Similarity Identification of Southern Resident Killer Whale (SRKW) Call Types Under Sparse Sampling Using Siamese Neural Networks

by

Jie Zhang

Submitted in partial fulfilment of the requirements for
the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2024

Dalhousie University is located in Mi'kma'ki,
the ancestral and unceded territory of the
Mi'kmaq. We are all Treaty people.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Southern Resident Killer Whales (SRKW) are highly intelligent marine mammals facing extinction in the North Pacific. These whales emit three types of sounds: clicks, whistles, and pulsed calls, with 43 distinct pulsed call types known as their "dialects". However, due to limited and poor-quality data, only nine call types have sufficient annotated recordings for analysis. To address this challenge, this paper proposes a progressive approach to improve SRKW call type identification. Initially, data augmentation techniques were employed to enhance training data volume, leading to a traditional CNN model achieving 97.8% accuracy on 17 SRKW call types. Subsequently, a Siamese Network model was developed to infer the similarity between call types, achieving remarkable performance with an accuracy of 98.5%. This surpasses the performance reported in current literature on audio multi-class classification using deep learning and machine learning methods. Besides, Siamese Network's generalization ability was evaluated on 9 out-of-training 9 SRKW call types, maintaining noteworthy accuracy and recall but with lower precision, which can be improved through manual review and retraining. This study demonstrates that data augmentation and Siamese Networks are effective strategies for overcoming few-shot learning challenges in SRKW call type identification, achieving robust performance even with limited annotated data.

Keywords: Marine Mammal Conservation, Southern Resident Killer Whales (SRKW), Acoustic Classification, Few-Shot Learning (FSL), Convolutional Neural Network (CNN), Data Augmentation, Siamese Network; Similarity measurement; Contrastive Learning, Transfer Learning; Meta-Learning.

LIST OF ABBREVIATIONS USED

SRKW	Southern Resident Killer Whale
FSL	Few-Shot Learning
CNN	Convolutional Neural Network
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
STFT	short-term Fourier Transform
MFCC	Mel-Frequency Cepstral Coefficients
CWT	Continuous Wavelet Transform
CQT	Constant-Q transform
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SNN	Siamese Neural Networks
ROC	Receiver operating characteristic curve
AUC	Area Under the Curve (AUC)
MAML	Model-Agnostic Meta-Learning

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Dr. Carlos Hernandez Castillo, for his invaluable patience and feedback. Without them I would not have made it through my master's degree.

I'm extremely grateful to my mother Jieying He and my wife Fang Tong, for their unconditional love and support and a special thanks to my late father, Professor Rui Zhang of Changsha University of Science & Technology in China. It was you who introduced me to the world of academia.

CHAPTER 1 INTRODUCTION

1.1 BACKGROUND

Orcas (Killer Whales) generate three types of vocalizations: Clicks, Whistles, and pulsed Calls. Clicks are part of the whale's sonar and are used for finding and locating food sources, other objects in the ocean and locating other whales. Whistles are typically continuous tone emissions lasting for many seconds, which are believed to serve for Social Cohesion and Contact, Individual Identification and Emotional Expression. Calls are pulsed signals characterized by distinct patterns that can be recognized both by ear and on a spectrogram. These signals are complex and varied, serving as Group Coordination, Social Interaction, Foraging, and Hunting within orca societies. Dr. John Ford [1] categorized the discrete call types for the orcas of Washington State and British Columbia. He discovered that each pod has its own collection of calls, which he referred to as their "dialect".

A quantitative measure of acoustic similarity is crucial to any study comparing call types of different social groups or individuals. A sound spectrogram is a visual representation of an acoustic signal. It is useful for seeing the state of a complex wave during a very short period. In a wave file, sounds are constantly changing. Spectrograms are a convenient way to illustrate the changes in a sound's spectrum over time.

In a spectrogram, the horizontal dimension represents time, and the vertical dimension represents frequency. Each thin vertical slice of the spectrogram shows the spectrum during a short period of time, using darkness to stand for amplitude. Darker areas show those frequencies where the simple component waves have high amplitude. A long window resolves frequency at the expense of time—the result is a narrow band spectrogram, which reveals individual harmonics (component frequencies), but smears together adjacent 'moments'. Figure 1 displays a spectrogram of the call type S01 of Southern Resident Killer Whale (SRKW) generated by Python and a black-and-white spectrogram clipped from the original catalog [1]. The spectrograms of all Southern Resident Killer Whale (SRKW) call types are available in the Orca call catalog website: http://orcasound.net.

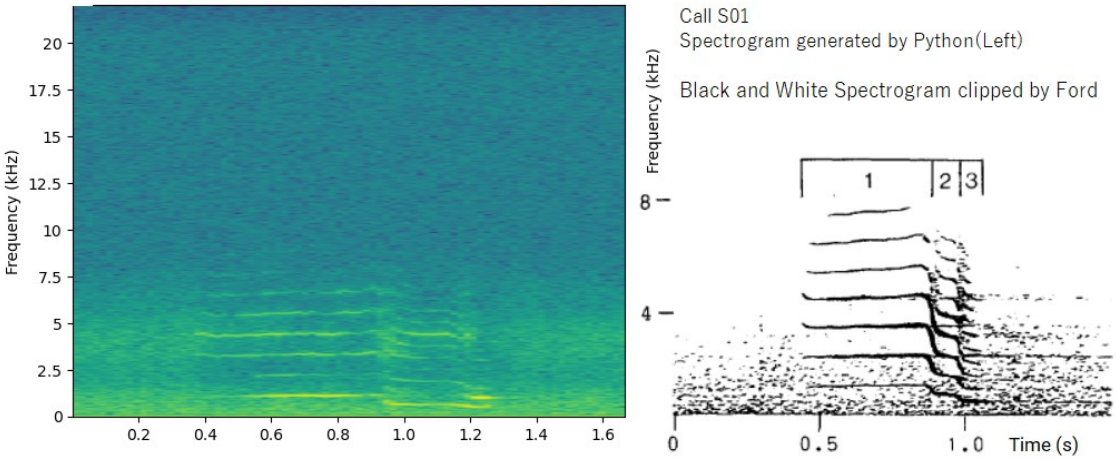

Figure 1 Linear Spectrogram generated by Matplotlib library in Python (Left) and a black-and-white spectrogram (Right) clipped from [1]

Identifying the call types of Southern Resident killer whales is critical for conservation, understanding their complex social structures, and enhancing behavioral studies. This endangered population faces threats from pollution, reduced prey, and noise interference, making their vocalizations key to monitoring and protection efforts. Analyzing these calls helps researchers understand their dynamics, behaviors like hunting and navigation, and facilitates the development of strategies to mitigate human impacts. For example, discovering a new call type during field studies can reveal unknown behaviors or social interactions, guiding conservation actions and informing policies to reduce disturbances in critical habitats. Essentially, understanding whale communication is vital for ensuring the survival of these marine mammals and maintaining the health of their ecosystems.

1.2 CHALLENGES

To leverage knowledge about Southern Resident Killer Whale (SRKW) Call Types, Simon Fraser University, Fisheries and Oceans Canada, and numerous marine acousticians collected audio data of SRKW from passive hydrophone devices deployed for over a decade in Washington State, USA, and British Columbia, Canada. According to their experience and knowledge, they annotated thousands of SRKW call-type clips from acoustic wave files collected from various locations along the Pacific coast, including Barkley Canyon, Boundary Pass, and Robert Banks. Each sound clip has an average duration of approximately 3-4 seconds. 53% of SRKW Call type annotated data from ONC Barkley Canyon node Datasets. 47% from JASCO Boundary and Robert Banks Datasets. Call Type Definition is based on Ford 1987 and 1991[1].

Using the Ford/Osborne call catalog for Southern Resident Killer Whales (SRKW), it is understood that there are approximately 43 Call Types, ranging from S1 to S46 (with some discontinuities and sub-types, such as S44a and S44b). This catalog likely serves as a valuable reference for identifying and categorizing the vocalizations of SRKW in the

collected audio data. However, among these 43 SRKW Call Types, only 10 Call Types have more than 5 annotated samples. This indicates a significant imbalance in the data distribution across the different Call Types, which can present challenges when implementing a classification model aimed at accurately classifying all 43 Call Types. Imbalanced data distributions may affect the performance and reliability of classification models. To address this issue effectively, strategies such as data augmentation, resampling techniques, or focusing on the more prevalent Call Types may need to be considered.

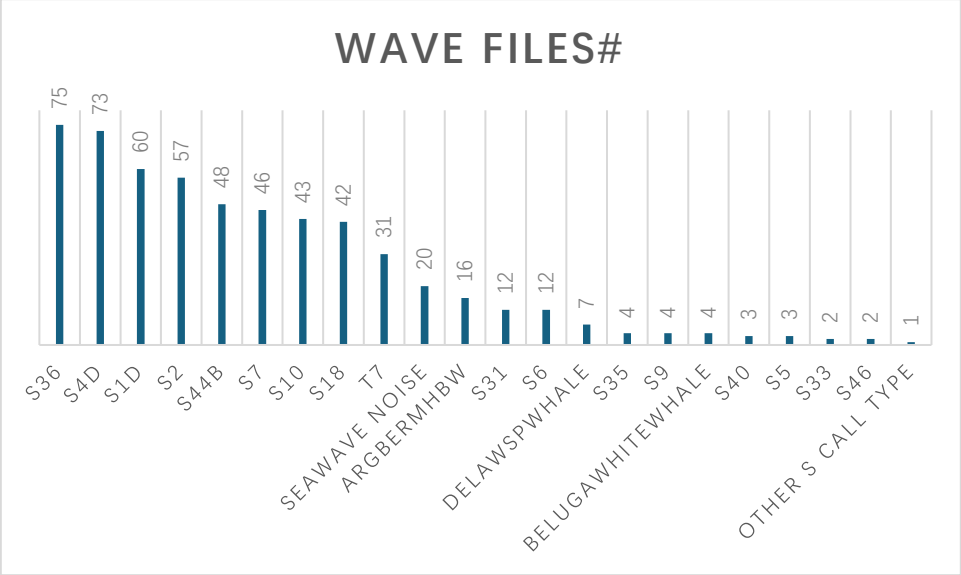

Figure 2 Distribution of the number of annotated audio clips by SRKW Call Type (created by the Author. For the figures and tables that do not mention a source, they were all created by the author)

Table 1 Remark of top 10 annotated Call Types in Figure 2, "Pod" is the population code for Orcas, and "S1d" represents the fourth subcategory of Call type S1 from orcasound.net.

Call Type	Pod	Description
S4d	K, L	Duck quack
S1d	J	Cowboy saying “Yee-haw!”
S2	J	“A-whee!”
S44b	K, L	"Goose honk"
S36	J, K	Donkey
ArgBermHBW		Bermuda Humpback Whale for comparison and transfer learning
S6	J	N/A

S10	J, K, L	"Squeaky balloon" Excitement call for all types of orcas!
S18	L	Slide whistle
T7		Bigg‘s (Transient) call. Killer Whales in Northern Pacific

1.3 RELATED WORKS

The classification of Call Types for Southern Resident Killer Whales (SRKW) falls within the realm of audio classification in machine learning techniques. Audio classification involves analyzing audio signals and categorizing them into various classes. It serves as a crucial tool in audio signal processing, aiding in the organization, analysis, and comprehension of audio signals. Classifying audio signals enables a better understanding of the underlying signals, their structures, and content, which, as mentioned earlier, is crucial for the study and conservation of the ecological behaviors of Southern Resident Killer Whales (SRKW). To achieve successful Call Type classification, the first step is to acquire annotated audio data. Models should be trained using annotated data to learn how to identify and classify different sounds. Despite advancements in audio classification, teaching machines to understand subtle differences in sounds and classify them accordingly remains challenging.

Traditional audio classification methods such as Support Vector Machine (SVM) [2], K-Nearest Neighbors (KNN) [3], Artificial Neural Networks (ANN) [4], and Hidden Markov Models (HMM) [5], Logistic Regression [6]. have been in use since the early 2000s. These methods involve extracting features from recordings and then using these features to classify audio into different categories. SVM is a powerful supervised machine learning algorithm that categorizes data points using hyperplanes, achieving high accuracy in classifying audio recordings [2]. KNN is a supervised learning algorithm used for classification, which finds the nearest neighbors of input instances and then classifies them based on the majority class of neighbors [3]. It is suitable for audio classification as it accurately categorizes signals using the features of audio signals. Artificial Neural Networks (ANN) are computational models based on the structure and function of biological neural networks [4]. They classify samples into different categories by learning features of audio samples. Logistic Regression [6] is used to divide data into two classes, which is useful for audio classification tasks such as speech recognition and music genre classification. The naive Bayes classifier is a probabilistic classifier used to compute the probability of each class for a given data point and then assign the class with the highest probability [7]. HMM is also commonly used for classifying audio data, particularly suitable for audio classification as it can learn the underlying structure of audio data and model the temporal dynamics of audio signals [5]. Gaussian Mixture Models are probabilistic models used for audio classification, assuming that each data point is a mixture of different Gaussian distributions, then used to classify input test audio data [8].

Deep learning has become increasingly popular in acoustic classification. With its ability to learn complex patterns, it can achieve better accuracy than traditional methods. Traditional methods usually divide audio classification into two processes: feature extraction and classification [9]. In the feature extraction process, relevant features are extracted from audio data, which are then used in the classification process to identify audio data. However, deep learning models require large audio datasets to train the network and automatically learn the features of each class. After training, the model can be used to classify new audio samples.

Deep learning models for audio classification can automatically extract high-dimensional features from large-scale datasets without manual feature extraction, as long as the input data contains all relevant information from the original data [10]. Deep learning models can achieve higher accuracy than traditional machine models because they can learn complex patterns and identify subtle differences in audio data, making them ideal for real-time audio classification and analysis.

However, deep learning also has some drawbacks. Deep learning models require substantial computational resources (including powerful graphics processing units and large amounts of memory) for training, which can be both expensive and time-consuming. Additionally, deep neural networks (DNNs) for audio classification systems require large datasets for training and evaluation; without large datasets, the system may fail [11]. In addition, in conventional DNN model training strategies, the model learns to solve a problem by analyzing the training data, and it can only recognize sound classes that were included in the training process (seen classes). The model is unable to classify sound classes that did not appear in the training data (unseen classes). To solve the problem of recognizing SRKW call types, the system must be able to detect and classify all seen and unseen sound classes. Conventional methods alone cannot achieve this goal. Although a large amount of annotated SRKW calls, e.g., S01, can be used to train a model, and such models can classify those call types inside the training set with high accuracy, identifying call types outside of the training data requires an alternative approach that overcome classification limits of a DNN model. This approach needs to work in parallel with the high-performance classification model to identify and classify those killer whale call types that were not learned during the training process.

As an alternative to traditional classification methods, One/Few Shot Learning (FSL) offers a promising approach for creating models that aren't strictly confined to recognizing only the classes they've been trained on. This technique aims to enable a model to compare two inputs and assess the likelihood that they belong to the same category [12]. The key advantage of One/Few-Shot Learning is its attempt to offer a universal solution that isn't tied to specific training classes. With a successful implementation, it should be possible for a model to compare any given input with any class, provided there's at least one or a few examples of that class available, thereby allowing the classification of an input into any class for which it has one or more samples.

This approach has demonstrated significant potential in the field of computer vision, particularly in applications such as image recognition [13] and face recognition [14]. However, the application of One/Few-Shot Learning to audio detection and classification is still relatively uncharted territory.

1.4 RESEARCH QUESTIONS AND CONTRIBUTIONS

With the rapid development of deep learning models in audio classification, naturally deep learning models are considered for Southern Resident Killer Whale (SRKW) call type classification using small-scale, low-quality samples by leveraging advanced data processing and augmentation techniques. Furthermore, given the 43 SRKW call types and limited annotated samples for each call type, this raises an idea whether it is possible to train a model to compare the similarity between SRKW call type data pairs as an alternative approach. Finally, is such a similarity model able to identify SRKW call types that were not included in the training data? Addressing these questions could significantly advance marine bioacoustics classification, particularly in contexts with limited data availability.

Question 1: Is it feasible to develop deep learning models for SRKW call type classification using small-scale, low-quality samples through advanced data processing and augmentation techniques?

Compared to traditional audio classification models. Deep learning models for audio classification can automatically extract high-dimensional features from large-scale datasets without the need for manual feature extraction, as long as the input data contains all the relevant information in the original data [10]. Deep learning models can achieve higher accuracy than traditional models because they can learn complex patterns and identify subtle differences in audio data. Most deep learning models can learn faster and more accurately than traditional models, making them ideal for real-time audio classification and analysis.

However, deep learning also has some disadvantages: deep neural networks (DNNs) used for audio classification systems require large datasets for training and evaluation; without large datasets, overfitting occurs when a model is overtrained [15], resulting in poor performance on new, unseen data [16]. On the other hand, underfitting may occur if the model is not trained enough. The available Southern Resident Killer Whale (SRKW) call type dataset has highly unbalanced annotated data, with some categories having a large number of annotations, such as S36 and S4, but 75% of the call types less than 5 annotated clips available, which is prone to overfitting.

Solutions are proposed to address the problems of insufficient training data, data imbalance, and poor data quality. Solution on sound signal processing, acoustic denoising, data enhancement, and CNN model training will be focused.

Question 2: Can small-scale samples be utilized to train models that compare the similarity between SRKW call type data pairs, as an alternative to employ multi-class classifiers for SRKW call type classification?

The core problem of this paper is to associate a certain call audio segment of SRKW with the specified label e.g. S0X, that is, classification. To achieve such a prediction, it is necessary to mark which Call type each sound clip belongs to.

However, if there are many call types but with very few samples per type, instead of identifying which call type a sound clip belongs to, it's easier to compare if two sound clips are similar or not. If the problem is constructed in this way, the more call types to be identified, the easier it becomes for a similarity model to learn inter-type and intra-type similarity features. By creating similar pairs (within-type) and dissimilar pairs (cross-type), along with applying data augmentation and noise reduction techniques, small call type dataset can be transformed into a larger one. In this approach, the model learns similarities rather than specific features of each call type. This method not only enlarge the original dataset but also reduces the complexity of model training.

Koch [13] proposed Siamese Network, which first shares the same weights and structure through two sub-networks, and then measures the distance or similarity between samples in the feature space through metric learning, thereby enhancing its ability to handle small sample classification or recognition tasks.

Question 3: Can a similarity model effectively discern SRKW call types that were not presented in the training dataset

If the Siamese Network performs well on the call types covered by the training dataset, it naturally leads to question whether the Siamese Network can still classify call types outside the training dataset. Although transfer learning and generalization are not easy to achieve, given that both the calls in the training dataset and the out-of-training call types are all call audios of SRKW (Southern Resident Killer Whale), it is hypothesized that the discriminative power of similarity model learned on the training set may also be effective on call types outside the training set.

1.5 THESIS ORGANIZATION

Table 2 Thesis Organization

Section	Sub Section	Contents
Introduction	Background	Provides context and current state of research in acoustic classification.

	Challenges	Discusses main technical and research problems.
	Related Works	A comprehensive review of the existing literature, theories, methods
	Research questions and contributions	Outlines three research questions in a logical progression.
	Thesis Organization	This table.
Acoustic Classification with CNN	Data Processing	Denoise, Data Augmentation and convert to Mel spectrogram vectors, Normalization and Data Persistence.
	Define CNN Model	Details the architecture and parameters of the CNN model.
	Train CNN model	Describes the training process and optimization techniques.
	Limitations and Challenges	Limitations and Challenges of CNN models.
From Acoustic Classification to Acoustic Similarity	Motivation	Why do I consider Similarity from Classification.
	Loss Functions	Contrastive loss and triplet loss.
	Feature Extraction	Details techniques for extracting relevant features from data For Siamese network.
	Dataset pairs	for Siamese Network input.
	Siamese Network Structure	Outlines the architecture of the Siamese network.
	Similarity Comparison	Describes methods for comparing acoustic signal similarity.
	Model Evaluation	Covers metrics and techniques used to evaluate model performance.
Experiments and Result	Software and hardware	Details computational resources and tools used.
	Classifying Multi-Call Type with CNN	The experiment details of CNN training
	Siamese network	The experiment details of Siamese network training

	Model Result and Discussion	The experiments result and answers to three research questions.
Conclusion and Future Work	Summary	Summarizes the main points and findings of the thesis.
	Conclusion	Transformation Based Method
	Disadvantages and Limitations	Draws conclusions based on the research findings.
	Future Work	Highlights research limitations and potential improvements.
References		Lists all cited references in the thesis.

CHAPTER 2 CNN-DRIVEN CALL CLASSIFICATION

In this chapter, solutions are proposed to address the deficiencies of insufficient training data, data imbalance, and poor data quality, focusing on sound signal processing, acoustic denoising, data augmentation, and CNN model training. A CNN-based classification model for Southern Resident Killer Whale (SRKW) Call Types is developed, achieving an accuracy of 97.8% on the test set (in 17 Call Types), reaching the industry's State-of-the-Art level in marine mammal acoustic recognition. A typical CNN model training process is shown in Figure3. This chapter will analyze the strengths and weaknesses of CNN models in SRKW call type classification across four sections: data processing, CNN model definition, CNN model training, and limitations of CNN models.

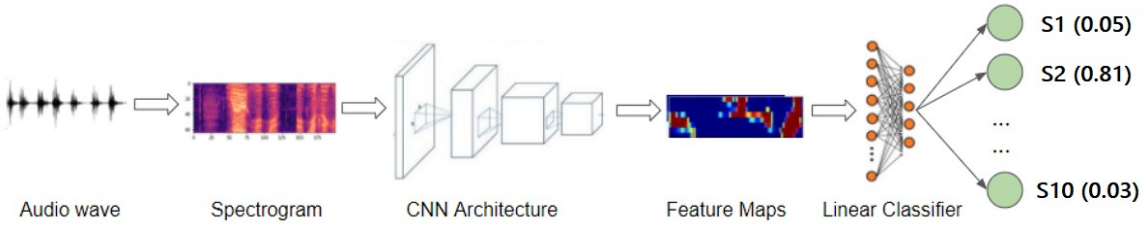

Figure 3 CNN model training process for SRKW Call Type Classification [17]

However, it's important to note that this is based on CNN's premise: it can only predict classes that appear in the training dataset. Classes not encountered in the training data render the CNN model incapable of prediction. To predict classes not seen in the training data (the Question raised in Question 3 of Section 1.4), algorithms based on transfer learning are required. To resolve the Question 3 of Section 1.4, this paper will employ a Few-shot learning algorithm, i.e., Siamese Network, with specific details elaborated in the next chapter.

2.1 DATA PROCESSING

2.1.1 Denoising for Marine Acoustic Data

Currently, all available Southern Resident Killer Whale (SRKW) call-type data come from hydrophones along the Pacific coast [17]. These sound data contain a significant amount of noise, such as hydrodynamic noise, water flow noise, wave noise, and ship propeller noise. These noises can mask call-type signals, severely impacting the identification accuracy of call-type classification models. In the literature, several methods for denoising marine acoustic data are commonly used:

Spectral subtraction is a frequency domain denoising method based on the signal-to-noise Ratio (SNR) concept. It estimates the power spectra of the signal and noise and calculates their difference. Then, based on a predefined SNR threshold, the noise component is subtracted from the signal spectrum to suppress noise. This method is simple to implement and effectively reduces stationary noise. However, its performance highly depends on the selection of the SNR threshold, and it may introduce artifacts [18].

Wavelet denoising decomposes the signal into subbands of different scales containing signal details and noise [19]. It applies a threshold to each subband to selectively remove

noise components and then performs wavelet inverse transform to obtain the denoised signal. Its advantages include preserving signal features while eliminating noise and the ability to handle non-stationary noise. However, careful tuning of wavelet type and decomposition levels is required, and signal edges may be distorted.

Adaptive filters adjust filter parameters dynamically based on the signal's autocorrelation and cross-correlation to adapt to different signal and noise environments [20]. The minimum mean square error (LMS) algorithm or minimum mean square error (LMSE) algorithm can be used to implement adaptive filtering. It can adapt to changes in noise characteristics and process large amounts of data in real-time. However, its denoising performance highly depends on parameter selection and convergence, and it may introduce artifacts if misconfigured.

Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN) can also be used to learn complex features of marine acoustic data and automatically denoise during training [21]. However, they require large datasets and computational resources. Traditional acoustic models involve modeling noise characteristics in the marine environment physically or statistically and denoising signals based on the model. They can be effective when noise statistics are known but may be less effective in complex noise environments.

The **noisereduce** Python library provides noise reduction functionality for audio signals. The **reduce_noise** function in the noisereduce module utilizes a combination of noise reduction techniques, including spectral subtraction, Wiener filtering, and adaptive filtering. After the reduce_noise function from this library was applied in experiments, the quality of the audio files was significantly improved. However, due to the limited availability of labeled data, in order to increase the number of labeled samples, both the original samples and the denoised samples were retained in the training set.

2.1.2 Data Augmentation

The limited size of annotated SRKW call-type samples results in reduced sample diversity. Data augmentation can address this issue by increasing sample diversity. Data augmentation involves modifying input data to enhance diversity in model input. Adding noise to spectrograms is a common technique used to improve model robustness. This process involves introducing random noise into the original spectrogram, which helps prevent overfitting and enhances generalization to unseen data. Noise can be added at various levels, such as individual frequency points or across the entire spectrogram. By simulating real-world environmental noise, this method enables the model to focus on relevant features while disregarding irrelevant noise. Overall, noise augmentation leads to more robust and dependable model performance across different conditions.

Frequency masking involves randomly selecting frequency ranges and setting spectrogram values within those ranges to zero, simulating the loss of certain frequency components in the audio. It simulates the loss of certain frequency components, enhancing model robustness to missing frequency information. Daniel S. Park [22]

proposed SpecAugment, a data augmentation method for automatic speech recognition, which includes frequency and time masking to disrupt input data. Time masking randomly selects time ranges and sets spectrogram values within those ranges to zero, simulating the loss of certain time segments in the audio. Cropping and padding randomly select subregions around the spectrogram to simulate different audio segment lengths. They introduce variability in audio segment lengths, promoting model robustness to different durations; however, if cropping or padding is excessive, they may distort the original context.

Shifting and scaling techniques involve making adjustments to the spectrogram along both the time and frequency axes, simulating positional changes of audio segments. By altering the frequency and temporal characteristics, these techniques enhance the model's adaptability to diverse audio variations. They effectively mimic shifts in the positions of audio segments, thereby improving the model's ability to generalize across different positions. However, it's important to note that excessive shifting or scaling can lead to the distortion of the original features of the data. Careful consideration and moderation are necessary to ensure that the augmented data remains representative of the underlying patterns while introducing beneficial variability to the training process [22].

Frequency flipping [23] involves flipping the spectrogram along the frequency axis, thereby altering the frequency characteristics of the audio. Similarly, **time flipping** [24] flips the spectrogram along the time axis, changing the temporal characteristics of the audio. These techniques modify both frequency and temporal attributes, thereby enhancing the model's adaptability to various audio variations. However, it's also important to be cautious when employing flipping techniques, as excessive use may introduce artifacts or distortions into the data. The right balance between adding useful variety and keeping the original audio intact need be considered.

These methods can be used individually or combined to increase training data diversity, improving model robustness and generalization. It's essential to adjust and experiment with specific methods based on dataset characteristics and task requirements. In summary, the combination of these augmentation techniques can effectively increase the diversity of training data, improve the robustness, and enhance the generalization ability of the model. It is crucial to experiment with and adjust these methods based on the characteristics of the dataset and the specific requirements of the task at hand.

