

Whale call detection with deep neural networks

E Fourie

 [orcid.org/ 0000-0002-0150-1531](https://orcid.org/0000-0002-0150-1531)

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor: Prof MH Davel
Co-supervisor: Mr DJJ Versfeld

Graduation: April 2024

Acknowledgements

This research was conducted within MUST Deep Learning, which is a research group within the Faculty of Engineering at North-West University, Potchefstroom. This study was supervised by Professor Marelie Davel and co-supervised by Professor Jaco Versveld from Stellenbosch University. It has been an enriching experience and a privilege to work both with MUST and with Professor Jaco Versveld.

I would like to express my gratitude to the following people:

- Prof. Marelie Davel, my supervisor, to whom I want to take a moment to express my heartfelt appreciation for your guidance, support, care, and mentorship during my time working under your supervision.
- Ulrike Janke, the project manager at MUST, I would like to thank you for keeping my head above water, always being there and, above all, for being patient with me.
- Prof. Jaco Versveld, my co-supervisor, for all the positive feedback and suggestions.
- Centre of High Performance Computing (CHPC), South Africa, for letting me use their resources.
- My fellow students at MUST for creating a great working environment, and for all the support and good times.
- I would also like to thank all my friends for making my time studying much more challenging, but also for making my life in all other aspects better.

Lastly, there are actually not enough words of appreciation for my family; all I can say is “thank you”.

Abstract

Passive acoustic monitoring with hydrophones makes it possible to detect the presence of marine animals over large areas. For monitoring to be cost-effective, this process should be fully automated. We explore a new approach to detecting whale calls, using an end-to-end neural architecture and traditional speech features. We compare the results of the new approach with a Convolutional Neural Network (CNN) applied to spectrograms, currently the standard approach to whale call detection. Experiments are conducted using the “Acoustic trends for the blue and fin whale library” from the Australian Antarctic Data Centre (AADC).

We experiment with different types of speech features (Mel Frequency Cepstral Coefficients (MFCC) and Filter banks (Fbanks)) and different ways of framing the task. We demonstrate that a time-delay neural network is a viable solution for whale call detection, with the additional benefit that spectrogram tuning – required to obtain high-quality spectrograms in challenging acoustic conditions – is no longer necessary.

Keywords: *convolutional neural networks, time-delay neural networks, spectrograms, mel frequency cepstral coefficients, filter banks*

Contents

List of Figures	xi
List of Tables	xvi
List of Acronyms	xx
1 Introduction	1
1.1 Background	1
1.2 Problem statement	3
1.3 Project scope	4
1.3.1 Data sets	4
1.3.2 Architectures	5
1.3.3 Toolkits	5
1.4 Research questions	6
1.5 Objectives	6
1.6 Research methodology	7
1.7 Dissertation overview	7
1.8 Publications	8
2 Background	9
2.1 Overview	9

2.2	Whale vocalisation	10
2.3	Related work	11
2.3.1	Right whale call detection	13
2.3.2	Long-finned and killer whale whistle detection and classification . .	14
2.3.3	Sperm whale calls detection	15
2.3.4	ORCA-SPOT	17
2.4	Feature extraction	19
2.4.1	Spectrograms	19
2.4.2	Filter banks	22
2.4.3	Mel frequency cepstral coefficient (MFCCs)	22
2.5	Deep neural networks	28
2.5.1	Convolutional neural network	28
2.5.2	Neural speech processing	30
2.5.3	SpeechBrain	37
2.6	Training of Deep Neural Networks (DNNs)	38
2.6.1	Gradient descent	39
2.6.2	Parameters	40
2.6.3	Hyperparameters	40
2.7	Optimisation tools	44
2.7.1	Automated hyperparameter tuning	45
2.7.2	Optuna	45
2.7.3	Orion	47
2.8	Evaluation metrics	47
2.8.1	Accuracy	48
2.8.2	Confusion Matrix	48

2.8.3	Precision and recall	49
2.8.4	F1-score	50
2.9	Conclusion	50
3	Data set and preprocessing	51
3.1	Overview	51
3.2	AADC corpus	52
3.2.1	Statistics on data set	55
3.2.2	Problems encountered in corpus	58
3.2.3	Subset selection of AADC	60
3.3	Preprocessing	61
3.3.1	Preprocessing for both methods	62
3.3.2	Additional preprocessing for CNN	65
3.3.3	Additional preprocessing for SpeechBrain	65
3.4	Data sets for experimentation	66
3.5	Discussion	67
4	CNN baseline	68
4.1	Overview	68
4.2	CNN architecture	69
4.3	Optimisation protocol	71
4.4	Spectrogram extraction	72
4.4.1	Raven Pro	72
4.4.2	Sonic Visualiser	78
4.4.3	Python	80
4.5	Analysis and Results	82

4.5.1	Stochastic Gradient Descent (SGD)-4	83
4.5.2	Adam-4	85
4.5.3	SGD-2	86
4.5.4	Adam-2	88
4.5.5	Discussion	90
4.6	Conclusion	91
5	Whale call detection using SpeechBrain	93
5.1	Overview	93
5.2	SpeechBrain	94
5.2.1	SpeechBrain’s pipeline	94
5.2.2	Adaptations made to SpeechBrain’s speaker identification for whale call detection	96
5.3	Out-of-the-box Time Delay Neural Network (TDNN)	100
5.3.1	Experimental setup	100
5.3.2	Results	101
5.3.3	Discussion	101
5.4	Framing whale call detection as different machine learning tasks	102
5.5	Experimental setup	103
5.6	Optimisation protocol	105
5.7	Analysis and Results	106
5.7.1	Small_3_cs data set	106
5.7.2	Large_3_4 data set	124
5.7.3	Conclusion	130
5.8	TDNN vs baseline CNN	131
5.9	Performance across sites	134

5.9.1	Data Set	134
5.9.2	Approach	134
5.9.3	Analysis and Results	135
5.9.4	Conclusion	135
5.10	Conclusion	135
6	Conclusion	136
6.1	Introduction	136
6.2	Key findings	137
6.3	Contributions	138
6.4	Future work	139
6.5	Conclusion	139
	References	141
A	Supplemental content	155
A.1	Appendix: Chapter 2	155
A.2	Appendix: Chapter 4	158
A.2.1	SGD-4	158
A.2.2	Adam-4	159
A.2.3	SGD-2	161
A.2.4	Adam-2	162
A.3	Appendix: Chapter 5	164
A.4	Small_3_cs	164
A.4.1	Fbanks-2	164
A.4.2	MFCCs-2	170
A.4.3	Fbanks-4	177

A.4.4 MFCCs-4	182
A.5 Appendix: Publication	189

List of Figures

2.1	Representation of the LeNet convolutional neural network. <i>Adapted from Shui et al. [5].</i>	14
2.2	LeNet CNN architecture used for long-finned and killer whale whistle identification and classification.	15
2.3	The architecture of the CNN-based echolocation click detector [8].	16
2.4	Model accuracies for common ResNet architectures by comparing architectures with and without max pooling (3×3 kernel, stride 2) in the first residual layer. <i>Adapted from Bergler et al. [7].</i>	18
2.5	ORCA-SPOT network architecture. <i>Adapted from Bergler et al. [7].</i> . . .	18
2.6	Fbanks on a Mel-Scale [45].	26
2.7	Visual illustration example of how no dilation is implemented.	32
2.8	Visual illustration example of how dilation is implemented.	34
2.9	The TDNN unit that is used in this study. Each layer has its kernel size (k) and delay (d).	37
2.10	Confusion Matrix representation for binary class.	49
3.1	Distribution of all the call durations from the AADC data set.	56
3.2	Distribution of blue and fin whale call duration from the AADC data set. .	57
3.3	Distribution of blue and fin whale call durations from three selected sites in the AADC data set.	61
3.4	Comparison between 1,000 Hz and 300 Hz of 20150115_170000.wav to see if the call is still within timestamp.	64

4.1	The architecture of the baseline CNN used in this study.	70
4.2	AADC spectrogram settings in Raven Pro.	73
4.3	AADC spectrogram results in Raven Pro.	74
4.4	<i>Floor</i> and <i>ceiling</i> settings for replication in Raven Pro.	75
4.5	Spectrogram settings for replication in Raven Pro.	76
4.6	Spectrogram result after replicating the settings in Raven Pro.	77
4.7	Spectrogram settings in Sonic Visualiser.	78
4.8	Spectrogram result obtained from settings in Sonic Visualiser.	79
4.9	Spectrogram obtained in Python with the settings from Raven Pro.	81
4.10	Confusion matrix for hyperparameter optimisation using Optuna with SGD as an optimiser for the four-second chunks on seed 42.	84
4.11	Confusion matrix for hyperparameter optimisation using Optuna with Adam as an optimiser for the four-second chunks on seed 42.	86
4.12	Confusion matrix for hyperparameter optimisation using Optuna with SGD as an optimiser for two-second chunks on seed 185.	88
4.13	Confusion matrix for hyperparameter optimisation using Optuna with Adam as an optimiser for the two-second chunks on seed 185.	90
5.1	The training data manifest for small_3_4 data set for Machine Learning (ML) task blue-fin-all-else.	97
A.1	Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 42 for chunks size of 4.	158
A.2	Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 185 for chunks size of 4.	159
A.3	Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 1986 for chunks size of 4.	159
A.4	Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 42 for chunks size of 4.	160
A.5	Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 185 for chunks size of 4.	160

A.6	Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 1986 for chunks size of 4.	161
A.7	Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 42 for chunk size of 2.	161
A.8	Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 185 for chunks size of 2.	162
A.9	Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 1986 for chunks size of 2.	162
A.10	Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 42 for chunks size of 2.	163
A.11	Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 185 for chunks size of 2.	163
A.12	Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 1986 for chunks size of 2.	164
A.13	The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.	165
A.14	The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.	166
A.15	The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.	167
A.16	The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.	168
A.17	The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.	169
A.18	The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.	170
A.19	The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.	171

A.20	The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.	172
A.21	The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.	173
A.22	The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.	174
A.23	The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.	175
A.24	The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.	176
A.25	The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.	177
A.26	The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.	178
A.27	The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.	179
A.28	The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.	180
A.29	The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.	181
A.30	The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.	182
A.31	The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.	183

A.32	The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.	184
A.33	The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.	185
A.34	The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.	186
A.35	The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.	187
A.36	The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.	188

List of Tables

2.1	Significant whale call detection and classification studies.	11
2.2	The DNN architecture embedding. Before the nonlinearity, at layer segment 6, the x-vectors are extracted. The softmax layer's N value reflects the number of training speakers.	36
2.3	List of speech tasks that SpeechBrain currently supports.	38
3.1	Summary of the different call types and their ID for blue and fin whale sounds in the International Whaling Commission Southern Ocean Research Partnership (IWC-SORP) library. [13].	53
3.2	Distribution and number of annotations at each site by classification type for the blue whales as reported by the AADC [13].	54
3.3	Distribution and number of annotations at each site by classification type for the fin whales as reported by the AADC [13].	55
3.4	Average, standard deviation, maximum and minimum for the call duration as well as the highest and lowest frequency call for the entire AADC data set.	56
3.5	Average, standard deviation, maximum and minimum for blue and fin whale call duration as well as the highest and lowest frequency call in the AADC data set.	57
3.6	Summary of the number of problems encountered per site in the AADC data set that had a mismatch in their sample start and sample end.	58
3.7	Summary of labelling problems encountered in the AADC data set.	59
3.8	Average, standard deviation, maximum and minimum for the blue and fin whale call duration as well as the highest and lowest frequency call in the AADC data set over three selected sites.	60

3.9	Size of data sets used for experimentation.	67
4.1	Hyperparameter ranges searched over for the first optimisation run.	72
4.2	Settings used for the spectrogram in python.	80
4.3	Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using SGD as an optimiser on the four-second chunks. The seed with the best validation loss is in <i>italics</i> and the average between the three seeds is in bold	83
4.4	The trial result for Experiment 1 using SGD as an optimiser and four-second chunks on seed 42.	84
4.5	Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using Adam as an optimiser on the four-second chunks. The seed with the best validation lost is in <i>italics</i> and the average between the three seeds is in bold	85
4.6	The trial result for Experiment 2 using Adam as an optimiser and four-second chunks on seed 42.	86
4.7	Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using SGD as an optimiser on the two-second chunks. The seed with the best validation lost is in <i>italics</i> and the average between the three seeds is in bold	87
4.8	The trial result for Experiment 3 using SGD as an optimiser and two-second chunks on seed 185.	87
4.9	Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using Adam as an optimiser on the two-second chunks. The seed with the best validation loss is in <i>italics</i> and the average between the three seeds is in bold	89
4.10	The trial result for Experiment 4 using Adam as an optimiser and two-second chunks on seed 185.	89
4.11	Summary of the average and standard deviation for the four different experiments in Chapter 4.	91
5.1	Hyperparameter values for the out-of-the-box SpeechBrain TDNN system.	101
5.2	Summary of the results obtained for Adam as an optimiser on the 2- and 4-second chunks for the initial TDNN.	101
5.3	Different ML task for SpeechBrain.	102

5.4	Hyperparameters value ranges for the Orion search.	105
5.5	Result for the ML tasks across the three seeds to see the impact of the deltas using Fbanks as the feature extraction method on 2-second chunks. Bold indicates the results for the best derivative option. <i>Italics</i> indicates the best result across the seeds.	108
5.6	Result for the ML tasks across the three seeds to see the impact of the deltas using MFCCs as the feature extraction method on 2-second chunks. Bold indicates the results for the best derivative option. <i>Italics</i> indicates the best result across the seeds.	112
5.7	Result for the ML tasks across the three seeds to see the impact of the deltas using Fbanks as the feature extraction method on 4-second chunks. Bold indicates the results for the best derivative option. <i>Italics</i> indicates the best result across the seeds.	116
5.8	Result for the ML tasks across the three seeds to see the impact of the deltas using MFCCs as the feature extraction method on 4-second chunks. Bold indicates the results for the best derivative option. <i>Italics</i> indicates the best result across the seeds.	119
5.9	Summary of the average and standard deviation results for the ML tasks. The results are for the small data set using four-second chunks. The best model for each ML task is in bold	123
5.10	Result for the ML tasks across the three seeds including deltas using Fbanks as the feature extraction. The large data set is used with 4-second chunks. Bold indicates the results for the average and standard deviation for the three seeds. <i>Italics</i> indicates the best result across the seeds.	125
5.11	Result for the ML tasks across the three seeds including deltas using MFCCs as the feature extraction. The large data set is used with 4-second chunks. Bold indicates the results for the average and standard deviation for the three seeds. <i>Italics</i> indicates the best result across the seeds.	128
5.12	Confidence intervals for the two-second chunks for the test set.	132
5.13	Summary of the average and standard deviation for the CNN results from Chapter 4 and the small_3_cs result from this chapter. The model with the best validation accuracy for the CNN and TDNNs is bold	133
5.14	Size of datasets used for experimentation.	134
5.15	Transfer learning results.	135
A.1	The different hyperparameter sweep algorithms used by Orion.	156

A.2	The different hyperparameter sweep algorithms used by Orion.	157
-----	--	-----

List of Acronyms

AADC Australian Antarctic Data Centre

ANN Artificial Neural Network

ASHA Asynchronous Successive Halving Algorithm

ASR Automatic Speech Recognition

Ax Adaptive Experimentation

BOHB Bayesian Optimisation Hyperband

BRP Bioacoustics Research Programme

CCIS Communications in Computer and Information Science

CNN Convolutional Neural Network

CSV Comma-separated Values

DCLDE Detection Classification, Localisation, and Density Estimation of Marine
Mammals

DCT Discrete Cosine Transform

DEHB Differential Evolutionary Hyperband

DNN Deep Neural Network

ECAPA Emphasised Channel Attention, Propagation and Aggregation

EER Equal Error Rate

FAR False Acceptance Rate

Fbanks Filter banks

FFT Fast Fourier transform

FP False Positive

FM Frequency Modulation

FN False Negative

FRR False Rejection Rate

GRU RNN Gated Recurrent Unit Recurrent Neural Network

HMM Hidden Markov Model

IWC-SORP International Whaling Commission Southern Ocean Research Partnership

JSON JavaScript Object Notation

kDMD Kernel dynamic mode decomposition

LSTM Long Short-Term Memory

mDMD Modified dynamic mode decomposition

MFCC Mel Frequency Cepstral Coefficients

ML Machine Learning

MLP Multilayer Perceptron

NOAA National Oceanic and Atmospheric Administration

PAM Passive Acoustic Methods

PB2 Population-Based Bandits

PBT Population-Based Training

ReLU Rectified Linear Unit

SACAIR Southern African Conference for Artificial Intelligence Research

SGD Stochastic Gradient Descent

SMBO Sequential Model-Based Global Optimisation

SNR Signal-to-Noise Ratio

STFT Short-time Fourier Transform

TDNN Time Delay Neural Network

TN True Negative

TP True Positive

TPE Tree-Structured Parzen Estimator

VGG Visual Geometry Group

WAV Waveform Audio File Format

WPT Wavelet Packet Transform

YAML Yet Another Markup Language

Chapter 1

Introduction

In this chapter, we offer an overview of the research we conduct and explain how it can contribute to the detection of whale calls. We specify the boundaries of our investigation and outline the research questions and goals that we aim to achieve.

1.1 Background

As years pass, human activities in the ocean, such as fishing and contamination, are intensifying, putting marine mammals in danger of extinction [1]. Because animals are vocal and acoustics propagate well in water, passive acoustic monitoring of their movement can provide valuable information on the distribution of marine mammals [2], [3].

Passive Acoustic Methods (PAM) use underwater microphones, also called hydrophones, to listen to and record signals from marine mammals for detection, classification, and localisation [4]. PAM performs better than visual observation as a monitoring method: it can be used at night, in bad weather, or under other conditions when visual observation is difficult [2].

Acoustic data collection and storage costs have decreased dramatically in the past decade. Today, terabytes of data are collected in a single project [5]. With the increasing volume of acoustic data, extracting meaningful ecological information becomes more and more expensive [5]. Machine learning enables the detection of signals in large data sets at a lower cost and with greater accuracy compared to human analysis [5].

Several successful machine learning approaches exist for whale call detection and classification, as discussed in Section 2.3. Currently, the most prominent approach uses Convolutional Neural Networks (CNNs) together with spectrograms and has been applied successfully to several data sets [2], [6]–[8]. However, a drawback of these spectrogram-based systems is that obtaining high-quality spectrograms over a wide range of underwater acoustic conditions can be challenging, and the process of training a model using spectrograms as features is computationally costly when data sets become large. The reason why the acoustic conditions can be challenging is because of noise and interference. The clarity of acoustic signals in underwater environments can be affected by a range of noise sources, such as biological sounds, shipping activities, and environmental factors. These sources of noise pose a challenge in obtaining spectrograms of high quality [9]. Interference typically refers to unwanted or disruptive signals that degrade the quality of the desired audio signal. This interference can come from various sources, such as background noise. Systems relying on spectrograms, particularly those that use deep learning architectures, can be computationally expensive [10]. Training deep neural networks involves a multitude of calculations and iterations, which become costly if input features are high-dimensional (as spectrograms are) and frequently sampled. Therefore, the processing of large spectrogram data sets requires substantial computational resources [10].

In this work, we explore an alternative approach to the task of whale call detection using a Time Delay Neural Network (TDNN)-based x-vector system, with traditional speech features as input. The speaker recognition task often uses an x-vector system, which is a speaker embedding system [11]. This system represents speaker characteristics in a vector space with fixed dimensions [11]. Machine learning for speech processing tasks typically relies on traditional speech features, which represent a range of acoustic attributes that capture distinct aspects of the speech signal, such as Mel Frequency Cepstral Coefficients

(MFCCs), Filter banks (Fbanks), Linear Predictive Coding (LPC) Coefficients, Delta and Delta-Delta Coefficients. Specifically, we experiment with MFCCs and Fbanks.

We have two reasons for choosing this approach. In future work, we aim to utilise transfer learning to extend some of the smaller South African whale call data sets with internationally available data or models, which is an approach that an x-vector system is particularly good at. In addition, our aim is to develop a system that does not require careful spectrogram calibration, especially when combining disparate data sets at a later point. It is challenging to obtain a well-calibrated spectrogram because it includes careful parameter selection (which includes manual reviews), robust signal processing techniques, rigorous calibration procedures, and validation.

Therefore, our objective is to demonstrate that a neural speech processing system based on a TDNN is a viable alternative technique for detecting whale calls. The TDNN we use is implemented using the SpeechBrain toolkit [12]. We develop initial models using the ‘Acoustic trends for the blue and fin whale library’ from the Australian Antarctic Data Centre (AADC) [13]. The results are compared with a spectrogram-based CNN developed using the same data set.

1.2 Problem statement

In recent years the number of PAM systems has increased [6]. This makes it difficult to process a large amount of PAM-generated sound data manually [5]. A labelled data set can be used to train classifiers to detect or recognise individual whale calls, units, phrases, and themes [8].

The preprocessing and extraction of the spectrogram for the standard approach, a CNN with spectrograms, is computationally expensive and time consuming when training the model. In this study, we want to explore whether an end-to-end Deep Neural Network (DNN) system, specifically a TDNN with x-vectors, is a viable approach for whale call detection, with transfer learning in mind for future use. It is not known whether the end-

to-end DNN system approach can be used for whale call detection or how performance will compare with the more standard CNN-based approach.

We want to explore two modelling approaches: (1) CNNs with spectrograms, as this is an established whale call detection technique; and (2) end-to-end DNNs with Automatic Speech Recognition (ASR)-like features, as this approach provides good results when moving speech processing models from one environment to another [14]. In the end, we would be able to more easily process a large amount of locally collected data that PAM can produce.

1.3 Project scope

According to the above problem statement, the scope of the investigation is limited to a number of data sets and architectures:

1.3.1 Data sets

The AADC corpus contains a library of acoustic trends from blue and fin whales. We exclusively select blue and fin whales because their vocalisations are the most prevalent throughout the data set (more details are given in Chapter 3). The library consists of 1,880.25 hours of audio recordings annotated at 11 different sites over a period of five years [13]. A total of 105,161 annotations are made at all sites, but the annotation numbers are not evenly distributed by site or classification [13]. The annotations made in this corpus are manually annotated. Manual annotations are performed by an analyst who has expert experience in the identification of blue and fin whale sounds. The recordings are visualised in Raven Pro.

Full details about the distribution of annotations among the different sites and classifications are shown in Section 3.2. Blue whale classifications were generally observed more frequently than fin whale sounds in most site years [13]. Techniques will be developed

primarily on this corpus, since it is easier to work with labelled data. We only consider this corpus because finding publicly available corpora that are *annotated* is not easy. This corpus provided enough data to train our models.

The AADC data set will be partitioned into a small and large data set. This partition is done because the CNN with spectrograms is computationally more expensive and time consuming than the new TDNN. With the current training setup (meaning hardware) and the available processing time that we have to train models, it is not possible to train models with very large data sets.

1.3.2 Architectures

The effectiveness of convolutional networks in detecting whale calls in the small data set will be evaluated against the baseline CNN network from a previously well-established CNN system using spectrograms. The end-to-end DNN architecture for ASR will be explored to the extent of detecting whale calls through ASR-like features. The end-to-end system architecture will utilise the SpeechBrain toolkit. This will be customised and extended to accommodate whale call features. The effectiveness of the end-to-end DNN system will be evaluated against the baseline model.

1.3.3 Toolkits

Pytorch, Sonic Visualiser, and Raven Pro will be used for spectrogram extraction, while the baseline CNN system will be developed in Pytorch. SpeechBrain [12], a PyTorch speech toolkit, will be investigated to use end-to-end DNNs for whale call detection. See Sections 2.5.3 and 5.2 for more information.

1.4 Research questions

The research question that needs to be answered through this study is:

- Is it feasible to apply end-to-end DNNs to whale calls for detection?
- How well does this approach compare with the more established CNNs and the spectrogram approach?
- Can ASR-like features be used to detect whale calls?
- Can a model developed in one domain be used without adaptation to detect whale calls from a partially matched domain?

1.5 Objectives

The main objective of this study is to develop an end-to-end DNN using ASR-like features. This tool can in future be used to help label unlabelled data in a whale corpus by detecting whale calls.

- Investigate and compare the performance of existing approaches to whale call detection and classification through a review of the literature.
- Develop a baseline CNN system to evaluate its effectiveness in detecting calls from whales within the domain.
- Evaluate the use of end-to-end DNNs to detect whale calls using ASR-like features.
- Analyse the implications of specific design choices (for example, regarding features, architecture, and/or model size).

1.6 Research methodology

This study can best be defined as an exploratory research study combining existing techniques to investigate their application in whale call detection. We will also explore a novel approach with end-to-end DNNs using ASR-like features for detecting whale calls. Empirical experiments will be conducted using different architectures and techniques, in order to develop a tool capable of detecting whale calls.

1.7 Dissertation overview

This dissertation consists of seven chapters and is structured as follows:

- Chapter 2 provides background information on the main techniques used in this study and the concepts that are used in the rest of the work.
- Chapter 3 explains the original data set, the preprocessing steps, and the data set used for experimentation.
- Chapter 4 is dedicated to describing the baseline CNN architecture, optimisation techniques, spectrogram extraction methods, and the experiments conducted with their analysis.
- Chapter 5 presents the TDNN used for whale call detection and compares it with the baseline CNN. We describe the experimental setup, the optimisation protocol, and the analysis of the results for TDNN.
- Chapter 6 concludes the study by summarising the key findings and offering suggestions for future work.

1.8 Publications

Parts of Chapters 3, 4 and 5 were published in the paper:

- Edrich Fourie, Marelie H. Davel and Jaco Versfeld, “Neural speech processing for whale call detection”, Artificial Intelligence Research (SACAIR 2022), Communications in Computer and Information Science, Vol 1734, pp 276-290, November 2022 [15].

The paper was presented at Southern African Conference for Artificial Intelligence Research (SACAIR) and included in the Communications in Computer and Information Science (CCIS) volume containing selected papers from the conference.

Chapter 2

Background

This chapter presents a literature review related to the study, covering whale vocalisation, various features used, different deep neural networks, their optimisation, evaluation metrics and related research.

2.1 Overview

This chapter provides background information on key concepts used throughout this study. We first describe whale vocalisation (Section 2.2) before reviewing previous studies on whale call detection and classification in Section 2.3. An important aspect of these systems is the first step when features are extracted from the original waveform. Different approaches to feature extraction are described in Section 2.4.

In Section 2.5, we provide an overview of the deep neural networks that are used to model the extracted features. The process to train and optimise neural networks is described next in Section 2.6. Section 2.7 describes the optimisation tools that are used for deep neural networks, before the conclusion of the chapter in Section 2.8 with the evaluation

metrics used to measure model performance during experiments.

2.2 Whale vocalisation

Vocalisation (producing sound) is the way whales communicate, find food, and find each other [16]. Sound production mechanisms differ from one family of cetaceans to another. Many types of whales travel in groups called ‘pods’ because they are very social animals [7], [16]. They socialise and communicate using a variety of calls (sounds) [16]. The three main types of sounds made by whales are; (1) clicks, (2) whistles, and (3) pulse calls.

1. **Clicks** are an echolocation tool used by whales that is believed to be primarily used for navigation and the identification of physical objects [16]. Echolocation is a method used by certain species to detect their surroundings when visual signals are insufficient, such as in low light conditions or in cloudy water [17]. They send a sound wave that bounces off an object; when the sound wave returns to the whale, it helps them identify the shape of the object [16]. Additionally, these clicks can help the whale distinguish between friendly creatures and predators [18]. Lastly, it has also been found that clicks are used during social interactions, which means that the click sound may have some communication function [16].
2. **Whistles** are generally used for communication in social situations. [2].
3. **Pulsed calls** are primarily used during social activities and are more frequent than whistles. To the human ear, these pulsed calls sound like squeaks, screams, and squawks [2], [16].

Several whale populations have been found to have different dialects of vocalisation among their pods [16]. Whales probably do this so that they can distinguish between members of their pod and strangers.

2.3 Related work

Various studies use speech features or spectrograms for the detection of whale calls. While the most prominent model used is a CNN, other models are also used, as listed in Table 2.1. We are unaware of any studies using a TDNN and x-vectors for the detection of whale calls.

We look at studies that apply CNNs with spectrograms for the detection and classification of whale calls. In Section 2.3.1, we look at studies of right whales that implement various CNNs. Long-finned whale studies are discussed in Section 2.3.2, sperm whale studies in Section 2.3.3 and, lastly, killer whale studies in Section 2.3.4. (Additional details on the terms used in this section can be found in Section 2.6.)

We exclude the studies done on Multilayer Perceptrons (MLPs), Long Short-Term Memory (LSTM) and Gated Recurrent Unit Recurrent Neural Networks (GRU RNNs) because of their lower accuracies compared to CNNs [8], [19]. The reason for not working with Hidden Markov Models (HMMs) is that the data set used has more than enough labelled data for a CNN to be the better option in this case [20]. Another reason is that the temporal structure of the sound signal is not necessary for CNNs, but rather for spectrograms.

Table 2.1: Significant whale call detection and classification studies.

Study	Data set	Feature type	Model	Accuracy
Acoustic Detection of Humpback whales Using a Convolutional Neural Network [4]	National Oceanic and Atmospheric Administration (NOAA)	Spectrograms	CNN	On the test set: 90% precision and 90% recall when a clip contains a humpback call.
Blue whale calls classification using Short-time Fourier Transform (STFT) and Wavelet Packet Transform (WPT) transforms and artificial neural network [19]	Saguenay-St. Lawrence Marine Park	STFT and WPT	MLP	Classification performance: STFT/MLP obtained 86.25% and WPT/MLP obtained 84.22% on different blue whale calls

Whistle detection and classification for whales based on convolutional neural networks [2]	15 sound samples with a total duration of 120 minutes	Spectrograms	LeNet-5	97% correct detection rate for whistle calls and 95% correct classification rate in the test set to distinguish between killer whales and long-finned whales
Detection and classification of marine mammal sounds using AlexNet with transfer learning [21]	Watkins Marine Mammal Sound Database	Spectrograms	AlexNet	Detection accuracy of 99.96% for whale calls and Classification accuracy of 97.42% to distinguish between killer whales and long-finned whales
Deep Machine Learning Techniques for the Detection and Classification of Sperm Whale Bioacoustics [8]	The Dominica coda data set	Spectrograms	Detection model: CNN Classification model: LSTM and GRU RNN	Detection accuracy of 99.5% for sperm whale clicks
ORCA-SPOT: An Automatic Killer Whale Sound Detection Toolkit Using Deep Learning [7]	The Orchive and OrcaLab	Spectrograms	ResNet	Positive-predictive-value of 93.2% is achieved for detecting killer whale calls
Deep neural networks for automated detection of marine mammal species [5]	Detection Classification, Localisation, and Density Estimation of Marine Mammals (DCLDE) 2013 workshop data, NOAA, Cornell University's Bioacoustics Research Programme (BRP) and Kaggle data competition	Spectrograms	LeNet, ResNet, BirdNet and Visual Geometry Group (VGG)	LeNet had the highest precision for a certain recall.

Classification of whale calls based on transfer learning and convolutional neural networks [22]	Whale Frequency Modulation (FM)	Spectrograms	CNN and transfer learning	Classification accuracy of 97.04% for two whale species and 91.47% classification accuracy for four whale sub-populations
Detection of baleen whale species using kernel dynamic mode decomposition-based feature extraction with a hidden markov model [23]	Hydrophone recordings of Southern right whale species found in the coastal waters of False Bay, Western Cape, South Africa. Mobysound.	Modified dynamic mode decomposition (mDMD). Kernel dynamic mode decomposition (kDMD).	HMM.	The kDMD-HMM generally performed better than the mDMD-HMM and DMD-HMM detectors.

2.3.1 Right whale call detection

In the study by Shiu *et al.* [5], DNNs were used to detect a stereotyped contact call (upcall) that is produced by North Atlantic right whales (*Eubalaena glacialis*). Upcalls produced by right whales are found in both sexes and all ages [5]. This FM up-sweep or ‘upcall’ is typically about one second long and sweeps upward with frequencies between 100 and 200 Hz approximately [24]. Acoustic monitoring systems are often able to detect upcalls due to their stereotyped nature [5].

The databases used for this study are: (1) DCLDE 2013 workshop data [25], (2) right whale recording collected by Cornell University’s BRP now called the Center for Conservation Bioacoustics and NOAA [16], [24], and (3) Kaggle data competition [5], [24].

Shiu *et al.* [5] explored multiple CNNs to detect right whale upcalls. The CNNs used were: (1) LeNet, (2) BirdNET, (3) VGG, and (4) ResNet. LeNet is based on the published architecture, which contains many of the fundamental components of CNNs [5]. The authors’ modifications in this study are: max-pooling is used instead of average-pooling, and for the activation function, tanh activation is used instead of Rectified Linear Unit (ReLU) [5]. Dropout is also applied to the input layer and after the two max-pooling

layers. BirdNET is a CNN that uses weakly supervised learning to identify bird species in acoustic recordings. After each max-pooling layer, a dropout layer with a probability of 0.2 is inserted. High L2 regularisation (0.2) is applied within the convolutional layers to prevent overfitting [5]. In VGG, complex features are learnt using filters with small kernel sizes and an increased network depth [5]. The gradient decreases with each layer; this creates a challenge with the backpropagation of the gradient [5]. ResNet solution mitigates this problem by propagating gradients downward through an identity function, allowing very deep architectures to be built [5].

The performance of the deep neural networks yielded considerably high precision and recall, but they all varied. LeNet has the highest precision and the lowest number of false positives for a given recall [5]. BirdNet has a high computational cost as a result of the complex architecture with many parameters. BirdNet performed just as well as LeNet [5].

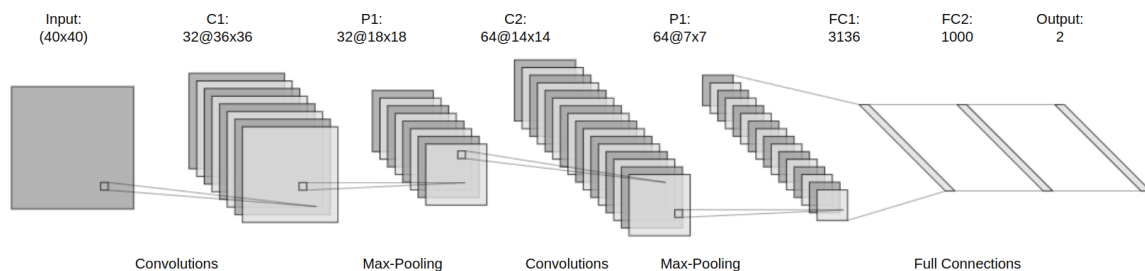


Figure 2.1: Representation of the LeNet convolutional neural network. *Adapted from Shui et al. [5].*

2.3.2 Long-finned and killer whale whistle detection and classification

In the study by Jiang *et al.* [2], 15 sound samples were recorded in waters near Antarctica, Canada, Norway, Mexico and the United States. The recordings contain mainly whistles from killer whales and long-finned whales. A total of 120 minutes of sound samples are used, sampled at 44,100 Hz [2].

The network used in this study for the detection and classification model draws on the

structure of LeNet-5. One reason is that there are contours similar to handwritten numbers to the whistles of whale species in the time-frequency spectrogram [2]. The structure of the detection model consists of two convolutional layers, each followed by a pooling layer. After the second pooling layer, a fully connected layer is added with a dropout layer with a dropout rate of 0.2. The last two layers consist of another fully connected layer and a softmax layer. The softmax layer converts the output into a pair of probabilities [2]. The model predicts whether the input signal contains a whistle signal or not. The ideal output for the input of a whistle signal will be (1,0) and (0,1) for the opposite [2]. The full approach for the detection can be viewed in Jiang *et al.* [2].

The classification model has the same structure and hyperparameters as the detection model, but the input data, output results, and training processes differ [2]. A classification model can identify the corresponding whale species based on the whistle spectrograms of the detected whistles. The full approach for the classification can be viewed in Jiang *et al.* [2]. Figure 2.2 shows the network architecture used for the detection and classification model. The detection models achieved a 97% correction rate and a 95% correct classification rate on the test set [2].

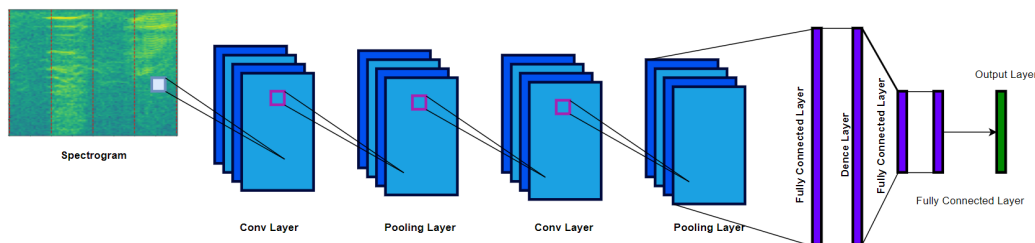


Figure 2.2: LeNet CNN architecture used for long-finned and killer whale whistle identification and classification.

2.3.3 Sperm whale calls detection

In the detection of sperm whale bioacoustics, spectrograms and CNNs are used. Bermant *et al.* [8] achieves a training detection accuracy of 99.5% and a testing accuracy of 100%. The classification accuracy obtained is 99.7%. The training set consists of 650

spectrograms, 325 belonging to each class (clicks and no clicks); for the testing set, 100 spectrograms are used, with 50 belonging to each class. The method followed in this article is [8]: (1) the preprocessing of the raw acoustic data; (2) the development of a time domain click detector that is based on a threshold.; (3) the generation of an image data set consisting of appropriately labelled spectrogram images; (4) constructing and training a CNN; (5) testing the trained CNN to assess its ability to generalise to new data; and (6) analysing the robustness of the network and investigating its feature extraction behaviour [8].

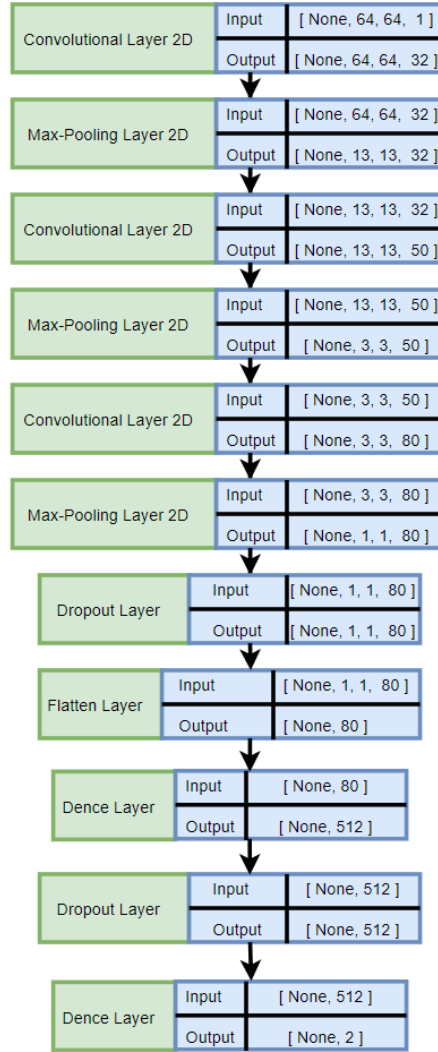


Figure 2.3: The architecture of the CNN-based echolocation click detector [8].

As shown in Figure 2.3, the detection model architecture consists of three consecutive convolutional and max-pooling layers, followed by fully connected dense layers [8]. Reg-

ularisation of dropout is included to prevent overfitting.

2.3.4 ORCA-SPOT

In the family of dolphins, killer whales (*Orcinus orca*) are the largest and one of several species with relatively well-studied and complex vocal cultures [7]. Much research has focused on killer whale acoustic behaviour in the Northeast Pacific, where the three ecotypes of killer whales found in this region are resident fish-eaters, transient mammal-eaters, and offshore types [7]. There is a wide variety of vocalisations used by killer whales, including clicks, whistles, and pulsed calls. Echolocation clicks are short pulses of variable duration (between 0.1 and 25 ms) that killer whales use for navigation and localisation, which are repeated at varied rates (between a few pulses per second to over 300 per second) [7]. Whistles have a narrow frequency band with no harmonic components or a few, typically between 1.5 and 18 kHz, and a duration of 50 ms to 12 seconds [7]. Pulsed calls by killer whales, the most common and intensively studied form of vocalisation, are characterised by sudden and patterned shifts in frequency based on the pulse repetition rate, which is usually between 250 and 2000 Hz [26].

The data sets used in this study are Orchiive [27] and OrcaLab [28], which include 23,511 underwater recordings of whale calls of around 45 minutes over 23 years and include 18,937.2 hours [7]. Several recruited researchers extracted about 15,480 human-labelled audio files (Orchiive Annotation Catalogue) through visual (using spectrograms) and aural (listening to the audio) comparisons. The total annotation time is around 12.3 hours [7].

The network architecture of the well-established ResNet was used to develop ORCA-SPOT, which is an automatic killer whale call detection toolkit [7]. In this study, two aspects were reviewed in detail [7]: (1) the depth of traditional ResNet architectures, and (2) in the first residual layer, the removal/preservation of the max-pooling layer. The following common ResNet variants were used: (1) ResNet18, (2) ResNet34, (3) ResNet50, and (4) ResNet101. The details of all these well-established ResNet architectures can be found in the work of He *et al.* [29].

Model	ORCA-SPOT-MAX-POOL						ORCA-SPOT-NO-MAX-POOL					
	Accuracy %			Statistics %			Accuracy %			Statistics %		
Arch	run1	run2	run3	max	mean	stdv	run1	run2	run3	max	mean	stdv
ResNet18	95.39	93.99	92.84	95.39	94.08	1.28	95.88	96.15	94.4	96.15	95.48	0.94
ResNet34	93.65	95.72	95.2	95.72	94.86	1.08	96.13	95.65	95.12	96.13	95.64	0.51
ResNet50	92.39	95.76	94.88	95.76	94.35	1.75	96.37	95.9	95.61	96.37	95.96	0.38
ResNet101	94.39	95.33	95.01	95.33	94.91	0.47	95.81	94.1	96.24	96.24	95.39	1.13

Figure 2.4: Model accuracies for common ResNet architectures by comparing architectures with and without max pooling (3×3 kernel, stride 2) in the first residual layer. *Adapted from Bergler et al. [7].*

Referring to Figure 2.4, Bergler *et al.* [7] concluded that the test accuracy does not improve due to deeper ResNet models. This phenomenon is observed regardless of whether the max-pooling layer is removed or maintained. On average, models without max-pooling in the first residual layer improved by 1% [7].

The network for ORCA-SPOT is based on the established ResNet18, where the first residual layer and the max-pooling layer are removed [7]. The reason for removing these layers is that it already causes a large loss of resolution at the initial stage, leading to a disadvantage regarding high-frequency subtle killer whale calls [7]. Global average pooling is applied to bottleneck training features following the last residual layer, including 512 feature maps with 16×8 hidden units. For the last step, the results are connected to a 512-D fully connected layer that projects the output into two output classes: ‘killer whale’ and ‘noise’. The network architecture for the ORCA-SPOT is shown in Figure 2.5 [7].

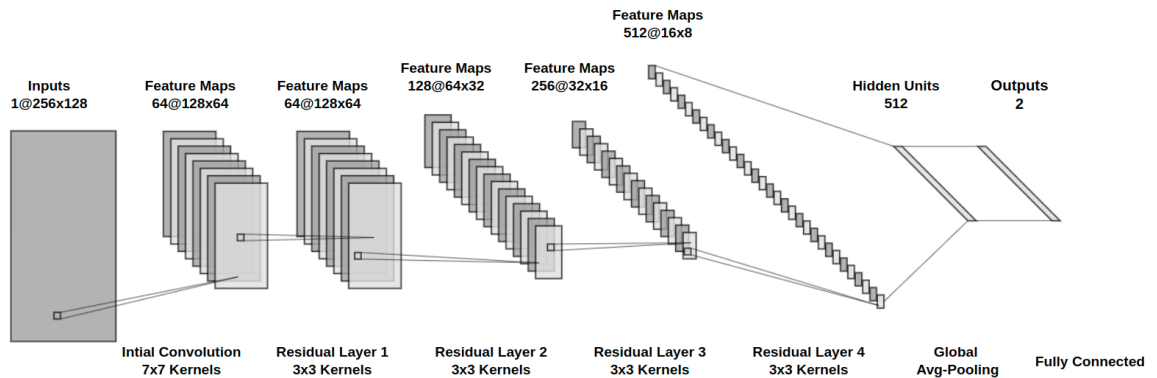


Figure 2.5: ORCA-SPOT network architecture. *Adapted from Bergler et al. [7].*

2.4 Feature extraction

Three types of speech features are used in this study: spectrograms (Section 2.4.1), Fbanks (Section 2.4.2) and MFCCs (Section 2.4.3).

2.4.1 Spectrograms

In spectrograms, we can visually see a signal's strength, or loudness, over time at different frequencies within a particular waveform [30]. Spectrograms are two-dimensional graphs with colours representing the third dimension [30]. The time axis is depicted on the horizontal axis, whereas the frequency axis is depicted on the vertical axis. The third dimension, indicated by colour, represents a particular frequency's amplitude (energy or loudness) at a particular point in time [30]. Frequency is the count of repeated occurrences of an event within a specific time period [31]. In the context of sound, the frequency is often measured in Hertz (Hz), where 1 Hz represents one cycle per second [31].

Spectrograms can be calculated using the following steps [32], [33]:

- The audio is divided into overlapping windows. This approach allows for achieving a balance between capturing the changes over time in the signal and maintaining the frequency resolution.
- The STFT is performed on each of these windows, and the absolute value is taken.
- Each of these windows has a resulting line representing the magnitude vs. frequency.
- The next step is to convert the resulting windows to decibels. This results in a rich image of the sound's structure.
- The last step is to arrange these windows back into the duration of the original audio and to then display the output image.

Some important settings we used in this study for spectrogram extraction are as follows:

-
- **Window size:** This refers to the decomposed signal chunks with a specified width. With a larger window size, we get better frequency resolution because more frequencies are being captured [10]. This allows for a more detailed analysis of the signal's frequency content over a longer period. (There is a trade-off between obtaining either a better frequency resolution or better time resolution.) However, the time localisation is poor [10].
 - **Window type:** The Hann function is most commonly used in random signal processing because it offers good frequency resolution, leakage protection, and fair amplitude accuracy [34]. Spectral leakage is a phenomenon in which the energy of a particular frequency component in a signal leaks to adjacent frequency bins [35]. This occurs when the signal does not align precisely with the frequency grid established by the selected analysis window [35]. Leakage protection involves methods and techniques to minimise or mitigate the effects of spectral leakage. The Hann window is defined as [36]:

$$w(n) = 0.5(1 - \cos \frac{2\pi n}{N-1}) \quad (2.1)$$

where n is the sample index and N is the total number of samples.

