selected False Positives



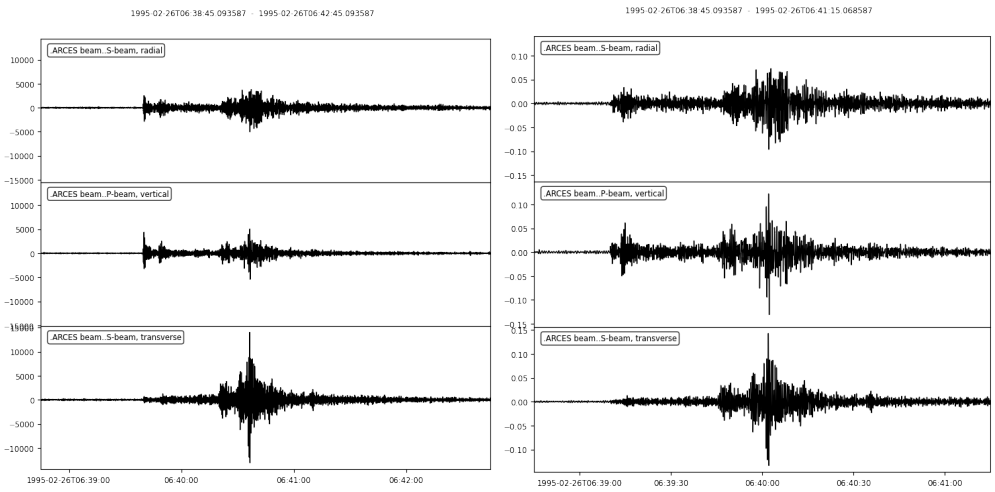
(a) *(b)*

Figure A.32: Predicted output: 0.87, correct output: 0. Type: Explosion, magnitude: 2.1, distance: 570 km. Outlier waveform, likely due to hardware error.



(a) *(b)*

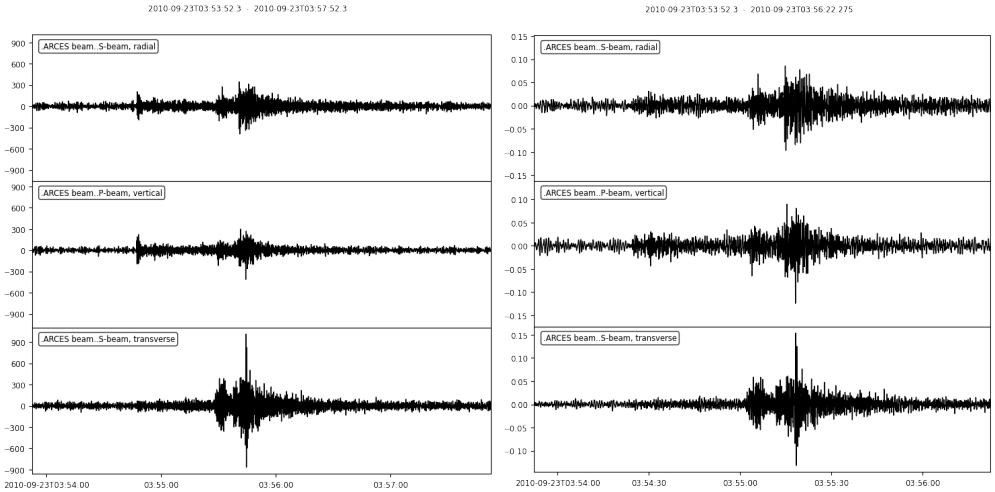
Figure A.33: Predicted output: 0.60, correct output: 0. Type: Explosion, magnitude: 2.2, distance: 921 km. Incorrectly labeled start time causes time augmentor to cut out significant portions of the event of interest