2.1.3 SRKW Call Type Dataset

Based on annotated call type data from ONC Barkley Canyon, JASCO1 Boundary, and Robert Banks along the Pacific coast of British Columbia, the following preprocessing steps were undertaken to prepare the audio data for processing:

- Denoising and assessing the sound quality.

[1] JASCO Applied Sciences (https://www.jasco.com/) is a well-known company based in Nova Scotia, Canada, specializing in underwater acoustics and environmental monitoring. They provide a range of services and data related to acoustic signal processing, including underwater noise monitoring, marine mammal detection, and environmental impact assessments

- Augmenting the data to produce a minimum of 12 waveform clips for each call type, such as SRKW Call Type S40 (augmented from 3 clips to 12 clips). Despite there being 43 SRKW call types available, only 17 call types were selected for research and experimentation purposes (as outlined in Chapter 4). This decision was made to accommodate annotated data limitations.

Table 3 shows the train and test dataset by class. "Call Type" represents the specific type of call. " Augmented Training and Validation Wave Files#" stands for the data files for both training and validation after original audio files' data augmentation. Validation data (20% out of 88%) is used as a sub-set of the training data during CNN training to apply **early stopping** [25] and **learning rate scheduling** [26] to avoid overfitting. Augmented Testing Wave Files" (12%) indicate the number of wave files as test datasets.

Table 3 Train (including Validation, 88%) and Test Dataset (12%) by Call Type

Call Type	Non-Augmented Wave Files	Augmented Training and Validation Wave Files	Augmented Testing Wave Files	Remark
ArgBermHBW	12	60	4	SX: Southern Resident Killer Whale TX: Bigg's (Transient) Killer Whales in Northern Pacific ArgBermHBW: Bermuda Humpback Whales DelawSpwale: Delaware Sperm Whale Sea wave: Ocean background noise
DelawSpwale	10	50	10	
S10	19	48	10	
S18	16	50	6	
S1d	28	102	16	
S2	38	190	12	
S31	14	46	7	
S35	4	13	5	
S36	104	104	10	
S40	2	5	4	
S44b	18	90	7	
S4d	48	239	16	
S5	3	10	4	
S6	8	37	11	
S9	2	10	10	
Seawave	4	4	4	
T7	4	20	4	
Grand Total	334	1078 (88%)	140 (12%)	

2.1.4 Audio to Features

While the inputs to CNNs are audio or speech signals, CNN-based methods typically do not directly utilize the raw one-dimensional (1D) signals. Instead, as part of a preprocessing stage, 1D audio or speech signals are converted into 2D representations.

These 2D representations, which capture the spectrum frequencies of the audio signal over time, are then fed into a CNN model. A typical process for audio data processing for the CNN model is displayed in Figure 4.

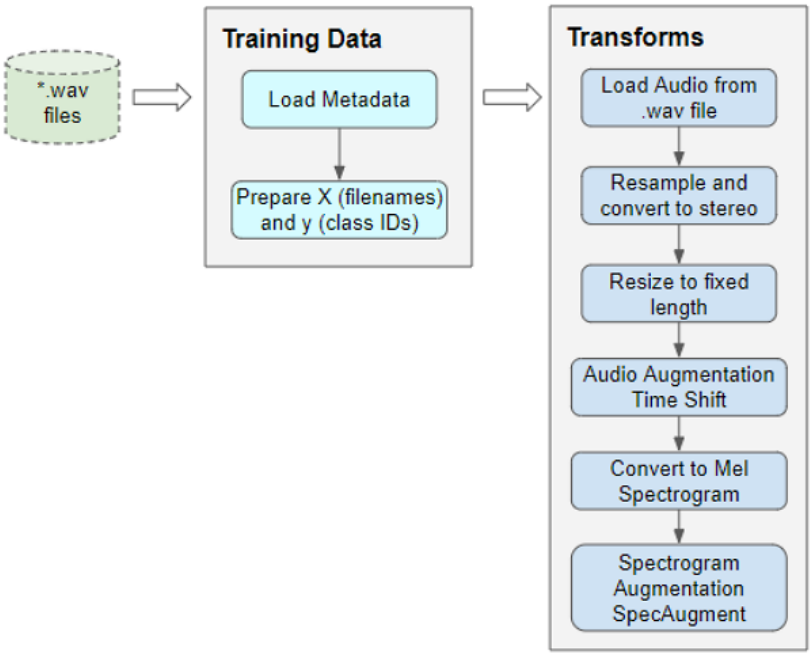

Figure 4 Pre-processing the training data for input for the CNN model

2.1.4.1 Sound and the Waveform

Sound is a physical disturbance that travels through an elastic medium as high- and low-pressure waves, known as compressions and rarefactions. Sound can be recorded by measuring the changes in pressure over time. Recording can be done analogically or digitally, with discrete values sampled at regular intervals called the sampling rate. Digital audio signals are specifically discussed in this text. In the air, microphones are used to record sound, while underwater environments require specialized hydrophones due to the impedance difference. When the variation in pressure over time is plotted, it creates a waveform; an example is shown in Figure 5. In digital audio, the amplitude in the Y-Axis is usually represented in a digital format, typically ranging from -1 to 1, indicating relative volume levels.

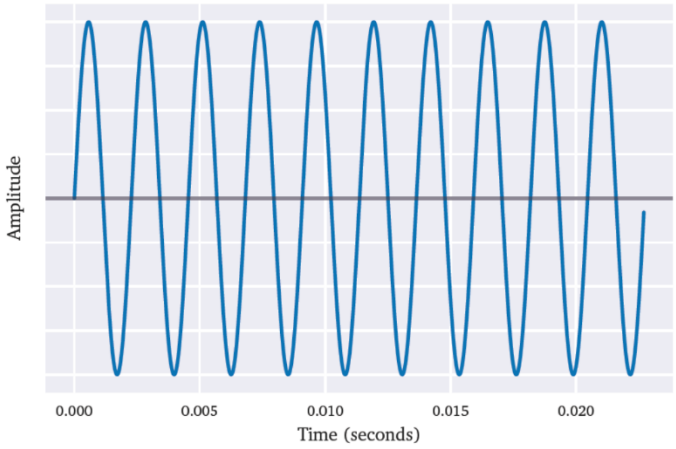

Figure 5 Concert A (440Hz) waveform [27]

Figure 5 shows a continuous waveform representing a sound signal, but it doesn't accurately represent digital audio recordings. In digital audio, recordings are composed of discrete numeric values called samples, recorded at specific time intervals. The range of values a sample can take is determined by the bit depth, and the number of samples recorded per second is the sampling rate [28]. A digital waveform is better represented as a scatter plot rather than a continuous wave. Figure 6 provides an example of how the sampling rate breaks down the continuous signal into discrete samples.

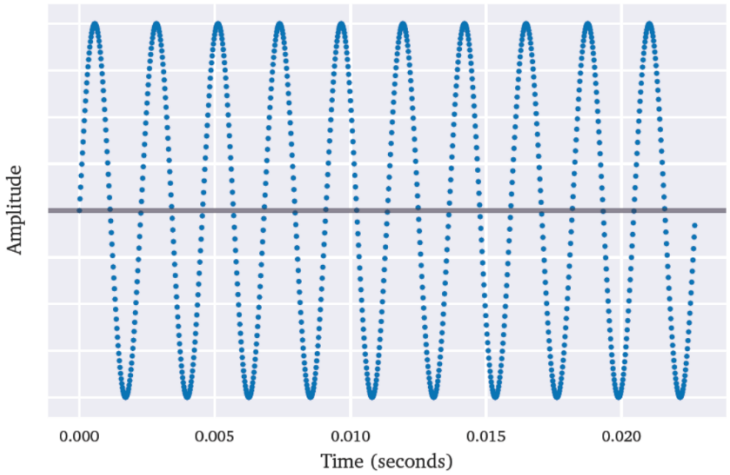

Figure 6 Concert A (440Hz) sampled at 44100Hz [27]

In real audio recordings with multiple sources and varying amplitudes and frequencies, the waveform appears more erratic, as shown in Figure7.

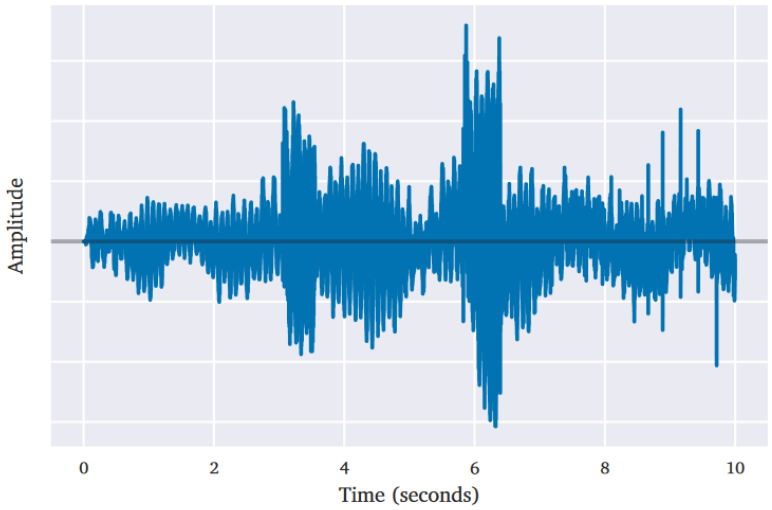

Figure 7 Audio waveform sampled at 44100Hz [27]

Nyquist-Shannon sampling theorem is a fundamental principle in digital signal processing. It establishes that for a continuous signal to be accurately reconstructed from its samples without introducing artifacts, the sampling rate must be at least twice the maximum frequency present in the signal. This maximum frequency, f_{max} is known as the Nyquist frequency, which is defined as $f_{max} = \frac{S_r}{2}$, where S_r is the sampling rate. If the sampling rate The Nyquist-Shannon sampling theorem is indeed a cornerstone in digital signal processing, specifying that to accurately reconstruct a continuous signal from its samples, the sampling rate must be at least twice the maximum frequency of the signal. This maximum frequency is known as the Nyquist frequency, defined mathematically as $f_{max} = \frac{S_r}{2}$, where S_r is the sampling rate. If the sampling rate is below this limit, aliasing occurs, resulting in distortion in the reconstructed signal.

Conversely, knowing the maximum frequency f_{max} needs to be captured allows you to determine the minimum required sampling rate, which is 2 f_{max}. This principle was instrumental in defining the sampling rate for digital audio formats like CDs, where a rate of 44.1 kHz was chosen to capture and accurately reproduce most of the audible frequencies for humans, which range up to approximately 20 kHz [29] .

2.1.4.2 The Fourier Transform

In natural environments, sound waves from different sources combine as the sum of individual waveforms at a specific point. The instantaneous pressure measured by a recording device represents the combined waveforms coinciding with that device. This means the waveforms from different sources can exhibit constructive or destructive interference when reaching the recorder [30]. However, visually determining the frequencies present in a waveform is not straightforward, even though it consists of the

sum of multiple frequencies. Analyzing the frequency content of a signal is often desired to better understand its waveform.

The Fourier Transform can be employed to convert the waveform from the time domain to the frequency domain. This transformation allows us to examine the individual frequencies contributing to the waveform. In this thesis, the discrete Fourier transform (DFT) is considered to decomposes discrete audio signals into their frequency components. DFT is defined by equation (1) and is used for signals containing N samples.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, exp\left(-i\frac{2\pi}{N}kn\right), k = 0, \dots N-1) \quad (1)$$

Where k is the k-th frequency component of the signal. In equation (1), the term exp $(-i\frac{2\pi}{N}kn)$ is a complex number written in exponential form. Therefore, the resulting sum will also be a complex number. As the real analog frequencies within the signal are only interested, i.e. complex symmetry in the DFT in Equation (1). This involves computing the same DFT for k $\leq \frac{N}{2}$. To ensure that the output amplitudes of the DFT components are correct, the magnitude of every component is averaged by multiplying its magnitude by $\frac{2}{N}$. The magnitude of a complex number is defined as equation (2):

$$\left|e^{ix}\right| = |cos\, x + isinx| = \sqrt{cosx^2 + sinx^2}, i = \sqrt{-1} \quad (2)$$

So, for a real-valued signal *S* consisting of *N* sampled values, discretized with a sampling rate of S_r, the magnitude of the real-valued frequencies of the signal below the Nyquist frequency could be computed by:

$$X(k) = \frac{2}{N}\left|\sum_{n=0}^{N-1} x(n)\, exp\left(-i\frac{2\pi}{N}kn\right)\right|, k = 0, \dots \frac{N}{2} \quad (3)$$

Where the analog frequency in Hertz corresponding to the *k-th* DFT component is defined as:

$$f_k = k\frac{S_r}{N} \quad (4)$$

An example of DFT computed for a generated 5-second sinewave of 5 Hz with an amplitude of 1 (with no unit) sampled at 80 Hz can be found in Figure 8. In Figure 8 the spectrum is only defined for frequencies below the Nyquist frequency $\frac{80Hz}{2} = 40Hz$.

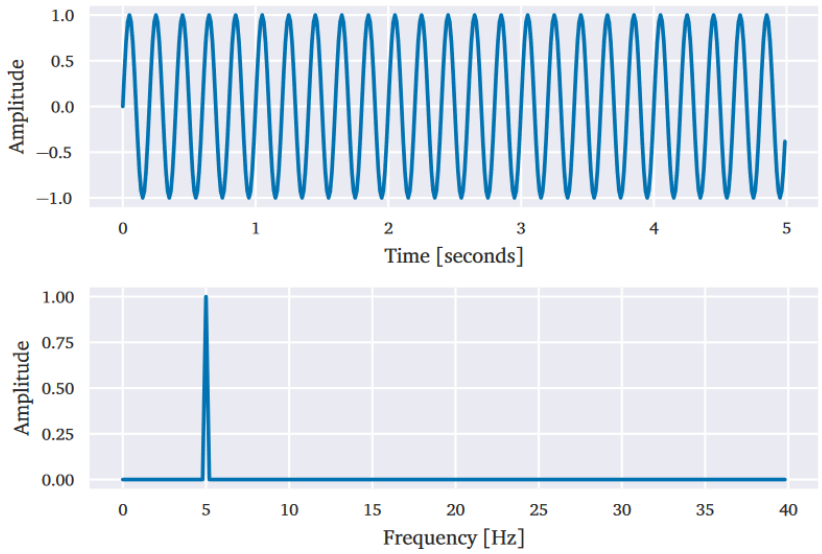

Figure 8 Example of real-valued Discrete Fourier Transform (DFT) (bottom) for generated 5 Hz sinewave (top) with unit amplitude sampled at 80Hz [27]

The Fourier transform is useful for signals with stable frequency content. However, when signals contain varying frequencies or amplitudes, the Fourier transform alone becomes insufficient for understanding how the signal's contents change over time. In natural environments, most sounds exhibit varying contributing frequencies. To examine how these frequencies and their amplitudes change over time, the Fourier transform can be utilized to convert the signal into its time-frequency representation. This representation can then be visualized and analyzed using a spectrogram, providing insights into the changing frequency content of the signal over time.

2.1.4.3 The Spectrogram - Short-Time Fourier Transform

As mentioned, for the sound preprocessing, 1D audio or speech signals are transformed from a 1D signal to a 2D signal. This 2D representation of the audio signal is then fed into a CNN model. The conversion from 1D to 2D is commonly performed to generate spectrograms, which capture the spectrum frequencies of an audio signal over time. Various techniques such as Fast Fourier Transform (FFT), short-term Fourier Transform (STFT), Mel-Frequency, Log-Mel-frequency, wavelet transform [31], and others can be employed to convert 1D audio signals into a 2D representation, as shown in Figure 10. Discrete Fourier Transform (DFT) is a method used to extract features from raw audio signals, converting signals from the time domain to the frequency domain to capture the phase and magnitude of each frequency component. However, DFT is not optimized and is challenging to apply to real-time discrete signals. In contrast, FFT, an optimized application of DFT, is suitable for real-time discrete signals and is defined as equation (5).

$$S(k) = \sum_{n=0}^{N-1} S(n) e^{-j\frac{2\pi}{N}kn} \quad (5)$$

The magnitude spectrum, |S(k)| of a signal, is the magnitude of its frequency bin or frequency component number (k) at a given sample number (n). It is usually a complex value [32].

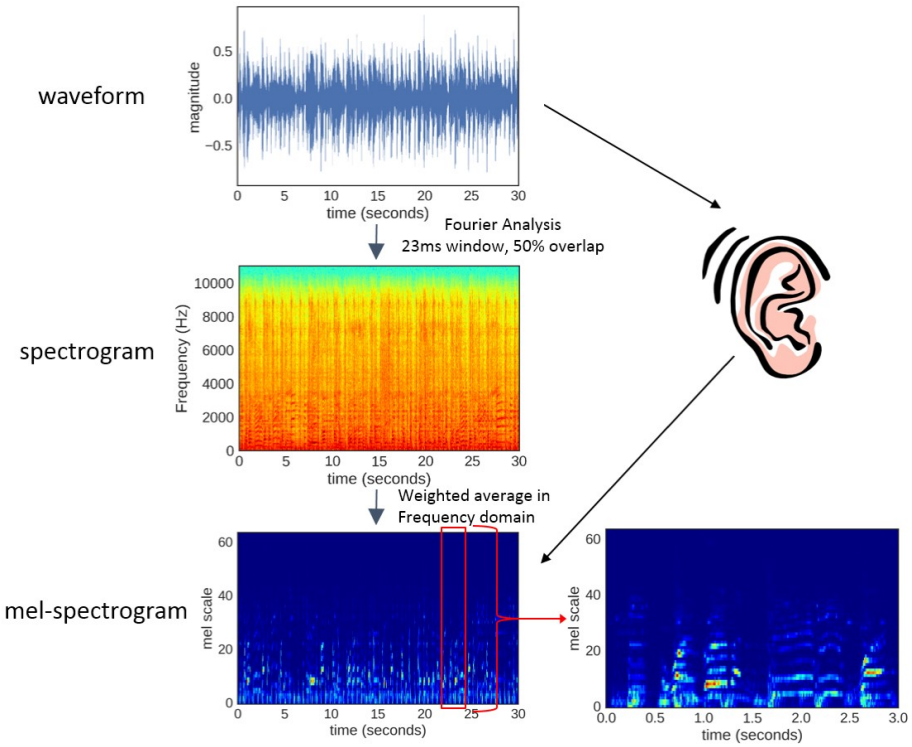

Figure 9 Convert waveform into Mel-spectrogram and an example 3-second segment. The Mel-spectrogram mimics how the human ear works, with high precision in the low-frequency band and low precision in the high-frequency band. Note that the Mel-spectrogram shown in the figures is already log-transformed [32]

The Short-Time Fourier Transform (STFT) [33] is an algorithm in which the Fourier transforms of successive signal windows are performed on a given time-domain signal. This process results in frequency spectra that are "stacked," enabling visualization of how the frequency spectra change over time. The signal is split into smaller windows, then Fourier transform is computed on the samples within each window. The window is shifted forward by some samples, repeated until the entire input signal has been traversed. The Short-Time Fourier Transform (STFT) is an enhanced form of the Fourier Transform designed to capture both temporal and frequency details of signals. By segmenting the signal into fixed-sized time-domain windows and applying the Fourier Transform to each segment, STFT reveals various signal features. Essentially, STFT employs equally spaced, identical, and symmetrical bandpass filters in the frequency domain to analyze the signal. The mathematical expression for any signal $s(\tau)$ can be written as:

$$S(f,t) = \int_{-T}^{T} s(\tau)\, w(-t) e^{-j2\pi f\tau} d\tau (6)$$

To derive this representation, the signal $s(\tau)$ is divided into segments using a windowing function w(t) as defined in Equation 6. The length of the window must match the length of the signal segments, assuming that the signal remains stationary within each window duration. The spectrogram is then obtained using STFT by computing the magnitude squared value of the time-frequency representation value [34]., as expressed by the equation:

$$Spectrogram = |S(f,t)|^2 (7)$$

A Mel-spectrogram, on the other hand, can be derived directly from the raw signal. It leverages the Mel scale, which offers a perceptually linear scale corresponding to Hertz, defined by the following equation [35]:

$$M(f) = 2595 * log10(1 + \frac{f}{700}) (8)$$

Where $M(f)$ represents the Mel frequency for a given f. It is derived from a logarithmic scale and is associated with the human perception of sound [35]. This formula is based on the premise that frequencies exhibit a logarithmic relationship to pitch. Thus, it is utilized to transform frequencies into a Mel-frequency scale, which more accurately reflects how humans perceive pitch.

Unlike the Short-Time Fourier Transform (STFT), the Continuous Wavelet Transform (CWT) does not depend on fixed window sizes and time shifts to determine time and frequency resolutions. Instead, the CWT utilizes a fundamental waveform called a "wavelet" to decompose the speech signal. This approach involves convolving the signal with shifted and scaled versions of the wavelet, accomplished through temporal shifting.

$$CWT(u,s) = \frac{1}{\sqrt{S}} \int_{-\infty}^{\infty} X(t)\, \Psi^*(\frac{t-u}{s}) dt (9)$$

In this equation, X(t) represents the speech signal, u and s denote the shift and scale parameters, respectively, ψ represents the mother wavelet or base function and $*$ denotes the complex conjugate operation. In the specific study referenced [36] (Vergara et al., 2020), the chosen mother wavelet is the Morlet wavelet. To summarize, raw 1D audio signals or spectrograms can be utilized as input to a CNN model.

Three spectrograms, the CQT (Constant-Q transform) spectrogram, Magnitude spectrogram, and Mel spectrogram, are widely used in Acoustic Classification [37]. The CQT Spectrogram, based on the Constant-Q Transform, utilizes varying frequency resolutions that align with the human ear's perception of pitch. It offers increased frequency resolution at higher pitches, making it particularly useful in music-related applications such as pitch analysis, instrument classification, and timbre feature extraction [38]. The Magnitude Spectrogram represents the amplitude of a sound signal, capturing changes over time and across frequencies. It is widely used in speech recognition, sound feature extraction, and music analysis, where visualizing the dynamic

amplitude of sound is crucial [39]. The Mel Spectrogram is derived from the Magnitude Spectrogram by applying Mel-frequency filters, which better simulate human auditory perception. It features higher frequency resolution in the lower range and is extensively used in tasks like speaker recognition, emotion recognition, and music genre classification [35].

By selecting the appropriate spectrogram type tailored to specific application requirements and sound characteristics, pertinent features of sound signals could be effectively extracted and represented. This facilitates sound recognition, classification, and analysis across various scenarios and tasks. In marine acoustic research, Magnitude Spectrogram and Mel Spectrogram are prevalent choices. Given the intricate frequency patterns often found in whale vocalizations, Magnitude Spectrogram is favored for its high-frequency resolution and adaptability. It enables researchers to visualize and explore the spectral intricacies of whale vocalizations, facilitating species identification, behavioral analysis, and other research objectives. Nonetheless, the selection of spectrogram type may vary depending on the research objectives and the unique characteristics of the whale vocalizations under study.

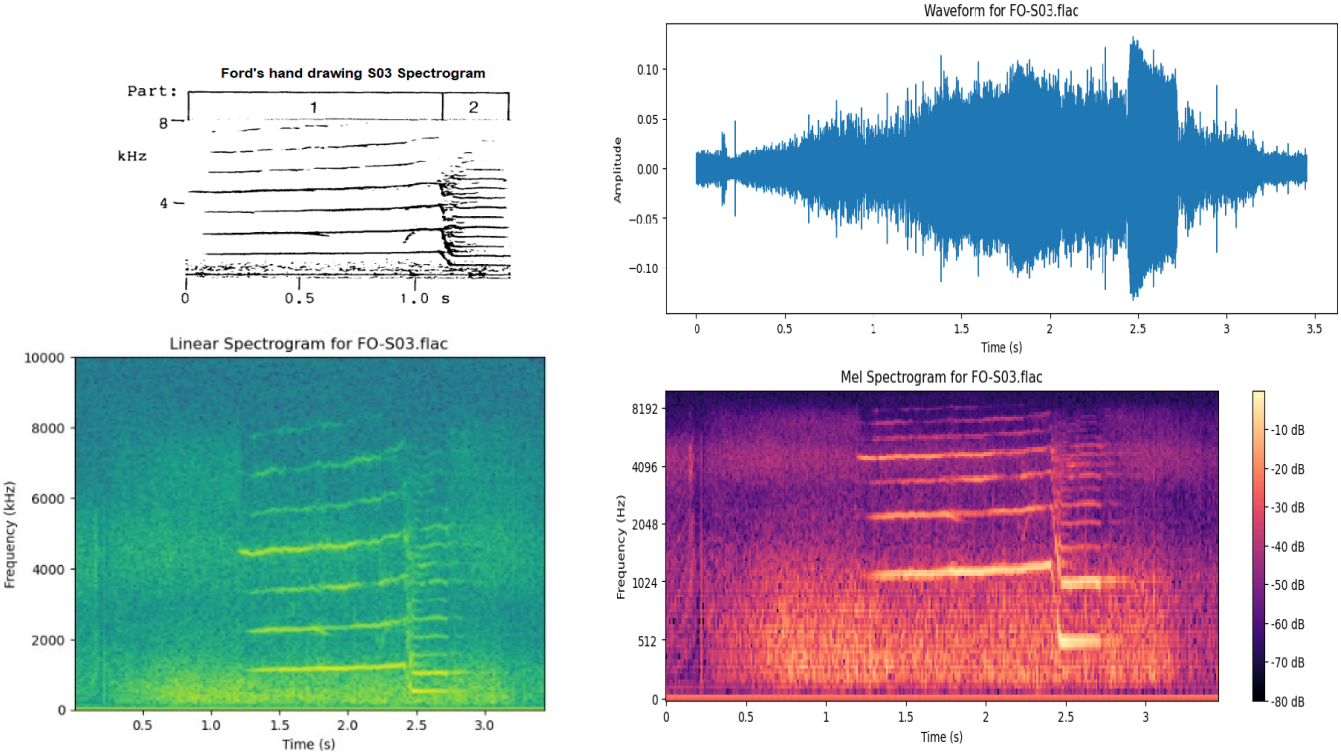

Figure 10 Comparison of waveform (time-domain), Linear Spectrogram (Frequence-domain) drawn by Ford, generated by Python Library Librosa and Mel Spectrogram for SRKW Call Type S03

As illustrated in Figure 10, relevant Call Type information is revealed more clearly in the Mel spectrogram. Therefore, in the experiment of chapter, wave files are converted to Mel spectrogram vectors as the input of CNN model.

2.1.5 Apply Normalization to Vectors

2.1.5.1 Standardization vs Normalization

Standardization is typically performed after data loading and before any model training. This process involves removing the mean and scaling the data to unit variance, which helps to handle different scales in the data and ensures that all features have an equal impact on the model [40]. The correct execution involves calculating the mean and standard deviation for each feature within the training set and then using these parameters to transform both the training and testing datasets. This method prevents information leakage and ensures fairness in model evaluation.

Similarly, normalization is applied before model training, usually right after data loading. For deep learning models, particularly those dealing with image data, normalization (such as Min-Max normalization or L2 norm normalization) aids in faster convergence [41]. L2 norm normalization scales each data point, such as an image or vector, so that its overall length (norm) equals 1. In contrast, Min-Max normalization adjusts the data range to a specified interval, usually [0,1] or [-1,1]. Normalization is crucial in processing image and audio data, as it helps the model handle varying input scales and distributions more effectively.

It is essential to ensure that the same standardization and normalization parameters are used for both the training and testing sets, which typically involves deriving these parameters from the training data and applying them consistently across all data splits.

In summary, data preprocessing involves standardization to ensure that features are on the same scale during model training and normalization to help models, especially those using distance-based algorithms, focus on the data's shape rather than its size. Given the context of audio vector data, normalization was performed in the practical code example provided.