The Hann window is used to obtain the windowed signal ($y(n)$) by multiplying each sample of the signal with the corresponding sample of the Hann window. As defined in the equation below:

$$y(n) = x(n) \cdot w(n) \quad (2.2)$$

where $x(n)$ is the original signal.

The Hann window is chosen due to its ability to achieve a favourable trade-off between the width of the main lobe and the suppression of side lobes. It has relatively low side lobes, which effectively mitigates spectral leakage.

- **Overlapping:** The proper spectrogram will have overlapping samples in each chunk [37]. Overlapping defines how many samples are overlapped in each window [37].
- **Zero padding:** The zero-padding technique is commonly used to make the size of an input sequence equal to a power of two [38]. The zero-padding method involves

adding zeros to the end of an input sequence to equal the number of samples to the next higher power of two [38]. If we have ten signal samples, we can increase the total number of samples by adding six zeros to equal 16 [38]. This number should be a power of two, such as 16 or 32.

- **Size of Fast Fourier transform (FFT) (nfft):** Essentially, the FFT is taken of each chunk, and nfft tells us how many FFT points are desired to be computed per chunk [39]. The nfft also provides a measure of how fine-grained the frequency resolution will be [39]. As the FFT points increase, the frequency resolution increases and finer details can be observed along the frequency axis [39]. As the number of FFT points increases the total duration of the signal in the time domain increases, and this leads to a decrease in the frequency space. This means that with the decrease in frequency spacing the frequency resolution and the ability to distinguish finer differences in the frequency domain becomes better.
- **Hop Size:** Hop size refers to the number of samples between consecutive windows in a signal analysis.
- **Floor:** The floor setting displays the floor colours for the powers that are less than or equal to the set floor value [40].
- **Ceiling:** Powers greater than or equal to the set ceiling value are displayed as the ceiling colour [40].

CMYK image

A CMYK digital image is an image that is represented and encoded using the CMYK colour model. In this colour model, colours are expressed as combinations of four colours: Cyan (C), Magenta (M), Yellow (Y), and Key (K) which is usually black [41]. A 32-bit CMYK image has four 8-bit channels ranging from 0-255, defining the intensity of that specific pixel [41].

2.4.2 Filter banks

Fbanks is a collection of bandpass filters used in image or signal processing to separate an input signal into a number of analysis signals, each of which carries a specific frequency sub-band from the original signal [42]. Fbanks are used to perform various tasks, such as bandwidth reduction, sample rate modifications, standard spectrum translation operations combinations, signal spectral composition, and decomposition [42]. In Section 2.4.3, we explain how Fbanks are obtained.

2.4.3 Mel frequency cepstral coefficient (MFCCs)

Different signal processing applications use MFCCs extensively in feature extraction [43]. Images, speech, cetacean vocalisation, gestures, and drone sound recognition are just a few examples of such applications [43]. An MFCCs is used to extract distinctive features from a time-varying signal. To extract features from the signal, it transforms the signal from the time domain into the Mel-scale frequency domain using two types of filters; (linearly spaced filters and logarithm spaced filters) [43], [44].

The processing of speech is an essential part of any speech system, whether ASR, speaker recognition, or something else. MFCCs and Fbanks are among the most popular features for human speech [45]. The procedure for computing MFCCs and Fbanks is almost the same. In both cases, Fbanks is computed, and following a few extra steps, we can obtain MFCCs. The following steps are followed to obtain the Fbanks and MFCCs [45]:

1. Pre-emphasis filter
2. Framing
3. Windowing
4. Fourier-Transform and Power Spectrum
5. Fbanks

6. Mel-frequency Cepstral Coefficients (MFCCs)

7. Mean normalisation

Pre-emphasis

In pre-emphasis, we apply a pre-emphasis filter to the speech audio signal because this will amplify the high frequencies. The reasons why we want to apply a pre-emphasis filter are [45]:

1. It will help to balance the frequency spectrum because if we look at the speech spectrum, we can see that the energy distributes more in the lower frequencies than in the higher frequencies. Therefore, the pre-emphasis filter will increase the energy in the high frequencies.
2. The balancing of the spectrum will help to avoid numerical problems during the Fourier transform operation.
3. The Signal-to-Noise Ratio (SNR) can also improve due to the pre-emphasis because the amplification of the higher frequencies tends to become stronger than the high-frequency noise components.

Let us say we have a signal x ; then the pre-emphasis filter can be applied using the first-order filter in the following equation [45]:

$$y(t) = x(t) - \alpha x(t - 1) \quad (2.3)$$

where typical values for the filter coefficient (α) are 0.95 or 0.97.

Framing

Framing is the process after pre-emphasis where we want to split the signal into frames of shorter times. The reason behind the shorter time frames is due to the frequency change

over time in the signal. When applying the Fourier transform operation over the entire signal, the frequency contours of the signal over time can be lost [45]. A solution to this problem is to assume that the frequency contours of the signal are stationary for a very short period. If we apply the Fourier transform operation on these short time frames and concatenate the adjacency frames, we can better approximate the frequency contours [45].

Windowing

We need to apply to each frame a window function. The reason for applying a window function is to counterbalance the Fourier transform's assumption that the data are infinite and to prevent spectral leakage [45]. The window function that we will use is the Hamming window of the following form [45]:

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) \quad (2.4)$$

where N is the window length and $0 \leq n \leq N-1$.

Fourier-transform and power spectrum

Fourier Transform and STFT are mathematical techniques that are used in signal processing to analyse and transform signals in the frequency domain.

- **Fourier Transform** is a mathematical process that converts a signal in the time domain into its corresponding representation in the frequency domain [46]. The reason behind using the Fourier transform is to break down the signal into its individual frequencies, thereby exposing the frequency components along with their corresponding amplitudes and phases [47]. The formula for the Fourier transform of a continuous function $f(t)$ is [47]:

$$F(w) = \int_{-\infty}^{\infty} f(t) \cdot e^{-iwt} dt \quad (2.5)$$

where $F(w)$ is the Fourier Transform of $f(t)$, w is the angular frequency (in radians per unit time), e is the base of the natural logarithm, also known as Euler's number, i is the imaginary unit. The integral is taken over all time t .

-
- The STFT is a technique used to analyse the frequency content of a signal over short, overlapping time intervals [48]. It provides a time-frequency representation of the signal, which is particularly useful for analysing signals that change in frequency content over time [48]. The squared magnitude of STFT is known as the time-frequency representation of the signal spectrogram [48]. Mathematically, the STFT of a signal $f(t)$ is defined as:

$$X(t, w) = \int_{-\infty}^{\infty} f(t) \cdot w(\tau - t) \cdot e^{-iw\tau} d\tau \quad (2.6)$$

where $X(t, w)$ is the STFT of the signal $f(t)$, w is the angular frequency, e is the base of the natural logarithm, i is the imaginary unit and $w(\tau - t)$ is the window function centered at time t . The integral is taken over all time τ .

We can calculate the STFT and then compute the power spectrum (periodogram) using the following equation [45]:

$$P = \frac{|STFT(x_i)|^2}{N} \quad (2.7)$$

where N is typically 256 or 512 and x_i is the i^{th} frame of the signal x .

When computing the STFT of a continuous-time signal, we will get two values: the phase, and the magnitude of the signal [45].

The magnitude indicates the strength or amplitude of the different frequency components found in the signal at different time points. Larger magnitudes usually indicate stronger or more noticeable frequency components in the signal. Understanding the spectral properties of the signal, such as identifying the main frequencies and how they change over time, relies on magnitude information.

The phase captures the relative timing or displacement of different frequency components within the signal at each time interval. It indicates how the various frequency components are aligned or synchronised. Although phase information is not directly included in power spectra, it can still be relevant in certain contexts, such as when reconstructing the signal from its spectral representation or when considering phase-sensitive applications.

However, in most cases, power spectra are primarily concerned with the magnitude com-

ponent, as they provide valuable information about the distribution of signal power across different frequencies.

Filter banks

This is the final step in computing Fbanks. To compute Fbanks, we need to apply triangular filters on a Mel scale to the power spectrum to extract frequency bands [45]. The Mel scale aims to imitate how the human ear perceives sound in a non-linear manner [45]. This is done by the Mel scale being more discriminative at lower frequencies and vice versa. To convert between Hertz (f) and Mel (m) the following equation can be used [45]:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \quad (2.8)$$

$$f = 700(10^{m/2595} - 1) \quad (2.9)$$

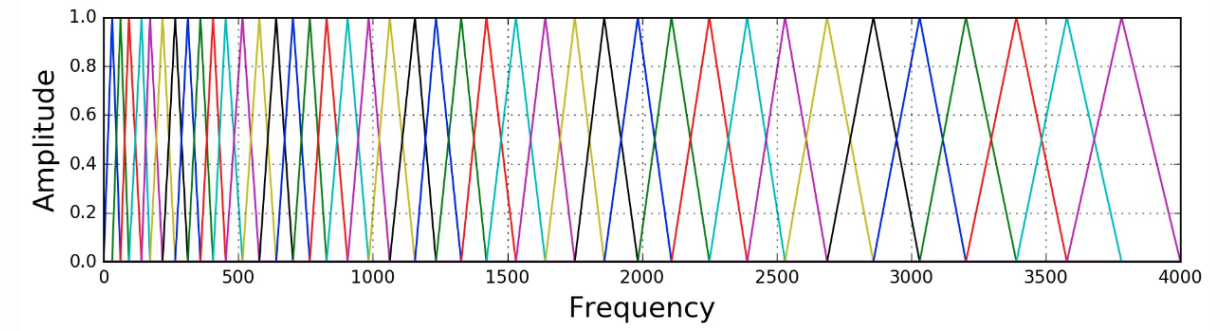


Figure 2.6: Fbanks on a Mel-Scale [45].

In Figure 2.6, we can see that each filter in the Fbanks is triangular. Each has a response of one at the centre frequency; the response decreases linearly towards zero until it reaches the centre frequency of the two neighbouring filters, where the response is zero [45].

We can apply the power spectrum we obtained in the previous step to the Fbanks to obtain the desired spectrogram [45]. If Mel-scaled Fbanks are the desired features we wanted, we can skip to the mean normalisation step; otherwise, we will continue to show how to obtain MFCCs [45].

Mel-frequency Cepstral Coefficients (MFCCs)

In some machine learning algorithms, the Fbanks coefficients could be problematic, as computed in the previous step, due to the coefficients being highly correlated [45]. Correlation refers to the extent to which two variables exhibit a relationship with each other [49]. A high correlation between the coefficients indicates that they are strongly associated or comparable to each other. Here is why highly correlated coefficients can be problematic [49]:

1. **Redundancy:** It is an indication that the coefficients are providing similar information. These redundant features can lead to overfitting (confusing the model) or inefficient use of computational resources.
2. **Instability:** The model can become unstable, as even small variations in the input data or slight perturbations can result in substantial changes in the coefficients. This instability can disrupt the learning process.
3. **Generalisation:** Models that are trained using features that have a high degree of correlation may not be able to effectively apply what they have learned to new, unseen data. While these models may successfully capture patterns that are specific to the training data, they may struggle to adapt to unfamiliar patterns.

To solve this problem, we can apply the Discrete Cosine Transform (DCT) to decorrelate the Fbanks coefficients. The DCT will return a compressed representation of the Fbanks [45]. A DCT represents a sequence of data points as a sum of cosine functions of different frequencies and amplitudes [50]. It can be used to compress an image [50].

Mean Normalisation

During mean normalisation, the mean of each feature over all samples in the training set is calculated and these means are then subtracted from all samples. By simply subtracting the mean of each coefficient from all the frames, we can balance the spectrum and improve

the SNR [45]. After mean normalisation, each feature will have a mean of zero, and the data (per recording) will be centred around the origin. This process helps to eliminate biases in the data and ensures that features with different scales do not dominate the learning process.

2.5 Deep neural networks

In this section, we look at the architectures of the two approaches we use: (1) CNN (Section 2.5.1), and (2) TDNN with x-vector (Section 2.5.2). Lastly, we look at the specific toolkit we use when implementing TDNNs, namely SpeechBrain, in Section 2.5.3.

2.5.1 Convolutional neural network

In the last decade, CNNs have produced unprecedented breakthroughs in various domains related to pattern recognition, including image processing and speech recognition [51].

CNNs is a well-known architecture for deep learning that is inspired by the natural visual perception process of living beings [52]. Hubel and Wiesel [53] discovered that cells in the visual cortex of animals are responsible for detecting light in receptive fields [52]. This discovery motivated Kunihiko Fukushima to create the neocognitron in 1980 [54], which may be regarded as CNN’s precursor [52].

CNNs are similar to conventional Artificial Neural Networks (ANNs) in that they consist of neurons that optimise themselves through learning [55]. Each neuron will continue to receive an input and execute an operation (such as a scalar product followed by a non-linear function), the fundamental building blocks of numerous ANNs [55]. Describing ANNs with a simple definition is difficult [56]. One way to conceptualise ANNs is to think of them as a black box that has numerous input and output channels [56]. These networks function use a large number of interconnected arithmetic units that are mainly in parallel [56]. It is crucial to note that ANNs are most effective when handling relationships

between inputs and outputs that are nonlinear in nature [56]. The input of a CNN for computer vision consists of images with numerical values in each pixel spatially grouped along the width, height, and depth (channels) dimensions. The objective of the whole architecture is to get a probability score for an image belonging to a given class by learning from these spatially ordered numerical data. The last layer includes loss functions connected to the classes and all standard techniques established for conventional MLPs are still applicable [55]. An MLP can be described as a combination of nonlinear activation functions, where each node (except the input nodes) receives a weighted sum of the outputs from the nodes in the previous layer and, if activated, passes the values along to all the nodes in the next layer. The key difference between CNNs and traditional MLPs is that CNNs employ convolutional kernels instead of fully connected nodes, making them better suited for tasks such as image recognition [51]. Additionally, CNNs can significantly reduce the number of parameters required in MLP [51]. CNNs have an architectural design that includes parameter sharing, local connectivity, and hierarchical feature learning [57]. These features enable CNNs to capture spatial patterns more efficiently compared to fully connected MLPs. As a result, CNNs require fewer parameters, making them highly effective for tasks involving structured data, such as image recognition.

The elements of the CNN used in this study are defined as follows:

- **Convolution layer:** The fundamental component of a CNN is a convolutional layer [58]. It includes a collection of filters (or kernels) whose parameters must be learnt during training [58]. Typically, the size of the filter is less than the image itself. Each filter convolves with the image to generate an activation map [58]. The input data is filtered by moving the filter across the entire input. At each position, the filter is multiplied element-wise with the overlapping region of the input. The resulting values are then added together to generate a single value in the output.
- **Kernel:** The kernel of a CNN is only a filter that is used to extract image features [59]. The kernel is a matrix that traverses the input data, performs the dot product with subregions of the input data, and obtains the output as a matrix of dot products [59], [60].

-
- **Stride:** The term striding will be used in the context of convolution, where stride refers to the relative offset imposed on the filter kernel [61].
 - **Padding:** Padding is significant for CNNs since it relates to the number of pixels added to an image, while the CNN kernel processes it [62]. If the padding in a CNN is set to zero, for instance, every added pixel value is the value zero [63]. However, if the zero padding is set to one, a one-pixel border with zero values is applied to the image [63].
 - **Batch normalisation:** In 2015, batch normalisation was proposed for stable learning [64]. Normalising the samples inside a mini-batch stabilises the distribution of non-linear inputs [65]. Normalisation is accomplished by adding extra layers to the network that adjust the mean and variance of each activation over a given mini-batch to zero and one, respectively [66]. To preserve the expressive capabilities of the model, trainable parameters are typically employed to adjust the batch normalised inputs by scaling and shifting [66].
 - **Max-pooling:** After a convolutional layer usually comes a pooling layer [67]. This is specifically after the application of a non-linearity (e.g. ReLU) to the feature maps generated by a convolutional layer [67]. Max-pooling is a pooling procedure that determines the maximum or biggest value in each feature map patch [67].
 - **Fully connected layer:** The fully connected layer resembles the arrangement of neurons in a conventional neural network [51]. Consequently, each node in a fully connected layer is linked to every node in the preceding and succeeding layers [51]. The input is multiplied by a weight matrix, and then a bias vector is added to the output of a fully connected layer [68].

2.5.2 Neural speech processing

Several neural approaches produce state-of-the-art results for speech-processing tasks such as speaker recognition and speech recognition [69]. Speaker recognition refers to the process of identifying or confirming a person's identity given a segment of speech, while

speech recognition refers to understanding the content of a speech utterance. We list a few.

Vaessen *et al.* [70] applied the wav2vec2 framework to speaker recognition and achieved a 1.88% Equal Error Rate (EER) on the VoxCeleb1 extended test, compared to the Emphasised Channel Attention, Propagation and Aggregation (ECAPA)-TDNN with a 1.69% EER (EER is defined as the point on the receiver operating characteristic (ROC) curve where the False Acceptance Rate (FAR) equals the False Rejection Rate (FRR) [71].) A study by Waibel *et al.* [72] used a TDNN for phoneme recognition that achieved a recognition rate of 98.5% compared to the HMM, which achieved 93.7%. Desplanques *et al.* [73] proposed the ECAPA-TDNN architecture that outperformed the state-of-the-art TDNN-based system on the VoxCeleb.

In speech, a TDNN is widely used for speech recognition and speaker identification. A TDNN with x-vectors is particularly successful for speaker identification [11], [74].

The TDNN model is used to model long-term temporal dependencies from short-term speech features [74]. In that model, all connections feed forward; TDNNs are comparable to multilayer neural networks in this regard [75]. In speech recognition, for a TDNN to be useful, the layered feed-forward neural network must have several properties [72].

- The network must have multiple layers, and the interconnections between units in each layer should be sufficient [72]. The reason for this is to ensure the network will have the ability to learn complex non-linear decision surfaces [72].
- The network must be able to depict the temporal links between events [72].
- The features learnt by the network should be invariant under translation in time [72].
- The label that must be learnt by the network should not be required to be in precise temporal alignment for the learning procedure [72].
- The network should be made to learn the training data by discovering patterns, which can be accomplished if the number of weights in the network is much less than the quantity of training data [72].

In speech, we want to look at frames that are a little into the future or the past. These past or future frames are referred to as contexts or delays. The process of looking at non-contiguous frames (skipping some of the frames) is referred to as dilation or subsampling. A TDNN consists of individual TDNN units, as illustrated in Figure 2.9.

The TDNN we will be looking at is essentially a 1D convolutional neural network without pooling and dilations [75]. The inputs to the TDNN network are the frames of the features we extract; the MFCCs or the Fbanks. The output is the probability distribution of each of the classes [76].

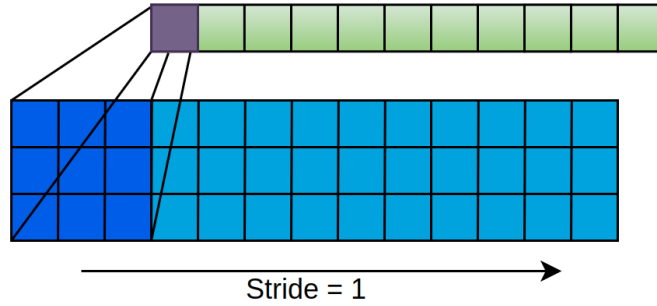


Figure 2.7: Visual illustration example of how no dilation is implemented.

If we look at the input as calculated mathematically in Section 2.4.3 and Section 2.4.2, we have an m -dimensional¹ input vector that contains the values of the MFCCs or Fbanks at time t [76]. The input vector is defined as $\mathbf{x}_t \in \mathbb{R}^m$. If we take these vectors and put them in a matrix, each vector represents one time step t of the signal [76]. If we observe the input over time, we then have the matrix of input features $\mathbf{X} \in \mathbb{R}^{m \times t}$ [76]. The kernel (trainable weight matrix) can be defined as $\mathbf{W} \in \mathbb{R}^{m \times l}$ [76]. In this case, the kernel has the same height (m) and width of l [76].

In Figure 2.7, we can see that the kernel $\mathbf{W}$ (the dark blue square) will slide over the input matrix $\mathbf{X} \in \mathbb{R}^{m \times t}$ (the light blue rectangle) with a stride of $s = 1$. The stride of s means that the kernel will make s steps at a time, before it is applied. The space on the input that the kernel covers is called the receptive field [76]. The output o (green rectangle) can be defined as the number of times the kernel can fit into the length of the

¹All mathematical equations follow the notation where scalars are italic letters, vectors are bold letters and matrices are capital bold letters.

input sequence. The following equation represents the output:

$$o = \lceil \frac{t - l + 2p}{s} \rceil + 1 \quad (2.10)$$

where t is the time step of the signal, l is the width of the kernel, p is the value of the padding, and s is the stride [76].

For a TDNN, the convolutional operation will be performed at each time step t . The convolution operation takes the element-wise multiplication of kernel weights over the input and sums those multiples together [76]. The results are then passed through a non-linear activation function ϕ (e.g. LeakyReLU). The output obtained after passing through the non-linear function can be defined as $z_q \in \mathbf{z}$, where $\mathbf{z}$ is the entire output vector after performing the activation function operation for each time step (the green rectangle in Figure 2.7) [76]. We can define the scalar output of a single element $z_q \in \mathbf{z}$ at time step $q \in \{1, 2, \dots, o\}$ with the following equation:

$$z_q = \phi(\mathbf{W} * \mathbf{X}_q + \mathbf{b}) \quad (2.11)$$

where the convolution operation is defined by the $*$ (asterisk) symbol, $\mathbf{b}$, is the bias that is added to a trainable neural network, and $\mathbf{X}_q$ is the input in the receptive field [76]. Bias refers to an additional parameter that is added to the weighted sum of inputs before applying the activation function in each neuron [77]. This bias term allows the neuron to adjust the decision boundary independently of the input data.

In the case of dilation, the convolutional summation will occur over only the input values that are under the kernel. This kernel now has gaps in its length, as seen in Figure 2.8.

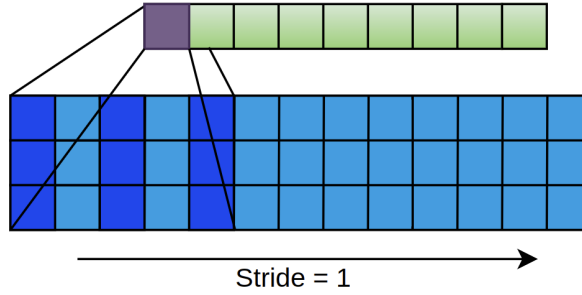


Figure 2.8: Visual illustration example of how dilation is implemented.

The deeper layers of a TDNN architecture handle hidden activations from a larger temporal context, while the early transformations are learnt on more limited contexts [74]. This indicates that higher layers may learn broader temporal correlations. As we reach higher layers in a TDNN, the temporal resolution at which each layer operates will also increase [74].

We will look at an x-vector system based on the network described in Table 2.2. This system includes layers that work on speech frames and a statistical pooling layer that aggregates representations at the frame level [78]. Furthermore, there are layers that operate at the segment level and ultimately a softmax output layer [78]. The softmax layer takes as input the raw output of the previous layer and transforms it into probabilities. This is done by the softmax function that applies the exponential function to the raw output and then normalising the resulting values to sum up to one. The output of the softmax layer can be interpreted as the model’s confidence or probability that the input belongs to each class.

The features are mean-normalised across a sliding window of up to 3 seconds and include 24-dimensional Fbanks with a frame duration of 25 ms. Let us consider that the input segments have T frames [78]. The architecture of the first five layers that operate on frame level is that of a time-delayed network [74]. Each of these frames has a small temporal context centred at the current frame t [11].

For example, we splice frames at the input at $\{t - 2, t - 1, t, t + 1, t + 2\}$ [11]. The next

two layers splice together the output of the previous layer at times $\{t - 2, t, t + 2\}$ and $\{t - 3, t, t + 3\}$, respectively [78]. In order to provide Frame 3 with a total context of 15 frames, this builds on the temporal context of the earlier layers [11]. The last two layers at the frame level do not have any additional temporal context [78].

The statistics pooling layer takes the output of the last frame level layer (Frame 5), aggregates over the input segment, and computes the mean and standard deviation [11]. For each input segment, a single 1,500-dimensional vector of statistics is produced. In order for future layers to act on the entire segment, this step accumulates information across the temporal dimension [11]. This can be seen in Table 2.2 with a layered context of $\{0\}$ and a total context of T . These statistics are concatenated together and propagated through the two segment-level layers [11]. The final layer through which it will propagate is the softmax output layer [78].

Table 2.2: The DNN architecture embedding. Before the nonlinearity, at layer segment 6, the x-vectors are extracted. The softmax layer’s N value reflects the number of training speakers.

Layer	Layer context	Total context	Input x output
Frame 1	$[t - 2, t + 2]$	5	120×512
Frame 2	$\{t - 2, t, t + 2\}$	9	$1,536 \times 512$
Frame 3	$\{t - 3, t, t + 3\}$	15	$1,536 \times 512$
Frame 4	$\{t\}$	15	512×512
Frame 5	$\{t\}$	15	$512 \times 1,500$
Statistic pooling	$[0, T)$	T	$1,500T \times 3,000$
Segment 6	0	T	$3,000 \times 512$
Segment 7	0	T	512×512
Softmax	0	T	$512 \times N$

SpeechBrain² is an open source speech processing toolkit that implements various neural processing models [12]. The model we use is a TDNN with x-vectors. More information on SpeechBrain is given in Section 2.5.3

²<https://speechbrain.github.io/>

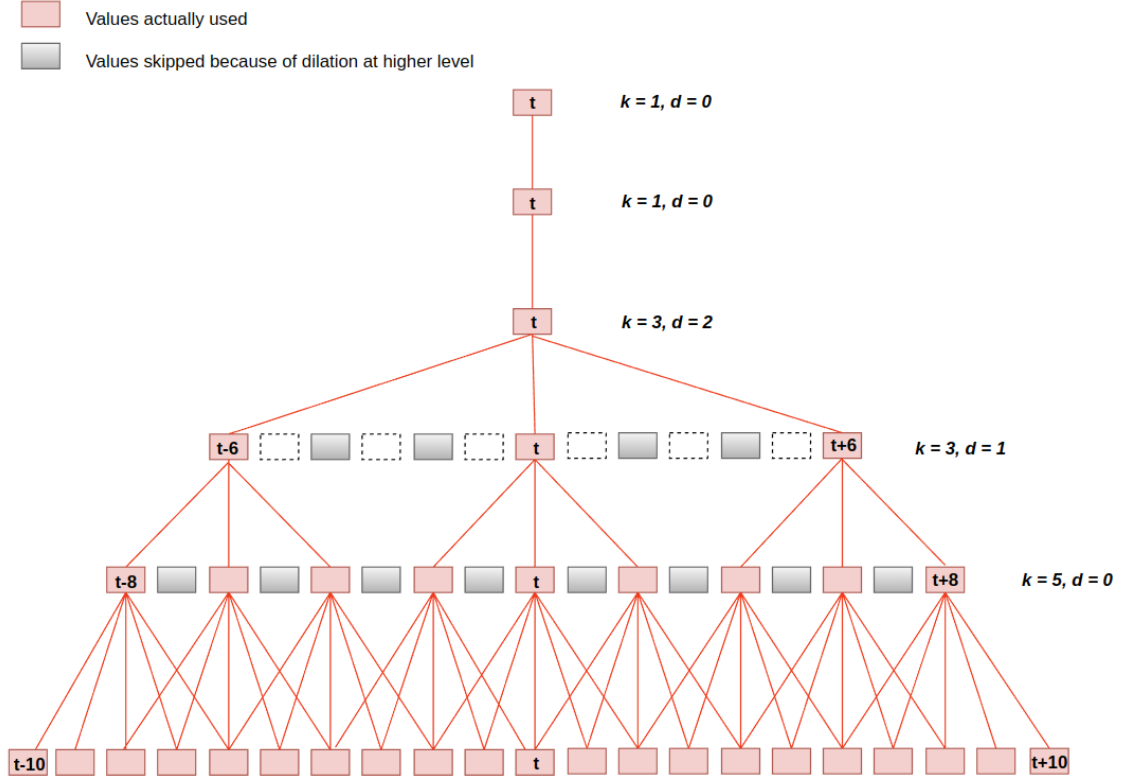


Figure 2.9: The TDNN unit that is used in this study. Each layer has its kernel size (k) and delay (d).

2.5.3 SpeechBrain

SpeechBrain is mainly created to speed up the research and development process of speech technologies. There are a variety of domains in which outstanding or competitive performance can be obtained [12]. It has a HuggingFace³ module, where SpeechBrain provides multiple pretrained models that can be quickly developed with nicely designed interfaces [12]. The key tasks of SpeechBrain are summarised in Table 2.3 [12].

³<https://huggingface.co/>

Table 2.3: List of speech tasks that SpeechBrain currently supports.

Task	Description
Speech recognition	Speech-to-text
Speaker recognition	Speaker verification/ID
Speaker diarisation	Detect who spoke when
Speech enhancement	Noisy to clean speech
Speech separation	Separate overlapped speech
Spoken language	Speech to intent/slots
Multi-microphone	Combining input signals

We focus on the speaker recognition part of SpeechBrain with a TDNN and x-vectors. To use SpeechBrain, the proposed data set must be adapted to the data manifest used by SpeechBrain. It will also be necessary to adjust the model to extract features needed for whale calls. More detail is given in Section 5.2.

The code for SpeechBrain is freely available on GitHub [12]. It is possible for anyone to make a copy of the code, create new features, report problems, or start a new conversation. [12]. The reason why we use SpeechBrain compared to other tools (like Kaldi or ESPnet) is that SpeechBrain is more modular and user-friendly with a focus on flexibility and ease of experimentation. Further SpeechBrain is built on PyTorch and provides higher-level abstractions specifically for speech-related tasks. We utilised a verified Python-based toolkit rather than implementing in Python from scratch.

2.6 Training of DNNs

In this section, we introduce what gradient descent is in Section 2.6.1. We also discuss some background on training parameters (Section 2.6.2) and training hyperparameters (Section 2.6.3).

2.6.1 Gradient descent

One of the most widely used algorithms for optimisation and by far the most popular method of optimising neural networks is gradient descent [79]. By updating the parameters in the opposite direction of the gradient of the objective function $\nabla_{\theta} J(\theta)$ with respect to the parameters, gradient descent is a technique to minimising an objective function $J(\theta)$ parameterised by a model's parameters $\theta \in \mathbb{R}$ [79]. The connection weights of the systems are adjusted to reduce the discrepancy between the expected outputs and the desired outputs of the system [80]. These weights represent the strength of connections between neurons in neural networks. During the training process, the model adjusts these weights based on the input data and the desired output, intending to minimise a chosen loss function. The model learns to make better predictions on new unseen data by adjusting the weights. Backpropagation uses an efficient way to calculate the gradient of the loss function with regard to weight values for the different inputs [80].

Adam, and Stochastic Gradient Descent (SGD) are two such gradient descent-based optimisation algorithms commonly used in the training of deep learning models [81]. Adam is an optimisation algorithm that considers the adaptive learning rates of all model parameters and adapts the learning rate to each parameter individually [82]. The algorithm combines the advantages of two popular optimisation algorithms, RProp and Adagrad, and has been shown to perform well in various deep learning tasks [82]. Adam uses moving averages of the gradients and squared gradients to provide a running estimate of the second raw moments of the gradients. These terms are used to compute adaptive learning rates [82].

SGD, on the other hand, is a simple optimisation algorithm that updates the model parameters by taking a step in the direction of the negative gradient of the loss function with respect to the parameters [79]. SGD can be seen as an approximation of batch gradient descent, where the gradient is calculated using only one batch (in some cases, one sample) at a time instead of the entire data set [79], [83]. In general, SGD is noisier than the usual gradient descent, and due to the randomness of its descent, it usually takes more steps to reach the minima [83]. Although it takes more steps to reach the minima

than the usual gradient descent, it is still computationally much less expensive [83].

2.6.2 Parameters

The various model coefficients introduced earlier are also referred to as the model parameters. The training algorithm optimises these coefficients during the learning process (in accordance with a predetermined optimisation strategy). It outputs an array of parameters that minimise the error [84].

As expected, initialising the parameters is an important action to take. The model's parameters are established before the commencement of the training process. The appropriate initialisation of these parameters plays an important role, influencing both the speed of convergence during training and the ultimate performance of the trained model [84]. As a result, depending on the activation function chosen, there are some suggestions to initialise parameters correctly [84].

He *et al.* hypothesised that Xavier initialisation and other schemes are not suitable for ReLU and extensions thereof [85]. Kaiming Initialisation, also known as He Initialisation, is a neural network initialisation technique that was developed for ReLU activations. The magnitudes of the input signals should not be reduced or increased exponentially as part of an appropriate initialisation approach [85].

To prevent this from occurring, He *et al.* propose the initialisation strategy:

$$\mathbf{w}_1 \sim N(0, \sqrt{\frac{2}{n_l}}) \quad (2.12)$$

For a given layer l with n_l input neurons, the He initialisation initialises the weights ($\mathbf{w}_1$) using a Gaussian distribution with a mean of 0 and a standard deviation of $\sqrt{\frac{2}{n}}$ [85].

2.6.3 Hyperparameters

Hyperparameters are key parameters that define the structure and behaviour of a machine learning model [86]. Additionally, the model will not update them according to the

optimisation plan; we will constantly need to manually intervene. [84].

The choice of hyperparameters greatly affects the performance of a deep learning model [87]. Setting the correct hyperparameters can lead to a well-performing model, while setting them poorly can result in underfitting or overfitting. As a result, finding the optimal hyperparameters is often a crucial step in the training process and can require significant experimentation and tuning [88].

Number of hidden layers

In a network, the hidden layer is located between the input and output layers. This layer is composed of a collection of small neurons and is responsible for transferring data and training layer information between the two layers [89].

In complex problems where accuracy and time complexity are the key limitations, hidden layers are crucial for neural network performance [89]. It is currently unclear how to determine the number of hidden layers and the number of neurons in each hidden layer [89].

Learning rate

The learning rate is a hyperparameter that can be changed and used to train neural networks. It typically has a small positive value between 0.0 and 1.0. This hyperparameter describes the backpropagation step, which involves updating parameters according to an optimisation function [84]. The learning rate or the step size refers to how often the weights are changed during each training iteration while moving toward a minimum loss function [90].

Faster training and greater test performance are initially made possible by a small initial learning rate; however, shortly after the learning rate is annealed, a large initial learning rate leads to superior generalisation [91]. At the same time, a learning rate that is too small might cause the process to stall [90]. In contrast to this, too high a learning rate can lead to the model converging too soon to an unsatisfactory result [90]. Carefully choosing

the learning rate is difficult when developing deep learning neural networks [90]. It could be the model's most crucial hyperparameter [90].

Activation function

An activation function is a mathematical function applied to the output of each neuron in a neural network [92]. It transforms the neuron's output into a value that is used as input to the next layer of the network [92]. Activation functions play a vital role in the behaviour and training of neural networks and are a crucial component of deep learning models [93].

Activation functions introduce non-linearity into the neural network, allowing it to learn complex relationships in the input data [92], [94]. Several activation functions include sigmoid, tanh, ReLU, and softmax [94]. The choice of activation function depends on the problem at hand and the desired properties of the model [95].

Batch size

The batch size is a hyperparameter in deep learning that determines the number of training examples used in one forward/backward pass [96]. A batch size of 1 means that the model updates its parameters after processing each training example, while a larger batch size means that the model processes multiple examples in one forward/backward pass before updating its parameters [96]. During the forward pass, the layer processes the input data and generates predictions for each example. After obtaining the predictions, the network then applies a loss function, which evaluates the extent to which the model's predictions align with the true target values. It quantifies the difference between the predicted values and the target values, providing a single scalar value that represents the performance of the model. The type of loss function to use depends on the machine learning problem. In the backward pass, the network computes the gradients of the loss function for each parameter. These gradients provide information on the necessary direction and magnitude of change to minimise the loss function. After computing the

gradients, the optimisation algorithm then adjusts the network parameters to minimise the loss function. This adjustment is made in the opposite direction of the gradient and is scaled by a learning rate hyperparameter. Using a larger batch size can be more computationally efficient [97]. It can result in faster convergence, directly averages out the noise and speed up the training process [97].

The optimal batch size for a given problem depends on several factors, including the size of the training set, the complexity of the model, and the computational resources available [98]. A common approach is to start with a smaller batch size and gradually increase it, monitoring the model's performance on the validation set to determine the best value [98]. It is also worth noting that batch normalisation is a popular technique in deep learning that normalises the activations of a layer across a batch of examples, to improve training stability [99]. A stable training process is when the loss value changes within a tolerance, having an acceptable variance across the training period.

Epochs

In machine learning, an epoch is a term used to describe a single pass over the entire training data set during the training process of a machine learning model [96]. During an epoch, the model examines the entire data set and updates its parameters, such as weights and biases, to reduce the loss and improve its accuracy [96].

The number of epochs is an important hyperparameter that can impact the model performance. A larger number of epochs can help the model learn more complex relationships within the data, but can also increase the risk of overfitting [100]. On the other hand, a smaller number of epochs may lead to underfitting, where the model fails to capture the underlying patterns in the data [100].

Dropout

Dropout is a regularisation technique used in machine learning to prevent overfitting in neural networks [101]. Regularisation is a technique that aims to reduce the test error even though it may not necessarily improve the training error. One way of doing this is to impose extra constraints or penalties on the parameters [102]. Its purpose is to discourage complex or unpredictable behaviour in the model, thereby encouraging the development of simpler and more generalisable models [102]. Overfitting occurs when a model performs well on the training data but does not generalise well to new unseen data [103].

Dropout works by randomly dropping out, or ‘turning off’, some of the neurons in the network during training [104]. This means that these neurons will not contribute to the forward or backward pass during that particular iteration of the training. By dropping out some of the neurons, the model becomes less sensitive to individual neurons’ specific weights and activations, which can lead to better generalisation and reduced overfitting [105].

2.7 Optimisation tools

This section first introduces the idea behind automated selection of hyperparameters, specifically, grid searches and Bayesian optimisation. Then, specific tools are introduced that accelerate the hyperparameter optimisation process. There are different tools that we can use for hyperparameter optimisation. The two that are relevant to this study are first Optuna (Section 2.7.2) for the CNN and Orion (Section 2.7.3) for SpeechBrain. Optuna was selected because it is the easiest to use among the three tools considered: (1) Ray Tune, (2) Hyperopt, and (3) Optuna [106]. Orion is the default hyperparameter optimisation tool made available with SpeechBrain.

2.7.1 Automated hyperparameter tuning

Automated selection of hyperparameters refers to the process of automating the search for the optimal hyperparameters of a model. We will look at two approaches for automating hyperparameter selection: grid searches and Bayesian optimisation.

Grid search is a brute-force approach in which a set of hyperparameters is selected, and the model is trained and evaluated for all possible combinations of these hyperparameters [107]. The model's performance is then measured, and the combination of hyperparameters that yields the best performance is selected as the optimal choice [107]. Grid search is simple but computationally expensive, especially if the search space is ample [107].

On the other hand, Bayesian optimisation is a more efficient and intelligent approach that uses a probabilistic model to estimate the performance of different hyperparameter configurations [108]. Bayesian optimisation iteratively chooses new hyperparameters to evaluate based on the results of previous evaluations, with the goal of finding the combination of hyperparameters that minimises/maximises the objective function [108]. This approach can be more efficient than grid search, especially when the search space is large and complex [108], [109].

Both grid search and Bayesian optimisation are widely used techniques for the automated selection of hyperparameters. The choice between these two approaches depends on the problem at hand, the complexity of the search space, and the available computational resources.

2.7.2 Optuna

Optuna is specifically designed for machine learning [110]. Optuna automatically searches for and finds optimal hyperparameter values by trial and error. In Optuna, hyperparameter values are chosen based on the history of previous trials [111]. These historical data are used to estimate a promising area and try values in it [106]. Optuna then estimates an even more promising region based on the new results. The process is repeated using

the history data of previous trials [112]. In particular, it utilises a Bayesian optimisation algorithm called Tree-Structured Parzen Estimator (TPE) [110].

Optuna also has a pruning algorithm, a technique used to reduce the size of decision trees in machine learning and search algorithms [106]. The pruning reduces the decision tree's size because non-critical and redundant instances of the tree sections are removed [106]. The pruning in Optuna is like an automated early-stopping that stops unpromising trials at early stages of the training [111]. Automated early stopping is a technique used in machine learning, to prevent overfitting and improve efficiency by terminating the training process when certain criteria are met [113]. If the performance of the model on the validation dataset stops improving or starts deteriorating, it indicates that the model is likely overfitting the training data. Automated early stopping eliminates the need for manual intervention in determining when to stop training. Instead, it allows the training process to automatically terminate when the model's performance on the validation dataset indicates that further training is unlikely to improve generalisation performance.

The basic concept of Optuna uses the terms study and trial. These terms are defined as follows:

- **Study:** optimisation based on an objective function.
- **Trial:** single execution of an objective function in each study [111].

A study aims to find the optimal hyperparameter value set through multiple trials. In the objective function, the user specifies the parameter that needs to be minimised/maximised and the hyperparameters that need to be searched over to minimise/maximise the selected parameter. Furthermore, the ranges of the hyperparameter search need to be specified, and Optuna will then suggest a value in that range for each trial.

2.7.3 Orion

Orion is an asynchronous framework for optimising black-box functions [114]. This tool serves as a meta-optimiser for machine learning models and training and a flexible platform for large-scale asynchronous optimisation.

Benefits of Orion are that it is effortless to adopt and can be extensively customised [115]:

- A single line of code is needed to adapt Orion to work with SpeechBrain.
- Orion is asynchronous, which makes it robust and simple to parallelise.
- Implements a variety of hyperparameter optimisation algorithms (see Table A.1 and Table A.2).
- Search-space definitions are easy to use.
- The default settings for the configuration system are complete.
- Orion uses a local or remote database.

2.8 Evaluation metrics

The evaluation metric commonly used for classification problems is accuracy, and for detection problems they are precision, recall, and F1 score. A confusion matrix is used for both classification and detection problems. Classification is a task in machine learning where the goal is to assign a label or category to an input based on its features. Detection is a task in machine learning in which the goal is to identify and localise objects within an input, along with classifying them.

The rationale behind the selection and definition of the specific metrics in this section is based on the findings of the literature review conducted in this study. It was concluded that these metrics are the primary ones used to evaluate classification and detection models.

2.8.1 Accuracy

Accuracy is a very simple performance metric in which the ratio between the number of correctly classified points and the number of total points is calculated [116]. Accuracy has a disadvantage when used with an imbalanced data set [116]. If the model classifies all data points as the majority class, the accuracy will be high, even though the model itself has no true classification ability [116].

2.8.2 Confusion Matrix

A confusion matrix is a summary of predicted results displayed in specific table layouts to allow visualisation of the performance measure for a binary classification problem (two classes) [116].

Binary class problems

This is represented as a table or matrix with four different combinations of predicted and actual values, as seen in Figure 2.10 [117]. The confusion matrix is useful when measuring recall, precision, and accuracy.

Figure 2.10 shows these combinations as True Positive (TP), False Positive (FP), False Negative (FN), True Negative (TN). For detection tasks, the confusion matrix is typically represented by a positive and a negative class. The positive class typically represents the presence of an event, whereas the negative class represents the absence of an event.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 2.10: Confusion Matrix representation for binary class.

Here is a simple definition for the following [116]:

- **True Positive:** When the predicted value and the actual value are positive.
- **True Negative:** When the predicted value and the actual value are negative.
- **False Positive: (Type 1 error)** When the predicted value is positive and the actual value is negative.
- **False Negative: (Type 2 error)** When the predicted value is negative and the actual value is positive.

2.8.3 Precision and recall

Here we define precision and recall:

-
- **Precision** is defined as the fraction of true positives over the number of true positives plus false positives [116]. Precision helps to understand how useful the results are [116].

$$Precision = \frac{TP}{TP + FP} \quad (2.13)$$

- **Recall** is defined as the fraction of true positives over the number of true positives plus the number of false negatives [116]. Recall helps to understand how complete the results are [116].

$$Recall = \frac{TP}{TP + FN} \quad (2.14)$$

2.8.4 F1-score

A model's F1-score is the harmonic mean of its precision and recall, and it defines the precision and recall together as a single metric [116]. A perfect model has an F1-score of 1. The formula for the F1-score can be seen in Equation 2.15 [116].

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \times \frac{precision \times recall}{precision + recall} = \frac{2TP}{2TP + FP + FN} \quad (2.15)$$

2.9 Conclusion

This chapter discusses fundamental concepts that are used throughout this study for whale call detection: whale vocalisation, work related to whale call detection, and different feature extraction techniques. We also looked at the two DNNs that will be used (1) CNNs and (2) TDNNs with x-vectors. We discuss the training of DNN and the importance of parameters and hyperparameters. Lastly, we discuss the various evaluation metrics that are used for whale call detection. The following chapter introduces the original data set, the preprocessing thereof, and the data set we created from the original data set for experimentation.

Chapter 3

Data set and preprocessing

We introduce the AADC data set that is used for this study. The AADC data set is an open source corpus that contains calls from blue and fin whales. We describe the corpus and the preprocessing steps and provide some useful statistics about the corpus.

3.1 Overview

We describe the AADC corpus, which provides the source data for this study, and discuss how it is processed for experimentation. We describe the original AADC corpus in Section 3.2. In Section 3.3, we show the preprocessing that is done on the original AADC data set. Finally, we conclude with the data sets we use for experimentation in Section 3.4.

3.2 AADC corpus

In this section, we take a closer look at the AADC data set by performing some statistical analysis and discussing problems we encountered with the corpus. We look at some statistics for the original AADC corpus to see the data distribution in Section 3.2.1. After that, we look at the problems we encounter with the original AADC corpus in Section 3.2.2. Finally, in Section 3.2.3, we create the subset we can use for further processing.

We will use the ‘Acoustic trends for the blue and fin whale library’ from the AADC. This library has been designed to allow automated algorithms to be applied to a selection of recordings that have been collected by the Acoustic Trends Working Group of the International Whaling Commission Southern Ocean Research Partnership (IWC-SORP) [13].

In the recorded data, there were four different call types for Antarctic blue whales and three different call types for Antarctic fin whales [13]. A detailed description of the annotated calls can be found in Table 3.1 [13]. The genus and species is indicated by the first two letters, where *Bm* is for the blue whales, and *Bp* is for the fin whales. The *Ant* is for subspecies; in this case, Antarctic blue whales. The rest of the classification represents the specific type of call. See Table 3.1 for the full description of each call type.

Table 3.1: Summary of the different call types and their ID for blue and fin whale sounds in the IWC-SORP library. [13].