(a) *(b)*

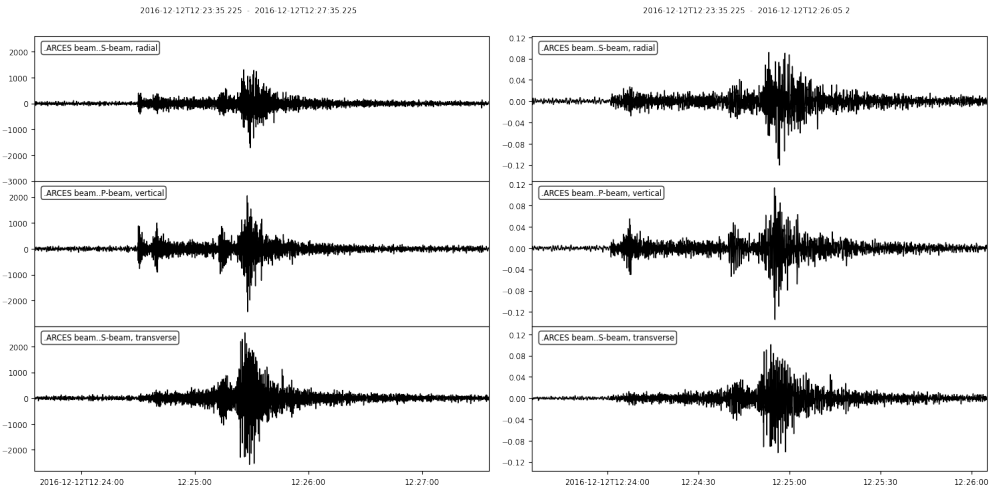
Figure A.34: Predicted output: 0.93, correct output: 0. Type: Explosion, magnitude: 3.0, distance: 392 km. Noisy waveform near the end of the raw waveform has been inserted right before the actual event. Additionally, the labeled time is inaccurate, causing time augmentor to cut out the start of the event.

A.6.2 Selected False Negatives



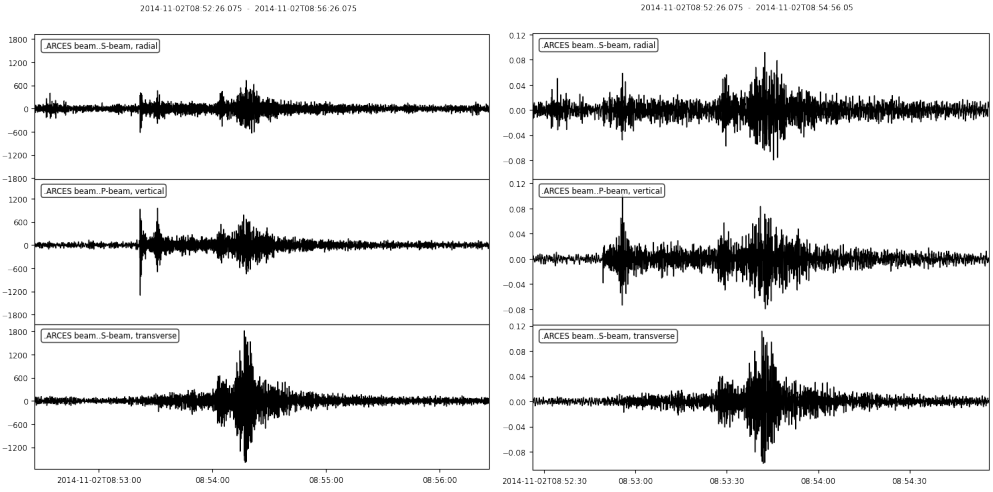
(a) *(b)*

Figure A.35: Predicted output: 0.18, correct output: 1. Type: Earthquake, magnitude: 2.1, distance: 404 km. The labeled start time is incorrect, causing time augmentor to cut out the start of the waveform.



(a) *(b)*

Figure A.36: Predicted output: 0.32, correct output: 1. Type: Earthquake, magnitude: 2.6, distance: 426 km. The labeled start time is incorrect, causing time augmentor to cut out the start of the waveform.