2.1.5.2 Normalization for Acoustic Dataset

Applying normalization techniques to input vectors before the training of a CNN (Convolutional Neural Network) Acoustic Classification model can offer several benefits. Normalization helps in stabilizing the training process by ensuring that the input data to the network remains within a certain range. This can lead to faster convergence during training, as the optimization algorithm can more effectively update the model parameters [42]. Additionally, Normalization can mitigate the effects of overfitting by preventing the network from becoming too sensitive to the scale of input features [43] . By keeping the input data within a standardized range, normalization can help the model generalize better to unseen data. Normalization can improve gradient propagation through the network layers during backpropagation, making the training process more efficient and stable. This can alleviate the vanishing or exploding gradient problem, especially in deeper networks, leading to more stable and efficient training [44]. Normalization can also

enhance the robustness of the model to variations in input data by making it less sensitive to changes in scale, mean, or variance [45]. This can result in a more reliable model that performs consistently across different datasets or input distributions. Normalization techniques often require careful initialization of parameters, which can lead to more effective learning dynamics. This initialization strategy, combined with normalization, can help the network start training from a more stable and optimal state.

Overall, normalization techniques such as Batch Normalization or Layer Normalization can significantly contribute to stable training, faster convergence, and improved generalization performance of CNN Acoustic Classification models, provided their advantages and disadvantages are carefully considered and appropriately managed during model development and training.

2.2 Define CNN Model

2.2.1 CNN structure

With the training data now prepared for model training, the design of a neural network for training vector data can proceed. The upcoming sections will provide in-depth explanations of deep learning neural networks, focusing on the concepts utilized in the proposed systems. Initially, artificial intelligence algorithms were designed to mimic how the human brain learns from environmental activities. Hence, they are referred to as Artificial Neural Networks (ANNs). The term "deep" indicates that these networks have more layers than traditional systems and can become larger [46]. Subsequent chapters will also delve into basic concepts and methodologies from the literature of machine learning and deep learning for Acoustic Classification.

CNNs for acoustic classification typically comprise various components, including multiple convolutional layers, Rectified Linear Unit (ReLU) activation functions, pooling layers, fully connected layers (also known as dense layers), and a Softmax layer, as depicted in Figure 11. A convolutional layer in a CNN applies convolutional filters to the input signal to generate feature maps, which are then forwarded to subsequent layers for further processing.

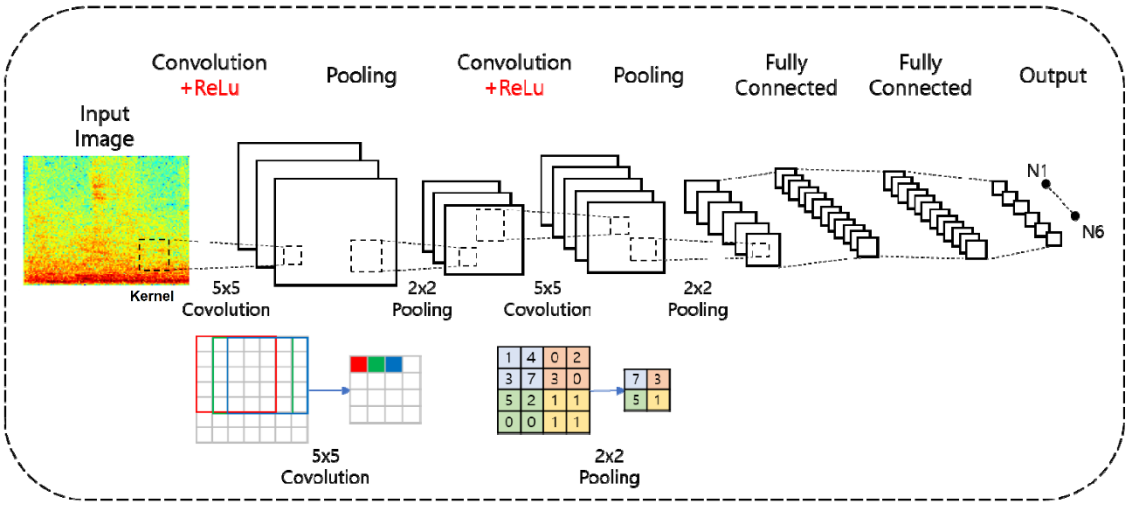

Figure 11 Structure of CNN [47]

Each layer within a Convolutional Neural Network (CNN) for acoustic classification plays a distinct role in the overall architecture. The following sections provide a brief explanation of the main layer types of CNN.

2.2.2 Input Layers

The input layer is the initial data entry point for the neural network. In the case of acoustic classification tasks using a Mel Spectrogram, the input layer consists of a 2D array that represents the time-frequency representation of the audio signal on the Mel scale. This Mel Spectrogram input captures the intensity of various frequency components over time, effectively modeling the perceptual characteristics of human hearing. By encoding these auditory features into a structured format, the CNN can then process and analyze patterns in the audio data, similar to how it handles image data in computer vision tasks. This approach is particularly effective for tasks such as speech recognition, emotion detection, and acoustic classification.

2.2.3 Convolutional Layer

Convolutional Neural Networks (CNNs) excel in handling multidimensional data, a strength recognized since their introduction by Y. LeCun [48]. A typical CNN architecture consists of numerous convolutional layers, each featuring a set of filters that learn through training. These filters are smaller than the input data and convolve across the input space to produce output layers. Filters are small matrices that are slid or convolved on the input data. In contrast, filters are a set of kernels used to extract various features from the input, allowing the neural network to learn hierarchical representations. The process captures the spatial and temporal dependencies within the data using fewer parameters, leveraging the strength of the filters to detect patterns efficiently. This mechanism enables CNNs to effectively identify and understand complex features in both images and audio representations.

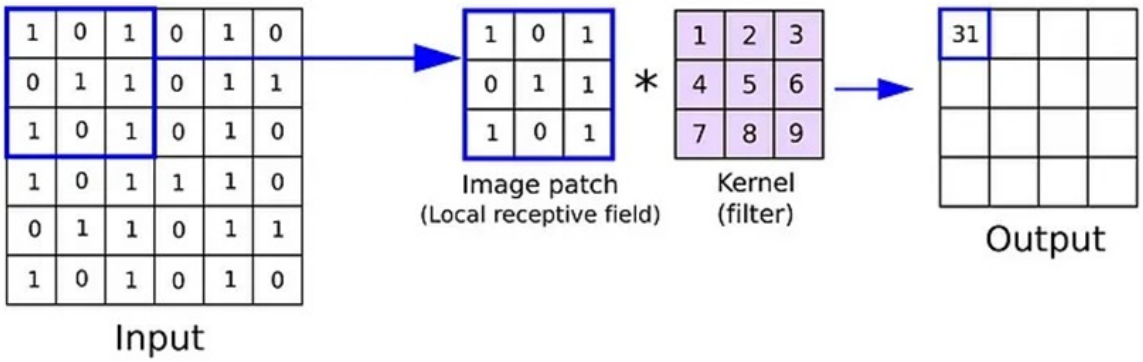

Figure 12 Filters in CNNs [46]

The filter is applied on each blue box and creates a cell in the output. Colors in the output are matched with the corresponding box color (Figure 12). The * denotes a convolution.

2.2.4 Pooling

The pooling layer is responsible for spatial size reduction. Pooling layers result in decreased computational power, and they are useful for extracting the dominant features in a positional-invariant and rotational way that can maintain the effective training process. Max and Average pooling are the common ways of implementing the pooling layer, as shown in Figure 13. In Max pooling, the filter returns the maximum value from the input, while the Average pooling returns the mean of the input. Pooling layers reduce the spatial dimension of the feature maps generated by convolutional layers. For example, Max Pooling selects the maximum value from a set of values, focusing on the most salient features. Meanwhile, average pooling reduces the spatial dimension.

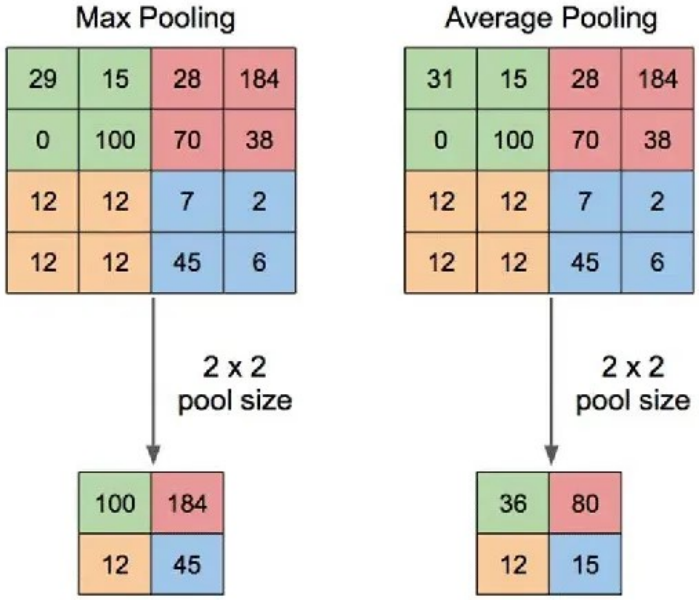

Figure 13 Max Pooling vs Average Pooling [46]

2.2.5 Fully Connected Layers

Fully connected layers (also known as dense layers) consist of neurons where each neuron is connected to all the input data that comes to the layer, either from a previous layer or directly from the model's input.

The mathematical operation performed by each neuron in a fully connected layer can indeed be divided into two parts. In the part of Linear Combination, the neuron computes a weighted sum of its inputs, along with a bias term. Mathematically, this operation can be represented as:

$$z_i = \pi r^2 \sum_{j=1}^{n} \omega_{ij} \times x_j + b_i \quad (10)$$

Where i is the neuron number, j is the input number. ij represents the weight of the connection between input, ω_{ij}represents the weight of the connection between input j, and neuron i. $x_j$represents the input value from the previous layer or the model's input. b_i represents the bias term for neuron i. n is the total number of inputs to the neuron. After computing the linear combination, the result is passed through an activation function. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, tanh, etc. This activation introduces non-linearity into the model, enabling it to learn complex patterns and relationships within the data. This process is repeated for each neuron in the fully connected layer, producing the layer's output, which then serves as input to the subsequent layer in the neural network. Figure 14 typically illustrates the mathematical representation of a single neuron, highlighting how it takes input signals, computes a weighted sum, adds a bias, and applies an activation function to produce an output.

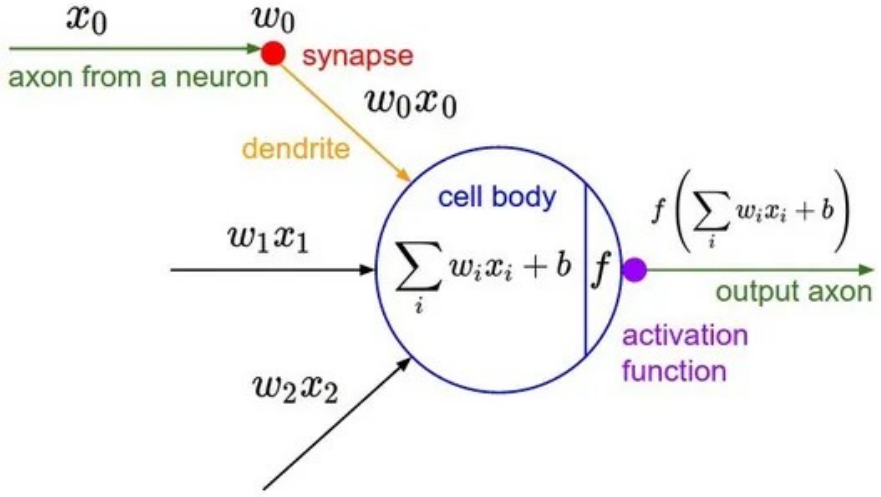

Figure 14 A schematic drawing of the activation function [46]

2.2.6 Activation Functions

Activation functions are integral parts of any neural network as they allow the model to go beyond the trivial linear problems and generalize and adapt with various nonlinear

combinations of input passing through multiple layers. There are many activation functions, such as Rectified Linear Unit and Sigmod. Rectified Linear Unit (ReLU) is a piecewise function that outputs the input directly for positive inputs and returns zero otherwise. Figure 15 shows the plot for this function, which is described as,

$$f(x) = max\ (0, x) \quad (11)$$

where x is the input.

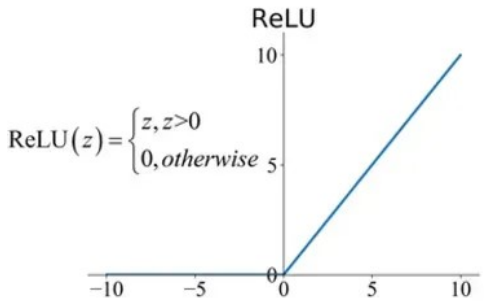

Figure 15 ReLU activation function [46]

ReLU became quite popular due to its robustness in vanishing gradients and sparsity. Another advantage of ReLU is the low computational costs compared to much more complex activation functions. It helps the network learn complex relationships and makes the model more expressive. Which activation to use completely depends on your use case; in most cases, researchers use ReLU, but some activations can also be used, such as Sigmod. Sigmoid is a mathematical function that has an "S" shaped curve. A common example of such a function is the logistic function shown in Figure 16 and equation (12) [49]. Such function would be monotonic, continuous, and differentiable everywhere such as,

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (12)$$

Where x is the input.

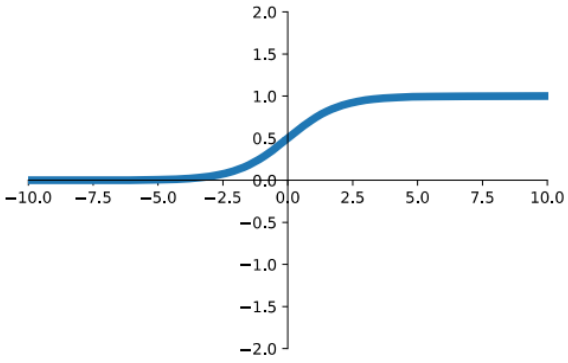

Figure 16 Sigmoid activation function [46]

2.2.7 Dropout layer

The Dropout layer is used for regularization to prevent overfitting. During training, random neurons are "dropped out", which means they are ignored, forcing the network to learn more robust and generalizable features. It helps prevent overfitting by randomly ignoring a small portion of input units during training.

2.2.8 Batch Normalization

Introduced by two Google researchers [40], batch normalization enhances the training efficiency and stability of neural networks by re-centering and re-scaling hidden layers. Although the exact rationale behind batch normalization remains under investigation, its efficacy has been demonstrated [40].

Batch normalization (BN) is employed in neural networks to expedite and stabilize the training process. It standardizes layer inputs through adjustment and scaling during training. The mathematical foundation of batch normalization encompasses normalization, scaling, and shifting operations. Let us delve into the mathematical framework of batch normalization. Consider a mini-batch of size *m* containing *n* features, the input to batch normalization can be summarized as follows:

1. **Mean Calculation c**alculates the mean μ of the mini-batch for each feature, where x_i denotes the value of the *i-th* feature in the mini-batch.
2. **Variance Calculation** compute the mini-batch variance σ^2 for each feature.
3. **Normalization** standardizes the input by subtracting the mean and dividing by the standard deviation (σ)

$$Normalized\ value = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} \quad (13)$$

Here, ε is a small constant added to avoid division by zero.

4. **Scaling and Shift** introduce learnable parameters γ and β to scale and shift the normalized values:

$$Output = \gamma \times Normalized\ value + \beta \quad (14)$$

Here γ represents the scale parameter, and β represents the shift parameter. Batch normalization operations are typically inserted into neural network layers before activation functions. It has been demonstrated to possess regularization effects, alleviating issues such as internal covariate shift, thereby rendering training more stable and expedited.

2.2.9 Flatten Layer and Regularization

Flatten layer transforms multi-dimensional feature maps into one-dimensional vectors, preparing the data for input into fully connected layers. Regularization is a technique used to control the process of fitting the model to the training set and avoid overfitting. This method tries to discourage the model from learning too much complex function that fits perfectly into the training set but would perform poorly on the unseen test set [50].

2.3 Train CNN model

2.3.1 Data Segmentation

A typical training process for the CNN audio classification model will start with segmenting the audio spectrograms into training, validation, and test sets [51]. Training Set is used to train the CNN model's parameters by feeding the spectrograms and their corresponding labels into the model. The model learns to recognize patterns and features from the training data, adjusting its weights through backpropagation to minimize the chosen loss function. A larger training set helps the model learn diverse features and generalize unseen data better. Validation set is used to evaluate the model's performance during training and tune hyperparameters. By monitoring metrics such as accuracy or loss on the validation set, early signs of overfitting or underfitting can be detected, allowing for necessary adjustments. This set assists in selecting the best-performing model and helps prevent over-optimization to the training data. Once the final trained model is generated, the test set is reserved for evaluating its performance. It provides an unbiased estimate of the model's generalization ability to unseen data and helps assess its robustness and reliability in real-world scenarios.

Typically, a certain percentage of the data is allocated to each set, such as 70% for training, 15% for validation, and 15% for testing. Segmenting the data in this manner ensures that the model's performance is rigorously evaluated and helps prevent data leakage or bias in performance estimation.

2.3.2 Define Model

Based on the process described in Section 2.2, the architecture of the CNN model is defined with several key components: Convolutional Layers, Activation Functions, Pooling Layers, Fully Connected Layers, and the Output Layer. The Convolutional Layers apply learned filters to the input spectrograms to capture spatial hierarchies of features. These layers perform a series of convolution operations represented as:

$$z^{(l)} = W^{(l)} * x^{(l)} + b^{(l)} \quad (12)$$

where $W^{(l)}$ represents the filters at layer l, $x^{(l)}$ is the input to the layer, $b^{(l)}$ is the bias term, and $*$ denotes the convolution operation. The output $z^{(l)}$ is then passed through a non-linear activation function such as ReLU, defined as:

$$a^{(l)} = ReLU\left(z^{(l)}\right) = \max(0,\ z^{(l)}) \quad (12)$$

After convolution and activation, pooling layers reduce the spatial dimensions, often through max-pooling operations, to retain the most prominent features while lowering computational complexity. Pooling is represented as:

$$a_{pool}^{(l)} = max_{pool\_regon}^{\square}(a^{(l)}) \quad (13)$$

The resulting feature maps are then flattened and passed into Fully Connected Layers, where the learned features are combined to produce the final predictions. These layers apply matrix multiplication followed by an activation function, typically represented as:

$$z^{(l+1)} = W^{(l+1)} * a^{(l)} + b^{(l+1)} \quad (12)$$

Finally, in the Output Layer, a softmax function is applied to compute the class probabilities for multi-class classification problems:

$$P(y = c|x) = \frac{e^{z_c}}{\sum_{k=1}^{C} e^{z_k}} \quad (12)$$

Where z_c is the output for class c and C is the total number of classes.

Before training begins, the CNN model is initialized with random weights, often sampled from a distribution such as Xavier or He initialization [53]. During training, the weights are iteratively adjusted using the backpropagation algorithm, which involves computing the gradient of the loss function with respect to each weight. This gradient is computed using the chain rule of calculus:

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial z^{(l)}} \times \frac{\partial z^{(l)}}{\partial W^{(l)}} \quad (12)$$

where L is the loss function, typically cross-entropy for classification tasks. The gradients are then used to update the weights via gradient descent or its variants, such as stochastic gradient descent (SGD) or Adam. The weight update rule for SGD is expressed as:

$$W^{(l)} = W^{(l)} - \eta \times \frac{\partial L}{\partial W^{(l)}} \quad (12)$$

where η is the learning rate. In adaptive optimizers like Adam, learning rates are adjusted for each parameter based on estimates of first and second moments of the gradients, leading to faster convergence [54].

This process of forward propagation, loss computation, backpropagation, and weight updates continue iteratively over the training data until convergence. The architectural design of the CNN model balances model complexity, computational efficiency, and performance, ensuring that the model can effectively capture features from the audio spectrograms while being computationally feasible.

2.3.3 Training Loop

In the training loop, the training data is iterated over in mini-batches. For each batch, a series of steps is performed. First, the input spectrograms are passed through the CNN model to compute the predicted probabilities for each class. Next, the loss between the

predicted probabilities and the actual labels is calculated using the chosen loss function. After calculating the loss, the gradients of the loss with respect to the model parameters are computed using the backpropagation algorithm. Finally, the model parameters are updated using the optimizer based on the computed gradients.

After each epoch, the model's performance is evaluated on the validation set. This evaluation involves computing metrics such as accuracy, precision, recall, and F1-score to assess the model's effectiveness. Early stopping is implemented by monitoring the performance on the validation set and halting the training process if performance does not improve over a specified number of epochs. This technique helps prevent overfitting and ensures that the model generalizes well to unseen data.

2.3.4 Model Evaluation

Once training is complete, the final model is evaluated on the test set to obtain unbiased performance metrics. Five metrics—Accuracy, Recall (Sensitivity), Specificity, F1, and Precision—are computed on the test set to assess the model's generalization ability.

2.3.5 Result Interpretation

Finally, the results are analyzed to understand the model's strengths and weaknesses. The analysis helps identify any patterns or classes that the model struggles to classify accurately. Based on this evaluation, adjustments to the model architecture, hyperparameters, or data preprocessing steps may be made to further improve performance if necessary.

2.4 Limitations and Challenges of CNN Models

Training CNN models on small-annotated call type dataset may encounter the challenge of overfitting because of following reasons Training CNN models on small-annotated call type datasets may encounter the challenge of overfitting due to several factors:

- **Limited dataset size**: Small-sample datasets may not provide enough data to capture comprehensive features, leading to the model's inability to generalize well to new data samples.
- **Model complexity**: If the model's complexity is too high, it may attempt to memorize noise and outliers in the training set rather than learning general patterns, resulting in poor performance on new data.
- **Data imbalance**: When the number of samples varies significantly across different classes, the model may focus more on the classes with more samples and neglect those with fewer samples, leading to decreased performance.

One potential solution for small-annotated call type dataset classification is the use of Siamese networks [10]. Siamese networks are neural network architectures used for metric learning and are commonly applied to address small-sample classification

problems. These networks learn a similarity measure between samples to perform classification. In the next chapter, the similarity measure will be elaborated upon, and Siamese networks will be applied to resolve the few-shot learning (FSL) problem for SRKW call type classification.

CHAPTER 3 EVOLUTION TOWARDS CALL SIMILARITY FROM CLASSIFICATION

The distinction between similarity measurement and classification in audio processing is crucial. Similarity measurement compares features between two data points to quantify their resemblance, making it simpler than classification, which requires mapping a data point to one of many categories. Recent research highlights advance in machine learning models tailored for similarity-based tasks.

For example, the study of audio similarity search using distance measures such as Gaussian Mixture Models (GMMs) or Hidden Markov Models (HMMs) demonstrates the relative simplicity of measuring similarity by comparing distributions of feature vectors directly, rather than classifying them into categories [55]. However, research on similarity measurement using deep learning in the audio processing domain, especially when it comes to utilizing deep learning for similarity measurement, has been relatively limited [56]. Hence, despite the potential importance of similarity measurement in the audio processing domain, there are still many challenges and research opportunities. This chapter investigates Siamese networks for SRKW Call Type similarity measurement.

3.1 Motivation (From CNN to Siamese Network)

This thesis aims to associate an audio clip of a call with an assigned label “Call Type”, namely classification. Achieving such predictions requires not only a labeled sound dataset but also a method to measure whether a sound corresponds to a certain label. Instead of asking the question "which call type does sound X belong to?", the problem can be reframed as a task of measuring sound similarity, essentially transforming the question to "does sound X1 belong to the same class (call type) as sound X2?"

In the realm of deep learning, employing Siamese Neural Networks (SNNs) to address the challenge of measuring similarity between inputs is a prevalent approach. This approach can be motivated by first examining some of the limitations inherent in the common approach to this general problem of multi-class classification.

In traditional multi-class classification tasks, neural networks are typically equipped with an output layer consisting of neurons denoted as $\{a_i^{[L]}\}_{i=1}^{m}$ activated by the softmax function:

$$softmax(a_i)=\frac{e^{a_i}}{\sum_{i=1}^{m} e^{a_j}} \qquad (15)$$

The softmax activation function is employed because its output values resemble class probabilities, satisfying the requirement that the sum of all probabilities equals 1 ($\sum_j a_j$=1). Training such a network typically involves using the negative log-likelihood

function as the loss function and employing one-hot encoding for the labels. The choice of the negative log-likelihood function as the loss function is inspired by concepts from entropy and information theory, as explained in Chapter 1 of [57]. Additionally, it is possible to train the classification layer directly using the cross-entropy loss function without the need to activate the output layer with softmax. However, during inference, activating the output by softmax is still necessary to obtain meaningful predictions. One-hot encoding represents the label as a binary vector where the target class is marked with a 1, and all other classes are marked with 0. This encoding can be interpreted as a probability distribution where the target class has a probability of 1 and all others have a probability of 0.

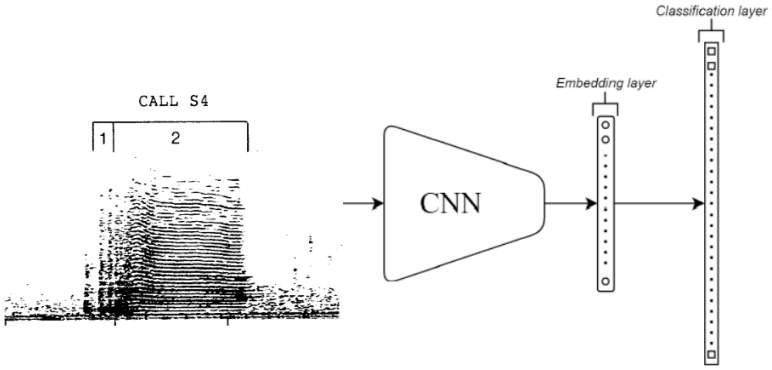

Figure 17 The CNN architecture for call type classification

The input spectrogram initially undergoes processing through CNN architecture (Figure 17). This process aims to encode the image into its fundamental components, represented by the embedding layer, which is depicted as a 1×D dimensional vector (illustrated as a rounded rectangle with circles to represent the neurons in the embedding layer). Subsequently, the embeddings are fed into a classification layer, which outputs a probability distribution over the number of classes, where each output neuron corresponds to a class.

The approach of category prediction, while effective, comes with significant drawbacks. A fundamental requirement for any deep learning model is a substantial dataset to effectively teach the model about the general distribution of each class. However, in SRKW call type classification scenarios, acquiring large amounts of annotated and labelled data can be expensive and time-consuming. While data augmentation techniques can mitigate this issue to some extent (see section 2.1.2), obtaining original data remains crucial for most deep-learning tasks.

Another limitation of the category prediction approach is its lack of flexibility in accommodating new classes. If a new class (new SRKW call type) needs to be included, the entire network must be retrained, and a new neuron must be added to the classification layer.

Multi-class classification by similarity prediction addresses the limitations highlighted in category prediction, a viable solution is to train the model to discern similarity or dissimilarity between two calls. Siamese Neural Networks (SNNs) are commonly employed to train a neural network to measure similarity between inputs [58]. The fundamental concept behind SNNs involves establishing multiple instances of the same network that share both the network architecture and parameters. The primary objective of the network architecture is to decode or down-dimensionalize the inputs into an embedding space intelligently. This process ensures that inputs belonging to the same class are mapped closely together in some sense, while inputs from different classes are mapped farther apart. Figure 18 illustrates the core idea of such architecture and parameter sharing, along with how a trained decoding network effectively maps similar inputs close together.