Call ID	Call type	Description
BmAnt-A	Antarctic blue whale unit A	Tone with a constant frequency between 25 and 28 Hz without other units. It is usually repeated every 60–70 seconds.
BmAnt-B	Antarctic blue whale unit AB	For the Antarctic blue whale, the unit A tone comes after the inter-tone down-sweep (unit B).
BmAnt-Z	Antarctic blue whale z-call; (AKA 3 unit vocalisation)	Upper and lower tonal units A and C in a z-call made by an Antarctic blue whale (and down-swept unit B either present or absent).
Bm-D	Blue whale FM (AKA D-calls)	Frequency-modulated down-swept cry of a blue whale. While this sound often lasts longer and is a lower frequency than FM calls from fin and minke whales, it is not always the case.
Bp20Hz	Fin whale 20 Hz pulse	Fin whales pulse at a frequency of 20 Hz; higher frequencies lack substantial energy.
Bp20Plus	Fin whale 20 Hz pulse with energy at higher frequencies (e.g. 89 or 99 Hz components)	A fin whale’s 20 Hz pulse with supplementary energy at higher frequencies (e.g. upper-frequency peak near 80-10 and Hz)
Bp-Downsweep	Fin whale FM calls (AKA ‘high frequency’ down-sweep; AKA 40 Hz pulse)	It is believed that fin whales call in a frequency-modulated, often down-swept manner. Compared to FM sounds from fin whales, blue whale cries are often shorter and have a somewhat higher frequency.
Unidentified	Unidentifiable sounds	Any biological sound that be accurately classified as one of the above mentioned. This featured noises that may be biological, but were quite distinct from the categories mentioned above. Additionally, there were FM down-sweeps which the analyst was unsure whether they were fin whale down-sweeps or blue whale D-calls.

The data set consists of 1,880.25 hours of audio duration across 11 different sites. The recording years were 2005, 2013, 2014, 2015 and 2017. In total, there were 105,161 annotations across all sites [13].

Table 3.2: Distribution and number of annotations at each site by classification type for the blue whales as reported by the AADC [13].

Site-Year	BmAnt-A	BmAnt-B	BmAnt-Z	Bm-D
Maud Rise 2014	2,189	37	28	70
G64 2015	827	157	29	66
S Kerguelen 2005	812	237	166	435
S Kerguelen 2014	2,557	1,177	563	435
S Kerguelen 2015	1,970	542	236	1,180
Casey 2014	3,681	1,398	1,091	679
Casey 2017	1,741	558	119	553
Ross Sea 2014	104	0	0	0
Balleny Island 2015	923	44	31	47
Elephant Island 2013	2,625	1,786	152	299
Elephant Island 2014	6,934	967	100	1,034
Total	24,363	6,903	2,515	4,798

Table 3.3: Distribution and number of annotations at each site by classification type for the fin whales as reported by the AADC [13].

Site-Year	Bp20Hz	Bp20Plus	Bp-Downsweep	Unidentified
Maud Rise 2014	23	5	6	464
G64 2015	2	1	46	325
S Kerguelen 2005	788	78	444	3,061
S Kerguelen 2014	1,920	1,826	344	4,961
S Kerguelen 2015	552	718	344	1,244
Casey 2014	17	0	0	5,648
Casey 2017	78	214	0	130
Ross sea 2014	0	0	0	255
Balleny Island 2015	951	148	78	18
Elephant Island 2013	3,662	1,859	1,042	22,927
Elephant Island 2014	4,940	2,912	3,660	890
Total	12,933	7,761	5,964	39,924

3.2.1 Statistics on data set

In this section, we will look at some statistics of the AADC data set to: (a) better understand the distribution of call types in the data set, and (b) get the highest and lowest frequencies observed in the data set, as well as the maximum and minimum duration observed for the calls.

The average, standard deviation, maximum, and minimum are calculated on the entire data set's call lengths (the duration of each call). High and low frequencies are just the highest and lowest frequencies recorded in the entire data set. We will plot the probability of a call duration to see where the most calls are grouped and what call types are in that grouped area. In Table 3.4, we show a summary of the statistics for the entire data set.

Table 3.4: Average, standard deviation, maximum and minimum for the call duration as well as the highest and lowest frequency call for the entire AADC data set.

Statistic	Value	Unit
Average duration	5.92	seconds
Duration standard deviation	18.55	seconds
Maximum duration	959.92	seconds
Minimum duration	0	seconds
Highest frequency	1,024	Hz
Lowest frequency	0	Hz

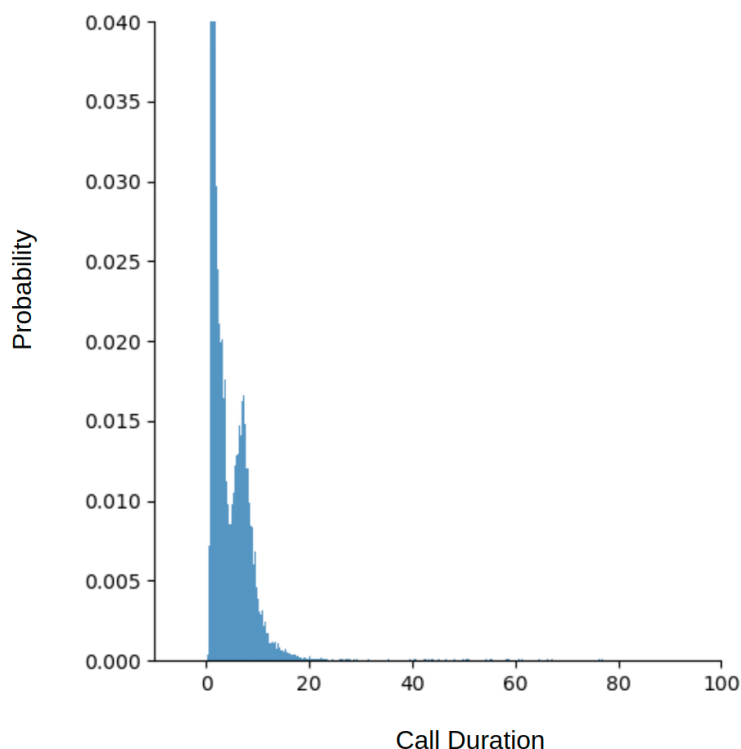


Figure 3.1: Distribution of all the call durations from the AADC data set.

The call duration is more compactly distributed from 0 to 20 seconds, as seen in Figure 3.1 and Table 3.4. The reason for the very long call durations is that humpback and minke whales have very long calls, and some of the unidentified sounds are long as well. We would like to use the call duration distributions that are grouped the most (around the blue and fin whale calls). Table 3.5 and Figure 3.2 showed us the statistics and distribution when we considered calls from only blue and fin whales. From this we can see that the data are

more grouped, meaning that the call durations have a higher similarity in length.

Table 3.5: Average, standard deviation, maximum and minimum for blue and fin whale call duration as well as the highest and lowest frequency call in the AADC data set.

Statistic	Value	Unit
Average duration	3.74	seconds
Duration standard deviation	2.85	seconds
Maximum duration	23.14	seconds
Minimum duration	0.29	seconds
Highest frequency	134.10	Hz
Lowest frequency	8.10	Hz

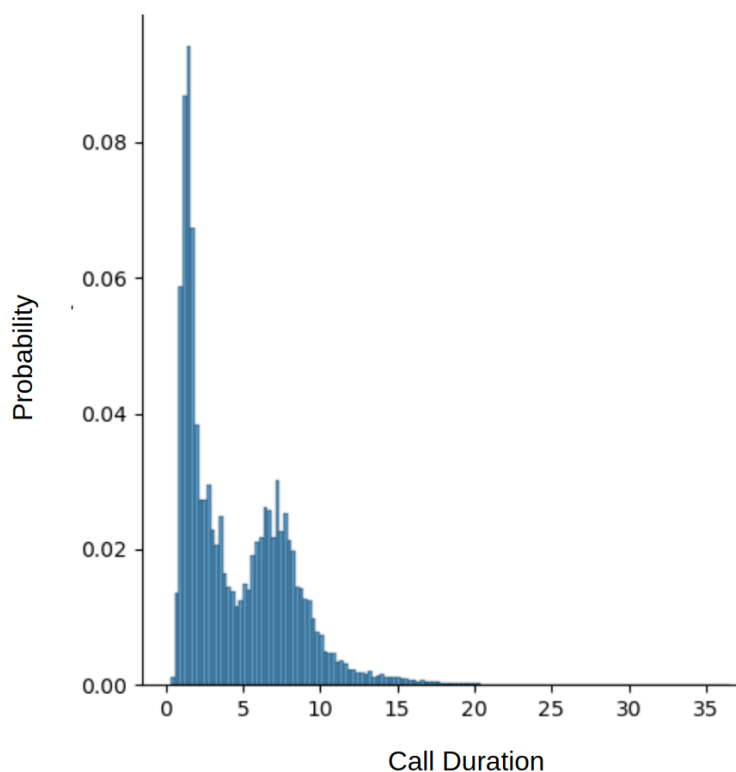


Figure 3.2: Distribution of blue and fin whale call duration from the AADC data set.

To conclude, we will create a subset from the entire AADC data set that we can use for experimentation. This subset will include only calls from blue whales and fin whales. We have also identified potential errors in Table 3.4, where the frequency and the minimum duration of calls are zero. We believe these could be mistakes, which we will investigate in the following section.

3.2.2 Problems encountered in corpus

In this section, we look at two different problems we encounter with the original AADC corpus: (a) mislabelled calls, and (b) negative sample durations. A Python script was written to check each annotation file to see if the *sample end* is larger than the *sample start*. In addition, other annotations are also added to the samples in some of the annotation files. This made us unsure as to which annotation to use.

When searching for all samples where the *Beg File Samp (samples)* is larger than *End File Samp (samples)*, the following segments are found in Table 3.6.

Table 3.6: Summary of the number of problems encountered per site in the AADC data set that had a mismatch in their sample start and sample end.

Site	Number of samples
BallenyIslands 2015	0
Casey 2014	5
Casey 2017	0
ElephantIslands 2013	56
ElephantIslands 2014	2
Greenwich64 2015	0
Kerguelen 2005	1
Kerguelen 2014	2
Kerguelen 2015	3
MaudRise 2014	1
RossSea 2014	0

When searching for all samples that have duplicate labels (the sample in an annotation file has another annotation inside the annotation file that is different from the specific annotation file), the following segments are found as shown in Table 3.7:

Table 3.7: Summary of labelling problems encountered in the AADC data set.

Site	Annotation file	Annotation given in file	Number of samples
BallenyIslands 2015	Humpback	Unid	20
Casey 2014	Unidentified	Bp20Hz	642
Casey 2017	BmAnt-A	BmAnt-B	3
	BmAnt-B	BmAnt-A	19
	BmAnt-B	Bm-D	1
	Bm-D	BmAnt-B	2
	BmAnt-Z	BmAnt-A	3
	BmAnt-Z	BmAnt-B	5
	Unidentified	BmAnt-A	1
ElephantIslands 2013	N/A	N/A	0
ElephantIslands 2014	N/A	N/A	0
Greenwich64 2015	N/A	N/A	0
Kerguelen 2005	N/A	N/A	0
Kerguelen 2014	Unidentified	Bp20Hz	17
Kerguelen 2015	Unidentified	Humpback	174
MaudRise 2014	BmAnt-A	Bp20Plus	10
	BmAnt-A	Unidentified	120
	BmAnt-A	Bm-D	3
	BmAnt-B	BmAnt-A	29
	BmAnt-B	Bm-D	1
	BmAnt-Z	BmAnt-A	27
	Minke	BmAnt-A	1
	Minke	Unidentified	1
	Unidentified	Bp-Downsweep	3
	Unidentified	BmAnt-A	1
	Unidentified	Minke	1
RossSea 2014	N/A	N/A	0

Examples

Some of the begin and end samples are wrong; for example, in Casey 2014 WAV file 200_2014-02-01_09-00-00.wav, the AADC annotated that a BmAnt_A call should start at 3,596 seconds and end at 0.147 seconds. We encountered 70 other samples with the same problems. Another problem we encountered was that other annotations were found in the specific call type annotation's file; for example, the MaudRise 2014 WAV file 20140112_110000_136A0020_04.wav was in the BmAnt_A annotated file, but inside the researcher had labelled it as an unidentified call. This makes it quite confusing to know which label given by AADC is correct, and there was too much data to check everything individually. We encountered 1,084 of these problems.

3.2.3 Subset selection of AADC

The goal of this section is to create a subset from the AADC data set to use from this point forward. The three sites from which we chose to create a subset are Balleny Island 2015, Elephant Island 2013 and Elephant Island 2014. The reason for choosing these sites is that they had fewer problems than the other sites. They also added most of the blue and fin whale calls. The subset of these three sites contained around 45,152 annotations of blue and fin whale calls. In Table 3.8 and Figure 3.3, we can see the statistics and distribution of the subset created from these three sites.

Table 3.8: Average, standard deviation, maximum and minimum for the blue and fin whale call duration as well as the highest and lowest frequency call in the AADC data set over three selected sites.

Stats	Value	Unit
Average duration	3.75	seconds
Duration standard deviation	2.85	seconds
Maximum duration	23.15	seconds
Minimum duration	0.29	seconds
Highest frequency	134.10	Hz
Lowest frequency	8.50	Hz

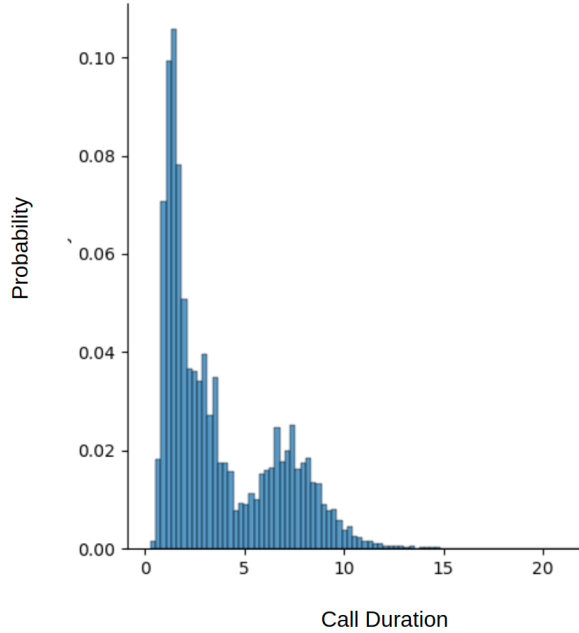


Figure 3.3: Distribution of blue and fin whale call durations from three selected sites in the AADC data set.

From this, we felt that chunks of four seconds would work well, but chunk length is not fixed. We created the subset primarily to obtain a more condensed call duration distribution for ease of handling. Additionally, if we were to utilise all the data during the training cycle, it would be excessively time-consuming.

3.3 Preprocessing

In this section, we look at the preprocessing required for the subset created in Section 3.2.3. We first look at the preprocessing that is the same for both CNN and SpeechBrain (Section 3.3.1). Next, we examine the additional preprocessing that is needed for the CNN (Section 3.3.2) and SpeechBrain (Section 3.3.3), respectively.

3.3.1 Preprocessing for both methods

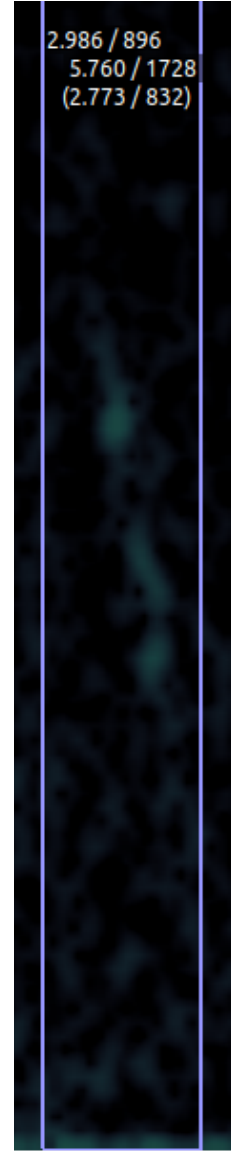
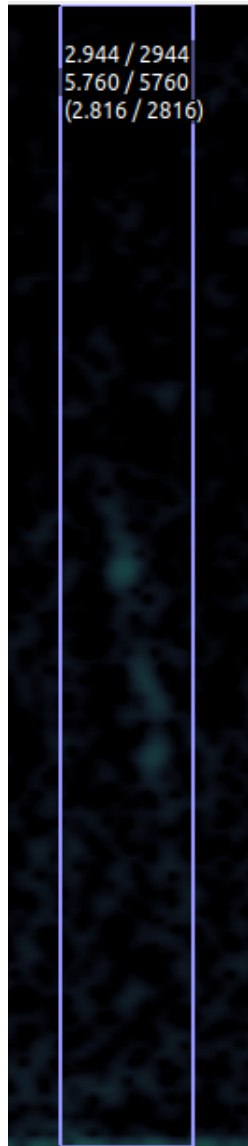
This section describes the preprocessing that is the same for both methods that we use during the study. The audio files are recorded at different sampling rates. Balleny Island 2015 is recorded at a sampling rate of 1,000 Hz, while Elephant Island 2014 and Elephant Island 2015 are recorded at sampling rates of 250 Hz. The highest recorded call from blue and fin whales was observed at approximately 135 Hz. In line with the Nyquist theorem, we resample all three sites to 300 Hz. All three sites were resampled at 300 Hz using the Secret Rabbit Code (SRC)¹. The sampling rate is the number of samples taken per unit of time from a continuous signal to convert it into a discrete signal. Downsampling, also known as decimation, is the process of reducing the sampling rate of a signal by discarding some of the samples while preserving essential information [118]. Upsampling, also known as interpolation, is the process of increasing the sampling rate of a signal by inserting additional samples between existing ones [119]. The Nyquist theorem, also known as the Nyquist-Shannon sampling theorem, states that to accurately reconstruct a continuous signal from its samples, the sampling rate must be at least twice the highest frequency present in the signal.

In the SRC library, the default interpolation method used for upsampling is typically a high-quality sinc-based interpolation. Sinc interpolation is a well-known method for achieving high-quality sample-rate conversion, especially in applications where preserving signal fidelity is crucial. Sinc interpolation involves applying a sinc function as the interpolation kernel, where the sinc function is the ideal interpolation function for band-limited signals. This method provides accurate reconstruction of the original signal, particularly when combined with appropriate filtering to prevent aliasing.

The following example will demonstrate that the calls are still at the same time stamp. We use 20150115_170000.wav from the Balleny Islands as an example. The AADC annotated that they found a call between sample 2,989 and 5,753 in the original WAV file at 1,000 Hz, which means that the call occurred between 2.989 and 5.753 seconds. As we can see from the images in Figure 3.4, the call still exists within the annotated time. We can

¹<http://www.mega-nerd.com/SRC/quality.html>

conclude that the resampling worked and did not move the call out of the time stamp; it was recorded and annotated.



(a) Original 1,000 Hz spectrogram of a whale call between 2.989 and 5.753 seconds. (b) Resampled 300 Hz spectrogram of a whale call between 2.989 and 5.753 seconds.

Figure 3.4: Comparison between 1,000 Hz and 300 Hz of 20150115_170000.wav to see if the call is still within timestamp.

3.3.2 Additional preprocessing for CNN

This section describes the additional preprocessing necessary for CNN. After resampling, we extracted the spectrograms using Python. The spectrograms are extracted as described in Section 4.4.3. The spectrograms are divided into smaller chunks of two and four seconds in duration. The samples are divided into two classes: ‘calls’, and ‘no-calls’. We use the annotations of the AADC to help us decide if the four-second chunk contains a call or not. If the chunk contains more than 30% of a call, it gets labelled as ‘calls’; otherwise, it gets labelled as ‘no-calls’. The no-call samples consist of background noise, other whales, and biological sounds.

We will show in later sections (Sections 4.2 and 4.5) that the CNN is capable of learning features and patterns from spectrograms even when only a portion of the call is present. The network’s ability to correctly identify a call may vary based on the specific characteristics of the call and the surrounding context in the spectrogram. In this case, the non-target sounds are relatively low in volume and do not overlap significantly with the target sound. Furthermore, the targeted sound of the blue and fin whales is prominent and not that complex, making it easier for the CNN to identify the whale calls.

3.3.3 Additional preprocessing for SpeechBrain

This section describes the additional preprocessing necessary for SpeechBrain. Only limited additional data processing was required. The resampled Waveform Audio File Format (WAV) files were segmented into smaller WAV files of two and four seconds duration. If the WAV file contained a whale call, it gets appropriately tagged (‘call’) otherwise it gets tagged as (‘no-call’). The data manifests are created from these files (more details are given in Sections 5.2 and 5.4).

3.4 Data sets for experimentation

In this section, we want to finalise the data set we will use for the training, validation, and testing of the models from this stage onwards. This allows us to evaluate the performance of our model on unseen data. By keeping a separate testing data set, we can measure how well our model generalises to new, unseen examples. This is crucial to assess the performance in the real world of a model. The validation set is used to adjust hyperparameters and select the best-performing model. By using a validation set, we can experiment with different hyperparameter values and select the ones that result in the best performance. Since the validation set has been used during tuning, it will provide an over-confidence estimate of the model’s performance. The test set provides a more realistic estimate of the model’s performance.

From Section 3.2.1, we concluded that we would only use the blue and fin whale calls for our data set and in Section 3.2.3 we decided to limit the data to three sites. We are limiting the data to the three selected sites because if we use all of the data we will run into a very long training cycle, and the other nine sites have numerous problems.

We selected two data sets for experimentation: a smaller one and a larger one. The smaller set is selected for CNN training because it is computationally more expensive to run CNN experiments. We selected as much data as we could process using the available infrastructure. The larger set is used for SpeechBrain training because the training cycle is computationally less expensive. The small and large sets are partitioned into a training, validation, and test set each, using the ratio 70:10:20. The size, number of calls, and overall duration of the two sets are shown in Table 3.9. Since more data are available than we can process, we construct a balanced data set: we first select the available calls and then select background data (‘no-calls’) of the same duration randomly from the available background data. A significant portion of the background data is discarded in the process. The audio files used for training, validation, and testing are all independent audio files, which means that a test audio file is not used to get the training or validation samples and vice versa.

The division of the two data sets into training, validation, and testing sets can be seen in Table 3.9. Here, small and large define the size of the data set, the number three (3) indicates the three sites we use, and the letters ‘cs’ are a placeholder for the chunk size used.

Table 3.9: Size of data sets used for experimentation.

ID	Set	Calls	No-Calls	Total	Duration (hours)
small_3_cs	Training	8,069	8,069	16,138	17.93
	Validation	1,153	1,152	2,305	2.56
	Testing	2,306	2,305	4,611	5.12
large_3_cs	Training	29,782	29,782	59,564	109.26
	Validation	4,314	4,314	8,628	15.61
	Testing	8,470	8,469	16,939	31.22

3.5 Discussion

This chapter introduced the original AADC corpus, where we show the number of sites contained in the corpus, the years the audio is recorded, and the number of annotations for each call or biological sound. We analysed the AADC corpus and defined the precise data sets that will be used in the rest of the study. We found some interesting statistics where the minimum duration of a call annotated is zero seconds. This raised some questions about whether there might be problems in the whole AADC corpus. The problems we encountered with mislabelled calls and negative sample durations in the original AADC corpus were confirmed. This analysis helped us to create a subset from the original corpus. The subset then went through some preprocessing that is the same for both cases and then further preprocessing that is specific for the CNN and SpeechBrain. Moving forward, the two data sets, ‘small_3_cs’ and ‘large_3_cs’, with sizes as shown in Table 3.9, are the data sets that are used for experimentation in the rest of this study.

Chapter 4

CNN baseline

In this chapter, we train a CNN on the small_3_cs data set to develop a baseline for comparing it to the performance of SpeechBrain in Chapter 5. We describe the extraction of the spectrogram, the experimental setup, discuss the optimisation protocol and report the results.

4.1 Overview

In this chapter, we use an established technique for whale call detection using a CNN trained on spectrograms. This will act as a baseline system which will be compared to the novel neural system developed in Chapter 5.

Section 4.2 explains the baseline CNN architecture, both its overall design and individual components. Furthermore, Section 4.3 outlines the optimisation protocol employed to achieve the best possible performance for the baseline CNN model. Our investigation into spectrogram extraction consists of the examination of different tools, as discussed in Section 4.4. We evaluate the feasibility of the different tools for our purposes to find the

most suitable solution. The experiments conducted are discussed in Section 4.5 with an analysis of the results. Finally, Section 4.6 concludes with a summary and key findings of this chapter.

4.2 CNN architecture

We use a standard CNN consisting of two convolutional layers and one fully connected layer. We chose this setup because this is a standard minimal setup [120], and we use this as a starting point before determining whether larger architectures are called for. The input layer to the CNN is the spectrograms from the audio file that will be fed to the network as fixed-size tensors. The output layer consists of two units that indicate the predicted class ('call' or 'no-call') for the last fully connected layer. Each hidden layer consists of a convolutional layer, batch normalisation, dropout, a series of kernels, each with a size 2×2 and a max pooling layer. In Figure 4.1, the architecture of the CNN can be seen, and more information about the architecture is listed.

ReLU is used as an activation function since it has been shown to accelerate the learning of DNNs and tends to achieve good results in general [121]. We used cross-entropy loss, the most popular activation function for classification tasks. Cross-entropy loss is a versatile and effective choice for training CNNs, particularly in classification tasks, due to its compatibility with softmax outputs, ease of optimisation, and ability to encourage well-calibrated predictions.

The network is initialised with a uniform Kaiming scheme [85], which is the default choice for ReLU-activated networks. The CNN is optimised using either Adam or SGD as an optimiser. In addition to SGD, Adam is chosen for its adaptive estimates of lower-order moments [122].

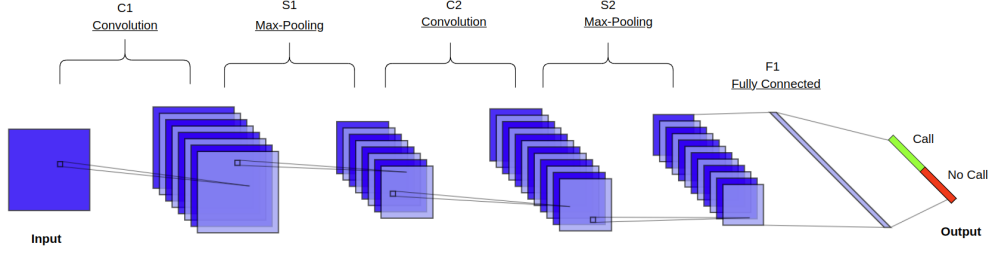


Figure 4.1: The architecture of the baseline CNN used in this study.

The different stages and layers for the CNN baseline architecture as in Figure 4.1 is defined as follows:

- Input is the spectrogram chunks that are fed to the CNN as tensors.
- C1 is a convolutional layer that contains four input channels (see Section 2.4.1 for more detail), n output channels and a kernel size of 2×2 . The convolution step (stride) is 1 with no padding. A ReLU function is used as the activation function of the output.
- Batch-norm is applied to C1.
- S1 is a pooling layer with a maximum pooling strategy with a pooling size of 2×2 .
- C2 is a convolutional layer containing n input channels, $n \times 2$ output channels and a kernel size of 2×2 . The convolution step (stride) is 1 with no padding. A ReLU function is used as the activation function of the output.
- Batch-norm is applied to C2.
- S2 is a pooling layer with a maximum pooling strategy with a pooling size of 2×2 .
- F1 is a fully connected layer containing k neurons, and each neuron is fully connected to all the output units of layer C1. The ReLU function is used as the activation function.

-
- The output is a fully connected layer containing two neurons, and each neuron is fully connected to all the output units of layer F1. This layer also gives the predicted class.

4.3 Optimisation protocol

The CNN as described in Section 4.2 is trained on the spectrogram tensors from the `small_3_cs` (Section 3.4) data set. Hyperparameter optimisation is done using the optimiser Optuna¹, which finds optimal hyperparameter values using different meta-optimisation algorithms. We use a TPE [110] as a Bayesian optimisation algorithm. The learning rate is a step learning rate schedule, which involves decaying the learning rate of each parameter by a factor of gamma every n epochs. The batch size, learning rate, dropout level, the convolution layers' number of in and out channels, and the depth of the fully connected layers are varied for each trial based on what Optuna selects. The number of epochs was kept at 30 because the training loss always converged before 30 epochs, and Optuna's pruning was implemented, which stopped an unpromising trial. The goal is to minimise the validation loss, which means that the model is making accurate predictions and performing well on the validation data; that is why Optuna is configured to minimise the validation loss. We also implemented early stopping by selecting the model with the best validation loss in the training cycle and saving that model.

The initial hyperparameter ranges that we optimised are shown in Table 4.1. After the first full Bayesian optimisation run, the hyperparameter values are checked to see if the hyperparameter values that gave the best results were within the initial ranges. If the hyperparameters were within the initial ranges, the model was saved. If the hyperparameter values were on the edges of the initial ranges, we adjusted the range and reran the Bayesian optimisation.

¹<https://optuna.org/>

Table 4.1: Hyperparameter ranges searched over for the first optimisation run.

Hyperparameters	Ranges
Batch size	32 and 64
Initial learning rate	0.1 - 0.0001
Dropout	0.1 - 0.5
Out Channel	32 - 128
Fully connected layer depth	100 - 1,000
Optimiser	Adam, SGD
Epochs	30

4.4 Spectrogram extraction

Spectrogram settings have a large effect, and there are many to experiment with. The aim here is to reproduce the results described in other studies or tools.

4.4.1 Raven Pro

This section will try to recreate a similar-looking spectrogram using the AADC settings in Raven Pro [13], https://data.aad.gov.au/metadata/AcousticTrends_BlueFinLibrary.

AADC Raven Pro settings

Here we will define the settings used by the AADC. The settings used by the AADC are as follows: (1) The page size is set to a size of 120 seconds, (2) window settings of 0-125 Hz, (3) the *floor* is adjusted that 20% of the background was white, and (4) the *ceiling* is adjusted so that it was between 30–40 dB above background. See Section 2.4.1 for more details on *floor* and *ceiling* as well as other definitions. The parameters of the spectrogram used by AADC are shown in Figure 4.2.

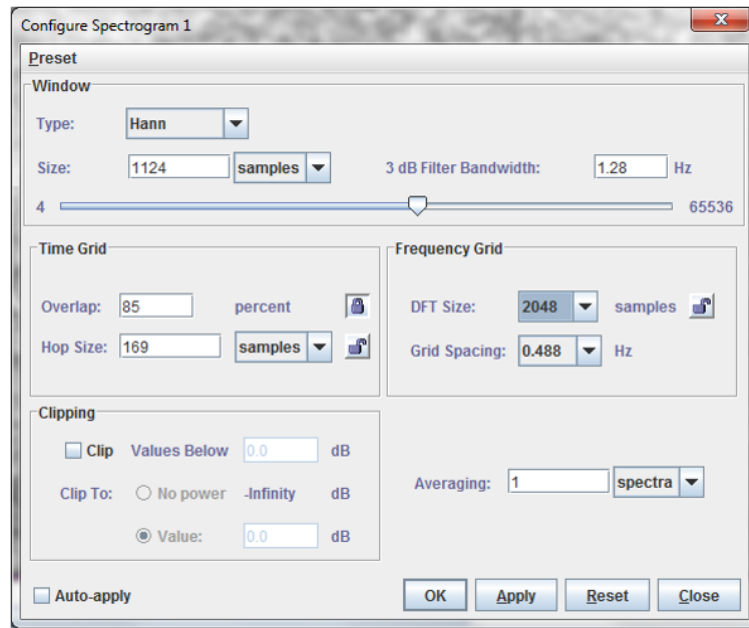


Figure 4.2: AADC spectrogram settings in Raven Pro.

The spectrogram results of the AADC obtained from the Raven Pro settings mentioned above are shown in Figure 4.3.

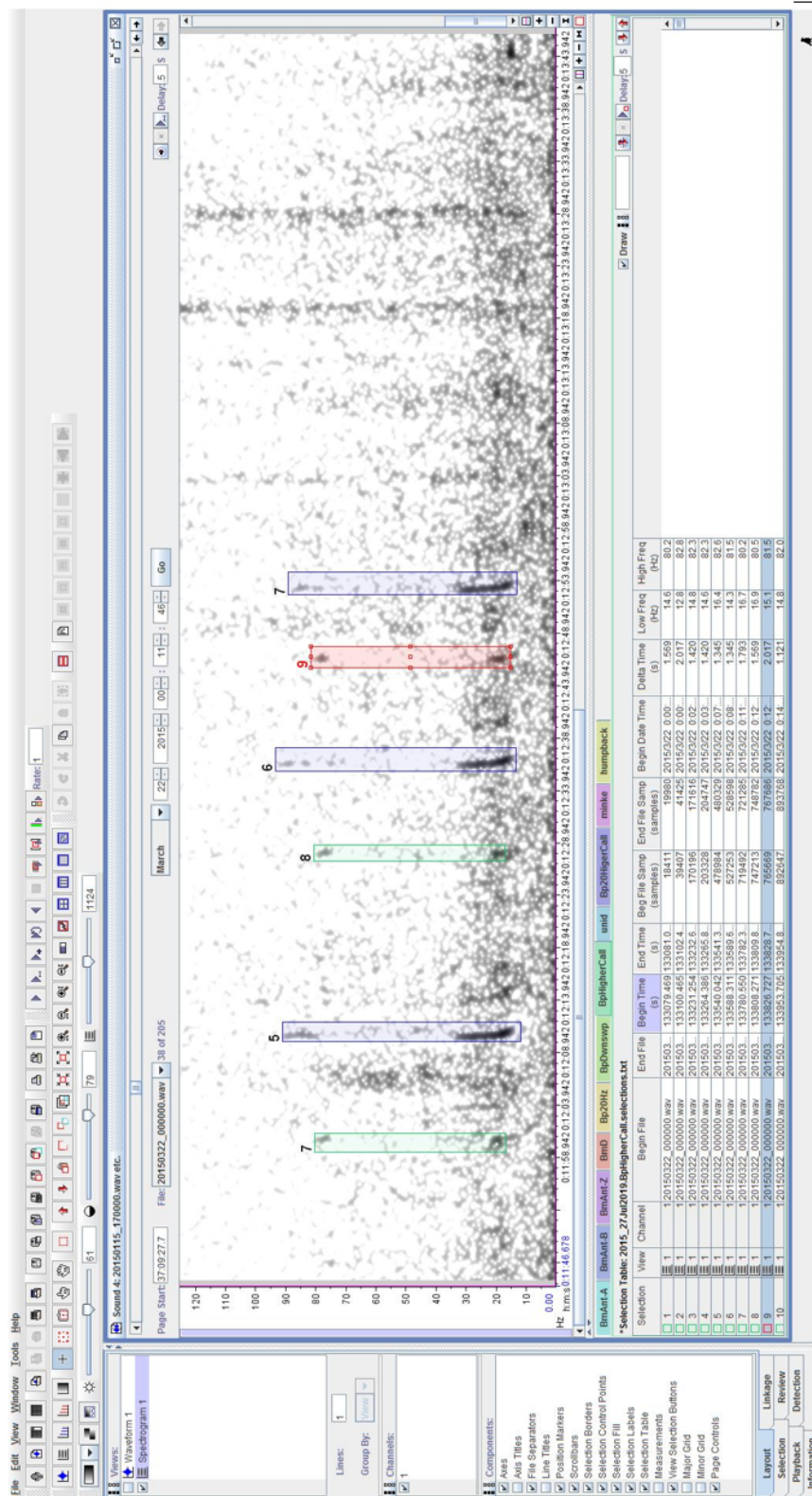


Figure 4.3: AADC spectrogram results in Raven Pro.

Replicated setting for Raven Pro

While replicating the spectrogram setting in Raven Pro, the *floor* and *ceiling* settings are the big problems. According to the Raven Pro manual, the *floor* setting displays the *floor colours* for the powers that are less than or equal to the set *floor* value [40]. Powers greater than or equal to the set *ceiling* value are displayed as the *ceiling colour* [40]. For the grey-scale spectrogram, the *floor* and *ceiling* colours are white, and black, respectively [40]. The settings used for the *floor* and *ceiling* are shown in Figure 4.4. The spectrogram setting as mentioned in Section 4.4.1 could be replicated exactly as shown in Figure 4.5

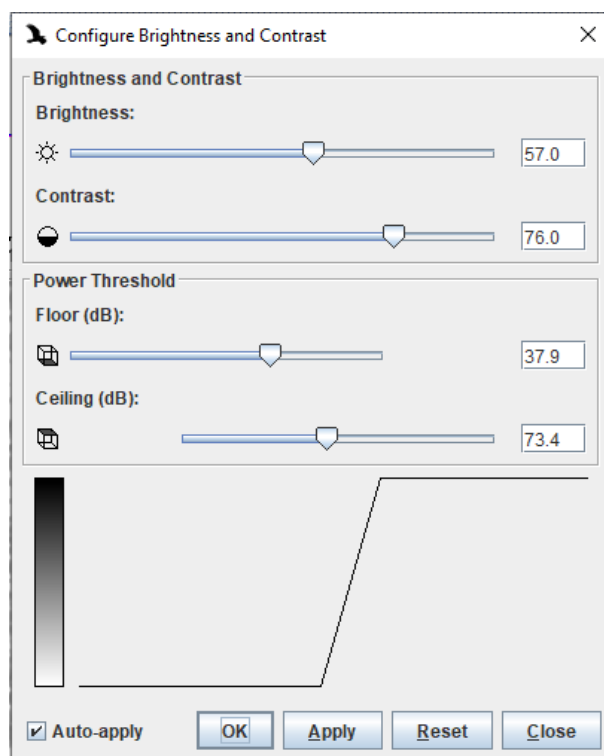


Figure 4.4: *Floor* and *ceiling* settings for replication in Raven Pro.

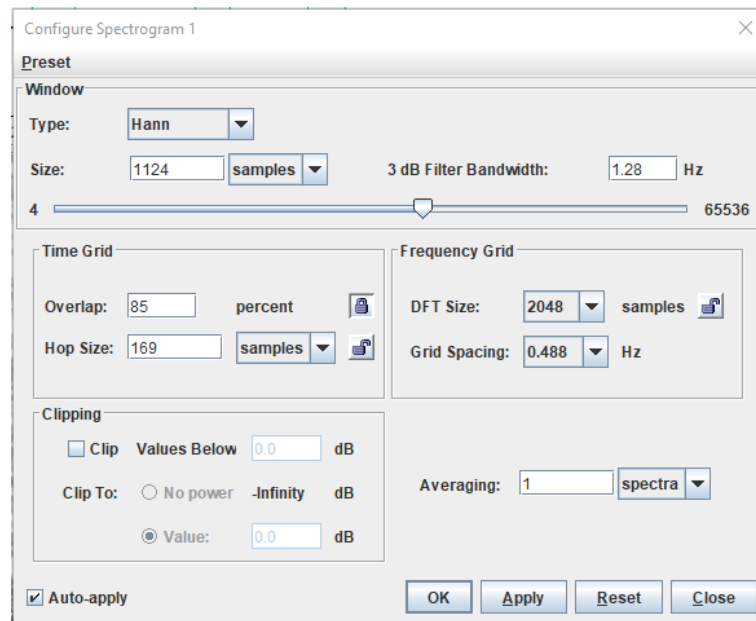


Figure 4.5: Spectrogram settings for replication in Raven Pro.

The resulting spectrogram that we obtained using the above-mentioned setting is shown in Figure 4.6. The quality of the replicated spectrogram is comparable to that of the original of the AADC.

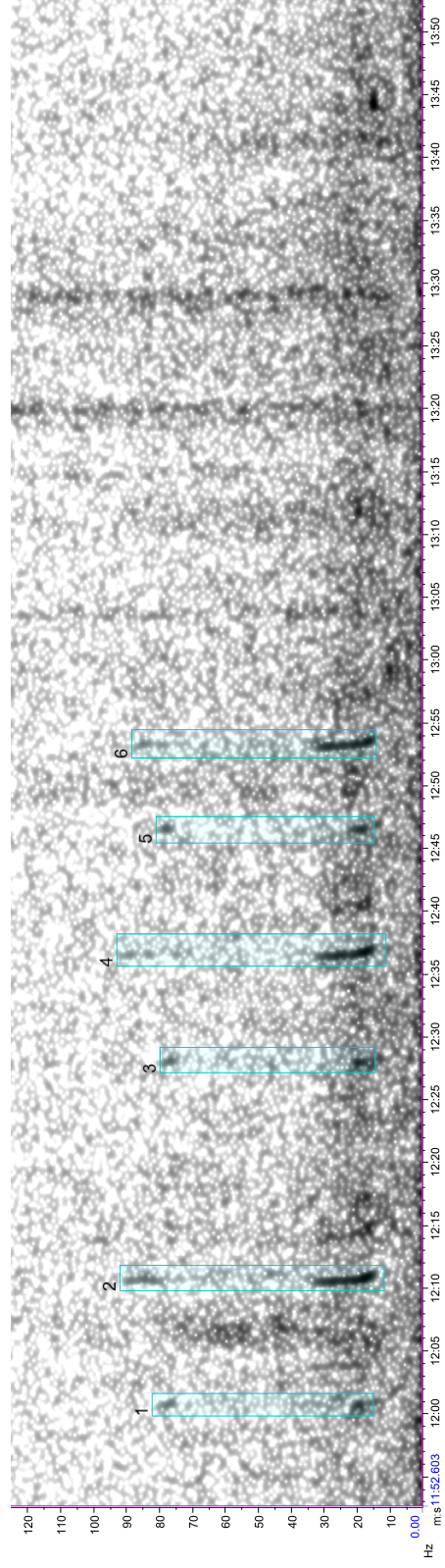


Figure 4.6: Spectrogram result after replicating the settings in Raven Pro.

4.4.2 Sonic Visualiser

We considered moving to Sonic Visualiser because it is an open source software application which we can use to automate the entire extraction of spectrograms. In comparison to this, Raven Pro is not open source and does not have the necessary functionality to automate the spectrogram extraction process. We tried to recreate the spectrogram from Raven Pro in Sonic Visualiser using the settings from Raven Pro so far as possible. Sonic Visualiser has fixed setting values to choose from, which makes it difficult to replicate the settings of Raven Pro. The setting for Sonic Visualiser is a window size of 1,024 with 87.5% overlap. The reason for these setting values is that the window size of 1,024 is closest to 1,124 (Raven Pro), and the overlap of 87.5% is the closest to 85% (Raven Pro). See Figure 4.7 for the spectrogram settings. In addition, the y-axis was limited to 125 Hz.

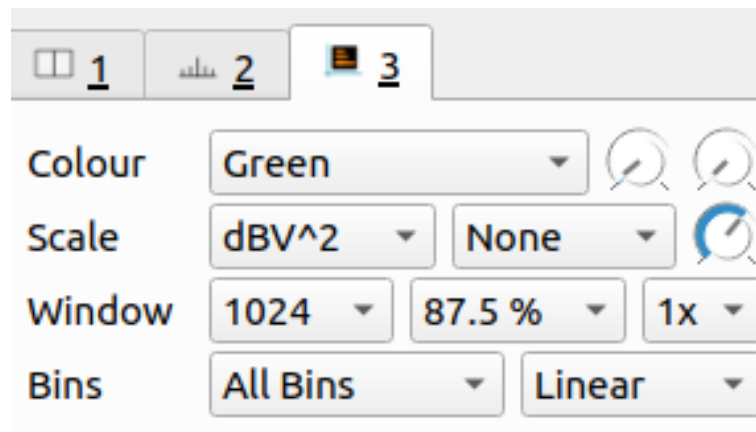


Figure 4.7: Spectrogram settings in Sonic Visualiser.

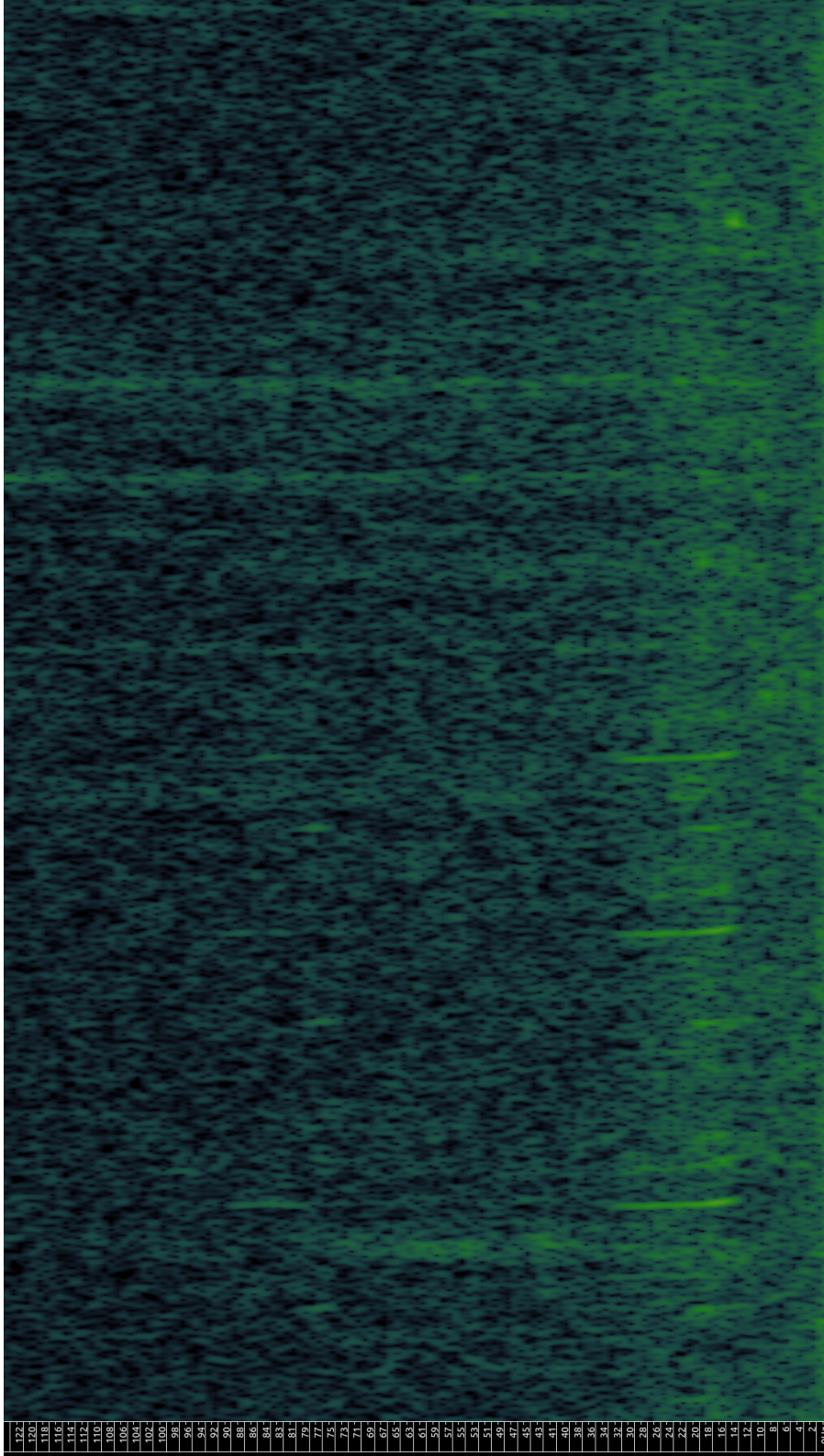


Figure 4.8: Spectrogram result obtained from settings in Sonic Visualiser.