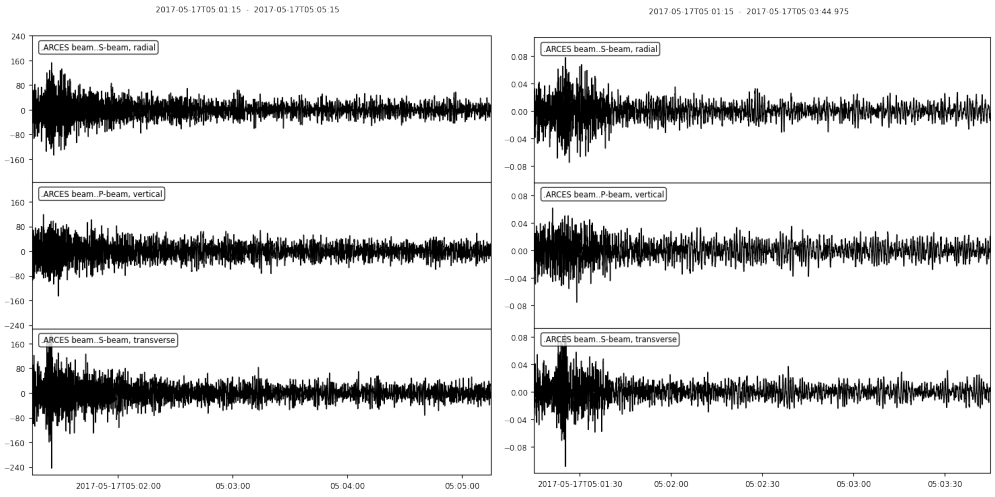


(a) *(b)*

Figure A.37: Predicted output: 0.33, correct output: 1. Type: Earthquake, magnitude: 2.2, distance: 393 km. The labeled start time is incorrect, causing time augmentor to cut out the start of the waveform.

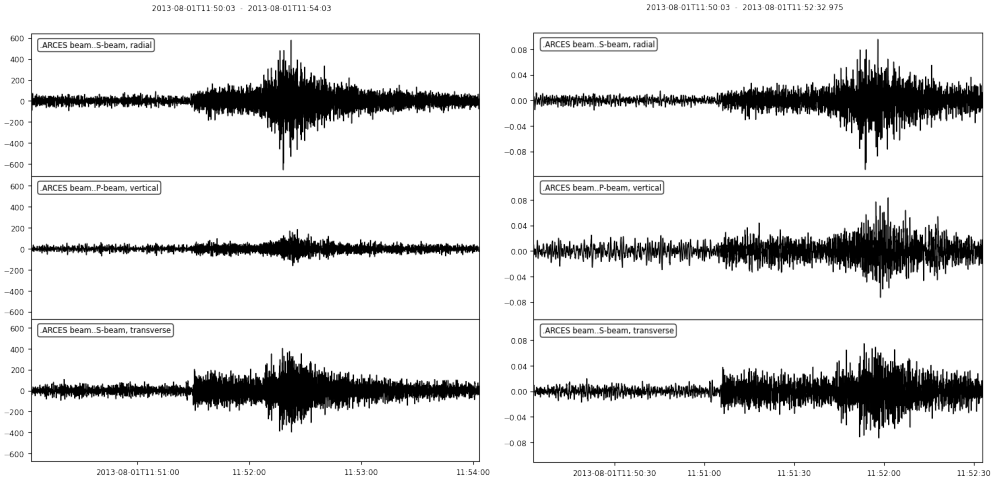
A.7 InceptionTime 3N

A.7.1 Selected False Positives



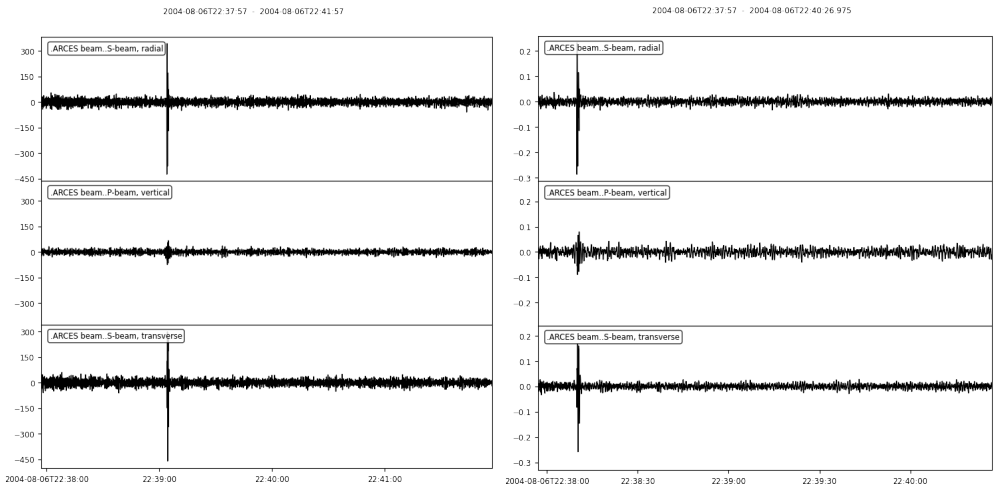
(a) *(b)*

Figure A.38: Predicted output: 0.95, correct output: 0. Type: Noise. The recording starts during an ongoing (presumably) seismic event, confusing the classifier. Shows insensitivity to location of event.



