AN ABSTRACT OF THE THESIS OF

Daniel De Leon for the degree of Master of Science in Computer Science presented on June 6, 2022.

Title: Automatic Classification of Ultrasonic Harbor Porpoise Clicks in a Varying Noise Environment.

Abstract approved:

_____

Patrick J. Donnelly

This study compares three approaches in the design of an autonomous machine listening agent that predicts harbor porpoise ultrasonic echolocation clicks in diverse noise environments. Considering the temporal variations of noisy coastal ocean soundscapes which the harbor porpoises inhabit, we propose a leave-one-day-out (LODO) cross-validation strategy in the training of a random forest classifier that successfully addressed the covariate shift present in our time-series data. To evaluate the efficacy of our approach to capture signals in this noisy environment, we compare three preprocessing approaches and two deep learning architectures on our harbor porpoise click data. We find that feature extraction strategies of mel frequency cepstral coefficients (MFCC) and short time Fourier transform (STFT) outperformed our novel approach, the heterodyned-Teager-Kaiser Energy Operator (TKEO), which shifts down the ultrasonic signal to a lower frequency in the time domain. Building on these results, we seek to improve the robustness of our porpoise click classifier for a real-world environment by implementing a second-stage stacked random forest ensemble on combinations of subsets of 42 deep learning base models that were trained from the folds of our LODO cross-validation and the three preprocessing approaches that were explored in this study. Our results demonstrate that experiments using the LODO cross-

validation strategy reported a difference between the average fold accuracy and a held-out test accuracy of 6%, while training without cross-validation and the equal *k*-fold cross-validation reported a 28.7% and 30.4% difference, respectively. From the three preprocessing approaches we implement, the models trained on MFCC produced the highest accuracy of 95.6% on the held-out test set while those trained on STFT and heterodyned-TKEO produced accuracies on the same held-out test set of 88.7% and 85.0%, respectively. Results from our stacked random forest show the greatest improvement in accuracy of 5.6% in the heterodyned-TKEO models while the STFT and MFCC models reported 4.5% and 1.9% improvements in accuracy, respectively. Highly varying noise environments are common across coastal areas inhabited by harbor porpoises. This study, with our proposed ensemble of different feature and model architectures, emphasizes the need to overcome such shifts in noise to design a robust porpoise click classifier that is ready for real-time deployment and able to generalize to all real-world conditions.

Automatic Classification of Ultrasonic Harbor Porpoise Clicks in a Varying Noise Environment

by
Daniel De Leon

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 6, 2022
Commencement June 2022

Master of Science thesis of Daniel De Leon presented on June 6, 2022

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering & Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Daniel De Leon, Author

ACKNOWLEDGEMENTS

I want to express my sincerest gratitude to my advisors Dr. Patrick Donnelly and Dr. Dave Mellinger for all the help and guidance throughout my MS studies and research. I came to OSU during the summer fires and COVID and I could not have had a better team to continue pushing me and enabling me to grow. None of this research would have been possible without your knowledge, wisdom and expertise. I am honored to have had this amazing opportunity to work closely with the both of you.

I also want to thank my parents and my brother for always being there for me and being the strongest foundation in my life. No estaría aqui haciendo lo que mas me gusta sin ustedes. Son mi mayor inspiración y luz de mi vida.

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

Table Page

LIST OF APPENDIX TABLES

DEDICATION

I want to dedicate this to my nephews, Dani and Diego. The both of you are my biggest source of inspiration. Your strength and resilience energize and motivate me to keep growing and staying curious. I hope to be the best example that I can be for the both of you, as an uncle and as a friend. It doesn't have to be listening to porpoises but I hope that the both of you grow to follow your curiosities and experience and explore the beauty of cosmos.

1. Introduction

1.1 Motivation

The Bay of Fundy, in Nova Scotia, Canada, hosts some of the strongest tidal currents in the world, ranging from 13 to 38 feet throughout the year [1]. As a result of these currents, this bay has been targeted as one of the best locations to harvest tidal energy via tidal turbines [2]. The bay, however, also hosts a large population of harbor porpoises. Unfortunately for harbor porpoises, and for the environment that they play a key role in, these tidal turbines are a serious physical threat that can hit porpoises passing by with their blades. Additionally, these turbines produce a significant amount of mechanical noise that could potentially and significantly interfere with the ultrasonic echolocation they use for foraging and communication [3, 4].

There is a need to establish a reliable acoustic system that can detect the presence of porpoise clicks. This system could (1) combine with previous statistical investigations to further investigate the impact that tidal turbines have on porpoise populations and porpoise click activity, and (2) potentially provide an opportunity for a real-time porpoise click detector that can switch off tidal turbines as soon as porpoise clicks were detected near the vicinity. Both use cases serve to benefit the well-being of the harbor porpoise by managing and mitigating our human impact on their environment.

1.2. Problem Statement

This investigation focuses on effectively classifying harbor porpoise vocalizations (clicks) recorded in Nova Scotia, Canada. More precisely, our objective is to create an autonomous machine listening agent that can predict if a harbor porpoise click exists in a given clip of audio. To solve this problem, we investigated whether a deep learning approach could detect these clicks and compared the performances of two deep model architectures: a convolutional neural network (CNN) and a long short-term memory (LSTM) recurrent neural network (RNN). However, since this time-series data contains high levels of background noise, this study also experiments with different cross-validation approaches to best represent our models' potential performance in

real-world conditions. Lastly, we leverage the CNN and LSTM model's individual predictions by constructing an ensemble method which resulted in a final classifier with better performance than any individual model itself.

1.3 Contributions and Novelty

In recent years, various approaches to autonomous classification of bioacoustic sound events have been explored and will be discussed in detail in Chapters 2 and 3. This study adds to these contributions in three ways. First, considering that background noise is always a difficult factor in marine soundscape, this study proposes a leave-one-day-out cross-validation approach over our limited observation data, introduced in Section 5.1, that attempts to evaluate how our autonomous classifier might perform in and generalize to real world situations. Second, we introduce an approach of heterodyning of ultrasonic audio as a preprocessing approach prior to the input of deep learning models. By shifting the high frequency clicks down to lower frequencies, discussed further in Section 5.2.3.4, we significantly reduce the dimensionality of input data when training deep learning models on raw audio signals. Lastly, from the 42 classifiers that are trained from a combination of approaches (three different preprocessing approaches, two deep learning models and seven folds from the cross validation), we train a second stage random forest ensemble model from variations of the output of each model and demonstrate a significant improvement in performance over any individual architecture and feature approach.

2. Preliminaries

This section provides context about the different domains that are discussed in the course of this investigation. The three relevant domains that are reviewed in this section are porpoise bioacoustics, marine mammal signal processing, and machine learning.

This section is organized into three subsections, each introducing one of these domains. Section 2.1 explains the characteristics of porpoise clicks. Section 2.2 reviews marine mammal bioacoustics and elaborates on the current best known practices in signal processing of cetacean echolocation clicks. And finally, section 2.3 introduces machine learning with a focus on time-series analysis.

2.1. Harbor porpoises and their clicks

Phocoena phocoena, commonly known as harbor porpoises, are cetaceans, sharing the same infraorder as whales and dolphins. Like other toothed cetaceans, harbor porpoises rely on echolocation to hunt and navigate in the marine environment. Similarly to dolphins, harbor porpoises emit ultrasonic clicks with the intention of hearing echoes that bounce off of their prey and other objects, providing information such as size, structure, material composition and shape of objects [5, 6]. This ability is known as echolocation or biological sonar. Figure 2.1.1 shows an illustration of a harbor porpoise.

Figure 2.1.1. Illustration of an adult harbor porpoise courtesy of NOAA Fisheries. Adult porpoises weigh between 135 and 170 lbs and range between 5 and 5.5 ft in length [7].

Compared to most dolphins, however, harbor porpoises emit clicks at a much narrower bandwidth, ranging between 6 to 26 kHz at a center frequency between 130 and 140 kHz [3]. Thus,

harbor porpoise clicks are classed as narrow-bandwidth high-frequency (NBHF) clicks. Other smaller cetaceans, including dolphins in the genus Cephalorhynchus and dwarf sperm whales, also emit NBHF clicks. Experts currently believe that species developed NBHF clicks from selection pressure to avoid predation from killer whales [3, 8, 9]. Harbor porpoise click durations last between 44 and 115 μs with a variable interclick interval between 30 and 100 ms. It has been reported, however, that when porpoises are within two meters of their prey, they can have an inter-click interval as short as 1.5 ms [3]. Figure 2.1.2 shows a spectrogram image of a typical train of clicks from a harbor porpoise.

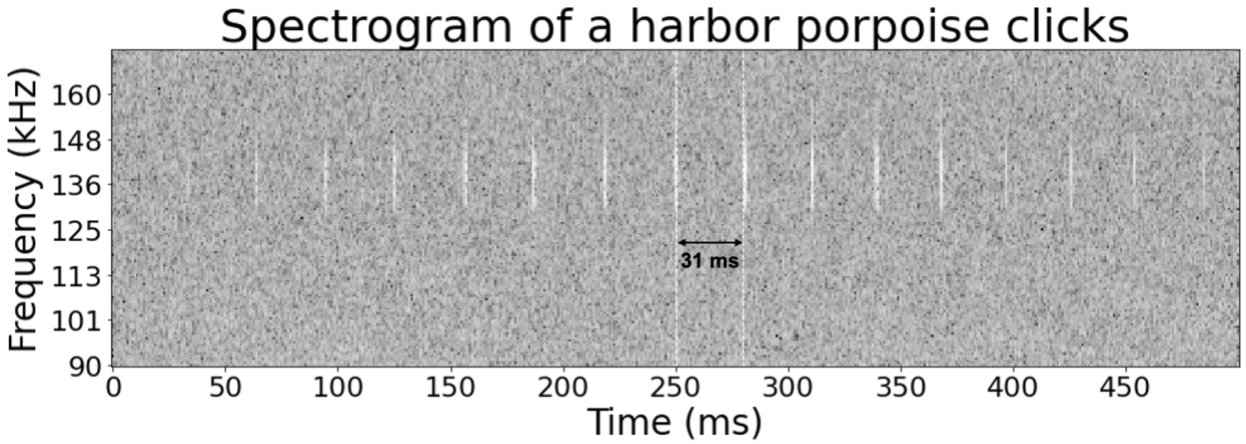

Figure 2.1.2. Grayscale spectrogram image of a harbor porpoise click train. The y-axis represents the linear frequency in kHz (where only the 90-170 kHz band is visualized for clarity) while the x-axis is time in milliseconds. The color corresponds to the dB magnitude of the sound. The two white, dashed, vertical lines highlight the interclick interval between two clicks at 250 and 281 ms, showing an interval time of 31 ms.

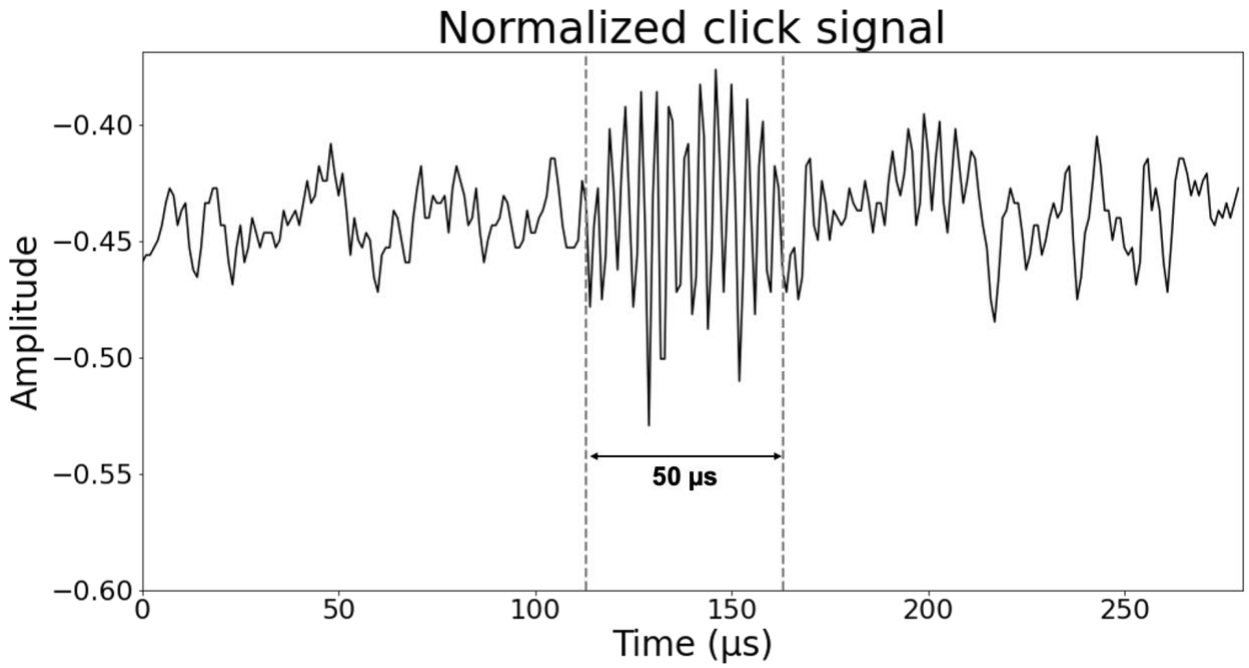

Figure 2.1.3. Waveform of a harbor porpoise click. The y-axis is the amplitude recorded from a hydrophone, normalized across the half-second window shown in Figure 2.1.2. The x-axis is time microseconds. The vertical lines at times 113 and 163 µs highlight the estimated duration of one click. This is the same click at time 250 ms in Figure 2.1.2.

Sørenson et al. [10] identified two different reasons harbor porpoises make clicks: communication and foraging. After tagging six harbor porpoises, Sørenson et al. concluded that although inter-click intervals varied slightly, the acoustic characteristics such as duration, centroid frequency and bandwidth were the same in both cases.

Although harbor porpoises navigate and survey coastal waters via ultrasonic echolocation, Miller et al. in 2013 identified that environmental noise, such as rain and ship noise, at much lower frequencies has shown to interrupt their foraging behavior [3]. In 2019, Tollit et al. observed via generalized estimating equations that harbor porpoise click activity significantly dropped when turbines were operational in Minas Passage off the coast of Nova Scotia, Canada [4] – the same location where our hydrophone data was collected.

2.2. Audio processing and feature extraction

There have been many signal extraction approaches that have been applied to marine mammal bioacoustics. In this investigation, we chose to use three of the most popularly used approaches for echolocation click detection for the input into deep learning architectures. In this section, we provide background for those three methods: short-time Fourier transform (Section 2.2.1), Mel frequency cepstral coefficients (Section 2.2.2), Teager-Kaiser energy operator (Section 2.2.3). We will also introduce heterodyning because we used it in combination with the Teager-Kaiser energy operator (Section 2.2.4).

2.2.1. Short-time Fourier transform

In order to discuss the short time Fourier transform (STFT), it is first necessary to establish the discrete Fourier transform (DFT). The DFT is defined to operate on a signal *x* by the following equation:

$$\hat{x}(k) = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi nk/N}$$

Here N is the total number of samples in the given signal and n is the sample number in the range [0,N-1]. Variable *k* is the wavenumber that corresponds to the discrete frequency 2**π**k/N that is translated from Euler's formula: $e^{ix} = \cos(x) + i \sin(x)$. The output of the DFT, $\hat{x}(k)$, is consequently a one-dimensional spectral vector of Fourier coefficients that represents the signal *x* in the frequency domain. Although the DFT is a great way to observe the frequency characteristics of a given audio signal, it fails to provide any information with respect to time.

The STFT was subsequently developed after the DFT to address this limitation by having with temporal dimension in addition to the spectral vector. The STFT performs the DFT over a sliding frame throughout the signal. The STFT is defined to operate on a signal *x* by the following equation:

$$\hat{X}(m,k) = \sum_{n=0}^{N-1} x[n+mH] \cdot \omega[n] \cdot e^{-j2\pi nk/N}$$

Observe that the STFT extends the DFT with the new variables m and H; m ranges from [0, M-1] where M is the total number of frames chosen to represent the signal throughout time, and H represents the hop size which is the distance, in samples, between each consecutive frame. This allows for frame overlapping. The new factor, $\omega(n)$, is a windowing function that is also not part of the DFT. This function can simply be just the rectangular function that controls what samples to perform the STFT on within each frame. The Hann or Hamming window functions are the most common windowing functions that serve to remove spectral leakage between spectrum coefficients.