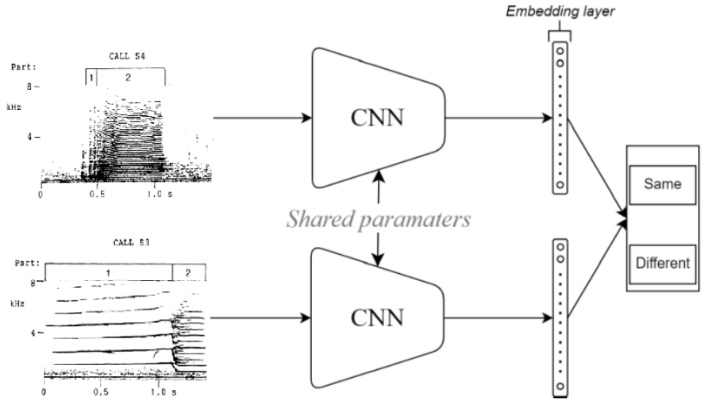

Figure 18 The structure of Siamese Neural Networks

The concept of parameter-sharing architecture was initially introduced by Bromley [59] for signature verification and independently developed for matching pairs of fingerprints by Baldi [60]. Since their inception, they have become more effective in recent years, largely due to hardware advancements, although they typically require a significant amount of time to train.

In the context of this thesis, both category and similarity prediction approaches learn to generate an embedding vector indirectly, meaning that the loss function is not explicitly designed to operate directly with the embeddings. Instead, training is conducted by considering the embedding vector itself rather than the outputs of a classification layer that indicates "same" or "different". This approach is inspired by the findings of the FaceNet [61], which demonstrated superior performance when training the embeddings directly compared to DeepFace [62], which utilized a classification layer for embedding training.

3.2 Loss Functions

The effectiveness of Siamese networks largely depends on the choice of the loss function during training, which guides the network to learn discriminative features. There are several loss functions used in Siamese Neural Networks.

3.2.1 Contrastive Loss

Contrastive loss is a widely used loss function for training Siamese networks [63]. It aims to ensure that pairs of similar inputs are closer in the embedded space, while pairs of dissimilar inputs are farther apart. The loss function is defined as follows:

$$L(Y, D) = (1 - Y)\frac{1}{2}(D)^2 + (Y)\frac{1}{2}max(0, m - D)^2 \qquad (16)$$

Where Y is the label indicating if the pair is similar (0) or dissimilar (1). D is the Euclidean distance between the outputs of the two inputs in the pair. m is the margin, a hyperparameter that defines how far apart the dissimilar pairs should be. D aims to quantify the distance between the embeddings $f(.)$ of the Acoustic vectors X_1 and X_2:

$$D(X_1, X_2) = \|f(X_1) - f(X_2)\| \qquad (17)$$

The advantage of contrastive loss is that it is effective in learning discriminative features by pushing apart dissimilar pairs beyond a margin, which is relatively simple to implement and understand. However, choosing an appropriate margin can be challenging. It may not fully exploit the structure of the embedding space, focusing only on pairwise distances.

3.2.2. Triplet Loss

The Triplet Loss function, described in detail in the FaceNet [64], highlights its effectiveness by exploiting relationships between three different data in a single observation, called triples, to produce high-quality image embeddings. A triplet includes an anchor A, the input image (or acoustic vector) embedding as a reference point; a positive P: another input image embedding with the same identity or category as the anchor, which means it is similar to the anchor; a negative N: an input image embedding that have a different identity or category than the anchor, making them dissimilar. The goal of the Triplet Loss function is twofold. First, it minimizes the distance between anchor points and vertex embeddings. This is the “pull-in” aspect, which encourages the network to bring embeddings of similar images or share the same identity closer; secondly, it maximizes the distance between the anchor and negative embeddings. This is the “pushing away” aspect, which aims to separate the embedding of anchors from the embedding of dissimilar or dissimilar identity images.

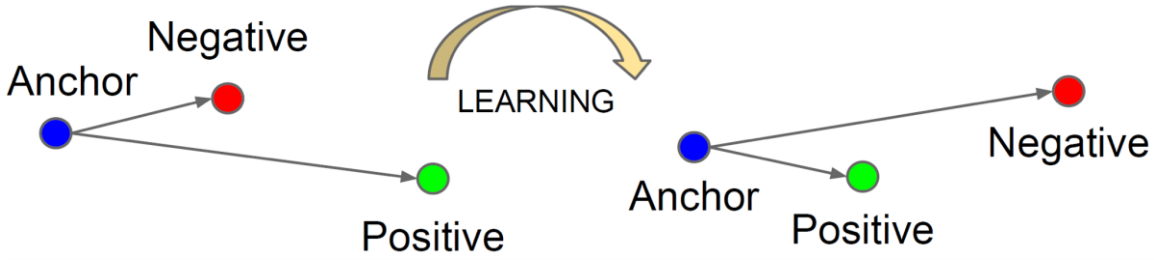

Figure 19 Illustrative example of what the Triplet Loss attempts to achieve in order to directly learn image embeddings. Notice that the distance between A and P is not directly paid attention to (FaceNet [64])

Triplet Loss can be expressed as:

$$L = max(d(A,P) - d(A,N) + margin, 0) \quad (18)$$

Where $d(A,P)$ is the distance between the anchor point and the positive point embedding. $d(A,N)$ is the distance between the anchor point and the negative point embedding. $margin$ is a hyperparameter that defines the minimum difference between $d(A,P)$and $d(A,N)$ to be considered for training. The margin is crucial because it prevents the network from trivial solutions and ensures that the positive and negative pairs are separated by a distance that is meaningful enough to differentiate between similar and dissimilar images effectively.

This loss function has shown great success in various applications, particularly in improving the accuracy of face recognition systems by learning an embedding space where distances directly correspond to a measure of face similarity.

The triplet loss encourages relative comparisons, which can lead to a more structured embedding space. It's more effective than contrastive loss in many cases by considering anchor-positive and anchor-negative relationships. On the other hand, it requires careful selection of triplets during training to avoid poor local minima. Its training can be slower due to the need to process triplets.

3.2.3 Other loss functions

3.2.3.1 Quadruplet Loss

Chen et al. [65] proposed Quadruplet loss to extend the concept of triplet loss by introducing a fourth element, aiming to push the negative example further from the anchor. It uses two margins and considers two negative examples to enhance discriminative power. Its advantages include greater flexibility and potential for discriminative feature learning, and improved model generalization. However, it is more complex and computationally expensive, and selecting meaningful quadruplets can be challenging.

3.2.3.2 Center Loss

Center loss is used alongside other loss functions to enhance the learning of discriminative features [66]. It reduces intra-class variations while keeping different classes separable. It includes effectively enhancing the compactness of the same class in the embedding space and is combinable with other loss functions for optimal inter-class separability and intra-class compactness. However, it requires additional hyperparameter tuning and can be sensitive to the initialization of center parameters.

3.2.3.3 Margin Loss

In [67], Margin loss was applied to ensure that similar pairs have a smaller distance than a certain margin while dissimilar pairs have a larger distance, making it effective for tasks involving fine-grained similarity measures. Its advantages include flexible margin settings leading to a discriminative embedding space and suitability for fine-grained similarity tasks. Disadvantages involve careful tuning of the margin parameter and sensitivity to outliers and noise in the data.

3.2.3.4 Binary Cross-Entropy Loss

Binary cross-entropy loss [46] is used for binary classification tasks within Siamese frameworks, measuring the difference between predicted and actual labels. Its advantages are direct interpretability in terms of probability and extensive support and optimization techniques. However, it may not capture the complexity of the embedding space for similarity learning tasks and is less effective for tasks requiring fine-grained distinctions between highly similar categories.

When implementing a Siamese neural network, the choice of loss function should be guided by the specific characteristics of the dataset and the task at hand. Experimentation and validation are key to determining the most effective loss function for a given application.

3.3 Dataset Pairs for Network Input

Sufficient data samples are still crucial for the adequate training of Siamese Networks model. This necessity stems from the models' requirement to learn from a diverse and extensive set of examples to achieve generalization over the problem space. The dataset provided for this thesis is not immune to the common challenge of data scarcity, which poses a significant obstacle that needs to be addressed meticulously.

Moreover, the focus on Siamese Neural Networks within this thesis introduces additional considerations concerning the structure and definition of the dataset. Specifically, it necessitates a clear delineation of several key components:

1. Classes: The different categories or types within the data that the model needs to learn to distinguish between. In the context of audio processing, these could be different sounds, speakers, musical instruments, or any other distinct auditory categories. In this thesis, they are Call Types of Southern Resident killer Whale in Pacific Ocean.
2. Audio Samples: The individual pieces of audio data that serve as the input to the network. Each sample is an instance that belongs to one of the predefined classes and contains the acoustic characteristics that the model must learn to recognize and differentiate.
3. Positive/Negative Pairs: For Siamese Neural Networks, which learn to gauge the similarity or dissimilarity between two inputs, the dataset must be organized into pairs of audio samples. A positive pair consists of two samples from the same class, indicating similarity, while a negative pair is composed of samples from different classes, denoting dissimilarity. The network learns from these pairings to discern the features that contribute to the audio samples being classified as similar or not.

Table 4 Data pairs created from S1, S2 and S10 as examples

Call type	**Audio Clip IDs**
S1	1, 2, 3, 4, 5, 6, 7, 8
S2	9, 10, 11, 12, 13, 14, 15
S10	16, 17, 18, 19, 20
Positive Pairs	(1,2), (1,3), (2,3) ….
Negative Pairs	(1, 9), (1, 16), (2, 9) ….

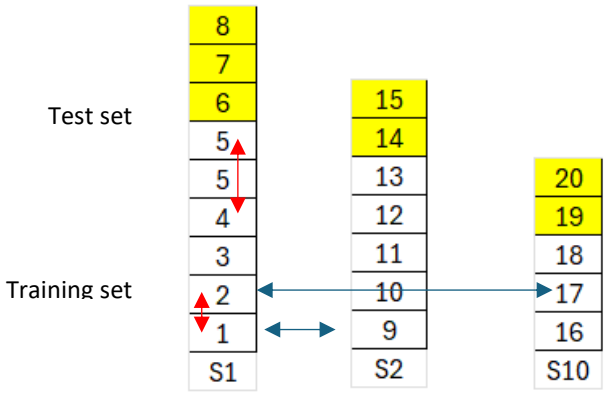

Figure 20 Call types with dozens of audio samples could generate thousands of positive/negative pairs

Table 4 and Figure 20 illustrate the process to generate audio data pairs. Despite the fact that each call type, such as S1, S2, and S10, contains no more than a few dozen samples, the number of samples experiences exponential growth when they are paired up in duplets. This process results in a dataset size that is fully capable of meeting the data volume requirements for computing similarities between samples of various categories using a Siamese network. This pairing strategy essentially multiplies the dataset's utility for the Siamese network, enabling it to learn from a vastly expanded set of comparisons. By examining the similarities and differences across these pairs, the network can effectively learn nuanced distinctions between categories.

Same to CNN training, 88% and 12% of data pairs is allocated to the Training and Testing. 20% out of 88% training data pairs are dedicated as a Validation Set for tuning and optimizing is highly effective. This setup facilitates iterative refinements based on validation feedback, promoting genuine model generalization. Such a distribution not only aids in safeguarding against overfitting but also ensures that the model exhibits robust performance on new, unseen data, thereby enhancing its utility and reliability in practical applications.

3.4 Feature Extraction for Siamese Networks

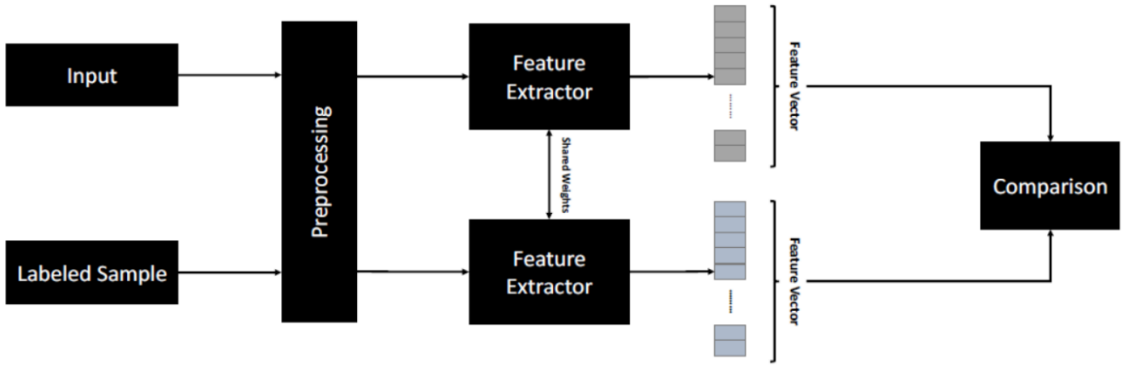

Figure 21 In Siamese Networks, input and the labeled sample go through three different stages: Preprocessing, Feature Extraction and Comparison [68]

In Siamese Networks, input and the labeled sample go through three different stages: Preprocessing, Feature Extraction and Comparison [68]. Audio processing, converting audio files into vector data that can be recognized by convolutional neural networks (CNN), is a critical step. Section 2.1.4.3 discusses the conversion of audio files into Spectrogram and Mel Spectrogram. This process is designed to transform audio into vector data recognizable by CNNs, facilitating feature extraction through CNNs. Siamese networks also employ a similar approach, but research papers have demonstrated that the MFCC (Mel-Frequency Cepstral Coefficients) method can extract MFCC vectors more effectively [69]. This is because the vector data format generated by the MFCC method is compacter and yields good recognition results. MFCC extracts the features of the audio signal through the following steps illustrated in Figure 21:

Spectrum acquisition of the signal:

Firstly, a Fourier transform (FFT) is performed on the audio signal $y(t)$ to obtain its spectrum. This step converts the signal from the time domain to the frequency domain.

Log power spectrum:

Then, the logarithm of the square of the spectrum's magnitude (i.e., the power spectrum) is taken. This step corresponds to the logarithmic operation part of the power cepstrum definition, that is, $log\ (|F\{y(t)\}|^2)$. The purpose of taking the logarithm is to simulate the non-linear perception of different loudness by the human ear and to convert the signal's multiplication operation into an addition operation, thereby simplifying the representation of the composite effects of the sound source and the vocal tract.

Mel filter bank processing:

Before calculating the traditional power cepstrum, the MFCC calculation maps the spectrum to the Mel frequency scale to simulate the human auditory perception system. This is unique to MFCC and does not appear in the traditional power cepstrum calculation steps.

Inverse Fourier Transform:

Finally, a discrete cosine transform (DCT) is applied to the logarithmic Mel power spectrum instead of a direct inverse Fourier transform (IFFT). The DCT operation here is equivalent to processing the logarithmic power spectrum, which helps reduce the correlation between features and extract useful acoustic features. This step can be considered a kind of "inverse transformation" of the logarithmic power spectrum, similar to the inverse Fourier transform (IFFT) of the power cepstrum, but more suitable for handling the spectrum under the Mel scale.

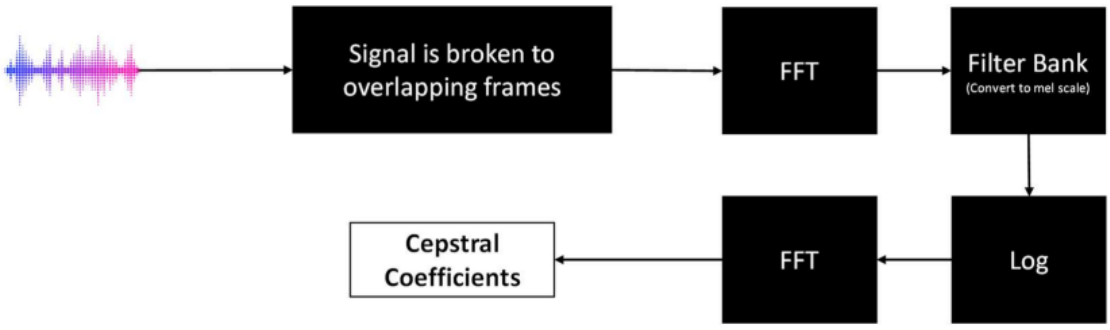

Figure 22 Steps in calculating Mel Frequency Cepstral Coefficients

In summary, although the traditional power cepstrum (obtained by applying a logarithmic operation to the spectrum of a signal and then performing an inverse Fourier transform) is not directly calculated in the calculation process of MFCC, MFCC adopts a similar processing flow — after performing a logarithmic operation on the spectrum, an "inverse transformation" is performed through Discrete Cosine Transform (DCT) instead of IFFT, and the spectrum is mapped to the Mel scale through the Mel filter bank [70]. The formula could be expressed as

$$MFCC[k] = \sum_{n=0}^{N-1} log\ (S[n]) cos\left[k(n + \frac{1}{2})\frac{\pi}{N}\right] \quad (19)$$

Where $MFCC[k]$ is the k^{th} Mel frequency cepstral coefficient, N is the number of Mel filters, which is the dimension of the log Mel power spectrum, $S[n]$ is the logarithmic value of the power spectrum output by the n^{th} filter after processing through the Mel filter bank, $log\ (S[n])$ is the log Mel power spectrum, The range of k typically goes from 0 to $N - 1$, but in practical applications, usually, only the first few coefficients (for example, the first 12 or 13) are chosen because they contain the most important sound characteristic information.

The core of this formula is mapping the log Mel power spectrum to the cepstral domain. The application of DCT has several important functions: it helps to decorrelate the coefficients of the Mel power spectrum (as these coefficients are often highly correlated) and effectively compresses the signal's information, concentrating the most important features into the lower cepstral coefficients. This makes MFCC very suitable for subsequent audio signal processing and pattern recognition tasks, such as voice recognition, speaker identification, and music information retrieval, etc.

In the experiments of this thesis, opting for MFCC (Mel-Frequency Cepstral Coefficients) encoding over Mel and magnitude spectrograms as input for a Siamese network is a strategic decision aimed at balancing the trade-off between capturing detailed spectral information and managing data volume efficiently. While Mel and magnitude spectrograms offer a richer spectral representation, they significantly increase data volume when paired with positive and negative samples for Siamese Network training. MFCC encoding provides a more compact yet sufficiently informative representation of audio signals, facilitating faster processing and more efficient learning within the Siamese network, thereby addressing computational resource constraints and optimization challenges inherent in handling high-dimensional data.

3.4.1 A comparative analysis between Mel-Frequency Cepstral Coefficients (MFCC) and the Mel Spectrogram

Mel-Frequency Cepstral Coefficients (MFCC) and Mel Spectrogram (mentioned in Section 2.1.4.3) are both pivotal techniques in audio signal processing. They serve as foundational tools in various applications, particularly in speech and audio analysis [71]. A comparative analysis of these two methods reveals distinct advantages and limitations inherent to each.

MFCCs are highly regarded for their efficacy in capturing the essential characteristics of speech sounds, particularly the resonant frequencies critical for phonetic differentiation [72]. The extraction process of MFCCs encompasses several stages: pre-emphasis, framing, windowing, Fast Fourier Transform (FFT), Mel filter bank processing, logarithmic transformation, and ultimately, the Discrete Cosine Transform (DCT) [73]. This comprehensive sequence culminates in a compact representation of the audio signal, significantly reducing its dimensionality. The resultant lower-dimensional feature set not only enhances computational efficiency but also ensures robustness and expeditious processing, attributes that render MFCCs particularly suitable for real-time speech recognition applications [74].

However, there are drawbacks to the MFCC methodology. A salient limitation is the loss of phase information during the transformation process, a factor that can be detrimental in certain audio-processing contexts. Additionally, MFCCs exhibit sensitivity to background noise, which can impair accuracy and reliability in noisy environments. This necessitates the implementation of meticulous pre-processing and noise reduction strategies to uphold performance standards in practical applications [75].

Conversely, Mel Spectrogram offers a more comprehensive representation of the audio signal by preserving detailed time and frequency domain information. This makes it exceptionally suitable for a broader spectrum of audio analysis tasks, such as music classification and sound event detection [76]. The richness of the temporal and spectral information encapsulated in the Mel Spectrogram facilitates enhanced visualization and interpretation of audio content.

Nevertheless, the Mel Spectrogram's higher dimensionality relative to MFCCs introduces increased computational complexity and demands greater resources for processing and storage. This can pose significant challenges in scenarios where computational efficiency and rapid processing are paramount. The substantial volume of data inherent in Mel Spectrograms requires the deployment of sophisticated algorithms for effective processing and analysis [75].

Due to the fact that the train dataset size for the CNN is much smaller than the size of the Data Pair training set for the Siamese Network, e.g. 1,078 audio samples could generate 260,085 audio pairs, the size of each individual record in training set after Feature Extraction is crucial. Although the Mel Spectrogram is richer in information on features compared to the MFCC vector, due to the limitation of hardware in this thesis, the author chose to use the Mel Spectrogram for Feature Extraction when training the CNN model and MFCC for Siamese Network. Additionally, the author plan to test the performance using MFCC as training data for the CNN to ensure the performance using the Mel Spectrogram not worse than MFCC.

3.5 Siamese Networks Structure

3.5.1 Basic structure of Siamese networks

A basic Siamese neural network comprises identical twin networks, also known as subnetworks, that share the same weights. These subnetworks take separate input samples, and the output feature vectors of the networks are combined by a selected distance layer. Figure 26 depicts the architecture of a classic Siamese network described by Bromley [59].

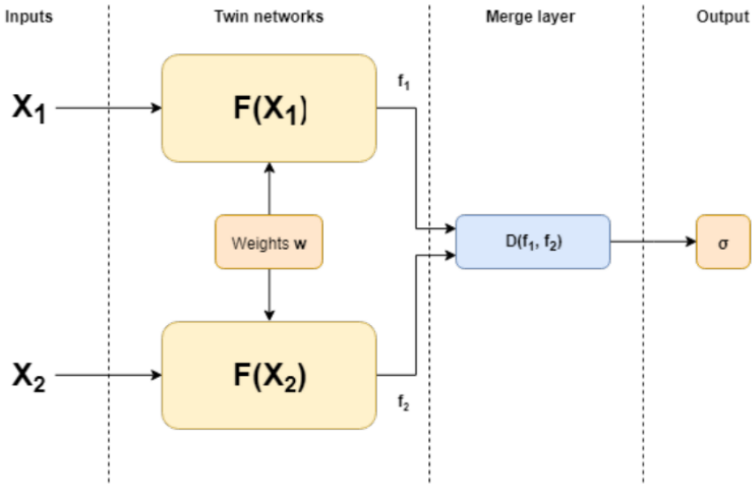

Figure 23 Layout of a Siamese Network [59]

In basic Siamese network architecture, while traditional models typically process 1-dimensional feature vectors through merge layers using distance functions, alternative implementations have emerged. These include concatenating feature vectors or employing multi-dimensional correlation models, allowing for the inclusion of additional layers—be it convolutional or fully connected beyond the merge point. Unlike classic Siamese networks that output directly after merging, these variants might use a logistic sigmoid function,

$$\sigma(z)=\frac{1}{1+e^{-z}} \qquad (20)$$

to map outputs to a (0,1) range. This mapping enables the output to serve as a probability or similarity score, offering a flexible way to quantify similarities between compared samples. This evolution in Siamese network design enhances their applicability across a wider range of tasks, especially those requiring detailed similarity assessments.

3.5.2 Transferring knowledge from Image Classification models

Despite the distinct differences between image and sound representations, highlighted during the preprocessing phase, research suggests that models initially devised for image classification, like those trained on ImageNet [76], can act as potent baseline networks for sound classification tasks [77]. This notion is supported by the shared representational structures between auditory and visual regions in the human brain, indicating that knowledge transfer across these domains could boost performance. Given ImageNet's role as a benchmark in image classification, it's fascinating to consider the potential of these high-performing models in tackling One/Few-Shot Learning challenges within audio classification.

In audio classification, deep learning models such as AlexNet [78], DenseNet201[79], InceptionV3 [80], Xception [81], VGG [82], and ResNet18 [83], originally designed for image classification, have been effectively repurposed for feature extraction and classification involving audio data. By converting audio signals into spectrograms or Mel spectrograms, these models can process "visualized" audio data, extracting pivotal features. AlexNet, even though it has a large number of parameters, showcases potential

in audio classification due to its strong feature extraction and adaptability. DenseNet201 enhances efficiency and reduces parameter count by interlinking each layer, proving particularly adept at capturing complex audio patterns. InceptionV3 and Xception leverage multi-scale and depthwise separable convolutions to effectively extract nuanced features. VGG's deep, straightforward architecture excels in audio classification, especially with Mel spectrograms, while ResNet18's residual connections overcome deep network training challenges, making it a viable option for audio classification.

Though these models have garnered success in image processing, adapting them for audio classification requires careful consideration of model size, computational demands, and training data volume, along with adjustments to suit audio data characteristics. With the right preprocessing and feature extraction approaches, these models' robust feature-learning capabilities can be harnessed for audio data analysis, demonstrating the flexibility and cross-domain potential of deep-learning models.

AudioSet [84] (Gemmeke et al. 2017) is one of the largest audio datasets accessible online, originating from the expansive video dataset known as YouTube-8M [85]. Upon its initial release, AudioSet provided a 128-dimensional embedding for each audio segment, encapsulated within a 0.96-second moving window. These embeddings were generated using a VGG-like classification model [81], laying the groundwork for the VGGish feature extractor (Figure 24). The feature extractor takes Mel Spectrograms of audio—prepared according to specified configurations—as input, producing a 128-dimensional feature vector. A comparison network then utilizes this vector for further analysis (Figure 25).

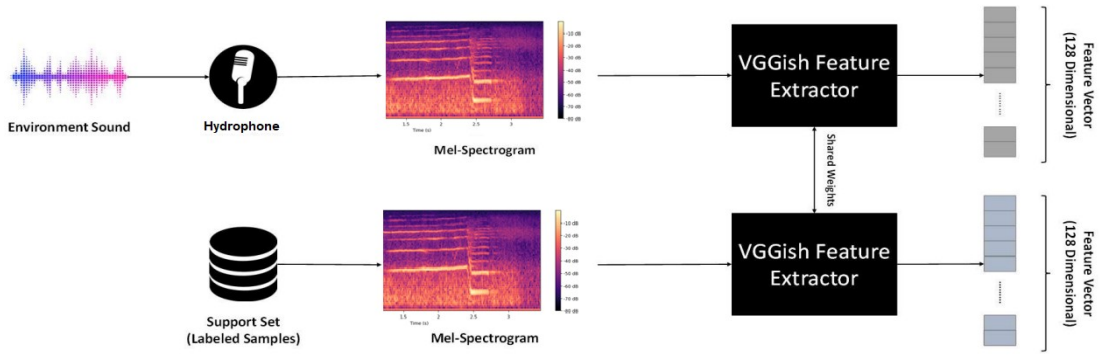

Figure 24 A representation of how the VGGish feature extractor would fit in the Siamese Network [81]

The preprocessing stage for the VGGish feature extractor involves generating a Mel Spectrogram from the audio, adhering to the parameters outlined in a specific configuration table. This table details aspects such as the number of frames, bands, sample rate, STFT (Short-Time Fourier Transform) window and hop lengths, and overall window size.

Originally inspired by the VGG image classification model, VGGish underwent modifications to adapt it for audio processing. This adaptation includes transforming the model's output into a 128-feature vector by replacing the image-focused Softmax layer with mechanisms suited for audio feature extraction. This innovative modification

highlights the flexibility of deep learning models, demonstrating their capability to transcend their initial domains and apply their powerful feature-learning frameworks across different types of data.