(a) *(b)*

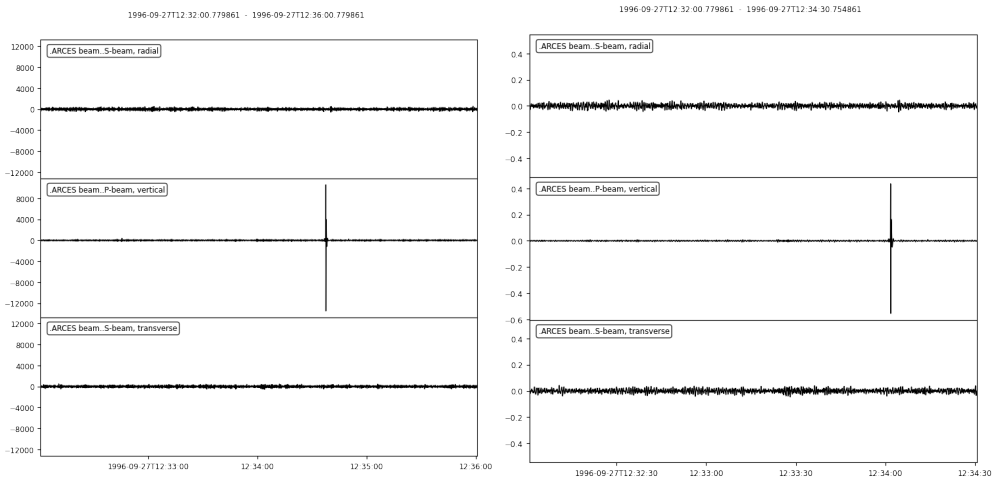
Figure A.39: Predicted output: 0.99, correct output: 0. Type: Noise. Mislabled event. Same as A.30.



(a) *(b)*

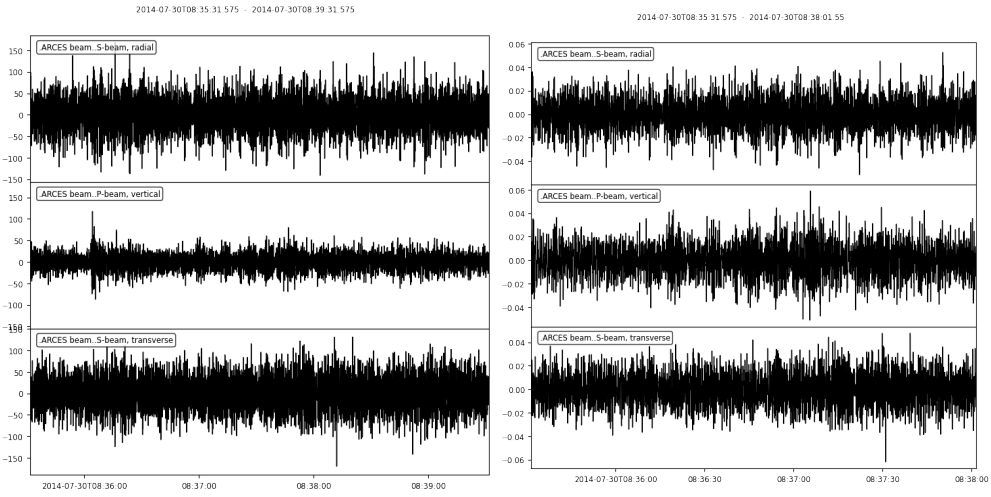
Figure A.40: Predicted output: 0.99, correct output: 0. Type: Noise. Short, but significant amplitude spike.