In conclusion, the settings used in Sonic Visualiser differ from Raven Pro, and furthermore, Sonic Visualiser does not have floor and ceiling settings. The results obtained from Sonic Visualiser are comparable to those of Raven Pro, as seen in Figure 4.8, where we can clearly see the whale calls. The downside we experienced with Sonic Visualiser is that the automated spectrogram extraction process is not that easy because it does not have built-in functionality to automate the process. For this reason, we also consider Python.

4.4.3 Python

In this section, we try to recreate the spectrogram from Raven Pro in Python using the settings from Raven Pro so far as possible, together with the torch-audio package in Python. The reason for moving from Raven Pro and Sonic Visualiser to Python is that Python is free and is able to automate the entire spectrogram extraction process without any complications. The settings are shown in Table 4.2 and the resulting spectrogram in Figure 4.9.

Table 4.2: Settings used for the spectrogram in python.

Setting	Value
n_fft	314
win_length	314
hop_length	85
window_fn	Hanning

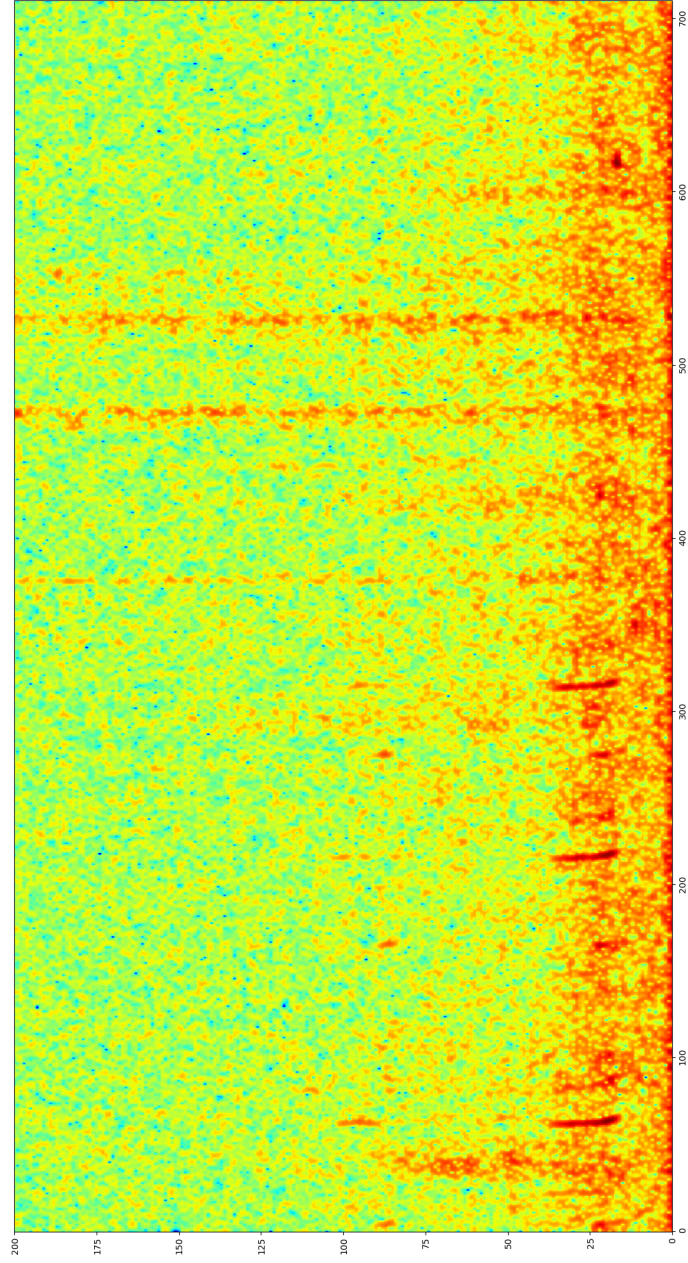


Figure 4.9: Spectrogram obtained in Python with the settings from Raven Pro.

When we compare the spectrogram obtained in Figure 4.9 with the spectrogram from Raven Pro, we can see the whale calls just as clearly. This also holds for other calls we looked at (humpback, minke, blue, and fin whales with their different call types). We compared the spectrograms between the different techniques through *visual inspection*, due to Raven Pro and Sonic Visualiser not automating the extraction process. We conclude that the spectrogram from Python will probably be sufficient for training the networks, even though we will only know this once the networks train successfully.

4.5 Analysis and Results

This section is dedicated to outlining a series of experiments intended for the baseline CNN. Our objective is to investigate specific aspects that we believe could significantly influence the system’s performance.

The specific aspects that change during the experiments are the chunk size (See Section 3.3.2) and the optimiser, as discussed in Section 4.3. Four different experiments (Sections 4.5.1 – 4.5.4) are conducted in which we vary the optimisers and chunk sizes to assess their impact on the CNN’s performance. Ultimately, we want to conclude which optimiser and chunk size perform the best.

In each experiment, we employ the `small_3_cs` data set, which consists of ‘calls’ and ‘no-calls’ segments with a chunk size of two and four seconds. The ‘no-calls’ segments included background noise that may also contain other biological sounds. The CNN architecture discussed in Section 4.2 and the optimisation protocol described in Section 4.3 are used.

The optimisation protocol searches over the hyperparameter ranges for five trials on a single seed, and this process will repeat for three different seeds. We then select the best model based on the lowest validation loss for the trial in each seed and report the results of that model. This is the best model per seed. Per technique, the results of the three best models (across the three seeds) are averaged. The averaged validation accuracy is used to select the technique.

The results we report on in each experiment are the accuracy, recall, and precision. The F1-score is not necessary because the data set is balanced and the accuracy is a summary thereof. The average and standard deviation of the results over the three seeds are calculated. This section concludes with a comparison of the averages of each experiment and then decides which model performed the best out of the series of experiments.

4.5.1 SGD-4

The objective of this experiment is to optimise the hyperparameters using SGD as the optimiser. The chunk size used in this experiment is four seconds in duration.

Table 4.3 summarises the results obtained from the best trial across the three different seeds. Based on the table, it can be inferred that seed 42 outperformed the other two seeds on validation accuracy, and therefore, we will provide more detailed information about this particular model.

Table 4.3: Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using SGD as an optimiser on the four-second chunks. The seed with the best validation loss is in *italics* and the average between the three seeds is in **bold**.

Seed	Accuracy (%)			Recall			Precision		
	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
<i>42</i>	<i>98.57</i>	<i>97.64</i>	<i>97.60</i>	<i>0.9986</i>	<i>0.9947</i>	<i>0.9959</i>	<i>0.9727</i>	<i>0.9559</i>	<i>0.9566</i>
185	97.35	97.20	96.83	0.9991	0.9972	0.9986	0.9480	0.9462	0.9382
1986	98.07	97.31	97.59	0.9954	0.9881	0.9894	0.9660	0.9566	0.9626
Avg	98.00	97.38	97.34	0.9977	0.9933	0.9946	0.9622	0.9529	0.9525
Std	0.5001	0.1870	0.3606	0.0016	0.0038	0.0039	0.0104	0.0047	0.0104

Table 4.4 presents the results of the five trials conducted for seed 42. Based on the validation loss values shown in the table, trial 2 performed the best, as it achieved the lowest validation loss.

Furthermore, the confusion matrices provide insight into the model’s performance. The recall indicates that the model is effective at capturing most of the positive instances

(calls), minimising the number of false negatives (instances that should have been positive, but which were predicted as negative (no-calls)). Further, when the model predicts a positive class, it is more likely to be correct and not a false alarm (false positive) due to the precision being this high.

Table 4.4: The trial result for Experiment 1 using SGD as an optimiser and four-second chunks on seed 42.

Trial	Validation loss	Learning rate	Dropout	Optimiser	Batch size	Full connected layers depth	Channel out
0	0.1518	0.0054	0.5	SGD	32	600	32
1	0.1679	0.0099	0.5	SGD	64	400	32
2	0.0953	0.0014	0.2	SGD	32	700	32
3	0.1376	0.0013	0.3	SGD	64	800	96
4	0.1966	0.0018	0.5	SGD	64	100	128

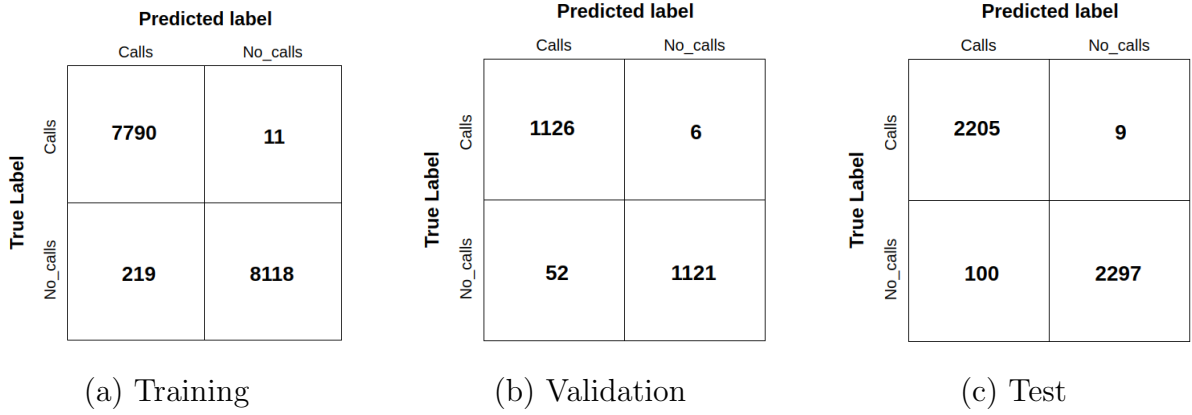


Figure 4.10: Confusion matrix for hyperparameter optimisation using Optuna with SGD as an optimiser for the four-second chunks on seed 42.

Based on the information presented in Table 4.3, we can infer that the model did not overfit the data due to the confusion matrix, accuracy, recall, and precision of the validation and test set being aligned with the results of the training set. The model does not struggle to generalise on the ‘unseen’ data (validation and test set) and produces reasonably good results. Additionally, trial 2’s hyperparameter values were within the range provided, which suggests that the model did not require extreme tuning to achieve its performance.

4.5.2 Adam-4

The objective of this experiment is to optimise the hyperparameters using Adam as the optimiser. The chunk size used in this experiment is four seconds in duration.

Table 4.5 presents a summary of the results obtained from the best trial across different seeds. On the basis of the table, it can be inferred that seeds 42 and 1986 outperformed the other seed on validation accuracy and recall. The reason for making the decision also based on the recall is that the validation accuracy for both seed 42 and 1986 is the same. In the case of this application, false negatives are more concerning; therefore, we prioritise a higher recall. We will provide more detailed information about the model trained with seed 42.

Table 4.5: Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using Adam as an optimiser on the four-second chunks. The seed with the best validation loss is in *italics* and the average between the three seeds is in **bold**.

Seed	Accuracy (%)			Recall			Precision		
	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
<i>42</i>	<i>99.43</i>	<i>98.00</i>	<i>97.90</i>	<i>1.0000</i>	<i>0.9982</i>	<i>0.9964</i>	<i>0.9888</i>	<i>0.9610</i>	<i>0.9618</i>
185	99.81	97.91	97.68	0.9970	0.9840	0.9842	0.9993	0.9736	0.9693
1986	99.89	98.00	98.00	0.9994	0.9946	0.9948	0.9984	0.9653	0.9661
Avg	99.71	97.97	97.86	0.9988	0.9662	0.9918	0.9955	0.9666	0.9657
Std	0.2007	0.0424	0.1337	0.0013	0.0060	0.0054	0.0048	0.0052	0.0031

Table 4.6 presents the results of the five trials conducted for seed 42. Based on the validation loss values shown in the table, trial 2 performed the best, as it achieved the lowest validation loss.

Furthermore, the confusion matrices provide insight into the model’s performance. The recall indicates that the model is effective at capturing most of the positive instances (calls), minimising the number of false negatives (instances that should have been positive, but were predicted as negative (no-calls)). Further, when the model predicts a positive class, it is more likely to be correct and not a false alarm (false positive) because of the

precision being this high.

		Predicted label	
		Calls	No_calls
True Label	Calls	7997	5
	No_calls	13	8123

(a) Training

		Predicted label	
		Calls	No_calls
True Label	Calls	1112	6
	No_calls	40	1147

(b) Validation

		Predicted label	
		Calls	No_calls
True Label	Calls	2277	12
	No_calls	80	2242

(c) Test

Figure 4.11: Confusion matrix for hyperparameter optimisation using Optuna with Adam as an optimiser for the four-second chunks on seed 42.

Based on the information presented in Table 4.5, we can infer that the model did not overfit the data due to the confusion matrix, accuracy, recall, and precision of the validation and test set being aligned with the results of the training set. The model does not struggle to generalise on the ‘unseen’ data (validation and test set) and produces reasonably good results. Furthermore, the hyperparameter values of trial 2 were within the range provided, suggesting that the model did not require extreme tuning to achieve its performance.

4.5.3 SGD-2

The objective of this experiment is to optimise the hyperparameters using SGD as the optimiser. The chunk size used in this experiment is two seconds in duration.

Table 4.7 presents a summary of the results obtained from the best trial across different

Table 4.6: The trial result for Experiment 2 using Adam as an optimiser and four-second chunks on seed 42.

Trial	Validation loss	Learning rate	Dropout	Optimiser	Batch size	Full connected layers depth	Channel out
0	0.0718	0.0023	0.3	Adam	32	700	32
1	0.1055	0.0001	0.4	Adam	32	700	128
2	0.0190	0.0001	0.2	Adam	32	800	64
3	0.0890	0.0114	0.0	Adam	64	200	96
4	0.0862	0.0033	0.1	Adam	64	900	96

seeds. Based on the table, it can be inferred that seed 185 outperformed the other two seeds on validation accuracy, and we will therefore provide more detailed information about this particular model.

Table 4.7: Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using SGD as an optimiser on the two-second chunks. The seed with the best validation loss is in *italics* and the average between the three seeds is in **bold**.

Seed	Accuracy (%)			Recall			Precision		
	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
42	90.24	88.20	87.81	0.9407	0.9226	0.9253	0.8597	0.8287	0.8250
<i>185</i>	<i>92.77</i>	<i>89.42</i>	<i>89.31</i>	<i>0.9557</i>	<i>0.9224</i>	<i>0.9260</i>	<i>0.8958</i>	<i>0.8607</i>	<i>0.8596</i>
1986	91.91	88.74	88.74	0.9474	0.9187	0.9231	0.8881	0.8463	0.8462
Avg	91.64	88.79	88.62	0.9479	0.9212	0.9248	0.8812	0.8452	0.8436
Std	1.050	0.4992	0.6182	0.0061	0.0018	0.0012	0.0155	0.0131	0.0142

Table 4.8 presents the results of the five trials conducted for seed 185. Based on the validation loss values shown in the table, trial 0 performed the best, as it achieved the lowest validation loss.

Furthermore, confusion matrices provide insight into the performance of the model. The recall indicates that the model is effective at capturing most of the positive instances (calls), minimising the number of false negatives (instances that should have been positive, but were predicted as negative (no-calls)). Furthermore, when the model predicts a positive class, it is more likely to be correct and not a false alarm (false positive) because the precision is this high.

Table 4.8: The trial result for Experiment 3 using SGD as an optimiser and two-second chunks on seed 185.

Trial	Validation loss	Learning rate	Dropout	Optimiser	Batch size	Full connected layers depth	Channel out
0	0.1034	0.0147	0.3	SGD	32	900	64
1	0.1507	0.0002	0.2	SGD	32	400	32
2	0.1299	0.0010	0.0	SGD	32	100	32
3	0.1212	0.0046	0.3	SGD	64	600	64
4	0.1269	0.0105	0.4	SGD	64	900	96

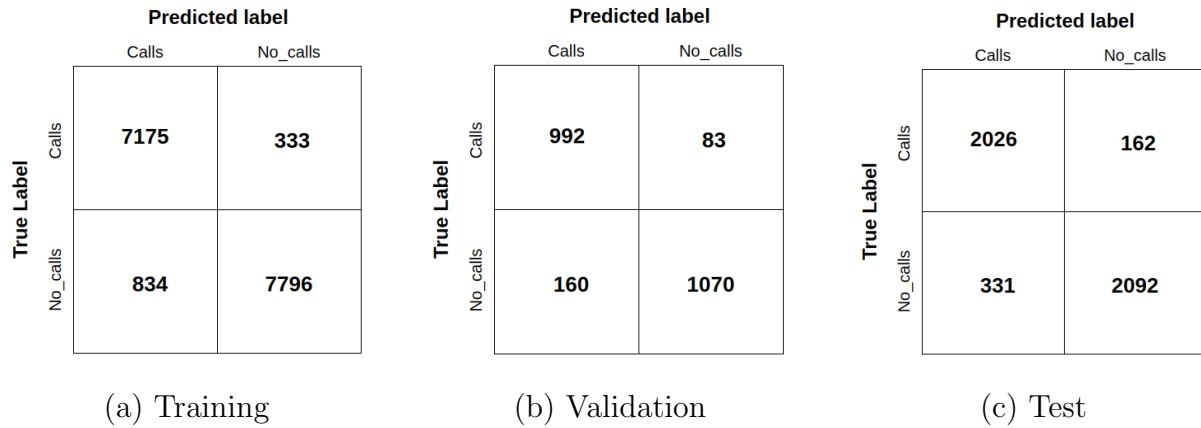


Figure 4.12: Confusion matrix for hyperparameter optimisation using Optuna with SGD as an optimiser for two-second chunks on seed 185.

Based on the information presented in Table 4.7, we can infer that the model did not overfit the data due to the confusion matrix, accuracy, recall, and precision of the validation and test set being aligned with the results of the training set. The model does not struggle to generalise on the ‘unseen’ data (validation and test set) and produces reasonably good results. Furthermore, the hyperparameter values of trial 0 were within the range provided, suggesting that the model did not require extreme tuning to achieve its performance.

4.5.4 Adam-2

The objective of this experiment is to optimise the hyperparameters using Adam as the optimiser. The chunk size used in this experiment is two seconds in duration.

Table 4.9 presents a summary of the results obtained from the best trial across different seeds. Based on the table, it can be inferred that seed 185 outperformed the other two seeds on validation accuracy, and we will therefore provide more detailed information about this particular model.

Table 4.9: Summary of the results obtained for each best trial on each seed with hyperparameter optimisation using Adam as an optimiser on the two-second chunks. The seed with the best validation loss is in *italics* and the average between the three seeds is in **bold**.

Seed	Accuracy (%)			Recall			Precision		
	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
42	93.65	87.49	87.27	0.9546	0.8908	0.8958	0.9170	0.8486	0.8461
185	<i>98.97</i>	<i>89.72</i>	<i>90.28</i>	<i>0.9943</i>	<i>0.9111</i>	<i>0.9237</i>	<i>0.9850</i>	<i>0.8803</i>	<i>0.8829</i>
1986	93.85	88.61	87.64	0.9824	0.9343	0.9382	0.8934	0.8269	0.8068
Avg	95.49	88.61	88.40	0.9771	0.9121	0.9192	0.9318	0.8519	0.8453
Std	2.4621	0.9103	1.3403	0.0166	0.0178	0.0176	0.0388	0.0219	0.0311

Table 4.10 presents the results of the five trials conducted for seed 185. Based on the validation loss values shown in the table, trial 2 performed the best, as it achieved the lowest validation loss.

Furthermore, confusion matrices provide insight into the performance of the model. The recall indicates that the model is effective at capturing most of the positive instances (calls), minimising the number of false negatives (instances that should have been positive, but were predicted as negative (no-calls)). Further, when the model predicts a positive class, it is more likely to be correct and not a false alarm (false positive) due to the precision being this high.

Table 4.10: The trial result for Experiment 4 using Adam as an optimiser and two-second chunks on seed 185.

Trial	Validation loss	Learning rate	Dropout	Optimiser	Batch size	Full connected layers depth	Channel out
0	0.1362	0.0003	0.1	Adam	32	400	96
1	0.1358	0.0003	0.1	Adam	64	600	64
2	0.1095	0.0021	0.1	Adam	32	700	128
3	0.1104	0.0032	0.1	Adam	32	300	64
4	0.1002	0.0060	0.1	Adam	32	800	64

		Predicted label	
		Calls	No_calls
True Label	Calls	2081	172
	No_calls	276	2082

(a) Training

		Predicted label	
		Calls	No_calls
True Label	Calls	1015	99
	No_calls	138	1054

(b) Validation

		Predicted label	
		Calls	No_calls
True Label	Calls	2081	172
	No_calls	276	2082

(c) Test

Figure 4.13: Confusion matrix for hyperparameter optimisation using Optuna with Adam as an optimiser for the two-second chunks on seed 185.

Based on the information presented in Table 4.9, we can infer that the model did not overfit the data due to the confusion matrix, accuracy, recall, and precision of the validation and test set being aligned with the results of the training set. The model does not struggle to generalise on the ‘unseen’ data (validation and test set) and produces reasonably good results. Furthermore, the hyperparameter values of trial 2 were within the range provided, suggesting that the model did not require extreme tuning to achieve its performance.

4.5.5 Discussion

In conclusion, for this section, we look at Table 4.11 and the main performance metrics that we consider are accuracy and recall. Accuracy is a crucial metric that provides us with a measure of the overall correctness of the models’ predictions. It also provides us with an easy way to compare the performance of our models to other models. We also look at recall as a main metric purely because it gives an idea of the model’s ability to correctly identify positive instances (calls). Considering this application, it is better to detect a negative instance (no call) as a positive instance (call) rather than missing a whale call.

Based on the main metrics and general findings, it can be concluded that Adam, as an optimiser, outperformed SGD. This is likely because Adam adjusts the learning rate for each

parameter based on historical gradients, enabling faster convergence and improved identification of minima, particularly in models with complex gradients or high-dimensional parameter spaces.

In addition, the four-second duration emerged as the most effective chunk size. This could be attributed to the average call length of the data set for blue and fin whales across the three sites, which was approximately 3.75 seconds (see Section 3.2.1). Consequently, the entire call can fit comfortably within the chosen chunk size.

Table 4.11: Summary of the average and standard deviation for the four different experiments in Chapter 4.

ID	Stat	Accuracy (%)			Recall			Precision		
		Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
S-4	Max	98.57	97.64	97.60	0.9986	0.9947	0.9959	0.9727	0.9559	0.9566
	Avg	98.00	97.38	97.34	0.9977	0.9933	0.9946	0.9622	0.9529	0.9525
	Std	0.5001	0.1870	0.3606	0.0016	0.0038	0.0039	0.0104	0.0047	0.0104
A-4	Max	99.43	98.00	97.90	1.0000	0.9982	0.9964	0.9888	0.9610	0.9618
	Avg	99.71	97.97	97.86	0.9988	0.9662	0.9918	0.9955	0.9666	0.9657
	Std	0.2007	0.0424	0.1337	0.0013	0.0060	0.0054	0.0048	0.0052	0.0031
S-2	Max	92.77	89.42	89.31	0.9557	0.9224	0.9260	0.8958	0.8607	0.8596
	Avg	91.64	88.79	88.62	0.9479	0.9212	0.9248	0.8812	0.8452	0.8436
	Std	1.050	0.4992	0.6182	0.0061	0.0018	0.0012	0.0155	0.0131	0.0142
A-2	Max	98.97	89.72	90.28	0.9943	0.9111	0.9237	0.9850	0.8803	0.8829
	Avg	95.49	88.61	88.40	0.9771	0.9121	0.9192	0.9318	0.8519	0.8453
	Std	2.4621	0.9103	1.3403	0.0166	0.0178	0.0176	0.0388	0.0219	0.0311

4.6 Conclusion

The baseline CNN is introduced in this chapter, where we describe the architecture and explain the optimisation protocol. We also looked at the different ways to extract spectrograms and found that Python is the most suitable for our application because of its ease of automation. Taking everything into consideration, we moved ahead to how and why we performed the four different experiments. We can conclude that the data set and the

preprocessing done in Chapter 3 was sufficient to establish a baseline CNN. The baseline CNN delivered comparable results to other standard CNN approaches (see Section 2.3).

Chapter 5

Whale call detection using SpeechBrain

In this chapter, we discuss the TDNN model from SpeechBrain that we adapted for whale call detection and how it performed on the `small_3_cs` and `large_3_cs` data sets. We compare TDNN to the baseline CNN in Chapter 4.

5.1 Overview

In this chapter, we will look at the viability of using TDNNs with x-vectors for whale call detection. The toolkit we use is SpeechBrain, as introduced in Section 2.5.3, and we describe how we use it to detect whale calls. We start by describing the SpeechBrain toolkit in more detail in Section 5.2, discussing the specific adjustments that were needed to utilise it for whale call detection. In Section 5.3, we run an initial experiment using the out-of-the-box TDNN.

There are different ways in which the whale call detection task can be framed. In Sec-

tion 5.4, we define three such Machine Learning (ML) tasks that we will use for experiments. The experimental setup and the optimisation protocol used for each SpeechBrain experiment are described in Section 5.5 and Section 5.6, respectively. In Section 5.7, we analyse the results obtained for the different ML tasks. In Section 5.8, we compare the TDNN to the baseline CNN in Chapter 4. Finally, Section 5.10 concludes with a summary and key findings of this chapter.

5.2 SpeechBrain

In the following sections, we will discuss the pipeline used by SpeechBrain (Section 5.2.1) and adaptations made to SpeechBrain’s speaker identification for whale call detection (Section 5.2.2).

5.2.1 SpeechBrain’s pipeline

This section will look at the recipe used by SpeechBrain that will form the basis for any of the key tasks. In SpeechBrain, there are two important files we will work with:

1. A Yet Another Markup Language (YAML) file where all hyperparameters are specified. The file also specifies which class or function will use the hyperparameters. Thus, it contains all the information to initialise the classes or functions when loading them.
2. An experiment file that contains the main scripts for training. This file trains a model by combining the classes or functions declared in the YAML file [12]. This file also defines the data processing pipeline and all computations from the input signal to the final cost function.

The steps for the SpeechBrain recipe are as follows:

-
1. **Data specification file:** The first step in SpeechBrain is that the user should prepare a data specification file (data manifest) in a Comma-separated Values (CSV) or JavaScript Object Notation (JSON) format that will report where to find the data and the labels to process. The user should create three manifest files; for training, validation, and testing.
 2. **Data processing pipeline:** SpeechBrain has a data reading pipeline that is fully customisable directly in the experiment file. The method takes as input the path to the WAV file specified in the data manifest file and returns the signal. In this step, the user can also add any processing (e.g., adding noise, or dynamic mixing) by coding with the desired pipeline. The data loader is also created with the Python data loader, which uses the different data manifest files to create data loaders with the desired batch size for training, validation, and testing.
 3. **Custom forward and cost calculation methods:** Users often want to customise the sequence of computations that go from the input to the final cost function. Users can specify them in the *forward* and *compute_objectives* methods. The input to these methods is the variable batch that contains all the entries specified in the data loader. In the forward method for the given batch, the features (STFT, Fbanks, MFCCs, etc.) are calculated, data augmentations (speed changes) are performed, and normalisation (mean and variance) can be performed. Depending on the problem, users can extract embeddings, call the model, perform linear transformations, apply softmax, etc. The forward method returns the output of the model. In the compute_objectives method, it will take the given batch and the predictions and return the loss for the validation stage.
 4. **Brain Class:** The brain class implements all the steps needed in the standard training and validation loops [12]. After defining the above-mentioned (data pipeline, forward, computed objective, and any other custom methods), we can call the fit method of the brain class for training and validation and the eval method for testing [12].

5.2.2 Adaptations made to SpeechBrain’s speaker identification for whale call detection

In the following sections, we will go through the recipe steps mentioned in Section 5.2.1 to configure SpeechBrain’s speaker identification for whale call detection.

Data Manifest

This section will create and look at the SpeechBrain data manifest. SpeechBrain has two formats for the data manifest: CSV and JSON. In the data manifest, we tell SpeechBrain where to find the audio data, the audio length, and the transcription. We chose the JSON format for the data manifests because it looks neater.

Using the data sets in Section 3.4, the paths, lengths, and annotations of each chunk of audio file given by AADC are used to create a data manifest for each ML task as defined in Table 5.3. The data manifest for each data set is divided into training, validation, and testing (as described above), in the ratio 70:10:20, respectively. More detail can be found in Section 5.4. There are no duplicates of any audio file in the training, validation, and testing sets; all the samples that will be used are unique.

The training data manifest for ML task blue-fin-all-else for the small_3.4 data set is shown in Figure 5.1.

```

{
  "20150721_190000_Sample_Sound_322": {
    "wav": "{data_root}/small_3_nbg/20150721_190000_Sample_Sound_322.wav",
    "length": 6,
    "spk_id": "no_calls"
  },
  "20140321_080000_AWI251-01_AU0231_250Hz_Sample_Sound_6": {
    "wav": "{data_root}/small_3_nbg/20140321_080000_AWI251-01_AU0231_250Hz_Sample_Sound_6.wav",
    "length": 5.04,
    "spk_id": "calls"
  },
  "20141224_190000_AWI251-01_AU0231_250Hz_Sample_Sound_0": {
    "wav": "{data_root}/small_3_nbg/20141224_190000_AWI251-01_AU0231_250Hz_Sample_Sound_0.wav",
    "length": 9.333333,
    "spk_id": "calls"
  },
  "20150122_180000_Sample_Sound_482": {
    "wav": "{data_root}/small_3_nbg/20150122_180000_Sample_Sound_482.wav",
    "length": 3,
    "spk_id": "no_calls"
  },
  "20130318_210000_AWI251-01_AU0231_250Hz_Sample_Sound_9": {
    "wav": "{data_root}/small_3_nbg/20130318_210000_AWI251-01_AU0231_250Hz_Sample_Sound_9.wav",
    "length": 3.806667,
    "spk_id": "calls"
  },
  "20150709_100000_Sample_Sound_293": {
    "wav": "{data_root}/small_3_nbg/20150709_100000_Sample_Sound_293.wav",
    "length": 7,
    "spk_id": "no_calls"
  },
  "20150419_060000_Sample_Sound_197": {
    "wav": "{data_root}/small_3_nbg/20150419_060000_Sample_Sound_197.wav",
    "length": 4.166667,
    "spk_id": "calls"
  },
  "20150605_220000_Sample_Sound_328": {
    "wav": "{data_root}/small_3_nbg/20150605_220000_Sample_Sound_328.wav",
    "length": 8,
    "spk_id": "no_calls"
  },
  "20140417_080000_AWI251-01_AU0231_250Hz_Sample_Sound_2": {
    "wav": "{data_root}/small_3_nbg/20140417_080000_AWI251-01_AU0231_250Hz_Sample_Sound_2.wav",
    "length": 5.076667,
    "spk_id": "calls"
  },
  "20150611_050000_Sample_Sound_299": {
    "wav": "{data_root}/small_3_nbg/20150611_050000_Sample_Sound_299.wav",
    "length": 4,
    "spk_id": "no_calls"
  },
  "20130601_120000_AWI251-01_AU0231_250Hz_Sample_Sound_6": {
    "wav": "{data_root}/small_3_nbg/20130601_120000_AWI251-01_AU0231_250Hz_Sample_Sound_6.wav",
    "length": 3.34,
    "spk_id": "calls"
  },
  "20150628_200000_Sample_Sound_517": {
    "wav": "{data_root}/small_3_nbg/20150628_200000_Sample_Sound_517.wav",
    "length": 3,
    "spk_id": "no_calls"
  },
}

```

Figure 5.1: The training data manifest for small_3.4 data set for ML task blue-fin-all-else.

Data processing pipeline

In this section, we will discuss the data processing pipeline, Step 2 of the recipe mentioned in Section 5.2.1, and how it is used for speaker identification and can be adapted to whale call detection. There are a few steps that we perform here:

1. **Initialisation of the label encoder:** This is where we use SpeechBrain’s *CategoricalEncoder* method for assigning a unique index to each of the observed labels (e.g ‘speaker01’: 0, ‘speaker02’: 1). It obtains the labels from the provided data manifest file. In our case, the label encoder will use ‘calls’: 0, ‘no-calls’: 1.
2. **Define audio pipeline:** Here, we fetch the audio files using the path in the data manifest file and get the signal of the audio file as a tensor using the general audio loading from Python.
3. **Define label pipeline:** Here, we retrieve the labels for the corresponding signal using the data manifest.
4. **Data loader:** Here, we create the data set and data loaders using Python’s data set and data loader libraries. The data set is created from the signals of the WAV files and their corresponding labels for the different data manifest files. The different data loaders for training, validation, and testing are created using the desired batch size.

Custom forward and cost computation methods

This section will discuss the custom forward and cost computation methods; Step 3 of the recipe mentioned in Section 5.2.1, and how it is used for speaker identification and can be adapted to whale call detection.

The *forward* method takes the variable batch that contains all the entries specified in the data loader. In the forward method for the given batch, the following processes occur:

1. **Prepare features:** In this method, data augmentations are applied to the WAV file signals, the features are computed, and normalisation is performed.

- **Data augmentation:** Adds speech changes + time and frequency dropouts (time-domain implementations). To help the performance of the system, a small speed change is implemented. The augmentations implemented in the time domain are:

- Dropping chunks of audio with zero amplitude or white noise.
- Dropping frequency bands with band-drop filters.
- Speed perturbation via resampling at a slightly different rate.

The only adaptations made to the data augmentations were to change the sample rate to 300 Hz and to remove the added noise and reverberation that are added as environmental corruptions for speech signals.

- **Compute features:** Generates features for input to the speech pipeline. We are using two different feature extraction methods:

- Fbanks
- MFCCs

The adaptations made for the compute features were to change the sample rate to 300Hz. We will also train models that will use deltas and without the deltas.

- **Normalisation:** Performs mean normalisation of the input tensor. We did not make adaptations to the normalisation method. The default implementation uses only mean normalisation.

2. **Embeddings:** SpeechBrain uses a TDNN model that extracts x-vectors (see Section 2.5.2 for more information). Statistical pooling is used at the top of the convolutional layers to convert a variable-length sentence into fixed-length embeddings. The only adaptation that we make to the embeddings is to change the input channel size to the appropriate size of the features calculated in the prepare features function.

-
3. **Predictions:** Here, SpeechBrain implements an MLP on top of the x-vector features. No adaptations were made to the classifier.

The *compute_objectives* method takes the predictions and the variable batch that contains all the entries specified in the data loader as input. The loss is computed in this method, given the predicted and target outputs.

5.3 Out-of-the-box TDNN

The goal of this section is to see whether the out-of-the-box TDNN system from SpeechBrain with no changes will deliver usable results. This will also allow us to see at the end in Section 5.8 if our changes made any improvements to the system.

5.3.1 Experimental setup

The *small_3.cs* data set is used for the initial setup. The SpeechBrain TDNN system is used just as it is, meaning there are no changes to the hyperparameter; the sample rate is kept at 16,000 Hz which is meant for human speech. No optimisation is done. We only adapt the manifest file to our data set and adjust the split ratio between the training, testing, and validation set to 70:20:10, respectively. We do this to make the results comparable between experiments.

The following Table 5.1 shows the out-of-the-box hyperparameters. Further, the optimiser that is used is Adam with a linear scheduler and LeakyReLU as the activation function.

Table 5.1: Hyperparameter values for the out-of-the-box SpeechBrain TDNN system.

Hyperparameters	Value
Learning rate	0.001
Embedding dimensions	512
TDNN channels	512
Batch size	16
Number mel filter banks	23
Feature lobes	Fbanks
Deltas	False
Epochs	35
Seed	1986

5.3.2 Results

Table 5.2 shows the results for the two- and four-second chunks using Adam as an optimiser for the TDNN.

Table 5.2: Summary of the results obtained for Adam as an optimiser on the 2- and 4-second chunks for the initial TDNN.

ID	Accuracy (%)		Precision		Recall	
	Valid	Test	Valid	Test	Valid	Test
tdnn_adam_2	69.97	56.52	0.6761	0.5486	0.7431	0.7341
tdnn_adam_4	69.98	69.96	0.1669	0.3570	0.6956	0.5811

5.3.3 Discussion

Looking at the results in Table 5.2 we can conclude the following:

- The drop in accuracy from the validation set (69.97%) to the test set (56.52%) for `tdnn_adam_2` indicates that the model does not generalise well on the unseen data. The difference between the validation and test set is an indication of overfitting, because the model became too tailored to the training data.
- The precision drop between the test set (0.3570) and the validation set (0.1669) for `tdnn_adam_4` is troubling. The low precision values indicated that a large number

of instances predicted as positive is actually negative. This is a sign that the model is busy overfitting.

The reason for the relatively high accuracy of the TDNN is due to the data set being balanced and the model predicting almost everything as ‘no-calls’. In the recall and precision case, the model score is very low, meaning that the model predicted most of the ‘calls’ as ‘no-calls’, which is bad because the model does not correctly predict and just starts guessing everything as ‘no-calls’. The negative rate is very high for this reason.

Overall, as we expected, the TDNN model performed very poorly. The main reason may be that the sampling rate is too high for the whale calls, which is around 300Hz. The sections that follow will implement the changes to the SpeechBrain TDNN system as well as compare different whale call farmings (as described in Section 5.4). We will make another comparison in Section 5.8 to determine if we could improve on the SpeechBrain TDNN system.

5.4 Framing whale call detection as different machine learning tasks

This section describes how the `small_3_cs` and `large_3_cs` data sets in Section 3.4 are used to create the data manifest files for the different ML tasks. The data manifest tells SpeechBrain where to find the audio data, the audio length, and the transcription. For each of the following ML tasks, the training, testing, and validation sets are divided in the ratio of 70:20:10, respectively.

Table 5.3: Different ML task for SpeechBrain.

Task ID	Calls	No-Calls
blue-fin-all-else	Blue + Fin	All else
blue-fin-background	Blue + Fin	Background only
all-whale-background	Any whale	Background only

-
- **Blue-fin-all-else:** In the blue-fin-all-else task, we consider two classes. The first class is ‘calls’, which only contains the calls from blue and fin whales. The second class is ‘no-calls’, and this class contains any other biological sound or noise.
 - **Blue-fin-background:** In the blue-fin-only task, we consider two classes. The first class is ‘calls’, which only contains the calls from blue and fin whales. The second class is ‘no-calls’, and this class contains only background noise.
 - **All-whale-background:** In the all-whale-background task, we consider two classes. The first class is ‘calls’, which contains calls from any whale. The second class is ‘no-calls’, and this class contains only background noise.

The ML task that is directly comparable to the CNN in Chapter 4 is the blue-fin-background.

5.5 Experimental setup

In this section, we motivate the decisions made and the principles followed to develop a whale call detection model using SpeechBrain. In the data augmentation and feature extraction below, the sample rate is changed to 300Hz.

Part of SpeechBrain’s standard preprocessing pipeline is to add data augmentation before feature extraction. The data augmentation we use is a time-domain approximation of the SpecAugment algorithm [123]. In this data augmentation, we have three default speed perturbations (95, 100, 105) for SpeechBrain to choose from.

We use Fbanks and MFCCs for feature extraction. In the Fbanks, we can use ‘deltas’, which will append derivatives and second derivatives to the features. When extracting MFCCs, the deltas are appended by default. The number of mel filters can also be adjusted in both Fbanks and MFCCs; we will evaluate the mel filters (23, 40, 80) as these are the default values that SpeechBrain suggests. After the features are computed, we can apply

mean and variance normalisation; we use the default where only mean normalisation is applied.

The TDNN constructed here has five time-delayed neural layers with the given dilation factors (1, 2, 3, 1, 1) and kernel size (5, 3, 3, 1, 1). LeakyReLU is used as an activation function, since it has been shown to accelerate the learning of DNNs and can achieve good results [121]. After each activation function, batch normalisation is used; this tends to improve speaker identification performance [12]. After the convolutional layers, a statistical pooling layer is used to convert a variable length tensor into a fixed length tensor [12]. After the statistical pooling layer, a final linear transformation is performed using a linear layer, with the number of neurons depending on the size of embedding required. The x-vectors are extracted here. Lastly, an MLP is implemented on top of the x-vector layer. The MLP consists of one fully connected layer followed by the final softmax layer for classification. The number of input neurons in the layer again depends on the size of the embeddings. The output neurons are the same as the number of classes that need to be classified. All networks are trained with the Adam [82] optimiser. A scheduler with a linear annealing technique is implemented.

We optimised the following hyperparameters: embedding dimensions, TDNN channels, and the number of mel filter banks. We use the default SpeechBrain ranges while we search over broader ranges for the learning rate, batch size, feature lobes, and deltas, as shown in Table 5.4. Other hyperparameters that we consider for optimisation are augmentation, TDNN blocks, TDNN kernel sizes, TDNN dilations, optimiser, activation function, and learning rate annealing. We did not search over these hyperparameters because the search quickly becomes computationally expensive.

The hyperparameter optimisation protocol is described in Section 5.6, and this is the same protocol used by all SpeechBrain experiments.

5.6 Optimisation protocol

The algorithm we use is again TPE, which is a sequential model-based optimisation approach [115]. The hyperparameters we searched over are shown in Table 5.4. Speech-Brain comes with a convenient wrapper called ‘hpopt’, which can report objective values to Orion [115]. This facilitates the optimisation process. This made the implementation quite easy: the main code is wrapped in a hyperparameter optimisation context and the objective set to the metric that Orion will optimise.

We trained the model until convergence, where if the model’s accuracy in the past 20% of the epochs is still increasing, another 20% of the past number of epochs is added to continue training. Minimum training is set at ten epochs. Training is stopped if the accuracy of the model did not increase in the past 20% of the epochs. We also implemented early stopping by selecting the model with the best validation accuracy in the training cycle and saving that model. For the hyperparameter sweep, we sweep the hyperparameters and their corresponding values, as seen in Table 5.4. The model sweeps over the selected hyperparameters with their corresponding values for ten trials, and after that, the model uses the best values from the sweep and does another hyperparameter sweep on only the learning rate (the most sensitive parameter) for final fine-tuning. During the hyperparameter sweep, the model is trained until convergence, and then only the model with the best validation accuracy in the training cycle for that trial is saved.

Table 5.4: Hyperparameters value ranges for the Orion search.

Hyperparameters	Search space
Learning rate	log_uniform(0.001, 0.1, precision=2)
Embedding dimensions	[128, 256, 512, 768, 1024]
TDNN channels	[128, 256, 512, 768, 1024]
Batch size	[8, 16, 32]
Number of mel filter banks	[23, 40, 80]
Feature lobes	[Fbanks, MFCCs]
Deltas	[True, False]

5.7 Analysis and Results

This section is dedicated to describing a series of experiments intended for SpeechBrain. We aim to investigate aspects that we believe could significantly influence the system’s performance.

The specific aspects (hyperparameters) that change during the experiments are the chunk size (see Section 3.3.2), and the feature extraction method discussed in Section 5.6. The objective is to determine the best feature extraction method between Fbanks and MFCCs. Additionally, we will assess the impact of applying derivatives (deltas) to each extraction method and compare the performance of the model with and without them. Finally, we will experiment with different chunk sizes of two and four seconds. These experiments are carried out on three different ML tasks as described in Section 5.4, on three different seeds for each ML task.

All experiments use the experimental setup and optimisation protocol in Sections 5.5 and 5.6, respectively. The results we report on in each experiment are accuracy, recall, precision, and negative rate. The F1-score is unnecessary because the data set is balanced and the accuracy is a summary. The average and standard deviation of the results are calculated over the three seeds. True in the tables indicates that SpeechBrain is appending derivatives (deltas) and second derivatives (deltas) to the features. False in the tables indicates that SpeechBrain is not appending derivatives (deltas) and second derivatives (deltas) to the features.

5.7.1 Small_3_cs data set

We first experiment with the smaller data set (small_3_cs) to determine the impact of the aspects (hyperparameters) that are under test. The reason for experimenting with the smaller data set (small_3_cs) is that it is very time-consuming to run these experiments on the large data set. Once we have concluded the impact of the variables under testing, we will experiment with the findings on the larger data set (large_3_cs). The confusion

matrix for all experiments conducted in this chapter can be found in Appendix A.3.

Fbanks-2

The goal of this experiment is to optimise the hyperparameters and to see whether the deltas of the Fbanks will have an effect on the validation and test accuracy of each ML task. The chunk size used in this section is two seconds in duration. The results of this section will be compared with the results of Section 5.7.1 (MFCCs-2) to determine which feature extraction method has the greatest impact on the results.

Table 5.5: Result for the ML tasks across the three seeds to see the impact of the deltas using Fbanks as the feature extraction method on 2-second chunks. **Bold** indicates the results for the best derivative option. *Italics* indicates the best result across the seeds.

Delta-Seed	Accuracy (%)		Precision		Recall	
	Valid	Test	Valid	Test	Valid	Test
blue-fin-all-else						
True-42	78.79	90.99	0.7728	0.9984	0.8151	0.8213
False-42	76.53	79.62	0.7975	0.9119	0.7109	0.6557
<i>True-185</i>	<i>78.87</i>	<i>94.12</i>	<i>0.7394</i>	<i>0.9937</i>	<i>0.8915</i>	<i>0.8880</i>
False-185	78.00	80.90	0.7405	0.9570	0.8620	0.6470
True-1986	78.09	72.11	0.7739	0.6887	0.7925	0.8065
False-1986	78.09	83.46	0.7226	0.9584	0.9088	0.6995
Avg-True	78.58	85.74	0.7620	0.8936	0.8330	0.8386
Std-True	0.3504	9.7222	0.0160	0.1449	0.0424	0.0354
Avg-False	77.54	81.33	0.7535	0.9424	0.8272	0.6674
Std-False	0.7151	1.5964	0.0319	0.0216	0.0844	0.0230
blue-fin-background						
True-42	96.96	91.56	0.9688	0.8766	0.9705	0.9644
False-42	96.48	94.75	0.9820	0.9506	0.9470	0.9440
True-185	96.36	92.47	0.9684	0.9029	0.9583	0.9518
False-185	95.79	91.12	0.9497	0.8703	0.9670	0.9666
<i>True-1986</i>	<i>97.31</i>	<i>94.86</i>	<i>0.9866</i>	<i>0.9449</i>	<i>0.9592</i>	<i>0.9527</i>
False-1986	97.05	94.53	0.9738	0.9351	0.9670	0.9570
Avg-True	96.88	92.96	0.9746	0.9081	0.9623	0.9563
Std-True	0.3923	1.3916	0.0085	0.0281	0.0056	0.0057
Avg-False	94.44	93.47	0.9685	0.9623	0.9603	0.9559
Std-False	0.5152	1.6618	0.0137	0.0056	0.0094	0.0093
all-whale-background						
True-42	92.67	92.78	0.9852	0.9669	0.8664	0.8859
False-42	94.97	94.67	0.9823	0.9661	0.9159	0.9258
<i>True-185</i>	<i>95.23</i>	<i>94.93</i>	<i>0.9798</i>	<i>0.9637</i>	<i>0.9237</i>	<i>0.9337</i>
False-185	93.76	93.82	0.9461	0.9505	0.9280	0.9245
True-1986	92.36	91.50	0.9453	0.9150	0.8994	0.9150
False-1986	93.58	92.78	0.9467	0.9258	0.9237	0.9301
Avg-True	93.42	93.07	0.9701	0.9485	0.8965	0.9115
Std-True	1.2861	1.4152	0.0177	0.0237	0.0235	0.0197
Avg-False	94.10	93.76	0.9584	0.9475	0.9224	0.9268
Std-False	0.6172	0.7729	0.0169	0.0166	0.0051	0.0024

We analyse Table 5.5 by looking at the impact of deltas in each ML task.