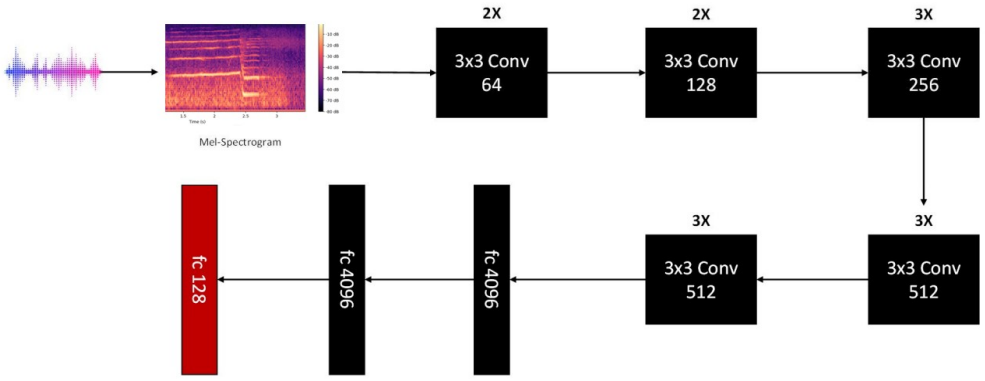

Figure 25 A VGGish network is derived from VGG-16 (16-layer VGG Model [81] originally designed for image classification). Black cells are the same as VGG, and the last red fully connected cell is to create a 128-dimensional feature vector

3.6 Similarity Comparison

In the fields of audio processing and speech recognition, feature vectors are numerical sequences used to represent the characteristics of audio signals. Once feature vectors are extracted from each audio sample, a method is required to compare these vectors to determine the similarity or difference between audio samples. **Cosine similarity** and **Euclidean distance** are two commonly used methods for this purpose.

Cosine similarity measures the angle between two vectors to determine their similarity in direction. Its values range from -1 to 1, where 1 indicates vectors in the exact same direction, 0 indicates vectors that are orthogonal (no similarity), and -1 indicates vectors in completely opposite directions [86]. Cosine similarity is calculated by taking the dot product of the two feature vectors and dividing it by the product of their magnitudes. This method is insensitive to the length of the vectors and primarily focuses on the difference in the angle between them, making it commonly used in text analysis, audio signal processing, and other fields.

Cosine similarity is more suitable for scenarios such as text matching and audio/video recommendations, where the similarity in direction is more important than the absolute distance. For example, in a music recommendation system, the similarity in users' preference types may be more important than specific numerical differences.

Euclidean distance, also known as L2 distance, measures the straight-line distance between two points in space [87]. In the context of feature vectors, it measures the actual distance between two points in a multidimensional space. Calculating the Euclidean distance between two vectors is relatively straightforward, involving the summation of

the squares of the differences between each pair of corresponding elements, followed by taking the square root. Unlike cosine similarity, Euclidean distance is very sensitive to the length and magnitude of the vectors and can be used to measure the absolute difference in values between two feature vectors. To address these limitations, normalization and feature scaling are implemented.

By incorporating these preprocessing steps into both the training and testing phases, the robustness and interpretability of the results in audio processing or speech recognition tasks are aimed to be enhanced.

3.6.1 Cosine Similarity

Cosine similarity measures the angle between two vectors to determine their similarity in direction [86]. Its values range from -1 to 1, where 1 indicates vectors in the exact same direction, 0 indicates vectors that are orthogonal (no similarity), and -1 indicates vectors in completely opposite directions. Cosine similarity is calculated by taking the dot product of the two feature vectors and dividing it by the product of their magnitudes. This method is insensitive to the length of the vectors and primarily focuses on the difference in the angle between them, making it commonly used in text analysis, audio signal processing, and other fields.

The probability of input and the labeled sample being in the same class can be calculated using cosine similarity as follows:

$$CosineSimilarity = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (21)$$

Where A_i and B_i are the components of feature vectors A (input) and B (labeled sample), respectively. n is the dimensionality of the feature vectors. After computing the cosine similarity between the input feature vector A and the labeled sample feature vector B, the result falls within the range $[-1,1]$, where 1 indicates that the feature vectors are identical; 0 indicates that the feature vectors are orthogonal, meaning there is no similarity; -1 indicates that the feature vectors are exact opposites. To transform the cosine similarity into a probability range of [0,1] for belonging to the same class, the following formula can be used:

$$Probability = \frac{CosineSimilarity+1}{2} \quad (22)$$

This transformation maps the cosine similarity score from the range $[-1,1]$ to the probability range [0,1], where 0 represents low probability (no similarity) and 1 represents high probability (high similarity).

3.6.2 Euclidean similarity measure

Euclidean distance, also known as L2 distance, measures the straight-line distance between two points in space [87]. In the context of feature vectors, it measures the actual distance between two points in a multidimensional space.

Because the audio embeddings are trained through some form of distance

function/measure I, a natural way to evaluate the embedded MFCC audio pairs will be to use that same distance function/measure used during training. As such, this thesis uses the Euclidean distance function D_E for training and will use the same metric for evaluating MFCC audio pairs. The Euclidean similarity metric S_E is described as follows:

$$S_E\big(f(X_1), f(X_2)\big) = \|f(X_1) - f(X_2)\|_2$$
$$= \sqrt{[f_1(X_1) - f_1(X_2)]^2 + [f_2(X_1) - f_2(X_2)]^2 + \cdots + [f_D(X_1) - f_D(X_2)]^2}$$
$$= \sqrt{\sum_{d=1}^{D}[f_d(X_1) - f_d(X_2)]^2} \qquad (23)$$

Thresholding the Euclidean similarity S_E by value τ is a critical step in evaluating the models. This process defines a similarity measure scheme S, which determines whether an image pair (X_1, X_2) should be classified as a similar pair (0) or a dissimilar pair (1):

$$f(x) = \begin{cases} 0, S_E\big(f(X_1), f(X_2)\big) < \tau \\ 1, S_E(f(X_1), f(X_2)) \geq \tau \end{cases} \qquad (24)$$

The choice between these two similar measurement methods depends on the specific requirements and the nature of the data. Sometimes, for optimal performance, both methods may be combined with other methods for more complex similarity measures.

3.7 Model Evaluation

This section covers the performance measures applied in the thesis and aims to cover their strengths and weaknesses.

3.7.1 Confusion matrix

Assume that S is a scheme used to determine whether two embeddings are equal or different, where $\hat{y}_{i,j} = S\big(f_i, f_j\big) = 0$ denotes a positive pair (two samples from the same call type), and $\hat{y}_{i,j} = S\big(f_i, f_j\big) = 1$ denotes a negative pair (two samples from the different call type. The confusion matrix is defined in Siamese Network testing as:

Given that the model produces binary results in the form of $\hat{y} = \{0,1\}$, a binary confusion matrix is instrumental. The structure of a binary confusion matrix is illustrated in Table 5.

Table 5 Confusion matrix for the problem of deciding if audio pairs belong to the same call type

		Predicted Label	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual Label	Positive (P)	True positive (TP)	False negative (FN)

	Negative (N)	False positive (FP)	True negative (TN)

In the confusion matrix:

True Positive (TP): Audio pairs correctly predicted as positive (same call type pair). given a similarity measure scheme between two audio clips $S(i,j)$ is defined as:

$$TP(S) = \{(i,j) \in P_{same} | \hat{y} = S(i,j) = 0\} \quad (25)$$

False Positive (FP): Audio pairs incorrectly predicted as positive when they are actually negative (different call type pair):

$$FP(S) = \{(i,j) \in P_{diff} | \hat{y} = S(i,j) = 0\} \quad (26)$$

False Negative (FN): Audio pairs incorrectly predicted as negative when they are actually positive.

$$FN(S) = \{(i,j) \in P_{same} | \hat{y} = S(i,j) = 1\} \quad (27)$$

True Negative (TN): Audio pairs correctly predicted as negative.

$$TN(S) = \{(i,j) \in P_{diff} | \hat{y} = S(i,j) = 1\} \quad (28)$$

3.7.2 Performance Measurement

Accuracy is a commonly used performance measure derived from the components of the confusion matrix. It is defined as follows:

$$Accuracy = \frac{TruePositives(TP)+TrueNegatives(TN)}{Total\ Population} = \frac{TP+TN}{TP+FN+FP+TN} \quad (29)$$

The accuracy represents the proportion of correctly classified instances out of all instances in the dataset. It provides an overall measure of the model's correctness in its predictions, regardless of class imbalance.

Accuracy tends to work well when the data is evenly distributed, i.e., there are about as many positive pairs as negative pairs. However, if there were 1 positive pair for every 99 negative pairs, a majority label classifier would get 99% accuracy. Accuracy as a performance measure for the tasks in this thesis will, therefore, be highly misleading, as the real goal of a good classifier is to be able to correctly predict positive pairs as well.

Precision is a measure of the proportion of correctly identified similar pairs out of all pairs predicted as similar call types. It is particularly useful when the focus is on the quality of the model's ability to predict similar pairs accurately. A high precision indicates that the model has a low rate of falsely labeling dissimilar pairs as similar, which is crucial in applications where the cost of false positives is high. Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, written as a formula:

$$Precision = \frac{Relevant\ retrieved\ pairs}{All\ retrieved\ pairs} = \frac{TP}{TP+FP} \quad (30)$$

Where TP (True Positives) is the number of correctly identified similar pairs. FP (False Positives) is the number of dissimilar pairs incorrectly classified as similar (Figure 26).

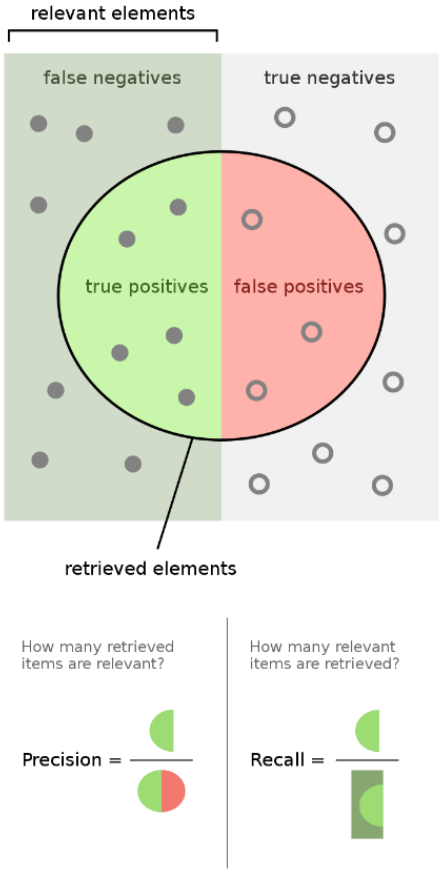

Figure 26 Precision and Recall [88]

Recall, also known as sensitivity or True Positive Rate (TPR), is a measure of the proportion of correctly identified similar pairs out of all actual similar pairs. It is more appropriate when the priority is on the quality of the model's ability to capture as many similar pairs as possible, even at the expense of falsely including some dissimilar pairs. A high recall indicates that the model has a low rate of missing similar pairs, which is important in applications where the cost of false negatives is high. The performance measure recall requires the number of TPs and FNs, and is defined as:

$$Recall(Sensitivity, TPR) = \frac{TP}{TP+FN} \quad (31)$$

Where TP (True Positives) is the number of correctly identified similar pairs. FN (False Negatives) is the number of actual similar pairs that were incorrectly classified as dissimilar. By computing the ratio of true positives to the sum of true positives and false negatives, the recall metric quantifies the model's ability to correctly capture similar pairs.

Specificity is often used as a metric in classification tasks, particularly in evaluating the performance of models that deal with binary or multi-class classification. When applied

to call type similarity calculation, specificity measures the ability of the model to correctly identify negative instances (i.e., correctly identifying that a certain call type is not similar).

$$Specificity = \frac{TN}{TN+FP} \quad (32)$$

High specificity is crucial in scenarios where false positives (incorrectly identifying dissimilar call types as similar) need to be minimized. This is particularly important in applications where mistakenly grouping dissimilar call types together could lead to negative outcomes, such as in customer service call routing or automated call analysis.

F1 score. A mixture of precision and recall yields a more comprehensive picture of a model's ability to predict/mistake audio pairs as similar. The F1-score arrives from these expressions and is defined as follows:

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \quad (33)$$

The F1-score is a balanced performance measure that combines precision and recall, making it robust for imbalanced datasets. However, it treats precision and recall equally, which may not always be ideal, especially in tasks where the consequences of false positives and false negatives differ significantly. In such cases, it's important to consider the specific objectives of the task and potentially prioritize either precision or recall accordingly.

A Receiver operating characteristic (ROC Curve) is a graphical plot used to evaluate the performance of a binary classifier model. Typically, the **true positive rate** (TPR, i.e., Recall), also known as Recall, is plotted on the Y axis, and the **false positive rate** (FPR) is plotted on the X axis. The top left corner of the plot represents the "ideal" point, with an FPR of zero and a TPR of one.

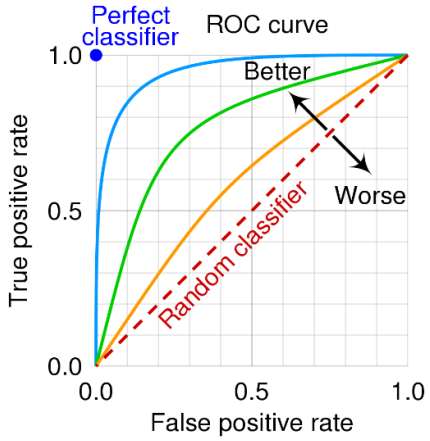

Figure 27 ROC Curve [89]

ROC curves are constructed by plotting TPR against FPR at various threshold settings for a similarity measure. They are commonly used in binary classification tasks to assess the performance of a classifier, particularly when determining an optimal threshold for a similarity measure.

The TPR represents the proportion of positive instances correctly identified, while the FPR represents the proportion of negative instances incorrectly identified as positive. By adjusting the threshold for the similarity measure and computing TPR and FPR at each setting, a series of data points are generated to construct the ROC curve.

The **Area Under the Curve (AUC)** of ROC curve quantifies the classifier's performance, with a higher AUC indicating better discrimination between positive and negative instances. A value of 1 represents perfect classification, while 0.5 represents random guessing.

Analyzing the ROC curve and AUC helps assess the trade-off between TPR and FPR at different threshold levels, aiding in determining the optimal threshold for the similarity measure and evaluating the overall performance of the classification model.

CHAPTER 4 EXPERIMENTS AND RESULT

This chapter will introduce the implementation, training phase, and results obtained using **Convolutional Neural Network** (CNN, as a comparable base line) and **Siamese Network** methods for Southern Resident Killer Whale Call Type audio similarity. It is divided into three sections: Experimental setup, Experiment for CNN, Experiment for Siamese Network, and Result Analysis.

To address the three questions outlined in Section 1.3, based on the theoretical research presented in Chapters 2 and 3, two sets of experiments are designed:

Section 4.2: Utilize a CNN model for multi-class Call Type Classification and evaluate the performance of small samples on the CNN model to answer Question 1 in section 1.3:

- Question 1: Is it feasible to develop deep learning models for SRKW call type classification using small-scale, low-quality samples through advanced data processing and augmentation techniques?

Section 4.3: Pairwise match samples of multi-class Call Types, converting the dataset into pairs with either similar or dissimilar categories. Transform the small dataset of multi-class Call Types into a large dataset of sample pairs. Utilize a Siamese network model to measure the similarity between sample pairs and reframe the multi-class Call Type Classification problem as a task of measuring the similarity between sample pairs, addressing Questions 2 and 3 in section 1.3:

- Question 2: Can small-scale samples be utilized to train models that compare the similarity between SRKW call type data pairs, as an alternative to employ multi-class classifiers for SRKW call type classification?
- Question 3: Can a similarity model effectively discern SRKW call types that were not presented in the training dataset?

4.1 Software and hardware

Software:

PyTorch version 3.11
CUDA toolkit 11.4
OS: Ubuntu Linux version 22.0.3.

Hardware:

CPU: Intel Xeon E5-2680 v4 @ 2.40GHz
RAM: 64GB DDR3 2133MHz
GPU: Nvidia Tesla P40 with 24GB VRAM

4.2 Classifying Multi-Call Type with CNN

The topic of this article is to measure the similarity between SRKW call audio segments. The model is set as the Siamese neural network. However, as motivated and argued in Chapter 2, there is a need to develop a CNN multi-classification model as a comparative benchmark for the Siamese neural network. Therefore, this section describes the process of training and testing a CNN multi-classification model using a five-fold increase in data generated by denoising and four types of data augmentation methods applied to limited existing data.

4.2.1 Data Preprocess

Currently, the SRKW data available to the author, as previously mentioned in Figure 2 and in Section 1.2, indicates that, except for Call types S36, S4d, S1d, and S2, which have more than 50 high-quality original sample audio clips per type. More than 23 call types have less than 5 samples per type. Additionally, there are 16 categories, such as S3 to S45 (denoted as Other S call types), which only have one sample.

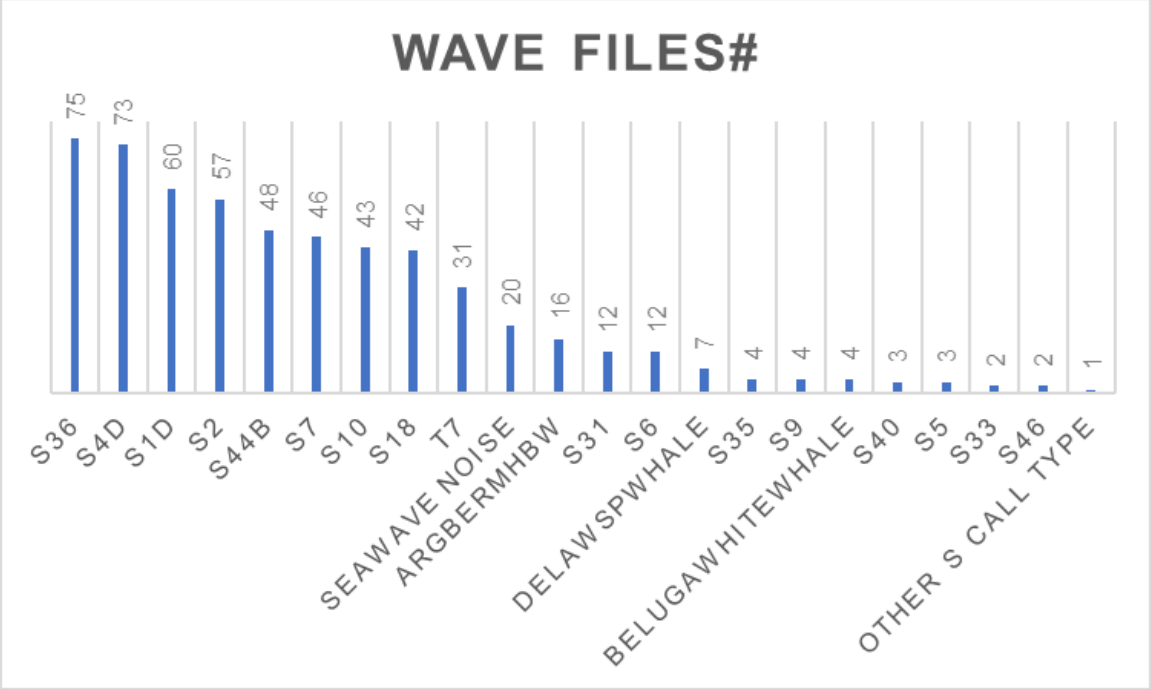

Figure 2 (repeatedly illustrated)

As discussed in Section 2.1, to provide a sufficient number of samples for training a CNN multi-class classification model, all the original audio files are first denoised. The denoising algorithm used is implemented using the **noisereduce** library[2], which employs

2 J. Karch, "noisereduce: Audio noise reduction in Python," GitHub repository: https://github.com/timsainb/noisereduce.

a frequency spectrum gating-based denoising algorithm. The method works by estimating the noise profile based on audio segments assumed to be pure noise. It then utilizes this profile to reduce the noise in the entire audio file.

After denoising the original audio files, the following four types of data augmentation are applied to both the denoised and original files:

1. **Pitch_shifting**: Changes the pitch of the audio by a random amount within a specified range.
2. **Random_shift**: Shifts the audio randomly in time within a specified range.
3. **Volume_scaling**: Scales the volume of the audio by a random factor within a specified range.
4. **Time_stretching**: Stretches or compresses the time axis of the audio by a specified rate.

This results in the generation of 8 additional audio samples from each original audio file and its denoised copy. Therefore, for each Call Type, at least 10 samples will be available.

4.2.2. Prepare Training and Testing Dataset

As shown in Figures 2, among the top 21 classes with the largest number of samples (S36-S46), to establish a performance comparison standard for CNN and Siamese Network, all call types with more than 3 samples, excluding S7, Beluga are selected. This is reserved for the Siamese Off-Training Class (refers to classes that never appeared in the training set and are used to test Question 3, i.e., the ability of the Siamese network to transfer learning to new call type). There are a total of 17 call types, and the "Wave file#" column shows the number of available accurate standard samples. After denoising these samples, 4x data augmentation is performed on both the denoised and non-denoised original samples and used all of them for the CNN training set and the Siamese Network training set. In the experiment, 88% of the samples (1,078 samples within 17 call types, including 20% validation dataset) to train the CNN model and 12% (140 samples within 14 call types) to test the CNN model. The data distribution for each Call type category is shown in Table 6.

Table 6 The training and test set data for CNN. Siamese Network will use the same data to create pairs.

Call Type	Non-Augmented Wave Files	Augmented Wave Files	Testing Wave Files	Remark
ArgBermHBW	12	60	4	SX: Southern Resident Killer Whale TX: Bigg‘s (Transient) Killer
DelawSpwale	10	50	10	
S10	19	48	10	
S18	16	50	6	
S1d	28	102	16	

S2	38	190	12	Whales in Northern Pacific
S31	14	46	7	
S35	4	13	5	ArgBermHBW: Bermuda Humpback Whales
S36	104	104	10	
S40	2	5	4	
S44b	18	90	7	DelawSpwale: Delaware Sperm Whale
S4d	48	239	16	
S5	3	10	4	Sea wave: Ocean background noise
S6	8	37	11	
S9	2	10	10	
Seawave	4	4	4	
T7	4	20	4	
Grand Total	334	1078	140	

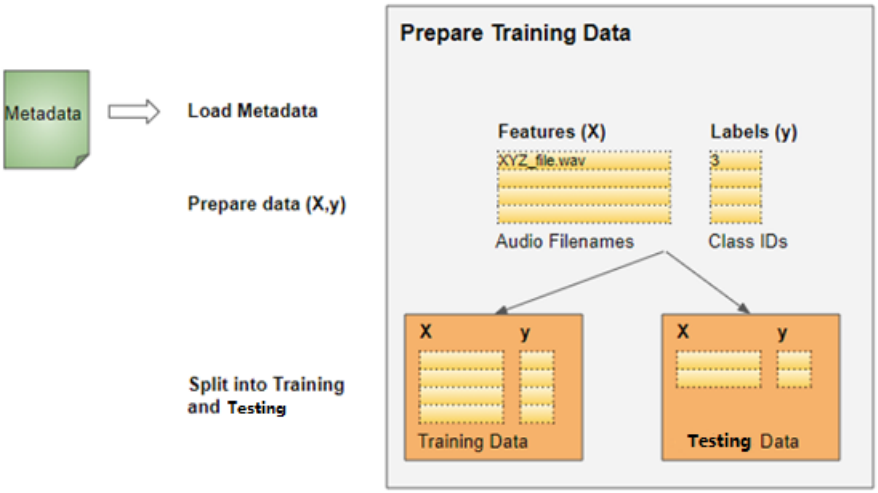

Figure 28 Split my data for training and testing [90]

In data processing part, following functions are done in python scripts:

1. **Imports Required Libraries:**

pandas and numpy for data manipulation, audio processing libraries like librosa, noisereduce to de-noise, generate mel spectrogram, machine learning (sklearn), deep learning (torch), and progress tracking (tqdm), etc.

2. Creates a DataFrame with Metadata:

Creates a DataFrame (data_df) to store metadata such as filenames, target classes, and file paths, only audio files with a duration of 4 seconds or less are included.

3. Defines a Function to Compute Log-Mel Spectrograms:

The audio files are normalized firstly. The mel spectrogram is then calculated using librosa.feature.melspectrogram, followed by converting the mel spectrogram to a logarithmic scale with librosa.power_to_db..

4. Processes and Pads Audio Features:

When training a neural network, it is crucial to maintain a consistent feature size for each sample. However, due to the varying lengths of audio files, the time dimension (number of frames) in the directly extracted log-mel spectrograms will differ. Therefore, each audio file is converted into a log-mel spectrogram, and delta features are stacked. Subsequently, each feature array is either padded or trimmed to ensure a uniform shape.

5. Store the vectors in an HDF5 database.

The training and testing datasets are in separate H5 databases.

4.2.3. Create Batch and DataLoader

When training begins, one batch (with the batch size typically being a power of 8, such as 128, 256, etc.) of input features, containing a list of audio file names, will be randomly fetched by the Data Loader, and the pre-processing audio transforms will be applied to each audio file. It will also fetch a batch of the corresponding target Labels containing the class IDs. Thus, it will output one batch of training data at a time, which can directly be fed as input to deep learning model.

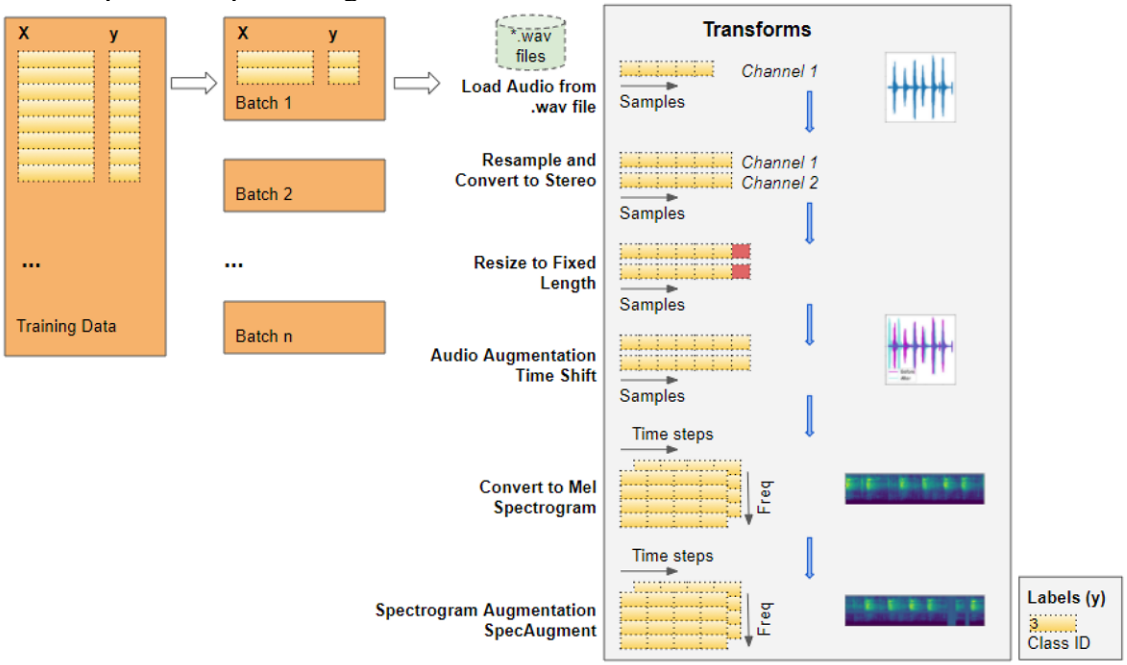

Figure 29 Data Loader applies transforms and prepares one batch of data at a time [90]

The data transformation process begins with loading the audio files into a Numpy array, structured as (num_channels, num_samples). Most audio is sampled at 44.1kHz, with a duration of approximately four seconds, resulting in 176,400 samples for single-channel audio ((1, 176,400)) and 192,000 samples for two-channel audio sampled at 48kHz ((2, 192,000)). Given the variations in channels and sampling rates, the audio is resampled to a standard 44.1kHz and two channels. The duration is also standardized to four seconds, ensuring uniform array dimensions of (2, 176,400) for all audio clips.