A.7.2 Selected False Negatives



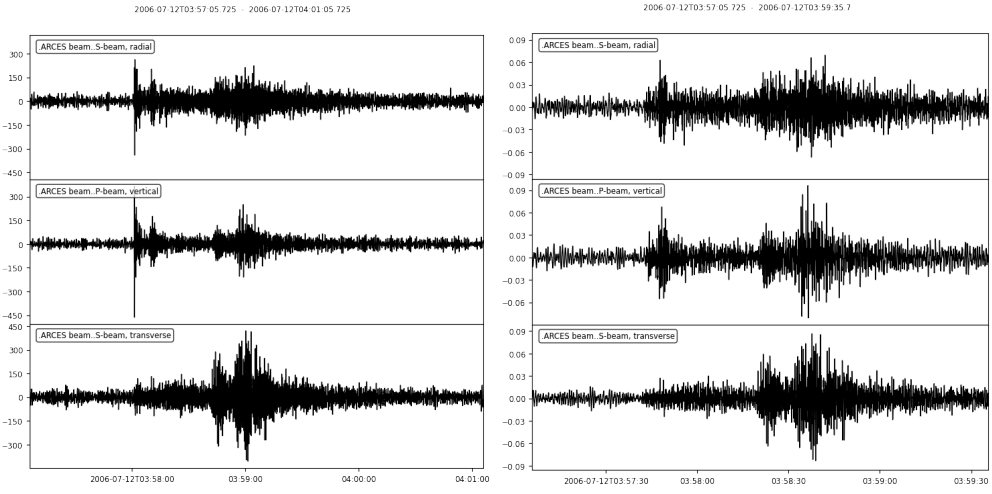
(a) *(b)*

Figure A.41: Predicted output: 0.34, correct output: 1. Type: Explosion, magnitude: 1.7, distance: 759 km. Very short but significant displacement in stacked counts in the vertical channel.



(a) *(b)*

Figure A.42: Predicted output: 0.22, correct output: 1. Type: Explosion, magnitude: 2.2, distance: 1106 km. Appears that the event of interest occur prior to the labeled start time, causing time augmentor to cut the event out of the input waveform.

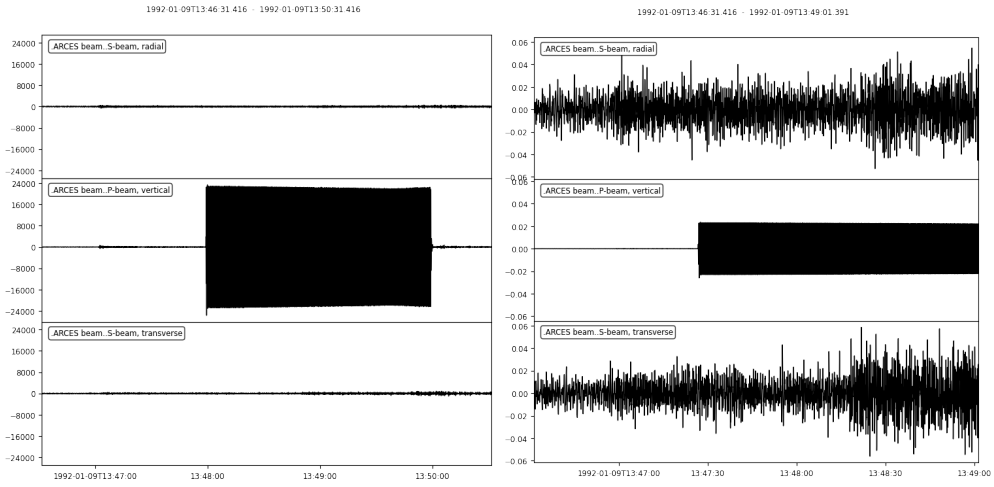


(a) *(b)*

Figure A.43: Predicted output: 0.44, correct output: 1. Type: Explosion, magnitude: 1.6, distance: 383 km. Incorrectly labeled start time. Still, the event is, at least visually, clearly a seismic event. Notable mistake.