Ultimately, the result of the STFT over the entire signal, x, is a two-dimensional matrix where each column corresponds to a frame, m, and each row corresponds to a frequency bin, k. From this STFT matrix, audio applications such as bioacoustics use the STFT to create a spectrogram image where each pixel value corresponds to each element, $\hat{X}(m, k)$, in the STFT matrix. An example of a spectrogram was shown previously in Figure 2.1.2.

2.2.2. Mel frequency cepstral coefficients

The name mel is abbreviated from melody and its calculation, as will be described in this section, aims to better align the human perception of pitch with the frequency of audio. Mel frequency cepstral coefficients (MFCCs) result from a few extra calculations on the output of the STFT described in the previous section. Figure 2.2.2.1 shows the process of extracting MFCCs from the time-domain audio signal. The rest of this section will walk through each of these steps after the STFT.

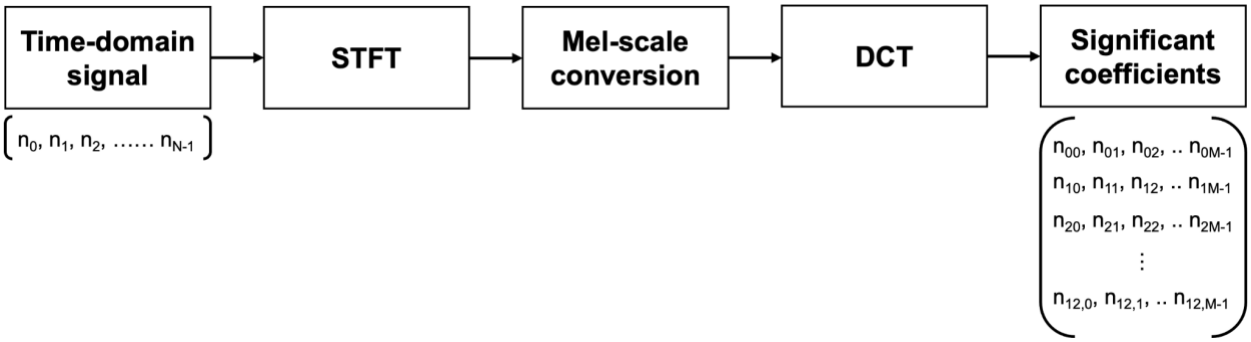

Figure 2.2.2.1. Block flow diagram of the MFCC extraction calculation on a clip of continuous audio. Each of these steps after the STFT is discussed in detail below. The calculation starts with a 1-dimensional array representing a time-domain signal and then outputs a 2-dimensional matrix of 13 discrete cosine transform (DCT) corresponding to the final MFCC.

Starting with the output of the STFT, the next step is to convert each $\hat{X}(m,k)$ into the mel scale. The relationship between mel and frequency is often defined with the following equation:

$$mel = 1127 \cdot ln(1 + f/700)$$

The constants, 1127 Hz and 700 Hz, were found experimentally via rigorous psychological surveys [11]. To convert the STFT matrix to the mel-scale filter banks, we first must convert the lowest and highest frequency in the STFT matrix to mel values. Then, divide the difference of those two mel values into an evenly spaced amount. These divisions are called mel bands and it is common practice to create 13 mel bands, although different numbers have been explored for different applications. From each of these mel bands, we then locate the center frequency converting the mel value back to frequency value with inverse of the above equation:

$$f = 700(e^{m/1127} - 1)$$

From this frequency, we then round to the nearest frequency bin in the STFT matrix. With the center frequency defined for each band, a triangular filter is then applied, starting from the center frequency of the previous band and ending with the center frequency of the proceeding mel band. Figure 2.2.2.2 shows an example of 6 mel bins and how the triangular filter is applied corresponding to the relative frequency bins.

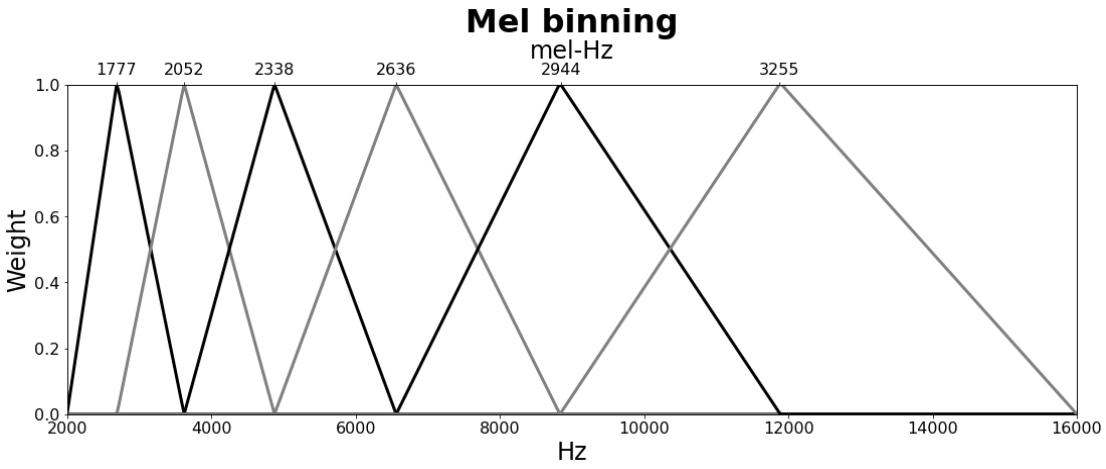

Figure 2.2.2.2 Example of a mel filter bank with 6 mel bins between 2,000 to 16,000 Hz. The x-axis is linear frequency. The bottom x-axis shows values in Hz while the top x-axis shows the values in mel-frequency. The y-axis is the weight corresponding to the triangular filter. As we see an increase in linear frequency, the base of the triangle filters increases logarithmically, which is how the mel spectrum attempts to mirror human pitch perception.

What results from this conversion from the STFT matrix to the mel scale is a matrix of m rows corresponding to each mel band, and the same number of columns as the STFT that correspond to each frame. Now that the signal is represented in the spectral domain, the discrete cosine transform (DCT) is calculated to transfer the signal over to the cepstrum domain. The DCT is similar to the inverse Fourier transform but is used here because it only calculates real-valued coefficients. The DCTs coefficients correspond to the amplitudes of the cosine frequencies that are present in the spectrum signal, and are then the resulting MFCCs. Traditionally 13 cosine signals are extracted, although other numbers can be used. This results in a final matrix of 13 rows, corresponding to the 13 DCT coefficients, and M columns, corresponding to the number of frames used defined in the initial STFT calculation as seen in Figure 2.2.2.1.

2.2.3. Teager-Kaiser energy operator

The Teager-Kaiser energy operator (TKEO) [12] is a signal conditioner that was first popularized on electromyography (EMG) signals [8]. It is now used in various applications from mechanics to image and audio processing. Its discrete equation, operated on a signal x, is defined as

$$\Psi[x(n)] = x^2(n) - x(n+1)x(n-1)$$

where n is the discrete sample number in *x*. The TKEO conditioner measures the instantaneous changes in energy at each sample by taking the difference of the squared value of the current sample and the product of the previous and the following samples in the signal. Figure 2.2.3.1 shows an example of TKEO being applied to audio that contains with harbor porpoise clicks.

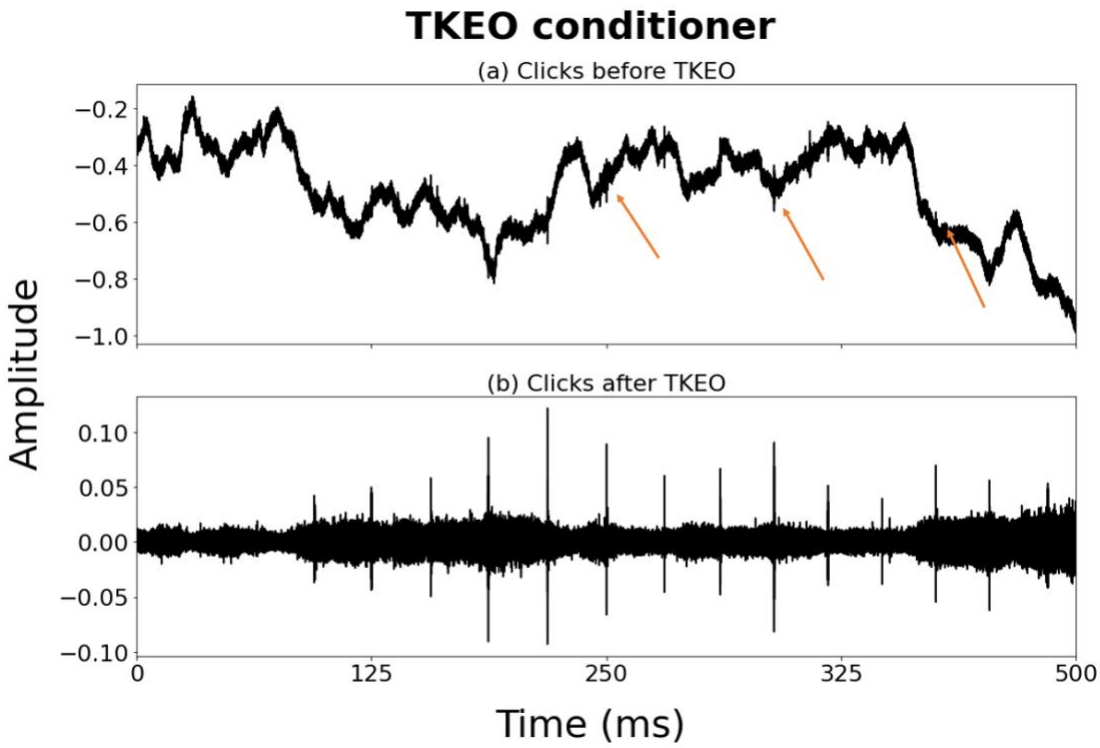

Figure 2.2.3.1. These two plots, show an audio signal before (a) and after (b) the application of the TKEO. The three arrows correspond to three examples of clicks within a train of harbor clicks. The TKEO significantly reduces the noise while also increasing the amplitude of each harbor porpoise click.

2.2.4. Heterodyne

Heterodyning is a signal processing method used for many applications in communications. At a high level, heterodyning shifts a frequency band into another band of equal width. For instance, by shifting ultrasonic frequencies down to the audible frequency band, heterodyning makes it possible for bat bioacousticians to hear bat echolocation clicks in real time, since a heterodyne bat detector shifts the ultrasonic bat calls down to the 20 Hz - 20 kHz range of human hearing.

Digital heterodyning can be broken down into three main steps. The first step is to perform a band pass filter on the frequency band of interest that is intended to be shifted. This removes all energy except that which resides in the frequency band of interest. The second step is then to shift that frequency band to the new frequency band, which in most cases would be starting at 0 Hz, by multiplying it by a sine wave whose frequency is the lower end of the original frequency band of interest. Lastly, a low pass filter is then executed to remove the energy produced from the copy of the heterodyned signal. To help visualize this process, Figure 2.2.4.1 shows a spectrogram of the same audio clip in Figure 2.1.2 after heterodyning from the 100-150 kHz band down to the 0-50 kHz band.

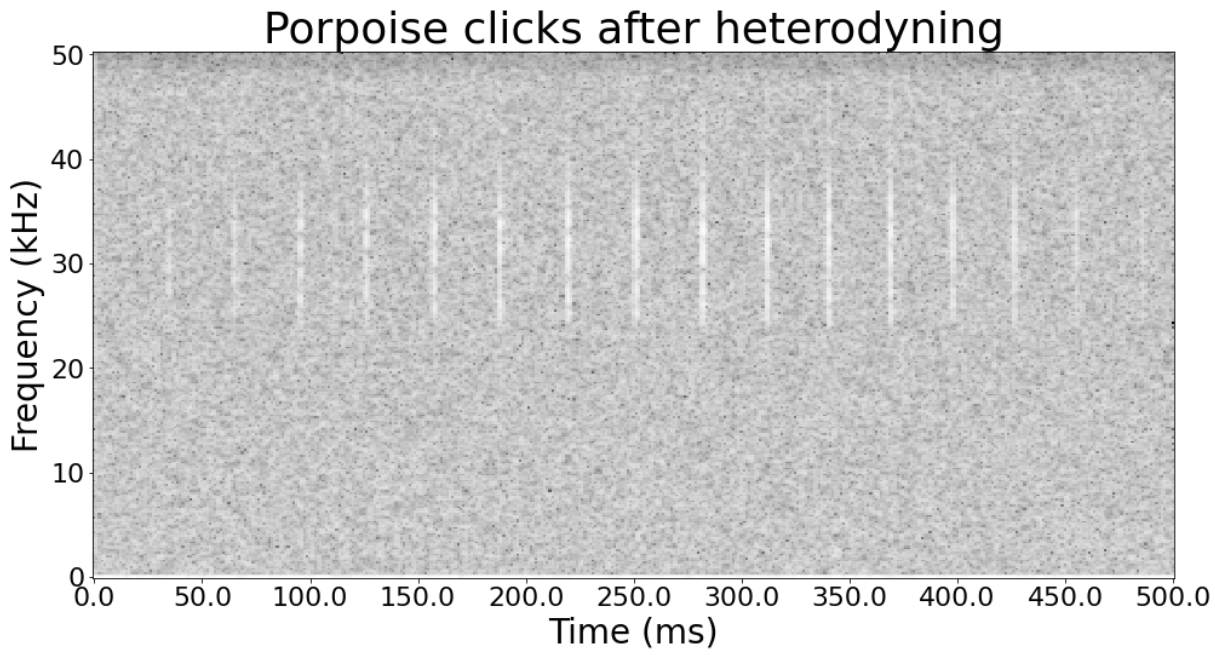

Figure 2.2.4.1. Spectrogram of porpoise click train after heterodyning. The x-axis remains the same as Figure 2.1.2 but the y-axis has now changed to frequencies from 0 to 50 kHz. We see that the heterodyne operation manages to maintain the clarity of the clicks while effectively reducing the sample rate from 512 kHz to 100 kHz.

2.3. Machine learning, CNNs and LSTMs

Before introducing the specifics of the Deep Learning (DL) approaches that are used in this study, it is important to first contextualize them within the broader field of machine learning (ML). ML algorithms predict or make decisions without domain knowledge embedded in the algorithm or explicit direction from the inherent design. Instead, ML algorithms are trained on large amounts of past data to make predictions/classifications about future data. This training on posterior data is how the algorithm learns, hence its name. Starting from the 1960s, many different ML models have been successfully developed and deployed to solve real-world problems. Some of the most popular models include decision trees [15], perceptron models [16], logistic regression models [17], naive Bayes models [18] and support vector machines [19].

Of the many different ML models, decision trees have been popularized not only for their success in ensembles, known as random forests [20], but they remain relevant in recent

conversations and studies about ML interpretability [21] due to their simplicity, interpretability, and feasibility to visualize. The importance of individual features of a random forest (RF) are identified by calculating the total criterion reduction for each feature in all trees of the RF. The features that result with the greatest total sum of criterion reduction have the most impact, or in other words, are the most important features for inference.

Building on its origins in the linear perceptron model, the multilayer perceptron model (MLP) or feedforward artificial neural network has been well established as an important ML algorithm to learn from non-linearly separable data [22]. As a supervised ML model, MLPs learn via multiple iterations of forward and back propagation. In forward propagation, input data passes through each neuron as a weighted sum and then transforms it with an activation function. In backpropagation, the weights that contribute to the summation of each neuron are updated via gradient descent [23]. It was not until the implementation of the rectified linear unit activation function [24] in MLPs, however, that ultimately led to the advent of deep learning (DL) – a subdomain of ML [25].