For the blue-fin-all-else ML task, from the results, we can conclude the following:

- The test set accuracy is generally higher than the validation set accuracy. The potential cause of this could be the small data set. In a small data set, the random variation or noise in the validation set can have a considerable effect. This could lead to a less reliable estimation of the model’s performance on the validation set in comparison to the test set.
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy, precision, and recall compared to not including derivatives. The highest average performance across seeds is achieved by including derivatives. With a slight improvement (1.04%) as regards not including derivatives.
- Across seeds, it is also notable that True-185 achieved the best overall results.

Another reason for the models not performing as well as the models from the other two ML tasks might be the number of other whale sounds and biological sounds found in the ‘no-calls’ class. Together with the small data set. We will see if the larger data set improves the performance of the models in Section 5.7.2 (Fbanks-large-4).

In the ML task blue-fin-background, from the results, we can conclude the following:

- The difference between the validation set accuracy and test accuracy is notably better than the blue-fin-all-else. Where the validation accuracy is higher than the test accuracy.
- The difference between the validation accuracy and the test accuracy can be considered acceptable due to the general consistency between the three seeds.
- Across seeds, it is evident that including derivatives generally leads to higher accuracy, precision, and recall compared to not including derivatives. The highest average performance across seeds is achieved by including derivatives. With an improvement of 2.44% in regard to not including derivatives.

-
- Across seeds, it is also notable that True-1986 achieved the best overall results.

We can conclude that the models generalise and are not overfitting because the models perform well in both the validation and the test set, with high accuracy, precision, and recall scores. High precision suggests that the positive predictions made by the models are accurate, and high recall indicates that the models are effectively capturing a large portion of the actual positive instances.

In the ML task all-whale-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is the smallest between the ML tasks.
- Across seeds, it is evident that not including derivatives generally leads to higher accuracy, precision, and recall compared to including derivatives.
- We note that True-185 is the only model that performs better when including derivatives. This model is also the model that achieved the best overall result across the seeds.
- The highest average performance across seeds is achieved by not including derivatives. With an improvement of 0.68% in regard to including derivatives.

We can conclude that the models generalise better than the blue-fin-background ML task. Models are not overfitting because the models perform well in both the validation and the test set, with high precision and recall scores. The confusion matrices for these experiments are shown in Figures A.13– A.18.

MFCCs-2

The goal of this experiment is to optimise the hyperparameters and to see whether the deltas of the MFCCs will have an effect on the validation and test accuracy of each ML task. The chunk size used in this section is two seconds in duration. The results of this

section will be compared with the results of Section 5.7.1 (Fbanks-2) to determine which feature extraction method has the greatest impact on the results.

Table 5.6: Result for the ML tasks across the three seeds to see the impact of the deltas using MFCCs as the feature extraction method on 2-second chunks. **Bold** indicates the results for the best derivative option. *Italics* indicates the best result across the seeds.

Delta-Seed	Accuracy (%)		Precision		Recall	
	Valid	Test	Valid	Test	Valid	Test
blue-fin-all-else						
True-42	76.96	66.81	0.7082	0.6110	0.9167	0.9254
False-42	77.44	83.07	0.7290	0.9687	0.8733	0.6834
True-185	76.14	64.24	0.7017	0.5917	0.9089	0.9184
False-185	76.70	80.49	0.7195	0.9240	0.8750	0.6643
<i>True-1986</i>	<i>77.57</i>	<i>64.15</i>	<i>0.7191</i>	<i>0.5931</i>	<i>0.9045</i>	<i>0.9007</i>
False-1986	76.96	94.99	0.7082	0.9744	0.9167	0.9241
Avg-True	76.89	65.12	0.7097	0.9586	0.9100	0.9148
Std-True	0.5859	1.1994	0.0072	0.0088	0.0050	0.0104
Avg-False	77.05	86.18	0.7189	0.9557	0.8883	0.7573
Std-False	0.6457	6.3159	0.0085	0.0225	0.0201	0.1182
blue-fin-background						
True-42	96.62	89.03	0.9963	0.9717	0.9358	0.8040
False-42	95.48	93.82	0.9557	0.9215	0.9540	0.9579
True-185	91.63	82.89	0.9979	0.9941	0.8342	0.6617
<i>False-185</i>	<i>97.57</i>	<i>95.62</i>	<i>0.9841</i>	<i>0.9780</i>	<i>0.9670</i>	<i>0.9544</i>
True-1986	95.41	86.88	0.9834	0.9368	0.9236	0.7910
False-1986	96.70	95.01	0.9873	0.9501	0.9462	0.9501
Avg-True	94.55	86.27	0.9925	0.9675	0.8979	0.7522
Std-True	2.1253	2.5439	0.0065	0.0236	0.0453	0.0642
Avg-False	96.55	94.82	0.9757	0.9499	0.9557	0.9541
Std-False	0.8572	0.7475	0.0142	0.0231	0.0086	0.0032
all-whale-background						
True-42	93.63	93.08	0.9701	0.9559	0.9003	0.9033
False-42	92.58	93.24	0.9950	0.9878	0.8560	0.8755
<i>True-185</i>	<i>94.36</i>	<i>93.28</i>	<i>0.9654</i>	<i>0.9366</i>	<i>0.9202</i>	<i>0.9284</i>
False-185	93.58	93.76	0.9902	0.9898	0.8803	0.8842
True-1986	93.15	92.54	0.9797	0.9610	0.8812	0.8868
False-1986	93.54	93.76	0.9922	0.9719	0.8777	0.9011
Avg-True	93.71	92.97	0.9717	0.9512	0.9006	0.9062
Std-True	0.4975	0.3126	0.0060	0.0105	0.0159	0.0171
Avg-False	93.23	93.59	0.9925	0.9832	0.8713	0.8869
Std-False	0.4623	0.2451	0.0020	0.0080	0.0109	0.0106

We analyse Table 5.6 by looking at the impact of deltas in each ML task.

In the ML task blue-fin-all-else, from the results, we can conclude the following:

- The test set accuracy is generally higher than the validation set accuracy for not including derivatives (deltas). The potential cause of this could be the small data set as discussed in the previous experiment (Fbanks-2)
- Including derivatives, the validation set accuracy is higher than the test set accuracy. The reason for this might be the derivatives because it provides a more comprehensive representation for tasks that require sensitivity to temporal variations.
- The highest average performance across seeds is achieved by not including derivatives. With a slight improvement (0.16%) as regards to not including derivatives.
- Across seeds, it is also notable that True-1986 achieved the best overall results.

We can conclude when derivatives are included that the models are not generalising well but are not overfitting. When derivatives are not included, the models perform poorly as in the Fbanks-2.

In the ML task blue-fin-background, from the results, we can conclude the following:

- The difference between the validation set accuracy and test accuracy is notably better than the blue-fin-all-else. Where the validation accuracy is higher than the test accuracy.
- The difference between the validation accuracy and the test accuracy when not including derivatives is larger than when including derivatives. This can be considered acceptable due to the consistency between the three seeds. We note that the recall results dropped; this indicated that the models are struggling to correctly identify all positive instances in the data set.
- Across seeds, it is evident that not including derivatives (deltas) generally leads to higher accuracy compared to not including derivatives. The highest average

performance across seeds is achieved by not including derivatives (deltas). With an improvement of 2.00% in regard to not including derivatives.

- Across seeds, it is also notable that False-185 achieved the best overall results.

The only thing that is of concern is that when implementing derivatives (deltas), the validation accuracy is significantly higher (8.28%) than the test accuracy. There may be a few reasons that can result in this:

1. Data mismatch: Validation and test data sets may possess dissimilar features, for example, data distribution, or noise levels. The test set may contain more other biological sounds than the validation set.
2. Small validation set: If the validation set is not large enough, it may not be able to provide an accurate assessment of the model performance. In the next sections, where we experiment with the 4-second chunks, we can decide if this is a problem.

We can conclude that the models generalise and are not overfitting because the models perform well in both the validation and the test set, with high precision and recall scores on the validation set. In the ML task all-whale-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is the smallest between the ML tasks.
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy, precision, and recall compared to including derivatives.
- We note that False-1986 is the only model that performs better when not including derivatives.
- The highest average performance across seeds is achieved by including derivatives (deltas). With an improvement of 0.48% in regard to not including derivatives.

-
- Across seeds, it is also notable that True-185 achieved the best overall results.

We can conclude that the models generalise better than the blue-fin-background ML task. Models are not overfitting because the models perform well in both the validation and the test set, with high precision and recall scores. The confusion matrices for these experiments are shown in Figures A.19– A.24.

Discussion of Fbanks-2 and MFCCs-2

The key finding for Section 5.7.1 (Fbanks-2) and 5.7.1 (MFCCs-2) are discussed and a conclusion is made. The first thing we noticed is that the implementation of the derivative differed between the different ML tasks. For MFCCs-2 it seems that not implementing derivatives caused an improvement in the accuracy of the validation set, except for all-whale-background ML task. The implementation of derivatives improved the accuracy of the validation set for Fbanks-2, except for all-whale-background ML task. If we look at the overall implementation of derivatives across the individual seeds, the model performs best when the derivative is implemented.

Fbanks-4

The goal of this experiment is to optimise the hyperparameters and to see whether the deltas of the Fbanks will have an effect on the validation and test accuracy of each ML task. The chunk size used in this section is four seconds in duration. The results of this section will be compared with the results of Section 5.7.1 (MFCCs-4) to determine which feature extraction method has the greatest impact on the results.

Table 5.7: Result for the ML tasks across the three seeds to see the impact of the deltas using Fbanks as the feature extraction method on 4-second chunks. **Bold** indicates the results for the best derivative option. *Italics* indicates the best result across the seeds.

Delta-Seed	Accuracy (%)		Precision		Recall	
	Valid	Test	Valid	Test	Valid	Test
blue-fin-all-else						
True-42	77.24	74.48	0.8078	0.7664	0.7147	0.7042
False-42	76.59	73.53	0.7820	0.7313	0.7372	0.7437
True-185	77.94	74.59	0.7564	0.7083	0.8239	0.8361
False-185	74.51	74.07	0.7216	0.7062	0.7979	0.8244
<i>True-1986</i>	<i>78.11</i>	<i>74.11</i>	<i>0.7555</i>	<i>0.7049</i>	<i>0.8309</i>	<i>0.8296</i>
False-1986	76.68	71.88	0.7560	0.6958	0.7875	0.7775
Avg-True	77.76	74.39	0.7732	0.7265	0.7898	0.7900
Std-True	0.3765	0.2053	0.0244	0.0282	0.0532	0.0607
Avg-False	75.93	73.16	0.7532	0.7111	0.7742	0.7819
Std-False	1.0024	0.9316	0.0247	0.0149	0.0265	0.3309
blue-fin-background						
True-42	97.53	96.68	0.9982	0.9972	0.9523	0.9362
False-42	97.70	97.16	0.9955	0.9977	0.9584	0.9454
True-185	97.61	96.83	0.9982	0.9954	0.9540	0.9410
False-185	97.53	96.78	0.9981	0.9938	0.9523	0.9375
<i>True-1986</i>	<i>98.01</i>	<i>97.40</i>	<i>0.9920</i>	<i>0.9884</i>	<i>0.9679</i>	<i>0.9592</i>
False-1986	97.75	97.09	0.9964	0.9950	0.9584	0.9466
Avg-True	97.72	96.97	0.9961	0.9937	0.9581	0.9455
Std-True	0.2100	0.3102	0.0029	0.0038	0.0070	0.0099
Avg-False	97.66	97.01	0.9967	0.9955	0.9564	0.9432
Std-False	0.0942	0.1651	0.0011	0.0016	0.0029	0.0040
all-whale-background						
<i>True-42</i>	<i>97.57</i>	<i>97.11</i>	<i>0.9973</i>	<i>0.9959</i>	<i>0.9540</i>	<i>0.9462</i>
False-42	97.44	96.90	0.9972	0.9914	0.9514	0.9462
True-185	97.53	96.31	1.0000	0.9995	0.9506	0.9267
False-185	97.44	97.33	0.9955	0.9950	0.9532	0.9514
True-1986	97.40	97.29	0.9937	0.9905	0.9540	0.9549
False-1986	97.35	96.64	0.9946	0.9904	0.9523	0.9419
Avg-True	97.50	96.90	0.9972	0.9953	0.9529	0.9426
Std-True	0.0726	0.4259	0.0026	0.0037	0.0016	0.0118
Avg-False	97.41	96.96	0.9958	0.9923	0.9523	0.9465
Std-False	0.0424	0.2845	0.0011	0.0020	0.0008	0.0039

We analyse Table 5.7 by looking at the impact of deltas on each ML task.

In the ML task blue-fin-all-else, from the results, we can conclude the following:

- The difference between the validation set accuracy and the test accuracy is significantly better than the blue-fin-all-else for Fbanks-2. Where the validation accuracy is higher than the test accuracy. This means that the size of the chunks has a significant influence on the performance of the models.
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy, precision, and recall compared to not including derivatives. The highest average performance across seeds is achieved by including derivatives (deltas). With an improvement of 1.83% in regard to not including derivatives.
- Across seeds, it is also notable that True-1986 achieved the best overall results.

The performance of the models may not be exceptional, but it demonstrates a commendable level of generalisation. This might be due to the fact that the negative class includes many other biological sounds, including other whale calls. The other two ML tasks include fewer biological sounds and no other whale sounds. Importantly, there is no evidence of overfitting.

In the ML task blue-fin-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is small between seeds that include and exclude derivatives.
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy compared to not including derivatives. The highest average performance across seeds is achieved by including derivatives (deltas). With an improvement of 0.06% in regard to not including derivatives.
- Across seeds, it is also notable that True-1986 achieved the best overall results.

The performance of the models is better than the blue-fin-all-else. The models also generalise much better than the other two ML tasks. Importantly, there is no evidence of overfitting, as the models exhibit strong performance in both validation and test sets, evident from consistently high precision and recall scores.

In the ML task all-whale-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is the smallest between the ML tasks.
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy, precision, and recall compared to including derivatives.
- The highest average performance across seeds is achieved by including derivatives (deltas). With an improvement of 0.09% in regard to not including derivatives.
- Across seeds, it is also notable that True-42 achieved the best overall results.

We can conclude that the models generalise better than the blue-fin-background ML task. Models are not overfitting because the models perform well in both the validation and the test set, with high precision and recall scores. The confusion matrices for these experiments are shown in Figures A.25– A.30.

MFCCs-4

The goal of this experiment is to optimise the hyperparameters and to see whether the deltas of the MFCCs will have an effect on the validation and test accuracy of each ML task. The chunk size used in this section is four seconds in duration. The results of this section will be compared with the results of Section 5.7.1 (Fbanks-4) to determine which feature extraction method has the greatest impact on the results.

Table 5.8: Result for the ML tasks across the three seeds to see the impact of the deltas using MFCCs as the feature extraction method on 4-second chunks. **Bold** indicates the results for the best derivative option. *Italics* indicates the best result across the seeds.

Delta-Seed	Accuracy (%)		Precision		Recall	
	Valid	Test	Valid	Test	Valid	Test
blue-fin-all-else						
True-42	77.06	71.90	0.7294	0.6717	0.8604	0.8569
False-42	77.37	73.76	0.7717	0.7246	0.7771	0.7667
<i>True-185</i>	<i>78.54</i>	<i>75.33</i>	<i>0.7649</i>	<i>0.7184</i>	<i>0.8239</i>	<i>0.8330</i>
False-185	78.33	75.78	0.7525	0.7435	0.8439	0.7871
True-1986	76.20	73.76	0.7649	0.7233	0.7563	0.7697
False-1986	75.64	68.54	0.6867	0.6247	0.9428	0.9284
Avg-True	77.27	73.66	0.7531	0.7045	0.8135	0.8199
Std-True	0.9664	1.4020	0.0167	0.0233	0.0431	0.0368
Avg-False	77.11	72.69	0.7370	0.6996	0.8546	0.8274
Std-False	1.1131	3.0504	0.0364	0.0521	0.0681	0.0719
blue-fin-background						
True-42	97.88	96.96	0.9973	0.9883	0.9601	0.9505
False-42	97.66	97.31	0.9867	0.9906	0.9662	0.9553
True-185	98.05	97.14	0.9955	0.9945	0.9653	0.9479
False-185	92.11	95.03	0.9118	0.9175	0.9324	0.9896
<i>True-1986</i>	<i>98.05</i>	<i>97.38</i>	<i>0.9955</i>	<i>0.9914</i>	<i>0.9653</i>	<i>0.9557</i>
False-1986	92.97	97.04	0.9550	0.9605	0.9020	0.9813
Avg-True	97.99	97.16	0.9961	0.9914	0.9636	0.9514
Std-True	0.0801	0.1720	0.0008	0.0025	0.0025	0.0032
Avg-False	94.25	96.46	0.9512	0.9562	0.9335	0.9754
Std-False	2.4390	1.0172	0.0307	0.0230	0.0262	0.0146
all-whale-background						
True-42	97.18	97.03	0.9954	0.9963	0.9480	0.9440
False-42	97.18	96.85	0.9964	0.9954	0.9471	0.9414
True-185	97.31	97.38	0.9928	0.9828	0.9532	0.9644
False-185	97.31	96.92	0.9973	0.9950	0.9488	0.9432
<i>True-1986</i>	<i>97.70</i>	<i>97.23</i>	<i>0.9973</i>	<i>0.9919</i>	<i>0.9566</i>	<i>0.9531</i>
False-1986	97.22	97.11	0.9955	0.9954	0.9488	0.9466
Avg-True	97.40	97.21	0.9952	0.9903	0.9526	0.9538
Std-True	0.2210	0.1552	0.0018	0.0056	0.0035	0.0083
Avg-False	97.24	96.96	0.9964	0.9953	0.9482	0.9437
Std-False	0.0544	0.1098	0.0073	0.0019	0.0080	0.0022

We analyse Table 5.8 by looking at the impact of deltas on each ML task.

In the ML task blue-fin-all-else, from the results, we can conclude the following:

- The difference between the validation set accuracy and test accuracy is notably better than the blue-fin-all-else for MFCCs-2. Where the validation accuracy is higher than the test accuracy. This means that the size of the chunks has a significant influence on the performance of the models.
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy, precision, and recall compared to not including derivatives. The highest average performance across seeds is achieved by including derivatives (deltas). With an improvement of 0.16% in regard to not including derivatives on the validation set and 0.97% on the test set.
- Across seeds, it is also notable that True-185 achieved the best overall results.

The performance of the models may not be exceptional, but it demonstrates a commendable level of generalisation. This might be due to the fact that the negative class includes many other biological sounds, including other whale calls. The other two ML tasks include fewer biological sounds and no other whale sounds. Importantly, there is no evidence of overfitting.

In the ML task blue-fin-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is small between seeds that include derivatives.
- The test set accuracy is generally higher than the validation set accuracy when derivatives is not included. The potential cause of this could be the small data set. In a small data set, the random variation or noise in the validation set can have a considerable effect. This could lead to a less reliable estimation of the model's performance on the validation set in comparison to the test set.

-
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy compared to not including derivatives. The highest average performance across seeds is achieved by including derivatives (deltas). With an improvement of 3.74% in regard to not including derivatives.
 - Across seeds, it is also notable that True-1986 achieved the best overall results.

The performance of the models is better than the blue-fin-all-else. The models also generalise much better when derivatives are included. Importantly, there is no evidence of overfitting, as the models exhibit strong performance in both validation and test sets, evident from consistently high precision and recall scores.

In the ML task all-whale-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is the smallest between the ML tasks.
- Across seeds, it is evident that including derivatives (deltas) generally leads to higher accuracy and recall compared to including derivatives.
- The highest average performance across seeds is achieved by including derivatives (deltas). With an improvement of 0.16% in regard to not including derivatives in the validation set and 0.25% in the test set.
- Across seeds, it is also notable that True-1986 achieved the best overall results.

We can conclude that the models generalise better than the blue-fin-background ML task. Models are not overfitting because the models perform well in both the validation and the test set, with high precision and recall scores. The confusion matrices for these experiments are shown in Figures A.31– A.36.

Discussion of Fbanks-4 and MFCCs-4

The key findings for Section 5.7.1 (Fbanks-4) and 5.7.1 (MFCCs-4) are discussed. The first thing we noticed is that the implementation of the derivative improved the accuracy of all ML tasks on the validation set. The individual models per seed that include derivatives (True) performed better than those that do not include derivatives (False). We conclude that the best-performing model is MFCCs when implementing derivatives.

Conclusion

We conclude Section 5.7.1 by suggesting that SpeechBrain’s TDNN with x-vectors is a feasible option for whale call detection. Experiment Fbanks-4 and MFCCs-4 delivered comparable results to the baseline CNN on the blue-fin-background and all-whale-background ML tasks. We found that the four-second chunks delivered the best results according to validation accuracy. In both feature extraction methods (Fbanks and MFCCs) including the derivatives improved the validation accuracy results. Lastly, comparing the results of the two feature extraction methods of ML tasks Table 5.9 we found:

- Fbanks-True-4 performed the best for the blue-fin-all-else task.
- MFCCs-True-4 is the best option for the blue-fin background ML task when derivatives are included.
- Fbanks-True-4 performed better on the all-whale-background ML task when measuring validation accuracy, but the MFCCs-True-4 performed better using test accuracy. The MFCCs also had the highest validation accuracy (97.70%) for the individual seed 1986 when including derivatives.

Table 5.9: Summary of the average and standard deviation results for the ML tasks. The results are for the small data set using four-second chunks. The best model for each ML task is in **bold**.

ID	Stat	Accuracy (%)		Precision		Recall	
		Valid	Test	Valid	Test	Valid	Test
blue-fin-all-else							
Fbanks-True-4	Avg Std	77.76 0.3765	74.39 0.2053	0.7732 0.0244	0.7265 0.0282	0.7898 0.0532	0.7900 0.0607
Fbanks-False-4	Avg Std	75.93 1.0024	73.16 0.9316	0.7532 0.0247	0.7111 0.0149	0.7742 0.0265	0.7819 0.3309
MFCCs-True-4	Avg Std	77.27 0.9664	73.66 1.4020	0.7531 0.0167	0.7045 0.0233	0.8135 0.0431	0.8199 0.0368
MFCCs-False-4	Avg Std	77.11 1.1131	72.69 3.0504	0.7370 0.0364	0.6996 0.0521	0.8546 0.0681	0.8274 0.0719
blue-fin-background							
Fbanks-True-4	Avg Std	97.72 0.2100	96.97 0.3102	0.9961 0.0029	0.9937 0.0038	0.9581 0.0070	0.9455 0.0099
Fbanks-False-4	Avg Std	97.66 0.0942	97.01 0.1651	0.9967 0.0011	0.9955 0.0016	0.9564 0.0029	0.9432 0.0040
MFCCs-True-4	Avg Std	97.99 0.0801	97.16 0.1720	0.9961 0.0008	0.9914 0.0025	0.9636 0.0025	0.9514 0.0032
MFCCs-False-4	Avg Std	94.25 2.4390	96.46 1.0172	0.9512 0.0307	0.9562 0.0230	0.9335 0.0262	0.9754 0.0146
all-whale-background							
Fbanks-True-4	Avg Std	97.50 0.0726	96.90 0.4259	0.9972 0.0026	0.9953 0.0037	0.9529 0.0016	0.9426 0.0118
Fbanks-False-4	Avg Std	97.41 0.0424	96.96 0.2845	0.9958 0.0011	0.9923 0.0020	0.9523 0.0008	0.9465 0.0039
MFCCs-True-4	Avg Std	97.40 0.2210	97.21 0.1552	0.9952 0.0018	0.9903 0.0056	0.9526 0.0035	0.9538 0.0083
MFCCs-False-4	Avg Std	97.24 0.0544	96.96 0.1098	0.9964 0.0073	0.9953 0.0019	0.9482 0.0080	0.9437 0.0022

As a final remark we will experiment on the large_3_cs data set only on the four-second chunks.

5.7.2 Large_3_4 data set

We experiment with the large data set (large_3_cs) after concluding in Section 5.7.1 that the four-second chunks performed the best together with including derivatives (deltas) to the feature extraction method. We examined both feature extraction techniques since their results were similar. The reason for not experimenting with two-second chunks is that the large data set is very time-consuming and the best results are obtained with the duration of four-second chunks, as seen in Section 5.7.1.

Fbanks-large-4

In this section, we performed multiple experiments to determine how Fbanks with four-second chunks will have an effect on the accuracy using the large data set. Finally, we will also conclude as to whether the larger data set improved the model.

Table 5.10: Result for the ML tasks across the three seeds including deltas using Fbanks as the feature extraction. The large data set is used with 4-second chunks. **Bold** indicates the results for the average and standard deviation for the three seeds. *Italics* indicates the best result across the seeds.

Delta-Seed	Accuracy (%)		Precision		Recall	
	Valid	Test	Valid	Test	Valid	Test
blue-fin-all-else						
True-42	80.51	79.01	0.7702	0.7662	0.8668	0.8349
<i>True-185</i>	<i>80.99</i>	<i>79.13</i>	<i>0.7845</i>	<i>0.7790</i>	<i>0.8516</i>	<i>0.8135</i>
True-1986	78.66	77.80	0.7915	0.7894	0.7752	0.7582
Avg-True	80.05	78.65	0.7823	0.7782	0.8312	0.8022
Std-True	1.0045	0.6007	0.0086	0.0095	0.0401	0.0323
blue-fin-background						
True-42	99.56	96.39	0.9974	0.9968	0.9937	0.9307
<i>True-185</i>	<i>99.67</i>	<i>95.49</i>	<i>0.9988</i>	<i>0.9965</i>	<i>0.9946</i>	<i>0.9130</i>
True-1986	99.59	96.25	0.9970	0.9971	0.9949	0.9277
Avg-True	99.61	96.04	0.9977	0.9968	0.9944	0.9238
Std-True	0.0464	0.3954	0.0008	0.0002	0.0005	0.0077
all-whale-background						
<i>True-42</i>	<i>98.08</i>	<i>97.67</i>	<i>0.9908</i>	<i>0.9889</i>	<i>0.9706</i>	<i>0.9643</i>
True-185	97.73	97.62	0.9952	0.9936	0.9592	0.9586
True-1986	97.88	96.74	0.9886	0.9879	0.9687	0.9463
Avg-True	97.90	97.34	0.9915	0.9901	0.9662	0.9564
Std-True	0.1434	0.4271	0.0027	0.0025	0.0050	0.0075

We analyse Table 5.10 by looking at the impact of deltas on each ML task.

In the ML task blue-fin-all-else, from the results, we can conclude the following:

- The margin difference between the accuracy of the validation set and the accuracy of the test is small. The size of the chunks and the data set size have a considerable effect on the models' performance.
- Comparing the results with the results of the previous section (Fbanks-2 and Fbanks-4). We can conclude that the larger data set improved the effect of random variation or noise in the validation set. The larger data set delivered more stable results than the smaller data set.
- The average performance across all seeds (Avg-True) shows consistent and competitive results, indicating that the models generalise well.
- Across seeds, it is also notable that True-185 achieved the best overall results.

The performance of the models may not be exceptional compared to the other ML tasks. However, in contrast to the preceding section featuring Fbanks-2 and Fbanks-4 models, the ones discussed here showcase a commendable level of generalisation. Importantly, there is no evidence of overfitting.

In the ML task blue-fin-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy can be considered acceptable due to the consistency between the three seeds.
- Comparing the results with the results of the previous section (Fbanks-2 and Fbanks-4). We can conclude that the larger data set improved the overall performance of the models.
- It appears that test recall has a significant drop, which could indicate difficulties in detecting positive cases.

-
- Across seeds, it is also notable that True-185 achieved the best overall results.

Importantly, there is no evidence of overfitting, as the models exhibit strong performance in both validation and test sets, evident from consistently high precision and recall scores.

In the ML task all-whale-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is the smallest between the ML tasks.
- Comparing the results with the results of the previous section (Fbanks-2 and Fbanks-4). We can conclude that the larger data set improved the overall performance of the models.
- The average performance across seeds (Avg-True) demonstrates consistently high accuracy, precision, and recall. This also indicates a robust generalisation to different seeds.
- Across seeds, it is also notable that True-42 achieved the best overall results.

We can conclude that the models generalise better than the blue-fin-background ML task. Models are not overfitting because the models perform well in both the validation and the test set, with high accuracy, precision and recall scores.

MFCCs-4-large

In this section, we performed multiple experiments to determine how MFCCs with four-second chunks will have an effect on the accuracy using the large data set. Finally, we will also conclude whether the larger data set improved the model.

Table 5.11: Result for the ML tasks across the three seeds including deltas using MFCCs as the feature extraction. The large data set is used with 4-second chunks. **Bold** indicates the results for the average and standard deviation for the three seeds. *Italics* indicates the best result across the seeds.

Delta-Seed	Accuracy (%)		Precision		Recall	
	Valid	Test	Valid	Test	Valid	Test
blue-fin-all-else						
True-42	80.51	79.06	0.7944	0.7917	0.8206	0.7888
True-185	80.56	78.99	0.7574	0.7529	0.8963	0.8630
<i>True-1986</i>	<i>80.70</i>	<i>79.22</i>	<i>0.7739</i>	<i>0.7714</i>	<i>0.8645</i>	<i>0.8306</i>
Avg-True	80.59	79.09	0.7752	0.7720	0.8605	0.8275
Std-True	0.0804	0.0963	0.0151	0.0158	0.0310	0.0304
blue-fin-background						
True-42	99.56	96.64	0.9967	0.9947	0.9944	0.9378
True-185	99.59	96.33	0.9970	0.9956	0.9949	0.9307
<i>True-1986</i>	<i>99.63</i>	<i>96.39</i>	<i>0.9972</i>	<i>0.9967</i>	<i>0.9953</i>	<i>0.9308</i>
Avg-True	99.60	96.45	0.9970	0.9957	0.9949	0.9331
Std-True	0.0216	0.1342	0.0002	0.0008	0.0004	0.0033
all-whale-background						
True-42	97.72	97.60	0.9884	0.9860	0.9708	0.9659
True-185	97.95	97.64	0.9926	0.9893	0.9662	0.9632
<i>True-1986</i>	<i>98.02</i>	<i>97.73</i>	<i>0.9891</i>	<i>0.9892</i>	<i>0.9710</i>	<i>0.9651</i>
Avg-True	97.90	97.65	0.9900	0.9882	0.9693	0.9647
Std-True	0.1281	0.0544	0.0018	0.0015	0.0022	0.0011

We analyse Table 5.11 by looking at the impact of deltas on each ML task.

In the ML task blue-fin-all-else, from the results, we can conclude the following:

- The margin difference between the accuracy of the validation set and the accuracy of the test is small. The size of the chunks and the data set size have a considerable effect on the models' performance.
- Comparing the results with the results of the previous section (MFCCs-2 and MFCCs-4). The larger data set delivered more stable results than the smaller data set.
- The average performance across true seeds (Avg-True) is consistent and competitive, with small standard deviations indicating stability.
- Across seeds, it is also notable that True-1986 achieved the best overall results.

The performance of the models may not be exceptional compared to the other ML tasks. However, in contrast to the preceding section featuring MFCCs-2 and MFCCs-4 models, the ones discussed here showcase a commendable level of generalisation. Importantly, there is no evidence of overfitting.

In the ML task blue-fin-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy can be considered acceptable due to the consistency between the three seeds.
- Comparing the results with the results of the previous section (MFCCs-2 and MFCCs-4). We can conclude that the larger data set improved the overall performance of the models.
- The average performance across seeds (Avg-True) maintains high accuracy, precision, and recall, with low standard deviations indicating stability.
- True-1986 consistently outperforms other seeds, demonstrating the highest accuracy, precision, and recall on both validation and test sets.

Importantly, there is no evidence of overfitting, as the models exhibit strong performance in both validation and test sets, evident from consistently high precision and recall scores.

In the ML task all-whale-background, from the results, we can conclude the following:

- The difference between the validation accuracy and the test accuracy is the smallest between the ML tasks.
- Comparing the results with the results of the previous section (MFCCs-2 and MFCCs-4). We can conclude that the larger data set improved the overall performance of the models.
- The average performance across seeds (Avg-True) demonstrates consistently high accuracy, precision, and recall. This also indicates a robust generalisation to different seeds.
- Similar to other seeds, True-1986 stands out, achieving the best results in accuracy, precision, and recall in both validation and test sets.

We can conclude that the models generalise better than the blue-fin-background ML task. Models are not overfitting because the models perform well in both the validation and the test set, with high accuracy, precision and recall scores.

5.7.3 Conclusion

We can definitively conclude that the large data set improved all ML tasks performance compared to the smaller data set. All the models generalised well and did not overfit. When comparing the two feature extraction methods, both performed similarly for the average result for the validation set. Except MFCCs performed on average better for the blue-fin-all-else ML task. MFCCs out performed Fbanks on the average accuracy of the test set in all ML tasks. We conclude that using four-second chunks with MFCCs as a feature extraction method on a larger data set delivered the best results overall.

5.8 TDNN vs baseline CNN

In this section, we compare the results of the baseline CNN in Chapter 4 with the results that are comparable in this chapter. A result that is directly comparable to the baseline CNN is the blue-fin-background ML task in Section 5.7.1.

In Table 5.13 the first part (CNN results) shows the averages and standard deviation results obtained for the baseline CNN in Chapter 4. The results are for the optimisers SGD and Adam. The second part of the table (TDNNs blue-fin-background) shows the average and standard deviation results obtained for the TDNNs in this chapter. All the results in this table are on the small_3_cs data set for the two- and four-second chunks.

Analysing the results in Table 5.13 we can conclude the following:

- The two-second chunks in both cases performed worse overall than the four-second chunks.
- The TDNNs performed better on both the Fbanks and MFCCs for the two-second chunks than the baseline CNN.
- Adam-4 are the best models for the baseline CNN and MFCC-True-4 are the best models for the TDNN. These models are comparable on the validation accuracy, but the CNN models produced a slightly higher test accuracy (Adam-4: 97.86% vs MFCCs-True-4: 97.16%).

We can calculate the confidence intervals for the different models. In this case, we are using a confidence level of 95% and assume a Gaussian distribution of means, as is typical for averages over samples.

Using the following equation, we can calculate the confidence level for each model [124].

$$\left[\hat{\mu}_n - 1.96 \left(\frac{\hat{\sigma}_n}{\sqrt{n}} \right), \hat{\mu}_n + 1.96 \left(\frac{\hat{\sigma}_n}{\sqrt{n}} \right) \right] \quad (5.1)$$

where $\hat{\mu}_n$ is the sample mean, $\hat{\sigma}_n$ is the sample standard deviation and n is the number of samples (here 3, given the 3 random seeds).

We calculate the confidence interval for the two-second chunks for the test set to determine if the differences are statistically significant.

Table 5.12: Confidence intervals for the two-second chunks for the test set.

Model	Confidence Interval
SGD-2	(87.98, 89.26)
Adam-2	(86.94, 89.86)
Fbanks-True-2	(91.42, 94.50)
Fbanks-False-2	(91.65, 95.29)
MFCC-True-2	(81.85, 90.68)
MFCC-False-2	(93.53, 96.11)

Therefore, based on the confidence intervals provided, SGD-2 and Adam-2 are statistically significant compared to Fbanks and MFCCs. Additionally, Fbanks and MFCCs also appear to have statistical significance between themselves.

Thus, we can conclude that an end-to-end TDNN system is comparable to the baseline CNN approach. The benefits we get from the TDNN system are that feature extraction is less computationally expensive. The time it took to run the experiments on the same data set was faster on the TDNN in comparison to the CNN.

Table 5.13: Summary of the average and standard deviation for the CNN results from Chapter 4 and the small_3_cs result from this chapter. The model with the best validation accuracy for the CNN and TDNNs is **bold**.

ID	Stat	Accuracy (%)		Precision		Recall	
		Valid	Test	Valid	Test	Valid	Test
CNN results							
SGD-2	Avg	88.79	88.62	0.9212	0.9248	0.8452	0.8436
	Std	0.4992	0.6182	0.0018	0.0012	0.0131	0.0142
Adam-2	Avg	88.61	88.40	0.9121	0.9192	0.8519	0.8453
	Std	0.9103	1.3403	0.0178	0.0176	0.0219	0.0311
SGD-4	Avg	97.38	97.34	0.9933	0.9946	0.9529	0.9525
	Std	0.1870	0.3606	0.0038	0.0039	0.0047	0.0104
Adam-4	Avg	97.97	97.86	0.9662	0.9918	0.9666	0.9657
	Std	0.0424	0.1337	0.0060	0.0054	0.0052	0.0031
TDNNs blue-fin-background							
Fbanks-True-2	Avg	96.88	92.96	0.9746	0.9081	0.9623	0.9563
	Std	0.3923	1.3916	0.0085	0.0281	0.0056	0.0057
Fbanks-False-2	Avg	94.44	93.47	0.9685	0.9623	0.9603	0.9559
	Std	0.5152	1.6618	0.0137	0.0056	0.0094	0.0093
MFCC-True-2	Avg	94.55	86.2667	0.9925	0.9675	0.8979	0.7522
	Std	2.1253	2.5439	0.0065	0.0236	0.0453	0.0642
MFCC-False-2	Avg	96.55	94.82	0.9757	0.9499	0.9557	0.9541
	Std	0.8572	0.7475	0.0142	0.0231	0.0086	0.0032
Fbanks-True-4	Avg	97.72	96.97	0.9961	0.9937	0.9581	0.9455
	Std	0.2100	0.3102	0.0029	0.0038	0.0070	0.0099
Fbanks-False-4	Avg	97.66	97.01	0.9967	0.9955	0.9564	0.9432
	Std	0.0942	0.1651	0.0011	0.0016	0.0029	0.0040
MFCC-True-4	Avg	97.99	97.16	0.9961	0.9914	0.9636	0.9514
	Std	0.0801	0.1720	0.0008	0.0025	0.0025	0.0032
MFCC-False-4	Avg	94.25	96.46	0.9512	0.9562	0.9335	0.9754
	Std	2.4390	1.0172	0.0307	0.0230	0.0262	0.0146

5.9 Performance across sites

Good results were achieved by training and testing on different data sets captured at the same site. How well does a model trained on one site perform on data collected from a different site? To answer this, we assess the effectiveness of CNNs as detailed in Chapter 4 and TDNNs with x-vectors, outlined in this chapter, specifically in the context of whale call detection across sites. Evaluation involves the use of the best performing models identified in both Chapter 4 (Adam-4 in Table 5.13) and for the small_3.cs data set in this chapter (MFCC-True-4 in Table 5.13).

The data set used in this section also comes from the AADC data set mentioned and prepared in Chapter 3. The main difference is that the sites we use are Kerguelen 2014 and 2015. The reason for this is that the best model from Chapters 4 and this chapter, as mentioned above, will be evaluated on these unseen sites to see how well the models will perform.

5.9.1 Data Set

The data set is shown in Table 5.14 where there is only an evaluation set.

Table 5.14: Size of datasets used for experimentation.

Id	Set	Calls	No Calls	Total	Duration (hours)
transfer_3_4	Evaluate	10,000	10,000	20,000	22.22

5.9.2 Approach

To evaluate the performance of the selected models on the transfer_3_4 data set, the following happened: (1) the evaluation data set was prepared for the CNN and TDNN models and (2) the selected models for both the CNN and TDNN were loaded and inference was done on the evaluation set.

5.9.3 Analysis and Results

The following results were obtained.

Table 5.15: Transfer learning results.

Id	Accuracy (%)	Precision	Recall
cnn_adam_4	63.21	0.5786	0.9726
tdnn_mfcc_4	54.08	0.5387	0.5664

5.9.4 Conclusion

We found that both models perform poorly and that site-specific fine-tuning will be required. We leave this for future work.

5.10 Conclusion

A TDNN-based approach is introduced and developed in this chapter, where an initial out-of-the-box run was performed on the small_3_cs data set. The out-of-the-box TDNN with x-vector performed very poorly in detecting whale calls. We then describe what SpeechBrain is and how it is adapted for whale call detection. In addition, we explain the experimental setup and the optimisation protocol and how the different ML tasks are framed. Taking everything into account, we move on to how and why we performed the different experiments in Section 5.7.

As a result, we found that for the data sets in Chapter 3, the novel approach developed in this chapter delivered results that are comparable to the baseline CNN developed in Chapter 4. We also found that models developed at one site are not directly applicable to data collected from different sites: fine-tuning to the new domain will be required for both the CNN and TDNN. We conclude that a TDNN with x-vectors is a viable approach for whale call detection and a viable avenue for further exploration.

Chapter 6

Conclusion

The key findings for this study are summarised and discussed, followed by the contributions made. We also reflect on the possibilities for future work that could use SpeechBrain, but more specifically TDNNs with ASR-like features for whale call detection.

6.1 Introduction

This study investigated an alternative approach to the task of whale call detection using a TDNN with ASR-like features. In this chapter, we discuss the key findings of this study and the contributions made. The original research objectives are reviewed with the key findings in mind. We conclude with suggestions for future research that could use the approach proposed in this study.

6.2 Key findings

We introduce a new approach for whale call detection using TDNNs with ASR-like features in this study. We developed a baseline CNN, which is the standard approach for whale call detection to compare with the new approach. The key findings of the study include:

- While establishing the baseline CNN we found that extracting the spectrograms from the audio files is computationally expensive and takes a lot of time. The reason for this was that the audio files needed preprocessing, which is discussed in Section 3.3. We looked at different tools that help with preprocessing and spectrogram extraction and found that the best tool to automate the process was Python.
- While developing and configuring the CNN baseline, we found that a standard CNN developed using the AADC data set (discussed in Chapter 3) performed comparable to well-established previous CNNs. We investigated the effect of Adam and SGD as the optimiser while changing the chunk size of the spectrograms between two and four seconds. The size of the chunks of the spectrogram had the greatest influence on the performance. Adam performed the best in the four-second chunk size (A-4: 97.86% and S-4: 97.34%). SGD performed the best in the two-second chunk size (A-2: 88.40% and S-2: 88.62%). Overall, we concluded that the four-second chunks, with Adam as optimiser, delivered the best results.
- When we started analysing the data set in more depth in Chapter 3 and performing some statistical analysis, we found that some human error is introduced into the data set, as discussed in Section 3.2.2. We tried to eliminate as many of these human errors as possible, but there may be more that we are unaware of. In the statistical analysis, we calculated some useful information that helped to better understand the data and to decide on the spectrogram chunk size.
- In the literature review in Chapter 2, we realised that there are no studies on whale call detection using ASR-like features with an end-to-end DNN system. We found that for the TDNN system of SpeechBrain and using a strong hyperparameter tuning

protocol, the only preprocessing that was necessary was the downsampling to 300Hz to make it comparable to the baseline CNN.

- During the experimentation stage of the TDNN system, we found that the ASR-like feature that performed the best is MFCCs with derivatives.
- In the end-to-end DNN system we found that it is less computationally expensive to extract the features and to train the model than using the baseline CNN. The TDNN system provided results comparable to the baseline CNN and it is less computationally expensive.
- We also found that models developed at one site are not directly applicable to data collected from different sites: fine-tuning to the new domain will be required for both the CNN and TDNN.

6.3 Contributions

During this study, several contributions are made to the field of marine bioacoustics and deep learning by introducing novel approaches to whale call detection. The contributions include:

- A novel approach to whale call detection was developed using an end-to-end TDNN system with ASR-like features. This approach is very different compared to the standard CNN approach for whale call detection.
- Minimal preprocessing for the end-to-end system makes it more computationally efficient and time efficient than the standard CNN approach with spectrograms. The extraction of the spectrograms and preprocessing them is computationally expensive and time-consuming.
- The effects of different feature extraction methods and their effects on whale call detection were considered. We concluded that the ASR-like feature MFCCs performed just as well as the standard spectrogram feature extraction method.

-
- The research carried out in this study serves as a new and more efficient way to detect whale calls. This encourages future work on deep learning techniques using the end-to-end system with ASR-like features for research on marine mammals.

6.4 Future work

The further development of SpeechBrain’s TDNN will focus on improving the network architecture for whale call detection and later implementing it for whale call classification.

The work will aim to address the following:

- Explore alternative TDNN architectures like the ECAPA-TDNN mentioned in SpeechBrain’s Hugging Face [12].
- Alternative hyperparameter optimisation methods, expanding the hyperparameter search space to more hyperparameter and broader ranges. Other fine-tuning methods will be considered.
- The end-to-end SpeechBrain system developed in this study is to be adapted to whale call classification to differentiate between the species types.
- Investigate the application of transfer learning techniques to help label corpora from partially matched domains. It is possible that adapting models for whale call detection could result in better results than developing new models from scratch, particularly when there is limited labelled data.

6.5 Conclusion

With the increase of whale call recordings, labelling these corpora becomes expensive and time-consuming, especially if a professional has to do it. Throughout this study, we researched the state-of-the-art approach (CNNs with spectrogram) to whale call detection.

We found that it is computationally expensive to extract spectrograms and that little to no research is done on whale call detection using an end-to-end TDNN system with x-vectors.

We proposed a novel approach using an end-to-end TDNN system with x-vectors from SpeechBrain with ASR-like features for whale call detection. This approach in Chapter 5 proved: (1) minimal preprocessing requirements that streamline the process, and (2) delivered results that are directly comparable to the baseline CNN in Chapter 4.

This research serves as a basis for further investigations into whale call detection, motivating exploration and experimentation with more complex TDNN models, transfer learning methods, and the ability to classify different whale species.

Bibliography

- [1] A. D. Davidson, A. G. Boyer, H. Kim, *et al.*, “Drivers and hotspots of extinction risk in marine mammals,” *Proceedings of the National Academy of Sciences*, vol. 109, no. 9, pp. 3395–3400, 2012.
- [2] J.-j. Jiang, L.-r. Bu, F.-j. Duan, *et al.*, “Whistle detection and classification for whales based on convolutional neural networks,” *Applied Acoustics*, vol. 150, pp. 169–178, 2019.
- [3] D. Cato, R. McCauley, T. Rogers, and M. Noad, “Passive acoustics for monitoring marine animals-progress and challenges,” in *Proceedings of ACOUSTICS*, vol. 2006, 2006, pp. 453–460.
- [4] M. Harvey. “Acoustic detection of humpback whales using a convolutional neural network.” (2018), [Online]. Available: <https://ai.googleblog.com/2018/10/acoustic-detection-of-humpback-whales.html>. (accessed: 23.07.2021).
- [5] Y. Shiu, K. Palmer, M. A. Roch, *et al.*, “Deep neural networks for automated detection of marine mammal species,” *Scientific reports*, vol. 10, no. 1, p. 607, 2020.
- [6] J.-j. Jiang, L.-r. Bu, X.-q. Wang, *et al.*, “Clicks classification of sperm whale and long-finned pilot whale based on continuous wavelet transform and artificial neural network,” *Applied acoustics*, vol. 141, pp. 26–34, 2018.
- [7] C. Bergler, H. Schröter, R. X. Cheng, *et al.*, “Orca-spot: An automatic killer whale sound detection toolkit using deep learning,” *Scientific reports*, vol. 9, no. 1, p. 10997, 2019.