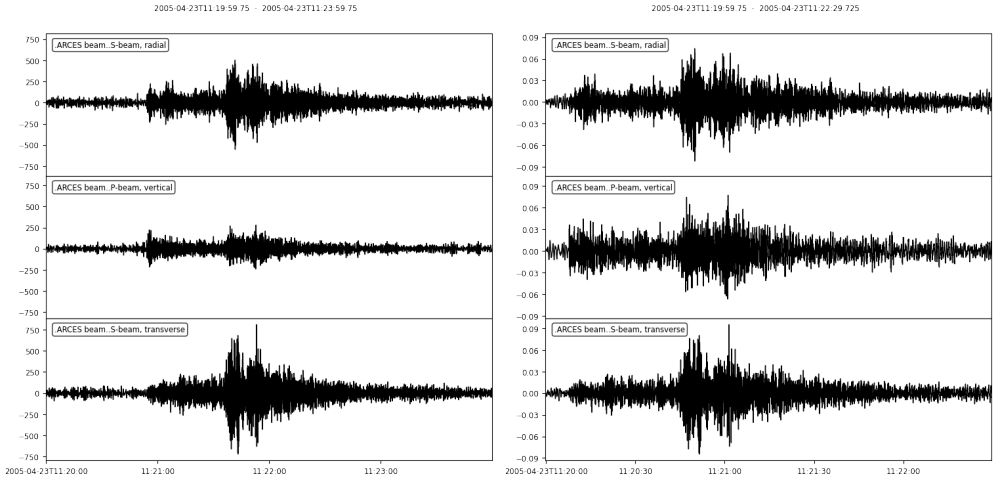
A.8 InceptionTime EE

A.8.1 Selected False Positives



(a) *(b)*

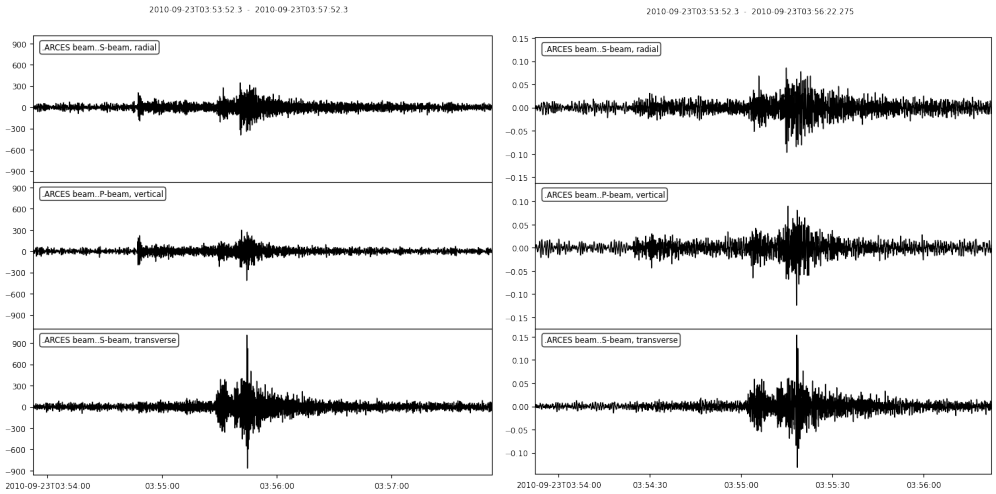
Figure A.44: Predicted output: 0.67, correct output: 0. Type: Explosion, magnitude: 2.4, distance: 1140 km. Hardware error.



(a) *(b)*

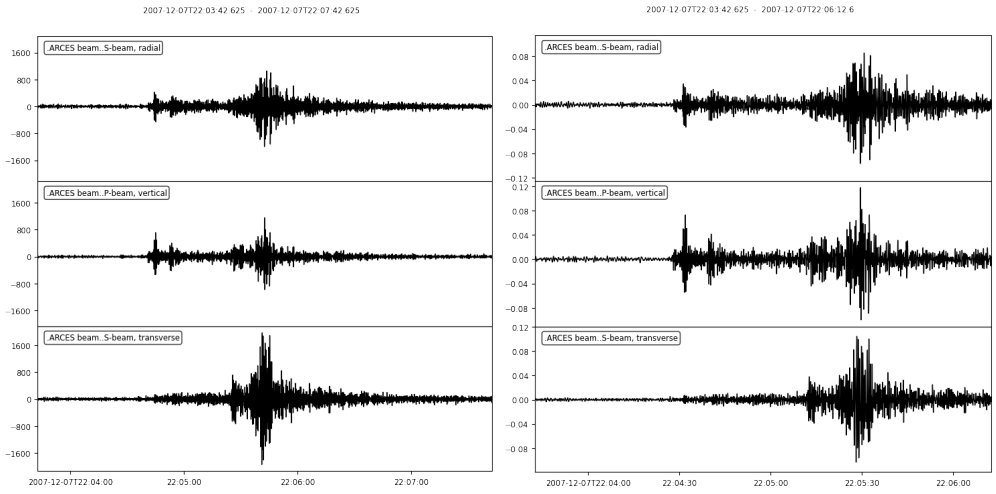
Figure A.45: Predicted output: 0.99, correct output: 0. Type: Explosion, magnitude: 1.6, distance: 409 km. The start time of this event is mislabeled, causing time augmentor to cut out the start of the event.

A.8.2 Selected False Negatives



(a) *(b)*

Figure A.46: Predicted output: 3e-8, correct output: 1. Type: Earthquake, magnitude: 2.4, distance: 404 km. The start time of this event is mislabeled, causing time augmentor to cut out the start of the event.