CNNs, similarly to MLPs, have input, hidden and output layers. The input, however, is an *n*-dimensional matrix that is usually of the form (height, width, channels). These matrices are often called tensors to abstract away from dimensionality. Instead of the standard perceptron model applied to each neuron in the MLP, CNNs perform convolutions at each layer, typically followed by a max pooling layer to reduce the tensor size while simultaneously maintaining important features. The more times this convolution-pooling pair is repeated, the "deeper" the CNN becomes. Typically, after all of the convolution-pooling layers, one to three MLP layers are connected to the end to feed to the final output. These final layers in a CNN are called fully connected layers because each neuron is connected to every element in the previous layer. For example, if there is one fully connected layer in the CNN, then the feature map before it is flattened and then each element is fed to each neuron. The purpose of the fully connected layer(s) is to learn the combinations of non-linear features that the convolutional layers extracted. A CNN with convolution-pooling layer is shown in Figure 2.3.1.

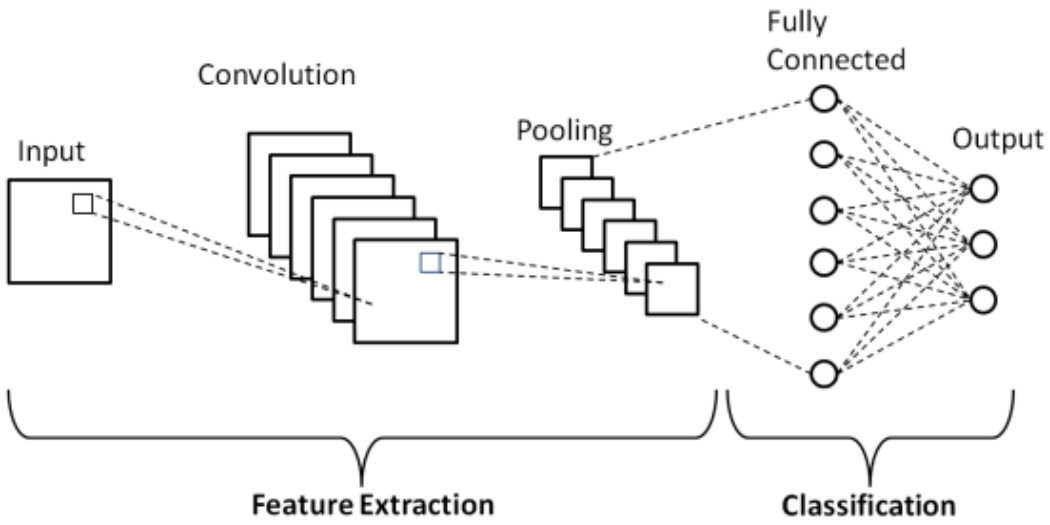

Figure 2.3.1. A simple CNN architecture (courtesy of [26]). This CNN has one convolutional layer. Notice how the height and width of the tensor remains the same before (the initial input) and after the convolution. The second layer is a pooling layer. Notice the reduction in height and width of the tensor – it is keeping the most important features that are being learned from the previous convolution layer. The pooling layer is then flattened and input into the fully connected layer. This layer learns the non-linear combinations of features extracted from the previous convolution-pooling layer and outputs the result to 3 nodes corresponding to three classes. The value at each node is the probability of the input example being that class.

Backpropagation in a CNN is executed the same on the final fully connected layers' weights and is also applied to the weights of each convolution filter. CNNs are most commonly chosen for image classification problems. In classical computer vision algorithms, a particular filter is usually rigorously tuned to effectively extract certain information from an image. With CNNs, however, various combinations of filters are learned and calibrated autonomously through the training of the CNN and have shown to greatly outperform classical computer vision approaches in image recognition problems [27]–[29].

Recurrent neural networks (RNNs) are also an extension of the MLP in that they have the same neural structure, but uniquely they introduce sequential propagation from one neuron (or grouping of neurons) to the next within the same layer. In other words, information from one sample (or grouping of samples) is applied to all that proceed it. These groupings of samples are

known as timesteps of the sequence. At each timestep, the standard perceptron operation is performed on the samples with a `tanh` activation. The output of each timestep, however, is then concatenated with the input of the proceeding timestep. This identifies sequential patterns in each observation passed through the RNN, making it effective in classifying and predicting temporal data. Figure 2.3.2 shows an example of an RNN with three timesteps.

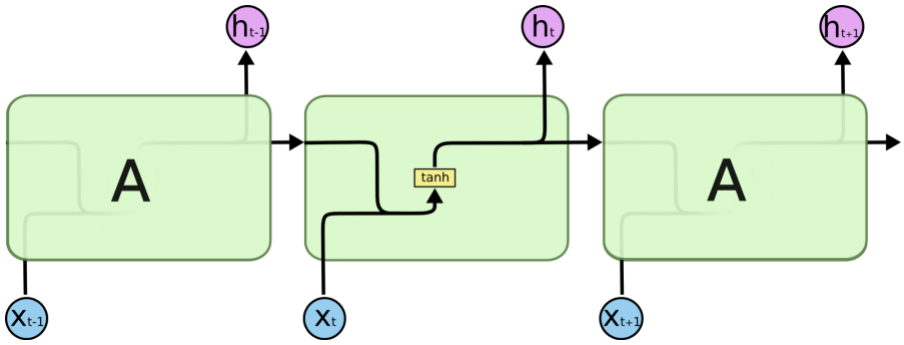

Figure 2.3.2. Example of an RNN with three timesteps (courtesy of [30]). Each input element X_t has its own RNN timestep. Note that the output of a timestep is then propagated into the proceeding timestep. RNNs can also be visualized with one timestep that feeds back to itself with every sample. This particular RNN outputs the results at each timestep.

However, over time it has been observed that RNNs struggled as increasingly longer sequences were processed. Often, important characteristics that occurred near the beginning of an observation were lost towards the end of the series of timesteps. This had made RNNs notorious for only having short-term memory [31]. Long short-term memory (LSTM) models were then introduced to attempt to solve this issue [32]. LSTMs introduce three more neural network layer operations within each timestep (with sigmoid activation) as well as another tensor that propagates through each timestep called the cell state. This cell state learns through training what information from each timestep to keep and discard, thus enabling “long-term memory”. Figure 2.3.3 shows an example of three LSTM timesteps.

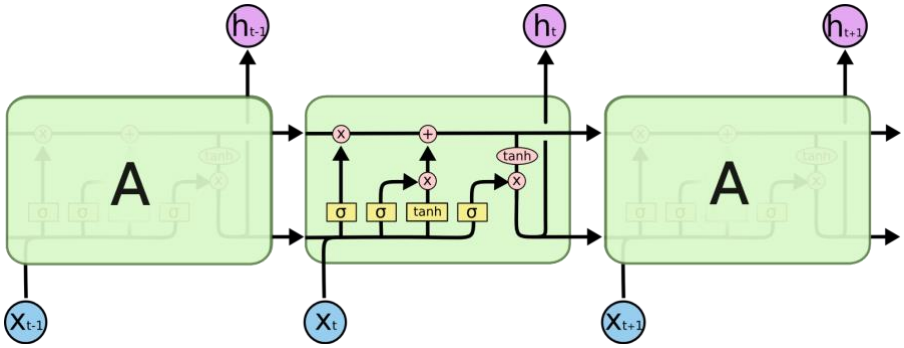

Figure 2.3.3. Example of LSTM with three timesteps courtesy of [30]. The cell state is the tensor running at the top of the timesteps. Notice the pointwise addition and multiplication of the cell state tensor at each timestep.

Both the CNN and LSTM frameworks are commonly employed on a variety of tasks today. In the following chapter, we will discuss such employments in bioacoustics and Chapter 5 will then introduce how we implement our own variations of the CNN and LSTM in this work.

3. Related work

In this research we attempt to find an approach to effectively and automatically classify ultrasonic harbor porpoise clicks present within a given audio clip that may also contain high amounts of environmental noise. This chapter surveys the approaches that have been used to solve similar problems. Although marine mammal bioacoustics have come a long way from the days of traditional signal processing approaches for event detection, in this chapter we review the various feature extraction techniques commonly used in bioacoustics and how they evolved to help machine learning and deep learning models learn to classify bioacoustic phenomena.

Section 3.1 provides an overview of the related work involving feature extraction and processing of acoustic signals. In recent years, machine learning (ML) and deep learning (DL) approaches have far surpassed classical approaches that relied heavily on the low-level feature extraction methods. Sections 3.2 and 3.3 will be dedicated to the contributions made in ML and DL (where ML is referring to all ML approaches that are not DL). Lastly, section 3.4 will summarize the entire chapter and provide context for the approaches that are explored further in this study.

3.1. Feature extraction

Prior to the rise of machine learning, bioacoustics and many general acoustic applications relied heavily on the fine-tuning of various feature extraction techniques. These feature extractions are often measured in both the time and time-frequency domains (where DFT is utilized).

In 1992, Fristrup and Watkins surveyed the most prominent extraction techniques on several marine mammal species' signals [33] such as upsweep mean, median frequency mean and amplitude frequency correlation. From their summary, some of the feature extraction calculations only required information in the time series signals, such as amplitude mean and standard deviation. Many other calculations, however, require not only time series data but their corresponding frequencies as well, such as the section of extraction methods in frequency modulation and both bandwidth sections.

In 2019, Mcloughlin et al., in the broader domain of bioacoustics, discusses the most common feature extraction methods used today [34]. These include some of the extraction methods introduced in table 3.1.1, and variations of them, such as the centroid frequency (similar to center frequency) and spectral flux (similar to frequency modulation). They also add zero crossing rate and spectral flatness as well as other low-level extraction methods that are still currently being used in bioacoustic classification.

In particular for cetacean clicks, the TKEO has been shown to significantly improve signal detection [14, 35, 36]. In 2006, Kandia et al. implemented the TKEO on sperm whale clicks before inputting into a forward-backward search algorithm. Their results show that TKEO provided a 30% increase in detection rate.

The difference between sperm whale clicks and harbor porpoise clicks, however, is that porpoises vocalize at a much higher frequency (100-150 kHz). This results in having an impractical amount of data to input directly to the LSTM due to the high sampling rate required to capture the ultrasonic information. Mirzaei et al. [37] had a similar dimensionality problem with ultrasonic bat calls and used MFCCs to represent the audio at a significantly reduced dimensionality. Their study showed a 10% increase in accuracy with respect to a fast Fourier transform preprocessing approach. Another method gaining popularity in the research literature in the implementation of passive acoustic monitoring (PAM) of bat echolocation is the process of heterodyning the input signal. Ingemar Ahlén states in his paper titled *Heterodyne and the time-expansion methods for identification of bats in the field* that "[heterodyning] is perhaps the best system to detect the most bats. The narrow frequency band that is transposed to audible sound must be tuned to the sounds made by the bat" [38] [39].

In the same way that heterodyning is used to bring ultrasonic bat calls down to audible frequencies to hear and detect, in this study we utilize heterodyning to shift the ultrasonic porpoise clicks down to audible hearing so that we can reduce the sample rate, which has the effect of reducing the number samples in each observation to feed to the neural network models.

3.2. Machine learning

Before discussing recent advances in deep learning of bioacoustic signals, it is important to review the prior successes of non-DL algorithms in the area of bioacoustics. Returning to Fristrup and Watkins' efforts in summarizing the best practices in marine mammal feature extraction, discussed in the previous section, a follow-up study published a year later (1993), titled "Marine Animal Sound Classification" [40], used decision trees with a feature vector including various outputs of low-level feature extraction calculations mentioned in [33]. These decision trees were implemented to combine the strengths of the individual low-level feature extraction algorithms.

Years later, many bioacoustic classification studies successfully implemented random forests, which are built upon many decision trees (as discussed in Section 2.3) [41, 42]. Similar success has also been reported with other machine learning models such as logistic regression [43], support vector machines [44] and multilayer perceptrons [45].

More specifically with cetacean clicks and pulses, studies such as Caruso et al. [46] show that ML algorithms are still effective in addressing detection of cetacean species. A feature vector including pulse duration, peak frequency, centroid frequency, bandwidth, Q_{RMS}, zero crossings and inter-pulse interval was used to train a cubic support vector machine (SVM) to detect Indo-Pacific humpback dolphins. Using five-fold cross-validation with a total of 51,238 examples, their SVM model reached a test accuracy of 89.9%. Consentino et al.'s PorCC model [47] is another ML approach that instead builds on the industry standard, PAMGaurd's [48] click detector, and utilizes a combination of a variation of an if/else tree and logistic regression coefficients to detect harbor porpoise clicks off the west coast of Scotland. The PorCC model uses the output of the PAMGuard Click Detector and then performs a series of logical selection statements combined with two logistic regression equations to detect if the signal was in fact a click. PorCC outperformed PAMGuard's classifier with an overall sensitivity index 3.4 vs. 2.0, respectively.

3.3. Deep learning

Building on the success of ML models mentioned in the previous section, recent DL models have also provided significant contributions to marine mammal bioacoustic detection and classification problems. Because of the advances in DL, and more specifically convolutional neural networks (CNN) and long short-term memory (LSTM) recurrent neural networks (RNN),

low-level signal processing and feature extraction approaches have been superseded by the short time Fourier transofrom (STFT) and mel-frequency cepstral coefficients (MFCCs). Shiu et al. [49] explored five industry standard DL models (LeNet [50], BirdNet [51], VGG [28], ResNet [52] and a Conv1D + GRU) to detect right whale up-calls. Their results showed that LeNet, the most shallow of the pure CNNs, "had the highest precision and lowest number of false positives for a given recall." It is important to note that BirdNet performed almost as well as LeNet but it was a more complex model that took longer to execute, showing that shallower, less complex models are suitable for detecting many marine mammal bioacoustic events. Other approaches that have begun to implement deep learning have also seen a lot of success and not only in classification of the existence of the particular species but in identifying the individual mammals that are making the clicks. Berment et al. [53], for example, used spectrograms with a CNN to successively classify sperm whale clicks but used an LSTM with time series audio to classify short repeated patterns of clicks (known as codas), vocal clans and individual sperm whales.

The next two subsections, 3.3.1 and 3.3.2, will review related works using strictly CNN and LSTM for bioacoustic classification, respectively.

3.3.1. CNN

With many variations and improvements over the years, CNNs have been successfully applied to solve many complex image classification problems [28, 54, 55]. Therefore, it is not surprising to see CNN architecture has also found success in bioacoustics tasks when combined with STFTs to create spectrograms.

Thomas et al., for example, use two staple CNNs, ResNet-50 and VGG-19, to classify three different types of whales, ambient and non-biological noise in a bay in Nova Scotia, Canada [56]. Using STFT spectrograms (with interpolation) as inputs, they see an accuracy of 94.4% and 95.9%, for the ResNet and VGG, respectively. Another study, by Duan et al., classifying chirps and whistles of dolphins used a custom CNN [57]. This model achieved a precision and recall of 91% and 84%, respectively. Given the size of this model, and the models used in Thomas et al., it further demonstrates that CNN models are effective for this classification task when they have a relatively small number of layers.

3.3.2. LSTM

Given their popularity in analyzing time-series data [58], LSTMs have also been given significant attention in bioacoustic research. For example, Gong et al. compare the performances of a deep neural network and LSTM on the classification of 35 different amphibians [59]. With MFCCs as input, their LSTM architecture consists of one LSTM layer with 10,240 timesteps that output to 35 predictive targets for each amphibian species. They concluded that the LSTM outperformed the deep neural network (an MLP with multiple hidden layers) showing greater precision and also a reduction in training time. Weninger and Schuller also compared an LSTM against variations of support vector machines and hidden Markov models in classifying biological orders including songbirds, non-songbirds, cats, dogs and primates [60]. Again, with MFCCs as input, the LSTM performed the best of all the classifiers when classifying two biological orders of the five. However, when the models were tasked to classify all five orders, the hidden Markov model outperformed the LSTM by 2.3%.