-
- [8] P. C. Bermant, M. M. Bronstein, R. J. Wood, S. Gero, and D. F. Gruber, “Deep machine learning techniques for the detection and classification of sperm whale bioacoustics,” *Scientific reports*, vol. 9, no. 1, p. 12 588, 2019.
- [9] J. C. Hassab, *Underwater signal and data processing*. CRC Press, 2018.
- [10] S. Nisar, O. U. Khan, M. Tariq, *et al.*, “An efficient adaptive window size selection method for improving spectrogram visualization,” *Computational intelligence and neuroscience*, vol. 2016, 2016.
- [11] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, “X-vectors: Robust dnn embeddings for speaker recognition,” in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, IEEE, 2018, pp. 5329–5333.
- [12] M. Ravanelli, T. Parcollet, P. Plantinga, *et al.*, “Speechbrain: A general-purpose speech toolkit,” *arXiv preprint arXiv:2106.04624*, 2021.
- [13] Miller, Brian S and Balcazar, Naysa and IWC-SORP/SOOS Acoustic Trends Working Group *et al.*, “An open access dataset for developing automated detectors of antarctic baleen whale sounds and performance evaluation of two commonly used detectors,” *Scientific Reports*, vol. 11, no. 1, p. 806, 2021.
- [14] Z. Zhang, J. Geiger, J. Pohjalainen, A. E.-D. Mousa, W. Jin, and B. Schuller, “Deep learning for environmentally robust speech recognition: An overview of recent developments,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 5, pp. 1–28, 2018.
- [15] E. Fourie, M. H. Davel, and J. Versfeld, “Neural speech processing for whale call detection,” in *Artificial Intelligence Research (SACAIR 2022)*, *Communications in Computer and Information Science*, vol. 1734, Nov. 2022, pp. 276–290.
- [16] National Oceanic and Atmospheric Administration. “Why do whales make sounds?” (1970), [Online]. Available: <https://oceanservice.noaa.gov/facts/whalesounds.html>. (accessed: 23.07.2021).
- [17] G. Jones, “Echolocation,” *Current Biology*, vol. 15, no. 13, R484–R488, 2005.
-

-
- [18] F. Al Abbar. “Why do sperm whales use so many types of clicks?” (2019), [Online]. Available: <https://whalewatchingazores.com/blog/why-do-sperm-whales-use-so-many-types/>. (accessed: 23.07.2021).
- [19] M. Bahoura and Y. Simard, “Blue whale calls classification using short-time fourier and wavelet packet transforms and artificial neural network,” *Digital Signal Processing*, vol. 20, no. 4, pp. 1256–1263, 2010.
- [20] M. Panzner and P. Cimiano, “Comparing hidden markov models and long short term memory neural networks for learning action representations,” in *Machine Learning, Optimization, and Big Data: Second International Workshop, MOD 2016, Volterra, Italy, August 26-29, 2016, Revised Selected Papers 2*, Springer, 2016, pp. 94–105.
- [21] T. Lu, B. Han, and F. Yu, “Detection and classification of marine mammal sounds using alexnet with transfer learning,” *Ecological Informatics*, vol. 62, p. 101 277, 2021.
- [22] H. Yuea, D. Wanga, L. Zhanga, Y. Wua, C. Baoa, and D. Wang, “Classification of whale calls based on transfer learning and convolutional neural network,” in *4th Underwater Acoustics Conference and Exhibition*, 2017, pp. 537–544.
- [23] A. Usman and D. Versfeld, “Detection of baleen whale species using kernel dynamic mode decomposition-based feature extraction with a hidden markov model,” *Ecological Informatics*, vol. 71, p. 101 766, 2022.
- [24] D. Gillespie, “Detection and classification of right whale calls using an ‘edge’ detector operating on a smoothed spectrogram,” *Canadian Acoustics*, vol. 32, no. 2, pp. 39–47, 2004.
- [25] D. Gillespie, “6th International Workshop on the Detection, Classification, Localization and Density Estimation of Marine Mammals,” Saint Andrews University (United Kingdom), Tech. Rep., 2013.
- [26] J. K. Ford, “Acoustic behaviour of resident killer whales (*Orcinus orca*) off vancouver island, british columbia,” *Canadian Journal of Zoology*, vol. 67, no. 3, pp. 727–745, 1989.

-
- [27] S. Ness, *The Orhive: A system for semi-automatic annotation and analysis of a large collection of bioacoustic recordings*. University of Victoria (Canada), 2013.
- [28] OrcaLab. “Orcalab - A Whale Research Station on Hanson Island.” (1970), [Online]. Available: <http://orcalab.org/>. (accessed: 23.07.2021).
- [29] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [30] Pacific Northwest Seismic Network. “What is a spectrogram?” (2012), [Online]. Available: <https://pnsn.org/spectrograms/what-is-a-spectrogram>. (accessed: 23.07.2021).
- [31] *Frequency*, 2022. [Online]. Available: <https://encyclopedia.pub/entry/34756>, (accessed: 17.02.2023).
- [32] The MathWorks, Inc. “Spectrogram computation in signal analyzer.” (1994), [Online]. Available: <https://www.mathworks.com/help/signal/ug/spectrogram-computation-in-signal-analyzer.html>. (accessed: 1.11.2022).
- [33] M. Learnere. “Learning from audio: Spectrograms.” (2021), [Online]. Available: <https://www.mathworks.com/help/signal/ug/spectrogram-computation-in-signal-analyzer.html>. (accessed: 1.11.2022).
- [34] J. Choi, B. Feeny, S. Peshin, *et al.* “Understanding FFT Windows.” (2013), [Online]. Available: https://www.egr.msu.edu/classes/me451/me451_labs/Fall_2013/Understanding_FFT_Windows.pdf. (accessed: 19.05.2022).
- [35] M. Cerna and A. F. Harvey, “The fundamentals of fft-based signal analysis and measurement,” Citeseer, Tech. Rep., 2000.
- [36] P. Podder, T. Z. Khan, M. H. Khan, and M. M. Rahman, “Comparative performance analysis of hamming, hanning and blackman window,” *International Journal of Computer Applications*, vol. 96, no. 18, pp. 1–7, 2014.

-
- [37] The MathWorks Inc., *Spectrogram using short-time Fourier transform - MATLAB spectrogram*, Natick, Massachusetts, United States, 2022. [Online]. Available: https://www.mathworks.com/help/signal/ref/spectrogram.html#responsive_offcanvas, (accessed: 17.11.2023).
- [38] National Instruments Corporation. “Zero Padding.” (2023), [Online]. Available: https://www.ni.com/docs/en-US/bundle/labwindows-cvi/page/advancedanalysisconcepts/lvac_zero_padding.html. (accessed: 25.11.2023).
- [39] R. Phan. “What is a spectrogram and how do I set its parameters?” (2015), [Online]. Available: <https://stackoverflow.com/questions/29321696/what-is-a-spectrogram-and-how-do-i-set-its-parameters>. (accessed: 25.11.2023).
- [40] C. RA, W. AM, and S. LM, *Raven pro 1.4 user’s manual*. Cornell Lab of Ornithology, Ithaca, NY, 2010.
- [41] N.-A. Pham, A. Morrison, J. Schwock, *et al.*, “Quantitative image analysis of immunohistochemical stains using a cmyk color model,” *Diagnostic pathology*, vol. 2, no. 1, pp. 1–10, 2007.
- [42] Electrical4U. “Filter Bank: What is it? (DCT, Polyphase, Gabor, Mel And FBMC).” (1970), [Online]. Available: <https://www.electrical4u.com/filter-bank/>. (accessed: 25.08.2022).
- [43] S. Majeed, H. HUSAIN, S. Samad, and T. Idbeaa, “Mel frequency cepstral coefficients (mfcc) feature extraction enhancement in the application of speech recognition: A comparison study,” *Journal of Theoretical and Applied Information Technology*, vol. 79, pp. 38–56, Sep. 2015.
- [44] O. Ogundile, A. Usman, O. Babalola, and D. Versfeld, “Dynamic mode decomposition: A feature extraction technique based hidden markov model for detection of mysticetes’ vocalisations,” *Ecological Informatics*, vol. 63, p. 101 306, 2021.
- [45] H. M. Fayek, *Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What’s In-Between*, 2016. [Online]. Available: <https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html>, (accessed: 17.11.2023).
-

-
- [46] C. Mateo and J. A. Talavera, “Short-time fourier transform with the window size fixed in the frequency domain,” *Digital Signal Processing*, vol. 77, pp. 13–21, 2018.
- [47] R. Owens. “Fourier transform theory.” (1997), [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT4/node2.html. (accessed: 06.02.2024).
- [48] The MathWorks Inc., *Short-time Fourier transform*, Natick, Massachusetts, United States, 2022. [Online]. Available: <https://www.mathworks.com/help/signal/ref/stft.html>, (accessed: 17.11.2023).
- [49] S. Mudadla. “Why we have to remove highly correlated features in Machine Learning?” (2023), [Online]. Available: <https://medium.com/@sujathamudadla1213/why-we-have-to-remove-highly-correlated-features-in-machine-learning-9a8416286f18#:~:text=Explanation%3A%20Models%20trained%20on%20datasets,making%20the%20model%20more%20robust>. (accessed: 06.02.2024).
- [50] M. Gupta and A. K. Garg, “Analysis of image compression algorithm using dct,” *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 1, pp. 515–521, 2012.
- [51] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.
- [52] J. Gu, Z. Wang, J. Kuen, *et al.*, “Recent advances in convolutional neural networks,” *Pattern recognition*, vol. 77, pp. 354–377, 2018.
- [53] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture of monkey striate cortex,” *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.
- [54] K. Fukushima and S. Miyake, “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,” in *Competition and cooperation in neural nets*, Springer, 1982, pp. 267–285.
- [55] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.
-

-
- [56] S. Agatonovic-Kustrin and R. Beresford, “Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research,” *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.
- [57] H. Patel, A. Thakkar, M. Pandya, and K. Makwana, “Neural network with deep learning architectures,” *Journal of Information and Optimization Sciences*, vol. 39, no. 1, pp. 31–38, 2018.
- [58] S. Mostafa and F.-X. Wu, “Diagnosis of autism spectrum disorder with convolutional autoencoder and structural mri images,” in *Neural Engineering Techniques for Autism Spectrum Disorder*, Elsevier, 2021, pp. 23–38.
- [59] R. Kadam, *Kernels (filters) in Convolutional Neural Network (CNN), let’s talk about them*. Aug. 2021. [Online]. Available: <https://medium.com/codex/kernels-filters-in-convolutional-neural-network-cnn-lets-talk-about-them-ee4e94f3319>, (accessed: 17.11.2023).
- [60] F. Dewangan, A. Y. Abdelaziz, and M. Biswal, “Load forecasting models in smart grid using smart meter information: A review,” *Energies*, vol. 16, no. 3, p. 1404, 2023.
- [61] C. Kong and S. Lucey, “Take it in your stride: Do we need striding in cnns?” *arXiv preprint arXiv:1712.02502*, 2017.
- [62] GeeksforGeeks, *CNN Introduction to padding*, May 2023. [Online]. Available: <https://www.geeksforgeeks.org/cnn-introduction-to-padding/>, (accessed: 17.11.2023).
- [63] M. Mandal, *Introduction to Convolutional Neural Networks (CNN)*, Nov. 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/#:~:text=Convolutional%20neural%20networks%20are%20composed,and%20outputs%20an%20activation%20value.>, (accessed: 17.11.2023).
- [64] W. Jung, D. Jung, B. Kim, S. Lee, W. Rhee, and J. H. Ahn, “Restructuring batch normalization to accelerate cnn training,” *Proceedings of Machine Learning and Systems*, vol. 1, pp. 14–26, 2019.
-

-
- [65] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pmlr, 2015, pp. 448–456.
- [66] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” *Advances in neural information processing systems*, vol. 31, 2018.
- [67] J. Brownlee, “A gentle introduction to pooling layers for convolutional neural networks,” *Machine Learning Mastery*, vol. 22, 2019.
- [68] D. Unzueta, *Fully Connected Layer vs. Convolutional Layer: Explained*, Oct. 2022. [Online]. Available: <https://builtin.com/machine-learning/fully-connected-layer>, (accessed: 17.11.2023).
- [69] A. Mehrish, N. Majumder, R. Bharadwaj, R. Mihalcea, and S. Poria, “A review of deep learning techniques for speech processing,” *Information Fusion*, p. 101 869, 2023.
- [70] N. Vaessen and D. A. Van Leeuwen, “Fine-tuning wav2vec2 for speaker recognition,” in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2022, pp. 7967–7971.
- [71] M. Singh and D. Pati, “Replay attack detection using excitation source and system features,” in *Advances in Ubiquitous Computing*, Elsevier, 2020, pp. 17–44.
- [72] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks,” in *Backpropagation*, Psychology Press, 2013, pp. 35–61.
- [73] B. Desplanques, J. Thienpondt, and K. Demuynck, “Ecapa-tdnn: Emphasized channel attention, propagation and aggregation in tdnn based speaker verification,” *arXiv preprint arXiv:2005.07143*, 2020.
- [74] V. Peddinti, D. Povey, and S. Khudanpur, “A time delay neural network architecture for efficient modeling of long temporal contexts,” in *Sixteenth annual conference of the international speech communication association*, 2015.
-

-
- [75] J. Profant, “Robust speaker verification using deep neural networks,” 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:189919971>, (accessed: 15.11.2022).
- [76] S. Tchistiakova, *Time delay neural network*, Nov. 2019. [Online]. Available: <https://kaleidoescape.github.io/tdnn/>, (accessed: 15.11.2022).
- [77] B. Fortuner, *Concepts - bias*, 2020. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html#bias, (accessed: 19.02.2024).
- [78] D. Snyder, D. Garcia-Romero, D. Povey, and S. Khudanpur, “Deep neural network embeddings for text-independent speaker verification.,” in *Interspeech*, vol. 2017, 2017, pp. 999–1003.
- [79] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.
- [80] C. Hashemi-Pour, A. Zola, and J. Vaughan, *What is a backpropagation algorithm and how does it work?* Aug. 2022. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/backpropagation-algorithm#:~:text=Backpropagation%2C%20or%20backward%20propagation%20of,data%20mining%20and%20machine%20learning>, (accessed: 15.03.2023).
- [81] J. Brownlee, *Gentle introduction to the adam optimization algorithm for deep learning*, Jan. 2021. [Online]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>, (accessed: 15.06.2022).
- [82] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [83] GeeksforGeeks, *ML Stochastic Gradient Descent SGD*, May 2023. [Online]. Available: <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>, (accessed: 25.11.2023).
- [84] V. Alto, “Neural networks: Parameters, hyperparameters and optimization strategies,” *Medium. Towards Data Science. July*, vol. 5, 2019.
-

-
- [85] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.
- [86] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, “Hyperparameter optimization for machine learning models based on bayesian optimization,” *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.
- [87] R. Silipo, *Machine learning algorithms and the art of hyperparameter selection*, Feb. 2020. [Online]. Available: <https://towardsdatascience.com/machine-learning-algorithms-and-the-art-of-hyperparameter-selection-279d3b04c281>, (accessed: 15.06.2022).
- [88] G. A. Lujan-Moreno, P. R. Howard, O. G. Rojas, and D. C. Montgomery, “Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study,” *Expert Systems with Applications*, vol. 109, pp. 195–205, 2018.
- [89] M. Uzair and N. Jamil, “Effects of hidden layers on the efficiency of neural networks,” in *2020 IEEE 23rd international multitopic conference (INMIC)*, IEEE, 2020, pp. 1–6.
- [90] J. Brownlee, *Understand the impact of learning rate on neural network performance*, Sep. 2020. [Online]. Available: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>.
- [91] Y. Li, C. Wei, and T. Ma, “Towards explaining the regularization effect of initial large learning rate in training neural networks,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [92] P. Baheti, *Activation functions in neural networks [12 types amp; use cases]*, May 2021. [Online]. Available: <https://www.v7labs.com/blog/neural-networks-activation-functions>, (accessed: 15.06.2022).
- [93] A. I. Rodriguez and X. D. Buitrago, “How to choose an activation function for deep learning,” *Tekhnê*, vol. 19, no. 1, pp. 23–32, 2022.
-

-
- [94] S. Sharma, S. Sharma, and A. Athaiya, “Activation functions in neural networks,” *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.
- [95] P. Potrimba, *What is an activation function? a complete guide*. Jan. 2023. [Online]. Available: <https://blog.roboflow.com/activation-function-computer-vision/>.
- [96] J. Brownlee, “What is the difference between a batch and an epoch in a neural network,” *Machine Learning Mastery*, vol. 20, 2018.
- [97] D. Huynh, *How to get 4x speedup and better generalization using the right batch size*, Nov. 2019. [Online]. Available: <https://towardsdatascience.com/implementing-a-batch-size-finder-in-fastai-how-to-get-a-4x-speedup-with-better-generalization-813d686f6bdf>, (accessed: 15.06.2022).
- [98] I. Kandel and M. Castelli, “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset,” *ICT express*, vol. 6, no. 4, pp. 312–315, 2020.
- [99] J. Brownlee, *A gentle introduction to batch normalization for deep neural networks*, Dec. 2019. [Online]. Available: <https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/>.
- [100] U. Vijay, *Early stopping to avoid overfitting in neural network—keras*, Sep. 2019. [Online]. Available: <https://medium.com/zero-equals-false/early-stopping-to-avoid-overfitting-in-neural-network-keras-b68c96ed05d9>, (accessed: 15.06.2022).
- [101] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [102] R. Moradi, R. Berangi, and B. Minaei, “A survey of regularization strategies for deep models,” *Artificial Intelligence Review*, vol. 53, pp. 3947–3986, 2020.
- [103] X. Ying, “An overview of overfitting and its solutions,” in *Journal of physics: Conference series*, IOP Publishing, vol. 1168, 2019, p. 022 022.
-

-
- [104] C. F. G. do Santos, D. Colombo, M. Roder, and J. P. Papa, “Maxdropout: Deep neural network regularization based on maximum output values,” in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, 2021, pp. 2671–2676.
- [105] I. Salehin and D.-K. Kang, “A review on dropout regularization approaches for deep neural networks within the scholarly domain,” *Electronics*, vol. 12, no. 14, p. 3106, 2023.
- [106] B. Akinremi, *Best tools for model tuning and hyperparameter optimization*, Nov. 2023. [Online]. Available: <https://neptune.ai/blog/best-tools-for-model-tuning-and-hyperparameter-optimization>, (accessed: 17.11.2023).
- [107] R. Shah, *Tune hyperparameters with gridsearchcv*, Aug. 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/06/tune-hyperparameters-with-gridsearchcv/>, (accessed: 17.11.2023).
- [108] W. Koehrsen, “A conceptual explanation of bayesian hyperparameter optimization for machine learning,” *Towards Data Science*, vol. 5, Jul. 2018.
- [109] SOULPAGE, *What is Hyperparameter Tuning*, May 2023. [Online]. Available: <https://dagshub.com/glossary/hyperparameter-tuning/>, (accessed: 17.11.2023).
- [110] Y. Lim, “State-of-the-Art Machine Learning Hyperparameter Optimization with Optuna,” *Towards Data Science*, Mar. 2022. [Online]. Available: <https://towardsdatascience.com/state-of-the-art-machine-learning-hyperparameter-optimization-with-optuna-a315d8564de1>.
- [111] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.
- [112] O. Team, *OPTUNA: an Automatic Hyperparameter Optimization Framework*, Apr. 2022. [Online]. Available: <https://odsc.com/blog/optuna-an-automatic-hyperparameter-optimization-framework/>, (accessed: 17.11.2023).
-

-
- [113] L. Prechelt, “Automatic early stopping using cross validation: Quantifying the criteria,” *Neural networks*, vol. 11, no. 4, pp. 761–767, 1998.
- [114] R. B. Garg, E. Hofgard, L. Tompkins, and H. Gray, “Exploration of different parameter optimization algorithms within the context of acts software framework,” *arXiv preprint arXiv:2211.00764*, 2022.
- [115] X. Bouthillier, C. Tsirigotis, F. Corneau-Tremblay, *et al.*, *Epistimio/orion: Asynchronous Distributed Hyperparameter Optimization*, version v0.2.6, 2022. DOI: 10.5281/zenodo.3478592. [Online]. Available: <https://doi.org/10.5281/zenodo.3478592>.
- [116] R. Vidiyala, “Performance metrics for classification Machine learning problems,” Aug. 2022. [Online]. Available: <https://towardsdatascience.com/performance-metrics-for-classification-machine-learning-problems-97e7e774a007>.
- [117] S. Narkhede, “Understanding Confusion Matrix - towards Data science,” Jun. 2021. [Online]. Available: <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>, (accessed: 22.11.2023).
- [118] *Decimation (signal processing)*, 2022. [Online]. Available: <https://encyclopedia.pub/entry/35884>, (accessed: 17.02.2023).
- [119] The MathWorks Inc., *Interpolation — increase sample rate by integer factor*, Natick, Massachusetts, United States, 2022. [Online]. Available: <https://www.mathworks.com/help/signal/ref/interp.html#References>, (accessed: 17.11.2023).
- [120] Nutan. “PyTorch Convolutional Neural Network With MNIST Dataset.” (2021), [Online]. Available: <https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118>. (accessed: 30.10.2021).
- [121] S. Imoscopi, “Machine learning for text-independent speaker verification,” 2016.
- [122] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016, <http://www.deeplearningbook.org>.
- [123] D. S. Park, W. Chan, Y. Zhang, *et al.*, “SpecAugment: A simple data augmentation method for automatic speech recognition,” *arXiv preprint arXiv:1904.08779*, 2019.
-

-
- [124] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into deep learning*. Cambridge University Press, 2023.
- [125] A. Bissuel, “Hyper-parameter optimization algorithms: A short review,” *Medium*, 2020.

Appendix A

Supplemental content

We provide supplementary information for the optimisation algorithms for Orion and additional results related to the experiments in Chapter 4 and Chapter 5.

A.1 Appendix: Chapter 2

The following Tables A.1 and A.2 show the different algorithms that Orion uses for hyperparameter optimisation.

Table A.1: The different hyperparameter sweep algorithms used by Orion.

Algorithm	Description
Random search	Random search is the most simple algorithm. It samples from given priors [115].
Grid search	Grid search is also one of the simplest algorithms. It works well for small search spaces of one or two dimensions, but is not considered for larger ones. It suffers when the number of hyperparameters grows exponentially [115].
Hyperband	Hyperband is a type of random search that incorporates some exploratory-exploit theory to determine the optimal time allocation for each configuration [125].
Asynchronous Successive Halving Algorithm (ASHA)	The asynchronous version of Hyperband may be thought of as a sophisticated random search that takes advantage of partial trial execution information to focus resources on the most promising trials [115].
Bayesian Optimisation Hyperband (BOHB)	Is a Bayesian Optimisation procedure that is integrated into the first rung of Hyperband brackets to choose hyperparameters to attempt. The first batch of trials will be chosen at random, while the second and subsequent batches will be chosen using Bayesian optimisation [115].
Differential Evolutionary Hyperband (DEHB)	Is an integration of a Differential Evolutionary algorithm with Hyperband. While BOHB uses Bayesian optimisation to select the hyperparameter to try at the first rung of subsequent brackets, DEHB uses Differential Evolution for both selecting the hyperparameters to try at the first rung of subsequent brackets and mutating the best sets of hyperparameters when promoting trials inside a bracket. Trials cannot be resumed after promotion to a higher fidelity level with DEHB. DEHB leads to different hyperparameter values and thus different trial ids. For this reason, trials cannot be resumed after promotions as for other variants of hyperbands [115].

Table A.2: The different hyperparameter sweep algorithms used by Orion.

Algorithm	Description
Population-Based Training (PBT)	Population-based training is an evolutionary technique that evolves trials from low fidelity to high fidelity (for example, the number of epochs) while reusing the model’s parameters. The fidelity levels have the effect of producing hyperparameter schedules [115].
Population-Based Bandits (PB2)	Instead of relying merely on random perturbations, Population-Based Bandits is a version of Population-Based Training that uses a probabilistic model to guide the search. During training, the optimisation curves are modelled using a time-varying Gaussian process in the PB2 implementation. The ray-tune implementation is used in this implementation. Orion’s version includes discrete and categorical dimensions and backtracking, which improves robustness to broken trials [115].
TPE	The TPE technique is a Sequential Model-Based Global Optimisation (SMBO) approach that creates models to suggest new spots based on previously observed trials [115].
Adaptive Experimentation (Ax)	Ax is a platform to optimise machine learning experiments, A/B testing, and simulations, among other types of studies. Ax may use multi-armed bandit optimisation to optimise discrete configurations (e.g., versions of an A/B test) and Bayesian optimisation to optimise continuous (e.g., integer or floating point)-valued configurations [115].

A.2 Appendix: Chapter 4

The confusion matrices of the analysis in Section 4.5 are provided here. This is treated as additional information for the experiments performed. Each experiment is performed on the small_3_cs data set across three different seeds using SGD or Adam as the optimiser.

A.2.1 SGD-4

This section provides the confusion matrices for SGD as optimiser on the 4-second chunks across the three different seeds.

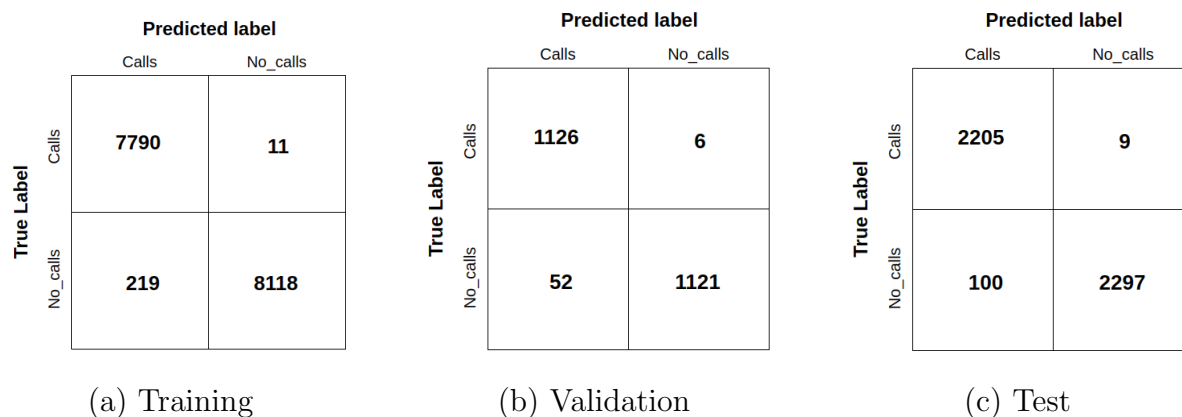


Figure A.1: Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 42 for chunks size of 4.

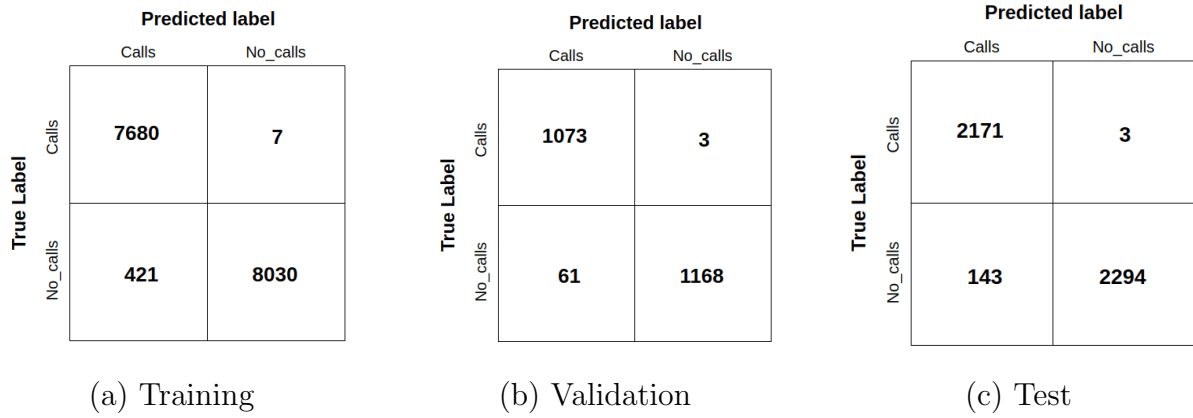


Figure A.2: Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 185 for chunks size of 4.

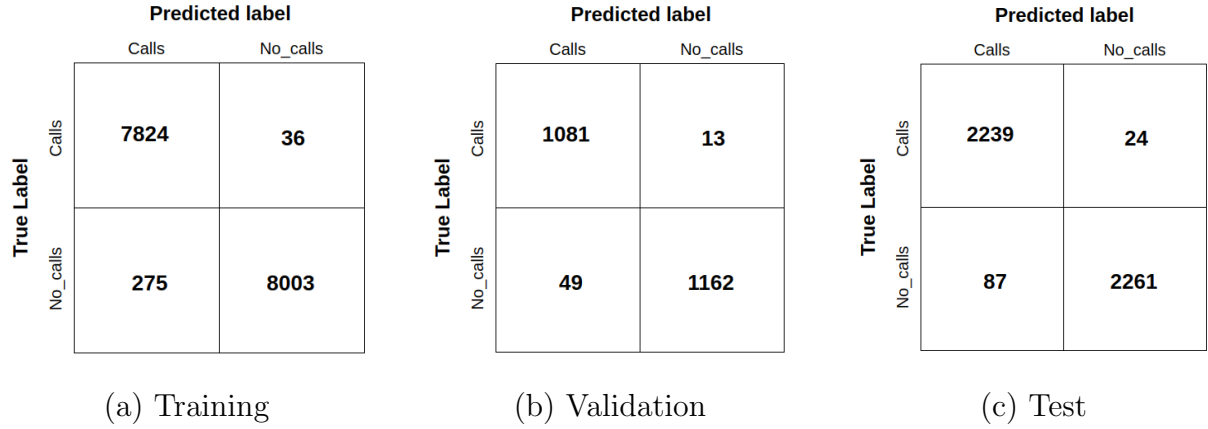


Figure A.3: Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 1986 for chunks size of 4.

A.2.2 Adam-4

This section provides the confusion matrices for Adam as optimiser on the 4-second chunks across the three different seeds.

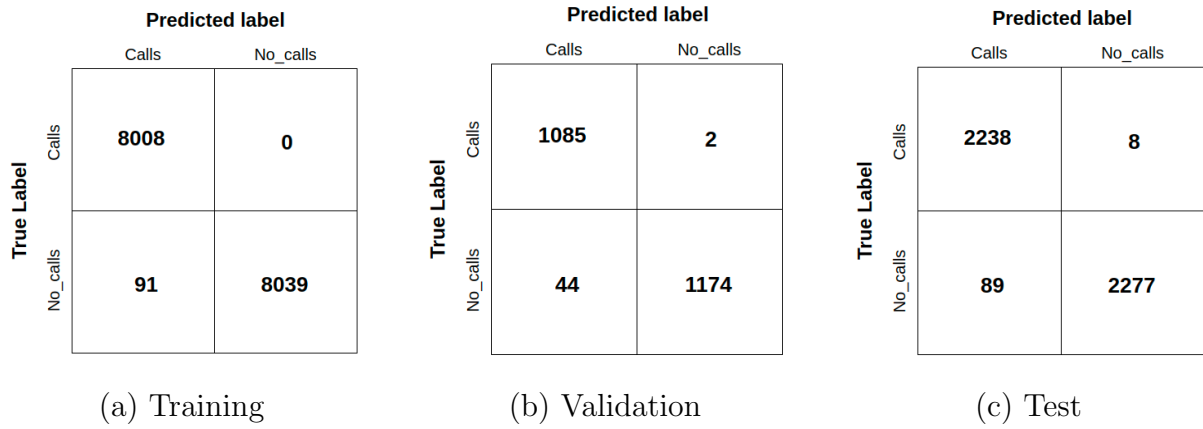


Figure A.4: Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 42 for chunks size of 4.

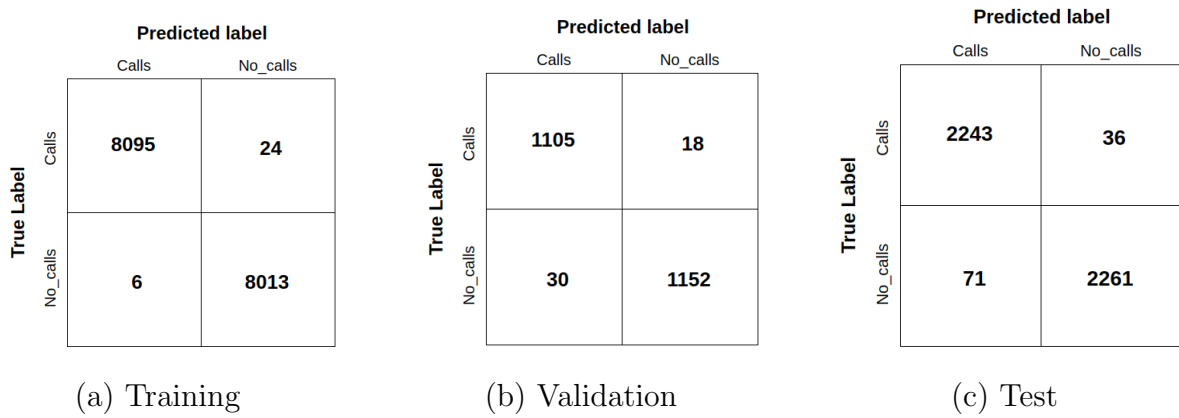


Figure A.5: Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 185 for chunks size of 4.

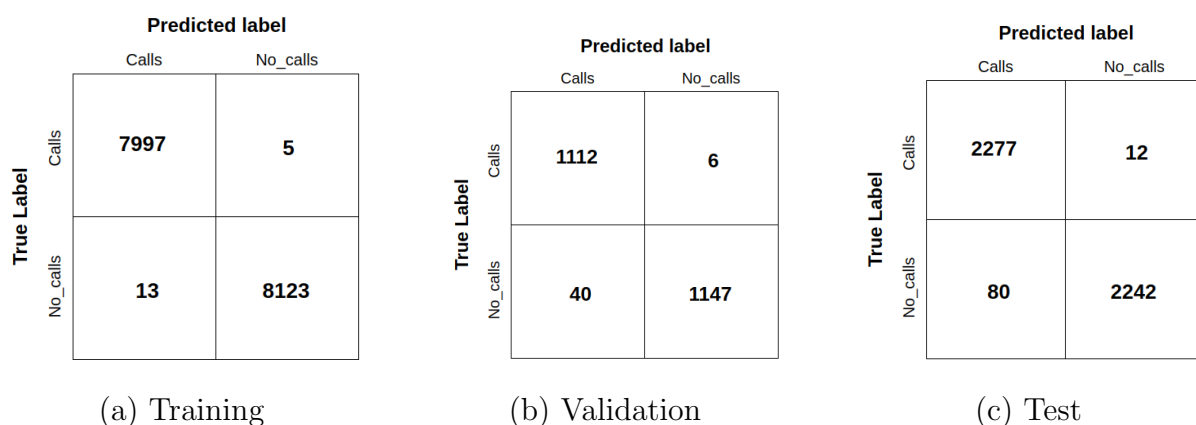


Figure A.6: Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 1986 for chunks size of 4.

A.2.3 SGD-2

This section provides the confusion matrices for SGD as optimiser on the 2-second chunks across the three different seeds.

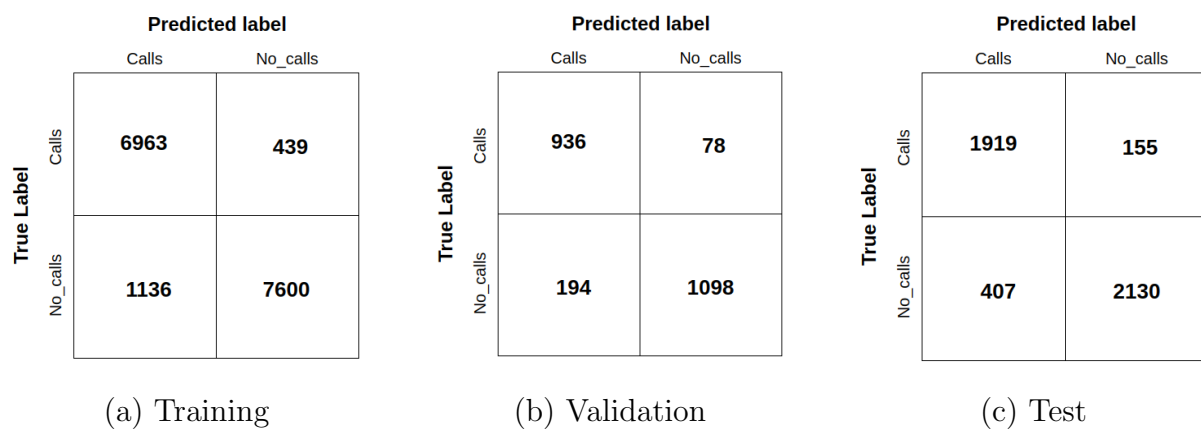


Figure A.7: Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 42 for chunk size of 2.

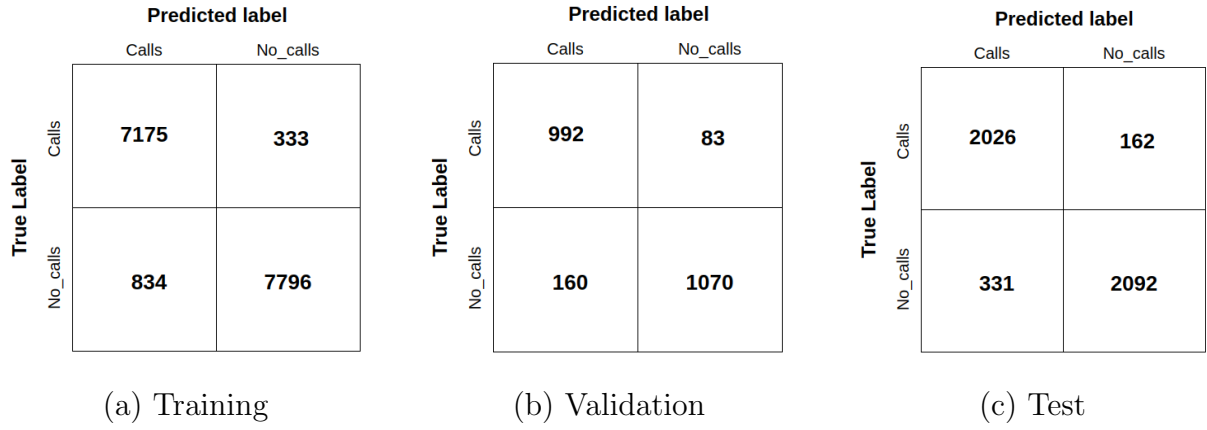


Figure A.8: Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 185 for chunks size of 2.

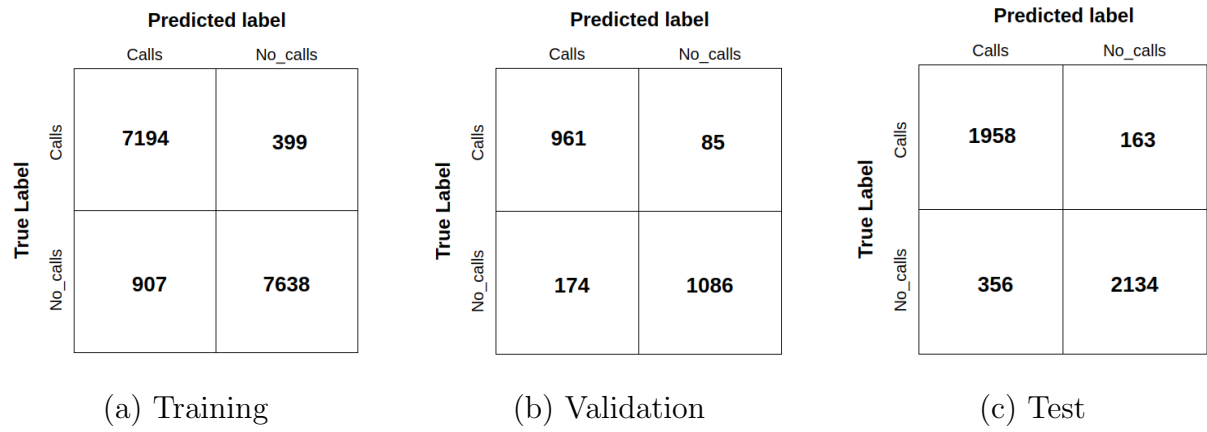


Figure A.9: Confusion matrix for hyperparameter optimisation using Optuna with SGD as optimiser on seed 1986 for chunks size of 2.

A.2.4 Adam-2

This section provides the confusion matrices for Adam as optimiser on the 2-second chunks across the three different seeds.

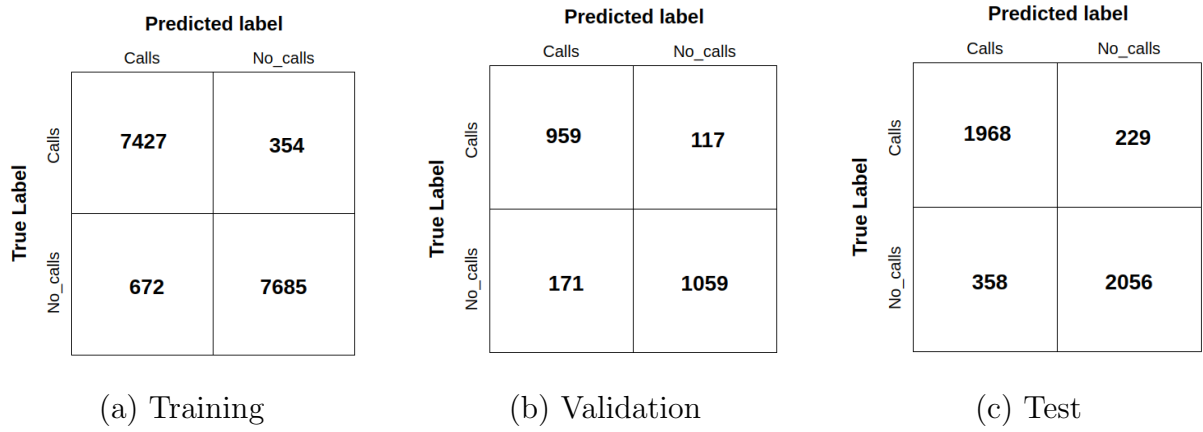


Figure A.10: Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 42 for chunks size of 2.

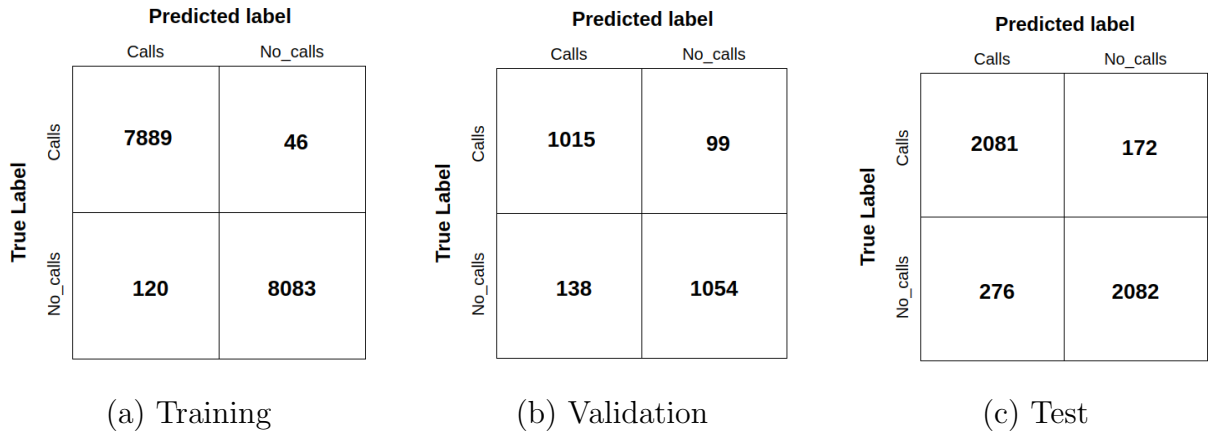


Figure A.11: Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 185 for chunks size of 2.

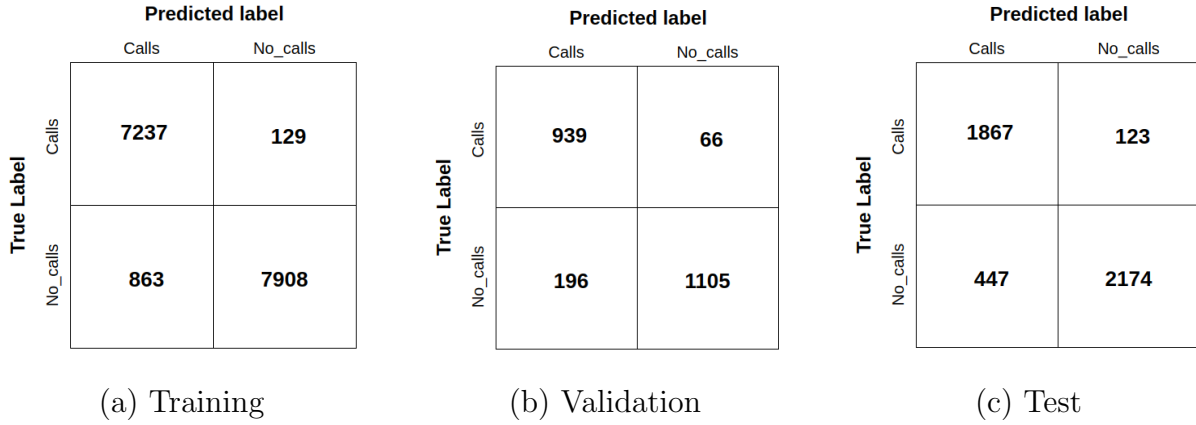


Figure A.12: Confusion matrix for hyperparameter optimisation using Optuna with Adam as optimiser on seed 1986 for chunks size of 2.

A.3 Appendix: Chapter 5

The confusion matrices of the analysis in Section 5.7.1 and Section 5.7.2 are provided here. This is treated as additional information on the experiments performed in Section 5.7. The first Section A.4 contains the confusion matrices for the small_3.cs data set and Section 5.7.2 contains the confusion matrices for the large_3.4 data set.