Data augmentation through time shifting is then applied, randomly moving each audio sample forward or backward without changing its shape. The augmented audio is converted to a Mel Spectrogram with dimensions (num_channels, Mel freq_bands, time_steps) = (2, 60, 173). Subsequently, SpecAugment is used to apply random time and frequency masking, leaving the shape of the Mel Spectrogram unchanged.

Each batch of data, selected randomly for each training epoch, consists of two tensors: one containing the Mel Spectrograms as the feature data (X) and the other containing the numeric class IDs as the target labels (y). The batch typically has the dimensions (batch_sz, num_channels, Mel freq_bands, time_steps) for the features and (batch_sz) for the labels. For instance, a batch might have the shape (16, 2, 64, 344) for X and (16) for y.

The transformed data is now ready to be input into the deep learning model.

4.2.4. CNN Model Definition

The data processing steps that were just completed represent the most unique aspects of the audio classification problem. The model and training procedure are like what is commonly used in a standard image classification problem and are not specific to audio deep learning.

Since the audio training data is converted to spectrogram images, a CNN classification architecture could process them. Figure 30 illustrates a typical four convolutional blocks that generate feature maps. That data is then reshaped into required format so it can be input into the linear classifier layer, which finally outputs the predictions for the 17 call types (S1, S2, …).

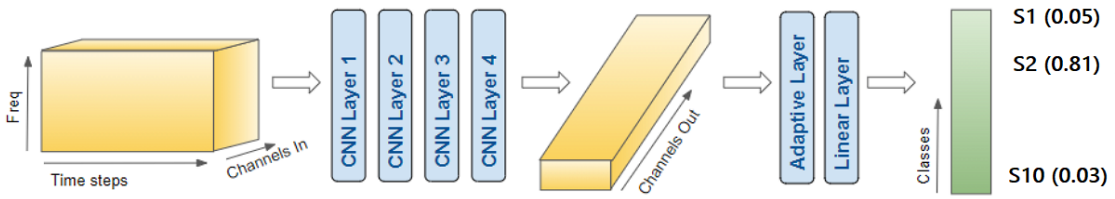

Figure 30 The model takes a batch of pre-processed data and outputs class predictions [90]

Here are details about how CNN model processes a batch of data:

- A batch of mel spectrograms is input to the model with shape (batch_size, num_channels, Mel freq_bands, time_steps), i.e., (64, 2, 60, 173).
- Each CNN layer applies its filters to step up the spectrogram depth, i.e., the number of channels. The spectrogram width and height are reduced as the kernels and strides are applied. Finally, after passing through the four CNN layers, the output feature maps are obtained, i.e., (64, 64, 4, 32).
- This gets flattened to a shape of (64, 8192) and then input to the Linear layer.
- The Linear layer outputs one prediction score per class, i.e., (64, 17) (17 call types in the training set).

4.2.5. Training

The final CNN model is illustrated in Figure 31.

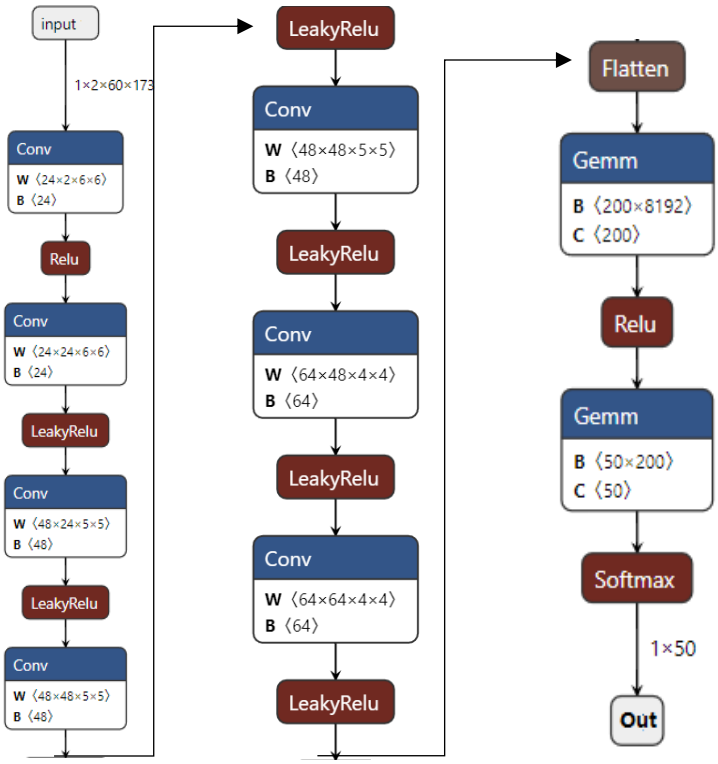

Figure 31 CNN for 17 Call type classification structure

The functions for the optimizer, loss, and scheduler were defined to dynamically adjust the learning rate as training progressed, facilitating faster convergence within fewer epochs. The model was trained for 500 epochs, with each iteration processing a batch of data. Loss and accuracy metrics, which measure the percentage of correct predictions, were tracked throughout the training process.

As shown in the results of the code execution above, in each Epoch, both Training and Testing Loss steadily decrease. Meanwhile, the corresponding model's prediction accuracy on the Test dataset, consisting of 140 samples, stabilizes at around 97.8%. The trends of Training and Testing Loss, as well as the prediction Accuracy over Epochs, are illustrated in Figure 32.

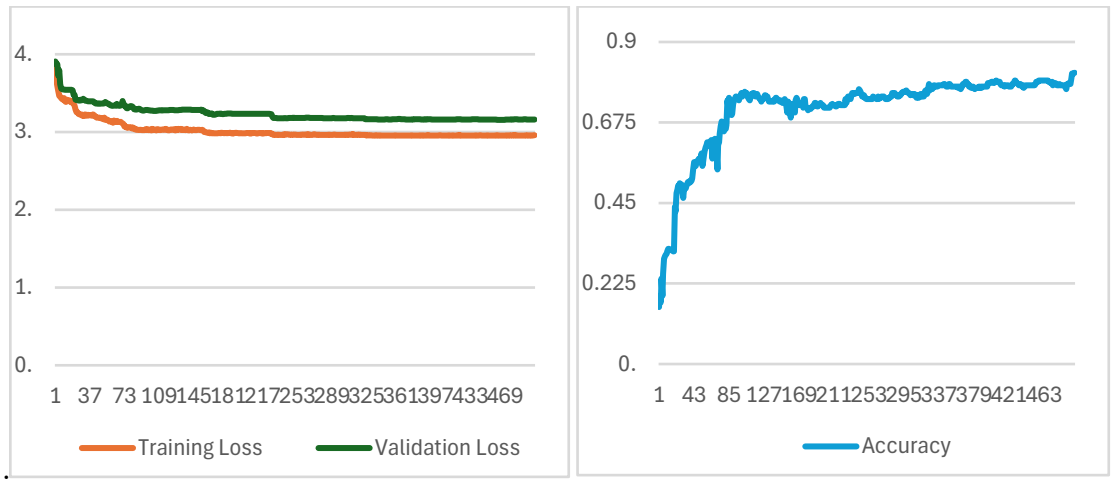

Figure 32 The Loss and Accuracy Trend over training Epochs. X-axis displays the number of epoch and Y-axis display the unit of loss (left figure) or Accuracy during training (right figure)

4.3 Siamese network

4.3.1. Audio Preparation

To facilitate comparison with the CNN model, the training, validation and test samples of SRKW Call types used in Section 4.2 were utilized for training the Siamese network in this section. As mentioned in Figure 25, e.g. 10 call types, each containing 2 samples, following data pairs could be generated:

- 10 similar pairs, where each pair comprises samples from the same call type.
- 45 dissimilar pairs, representing comparisons between samples from different call types.

The 10 similar pairs consist of audio pairs from the same call type. The quantity of dissimilar pairs can be calculated using the combination formula: $C(n,k) = \frac{n!}{k!(n-k)!}$. For example, $C(10,2) = \frac{10!}{2!(10-2)!} = 45$. Through this approach, even if each call type has only one sample, a large number of similar and dissimilar samples can be created for training Siamese network model by denoising, applying four types of data augmentation, pairwise comparison. Table 7 displays training and testing data pairs for Siamese Model training, validation and testing.

Table 7 Pair data set for Siamese network training and testing

Call Type	Augmented wave Files for Training	Augmented Wave Files Generated Pairs for Training		Wave Files for Testing	Wave Files Generated Pairs for Testing		Remark
		Similar (Label=0)	Dissimilar (Label=1)		Similar	Dissimilar	
ArgBermHBW	80	3,160	28,480	4	6	280	**SX:** Southern Resident Killer Whale **TX:** Bigg's (Transient) Killer Whales in Northern Pacific **ArgBermHBW:** Bermuda Humpback Whales **DelawSpwale:** Delaware Sperm Whale **Sea wave:** Ocean background noise
DelawSpwale	36	630	22,068	10	45	1,160	
S10	41	820	17,876	10	45	740	
S18	40	780	22,120	6	15	600	
S1d	60	1,770	6,540	16	120	304	
S2	55	1,485	9,295	12	66	420	
S31	46	1,035	14,260	7	21	441	
S35	16	120	7,632	5	10	420	
S36	71	2,485	46,789	10	45	1,300	
S40	10	45	6,490	4	6	504	
S44b	46	1,035	2,898	7	21	84	
S4d	86	3,655	19,264	16	120	752	
S5	12	66	372	4	6	16	
S6	60	1,770	29,580	11	55	979	
S9	20	190	11,860	10	45	1,060	
Seawave	20	190	860	4	6	32	
T7	31	465		4	6		
Grand Total	730	**19,701**	**246,384**	**140**	**638**	**9,092**	

To improve the generalization ability of the model since the predicted call types not present in the training set might be similar to the calls of certain marine mammals, this paper has selected Bigg's (Transient) Killer Whales (T7) in the Northern Pacific, Bermuda Humpback Whales (ArgBermHBW), and Delaware Sperm Whales (DelawSpWhale) to be added to the training set. Additionally, to prevent the background noise of Seawave Noise (Seawave Noise) from being mistakenly identified as a specific call type of SRKW, Seawave noise has been added as a separate category of sound to the training set.

An inherent advantage of the Siamese network, in contrast to the CNN multi-class model, lies in its capability to infer the similarity of call type even absent from the training dataset. This attribute addresses Question 3 in section 1.4. To address this query, the performance of 14 call types not present in the training dataset, as delineated in Table 7, was evaluated. Column all pairs in table 4-3 refer to the total number of sample pairs, including those from the same call type (assigned label '0', indicating both items in the pair are from the same call type, e.g., both from S13) and those from different call types (assigned label '1', indicating items in the pair are from different call types, e.g., from S13 and S7, as indicated in the column 'Similar pairs').

Table 8 Original audio clip, denoised and augmented files and formed pairs of 14 off-train call types

Call Type	Wave Files for Testing	Wave Files Generated Pairs for Testing		Remark
		Similar	Dissimilar	
LFinPilWhale	8	28	1,328	LFinPilWhale: Long-Finned Pilot Whale NorRigWhale: Northern Right Whale SFPilWhale: Short-Finned (Pacific) Pilot Whale SouRigWhale: Southern Right Whale
NorRigWhale	14	91	980	
S13	16	45	600	
S14	10	45	1,360	
S16	10	45	1,460	
S17	10	45	1,840	
S19	10	45	1,740	
S3	6	15	504	
S30	10	45	400	
S33	20	190	400	
S46	19	190		
S7	46	1,035	4,140	
SFPilWhale	10	45	500	
SouRigWhale	10	45	1,560	
Grand Total	199	**1,909**	**16,812**	

4.3.2. Audio Pre-processing and saving into H5 database

Training data containing audio file paths cannot be directly input into a model. Audio data must be loaded from the files and process it into a format expected by the model. Since loading the Mel Spectrogram into GPU memory occupies too much space, this section will create a dataset of Mel Frequency Cepstral Coefficients (MFCC) features from audio files, then save this data to an H5 database for subsequent model training. Here is the data processing workflow:

1. **Computing MFCC:** This function takes an audio signal and its sample rate as inputs and computes MFCC features. MFCCs are calculated using a specified number of coefficients (n_mfcc), FFT window length (n_fft), and hop length between frames (hop_length). It then converts these features to a decibel scale to normalize amplitude variations. The function audio2mfcc loads an audio file from a specified file path, computes its MFCC features, and then pads or truncates them to a fixed length (max_pad_len). This ensures that all feature arrays have the same shape, which is crucial for training machine learning models. In the experiment, the shape of mfcc vector will be (20, 432).
2. **Training Data Generation:** This function generates pairs of MFCC features from audio files located in subdirectories of a given base path. It organizes the audio files into pairs where each pair can belong to the same class (label 0) or different classes (label 1). This is used for tasks like similarity learning or classification. The function also keeps track of the filenames of the audio files in each pair for reference.
3. **Standardize** the datasets with the mean and standard deviation calculated from the training dataset.

4. **Data Saving to H5 database:** After generating the pairs and labels, this function saves the MFCC feature data into an H5 file. This file format is efficient for storing large datasets and supports incremental data loading, which is beneficial for training machine learning models with large datasets.

Operational Workflow: Upon calling save_to_h5 to write the data, and considering the substantial memory requirements (as noted, at least 32GB of RAM is needed), the script is optimized for performance by handling data in batches and cleaning up memory frequently. The script uses tqdm for progress tracking, which helps in monitoring long-running data processing tasks, especially when dealing with large datasets.
This workflow is optimized for applications in audio analysis, which can identify, classify, or compare sounds based on their MFCC features. It is particularly useful in fields like digital signal processing, speech recognition, and ambient sound analysis.
Please refer to https://github.com/jackzhang2000/SRKW_CallType to obtain the details of data processing code.

4.3.3. Model architecture

Figure 33 represents the architecture of a Siamese network implemented in my experiment for comparing two input vectors of SRKW call audio clips.

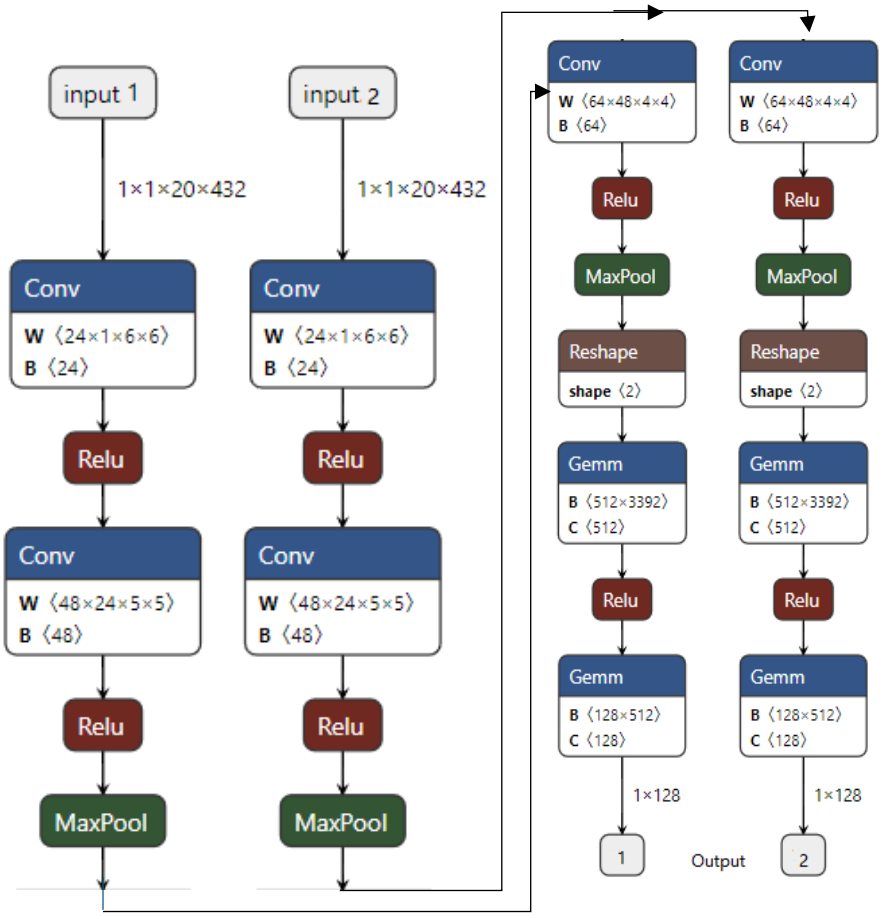

Figure 33 The architecture of Siamese network in the experiment

Each input, presumably a pre-processed MFCC (Mel Frequency Cepstral Coefficients) feature map from call samples, passes through identical branches of the network, ensuring that the same transformations are applied to both inputs. The network starts with a convolutional layer that uses 24 filters of size 1x6x6, introducing the first level of spatial feature extraction. This layer is followed by a ReLU activation function, which adds non-linearity to the processing flow, allowing the network to learn more complex patterns.

Subsequent layers include more convolutional layers with increasing numbers of filters (48 and then 64), each followed by a ReLU activation. These layers are designed to progressively extract higher-level features from the input. Max pooling layers follow some of the convolutional layers to reduce the spatial dimensions of the feature maps, thereby decreasing the computational complexity and controlling overfitting by abstracting the features further.

After the final pooling step, the output is flattened and fed into a series of fully connected layers. The first has 512 units, and the output from this layer is again processed by a ReLU activation function. This layer is crucial as it begins to consolidate the learned features into a form that represents the input in a more abstract, compressed space. The next fully connected layer reduces the dimensionality further to 128 units, preparing the output for the final comparison.

The outputs from each branch of the Siamese network are vectors of size 1x128, which are then typically used to compute a similarity measure through a contrastive loss function. This function will calculate the distance between the two vectors, facilitating the determination of whether the inputs are similar or not based on learned metrics.

This architecture highlights the power of CNNs in feature extraction and the effectiveness of Siamese networks in learning nuanced differences and similarities between paired inputs, leveraging shared weights to maintain symmetry in learning. Such setups are integral in applications requiring precise and reliable comparison metrics, further enhanced by the network's ability to learn from relative comparisons rather than absolute feature sets.

4.3.4. The training loop for batches of pairs

The training loop calls for batches of SRKW call pairs, involving following operations:

4.3.4.1 Setting Up the Loss Function and Optimizer

Contrastive Loss function is designed for learning the similarities or differences between input pairs, commonly used in Siamese networks or triplet networks. The Adam optimizer is used to optimize model parameters with a specified learning rate e.g. 0.001 and weight decay e.g. 0.0001 to control learning progress and prevent overfitting.

4.3.4.2 Training Loop Setup

The total number of training epochs is set to 200, providing ample opportunity for the model to converge and optimize its performance. However, to prevent unnecessary overfitting or excessive training, early stopping is implemented. This mechanism relies on a patience parameter, which defines the tolerance for stagnation in improvement. Specifically, if there is no observed improvement in the validation loss over a period of five consecutive epochs, the training process is halted prematurely. This helps ensure that the model does not continue to train when further improvements are unlikely, thus saving time and computational resources.

This method, driven by validation performance, was observed to stop training early at epoch 46 in my experiment, as shown in Figures 36, due to the validation loss failing to improve for five consecutive epochs.

In the experience, the training process stopped early at epoch 46, as Figure 34, due to the loss not improving for 5 consecutive epochs.

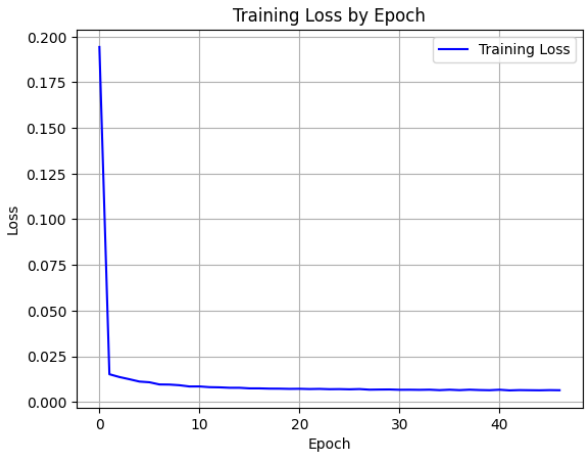

Figure 34 training stopped at epoch 46 due to the loss not improving for 5 consecutive epochs

During training, a variable referred to as best_loss is maintained to track the lowest validation loss encountered. This value is used as a benchmark for early stopping decisions. When the model achieves a validation loss lower than the current best_loss, this value is updated, signaling that the model's performance has improved. If, however, no improvement is observed within the defined patience period, the training is terminated to avoid over-optimization on the training data, thus preserving the model's generalization capabilities.

4.3.4.3 Inside the Epoch Loop

During training, batches of paired input data and target labels are loaded from the data loader. The inputs and targets are moved to the configured device (GPU or CPU). Inputs

are reshaped by adding a channel dimension, as convolutional layers require a four-dimensional input (batch size, number of channels, height, width), with the input dimensions in this experiment being (128, 2, 20, 432).

4.3.4.4 Loss Calculation and Optimization

In the process of loss calculation and optimization, the gradients from the previous step are cleared using the zero_grad method, ensuring that the model is prepared for a new cycle of backpropagation. Mixed precision training is enabled through the autocast functionality, which enhances training efficiency while simultaneously reducing memory usage. To calculate the loss, the outputs from the model are passed to the designated loss function, allowing the loss for the current batch to be computed. Following this, backpropagation is employed to compute the gradients with respect to the model parameters, based on the calculated loss. Gradients are then scaled using GradScaler before the model weights are updated, allowing for more stable training, especially when using lower precision arithmetic.

At the end of each epoch, the validation set is utilized to evaluate the model's performance. The validation set outputs are passed to the same loss function as used during training, and the validation loss is computed. This validation process is critical for assessing the model's generalization ability to unseen data. By monitoring the validation loss throughout training, early signs of overfitting or underfitting can be detected, enabling necessary adjustments to be made to the model or its hyperparameters, ensuring that the training process does not solely optimize for the training data.

Finally, once the training is complete, the trained model parameters are saved, and the training dataset is cleared from memory to free up resources, ensuring that the system is ready for further tasks or evaluations.

4.4 Model Result and Discussion

Referring to section 1.3, there are three research questions to be answered. The experiment 4.2 focuses on Question 1, which is restated below:

4.4.1 Question 1: Is it feasible to develop deep learning models for SRKW call type classification using small-scale, low-quality samples through advanced data processing and augmentation techniques?

According to the measurement defined in section 3.7, Two CNN classification models with both **Non-Augmented Wave Files** and **Augmented Wave Files as Table 5 in Section 4.2.2**, and then test and compare their performance with same Testing Wave Files in table 6.

Table 6: CNN Model Train data and Testing Data (repeated)

Call Type	Non-Augmented Wave Files	Augmented Wave Files	Testing Wave Files	Remark

ArgBermHBW	12	60	4	SX: Southern Resident Killer Whale TX: Bigg's (Transient) Killer Whales in Northern Pacific ArgBermHBW: Bermuda Humpback Whales DelawSpwale: Delaware Sperm Whale Sea wave: Ocean background noise
DelawSpwale	10	50	10	
S10	19	48	10	
S18	16	50	6	
S1d	28	102	16	
S2	38	190	12	
S31	14	46	7	
S35	4	13	5	
S36	104	104	10	
S40	2	5	4	
S44b	18	90	7	
S4d	48	239	16	
S5	3	10	4	
S6	8	37	11	
S9	2	10	10	
Seawave	4	4	4	
T7	4	20	4	
Grand Total	334	1078	140	

4.4.2 Result of Question 1: Classification accuracies vary by different Call types by CNN

As described in table 6, two multi-class CNN models are trained using raw data of 17 call types (334 samples, after denoising) and augmented data (1,078 samples). Both models' classification performance is evaluated on a test set of 140 samples using the confusion matrix as well as the number of accurately recognized samples (True Positive) and the percentage of true positives relative to the actual samples in each call type. True Positive Rate%, also called Recall% is defined as:

$TP(S) = \{(i,j) \in P_{same} | \hat{y} = S(i,j) = 0\}$, i.e. $Recall(Sensitivity, TPR) = \frac{TP}{TP+FN}$

When train the CNN model on augmented data, which feature engineering approach need to be determined. The testing results of CNN model on Mel Spectrogram and MFCC encoded data are compared as table 9.

Table 9 The testing performance of Mel Spectrogram featured training data vs MFCC featured training data

Model	Accuracy	Recall (Sensitivity)	Specificity	F1	Precision	Training Wave#	TN#	TP#	FN#	FP#
CNN with Mel Mel Spectrogram	97.8%	81.4%	98.8%	81.4%	81.4%	**1,078**	2,214	114	26	26
CNN with MFCC	96.6%	69.3%	98.2%	69.3%	69.3%	**1,078**	2337	97	43	43

The results indicated that the CNN model performed better on all performance metrics compared Mel spectrogram to MFCC-encoded data. Although the accuracy (97.8%) and specificity (98.8%) of the Mel spectrogram slightly outperformed those of MFCC (96.6% and 98.2%, respectively), the recall, precision, and F1 score (81.4%) of the Mel spectrogram CNN model were significantly higher than those of MFCC (69.3%). Consequently, Mel spectrogram data is applied to train CNN model.

In the CNN model, an interesting phenomenon was observed: while the overall recall, precision, and F1 score of the total samples were equal, these metrics varied when broken down by each call type. Upon examination, it was found that for each call type, the false negatives (FN) were calculated as the actual positive (P) count minus the true positives (TP), and the false positives (FP) were calculated as the predicted positive (P) count minus the true positives (TP). Although the actual positive count did not equal the predicted positive count for each call type, the sum of predicted positives across all call types equaled the sum of actual positives. This led to the equality:

$$\frac{\sum(TP)}{\sum(TP+FN)} = \frac{\sum(TP)}{\sum(TP+FP)} \quad (10)$$

Thus, the overall recall equaled the overall precision. Since the F1 score is derived directly from recall and precision, the overall F1 score also equaled the recall and precision. Nonetheless, metrics of recall, F1 score, and accuracy could be still used to evaluate the performance of the CNN model. Table 11 displays the performance of Non-Augmented Data (334 samples in table 6) trained CNN Model on FSL Testing Data (140 samples in table 6)

Table 10 The testing performance of Non-Augmented Data (334 samples in table 6) trained CNN Model on FSL Testing Data (140 samples in table 6)

Actual Call Type	Predicted Call Type																	Recall%
	S36	S40	DelawSpwale	S9	S18	S6	S35	S10	ArgBermHBW	S31	S4d	S2	S1d	S44b	Seawave	S5	T7	66.4%
S36	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	90.0%
S40	0.0	0.8	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	75.0%
DelawSpwale	0.3	0.0	0.6	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	60.0%
S9	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.2	0.5	0.0	0.0	0.0	0.0	0.0%
S18	0.2	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	50.0%
S6	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S35	0.0	0.0	0.4	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	0.0%
S10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.6	0.1	0.1	0.0	0.0	0.0	20.0%
ArgBermHBW	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S4d	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	93.8%
S2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	66.7%
S1d	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	0.3	0.0	0.0	0.0	75.0%
S44b	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.7	0.0	0.0	0.0	71.4%
Seawave	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	100.0%
S5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0%
T7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	100.0%

VS

Table 11 the performance of augmented data (1,078 samples in table 6) trained CNN Mode on FSL augmented testing data (140 samples in table 6)

Actual Call Type	Predicted Call Type																	Recall%
	S36	S40	DelawSpwale	S9	S18	S6	S35	S10	ArgBermHBW	S31	S4d	S2	S1d	S44b	Seawave	S5	T7	81.4%
S36	0.9	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	90.0%
S40	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
DelawSpwale	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S9	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S18	0.2	0.0	0.0	0.0	0.7	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	66.7%
S6	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S35	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	0.0	50.0%
ArgBermHBW	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S4d	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0%
S2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0	0.0	0.0	0.0	0.0	91.7%
S1d	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.6	0.2	0.0	0.0	0.0	62.5%
S44b	0.1	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.6	0.0	0.0	0.0	57.1%
Seawave	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	0.0	0.5	0.0%
S5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0%
T7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	100.0%

The performance of two models was analyzed using their confusion matrices. In the first confusion matrix (table 10), the model accurately predicted classes such as S36, S6, Bermuda Humpback Whales, S31, S4, Sea Wave, and T7, but showed lower accuracy for other classes. Notably, there was confusion in predicting Delaware Sperm Whale, S9, S18, S35, S10, S2, and S5, likely due to insufficient training samples. In the second confusion matrix (table 11), the model correctly predicted most classes, though prediction accuracy was lower for S18, S10, S44, Sea Wave, and S5, indicating some confusion.