In regards to cetacean vocalization specifically, there have been creative ways of implementing LSTMs. Duan is one such that, after processing the spectrogram, implements a line detector and a Frangi filter for preprocessing before inputting into an LSTM [61]. He compares this proposed method with the traditional TKEO approach established by Kandia et al. [14], discussed in section 3.1. The LSTM in Duan's approach has a variable length input such that the beginning time step corresponds to the lowest frequency of the line. The last time step corresponds to the highest frequency of the line. Given this 1-dimensional feature vector of STFT values corresponding to a detected line in the spectrogram, the LSTM classifies it as one of three beaked whales that are observed in the dataset. Duan finds that the LSTM with this unique preprocessing approach outperforms the TKEO algorithm showing higher precision and recall at lower signal-to-noise ratios.

3.4. Summary

The previous sections highlighted the various contributions to the problem of autonomous marine mammal bioacoustic classification. Starting with low-level feature extraction approaches summarized by Fristrup and Watkins in the early 90s, many of these approaches that found success are still being used today in the ever-changing environment of DL. In an approach similar to Mirzaei et al., we perform a comparative analysis of the different preprocessing approaches that were used in the aforementioned studies (MFFCs, STFT spectrograms and heterodyning with TKEO). Along with this comparison of preprocessing approaches, we compare the performance of a stand-alone CNN and LSTM to empirically compare their performance with different input data abstractions. Building on top of how these stand-alone models perform, we then analyze the performance of the ensemble on all the models (LSTM and CNN each with the three preprocessing approaches) to observe how they behave together. Similar to PorCC and Fristrup et al.'s implementation of decision trees, we stack a random forest on top the outputs of our individual DL models to utilize their individual strengths.

4. Data

The previous chapter reviewed the current state of the art approaches to cetacean and marine mammal bioacoustics. In this chapter we introduce the harbor porpoise data used in this study. This chapter is composed of three sections. Section 4.1 discusses all that was involved in the collection of data. Section 4.2 enumerates the processing steps performed on the collected data. Lastly, Section 4.3 summarizes all of our data by providing quantitative and qualitative observations.

4.1. Data collection

All of the audio used in this study was collected by the Fundy Ocean Research Center for Energy (FORCE) via their deployment of a JASCO Applied Sciences Autonomous Multichannel Acoustic Recorder (AMAR). This study's [62] goal was to assess the operational limitations relative to other passive acoustic monitoring (PAM) instruments when attempting to detect harbor porpoise clicks. The AMAR hydrophone was mounted on a subsea platform and deployed in Minas Passage in the Bay of Fundy, Nova Scotia, Canada. Figure 4.1.1 shows a seafloor elevation map where the platform was deployed. Figure 4.1.2 shows a map showing clearly where the Bay of Fundy is relative to the rest of Nova Scotia. Figure 4.1.3 shows the AMAR hydrophone on the platform before it was deployed.

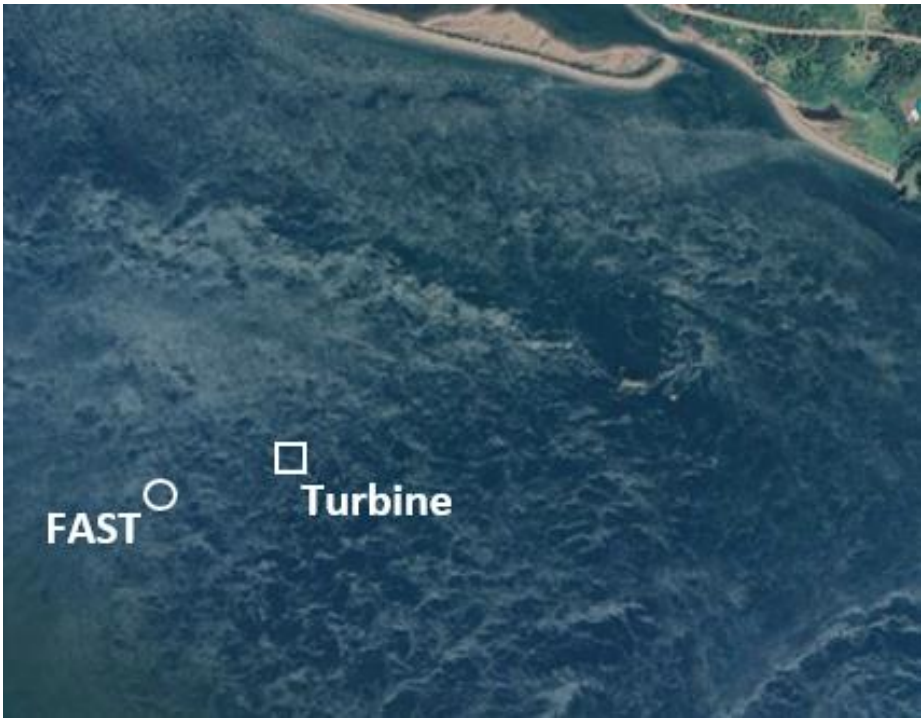

Figure 4.1.1. Map of the AMAR (courtesy of Map data Google, ©2022 CNES / Airbus) hydrophone deployment location. The circle is where the Fundy Advanced Sensor Technology (FAST) platform that carried the hydrophone is located. The square is the location of a nearby water turbine that was never operational during the collection of the audio.

Figure 4.1.2. Map of the coast of Nova Scotia (courtesy of [70]). The Bay of Fundy covers all of the region covered in light blue.

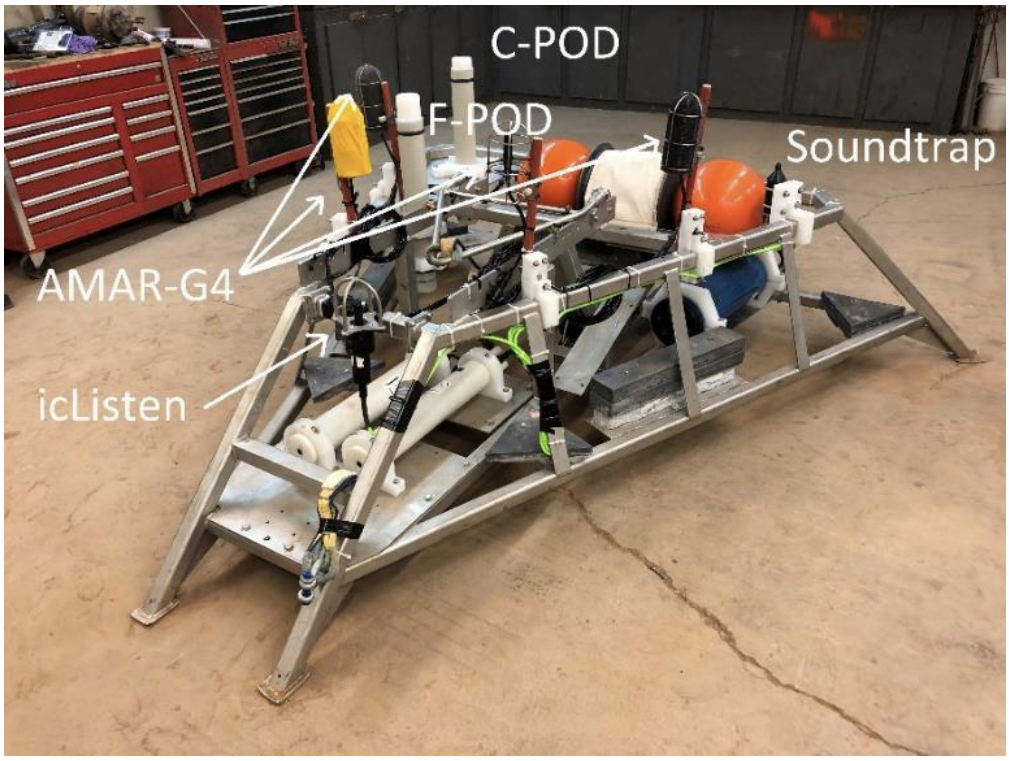

Figure 4.1.3. The AMAR hydrophone (courtesy of [62]) connected to the platform before deployment. The four arrows associated with the AMAR are pointing to its four channels. The remaining four devices are other PAM instruments used in the deployment to compare their effectiveness in collecting harbor porpoise audio.

There were a total of two deployments in 2019 in which the hydrophone recorded continuously, from June 4th to July 29th and September 5th to September 13th, resulting in a total of 47 days of audio. The hydrophone recorded at a sample rate of 512 kHz to fully capture the porpoise clicks that exist between approximately 110 kHz and 150 kHz (although energy from clicks can occasionally extend below and above these limits.

4.2. Data harvesting and processing

This continuous audio stream was stored as two-minute segments (henceforth referred to as “segments” for the rest of this work). From these segments, individual clicks were harvested.

Figure 4.2.1 shows a long term spectral average (LTSA) plot of a specific segment in which clicks were harvested.

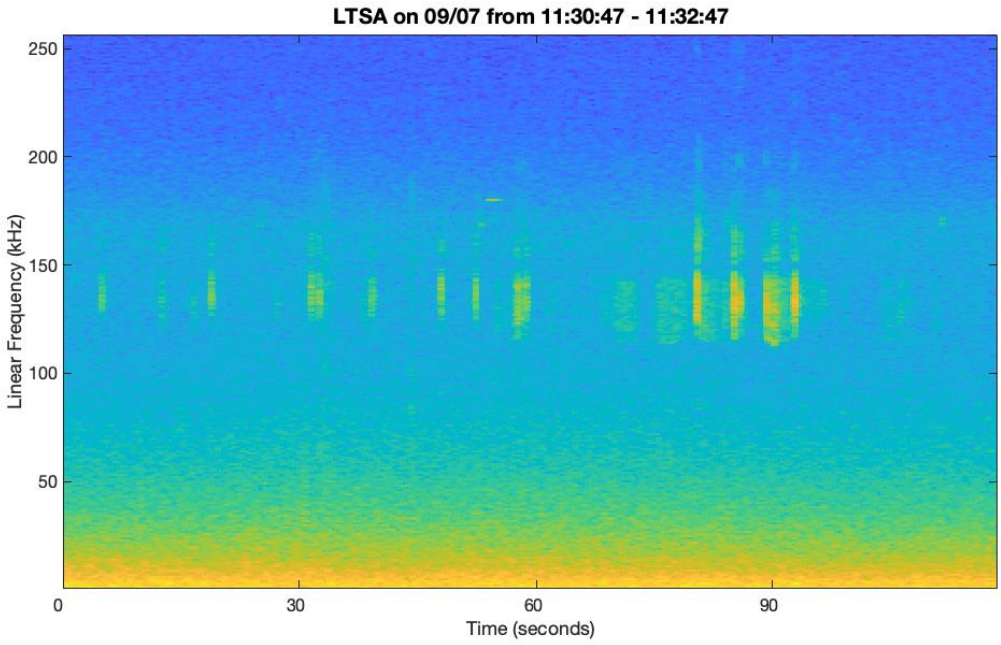

Figure 4.2.1. LTSA of a segment in which a total of 557 click examples were collected. Notice that higher energy pulses existing between 110 and 160 kHz. These are the result of a high amount of harbor porpoise click activity.

Of the 47 days of audio recorded from the AMAR, we selected eight days to label harbor porpoise clicks. From these eight days, 4,143 clicks were collected from a total of 25 segments. We used the energy detector in the bioacoustic analysis software Ishmael [63], via Matlab, to collect these clicks. The Ishmael porpoise detector ultimately detects possible clicks when the ratio of the energy of the porpoise click frequency to energy in a lower frequency band (below 110 kHz) is greater than a detection threshold chosen heuristically so as to have a false-negative rate of less than 5%. Based on these detections, timestamps were generated and were manually checked (by D.K. Mellinger and S. Fregosi) to confirm they were in fact clicks. With these timestamps, we then created a half-second window centered on each of these clicks. We defined these half-second windows to be the positive examples in the dataset because they contain clicks in them. These

positive examples belong to the `click` class. Each one of the eight days had varying numbers of click examples and were harvested from varying numbers of segments.

We defined the negative class, called `noClick`, when there is not a click present. The Ishmael detector was again applied to identify noise peaks but this time on segments that had been manually confirmed to not have clicks in them [62]. From these segments, which were from the same set of days as the `click` segments, the same number of `noClicks` were harvested resulting in a balanced dataset of `click` and `noClick` examples. These `noClicks` were harvested from segments near the click segments in an attempt to maintain consistent background noise. After building our final dataset, all the 8,286 `click` and `noClick` examples were then individually normalized to a level of -0.1 dB relative 0 dB with Sox [64].

4.3. Data analysis

Tables 1 and 2 in Appendix A summarize the profiles for the `clicks` and `noClicks` datasets, respectively, for the root-mean-square of all the samples (RMS_s) values across all 0.5 second observation windows that were collected. The RMS_s of each example was calculated with the following formula:

$$RMS = \sqrt{\frac{1}{n}\sum_{i} x_i^2}$$

Here, n is the number of samples in example x, which in our case is 256,000. Figure 4.3.1 shows the distribution of the RMS_s values for each day for `clicks` (top) and `noClicks` (bottom). We notice that two days stick out from the rest of the six days. Figure 4.3.2 presents a kernel density estimation (KDE) plot with Guassian representation and Scott bandwidth estimator that shows a clearer visualization of this phenomenon. Particularly for the clicks, all the days except 07/14 and 09/07 peak between an average RMS_s of 0.30 and 0.40. However, note that 07/14 has two peaks at around 0.35 and 0.70 and has a much wider distribution than the other days.

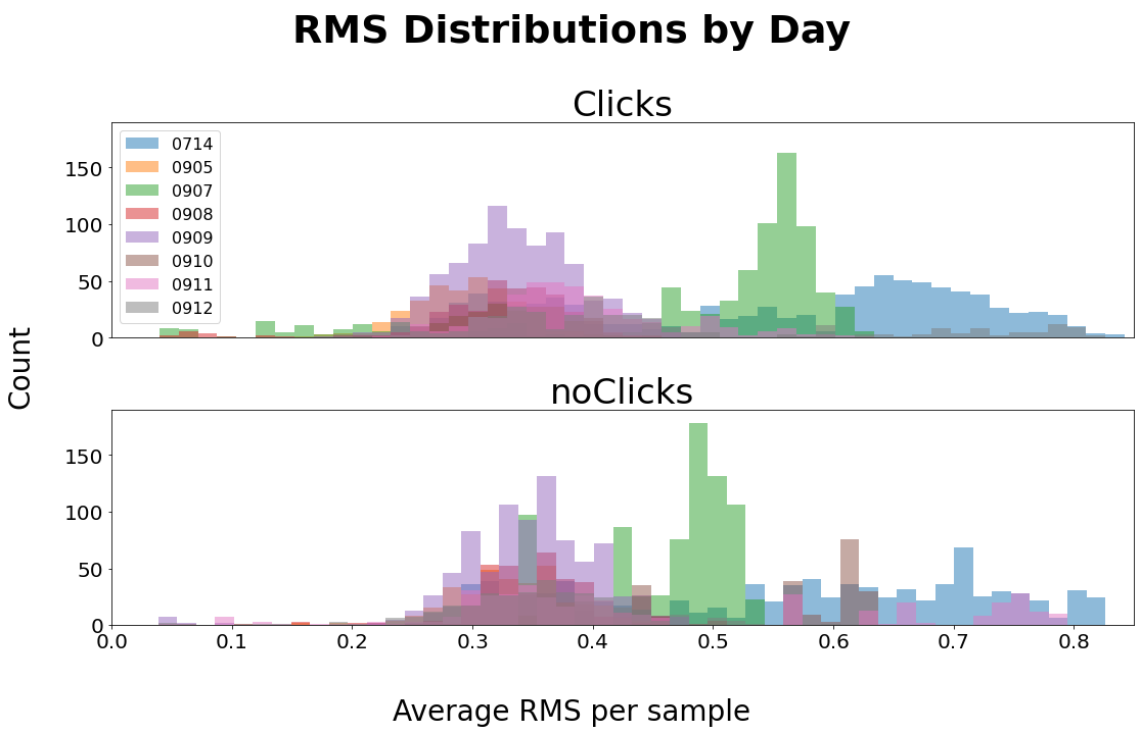

Figure 4.3.1. Histogram of RMS_s distributions of all examples broken up by day, over 40 bins. Notice the peak on 09/07 (green) and 07/14 (blue) deviate from the rest of the days.