A.4 Small_3_cs

In this section additional results are provided for the small_3_cs analysis section in Chapter 5. The confusion matrices are represented for each seed for the three different ML tasks.

A.4.1 Fbanks-2

In this section we provide the confusion matrices for the feature extraction method Fbanks with and without deltas using 2-second chunks.

Blue-fin-all-else

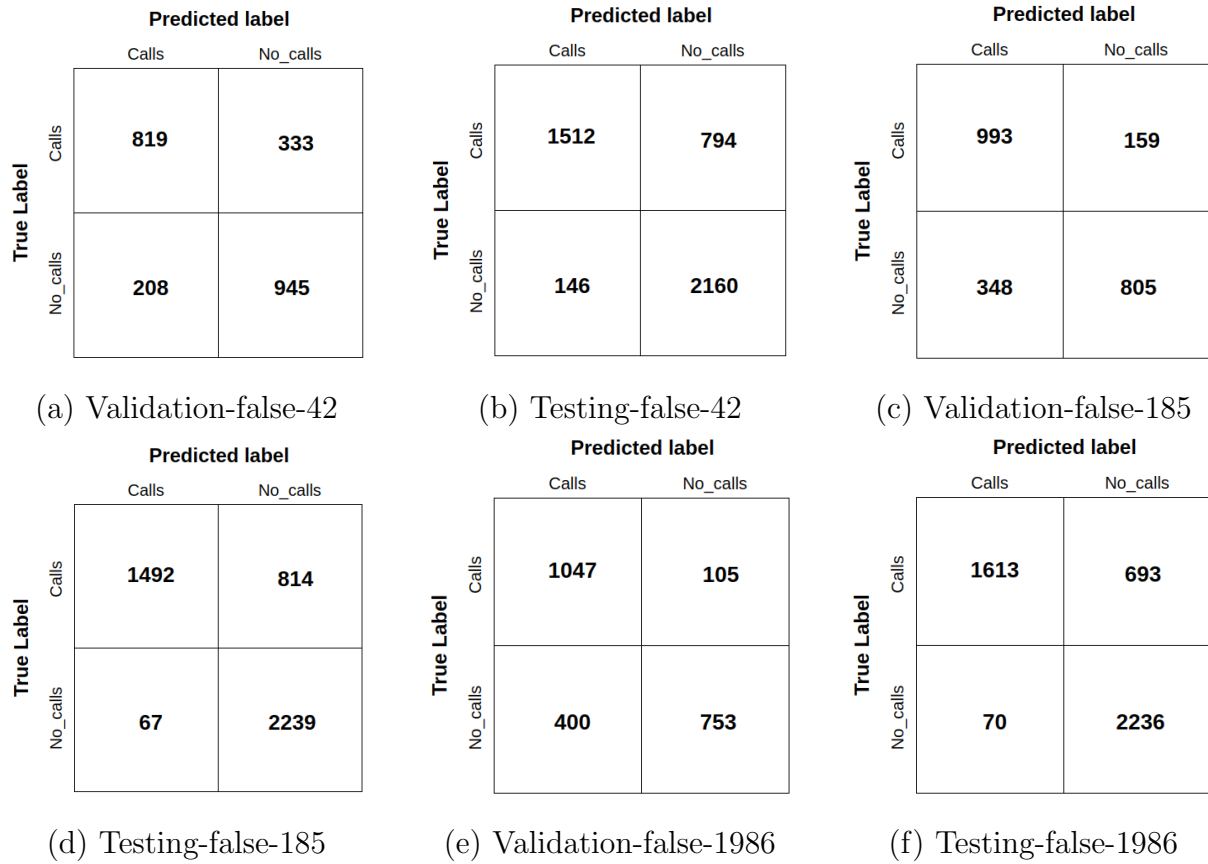


Figure A.13: The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.

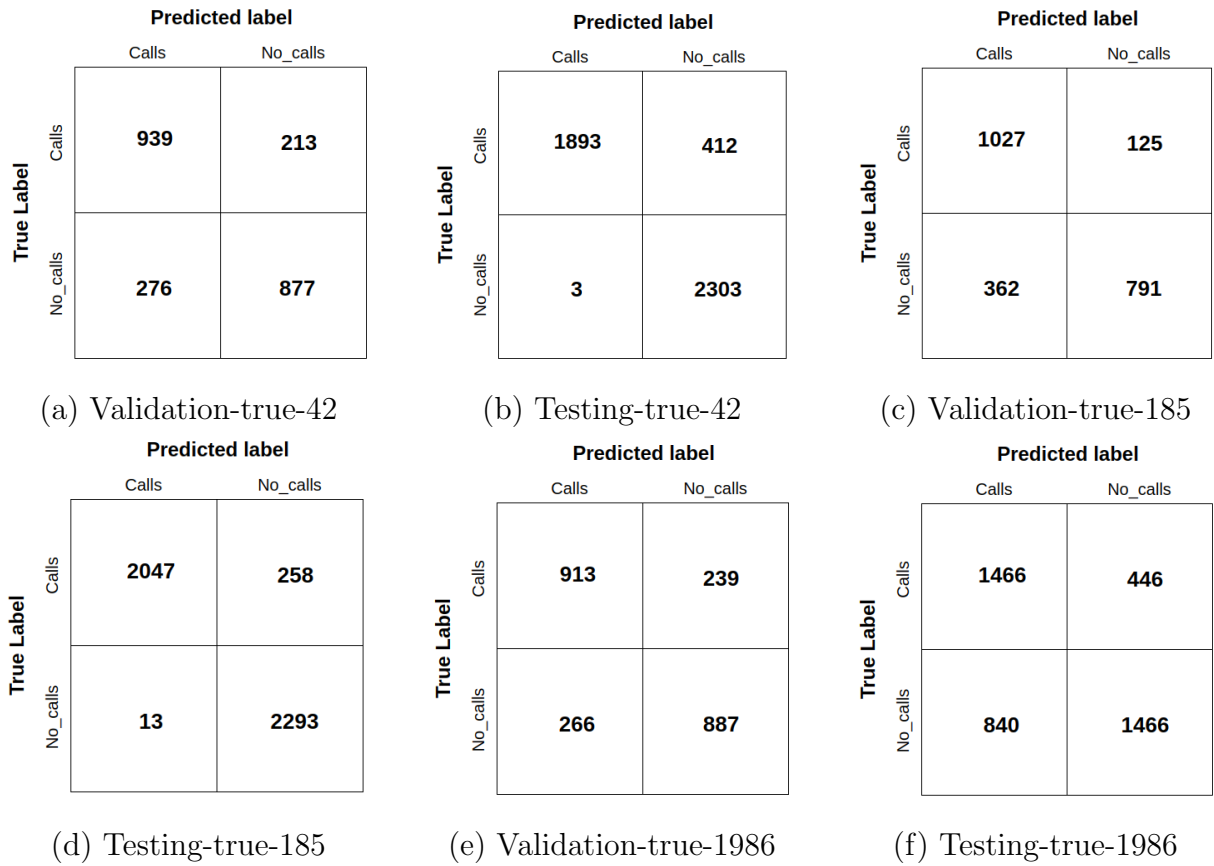


Figure A.14: The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.

Blue-fin-background

		Predicted label	
		Calls	No_calls
True Label	Calls	1091	61
	No_calls	20	1133

(a) Validation-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	2176	129
	No_calls	113	2193

(b) Testing-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1114	38
	No_calls	59	1094

(c) Validation-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	2228	77
	No_calls	332	1974

(d) Testing-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1114	38
	No_calls	30	1123

(e) Validation-false-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	2206	99
	No_calls	153	2153

(f) Testing-false-1986

Figure A.15: The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.

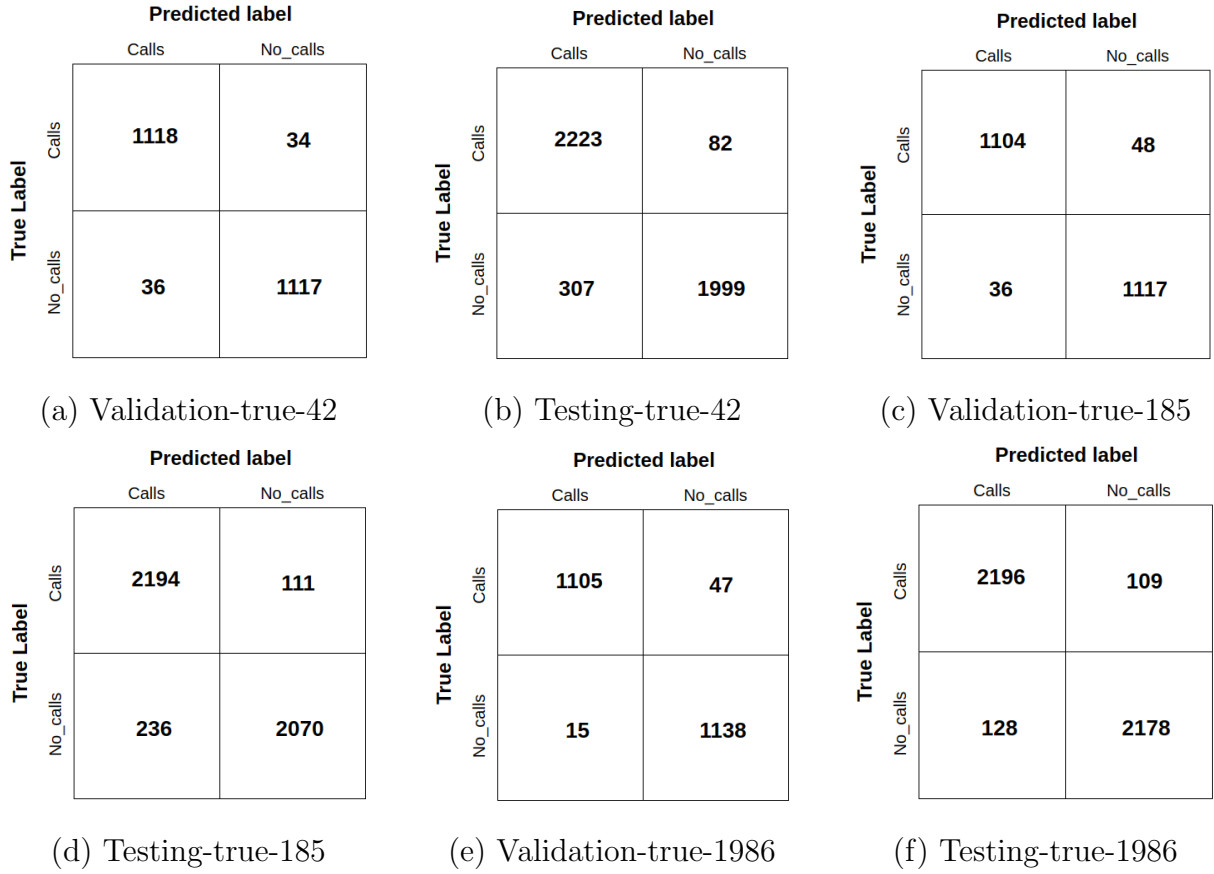


Figure A.16: The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.

All-whale-background

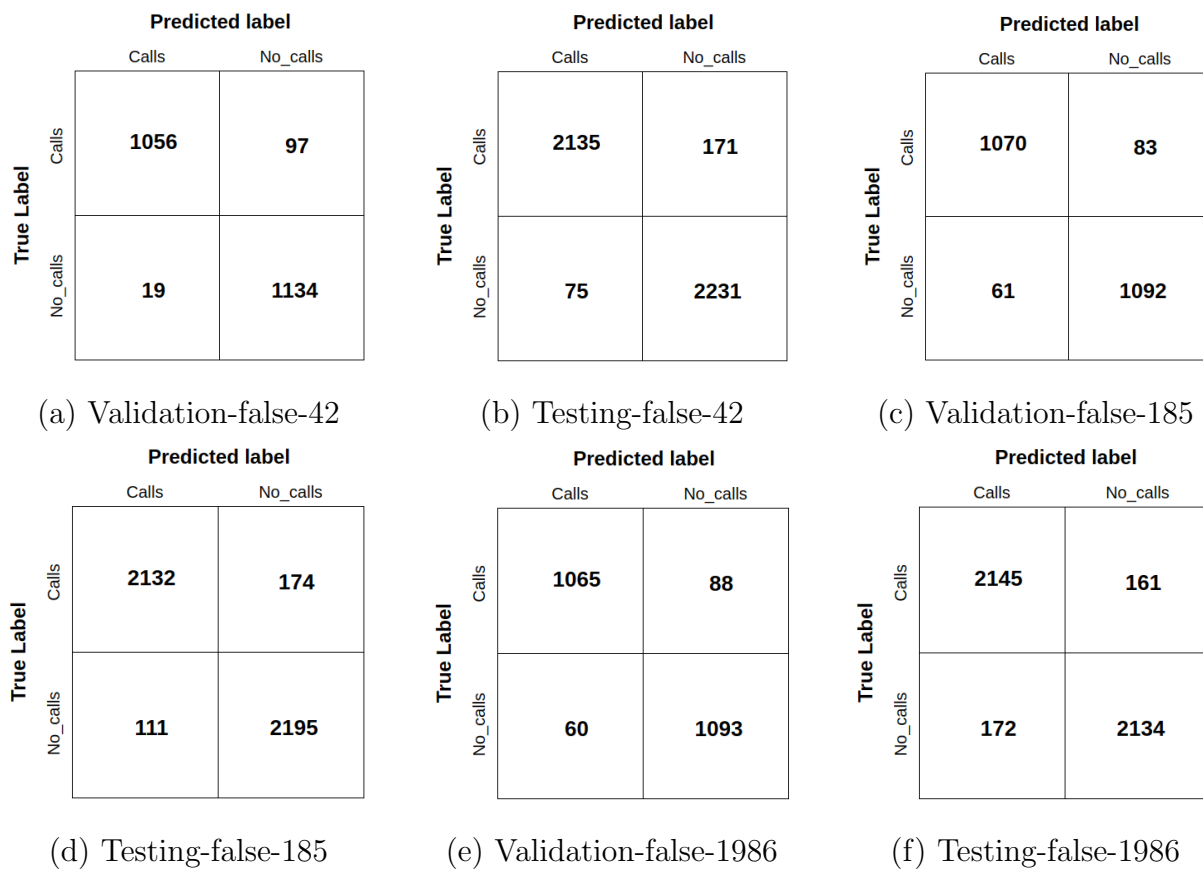


Figure A.17: The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.

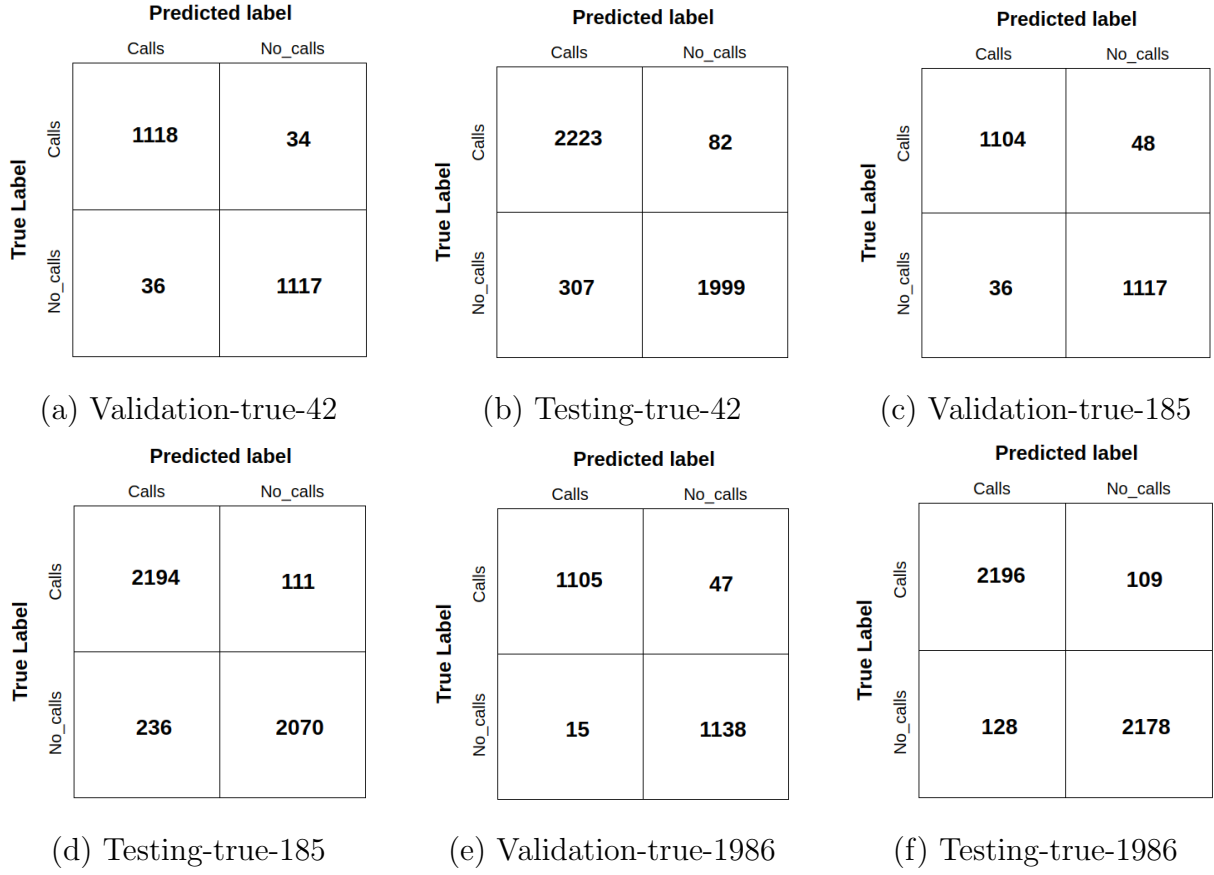


Figure A.18: The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.

A.4.2 MFCCs-2

In this section we provide the confusion matrices for the feature extraction method MFCCs with and without deltas using 2-second chunks.

Blue-fin-all-else

		Predicted label	
		Calls	No_calls
True Label	Calls	1006	146
	No_calls	374	779

(a) Validation-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1576	730
	No_calls	51	2255

(b) Testing-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1008	144
	No_calls	393	760

(c) Validation-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1532	774
	No_calls	126	2180

(d) Testing-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1056	96
	No_calls	435	718

(e) Validation-false-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	2130	175
	No_calls	56	2250

(f) Testing-false-1986

Figure A.19: The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.

		Predicted label	
		Calls	No_calls
True Label	Calls	1056	96
	No_calls	435	718

(a) Validation-true-42

		Predicted label	
		Calls	No_calls
True Label	Calls	2133	172
	No_calls	1358	948

(b) Testing-true-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1047	105
	No_calls	445	708

(c) Validation-true-185

		Predicted label	
		Calls	No_calls
True Label	Calls	2117	188
	No_calls	1461	845

(d) Testing-true-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1042	110
	No_calls	407	746

(e) Validation-true-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	2076	229
	No_calls	1424	882

(f) Testing-true-1986

Figure A.20: The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.

Blue-fin-background

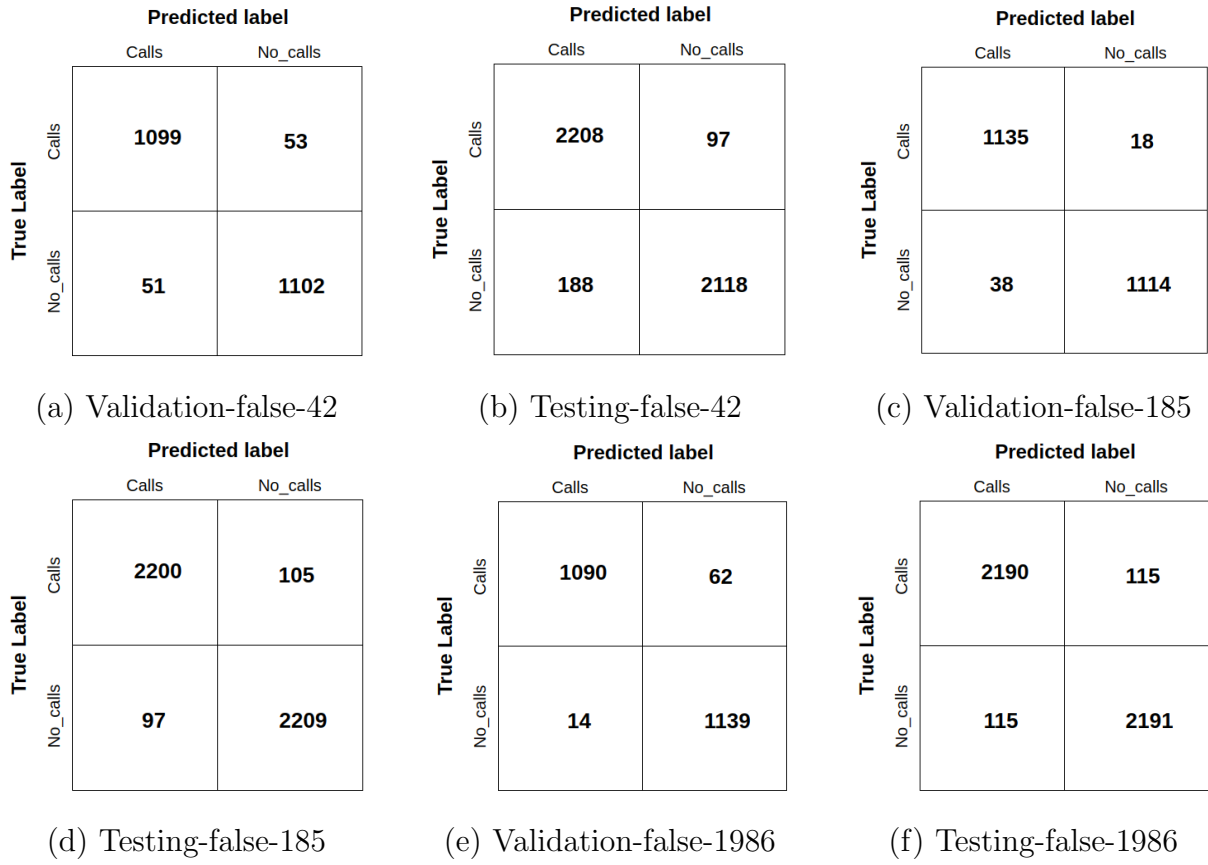


Figure A.21: The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.

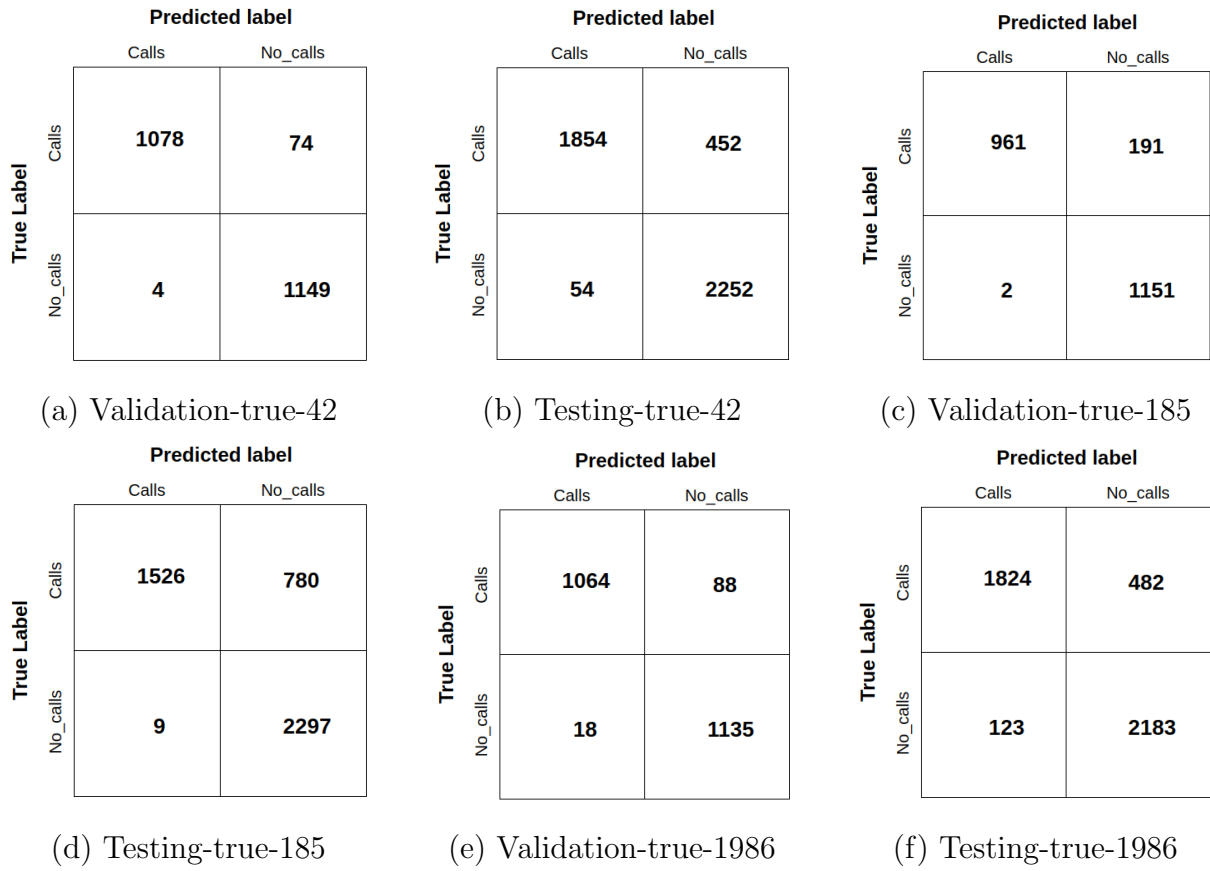


Figure A.22: The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.

All-whale-background

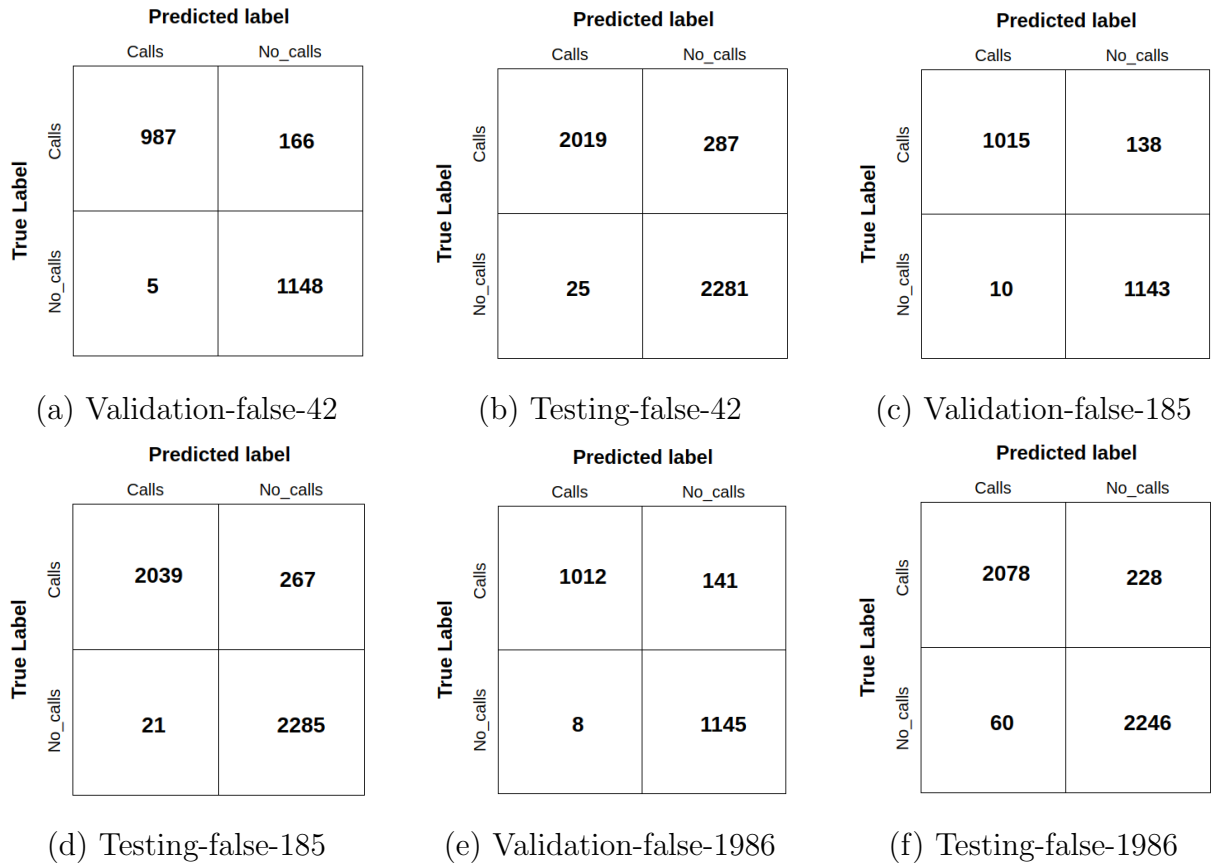


Figure A.23: The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to false.

		Predicted label	
		Calls	No_calls
True Label	Calls	1038	115
	No_calls	32	1121

(a) Validation-true-42

		Predicted label	
		Calls	No_calls
True Label	Calls	2083	223
	No_calls	96	2210

(b) Testing-true-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1061	92
	No_calls	38	1115

(c) Validation-true-185

		Predicted label	
		Calls	No_calls
True Label	Calls	2141	165
	No_calls	145	2161

(d) Testing-true-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1016	137
	No_calls	21	1132

(e) Validation-true-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	2045	261
	No_calls	83	2223

(f) Testing-true-1986

Figure A.24: The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.

A.4.3 Fbanks-4

In this section we provide the confusion matrices for the feature extraction method Fbanks with and without deltas using 4-second chunks.

Blue-fin-all-else

		Predicted label	
		Calls	No_calls
True Label	Calls	850	303
	No_calls	237	917

(a) Validation-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1715	591
	No_calls	630	1676

(b) Testing-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	920	233
	No_calls	355	799

(c) Validation-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1901	405
	No_calls	791	1515

(d) Testing-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	908	245
	No_calls	293	861

(e) Validation-false-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	1793	513
	No_calls	784	1522

(f) Testing-false-1986

Figure A.25: The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.

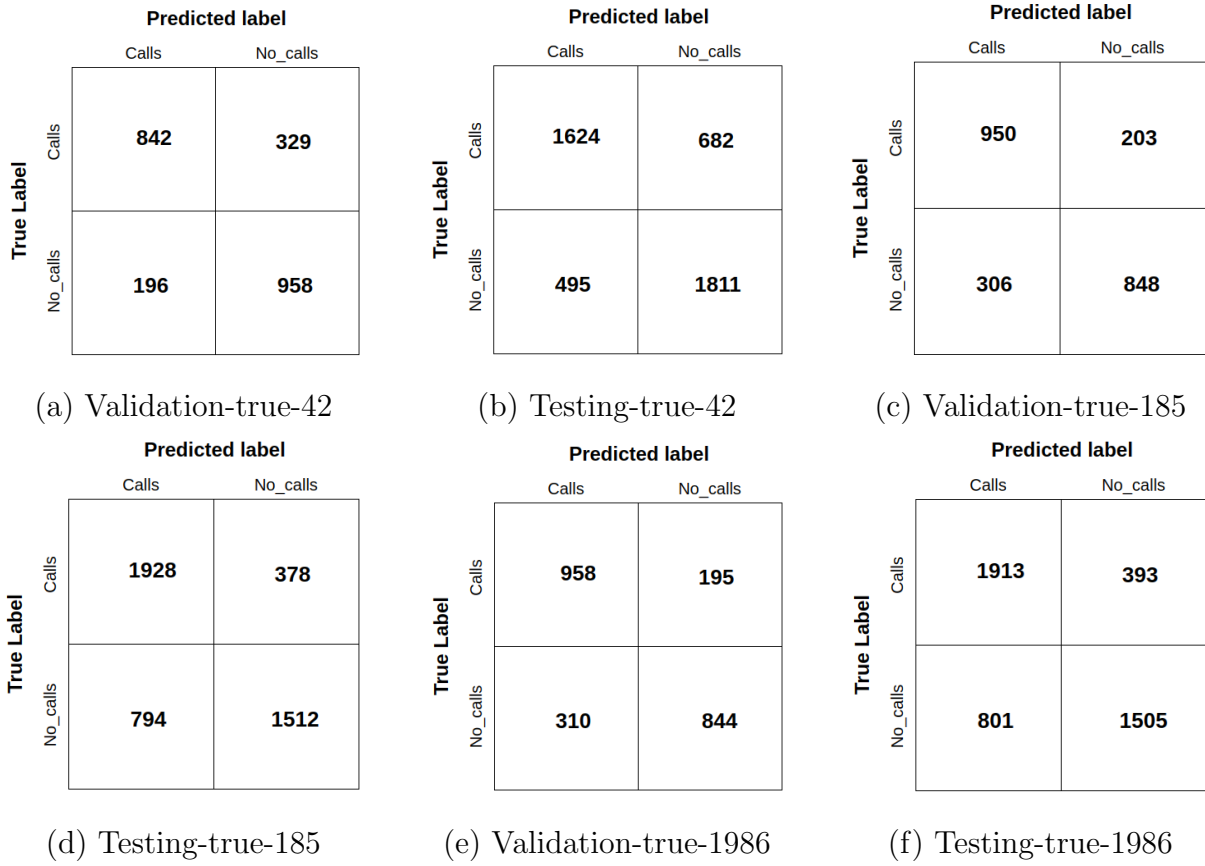


Figure A.26: The confusion matrices for ML task blue-fin-all-else with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.

Blue-fin-background

		Predicted label	
		Calls	No_calls
True Label	Calls	1105	48
	No_calls	5	1148

(a) Validation-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	2179	126
	No_calls	5	2300

(b) Testing-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1098	55
	No_calls	2	1151

(c) Validation-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	2161	144
	No_calls	5	2300

(d) Testing-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1105	48
	No_calls	4	1149

(e) Validation-false-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	2182	123
	No_calls	11	2294

(f) Testing-false-1986

Figure A.27: The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.

		Predicted label	
		Calls	No_calls
True Label	Calls	1098	55
	No_calls	2	1151

(a) Validation-true-42

		Predicted label	
		Calls	No_calls
True Label	Calls	2158	147
	No_calls	6	2299

(b) Testing-true-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1100	53
	No_calls	2	1151

(c) Validation-true-185

		Predicted label	
		Calls	No_calls
True Label	Calls	2169	136
	No_calls	10	2295

(d) Testing-true-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1116	37
	No_calls	9	1144

(e) Validation-true-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	2211	94
	No_calls	26	2279

(f) Testing-true-1986

Figure A.28: The confusion matrices for ML task blue-fin-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.

All-whale-background

		Predicted label	
		Calls	No_calls
True Label	Calls	1097	56
	No_calls	3	1150

(a) Validation-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	2181	124
	No_calls	19	2286

(b) Testing-false-42

		Predicted label	
		Calls	No_calls
True Label	Calls	1099	54
	No_calls	5	1148

(c) Validation-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	2193	112
	No_calls	11	2294

(d) Testing-false-185

		Predicted label	
		Calls	No_calls
True Label	Calls	1098	55
	No_calls	6	1147

(e) Validation-false-1986

		Predicted label	
		Calls	No_calls
True Label	Calls	2171	134
	No_calls	21	2284

(f) Testing-false-1986

Figure A.29: The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.

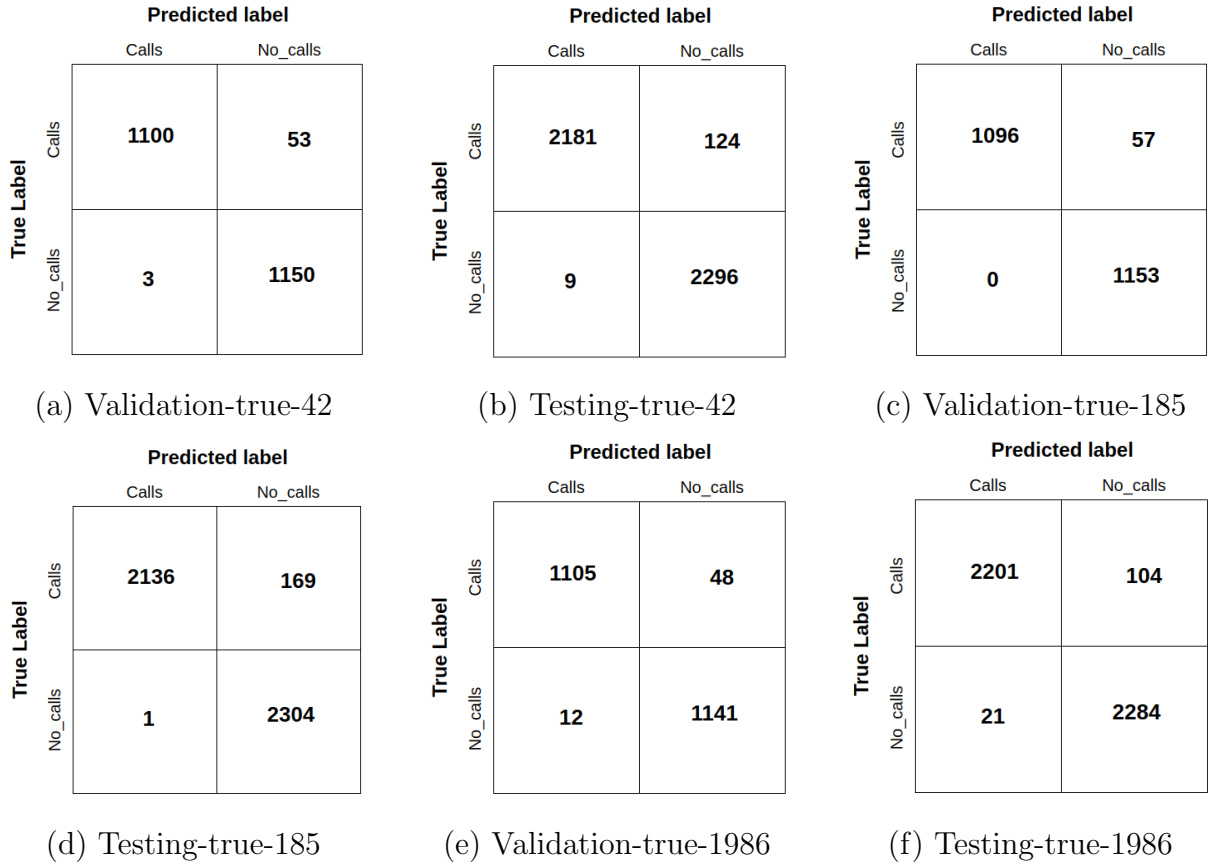


Figure A.30: The confusion matrices for ML task all-whale-background with Fbanks as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.

A.4.4 MFCCs-4

In this section we provide the confusion matrices for the feature extraction method MFCCs with and without deltas using 4-second chunks.

Blue-fin-all-else

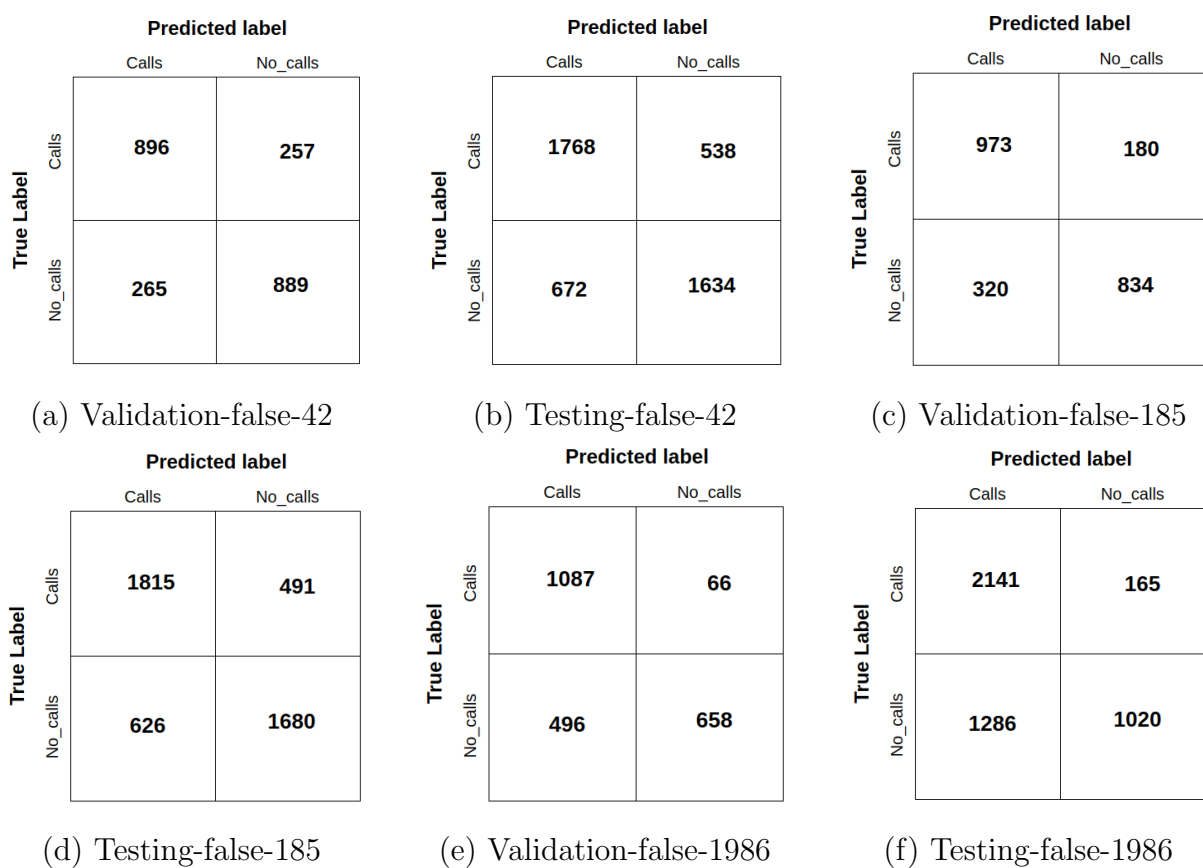


Figure A.31: The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.

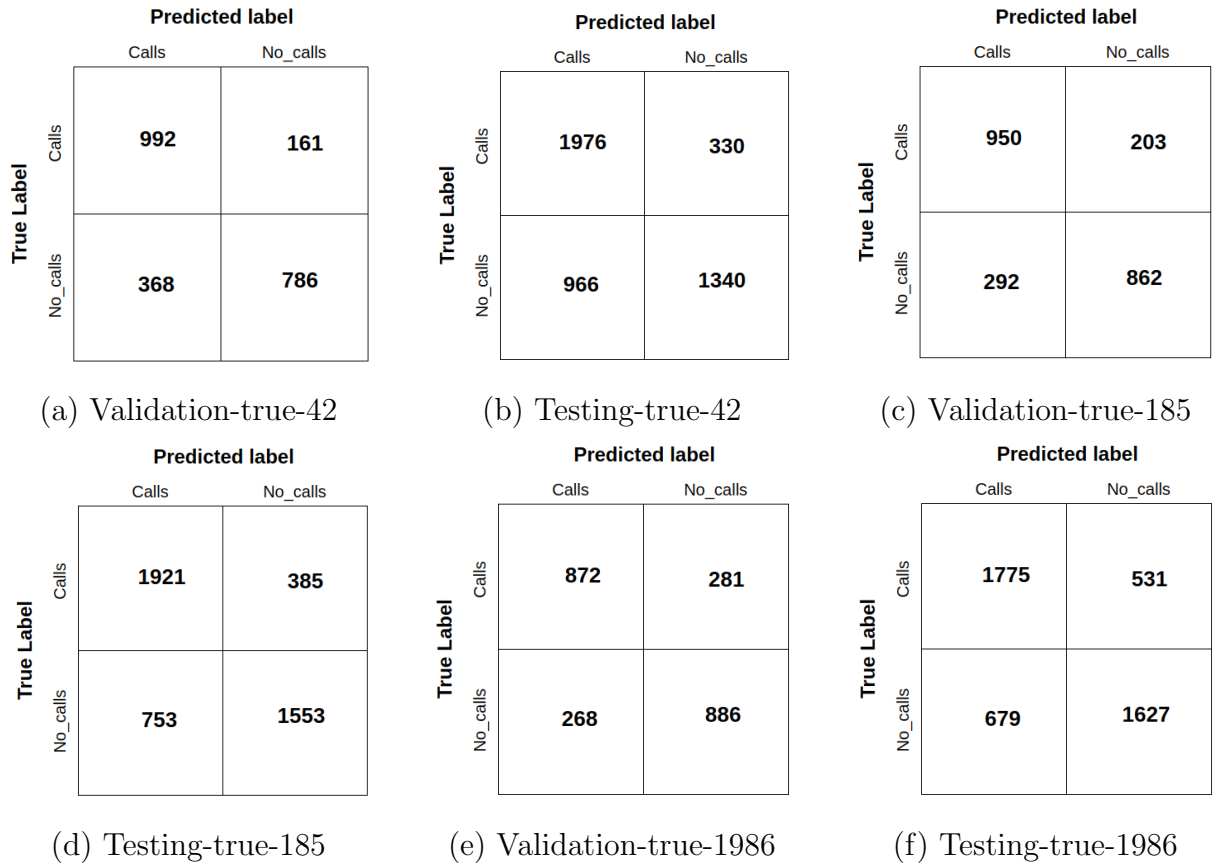


Figure A.32: The confusion matrices for ML task blue-fin-all-else with MFCCs as feature extraction method and using Orion as the optimisation protocol for 2-second chunks across three different seeds. Deltas are set to true.