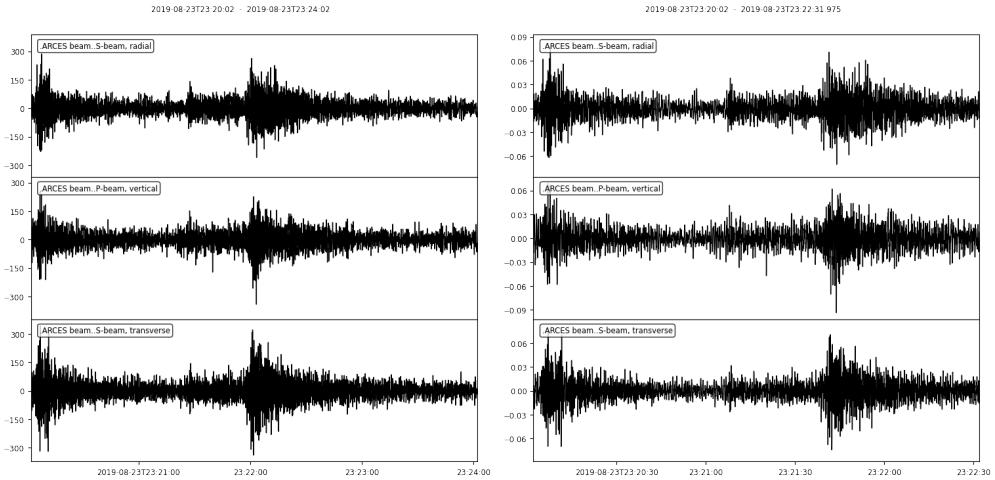


(a) *(b)*

Figure A.47: Predicted output: 2.7e-5, correct output: 1. Type: Earthquake, magnitude: 2.5, distance: 432 km. Relatively clear event, with high signal to noise ratio, not negatively affected by augmentation. Notable mistake.

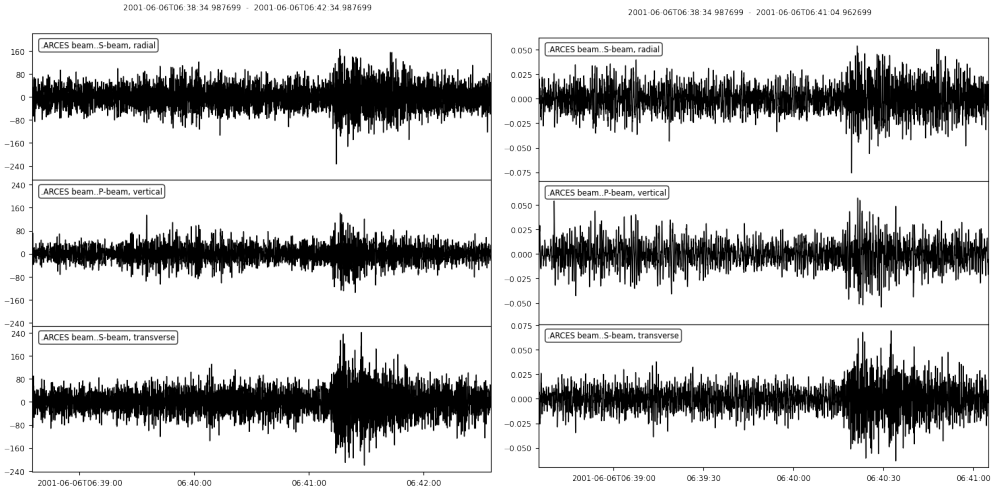
A.9 Final Model

A.9.1 Noise Events Incorrectly Classified As Earthquakes



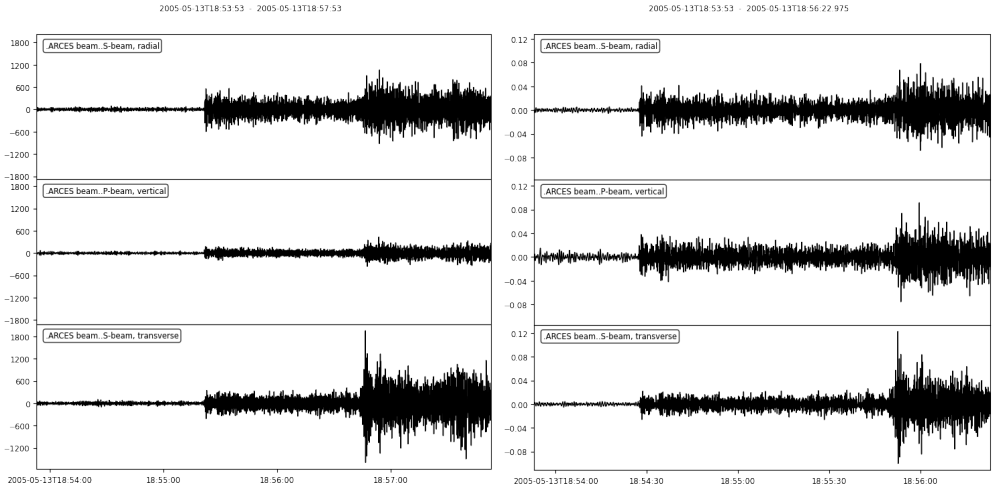
(a) *(b)*

Figure A.48: 3N prediction: 0.97, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. The waveform deviates from the typical noise recording.