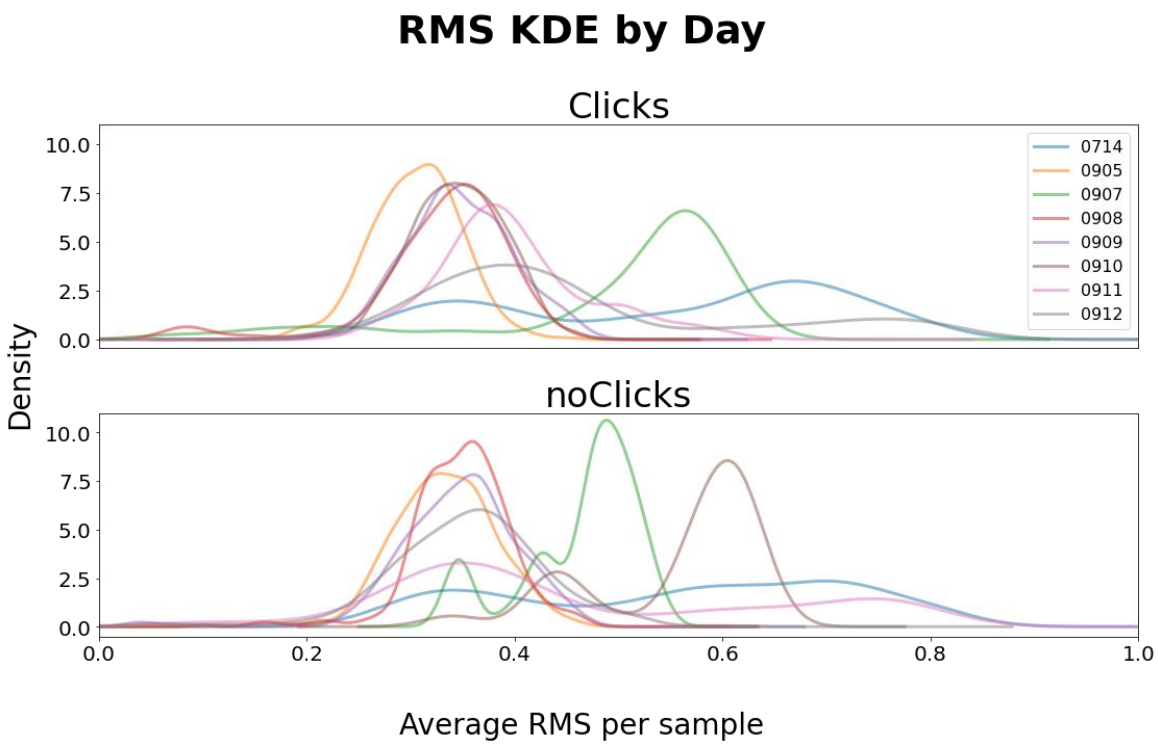

Figure 4.3.2. KDE Gaussian representation of the RMS_s distribution using `scott` bandwidth estimator. From this plot, it is clearer than Figure 4.3.1 that 09/07 and 07/14 have unique distributions for clicks that differ from other days. It is also visible now that 09/10 has another unique density distribution for `noClick`s but not for its clicks.

One possible reason why 07/14 is significantly different from the rest is that it was recorded almost two months before the other days. The noise environment could be different because of changes in tide trends and patterns during that time of the year. Given this significant covariate shift in our dataset, it is necessary for our system to learn how to classify porpoise clicks in various noise environments if we hope to generalize for deployment in a real-time setting.

Figure 4.3.3 shows the LTSA plots of each entire day, the locations of the selected segments, and ultimately the locations where the `clicks` and `noClicks` were harvested. From these LTSAs, we observe that the high noise regions (the yellow bumps) are smoother and the broadband noise pulses (yellow columns) are less prominent on 07/14 with respect to the rest of the days. From these LTSAs, we can qualitatively verify that we are gathering a diverse set of examples from segments across different noise regions types and times of day. For example, segments selected in 09/05 and 09/07 are in similar noise regions as we see an initial spike noise

where these segments exist. That background in those segments is significantly different from the segments in 07/14. The segments on day 07/14 at around hour 15 is during a relatively quiet period with regards to that background noise. With this variation in noise profiles, we hope to establish a dataset that effectively models the real world. This would lead to training robust DL models that may better generalize to new unseen data.

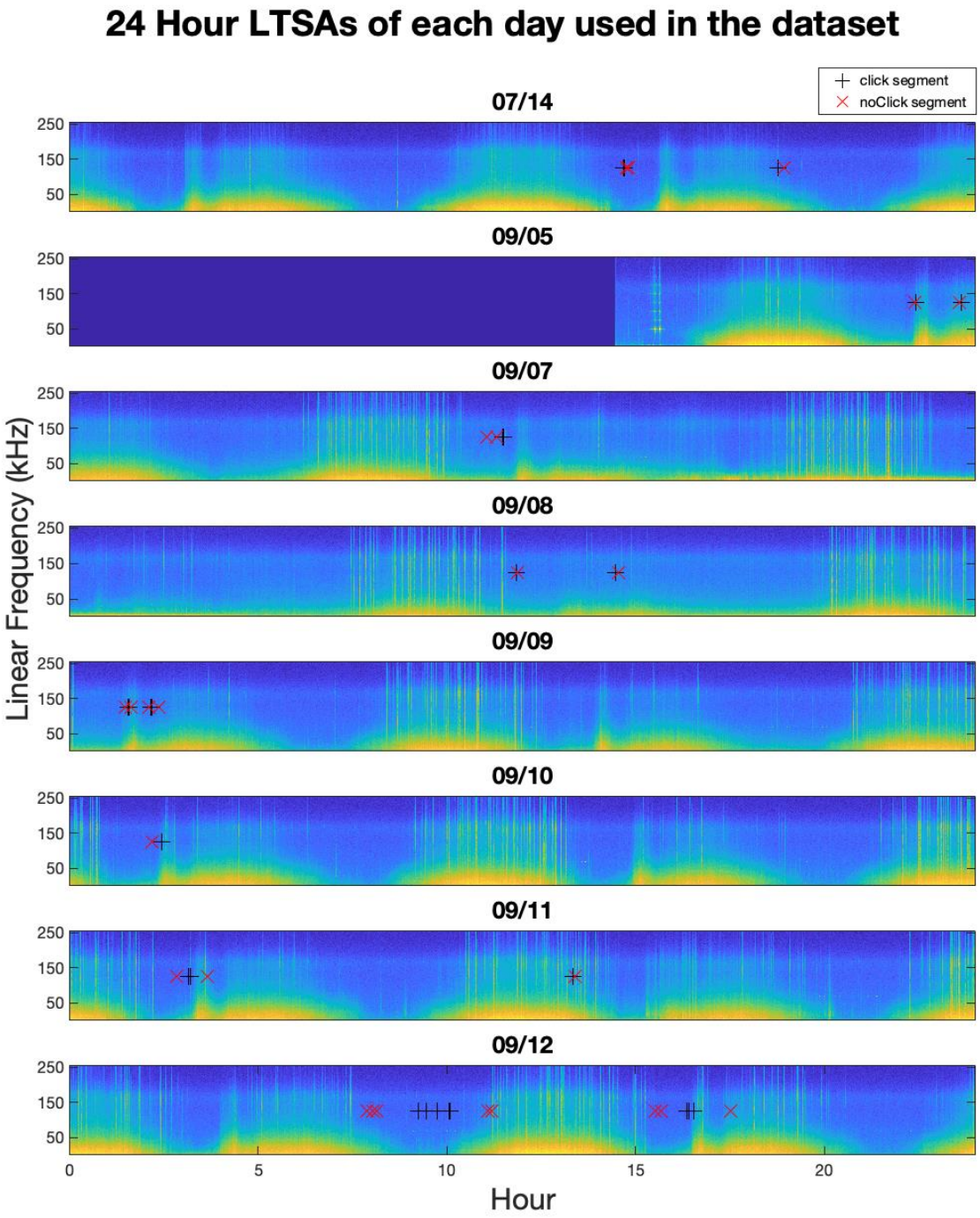

Figure 4.3.3. 24-hour LTSAs of each day with markers locating where the click and `noClick` segments exist. Note that 09/05 was not a complete day as it was the first day of recording for the month of September. The y-axis of each LTSA linearly spans from 0 to 256 kHz.

5. Experimental design and methods

In this chapter we present the experimental approaches taken in our exploratory investigation to address the porpoise click classification problem. Since our time series dataset, as described in Chapter 3, showed varying inter-day distributions of RMS_s sample values, Section 5.1 will describe the exploration and approach in deciding which cross-validation (CV) method to use for the remainder of our research. Section 5.2 then discusses the design and implementation of the CNN and LSTM models trained on datasets preprocessed using MFCC, STFT and TKEO. Lastly, Section 5.3 explains how we added a final, post-classification stacking ensemble step to further improve classification performance.

5.1. Establishing cross-validation with baseline ML

This section motivates how and why we chose to use cross-validation (CV) for the training of our classifiers in this study. It also provides initial baseline results for naively classifying harbor porpoise clicks. Section 5.1.1 introduces the ML model and feature extraction techniques. Section 5.1.2 then discusses the three different training approaches that are examined in the experiment as well as the general structure of the experiment. Section 5.1.3 then provides results followed by discussion of implications of these results.

5.1.1. Introduction

To better understand the challenges of the porpoise dataset while securing an initial baseline result for reference, we trained a random forest (RF) classifier on our dataset. The RF was built using the Scikit Learn library [65] and had the following parameters:

- 100 trees with bootstrapping
- Gini criterion
- no limits on leaves and height

The following feature extraction methods were used to generate a feature vector for each observation:

- amplitude envelope (AE)
- root mean square (RMS)
- zero crossing rate (ZCR)
- spectral centroid (SC)

These four low-level feature extraction algorithms were taken from the approaches described in Section 3.1. Each of these features was calculated over a sliding window across each half-second audio clip. There were a total of 19 frames with 50% overlap for each clip (frame size = 25,600 samples or 50 ms, hop size = 12,800 samples or 25 ms). A total of 19 frames were chosen because the porpoise's minimum inter-click interval is observed to be roughly 30 ms (see Figure 2.1.2). This meant that there are at most 17 to 18 porpoise clicks in a half-second audio clip observation (500 ms / 30 ms ≈ 17).

Within each of the 19 frames, each of the four feature extraction methods listed above were extracted across a smaller sliding window (frame size = 1,024 samples or 2 ms, hop size = 512 or 1 ms). Each of these smaller windows was centered, with zero padding to attain uniform length. The average across all of these smaller windows was calculated to produce a single average value for each of the 19 windows. Since we had 19 windows corresponding to every feature extraction method, we were left with a total of 76 features.

5.1.1. Experimental approach

Chapter 4 identified variable noise profile distributions in the observations across the eight days. These varying distributions across time are what cause covariate shifting. A covariate shift in data is when the distribution of data shifts between training and testing. Simply running a randomized test/train/validation split on the entire data, that we suspected to have covariate shift, could result in a misleading test performance. Because of this, it was imperative to implement a cross validation (CV) strategy to obtain the most reliable test results.

We experimented with two forms of CV and compared them against the RF without CV (i.e., a randomized held-out train/test split). The two CV approaches we experimented with are traditional k-fold CV and the other we refer to as “leave-one-day-out” (LODO) CV.

K-fold CV is when the entire dataset is randomly split into k subsets, referred to as folds. Each fold is used successively to train a classifier using the fold as the test set with remaining data as the training set. This results in k different iterations of training and k classifiers, each with unique test sets. For this experiment we chose 7 folds and shuffled our data prior to partitioning.

LODO is similar to *k*-fold CV in that there are unique test sets at each fold. Each of these folds of LODO, however, a single day is dedicated for testing, while the other remaining days are used for training. Through this process, a new classifier learns on a different set of days at each fold. Since we have eight days over our entire dataset, we have eight folds corresponding to each day that is held out for testing. We refer to running LODO on all eight days as 8-LODO.

Aside from these two CV approaches, we also implemented training without CV where we shuffled the data and then split it with 70% for training and 30% for testing. To compare these three different validation approaches, we selected one entire day to be the held-out day to test on. In other words, this day was not included in the CV methods nor was it seen in the traditional 70/30 split. For the LODO approach, this meant that we used 7-LODO (instead of the 8-LODO mentioned previously). We intentionally chose the day 07/14 to be this held-out day for two reasons: (1) First, it is almost two months apart from the rest of the data. This is important because one of the main reasons for covariate shift is significant changes over time (in this case, almost two months). (2) Also, a closer look at Figure 4.3.3 in chapter 4 reveals that 07/14 has the most distinctive RMS_s distribution among the rest of the days, with two prominent peaks at 0.35 and 0.70 average RMS_s per observation. The goal of this experiment is to observe the difference between the sampling method’s accuracy on its dedicated test set and its accuracy on the held-out day 07/14.

5.1.1. Results

To discuss our results more clearly, we visualize performance through confusion matrices to get a better understanding of the classifiers' performance. From the confusion matrices, we can calculate accuracy, precision, recall and F_1-score [66]. Figure 5.1.1.1 shows the test results as confusion matrices for the three different approaches. Table 5.1.1.1 shows the metrics for each confusion matrix. These test results are from only the seven days in September. For the traditional training without CV (labeled "No CV"), this means that 30% of the examples from September were randomly selected for testing. For the 7-fold CV, all the examples from September were shuffled and then seven unique test sets were selected. This means that every fold of testing was 1/7th the size of the dataset of September (n = 919 - 920). Because 7-fold CV trains seven distinct RFs, the results correspond to the summation of all of the folds. For 7-LODO, every day from September was used as an individual test set. Just like the 7-fold CV experiment, the test results were summed across all of the seven folds.

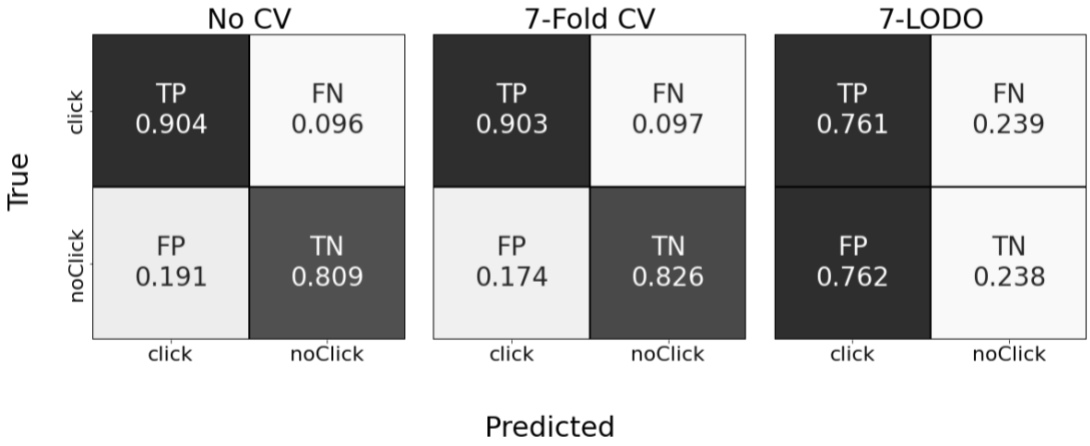

Figure 5.1.1.1. Results from the RF tests in September. These three confusion matrices compare test results of the three different training paradigms for the Random Forest model. No CV is tested on 30% of the examples in September resulting in n = 1,932. For both CV approaches, the summation of all fold results were calculated meaning n = 6,438.

	No CV	7-Fold CV	7-LODO	Mean
Accuracy	0.857	0.865	0.500	0.741
Precision	0.825	0.839	0.500	0.721
Recall	0.904	0.903	0.761	0.856
F1 Score	0.863	0.870	0.603	0.779

Table 5.1.1.1. Results from the RF tests in September. Each row corresponds to the results metric calculated from the confusion matrices in Figure 5.1.1.1

These results correspond to the test days that were trained in the same month of September. The following Figure 5.1.1.2, and its accompanying table 5.1.1.2, show the results and metrics for each training approach's performance predicting on the held-out day, 07/14. The same models that were trained and tested in September are now tested on the sole day in July. Since both of the CV approaches have seven RFs associated with them, their results are the summation of all their models that were trained on the seven days of September.