Blue-fin-background

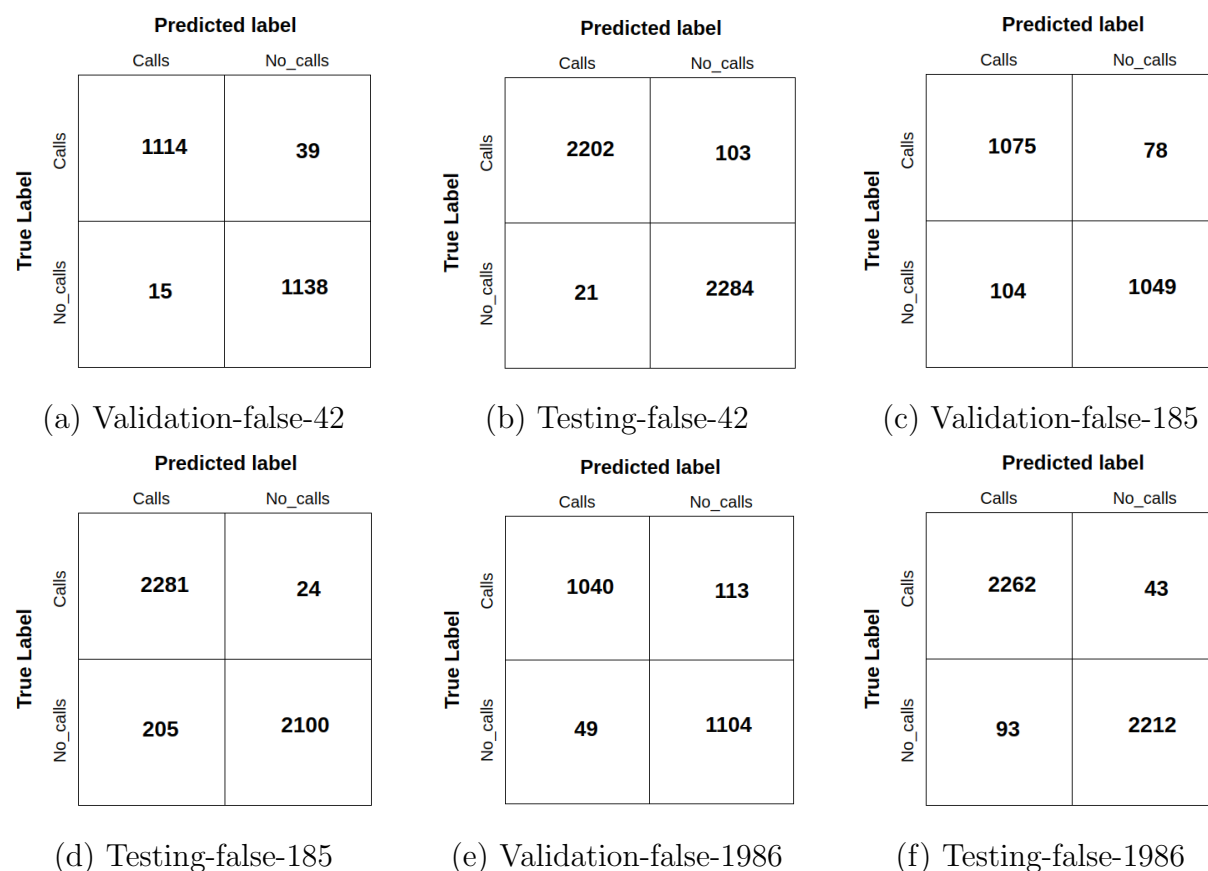


Figure A.33: The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.

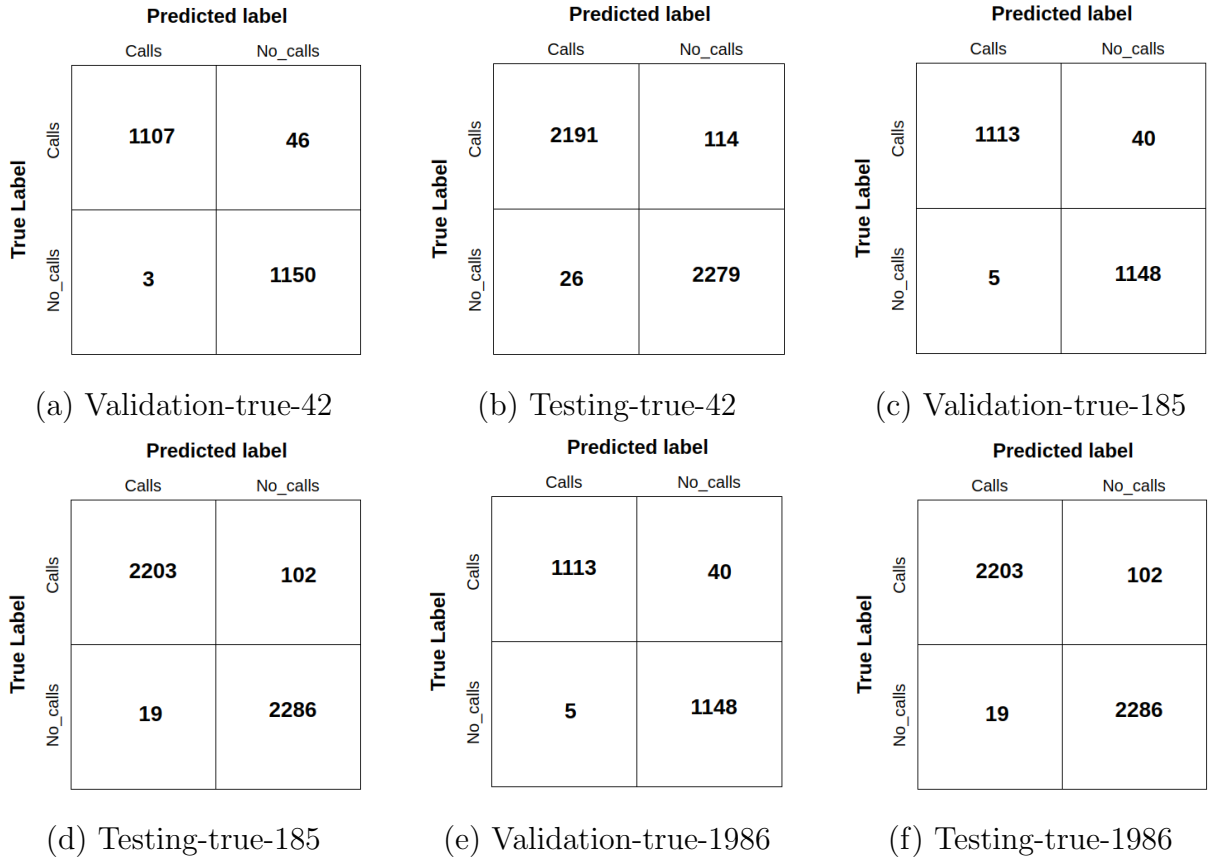


Figure A.34: The confusion matrices for ML task blue-fin-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.

All-whale-background

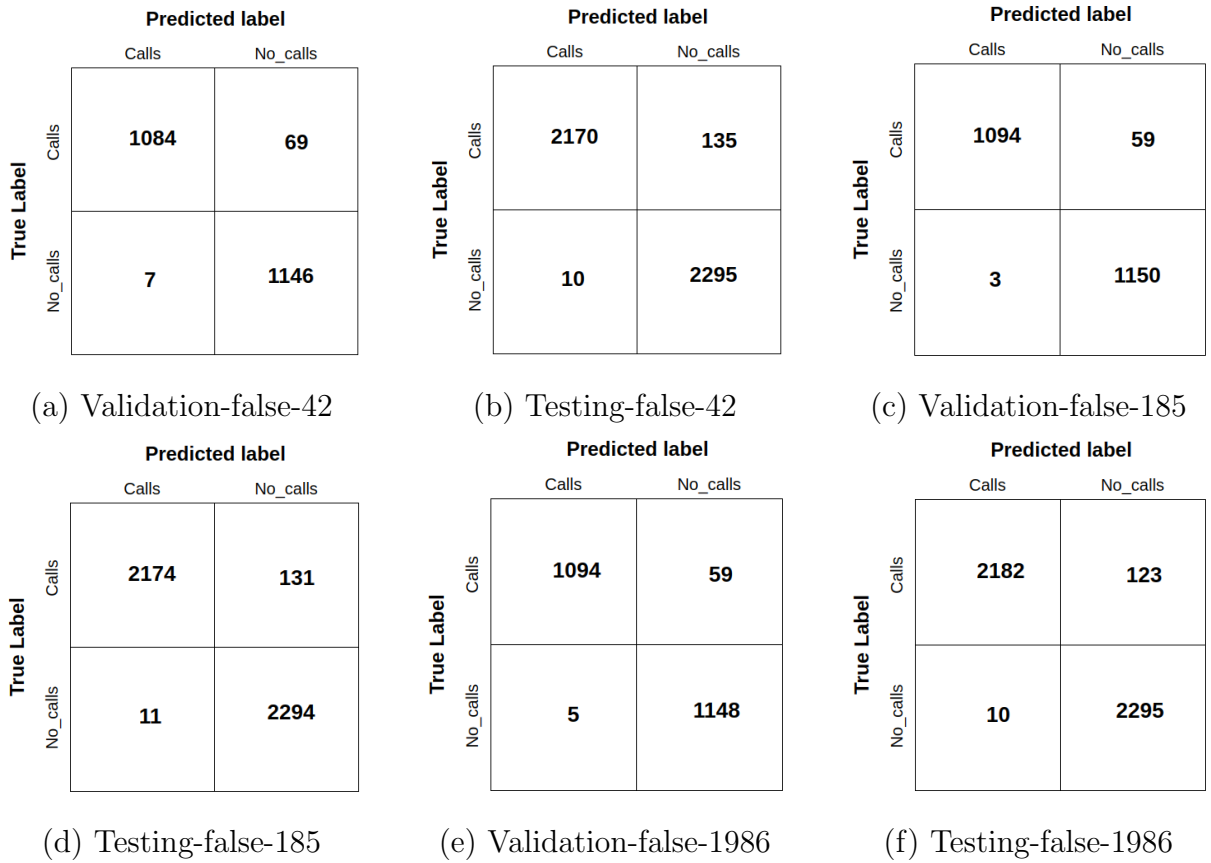


Figure A.35: The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to false.

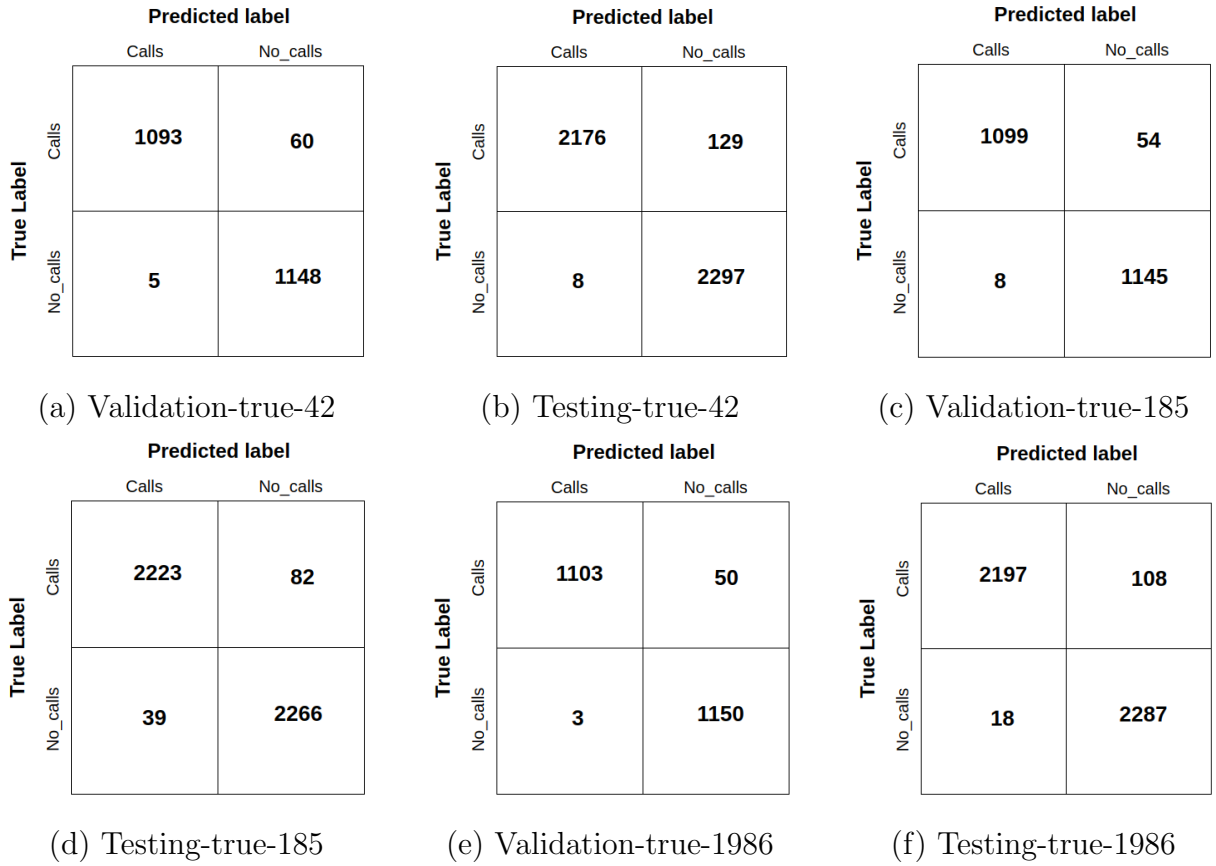


Figure A.36: The confusion matrices for ML task all-whale-background with MFCCs as feature extraction method and using Orion as the optimisation protocol for 4-second chunks across three different seeds. Deltas are set to true.

A.5 Appendix: Publication

Neural Speech Processing for Whale Call Detection

Edrich Fourie^{1,3}[0000–0002–0150–1531], Marelle H. Davel^{1,3,4}[0000–0003–3103–5858],
and Jaco Versfeld²[0000–0001–8327–2244]

¹ Faculty of Engineering, North-West University, South Africa

² Department of Electrical and Electronic Engineering, Stellenbosch University,
South Africa

Abstract. Passive acoustic monitoring with hydrophones makes it possible to detect the presence of marine animals over large areas. For monitoring to be cost-effective, this process should be fully automated. We explore a new approach to detecting whale calls, using an end-to-end neural architecture and traditional speech features. We compare the results of the new approach with a convolutional neural network (CNN) applied to spectrograms, currently the standard approach to whale call detection. Experiments are conducted using the “Acoustic trends for the blue and fin whale library” from the Australian Antarctic Data Centre (AADC). We experiment with different types of speech features (mel frequency cepstral coefficients and filter banks) and different ways of framing the task. We demonstrate that a time delay neural network is a viable solution for whale call detection, with the additional benefit that spectrogram tuning – required to obtain high-quality spectrograms in challenging acoustic conditions – is no longer necessary. While the initial speech feature-based system (accuracy 96%) did not outperform the CNN (accuracy 98%) when trained on exactly the same dataset, it presents a viable approach to explore further.

Keywords: Convolutional neural network · Time delay neural network · Speech features · Whale call detection · Australian Antarctic Data Centre.

1 Introduction

As human marine activities such as fishing and shipping have increased over the years, marine mammals’ existence is increasingly being threatened [1]. Because animals are vocal and acoustics propagate well in water, passive acoustic monitoring of their movement can provide valuable information on marine mammal distribution [2].

³ Centre for Artificial Intelligence Research (CAIR), South Africa

⁴ National Institute for Theoretical and Computational Sciences (NITheCS)

Passive acoustic methods (PAM) use underwater microphones, also called hydrophones, to listen to and record signals of marine mammals for detection, classification and localisation. PAM performs better than visual observation as a monitoring method: it may be used at night, in bad weather, or under other conditions when visual observation is difficult [3].

Acoustic data collection and storage costs have decreased dramatically over the past decade. Nowadays, terabytes of data are collected in a single project [4]. With an increasing volume of acoustic data, extracting meaningful ecological information becomes more and more expensive [4]. Machine learning makes it possible to identify signals in large datasets at a relatively low cost and more consistently than human analysis [5].

Several successful machine learning approaches exist to whale call detection and classification, as discussed in more detail in Section 2. Currently, the most prominent approach uses convolutional neural networks (CNNs) together with spectrograms and has been applied successfully to several datasets [2], [3], [6], [7]. However, a drawback of these spectrogram-based systems is that obtaining high-quality spectrograms over a wide range of underwater acoustic conditions can be challenging, and the process of training a model using spectrograms as features is computationally costly when datasets get large.

In this work, we explore an alternative approach to the task of whale call detection using a time delay neural network (TDNN)-based x-vector system, with traditional speech features as input. Specifically, we experiment with mel frequency cepstral coefficients (MFCCs) and filter banks (Fbanks).

We have two reasons for choosing this approach: In future work, we aim to utilise transfer learning to extend some of the smaller South African whale call datasets with internationally available data or models, an approach that an x-vector system is particularly good at. In addition, we aim to develop a system that does not require careful spectrogram calibration, especially when combining disparate datasets at a later point.

We, therefore, aim to demonstrate that a neural speech processing system based on a TDNN is a viable alternative technique for whale call detection. The TDNN we use is implemented using the Speechbrain toolkit [8]. We develop initial models using the “Acoustic trends for the blue and fin whale library” from the Australian Antarctic Data Centre (AADC) [9]. Results are compared with a spectrogram-based CNN developed using the same dataset.

The layout of this paper is as follows: Section 2 gives a brief overview of related work in the field. Background on neural speech processing and speech features are discussed in Section 3. In Section 4, we discuss the dataset, both its original version and how it was processed for experimentation. Section 5 discusses the CNN baseline. In Section 6, we analyse whale call detection using Speechbrain. In Section 7, we analyse the results from the different systems. Finally, we discuss key findings in Section 8.

2 Related Work

Various studies use either speech features or spectrograms for whale call detection. While the most prominent model used is a convolutional neural network, other types of models are also used, as listed in Table 1. We are unaware of any studies using a TDNN and x-vectors for whale call detection.

Table 1: Significant whale call detection and classification studies.

Study	Dataset	Feature type	Model	Accuracy
Acoustic Detection of Humpback Whales Using a Convolutional Neural Network [1]	National Oceanic and Atmospheric Administration (NOAA)	Spectrograms	CNN	On the test set: 90% precision and 90% recall when a clip contains a humpback call.
Blue whale calls classification using short-time Fourier and wavelet packet transforms and artificial neural network [10]	Saguenay-St. Lawrence Marine Park	Short-time Fourier Transform (STFT) and wavelet packet transform (WPT)	MLP	Classification performance: STFT/MLP obtained 86.25% and WPT/MLP obtained 84.22% on different blue whale calls
Whistle detection and classification for whales based on convolutional neural networks [2]	15 sound samples with a total duration of 120 minutes	Spectrograms	LeNet-5	97% correct detection rate for whistle calls and 95% correct classification rate on the testing set for distinguishing between killer whales and long-finned whales
Detection and classification of marine mammal sounds using AlexNet with transfer learning [11]	Watkins Marine Mammal Sound Database	Spectrograms	AlexNet	Detection accuracy of 99.96% for whale calls and Classification accuracy of 97.42% for distinguishing between killer whales and long-finned whales
Deep Machine Learning Techniques for the Detection and Classification of Sperm Whale Bioacoustics [7]	The Dominica coda dataset	Spectrograms	Detection model: CNN Classification model: LSTM and GRU RNN	Detection accuracy of 99.5% for sperm whale clicks
ORCA-SPOT: An Automatic Killer Whale Sound Detection Toolkit Using Deep Learning [6]	The Archive and OrcaLab	Spectrograms	ResNet	Positive-predictive-value of 93.2% is achieved for detecting killer whale calls
Deep neural networks for automated detection of marine mammal species [4]	DCLDE 2013 workshop data, NOAA, Cornell University's Bioacoustics Research Program (BRP) and Kaggle data competition	Spectrograms	LeNet, ResNet, BirdNet and VGG	LeNet had the highest precision for a certain recall.
Classification of whale calls based on transfer learning and convolutional neural networks [12]	Whale FM	Spectrograms	CNN and transfer learning	Classification accuracy of 97.04% for two whale species and 91.47% classification accuracy for four whale sub-populations

Detection of baleen whale species using kernel dynamic mode decomposition-based feature extraction with a hidden Markov model [13]	Hydrophone recordings of SRW species found in the coastal water of False Bay, Western Cape, South Africa. Mobysound.	Modified dynamic mode decomposition (mDMD). Kernel dynamic mode decomposition (kDMD).	Hidden Markov model (HMM).	The kDMD-HMM generally performed better than the mDMD-HMM and DMD-HMM detectors.
--	--	---	----------------------------	--

3 Approach

We briefly describe neural approaches to different speech processing tasks, as well as the features we use in this work.

3.1 Neural Speech Processing

Several neural approaches produce state-of-the-art results for speech processing tasks such as speaker recognition and speech recognition. Speaker recognition refers to the process of identifying or confirming a person’s identity given a segment of speech, while speech recognition refers to understanding the content of a speech utterance. We list a few.

Vaessen *et al.* [14] applied the wav2vec2 framework to speaker recognition and achieved on the test for the extended VoxCeleb1 a 1.88% Equal Error Rate (EER) compared to the ECAPA-TDNN with a 1.69% EER. (EER is defined as the point on the receiver operating characteristic (ROC) curve where the False Acceptance Rate (FAR) equals the False Rejection Rate (FRR). [15].) A study by Waibel *et al.* [16] used a TDNN for phoneme recognition which achieved a recognition rate of 98.5% compared to the Hidden Markov Models (HMM), which achieved 93.7%. Desplanques *et al.* [17] proposed the ECAPA-TDNN architecture that outperformed state-of-the-art TDNN-based system on the VoxCeleb. In speech, a TDNN is widely used for speech recognition and speaker identification. A TDNN with x-vectors is particularly successful for speaker identification [18], [19].

SpeechBrain ⁵ is an open-source speech processing toolkit that implements various neural processing models. The model that we use is a TDNN with x-vectors. The inputs to the TDNN network are the frames of the extracted input features, either MFCCs or Fbanks.

The outputs are the probability distributions of each of the classes. In speech, we want to look at frames that are a little into the future or the past. These past or future frames are referred to as the context or delays. The process of looking at non-contiguous frames (skipping some of the frames) is referred to as dilation or subsampling. A TDNN consists of individual TDNN units, as illustrated in Figure 1.

⁵ <https://speechbrain.github.io/>

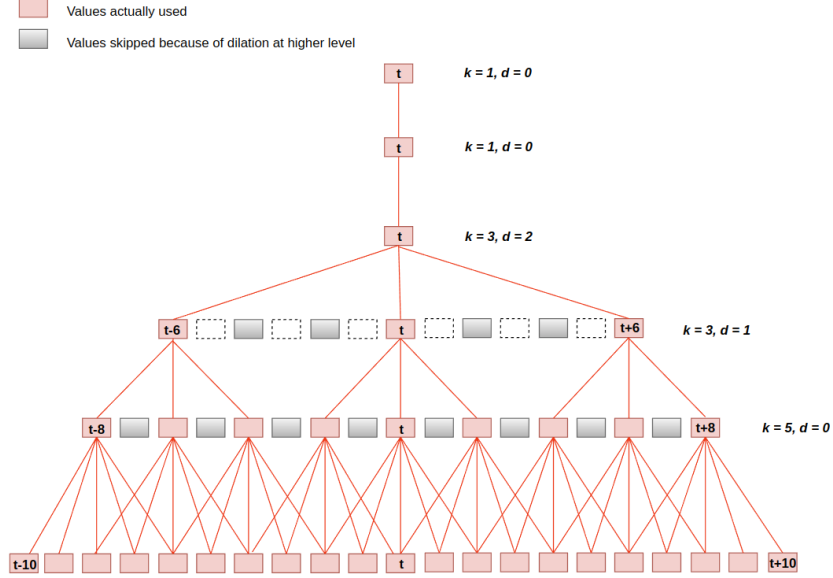


Fig. 1: The TDNN unit used in this study. Each layer has its own kernel size (k) and delay (d).

3.2 Speech Features

Three types of speech features are used in this study: spectrograms, Fbanks and MFCCs. While MFCCs and Fbanks are usually used to mimic human auditory processes, the frequency of the filters is adjusted to fall in the spectrum of whale sounds.

Spectrograms In spectrograms, you can visually see the strength of a signal over time at different frequencies within a particular waveform [20]. Spectrograms are two-dimensional graphs with colours representing the third dimension [20]. The horizontal axis represents time, while the vertical axis represents frequency. The amplitude (or energy/loudness) of a particular frequency at a particular time is represented by the third dimension (colour) [20].

Filter Banks A Fbank is a collection of bandpass filters used in image or signal processing to separate an input signal into a number of analysis signals, each of which carries a specific frequency sub-band from the original signal [21]. A filter bank is used to carry out a range of tasks, such as bandwidth reduction, sample rate modifications, combinations of the standard spectrum translation operations, and signal spectral composition and decomposition [21].

Mel Frequency Cepstral Coefficients (MFCCs) Different signal processing applications have extensively used MFCCs in feature extraction [22]. Image identification, cetacean vocalisation and speech, gestures and drone sound recognition are just a few examples of such applications [22]. An MFCC is used to extract distinctive features from a time-varying signal. To extract features from the signal, it transforms the signal from the time domain into the Mel-scale frequency domain using two types of filters (linear spaced filters and logarithm spaced filters) [22].

4 Dataset

We first describe the original dataset (Section 4.1) before discussing our further processing of the data in Section 4.2.

4.1 The AADC Dataset

The AADC library was created to enable automated algorithms to be applied to a representative sample of recordings curated by the Acoustic Trends Working Group of the International Whaling Commission’s Southern Ocean Research Partnership (IWC-SORP) [9]. In the recorded data, there are four different call types for Antarctic Blue whales and three different call types for Antarctic Fin whales [9]. A detailed outline of calls that are annotated can be seen in Table 2 [9] in order to provide an indication of the diversity of calls. The first two letters correspond to the genus and species, where ‘Bm’ is for Blue whales, and ‘Bp’ is for Fin whales. The ‘Ant’ is for sub-species, in this case, Antarctic Blue whales. The remainder of the classification represents the particular call type.

Table 2: Classification and labelling system for Blue and Fin whale sounds in the SORP Library of Annotated Recordings [9].

Call label	Call Type	Description
BmAnt-A	Antarctic Blue whale unit A	Tone with a constant frequency between 25 and 28 Hz (depending on the year) without other units. Usually repeated every 60-70 seconds.
BmAnt-B	Antarctic Blue whale unit AB	The unit A tone follows an inter-tone down-sweep (unit B) for the Antarctic Blue whale.
BmAnt-Z	Antarctic Blue whale z-call; (AKA 3 unit vocalisation)	An Antarctic Blue whale z-call with upper and lower tonal units A and C present (and down-swept unit B either present or absent).
Bm-D	Blue whale FM (AKA D-calls)	A Blue whale’s down-swept frequency-modulated call. Typically, but not always, longer in duration and lower in frequency than FM calls from Fin and Minke whales.
Bp20Hz	Fin whale 20 Hz pulse	Fin whale pulse at 20 Hz, with no significant energy at higher frequencies.
Bp20Plus	Fin whale 20 Hz pulse with energy at higher frequencies (e.g. 89 or 99 Hz components)	Pulse of a Fin whale at 20 Hz, with secondary energy at higher frequencies (e.g. upper-frequency peak near 80-10 and Hz)
Bp-Downsweep	Fin whale FM calls (AKA ‘high frequency’ down-sweep; AKA 40 Hz pulse)	Fin whales are thought to make frequency-modulated, normally down-swept calls. Blue whale calls are typically shorter in length and have a slightly higher frequency than FM calls from fin whales.

Unidentified	Unidentifiable sounds	It includes any transient biological sound that cannot be confidently classified as one of the above. This included potentially biological sounds but significantly different from the categories above. It also included FM down-sweeps in which the analyst was uncertain whether they were Blue whale D sounds or Fin whale down-sweeps.
--------------	-----------------------	---

The dataset consists of 1,880.25 hours of audio across 11 different sites. The recording years were 2005, 2013, 2014, 2015 and 2017. In total, there were 105,161 annotations across all sites [9]. When analysing the data, it became clear that 3 of the 11 sites were better curated than the others. For 9 of the sites, some of the metadata contained inconsistent labels (the same call labelled with different annotations) and duration errors (calls starting before they end). Therefore, we select only three sites for experimentation: Elephant Island (2013 and 2014) and Balleny Island (2015). In Table 3, the three sites that we use can be seen together with the number of annotations per call type for the Blue and Fin whale calls.

Table 3: Number of annotations at each site by classification type for the Blue and Fin whale [9].

Site-Year	BmAnt-A	BmAnt-B	BmAnt-Z	Bm-D
Balleny Island 2015	923	44	31	47
Elephant Island 2013	2,625	1786	152	299
Elephant Island 2014	6,934	967	100	1,034
Total	10,482	2,797	283	1,380
Site-Year	Bp20Hz	Bp20Plus	Bp-Downsweep	Unidentified
Balleny Island 2015	951	148	78	18
Elephant Island 2013	3,662	1,859	1,042	22,927
Elephant Island 2014	4,940	2,912	3,660	890
Total	9,553	4,919	4,780	23,835

4.2 Data Processing and Event Selection

The audio files were recorded at different sampling rates. The highest recorded Blue and Fin whale call was observed at approximately 135Hz. In line with the Nyquist theorem, we resample all three sites to 300Hz.

We select two datasets for experimentation: a smaller and a larger one. The smaller set is selected for the training of the CNN (and a directly comparable SpeechBrain system) because it is computationally more expensive to run experiments. We selected as much data as we could process using the available infrastructure. The larger set is used for the training of additional Speechbrain systems because the training cycle is computationally less expensive.

The small and large sets are partitioned into a training, validation and test set each, using the ratio 70:10:20. The large set is used for the different machine learning (ML) tasks described in Section 6.1. The size, number of calls and overall duration of the two sets are shown in Table 4. Since more data is available than we can process, we construct a balanced dataset: we first select the available calls and then select background data ('no calls') of the same duration randomly from the available background data. A significant portion of background data is discarded in the process.

Table 4: Size of datasets used for experimentation.

Id	Set	Calls	No Calls	Total	Duration (hours)
Small	Training	8,069	8,069	16,138	17.93
	Validation	1,153	1,152	2,305	2.56
	Testing	2,306	2,305	4,611	5.12
Large	Training	29,782	29,782	59,564	109.26
	Validation	4,314	4,314	8,628	15.61
	Testing	8,470	8,469	16,939	31.22

5 CNN Baseline

This section describes the additional data processing required to develop the CNN baseline. We also describe the architecture used and the hyperparameter optimisation process.

5.1 Additional Data Processing

After resampling, we extracted the spectrograms using Python. The spectrograms are calculated with a Hanning window, a window length of 314, a hop length of 85 and a fast Fourier transform (FFT) with a size of 314. The spectrograms are divided into smaller chunks of 4 seconds in duration. The samples are divided into two classes: calls and no-calls. We use the annotations from the AADC to help us decide if the four-second chunk contains a call or not. If the chunk contains more than 40% of a call, it gets labelled as calls; otherwise, it gets labelled as no-calls. The no-call samples consist of background noise, other whales and biological sounds. Table 4 shows a summary of the dataset.

5.2 Architecture

The CNN architecture we use is a standard CNN, consisting of two hidden CNN layers and two fully-connected layers. The input for the first fully-connected layer varies depending on the architecture, and the output layer has two units that are used to indicate the predicted class for the last fully-connected layer. Each hidden CNN layer consists of a convolutional layer, batch normalisation, dropout, kernel with a size of two and max pooling layer.

The leaky rectified linear unit (LeakyReLU) is used as an activation function since it has been shown to speed up the learning of DNNs and tends to achieve good results in general[23]. The network is initialised with a uniform Kaiming scheme [24], which is the default choice for ReLU-activated networks. The CNN is optimised using either Adam or Stochastic Gradient Descent (SGD) as an optimiser. In addition to SGD, Adam is chosen for its adaptive estimates of lower-order moments [25].

5.3 Optimisation Protocol

The networks are trained on the spectrogram tensors from the Blue and Fin whale sounds for the CNN architecture of Section 5.2. The hyperparameter optimisation is done using the hyperparameter optimiser Optuna⁶, which finds optimal hyperparameter values using different meta-optimisation algorithms. We use a Bayesian optimization algorithm referred to as a Tree-structured Parzen Estimator (TPE) [26]. The learning rate uses a step learning rate that decays the learning rate of each parameter group by gamma every n epochs. The batch size, learning rate, dropout level, the number of in and out channels of the convolution layers and the depth of the fully-connected layers are varied for each trial based on what Optuna selects. The number of epochs was kept at 30 because the training loss always converged before 30 epochs, and Optuna’s pruning was implemented, which stopped an unpromising trial. Optuna is configured to minimise the validation loss. The hyperparameter ranges over which we optimised are shown in Table 5.

Table 5: Hyperparameter ranges searched over during optimisation.

Hyperparameters	Ranges
Batch size	32 and 64
Initial learning rate	0.1 - 0.0001
Dropout	0.1 - 0.5
Out Channel	32 - 128
Fully connected layer depth	100 - 1000
Optimiser	Adam, SGD
Epochs	30

The best accuracy for the CNN network was achieved with Adam as the optimiser and a learning rate of 0.0023. The optimal batch size was 32, with a dropout level of 0.3. The output channel for the convolutional layer was 32, with a 700-node fully-connected layer. Test accuracy of 97.92% is achieved, as discussed further in Section 7.

6 Whale Call Detection Using Speechbrain

In this section, we describe the different ML tasks and the additional data processing required for Speechbrain. We also describe the architecture used and the hyperparameter optimisation process.

6.1 Framing Whale Call Detection as Different Machine Learning Tasks

This section describes how the large dataset in Section 4.2 is used to create the data manifest files for the different ML tasks. In the data manifest, we tell Speechbrain where to find the audio data, the audio length, and the transcription.

⁶ <https://optuna.org/>

- **Blue-fin-only:** In the blue-fin-only task, we consider two classes. The first class is ‘Calls’, which only contains the calls from Blue and Fin whales. The second class is ‘No Calls’, and this class contains only background noise.
- **All-whale-calls:** In the all-whale-calls task, we consider two classes. The first class is ‘Calls’, which contains the calls from any whale. The second class is ‘No Calls’, and this class contains only background noise.

In Table 6, the training, validation and testing sets for the two ML tasks are shown. The reason why we want to compare these different ML tasks is to see to what extent Speechbrain will perform well at identifying either specific whale calls or whale calls in general.

Table 6: Datasets used for the two Speechbrain tasks.

ML Task ID	Training			Validation			Testing		
	Size	Calls	No Call	Size	Calls	No Call	Size	Calls	No Call
blue-fin-only	59,022	29,511	29,511	8,550	4,275	4,275	16,762	8,381	8,381
all-whale-calls	59,564	29,782	29,782	8,628	4,314	4,314	16,939	8,470	8,469

6.2 Additional Data Processing

Only limited additional data processing was required. The resampled wav files were split into files containing calls and files that contained just background noise according to the tasks just mentioned. Calls were divided into segments and appropriately tagged.

6.3 Architecture

In this section, we motivate the decisions made, and principles followed to develop a whale call detection model using Speechbrain.

In the data augmentation and feature extraction below, the sample rate is changed to 300Hz (as previously discussed in Section 4.2). Part of Speechbrain’s standard preprocessing pipeline is to add data augmentation before feature extraction. The data augmentation we use is a time-domain approximation of the SpecAugment algorithm [27]. In this data augmentation, we have three default speed perturbations (95, 100, 105) for Speechbrain to choose from.

We use Fbanks and MFCCs for feature extraction. In the Fbanks, we can use ‘deltas’, which will append derivatives and second derivatives to the features. When extracting MFCCs, the deltas are appended by default. The number of mel filters can also be adjusted in both Fbanks and MFCC; we will evaluate 23, 40 and 80 mel filters as these are the default values Speechbrain suggests. After the features are computed, we can apply mean and variance normalisation; we use the default where only mean normalisation is applied.

The TDNN constructed here has five time-delayed neural layers with the given dilation factors (0, 1, 2, 0, 0) and kernel size (5, 3, 3, 1, 1). We use the default values for dilations and kernel size. The LeakyReLU is used for the same reason as before. After each activation function, batch normalisation is used;

this tends to improve the speaker identification performance [8]. After the convolutional layers, a statistic pooling layer is used to convert a tensor of variable length into a fixed-length tensor [8]. After the statistic pooling layer, a final linear transformation is performed using a linear layer with the number of neurons depending on the size of embedding required. The x-vectors are extracted here. Lastly, a multilayer perceptron (MLP) is implemented on top of the x-vector layer. The MLP consists of one fully-connected layer followed by the final softmax layer for classification. The number of input neurons in the layer depends on the size of the embeddings. The output neurons are the same as the number of classes that needs to be classified.

All the networks are trained with the Adam [28] optimiser. A scheduler with a linear annealing technique is implemented. We optimised the following hyperparameters using the validation set: embedding dimensions, TDNN channels, and the number of mel filter banks; we use the default Speechbrain ranges while we search over broader ranges for the learning rate, batch size, feature lobes and deltas, as shown in Table 7.

6.4 Optimisation Protocol

The algorithm we use is again TPE, which is a sequential model-based optimisation approach [29]. The hyperparameters that we searched over are shown in Table 7. Speechbrain comes with a convenience wrapper called ‘hpopt’, which can report objective values to Orion [30]. This facilitates the optimisation process.

We trained the model until convergence where, if the model’s accuracy in the past 20% of the epochs was still increasing, another 20% of the past number of epochs is added to continue training. Minimum training is set at 10 epochs. If the model’s accuracy did not increase in the past 20% of the epochs, training is stopped. We also implemented early stopping by selecting the model with the best validation accuracy in the training cycle and saving that model. For the hyperparameter sweep, we swept over the hyperparameters and their corresponding values, as seen in Table 7. The model sweeps over the selected hyperparameters with their corresponding values for ten trails, and after that, the model uses the best values from the sweep and does another hyperparameter sweep on only the learning rate (the most sensitive parameter) for final finetuning.

Table 7: Hyperparameters value ranges for the Orion search.

Hyperparameters	Search space
Learning rate	log_uniform(0.001, 0.1, precision=2)
Embedding dimensions	[128, 256, 512, 768, 1024]
TDNN channels	[128, 256, 512, 768, 1024]
Batch size	[8, 16, 32]
Number mel filter banks	[23, 40, 80]
Feature lobes	[Fbank, MFCC]
Deltas	[True, False]

7 Analysis and Results

In this section, we describe the results obtained by both the CNN and Speechbrain in order to determine if Speechbrain is a feasible option for whale call detection.

7.1 CNN

The CNN performed well, resulting in a best validation accuracy of 97.98%. Only the best models obtained with SGD and Adam, respectively, were evaluated on the test set; results are shown in Table 8. We can conclude from Table 8 that the model is not overfitting and has delivered good results. For the best trial in both cases, the hyperparameter values are not at the edges of the range provided.

Table 8: CNN results obtained after hyperparameter optimisation with Optuna, using SGD and Adam as optimiser, respectively.

	SGD	Adam
Training loss	0.0573	0.0092
Training accuracy	98.07%	99.34%
Training recall positives	0.9660	0.9887
Training recall negatives	0.9955	1
Validation loss	0.0107	0.0001
Validation accuracy	97.31%	97.98%
Validation recall positives	0.9600	0.9606
Validation recall negatives	0.9900	0.9982
Testing accuracy	97.59%	97.92%
Testing recall positive	0.9626	0.9622
Testing recall negative	0.9895	0.9965

7.2 Speechbrain’s TDNN

The hyperparameter sweep is performed for 10 trials for each ML task. The hyperparameter values for the model with the best validation accuracy are then used to train the network again. After the network is trained, we test how the network performs on the test set. In both cases, MFCCs with deltas were the best-performing features.

Table 9: TDNN results for the different ML tasks, after hyperparameter optimisation with Orion.

ML task	Set	Accuracy	Precision	Recall	Negative rate
blue-fin-only	Validation	93.36%	0.8827	1.0000	0.8671
	Test	95.89%	0.9382	0.9983	0.8976
all-whale-calls	Validation	97.79%	0.9638	0.9930	0.9627
	Test	97.67%	0.9624	0.9911	0.9614
cnn-compare	Validation	89.97%	0.8403	0.9235	0.8840
	Test	95.72%	0.9953	0.9186	0.9957

The result of the learning task ‘cnn-compare’ is directly comparable to the results of the CNN and the results for Speechbrain are less accurate than the CNN by more than 2% on the test set. This learning task is trained on the exact same data as used for the CNN. While the next results are not directly comparable

(the CNN setup we used was restricted by the size of the dataset it could process) the Speechbrain system ‘blue-fin-only’ produced results that were slightly less accurate than the CNN but of the same order. However, the second machine learning task ‘all-whale-calls’ produced very competitive results. Finally, when evaluating all runs, it was clear that MFCCs with deltas outperformed Fbanks: each of the best systems selected by Optuna used MFCCs.

8 Conclusion

Our purpose with this study was to determine whether neural speech processing and specifically, a TDNN with x-vectors, could be a useful tool for whale call detection. We are specifically interested in this question since x-vectors lend themselves to data sharing across environments. This is important in the South African environment where local whale call databases are scarce. Since calls are split randomly between the training, validation and test sets, it is possible that some similar conditions may be observed within the data of a single site, even though conditions across sites could be very different. This should be taken into account when interpreting the high accuracies reported. In future work, we aim to determine how well these networks transfer to unseen sites, as well as the amount of further tuning that would be required.

In order to answer this question, we develop both a CNN-based and TDNN-based call detection system. The CNN-based system used spectrograms as features, and the TDNN-based system used either Fbanks or MFCCs. When performance is compared on a similar task (detecting Blue or Fin whales in background noise) the CNN performed better (97.92% vs 95.72% test accuracy). However, the results of the TDNN-based system were particularly promising when identifying a variety of whale calls in background noise (97.79% test accuracy). In both TDNN setups, MFCCs with deltas were the most successful features to use. We conclude that neural speech processing is a viable approach to whale call detection and one that we will be exploring further in future work.

References

- [1] M. Harvey. “Acoustic Detection of Humpback Whales Using a Convolutional Neural Network.” (2018), [Online]. Available: <https://ai.googleblog.com/2018/10/acoustic-detection-of-humpback-whales.html>. (accessed: 23.07.2021).
- [2] J.-j. Jiang, L.-r. Bu, F.-j. Duan, *et al.*, “Whistle detection and classification for whales based on convolutional neural networks,” *Applied Acoustics*, vol. 150, pp. 169–178, 2019.
- [3] J.-j. Jiang, L.-r. Bu, X.-q. Wang, *et al.*, “Clicks classification of sperm whale and long-finned pilot whale based on continuous wavelet transform and artificial neural network,” *Applied Acoustics*, vol. 141, pp. 26–34, 2018.

- [4] Y. Shiu, K. Palmer, M. A. Roch, *et al.*, “Deep neural networks for automated detection of marine mammal species,” *Scientific Reports*, vol. 10, no. 1, pp. 1–12, 2020.
- [5] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, “A committee of neural networks for traffic sign classification,” in *The 2011 International Joint Conference on Neural Networks*, IEEE, 2011, pp. 1918–1921.
- [6] C. Bergler, H. Schröter, R. X. Cheng, *et al.*, “Orca-spot: An automatic killer whale sound detection toolkit using deep learning,” *Scientific Reports*, vol. 9, no. 1, pp. 1–17, 2019.
- [7] P. C. Bermant, M. M. Bronstein, R. J. Wood, S. Gero, and D. F. Gruber, “Deep machine learning techniques for the detection and classification of sperm whale bioacoustics,” *Scientific Reports*, vol. 9, no. 1, pp. 1–10, 2019.
- [8] M. Ravanelli, T. Parcollet, P. Plantinga, *et al.*, *SpeechBrain: A general-purpose speech toolkit*, arXiv:2106.04624, 2021. arXiv: 2106.04624 [eess.AS].
- [9] B. S. Miller, N. Balcazar, S. Nieukirk, *et al.*, “An open access dataset for developing automated detectors of antarctic baleen whale sounds and performance evaluation of two commonly used detectors,” *Scientific Reports*, vol. 11, no. 1, pp. 1–18, 2021.
- [10] M. Bahoura and Y. Simard, “Blue whale calls classification using short-time fourier and wavelet packet transforms and artificial neural network,” *Digital Signal Processing*, vol. 20, no. 4, pp. 1256–1263, 2010.
- [11] T. Lu, B. Han, and F. Yu, “Detection and classification of marine mammal sounds using alexnet with transfer learning,” *Ecological Informatics*, vol. 62, p. 101277, 2021.
- [12] H. Yuea, D. Wanga, L. Zhanga, Y. Wua, C. Baoa, and D. Wang, “Classification of whale calls based on transfer learning and convolutional neural network,” *4th Underwater Acoustics Conference and Exhibition*, pp. 537–544, 2017.
- [13] A. Usman and D. Versfeld, “Detection of baleen whale species using kernel dynamic mode decomposition-based feature extraction with a hidden markov model,” *Ecological Informatics*, vol. 71, p. 101766, 2022.
- [14] N. Vaessen and D. A. Van Leeuwen, “Fine-tuning wav2vec2 for speaker recognition,” in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2022, pp. 7967–7971.
- [15] M. Singh and D. Pati, “Replay attack detection using excitation source and system features,” in *Advances in Ubiquitous Computing*, Elsevier, 2020, pp. 17–44.
- [16] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, “Phoneme recognition using time-delay neural networks,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989. DOI: 10.1109/29.21701.
- [17] B. Desplanques, J. Thienpondt, and K. Demuynck, “Ecapa-tdnn: Emphasized channel attention, propagation and aggregation in tdnn based speaker verification,” *arXiv preprint arXiv:2005.07143*, 2020.

- [18] V. Peddinti, D. Povey, and S. Khudanpur, “A time delay neural network architecture for efficient modeling of long temporal contexts,” in *Sixteenth annual conference of the international speech communication association*, 2015.
- [19] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, “X-vectors: Robust dnn embeddings for speaker recognition,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5329–5333. DOI: 10.1109/ICASSP.2018.8461375.
- [20] Pacific Northwest Seismic Network. “What is a spectrogram?” (2012), [Online]. Available: <https://pnsn.org/spectrograms/what-is-a-spectrogram>. (accessed: 23.07.2021).
- [21] Electrical4U. “Filter bank: What is it? (dct, polyphase, gabor, mel and fbmc).” (2021), [Online]. Available: <https://www.electrical4u.com/filter-bank/> (visited on 08/25/2022).
- [22] O. Ogundile, A. Usman, O. Babalola, and D. Versfeld, “Dynamic mode decomposition: A feature extraction technique based hidden markov model for detection of mysticetes’ vocalisations,” *Ecological Informatics*, vol. 63, p. 101306, 2021.
- [23] S. Imoscopi, “Machine learning for text-independent speaker verification,” 2016.
- [24] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.
- [25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [26] Y. Lim. “State-of-the-art machine learning hyperparameter optimization with optuna.” (2021), (visited on 04/08/2022).
- [27] D. S. Park, W. Chan, Y. Zhang, *et al.*, “SpecAugment: A simple data augmentation method for automatic speech recognition,” *arXiv preprint arXiv:1904.08779*, 2019.
- [28] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [29] C. Ayuya. “Using random search to optimize hyperparameters.” (), [Online]. Available: <https://www.section.io/engineering-education/random-search-hyperparameters/>. (accessed: 19.05.2022).
- [30] X. Bouthillier, C. Tsirigotis, F. Corneau-Tremblay, *et al.*, *Epistimio/orion: Asynchronous Distributed Hyperparameter Optimization*, version v0.2.4, May 2022. DOI: 10.5281/zenodo.3478592. [Online]. Available: <https://doi.org/10.5281/zenodo.3478592>.