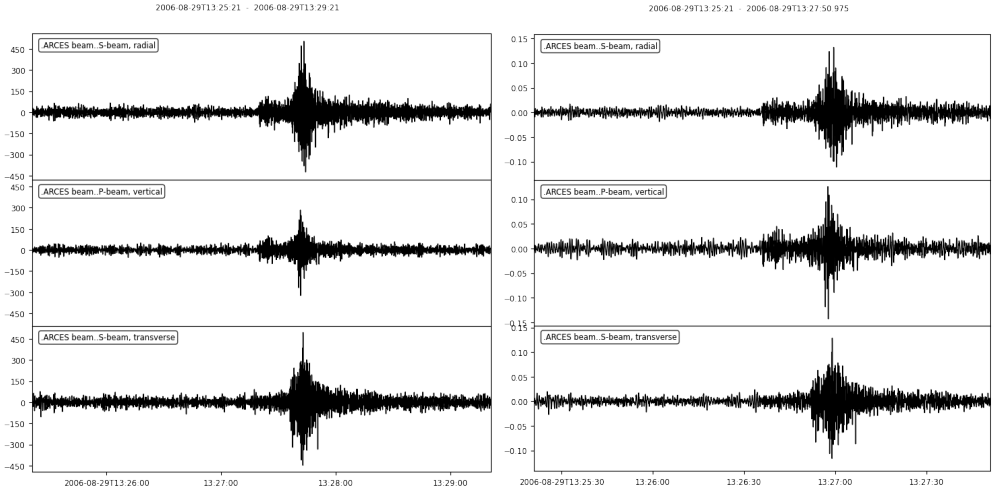
(a) *(b)*

Figure A.49: 3N prediction: 0.97, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. The waveform deviates from the typical noise recording.



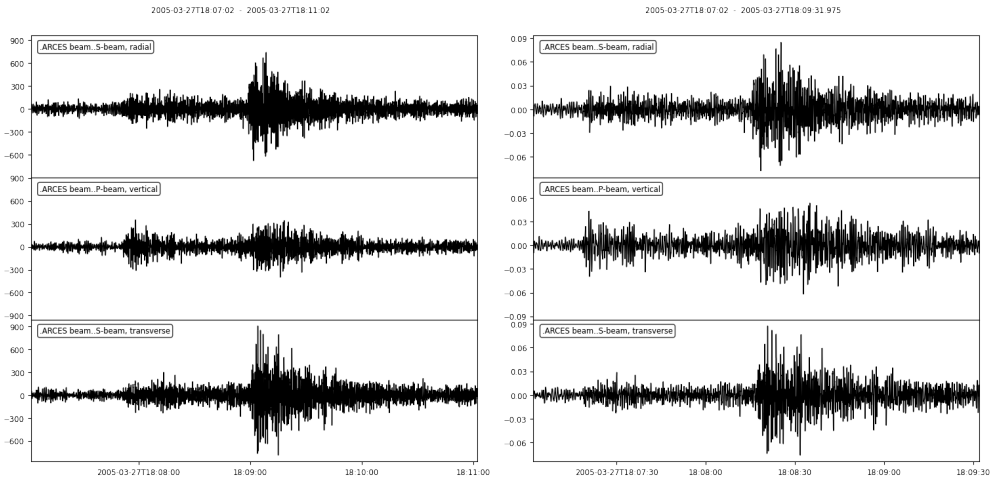
(a) *(b)*

Figure A.50: 3N prediction: 0.99, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. This event has much stronger displacement in stacked counts than typical noise recordings. It also appears to contain a clear P and S wave.



(a) *(b)*

Figure A.51: 3N prediction: 0.97, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. Short, but distinct segment deviating from the norm.



(a) *(b)*

Figure A.52: 3N prediction: 0.99, EE prediction: 0.99. Waveform label: Noise. Resulting prediction: Earthquake. Another event predicted to not be noise with very high certainty and an earthquake event with very high certainty.