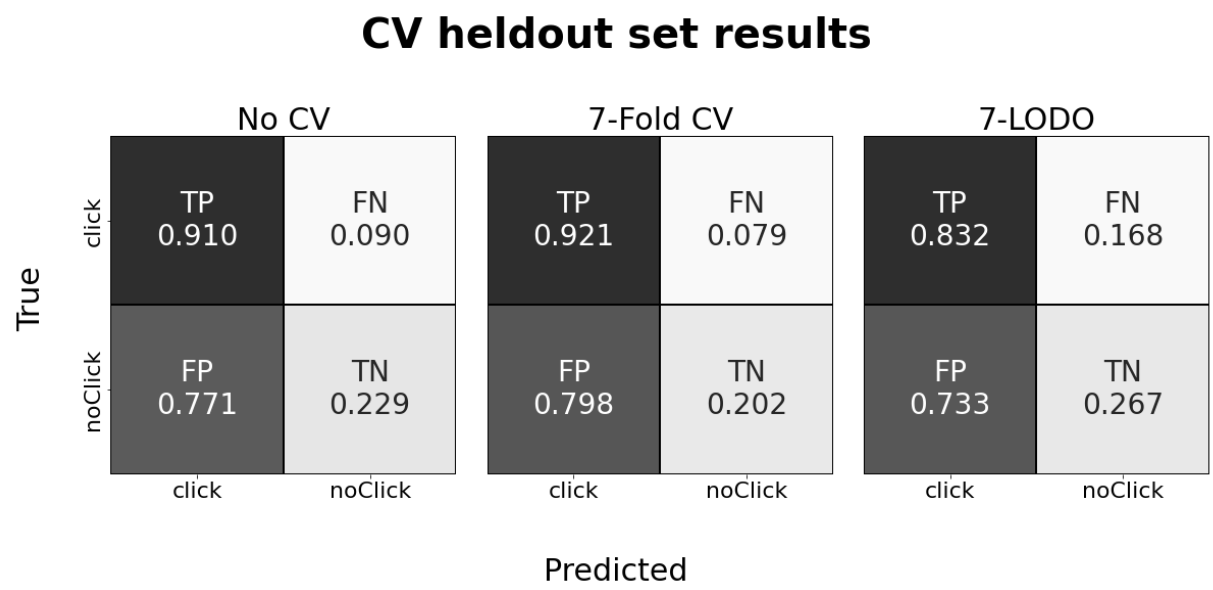

Figure 5.1.1.2. Results from the RF held-out test day in July, 07/14. These three confusion matrices compare test results of the three different training paradigms for the Random Forest model that was trained on days in September only. No CV only has one RF resulting in `n =` 1,848. For both CV approaches, the summation of all RFs results were calculated meaning `n =` 12,936.

	No CV	**7-Fold CV**	**7-LODO**	**Mean**
Accuracy	0.570	0.561	0.550	0.560
Precision	0.542	0.536	0.532	0.537
Recall	0.910	0.921	0.832	0.888
F1 Score	0.679	0.677	0.649	0.668

Table 5.1.1.2. Results from the RF tests in July. Each row corresponds to the results metric calculated from the confusion matrices in Figure 5.1.1.2.

5.1.1. Discussion

The best training method is defined to be the one that has the least difference between its test day results and the left-out day results. This ultimately translates to reliable test results that are generalizable to real-world harbor porpoise data. This is an important factor of our study because in comparing the DL architectures with the different preprocessing approaches, it is necessary that

we capture behavior that is consistent with new unseen acoustic data from a highly noise variant environment such as the Bay of Fundy.

Figure and Table 5.1.1.1 show that a RF with no CV (85.7% accurate, n=2,486) and 7-fold CV (86.5% accurate, n=8,286) greatly outperform 7-LODO (50.0% accurate, n=8,286) on their respective test data. From this we might be tempted to conclude that 7-LODO is the least successful training method of the three. Considering Figure and Table 5.1.1.2, however, we see a different outcome. Now, no-CV (57.0% accurate, *n* clicks=1848) and 7-fold CV (56.1% accurate, n=12,936) only slightly outperform 7-LODO (56.0% accurate, n=12,936) when predicting examples on the held-out day, 07/14.

We observed in, chapter 4, a covariate shift in the RMS_S distributions between 07/14 and the rest of the days in September. This variation of noise is to be expected in the ocean soundscape and conclusions made from this study must be derived from reliable training methods that accurately emulate behavior consistent in the real world environment.

With an accuracy difference between the test set and held-out set of 28.7% and 30.4%, it is clear that no-CV and 7-fold CV, respectively, greatly misled us from their September test results suggesting that the RFs are performing relatively well. 7-LODO, on the other hand, shows an accuracy difference of only 6% between the test and held-out results. Although 7-LODO performed slightly worse (~1%) than its competing training methods on the 07/14 examples, it is the most reliable training method for this dataset and is therefore used in the following experiments.

5.2. CNN and LSTM

This section is divided into three subsections to discuss our formulation of experiments involving DL to address the porpoise click classification problem. Subsections 5.2.1. and 5.2.2. describe the CNN and LSTM architectures, respectively. Subsequently, section 5.2.3. will elaborate on the three different preprocessing approaches that are applied to both DL models.

5.2.1. CNN Architecture

The CNN architecture we use is from a model implemented in an application [56] to predict the genre of music given an audio snippet as input [67] using Tensorflow-Keras python libraries.

We adapted this architecture for our task by removing an entire convolution-pooling layer to avoid overfitting on our small dataset as well as reducing the widths of the two final fully connected layers to 32 neurons and one neuron (for binary classification) with a final sigmoid activation function and a binary cross entropy loss function. Figure Figure 5.2.1.1 visualizes CNN architecture using the DL model visualizer tool, Netron [68].

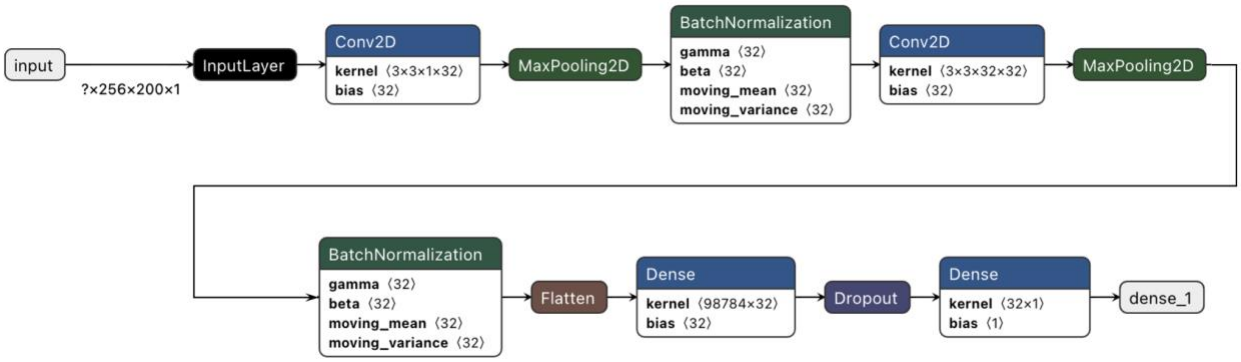

Figure 5.2.1.1. The CNN architecture. Input dimensionality varied for each of the preprocessing approaches mentioned in the next section.

5.2.2. LSTM Architecture

Similarly, we also adapted the LSTM from a music genre classification model [69]. We found a slightly better performance in increasing the width of the second-to-last fully connected layer to 256 neurons. Similarly to the CNN model, the last layer is a single neuron with the sigmoid activation function for classification with the binary cross entropy loss function. Figure 5.2.2.1 visualizes the LSTM, again, using the Netron tool.

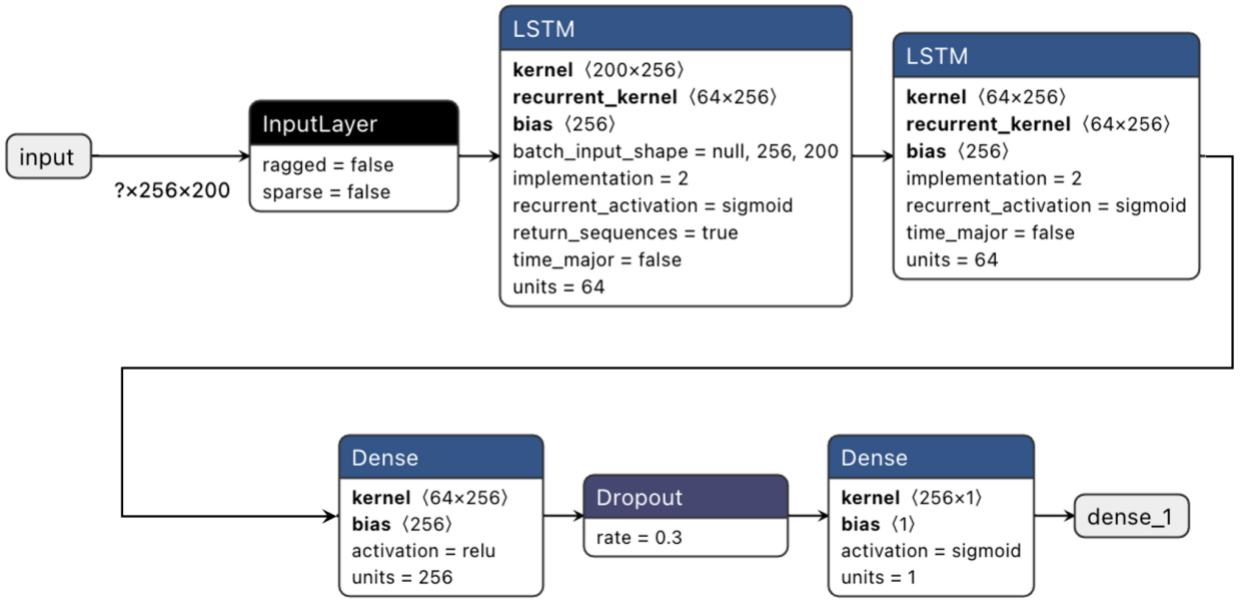

Figure 5.2.2.1. The LSTM architecture. The input dimensionality varied for each of the preprocessing steps.

5.2.3. Preprocessing

In a comparative analysis, three separate preprocessing approaches were applied to the data for input into the CNN and LSTM. To further discuss their implementation, this section will be divided into three subsections:

1. MFCCs
2. STFTs
3. heterodyned-TKEO

Our aim is to empirically compare the effectiveness of these approaches on the porpoise click signals. As mentioned in chapter 3, each observation is made up of 256,000 samples, which equates to 512 KB. Since this is a relatively large input for an LSTM to process, we are also considering how much these preprocessing approaches reduce the size of our data while at the same time representing the audio effectively. Figure 5.2.3.1 and 5.2.3.2 show examples for a click and a `noClick` for the three preprocessing approaches, respectively.

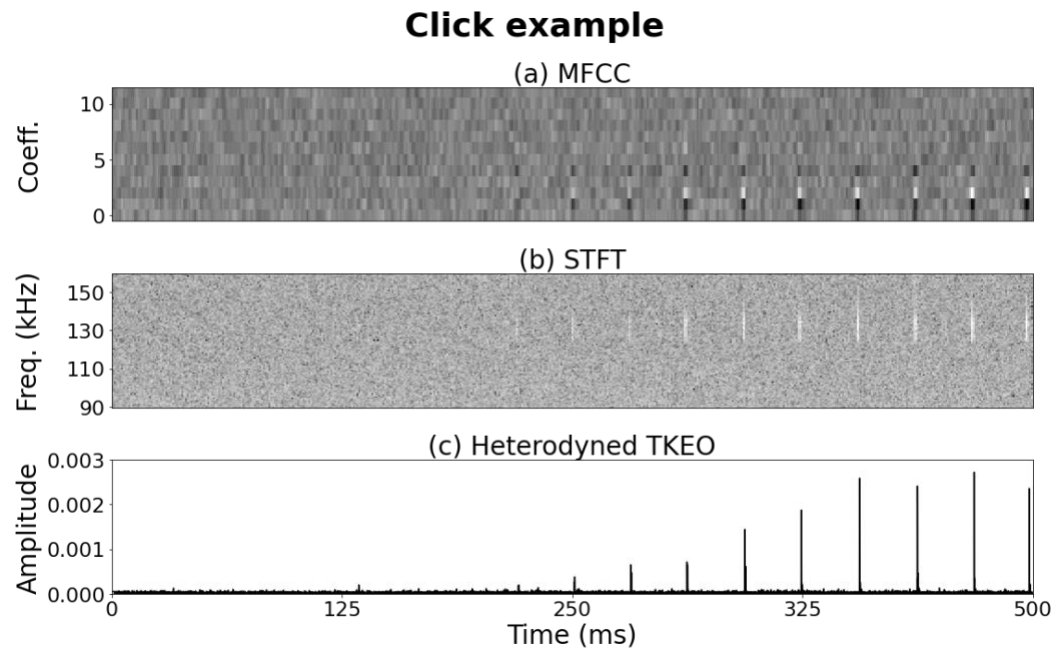

Figure 5.2.3.1. Click example shown for the three different feature sets: (a) MFCC, (b) STFT and (c) heterodyned TKEO.

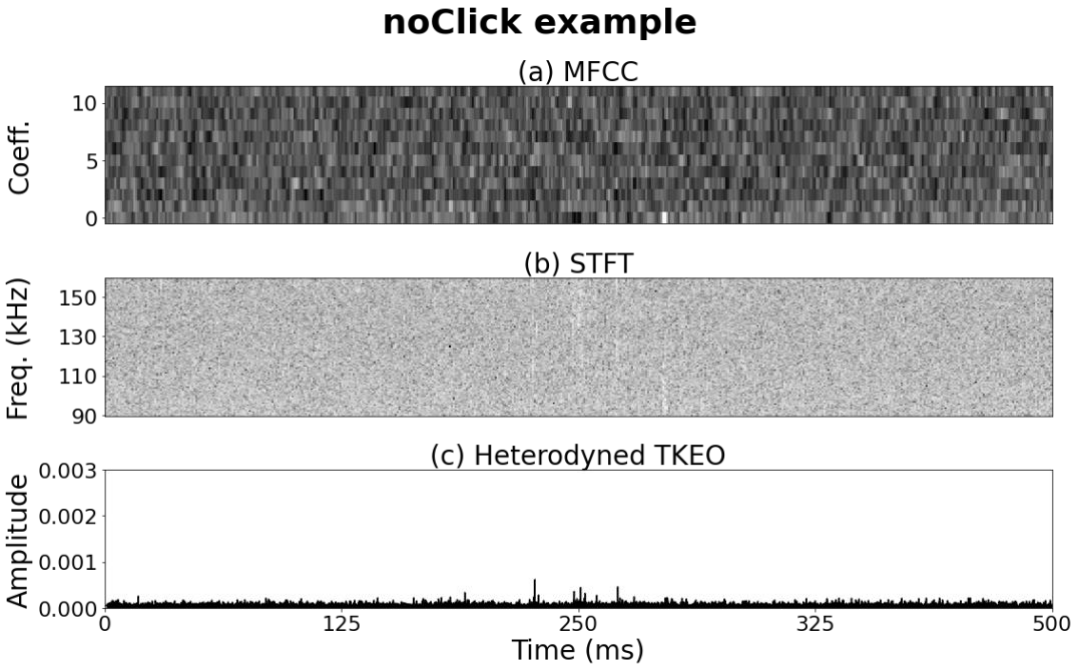

Figure 5.2.3.2. `noClick` example shown for three different feature sets: (a) MFCC, (b) STFT and (c) heterodyned TKEO.