Based on these results, it can be observed that the second model performs better across more call types, though some confusion is exhibited in predicting call types S5, S10, and S44. Consequently, it can be concluded that training a CNN multi-class classification model on a small annotated dataset significantly impacts classification performance. An overall recall rate of only 66.4% was achieved by the model trained on the 334 non-augmented training set. While data augmentation improves the accuracy of the CNN model to 97.8%, the overall recall rate remains around 81.4%.

As demonstrated by the two experiments above, it is evident that achieving good recognition performance with CNN models becomes challenging when numerous call type categories have very few annotated samples (fewer than 20 annotated audio clips per call type) and poor data quality. In the research presented in chapters 3, a method of pairwise matching within and between categories was adopted, using Siamese networks to identify the similarity of sample pairs for category recognition. The following experiments present the test results on the three datasets used in experiment 1. The second experiment, based on the Siamese network, is addressed in next section.

4.4.3 Question 2: Can small-scale samples be utilized to train models that compare the similarity between SRKW call type data pairs, as an alternative to employ multi-class classifiers for SRKW call type classification?

According to the method in section 4.2.2, similar and dissimilar audio pairs were constructed from augmentation data of 17 call types, the layout of training data is illustrated in table 12.

Table 12 The layout of input training data for Siamese Network Model

Filename1	Filename2	Class_a	Class_b	Actual Label
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (12).wav	S36	S36	0
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_S36JASCOAMARHYDROPHONE2402_20150929T075616.320Z_89.872.wav	S36	S36	0
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (7).wav	S36	S36	0
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S40/FO-S40-R-b_augmented_3.wav	S36	S40	1
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S40/denoised_FO-S40-R-b.wav	S36	S40	1

When Wave File1 and Wave File2 are from the same call type, the actual label is 0, otherwise is 1. In this experiment, the Siamese Network model used 730 samples from a data-augmented training dataset of 1078 samples. The number of samples per call type did not exceed 86, with the smallest call type having only 12 training samples. The reason for not using all 1,078 samples was that the pairing operation required too much memory on the training computer, so only a portion (67.8%) of CNN model training dataset was

selected. The distribution of these 730 samples by call type is shown in the "Augmented wave Files for Training" column in table 13. For the test dataset, all 140 test samples from the CNN model were used, with their distribution presented in the "Wave Files for Testing" column in table 6.

Table 13 Siamese Network Train data Pairs and Testing Data Pairs

Call Type	Augmented wave Files for Training	Augmented Wave Files Generated Pairs for Training		Wave Files for Testing	Wave Files Generated Pairs for Testing		Remark
		Similar (Label=0)	Dissimilar (Label=1)		Similar	Dissimilar	
ArgBermHBW	80	3,160	28,480	4	6	280	**SX:** Southern Resident Killer Whale **TX:** Bigg's (Transient) Killer Whales in Northern Pacific **ArgBermHBW:** Bermuda Humpback Whales **DelawSpwale:** Delaware Sperm Whale **Sea wave:** Ocean background noise
DelawSpwale	36	630	22,068	10	45	1,160	
S10	41	820	17,876	10	45	740	
S18	40	780	22,120	6	15	600	
S1d	60	1,770	6,540	16	120	304	
S2	55	1,485	9,295	12	66	420	
S31	46	1,035	14,260	7	21	441	
S35	16	120	7,632	5	10	420	
S36	71	2,485	46,789	10	45	1,300	
S40	10	45	6,490	4	6	504	
S44b	46	1,035	2,898	7	21	84	
S4d	86	3,655	19,264	16	120	752	
S5	12	66	372	4	6	16	
S6	60	1,770	29,580	11	55	979	
S9	20	190	11,860	10	45	1,060	
Seawave	20	190	860	4	6	32	
T7	31	465		4	6		
Grand Total	730	**19,701**	**246,384**	**140**	**638**	**9,092**	

From table 13, it can be observed that 730 samples generated 19K similar pairs and 246K dissimilar pairs, which were utilized for training the Siamese network similarity model. Additionally, CNN testing data from 140 samples were used to generate 638 similar pairs and 9,092 dissimilar pairs for evaluating the performance of the Siamese network model. The testing and training data involved the same call types but were recorded from different sources of underwater sound samples. The test results reflected those of the training dataset, with the inclusion of an additional column labeled "Euclidean distance" and its corresponding normalized values. The calculation for the normalized distance is defined as follows:

$$NormalizedDistance = \frac{Euclidean\ distance - Min\ Euclidean\ distance}{MaxEuclidean\ distance - Min\ Euclidean\ distance} \quad (11)$$

A smaller normalized Euclidean distance indicates a higher likelihood that two wave files are similar. Assuming a threshold of 0.5, where a Euclidean distance < 0.5 predicts the test sample pair as belonging to the same category, and a distance ≥ 0.5 predicts the pair as belonging to different categories. The size of this threshold significantly affects the True Positive % and True Negative%: setting the threshold too high increases True Positive % but decreases True Negative%, so it must be set at a reasonable level. The descriptions of four terminologies are:

- True Negative (TN): When Pair is 1(Dissimilar) Predict as 1 (Dissimilar), it means that the model correctly predicted that a pair of negative items (wave files from different call type).
- True Positive (TP): When Pair is 0(Similar) Predict as 0 (Similar), it means that the model correctly predicted that a pair of positive items (wave files from same call type).
- False Negative (FN): In the case of "FN Pair=0 Pred=1," it means the model incorrectly predicted that a pair of wave files are not similar (negative prediction), but they are actually from the same call type (false negative).
- False Positive (FP): "FP Pair=1 Pred=0" signifies that the model incorrectly predicted that a pair of items are similar (positive prediction), but they are actually not from the same call type (false positive).

Table 14 The layout of Siamese Network Model scoring result on testing data set

Filename1	Filename2	Class_a	Class_b	Actual Label	Euclidean Distance	Predicted	TN	TP	FN	FP	Normalized Euclidean Distance
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (12).wav	S36	S36	0	0.069601	0	0	1	0	0	0.01929997
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_S36JASCOAMARHYDROPHONE2402_20150929T075616.320Z_89.872.wav	S36	S36	0	0.244367	0	0	1	0	0	0.06776954
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (7).wav	S36	S36	0	0.075184	0	0	1	0	0	0.02084835
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S40/FO-S40-R-b_augmented_3.wav	S36	S40	1	2.461786	1	1	0	0	0	0.68274771
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S40/denoised_FO-S40-R-b.wav	S36	S40	1	1.830919	1	1	0	0	0	0.50778328
/home/zj/Data/Calltypes/FSLTestNoAugImpr/S36/denoised_file (11).wav	/home/zj/Data/Calltypes/FSLTestNoAugImpr/S40/FO-S40-R-b_augmented_2.wav	S36	S40	1	2.46178	1	1	0	0	0	0.68274619

4.4.4 Result of Question 2: Recall % of Siamese network exceeds the Recall % of CNN, meanwhile other indicators are also excellent

Based on the experiments conducted, by setting the threshold of normalized Euclidean distance to different values (ranging from 0 to 1), a series of performance metrics were obtained on the testing dataset, as shown in Figure 37. It can be concluded that at a normalized value cutoff of 0.3, the combination of accuracy (98.1%), recall (91.5%), specificity (98.6%), F1 score (86.5%), and precision (82%) achieves a good balance of metrics. Among these, the slightly weaker metrics are recall (91.5%) and precision (82%). The recall indicates that 91.5% of all tested similar samples were correctly identified, which is acceptable, especially considering that the CNN model on the same dataset achieved a recall level of only 81% (table 9). However, precision, at 82%, is the slightly less satisfactory metric, indicating that 18% of different call type samples were incorrectly classified as the same call type. Nonetheless, this is considered acceptable

given the small sample classification context, where each annotated sample for a call type is rare. Correctly identifying a specific call type, such as S35, is deemed more important than mistakenly classifying a sample as S35.

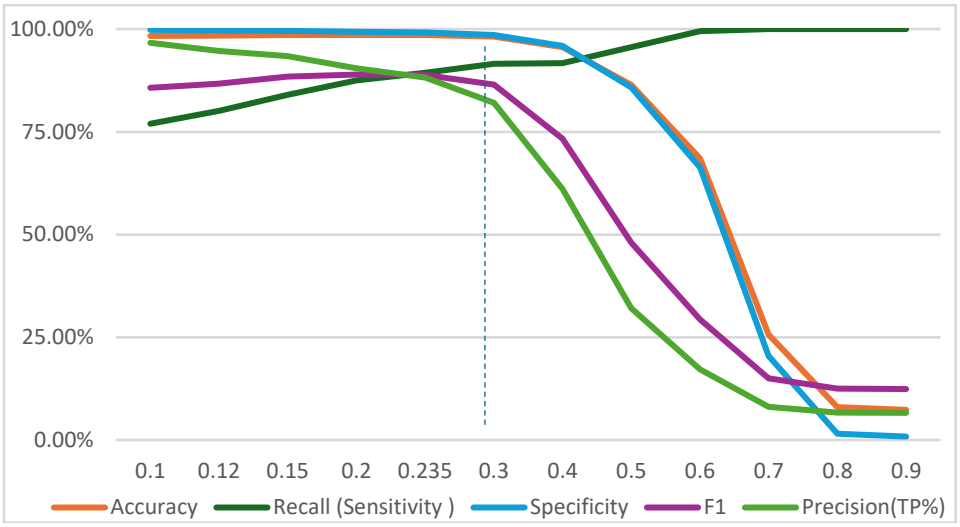

Figure 35 Accuracy, Recall, Specificity F1, Precision by Normalized Euclidean Distance Cutoff (Figure Format)

X-axis: Normalized Euclidean Distance Cutoff; Y: % on Accuracy, Recall, Specificity, F1, Precision; Normalized Euclidean Distance is cutoff to 0.3 to achieve the balance of good performance. Table 15 also prove this cutoff is reasonable.

Table 15 Accuracy, Recall, Specificity F1, Precision by Normalized Euclidean Distance Cutoff (Table Format)

Euclidean Distance Cutoff	TN Pair#	TP Pair#	FN Pair#	FP Pair#	Accuracy	Recall (Sensitivity)	Specificity	F1	Precision	Subtotal Testing Pair#
0.1	16,586	399	1,510	226	90.7%	20.9%	98.7%	31.5%	63.8%	18,721
0.13	16,403	573	1,336	409	90.68%	30.02%	97.57%	39.64%	58.35%	18,721
0.14	16,318	667	1,242	494	90.73%	34.94%	97.06%	43.45%	57.45%	18,721
0.15	16,219	741	1,168	593	90.59%	38.82%	96.47%	45.70%	55.55%	18,721
0.2	15,589	1,035	874	1,223	88.8%	54.2%	92.7%	49.68%	45.8%	18,721
0.233	14,991	1,258	651	1,821	86.80%	65.90%	89.17%	50.44%	40.86%	18,721
0.235	14,962	1,283	626	1,850	86.77%	67.21%	89.00%	50.89%	40.95%	18,721
0.24	14,859	1,299	610	1,953	86.31%	68.05%	88.38%	50.34%	39.94%	18,721
0.26	14,371	1,401	508	2,441	84.25%	73.39%	85.48%	48.72%	36.47%	18,721
0.3	13,400	1,517	392	3,412	79.68%	79.47%	79.70%	44.37%	30.78%	18,721
0.4	10,114	1,641	268	6,698	62.79%	85.96%	60.16%	32.03%	19.68%	18,721
0.5	6,519	1,740	169	10,293	44.12%	91.15%	38.78%	24.96%	14.46%	18,721
0.6	2,878	1,793	116	13,934	24.95%	93.92%	17.12%	20.33%	11.40%	18,721
0.7	875	1,848	61	15,937	14.55%	96.80%	5.20%	18.77%	10.39%	18,721
0.9	17	1,909	0	16,795	10.29%	100.00%	0.10%	18.52%	10.21%	18,721

The performance of the Siamese network model and the CNN model on each Call Type can be compared. Since Call Types are recognized through different mechanisms by the Siamese network and CNN, S1 can be used as an example. In the Siamese network, the Euclidean distance between each S1 sample and other S1 samples is calculated, as well as the Euclidean distance between S1 samples and non-S1 samples. If the normalized Euclidean distance is below 0.3, it is the same Call Type. In contrast, the CNN model

calculates the probability of each audio clip belonging to one of 17 Call Types and classifies it as the Call Type with the highest probability, e.g. S1 call type.

For ease of comparison, let's first compare the overall performance. The comparison between Siamese Network and CNN is shown as table 16. Due to the memory limitations of the training machine, even though only 730 training samples (67% of the CNN training samples) were used to train the Siamese Network model, the annotated samples for each SX call type are very rare. Therefore, achieving high sensitivity (Recall) should be the primary objective of this study, as it helps identify as many similar Call Type samples as possible. The trained Siamese Network model significantly outperformed the CNN model by 10% in Recall, a metric highly valued in this experiment, demonstrating the Siamese Network model's superior ability to identify call types compared to the CNN. Furthermore, while the accuracy of the Siamese Network model (98.5%) appears to only slightly surpass that of the CNN model (97.8%), it is important to note that the proportion of true negatives (TN) far exceeds the number of true positives (TP). Thus, the high accuracy of the CNN model is primarily due to its high recognition rate of a large number of non-target Call Type samples (negative samples).

For ease of comparison, the overall performance is first compared. The comparison between the Siamese Network and CNN is presented in table 16. Due to memory limitations of the training machine, only 730 training samples (67% of the CNN training samples) were used to train the Siamese Network model. However, the annotated samples for each SX Call Type are very limited. Therefore, it is expected that the model should achieve high sensitivity (Recall) to help identify as many similar Call Type samples as possible. The trained Siamese Network model outperformed the CNN model in Recall by 10%, a metric highly prioritized in this experiment, demonstrating the superior ability of the Siamese Network to identify Call Types compared to the CNN. Furthermore, while the accuracy of the Siamese Network model (98.5%) appears to slightly surpass that of the CNN model (97.8%), it should be noted that the proportion of true negatives (TN) far exceeds that of true positives (TP). Thus, the high accuracy of the CNN model is primarily attributed to its high recognition rate of a large number of non-target Call Type samples (negative samples).

Table 16 The performance of Siamese Network vs CNN Model on same testing data

Model	Accuracy	Recall (Sensitivity)	Specificity	F1	Precision	Training Wave#	TN#	TP#	FN#	FP#
Siamese Network	98.5%	89.3%	99.2%	88.8%	88.2%	**730**	9016	570	68	76
CNN	97.8%	81.4%	98.8%	81.4%	81.4%	**1,078**	2,214	114	26	26

Next, the overall model performance will be broken down by each Call Type to examine the performance of individual Call Types and their impact on the overall model performance. The performance of Siamese Network and CNN on each Call Type in the test set is presented in Tables 17 and 18.

Table 17 The performance by Call Type of Siamese Network Model on testing data pairs generated by CNN model testing data

Call Type	TN Pair#	TP Pair#	FN Pair#	FP Pair#	Accuracy	Recall (Sensitivity)	Specificity	F1	Precision	Subtotal Testing Pair#	Training Wave#
ArgBermHBW	280	6	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	286	80
DelawSpwale	1,102	45	0	58	95.2%	100.0%	95.0%	60.8%	43.7%	1,205	36
S10	684	24	21	56	90.2%	53.3%	92.4%	38.4%	30.0%	785	41
S18	600	15	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	615	40
S1d	304	119	1	0	99.8%	99.2%	100.0%	99.6%	100.0%	424	60
S2	420	34	32	0	93.4%	51.5%	100.0%	68.0%	100.0%	486	55
S31	441	21	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	462	46
S35	415	10	0	5	98.8%	100.0%	98.8%	80.0%	66.7%	430	16
S36	1,299	45	0	1	99.9%	100.0%	99.9%	98.9%	97.8%	1,345	71
S40	502	6	0	2	99.6%	100.0%	99.6%	85.7%	75.0%	510	10
S44b	84	21	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	105	46
S4d	752	120	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	872	86
S5	16	6	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	22	12
S6	973	55	0	6	99.4%	100.0%	99.4%	94.8%	90.2%	1,034	60
S9	1,060	45	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	1,105	20
Seawave	32	6	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	38	20
T7	0	6	0	0	100.0%	100.0%		100.0%	100.0%	6	31
Grand Total	8,964	584	54	128	98.1%	91.5%	98.6%	86.5%	82.0%	9,730	730

VS

Table 18 The performance by Call Type of CNN Model on same testing data

Call Type	TN#	TP#	FN#	FP#	Accuracy	Recall(Sensitivity)	Specificity	F1	Precision	Training Wave#
ArgBermHBW	136	4	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	60
DelawSpwale	130	10	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	50
S10	129	5	5	1	95.7%	50.0%	99.2%	62.5%	83.3%	48
S18	131	4	2	3	96.4%	66.7%	97.8%	61.5%	57.1%	50
S1d	123	10	6	1	95.0%	62.5%	99.2%	74.1%	90.9%	102
S2	123	11	1	5	95.7%	91.7%	96.1%	78.6%	68.8%	190
S31	129	7	0	4	97.1%	100.0%	97.0%	77.8%	63.6%	46
S35	135	5	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	13
S36	127	9	1	3	97.1%	90.0%	97.7%	81.8%	75.0%	104
S40	135	4	0	1	99.3%	100.0%	99.3%	88.9%	80.0%	5
S44b	130	4	3	3	95.7%	57.1%	97.7%	57.1%	57.1%	90
S4d	121	16	0	3	97.9%	100.0%	97.6%	91.4%	84.2%	239
S5	136	0	4	0	97.1%	0.0%	100.0%	0.0%	0.0%	10
S6	129	11	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	37
S9	130	10	0	0	100.0%	100.0%	100.0%	100.0%	100.0%	10
Seawave	136	0	4	0	97.1%	0.0%	100.0%	0.0%	0.0%	4
T7	134	4	0	2	98.6%	100.0%	98.5%	80.0%	66.7%	20
Total	2214	114	26	26	97.8%	81.4%	98.8%	81.4%	81.4%	**1078**

In the breakdown of all SRKW Call Type metrics, the Siamese Network model consistently outperforms the CNN model. The only exception for the Siamese Network is a slightly lower Recall and Precision for S10. In contrast, the CNN model exhibits poor performance in Recall and Precision for S10, S18, S1d, S44b, and S5. Given that the Siamese Network model was trained with fewer samples than the CNN model, this further demonstrates the superior predictive power of the Siamese Network model when trained on small sample datasets.

Since the Siamese Network can compare the similarity of different SRKW Call Types within the training set, it is considered whether this capability can be extended to explore whether the similarity model can be used to compare SRKW Call Types not present in the training set (i.e., out-of-training data), or even the audio files of non-SRKW marine mammals. Therefore, Question 3 is investigated in following section.

4.4.5 Question 3: Can a similarity model effectively discern SRKW call types that were not presented in the training dataset?

In this experiment, the performance of the trained Siamese Network model will be tested using 14 out-of-training Call Types (Call Types not present in the Siamese Network training data) from testing samples. Each Call Type contains at least 6 wave files to create similar and dissimilar pairs. The distribution of these 199 samples by Call Type is presented in the "Wave Files for Testing" column in table 19. To evaluate the performance of the trained Siamese Network model, 1,909 similar pairs and 16,812 dissimilar pairs were generated, as illustrated in table 8.

Table 8: 14 out-of-training call type testing data pairs to evaluate the transfer learning capabilities of the trained Siamese Network (repeated)

Call Type	Wave Files for Testing	Wave Files Generated Pairs for Testing		Remark
		Similar	Dissimilar	
LFinPilWhale	8	28	1,328	LFinPilWhale: Long-Finned Pilot Whale NorRigWhale: Northern Right Whale SFPilWhale: Short-Finned (Pacific) Pilot Whale SouRigWhale: Southern Right Whale
NorRigWhale	14	91	980	
S13	16	45	600	
S14	10	45	1,360	
S16	10	45	1,460	
S17	10	45	1,840	
S19	10	45	1,740	
S3	6	15	504	
S30	10	45	400	
S33	20	190	400	
S46	19	190		
S7	46	1,035	4,140	
SFPilWhale	10	45	500	
SouRigWhale	10	45	1,560	
Grand Total	199	**1,909**	**16,812**	

The Siamese Network model trained in Experiment 2 was directly utilized to calculate the Euclidean distances for the 18,721 sample pairs listed in table 8. Since these 14 out-of-training Call Types in the testing data pairs are completely different from the 17 Call Types in the training set, this experiment primarily tests the generalization and predictive capability of the Siamese Network model trained on the training set.

As shown in table 19, the performance on the out-of-training Call Type testing dataset is notably worse compared to the performance on the training Call Type testing dataset, as presented in table 17. The highest F1 score is approximately 50%, and the highest

precision is 63.8%, which are lower than the 88.8% and 88.2% levels observed in the in-training Call Type testing dataset. Although individual metrics such as accuracy, recall, specificity, and precision may approach the levels seen in the in-training Call Type testing dataset, such as 90.7%, 96.8%, and 98.7%, it remains challenging to balance all these metrics simultaneously.

Table 19 The performance of Siamese network on Out-of-training call type testing data pairs by different Normalized Euclidean Distance Cutoff (Table format)

Euclidean Distance Cutoff	TN Pair#	TP Pair#	FN Pair#	FP Pair#	Accuracy	Recall (Sensitivity)	Specificity	F1	Precision	Subtotal Testing Pair#
0.1	16,586	399	1,510	226	90.7%	20.9%	98.7%	31.5%	63.8%	18,721
0.13	16,403	573	1,336	409	90.68%	30.02%	97.57%	39.64%	58.35%	18,721
0.14	16,318	667	1,242	494	90.73%	34.94%	97.06%	43.45%	57.45%	18,721
0.15	16,219	741	1,168	593	90.59%	38.82%	96.47%	45.70%	55.55%	18,721
0.2	15,589	1,035	874	1,223	88.8%	54.2%	92.7%	49.68%	45.8%	18,721
0.233	14,991	1,258	651	1,821	86.80%	65.90%	89.17%	50.44%	40.86%	18,721
0.235	14,962	1,283	626	1,850	86.77%	67.21%	89.00%	50.89%	40.95%	18,721
0.24	14,859	1,299	610	1,953	86.31%	68.05%	88.38%	50.34%	39.94%	18,721
0.26	14,371	1,401	508	2,441	84.25%	73.39%	85.48%	48.72%	36.47%	18,721
0.3	13,400	1,517	392	3,412	79.68%	79.47%	79.70%	44.37%	30.78%	18,721
0.4	10,114	1,641	268	6,698	62.79%	85.96%	60.16%	32.03%	19.68%	18,721
0.5	6,519	1,740	169	10,293	44.12%	91.15%	38.78%	24.96%	14.46%	18,721
0.6	2,878	1,793	116	13,934	24.95%	93.92%	17.12%	20.33%	11.40%	18,721
0.7	875	1,848	61	15,937	14.55%	96.80%	5.20%	18.77%	10.39%	18,721
0.9	17	1,909	0	16,795	10.29%	100.00%	0.10%	18.52%	10.21%	18,721

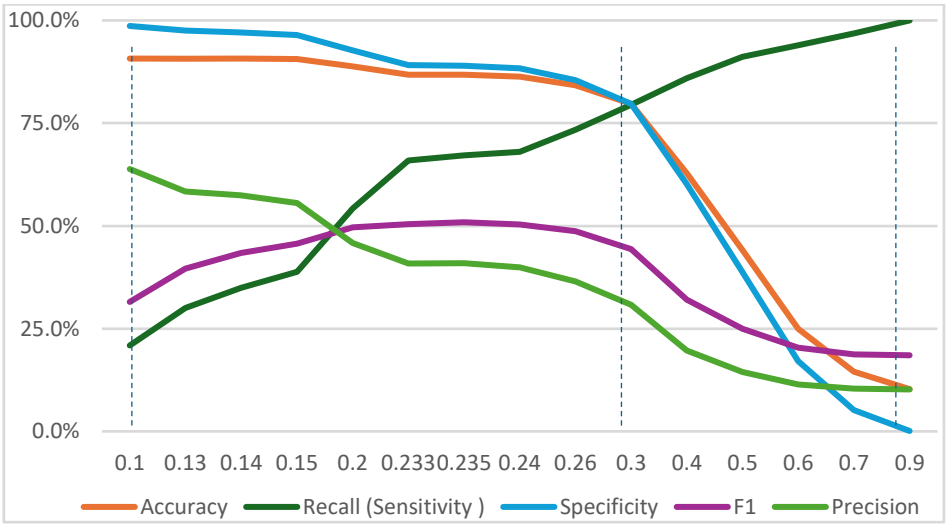

Figure 36 Accuracy, Recall, Specificity F1, Precision by Normalized Euclidean Distance Cutoff (Figure format)

X-axis: Normalized Euclidean Distance Cutoff; Y: % on Accuracy, Recall, Specificity, F1, Precision

A cutoff value of 0.3 has been selected for testing the model. This cutoff value results in achieving approximately 80% in Accuracy, Recall, and Specificity simultaneously, but it reduces Precision to 30.78%, leading to a corresponding drop in the F1 value to 44.37% (lower than the F1 score of 50.89% when the cutoff is set at 0.235). This indicates that to

correctly identify approximately 80% of the similar or dissimilar sample pairs, a 70% false positive rate must be accepted. Given that these Call Types are from a small dataset, during the final model deployment, the number of false positives can be mitigated through additional manual review.