5.2.3.1. MFCC

We generated the MFCCs using the Python Librosa library function, feature.mfcc. The MFCC is calculated on the normalized wav file examples, introduced in Chapter 4, and saved as a two-dimensional array. The MFCCs were calculated with the following parameters:

- 128 mels
- Mel conversion applied to 100-160 kHz frequency band
- STFT parameters:
    - 2,048 sample DFT frame size (frames are centered so the signal is reflection padded to accommodate the beginning and end frames)
    - 512 sample hop length
    - Hanning window (of the same length as the frame size)
- 13 MFCCs

These parameters result in a final MFCC matrix shape of (13, 501). The thirteen rows correspond to the final output of the DCT we defined and the 501 columns are the DFT bins that are generated at every 512 samples. The extra 501st bin is the result of the reflection padding for the frame centering used initially for the STFT calculation. This MFCC matrix results in a total of 6,513 values and a final file size of 26 KB. Figures 5.3.2.1.a and 5.3.2.1.b show grayscale visualizations of `click` and `noClick` examples, respectively. For the sake of clarity, the first coefficient is removed because it is a constant offset for the entire MFCC. Since this constant is significantly smaller than the rest of the MFCC values, it greatly impacts the normalization of the grayscale, leaving the remaining coefficients hardly visible.

5.2.3.2. STFT

Similarly to the MFCCs, STFTs were calculated using Librosa's STFT function. Given a normalized half-second waveform from our porpoise dataset, the STFT returns a complex valued two-dimensional array that represents the audio in the time-frequency domain. The STFT parameters are almost identical to the STFT parameters used for the MFCC with the exception of the frame size being 1,028 samples instead of 2,048, as the smaller frame size, from visual

inspection, resulted in a clearer image when visualized with a spectrogram. This results in a two-dimensional array of the shape (515, 501).

After calculating the STFT matrix, it is then converted to decibels via Librosa's `power_to_db` function so that we can see the units of the STFT on a relative log scale. This function uses the following equation:

$$\hat{X}_{dB}(m,k) = 10log_{10}(\hat{X}(m,k))$$

After converting the STFT to decibels, only the matrix rows representing the 100 - 150 kHz band are kept. This results in a reduced matrix shape of (140, 501). With 70,140 values, the saved file representing each observation as an STFT is 275 KB. Figure 5.2.3.1.a and 5.2.3.2.b both visualize a click and `noClick` example as an STFT spectrogram.

5.2.3.3. Heterodyned-TKEO

This preprocessing approach differs significantly from the previous two approaches in that it stays in the time domain rather than transforming to the time-frequency domain. Before normalizing, the heterodyne function (courtesy of D.K. Mellinger [71]) was applied to each observation using a variation of Matlab's signal processing toolkit[1] `fir2` function for the bandpass and lowpass filtering The parameters for the heterodyne are the following:

- Heterodyne band frequency: 100 - 150 kHz
- New bottom frequency: 0 Hz
- FIR Filter length: 512 samples
- Transition bandwidth: 1,000 Hz
- Roll off dB: 60 dB
- Decimation: every 5 samples

[1] https://www.mathworks.com/products/signal.html

Figure 5.2.3.3.1 shows the frequency response of the heterodyne with these parameters. This operation ultimately results in a final waveform of 51,200 samples that is audible to the human ear. Figure 5.2.3.3.2 shows the waveform, before (a) and after (b) heterodyning.

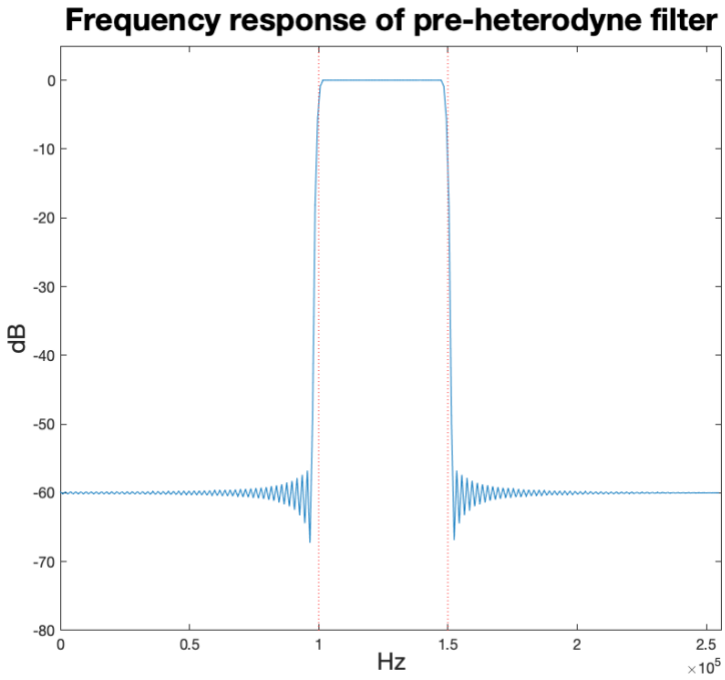

Figure 5.2.3.3.1. Example of a frequency response of the FIR bandpass filter.

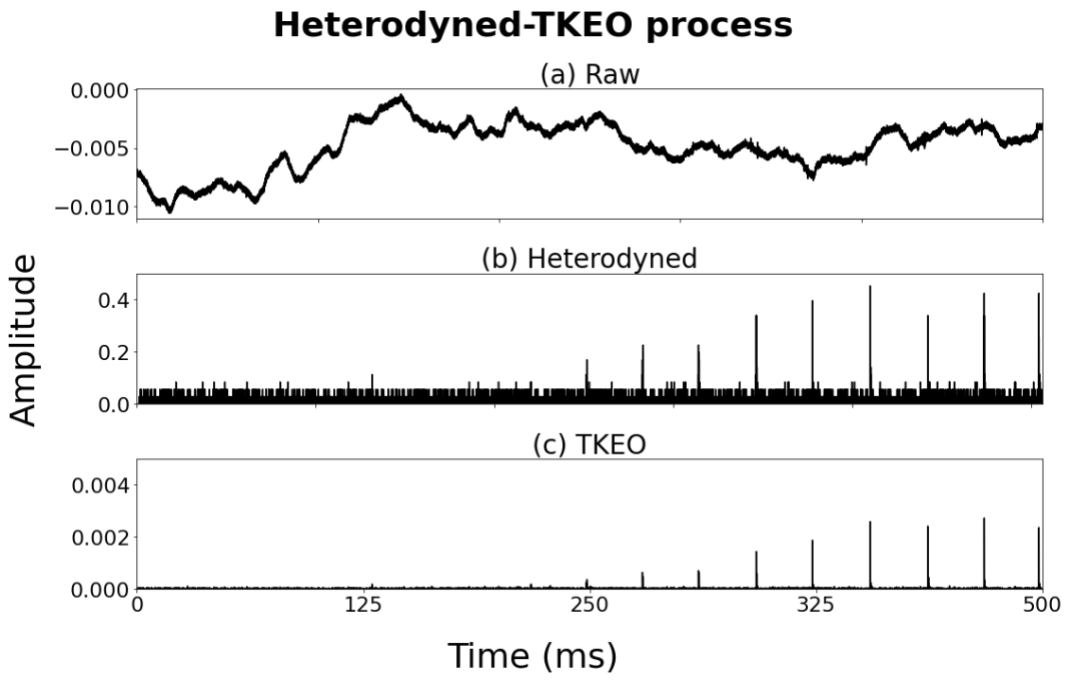

Figure 5.2.3.3.2. Waveforms of each step of the heterodyned-TKEO process. (a) The raw audio that holds 256,000 samples. (b) The result of heterodyning and then normalizing the raw audio example, which has 51,200 samples. (c) The result of using the TKEO operator on the heterodyned audio, also 51,200 samples.

Once heterodyned, all of the observations were then normalized as described in chapter 3. The TKEO operator was then applied to each observation. Figure 5.2.3.3.2 shows a heterodyned waveform before (b) and after (c) the TKEO operator. The final shape after the TKEO is still (1, 51200) since TKEO is just a signal conditioner. An input size of this shape, however, is still unfeasible for the LSTM since 51,200 timesteps would require too much memory to compile such a model on our hardware. We needed to transform the one-dimensional array to a two-dimensional matrix such that each column represented a block of samples but we did not know how many samples that block should be. These blocks, in turn, correspond to the timesteps used in the first layer of the LSTM. We devised a hyperparameter search assessment to decide the shape of the TKEO two-dimensional array.

Given that porpoise clicks, at most, last a duration of 0.115 ms, a timestep resolution representing this duration would result in roughly twelve samples per timestep with around 4,348

timesteps. The following formulas define the relationship between the number of timesteps and samples within each timestep:

$$N_t = \frac{t}{t_b}$$

$$N_b = \frac{N_s}{N_t}$$

such that:

- N_t = number of timesteps (i.e. number of columns)
- N_s = total number of samples in the audio clip (= 51,200)
- N_b = number of samples in each time step (i.e. number of rows)
- t = total time of audio clip (= 500 ms)
- t_b = time represented in each timestep

With these equations we performed a sweep across the height and width of our input shape (N_b and N_t, respectively) into the LSTM to find which shape best captured the porpoise clicks in accordance with the LSTM constraints. Since 4,348 timesteps is still large for the LSTM, we started our sweep at 1,600 timesteps (N_t). This results in a t_b of 0.3125 ms. Since interclick interval has been reported to be as short as 1.5 ms in special cases (see chapter 2.1), this resolution is still appropriate to capture, at most, one click per timestep. Ultimately, this resolution translates to an input shape of (32, 1600), meaning that there are 32 samples per timestep. We ran the LSTM with this input dimension on all of the data with a 70/30 split to observe how it learned over 100 epochs. We then reduced the number of columns by half and retrained the LSTM with this new input dimension. We repeated this halve-and-reprocess operation three more times. Figure 5.2.3.3.3 shows the validation accuracies and losses for the LSTMs with the different input shapes. From this figure, we can observe that the dimension (256, 200) is the best-performing shape. Each timestep holds 256 samples, which corresponds to 2.5 ms. Therefore, this is the shape we chose to use for the remaining heterodyned-TKEO experiments (shortened to just TKEO for the remainder of this work).

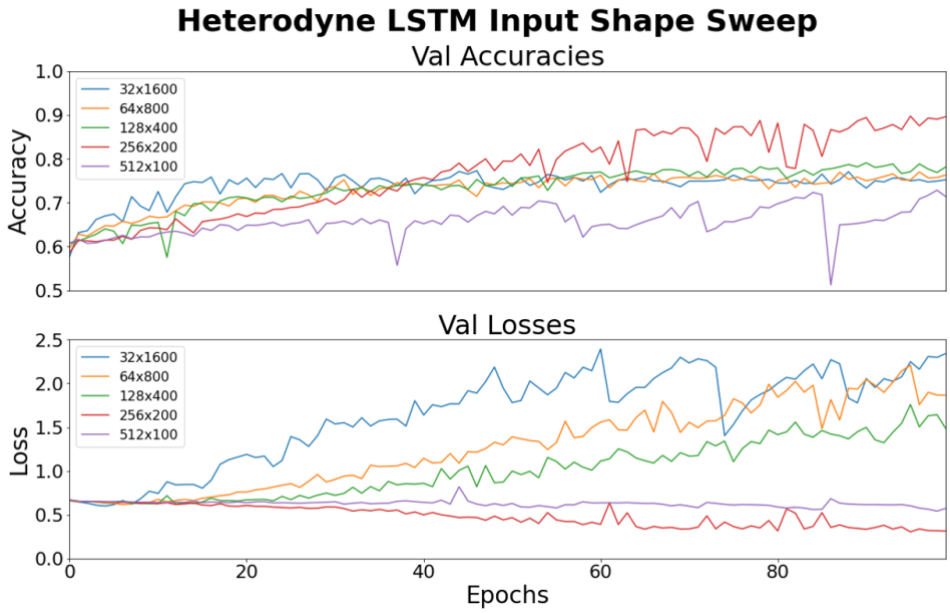

Figure 5.2.3.3.3. Validation accuracy and loss of the LSTM over varying input shapes.

The LSTM that had an input dimension of (256, 200) outperformed the other dimensions by having the lowest and highest test loss and accuracy, respectively. This is why it is the dimension we use for the remaining heterodyned-TKEO experiments.

5.3. Ensemble learning

The experiments described in the previous made evident that models were distinctively learning particular characteristics of the audio. Based on this observation, we believed that assembling all of the models generated from the LSTM and CNN experiments would unite the strength of each model and lead to an overall better performance.

We designed a stacked ensemble with a Random Forest combiner with the same hyperparameters as defined in Section 5.1. With more traditional, simple, ensemble approaches, it is typical to see a variation of weighted average classification of all the models' outputs. For our stacking ensemble method, however, each combiner classifier is trained on the predictions of all the models with LODO CV and then tested on the held-out day of 07/14 for comparison on unseen data. Figure 5.3.1 illustrates the difference between a traditional weighted ensemble and our RF stacked ensemble

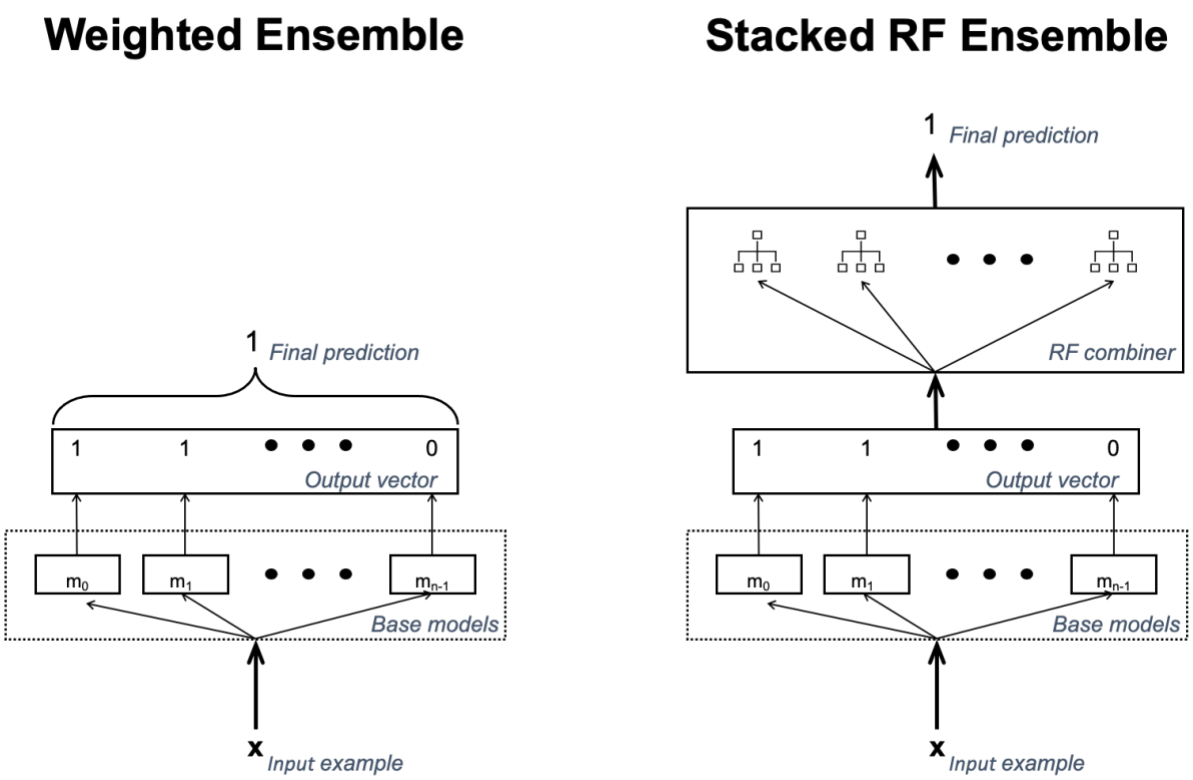

Figure 5.3.1. Illustrative comparison of a simple weighted ensemble (left) and our stacked RF ensemble (right). The last step of the weighted ensemble simply takes an average (that could be weighted) on the output vector provided by the base models. The stacked RF ensemble uses the output vector as in the input the RF combiner.

The feature vector for our ensemble approach is simply the prediction of each of the base models. In our structure, these base models are the CNN and LSTM models that were trained from the preprocessing LODO CV experiments described in Section 5.2. Since LODO CV with a held-out day has seven folds (a fold for each day), there are a total of seven models created for each preprocessing method. With the two architecture types, LSTM and CNN, and three preprocessing methods, there are a total 42 models created. Given an observation, the prediction of each of these models (`click` or `noClick`) makes up the feature vector, of length 42, that is the input into the stacked RF combiner.