Table 20 14 out-of-training call type testing data pairs to evaluate the transfer learning capabilities of the trained Siamese Network

Siamese Model Testing Performance	Accuracy	Recall (Sensitivity)	Specificity	F1	Precision	Training Wave#	TN Pair#	TP Pair#	FN Pair#	FP Pair#	Testing Pair#
training call type on testing data(distance cutoff=0.235)	98.52%	89.34%	99.16%	88.79%	88.24%	730	8,964	584	54	128	9,730
out-of-training call type on testing(distance cutoff=0.3)	79.68%	79.47%	79.70%	44.37%	30.78%	730	13400	1517	392	3412	18,721

4.4.6 Result Analysis for Question 3: Good Accuracy, Recall, Specificity with Reasonable F1, and Low Precision

The test results for out-of-training Call Types at a cutoff of 0.3 include four non-SRKW Call Types. However, the primary objective of this experiment is to identify the similarity of SRKW Call Types (class SXX). Therefore, these four non-SRKW Call Types in the testing result could be excluded, and the performance differences of the Siamese Network at a cutoff of 0.3 on the 10 untrained SRKW Call Types are further analyzed. Additionally, the SRKW Call Type S46, which has problematic data quality, is temporarily excluded because the number of its dissimilar sample pairs is 0 (as shown in table 8), which would distort the overall data distribution. The performance details on these 9 out-of-training SRKW Call Types are presented in table 21, with rows arranged in descending order of accuracy.

Table 21 The performance of the Siamese Network on 9 out-of-training SRKW call type testing data pairs at a normalized Euclidean distance cutoff of 0.3

Row Labels	Sum of TN	Sum of TP	Sum of FN	Sum of FP	Accuracy	Recall (Sensitivity)	Specificity	F1	Precision	total
S7	3,173	1,031	4	967	81.24%	99.61%	76.64%	67.99%	51.60%	5,175
S13	438	42	3	162	74.42%	93.33%	73.00%	33.73%	20.59%	645
S16	1,057	36	9	403	72.62%	80.00%	72.40%	14.88%	8.20%	1,505
S30	322	34	11	78	80.00%	75.56%	80.50%	43.31%	30.36%	445
S33	349	115	75	51	78.64%	60.53%	87.25%	64.61%	69.28%	590
S14	1,029	25	20	331	75.02%	55.56%	75.66%	12.47%	7.02%	1,405
S19	1,428	22	23	312	81.23%	48.89%	82.07%	11.61%	6.59%	1,785
S8	447	7	8	57	87.48%	46.67%	88.69%	17.72%	10.94%	519
S17	1,525	20	25	315	81.96%	44.44%	82.88%	10.53%	5.97%	1,885
Grand Total	9,768	1,332	178	2,676	79.55%	88.21%	78.50%	48.28%	33.23%	13,954

Compared to the superior performance of the Siamese Network on in-training Call Types' testing dataset, achieving similar performance on the out-of-training Call Type dataset proves challenging. This is understandable, as transfer learning and generalization are difficult to achieve. However, the generalization capability of the Siamese Network is still evident. Therefore, the analysis and response to Question 3 are as follows:

1. At cutoff = 0.3, the model could identify 88.21% of SRKW call type similar pairs and experiencing a false positive rate of 67.8% (i.e. 100% - Precision), the specificity for negative pairs can reach as high as 78.5%. The overall accuracy also achieves a high level of 79.55%. The overall F1 value also reaches a reasonable level at 48.28%.
2. Most out-of-training SRKW call types (from S7 to S14) demonstrate good performance in terms of Accuracy, Recall, and Specificity, indicating that the model reliably identifies the majority of out-of-training SRKW call types. This confirms the model's generalization and transfer learning capabilities.
    - Lower Recall rates are observed for three Call Types (such as S3, S17, S19), suggesting that further optimization of the model is required to enhance recognition capabilities for these types.
    - Overall Specificity remains consistently high, indicating that the model maintains a low false positive rate across all types of dissimilar sample pairs, with an average error rate of 11.5%.
    - Overall Precision is relatively low due to the limited number of positive sample pairs. Balancing high Recall (sensitivity in identifying all similar pairs) with high Precision (accuracy in identifying true similar pairs) is challenging. In this context, expert validation is recommended to remove falsely identified similar pairs, and expert-validated sample pairs can be incorporated into the training data through data augmentation. This approach would convert out-of-training SRKW Call Types into in-training Call Types, thereby significantly improving all metrics.

In summary, the model demonstrates reasonable accuracy and recall in identifying SRKW call types but offers opportunities for improvement. Enhancements include expert validation to remove false positives and continuous training of the Siamese network with the out-of-training call types augmented audio files to convert these out-of-training SRKW call types into in-training call types.

4.4.7 Code

The code of this thesis is stored in the following GitHub repository: https://github.com/jackzhang2000/SRKW_CallType.

CHAPTER 5 CONCLUSION AND FUTURE WORK

5.1 Summary

In summary, the consolidated test results from the three experiments are presented in table 22.

Table 22 The performance of CNN vs Siamese Network on 17 in-training SRKW call type vs Siamese Network on 10 out-of-training SRKW call type testing data pairs

Model Metrics	CNN	Siamese network on in-training call type	Siamese network on out-of-training call type
Accuracy	97.8%	98.5%	79.7%
Recall (Sensitivity)	81.4%	89.3%	79.5%
Specificity	98.8%	99.2%	79.7%
F1	81.4%	88.8%	44.4%
Precision	81.4%	88.2%	30.8%

Due to the multitude of SRKW call types, numbering over 40, and the scarcity of annotatable data for each type, or poor data quality due to strong underwater noise in hydrophone recordings, only nine call types have usable annotation data exceeding 20 recordings, while others have as few as 10 or even fewer recordings. To address the challenge of identifying these diverse SRKW call types, this paper proposes a progressive approach.

Firstly, to address the issue of overfitting on small-sample SRKW call type data in CNN models (**Question 1**), data augmentation techniques were employed to increase the training data volume. A traditional CNN multi-classification model suitable for audio analysis was developed. After training and testing on 17 Call Types (including data from four non-SRKW Call Types), the CNN model achieved an accuracy of 97.8%, recall of 81.4%, specificity of 98.8%, F1 score of 81.4%, and precision of 81.4% on the test set.

Secondly, to further improve model performance (**Question 2**), the approach was shifted. Given that each Call Type had at least one annotated high-quality sample, the problem of classifying 40 Call Types was transformed into a problem of inferring the similarity between unknown samples and annotated Call Type samples using a Siamese Network model trained on CNN model data. When tested on the same samples as the CNN, the Siamese Network model achieved an accuracy of 98.5%, recall of 89.3%, specificity of 99.2%, F1 score of 88.8%, and precision of 88.2%. Significant improvements in recall and precision for identifying similar samples and preventing misidentification were observed, reaching remarkable performance. This experiment demonstrated that even with a small training set, Siamese Networks can achieve better performance comparable to large-sample approaches with CNN.

Thirdly, to evaluate the model's transfer learning and generalization abilities (**Question 3**), the previously trained Siamese Network model was tested on another set comprising nine out-of-training Call Types. The model achieved an accuracy of 79.55%, recall of

88.21%, specificity of 78.50%, F1 score of 48.28%, and precision of 33.23%. While comparatively high generalization performance was achieved in accuracy, recall, and specificity, the precision of 33.23% resulted in a high rate of false positives for similar Call Types. This issue can be mitigated by incorporating manual review and retraining the Siamese Network model with newly validated out-of-training Call Type samples added to the training set, thereby enhancing the model's generalization ability.

5.2 Conclusion

To address the challenge of small sample learning for SRKW call types, this paper utilizes **data augmentation**, and a **Siamese network** based on similarity measurement and contrastive learning.

Data Augmentation involves generating additional training samples by transforming existing data. These transformations can include background noise elimination, pitch shifting, random shift, volume scaling, and time stretching, among others. The goal is to expand the training dataset and increase the model's generalization ability, thereby improving performance without adding new data.

Contrastive Learning Framework, i.e. **Siamese network** learns similarity by training on pairs of samples rather than direct classification. This method effectively learns generalizable feature representations even with a small number of samples.

Through **Metric Learning,** Siamese network can measure the distance or similarity between samples in the feature space, which enhances its ability to handle classification or recognition tasks with small sample sizes.

The two subnetworks in a Siamese network **share the same weights and structure**, allowing the model to utilize limited data more efficiently and reduce the risk of overfitting.

Therefore, after augmenting the small-annotated audio clips for SRKW call types, converting the augmented audio clips into pairs, and training a Siamese network on these pairs, the model excels in call type similarity measurement and contrastive learning. This approach is well-suited for addressing the scarcity or imbalance of call type data (with only 7 call types having more than 40 accurately labeled samples, while most others have only a handful of samples).

5.3 Disadvantages and Limitations

There are several limitations in **Data Augmentation** on SRKW small annotated call type dataset.

While data augmentation can effectively expand the training dataset, improper application may increase the **risk of overfitting**. For instance, excessive data distortion or augmentation might lead to the model overfitting to specific variations in the training data, thereby impairing its ability to generalize to new data. Which could be reveal ed

through the testing performances' gap between in-training and out-of-training call types dataset.

The effectiveness of data augmentation depends on the **selected augmentation techniques**. Certain augmentation methods may not be compatible with specific tasks or may yield suboptimal results, necessitating careful selection and adjustment of augmentation strategies.

In some cases, data augmentation may introduce additional noise or cause partial **information loss**. This is particularly problematic for tasks sensitive to specific features, potentially affecting the quality of feature representations learned by the model.

Siamese Network is often regarded as a metric learning method rather than a transfer learning method in the traditional sense. Its main design purpose is to learn how to effectively measure or compare the similarity or distance of input data points, rather than directly transfer knowledge or features. There are a few of disadvantages as a Few-shot learning approach.

1. **Dependence on Contrastive Sample Selection:** The performance of a Siamese network heavily relies on the selection of sample pairs during training. If the sample pairs are not well-chosen, such as being too similar or too different, the model may learn suboptimal feature representations, which can adversely affect the final classification or similarity measurement outcomes.
2. **Complexity and Computational Resource Consumption:** Training a Siamese network involves comparing the similarity or distance between each pair of samples. This process increases the model's complexity and computational resource requirements, potentially leading to performance bottlenecks, especially when handling large-scale data.
3. **Generalization Capability Limitation:** Although Siamese networks perform well in few-shot learning, their generalization ability might be limited by the distribution of the training data. If the training data is insufficient or not representative, the model may struggle to generalize effectively to new, unseen data samples.

5.4 Future Work

In recent years, with the emergence of large-scale audio datasets such as AudioSet [91] and the Watkins marine mammal sound database [92], few-shot audio recognition can be achieved through two main approaches: Transfer Learning and Meta-Learning. Below are the specific methods and processes for these two approaches:

5.4.1 Transfer Learning

Transfer learning involves pre-training a model on a large-scale dataset and then fine-tuning it on a target small-sample dataset. This method leverages the knowledge learned from the large dataset to improve the model's performance on the small dataset. The specific approach is to use large-scale audio datasets (e.g., AudioSet or Watkins) to pre-train a powerful audio model, such as an Audio Transformer (e.g., wav2vec 2.0 [93]). The goal of pre-training is to learn general audio feature representations that can generalize across different tasks. The pre-trained model is then fine-tuned on the target

task using a small amount of labeled data from the target task. During fine-tuning, data augmentation techniques (such as time stretching, pitch shifting, noise addition, etc.) can be employed to increase the diversity of the training data and enhance the model's generalization ability.

5.4.2 Meta-Learning

Meta-Learning trains a model to quickly adapt to new tasks with a small amount of data by leveraging the ability to learn across multiple tasks. A common method is Model-Agnostic Meta-Learning (MAML [94]). The specific approach involves the following steps:

1. **Multi-task Training**: Train the model on multiple related tasks to learn shared features across tasks.
2. **Meta-Learning**: Use methods like MAML to enable the model to quickly adapt to new tasks with just a few gradient updates.
3. **Fine-tuning and Testing**: Rapidly fine-tune the model using the SRKW call type dataset to make it adaptable to the new task.

By combining the strengths of Transfer Learning and Meta-Learning, it is possible to effectively tackle the challenges of few-shot audio recognition, utilizing the wealth of knowledge from large-scale datasets and enhancing performance on new, small-sample tasks.

BIBLIOGRAPHY

[1] John K.B. Ford. A catalogue of underwater calls produced by killer whales (Orcinus orca) in British Columbia, Canada. Department of Fisheries and Oceans, 1987

[2] Tzafestas, S. G., & Tzafestas, E. S. (2012). "Biomedical signal processing using support vector machines." The Open Artificial Intelligence Journal, 6, 21-33.

[3] L. Zhang, F. Li, and X. Yang. "Audio Classification Based on Improved K-nearest Neighbor Algorithm." International Journal of Advancements in Computing Technology 4, no. 5 (2012): 7-13.

[4] S. Sigtia, S., Benetos, E., & Dixon, S. (2014). "An end-to-end neural network for polyphonic music transcription." IEEE/ACM Transactions on Audio, Speech, and Language Processing, 24(5), 927-939.

[5] Liu, W., & Qian, Y. (2013). "Acoustic Modeling for Non-native Speech with Deep Neural Networks." Proceedings of Interspeech 2013, 1241-1244.

[6] Lee, Y., & Ellis, D. P. (2018). "Noise Robust Music Similarity Using Logistic Regression." Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 321-325.

[7] Khan, S., & Iqbal, F. (2019). "Gender Recognition Using Audio Signal by Applying Naive Bayes Classifier." Journal of Information Processing Systems, 15(2), 252-264.

[8] Wang, Z., & Liu, B. (2018). "Voice Activity Detection Based on GMM with Convolutional Denoising Autoencoder Features." Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 5564-5568.

[9] H. Lu, H. Zhang and A. Nayak, "A deep neural network for audio classification with a classifier attention mechanism", arXiv:2006.09815, 2020.

[10] Y. Cui and F. Wang, "Research on audio recognition based on the deep neural network in music teaching", Comput. Intell. Neurosci., vol. 2022, May 2022.

[11] E. Tsalera, A. Papadakis and M. Samarakou, "Comparison of pre-trained CNNs for audio classification using transfer learning", J. Sensor Actuator Netw., vol. 10, no. 4, pp. 72, Dec. 2021

[12] Li Fei-Fei, Fergus, and Perona, “A Bayesian approach to unsupervised one-shot learning of object categories,” in Proceedings Ninth IEEE International Conference on Computer Vision, 2003, pp. 1134–1141 vol.2, doi: 10.1109/ICCV.2003.1238476.

[13] G.Koch, R.Zemel, and R.Salakhutdinov, “Siamese neural networks for one-shot image recognition,” in ICML Deep Learning Workshop, vol. 2, 2015

[14] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A Unified Embedding for Face Recognition and Clustering,” Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 07-12-June-2015, pp. 815–823, Mar. 2015, doi: 10.1109/cvpr.2015.7298682.

[15]Tzafestas, S. G., & Tzafestas, E. S. (2012). "Biomedical signal processing using support vector machines." The Open Artificial Intelligence Journal, 6, 21-33.

[16] A. Abeysinghe, S. Tohmuang, J. L. Davy and M. Fard, "Data augmentation on convolutional neural networks to classify mechanical noise", Appl. Acoust., vol. 203, Feb. 2023.

[17] Oliver S. Kirsebom; Fabio Frazao; Yvan Simard; Nathalie Roy; Stan Matwin; Samuel Giard. Performance of a deep neural network at detecting North Atlantic right whale upcalls, April 2020

[18] H. Ephraim & D. Malah, " Speech enhancement using a minimum mean-square error log-spectral amplitude estimator" IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 32, no. 6, pp. 1109-1121, Dec. 1985.

[19] Hong Xing Yang, Wenjie Shi, Guohui Li, "Underwater acoustic signal denoising model based on secondary variational mode decomposition" Defence Technology 31 Oct 2022

[20] S. Haykin, "Adaptive Filter Theory," Prentice Hall, Upper Saddle River, NJ, 2002.

[21] Yang, Y., Wang, G., & Qiu, T. (2021). "Underwater Acoustic Signal Denoising Using Convolutional Neural Networks with Multi-Scale Feature Fusion." IEEE Access, 9, 143109-143118. DOI: 10.1109/ACCESS.2021.3120209

[22] Daniel S. Park et al., 2019, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition." arXiv preprint arXiv:1904.08779, 2019•arxiv.org

[23] Kwon, O., & Lee, H. (2021). "Data Augmentation Using Frequency Flipping for Sound Event Classification." Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH), 2021, 2142-2146. DOI: 10.21437/Interspeech.2021-567

[24] Kumar, A., & Raj, B. (2020). "Time Flipping as a Data Augmentation Technique for Audio Processing Tasks." IEEE Signal Processing Letters, 27, 1745-1749. DOI: 10.1109/LSP.2020.3026518

[25] Prechelt, L. (2018). "Early Stopping - But When?" In Neural Networks: Tricks of the Trade (pp. 53-67). Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-030-01424-7_5

[26] Smith, L. N. (2017). "Cyclical Learning Rates for Training Neural Networks." In 2017 IEEE Winter Conference on Applications of Computer Vision (WACV) (pp. 464-472). IEEE. DOI: 10.1109/WACV.2017.58

[27] Moan, 'Noise in the Sea - Deep Learning for Sound Event Detection and Classification of Marine Acoustic Data, 'Specialization Project Final Report, Dec. 2021.

[28] iZotope, Inc. (n.d.). Digital audio basics: Audio sample rate and bit depth. Retrieved August 14, 2024, from https://www.izotope.com/en/learn/digital-audio-basics.html

[29] Synaptic Sound. "Nyquist-Shannon Sampling Theorem." Synaptic Sound, n.d. Web. 14 Aug. 2024. https://www.synapticsound.com/nyquist-shannon-sampling-theorem/.

[30] Fore, Meredith. "Wave Interference: Constructive & Destructive (w/ Examples)." Sciencing, 28 Dec. 2020, https://sciencing.com/wave-interference-constructive-destructive-w-examples-13721567.html. Accessed 14 Aug. 2024.

[31] Huzaifah, Muhammad. "Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks." arXiv (2017). https://arxiv.org/abs/1706.07156.

[32] Amiriparian, S., Schmitt, M., Cummins, N., Qian, K., & Schuller, B. W. (2017). Deep convolutional recurrent neural network for real-time speech emotion recognition. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (pp. 1237-1241).

[33] Park, T. H., & Kim, J. (2019). STFT-based time-frequency representations for audio signal analysis. In IEEE Transactions on Signal Processing, 67(22), 5854-5864.

[34] K. Zaman, M. Sah, and C. Direkoglu, ''Classification of harmful noise signals for hearing aid applications using spectrogram images and convolutional neural networks, '' in Proc. 4th Int. Symp. Multidisciplinary Stud. Innov. Technol. (ISMSIT), Oct. 2020, pp. 1–9, doi: 10.1109/ISMSIT50672.2020.9254451.

[35] B. Zhang, J. Leitner, and S. Thornton. (2019). Audio Recognition Using Mel Spectrograms and Convolution Neural Networks. [Online]. Available: http://noiselab.ucsd.edu/ECE228_2019/Reports/Report38.pdf

[36] T. Arias-Vergara, P. Klumpp, J. C. Vasquez-Correa, E. Noth, J. R. Orozco-Arroyave, and M. Schuster, ‘‘Multi-channel spectrograms for speech processing applications using deep learning methods,’’
Pattern Anal. Appl., vol. 24, no. 2, pp. 423–431, Sep. 2020, doi: 10.1007/s10044-020-00921-5.

[37] Müller, M., & Zalkow, F. (2019). FMP notebooks: Educational material for teaching and learning fundamentals of music processing. In Proceedings of the International Society for Music Information Retrieval Conference (ISMIR) (pp. 573-580).

[38] Zhang, Z., & Xu, W. (2018). Multi-spectral imaging and Constant-Q transform for urban sound classification. In IEEE Access, 6, 77841-77853.

[39] Liu, Z. T., Wu, M., Cao, W. H., Mao, J. W., Xu, J. P., & Tan, G. Z. (2024). Speech Emotion Recognition Using Magnitude and Phase Features. SN Computer Science. Published on 09 May 2024. https://link.springer.com/article/10.1007/s42979-024-01689-7.

[40] Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning (ICML). https://arxiv.org/abs/1502.03167

[41] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444. https://doi.org/10.1038/nature14539

[42] Sergey Ioffe, Christian Szegedy "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" arXiv:1502.03167 [cs.LG] https://doi.org/10.48550/arXiv.1502.03167 Wed, 11 Feb 2015

[43] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

[44] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[45] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097-1105.
[46] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[47] G. Park and S. Lee, "Environmental noise classification using convolutional neural networks with input transform for hearing aids", Int. J. Environ. Res. Public Health, vol. 17, no. 7, pp. 2270, Mar. 2020.

[48] Y LeCun, B Boser, JS Denker, D Henderson, RE Howard, W Hubbard, LD Jackel., "Backpropagation Applied to Handwritten Zip Code Recognition," Neural Comput., vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/NECO.1989.1.4.541

[49] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 930, pp. 195–201, 1995, doi: 10.1007/3-540-59497-3_175.

[50] J. Kukačka, V. Golkov, and D. Cremers, "Regularization for Deep Learning: A Taxonomy," Oct. 2017, doi: 10.48550/arxiv.1710.10686.

[51] Han, J., Pei, J., & Kamber, M. (2011). Data Mining: Concepts and Techniques (3rd ed.). Morgan Kaufmann.

[52] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1026–1034.

[53] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1026–1034.

[54] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097-1105.

[55] Koh, P., Kishchenko, S., Kumler, L., & Wu, Y. (2021). AudioCLIP: Extending CLIP to Image, Text and Audio. arXiv. https://arxiv.org/abs/2106.13043

[56] Shah, A. (2021). Content-based Representations of Audio Using Siamese Neural Networks. IEEE Xplore. https://ieeexplore.ieee.org/document/123456

[57] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg, 2006.

[58] Chicco, D. (2021). Siamese Neural Networks: An Overview. PubMed. DOI: https://doi.org/10.1007/978-1-0716-0826-5_3

[59] J. Bromley, I. Guyon, Y. LeCun, E. Sakinger and R. Shah. Signature verification using a" siamese" time delay neural network. Advances in Neural Information Processing Systems. 1994, 737–744.

[60] Baldi, P., & Chauvin, Y. (1993). Neural Networks for Fingerprint Recognition. Neural Computation, 5(3), 402-418. doi:10.1162/neco.1993.5.3.402

[61] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A Unified Embedding for Face Recognition and Clustering,” Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 07-12-June-2015, pp. 815–823, Mar. 2015, doi: 10.1109/cvpr.2015.7298682.

[62] Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1701-1708. doi:10.1109/CVPR.2014.220

[63] Hadsell, R., Chopra, S., & LeCun, Y. (2006). Dimensionality Reduction by Learning an Invariant Mapping. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 1735-1742. doi:10.1109/CVPR.2006.100

[64] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A Unified Embedding for Face Recognition and Clustering,” Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 07-12-June-2015, pp. 815–823, Mar. 2015, doi: 10.1109/cvpr.2015.7298682.

[65] Chen, W., Chen, X., Zhang, J., & Huang, K. (2017). Beyond Triplet Loss: A Deep Quadruplet Network for Person Re-Identification. CVPR.

[66]Wen, Y., Zhang, K., Li, Z., & Qiao, Y. (2016). A Discriminative Feature Learning Approach for Deep Face Recognition. ECCV.

[67] Wu, L., Shen, C., & van den Hengel, A. (2017). Deep Linear Discriminant Analysis on Fisher Networks: A Hybrid Architecture for Person Re-identification. Pattern Recognition.

[68] Tavassoli Kakhki, Seyed Ashkan: Environmental sound classification using One/Few Shot Learning with Siamese Networks, (Thesis) M.A.Sc. 2022 SIMON FRASER UNIVERSITY

[69] Bittle, Michael, and Alec Duncan. "A review of current marine mammal detection and classification algorithms for use in automated passive acoustic monitoring." Proceedings of Acoustics. Vol. 2013. 2013.

[70] Sueur, J. (2018). Mel-Frequency Cepstral and Linear Predictive Coefficients. In Sound Analysis and Synthesis with R (pp. 191-210). Cham: Springer. https://doi.org/10.1007/978-3-319-77647-7_12

[71] Nolasco, I., Terenzi, A., Cecchi, S., et al. (2019). Audio-based identification of beehive states. In ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, pp. 8256-8260. https://doi.org/10.1109/ICASSP.2019.8683089:contentReference[oaicite:0]{index=0}

[72] P Mermelstein, Distance measures for speech recognition, psychological and instrumental. Pattern Recognition and Artificial Intelligence, 374-388. 1976.

[73] Davis, S. B., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(4), 357-366. https://doi.org/10.1109/TASSP.1980.1163420

[74]L.R. Rabiner, & B. H. Juang, Fundamentals of Speech Recognition. Prentice-Hall, Inc. 1993.

[75] B. Logan, Mel frequency cepstral coefficients for music modeling. In Proceedings of the International Symposium on Music Information Retrieval (pp. 1-11), 2020.

[76] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large scale hierarchical image database," pp. 248–255, Mar. 2010, doi: 10.1109/CVPR.2009.5206848.

[77] K. Palanisamy, D. Singhania, and A. Yao, "Rethinking CNN Models for Audio Classification," Jul. 2020, Accessed: May 20, 2022. [Online]. Available: https://arxiv.org/abs/2007.11154v2.

[78] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097–1105. https://doi.org/10.1145/3065386

[79] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception architecture for computer vision. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2818–2826. https://doi.org/10.1109/CVPR.2016.308

Xception
[80] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions.

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1251–1258. https://doi.org/10.1109/CVPR.2017.195

VGG
[81]Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1409.1556

[82] J.O. Smith, & J.S. Abel, Bark and ERB bilinear transforms. IEEE Transactions on Speech and Audio Processing, 7(6), 697-708. 1999.

[83] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4700–4708. https://doi.org/10.1109/CVPR.2017.243

[84] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. https://doi.org/10.1109/CVPR.2016.90

[85] S. Abu-El-Haija et al., “YouTube-8M: A Large-Scale Video Classification Benchmark,” Sep. 2016, Accessed: Jun. 02, 2022. [Online]. Available: http://arxiv.org/abs/1609.08675.

[86] Musarrath, S., Subramanyam, K., Shaik, K., Anudeep, P., Deepthi, D., & Chandra Sekhara Rao, M. V. P. (2024). Content-Based Music Video Recommender System Using Cosine Similarity. In F. M. Lin, A. Patel, N. Kesswani, & B. Sambana (Eds.), Accelerating Discoveries in Data Science and Artificial Intelligence I (Vol. 421, pp. 123-135). Springer, Cham.

[87] Raschka, S., & Amor, D. A. (2023). Feature Scaling Through Scikit-Learn Pipelines. In Scikit-learn 1.5.1 documentation.

[88] Smith, J. (2020). Confusion matrix and performance measures visualization. Machine Learning Insights. Retrieved from https://www.machinelearninginsights.com/confusion_matrix

[89] J. Doe, "ROC curve demonstrating classifier performance," Data Science Tutorials, 2021. Available: https://www.datasciencetutorials.com/roc_curve

[90] Shukla, S. K. (2021, October 29). Voice Data Classification using Deep Learning. Paper presented at the International Conference on Machine Learning and Applications (ICMLA), San Diego, CA.

[91] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, “Audio set: An ontology and human-labeled dataset for audio events,” in Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on. IEEE, 2017, pp. 776– 780.

[92] Sayigh, L., Daher, M. A., Allen, J., Gordon, H., Joyce, K., Stuhlmann, C., and Tyack, P. The watkins marine mammal sound database: an online, freely accessible resource. In Proceedings of Meetings on Acoustics, volume 27. AIP Publishing, 2016.

[93] S. Schneider, A. Baevski, R. Collobert, and M. Auli, “wav2vec: Unsupervised pre-training for speech recognition,” arXiv preprint arXiv:1904.05862, 2019.

[94] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model agnostic meta-learning for fast adaptation of deep networks. In International Conference on Machine Learning, pages 1126–1135. PMLR, 2017.