6. Results & Discussion

This chapter is organized into two sections that reflect the experimental procedures addressed in chapter 5. Within each section, there is a report of the observed results followed by a discussion of the interpretation of these results. The first section is concerned with the results comparing the different preprocessing approaches with the LSTM and CNN. Section 2 will provide the results of the stacked RF ensemble method and compare with the results of the previous section.

6.1. Preprocessing approaches

This section is divided into three subsections. Sections 6.1.1, 6.1.2 and 6.1.3 report and discuss the results of the MFCC, STFT and heterodyned-TKEO approaches, respectively. Each of these sections will be further subdivided into CNN and LSTM discussions. These three sections will then be followed by a discussion, Section 6.1.4, that summarizes all of the findings of our preprocessing approaches.

Before examining the results of each approach, however, we will first summarize the overall results. Figure 6.1.1 and Table 6.1.1 present the results of every preprocessing approach for each architecture. The array of confusion matrices in Figure 6.1.1, and all of those that proceed, present the groundtruth (True) values on the y-axis and the model's predictions on the x-axis. From this orientation, the true positive (TP), false negative (FN), false positive (FP) and true negative (TN) classification types are displayed. Below each classification type label is its respective proportion of observations that are normalized across the truth. That is, each row of the confusion matrix sums up to 1. Hypothetically, the perfect model, for example, would have values of 1.0 and 0.0 for TP and FN on the top row, respectively, while the bottom row contains values of 0.0 and 1.0 for FP and TN, respectively. Table 6.1.1, and all the tables that proceed each confusion matrix array, summarize the metrics that can be derived from the confusion matrices. The four metrics we use are accuracy, precision, recall and F_1 score, defined below:

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn}$$

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

$$F_1 Score = 2\frac{precison \cdot recall}{prexison + recall}$$

Each column in the table is titled according to its corresponding confusion matrix that is visualized in the figure that precedes the table.

In Table 6.1.1 and Figure 6.1.1, each model represents a collection of models via a seven-day LODO (7-LODO) on all days but the held-out day, 07/14. There are seven models because each fold of LODO trains its own individual model based on the unique test/train/validation split. We assess the entire performance of a 7-LODO by first calculating the confusion matrix for each fold's model on the held-out day, 07/14. Then we sum all of the confusion matrices from each of the seven models into one final confusion matrix. These final confusion matrices for each combination of preprocessing approach and DL model are displayed in Figure 6.1.1.

MFCCs perform the best out of the three preprocessing methods. The STFT and TKEO approaches are then second and third best, respectively. These results are discussed further in sections 6.1.1.3, 6.1.2.3, 6.1.3.3 and Section 6.1.4.

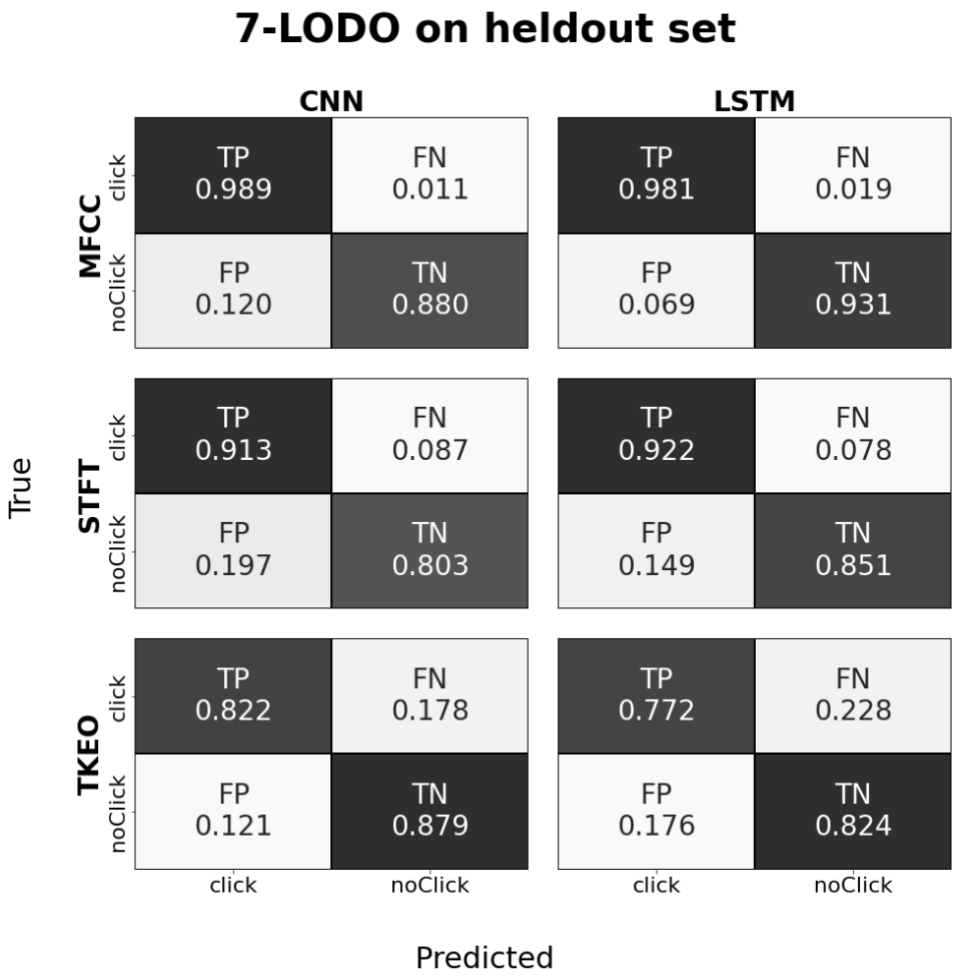

Figure 6.1.1. Confusion matrices for the two DL models (CNN and LSTM) for each of the three preprocessing experiments on the held-out 07/14 day with 7-LODO.

	MFCC		**STFT**		**TKEO**		
	CNN	**LSTM**	**CNN**	**LSTM**	**CNN**	**LSTM**	**Mean**
Accuracy	0.935	**0.956**	0.858	0.887	0.850	0.798	0.881
Precision	0.892	0.935	0.822	0.861	0.872	0.814	0.866
Recall	0.989	0.981	0.913	0.922	0.822	0.772	0.900
F1 Score	0.938	0.957	0.865	0.891	0.846	0.793	0.882

Table 6.1.1. Result metrics for each of the six experiments on the held-out 07/14 day with 7-LODO. The LSTM trained with MFCC input reported the highest accuracy predicting in

predicting the held-out day with an accuracy of 95.6%. The last column presents the mean of each metric across all experiments.

6.1.1. MFCC

We found that the MFCC was the best performing preprocessing method on our held-out set with 7-LODO. The following two sections report the results of using MFCC input on our two models, CNN and LSTM with 8-LODO. The last section will interpret these results.

6.1.1.1. CNN

Figure 6.1.1.1.1 and Table 6.1.1.1.1 show the CNN with MFCC input results for training with 8-LODO. For 8-LODO, each model in the confusion matrix array corresponds to only one model, where each model is tested on the day that was left out for that particular fold. This is contrary to the 7-LODO confusion matrix array presented in Section 6.1. The title of each confusion matrix in Figure 6.1.1.1.1, and all 8-LODO confusion matrix array figures that follow, is the day that was left out and used for testing for that fold. In other words, each model is named after its assigned test day. Table 6.1.1.1.1 shows, in each column, the four performance metrics for each model that are derived from their corresponding confusion matrices. Since each day has a varying number of examples, it should be noted that each confusion matrix in 8-LODO has a different number for `n` (c.f. Table 4.3.3).

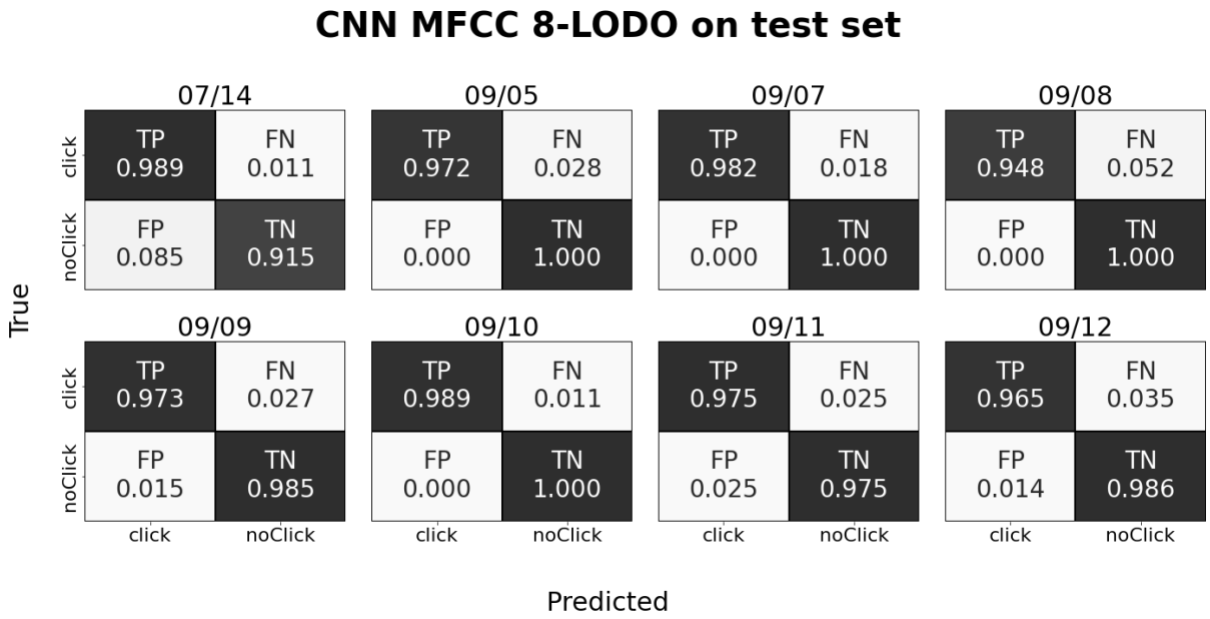

Figure 6.1.1.1.1. Confusion matrices over each fold of 8-LODO CV for the CNN with MFCC input. Each model is named after its held-out test day. Days 09/05, 09/07, 09/08 and 09/10 are 100% accurate on `noClicks` because they have a TN rate 1.0. Note, however, 09/05 and 09/08 have the third highest and highest FN rates, respectively.

	07/14	**09/05**	**09/07**	**09/08**	**09/09**	**09/10**	**09/11**	**09/12**	**Mean**
Accuracy	0.952	0.986	0.991	0.974	0.979	**0.994**	0.975	0.975	0.978
Precision	0.921	1.000	1.000	1.000	0.984	1.000	0.975	0.986	0.983
Recall	0.989	0.972	0.982	0.948	0.973	0.989	0.975	0.965	0.974
F1 Score	0.953	0.986	0.991	0.973	0.979	0.994	0.975	0.975	0.978

Table 6.1.1.1.1. Results metrics for each day for the CNN with MFCC input. Each column corresponds to each model represented by the matrices in Figure 6.1.1.1.1. The mean over all the models for each metric is presented in the last column. The same days 09/05, 09/07, 09/08 and 09/10 show 100% precision while 09/08 has the lowest recall at 94.8%. 09/10 has the best accuracy and F_1-score of all the eight models.

6.1.1.2. LSTM

In the same format as the previous section, 6.1.1.1, Figure 6.1.1.2.1 and Table 6.1.1.2.1. show results for the LSTM architecture with MFCC input. Each fold's confusion matrix is presented and named after its held-out test day. As previously with the CNN, each confusion matrix corresponds to the test results of each fold of 8-LODO where the set is all of the examples in its corresponding day.

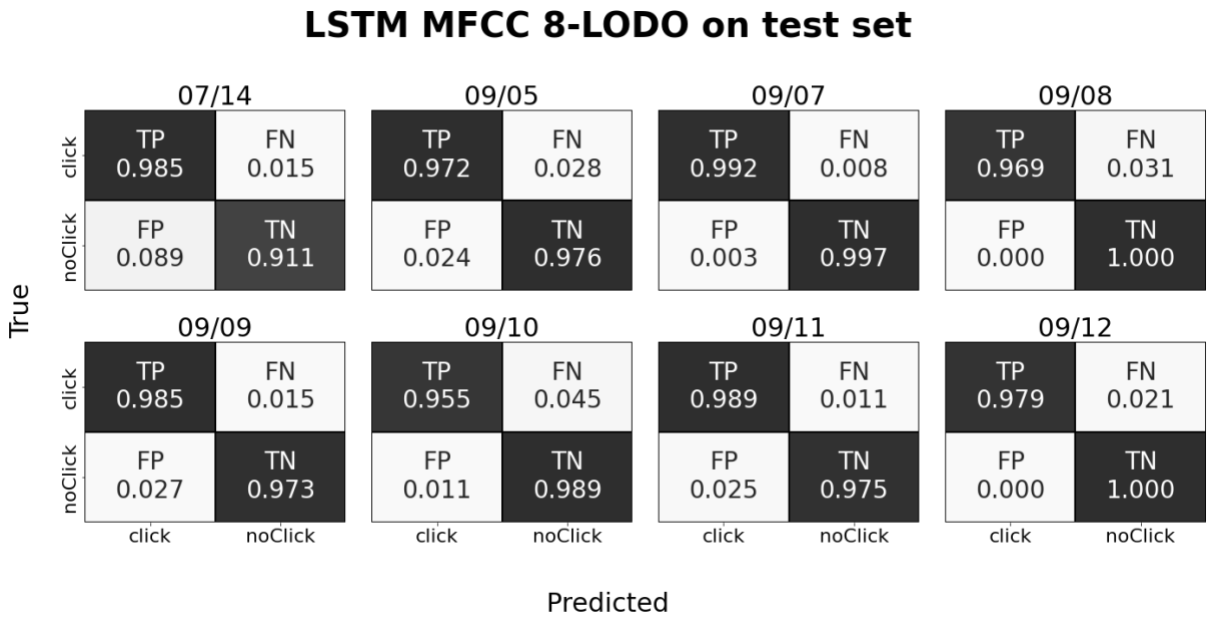

Figure 6.1.1.2.1. Confusion matrices over each fold of 8-LODO CV for the LSTM with MFCC input. Each model is named after its held-out test day. Days 09/08 and 09/12 have a perfect TN of 1.0 but have relatively high FN rates. 09/07 holds the highest TP rate at 0.992.

	07/14	09/05	09/07	09/08	09/09	09/10	09/11	09/12	Mean
Accuracy	0.948	0.974	**0.995**	0.984	0.979	0.972	0.982	0.989	0.978
Precision	0.917	0.976	0.997	1.000	0.974	0.988	0.976	1.000	0.979
Recall	0.985	0.972	0.992	0.969	0.985	0.955	0.989	0.979	0.978
F1 Score	0.950	0.974	0.995	0.984	0.979	0.972	0.982	0.989	0.978

Table 6.1.1.2.1. Result metrics for each day for the LSTM with MFCC input. Each column corresponds to each model represented by the matrices in Figure 6.1.1.2.1, so the date shown is the held-out day that was used for testing. The mean over all the models for each metric is presented in the last column. As is consistent with the confusion matrices, 09/08 and 09/12 have 100% precision since their FP rates are both 0.0. The most accurate day by a significant margin is 09/07 with an accuracy and F_1 score of 99.5%.

6.1.1.3. Discussion

Across all days of 8-LODO, on average the CNN and LSTM are approximately equal in accuracy at 97.8% (*n* examples = 8,286). Although they are similar in accuracy, they misclassify different examples. Both models, nonetheless, struggle the most on the 7/14 fold as we initially hypothesized. It is surprising, however, that both CNN and LSTM architectures perform second best (99.1%) and best (99.5%), respectively, on 09/07 given that the peak RMS_s distribution is almost 0.2 RMS_s greater than the remaining days, excluding 07/14. Given that the 09/07 model is trained on data including 07/14, the diversity of RMS_s that that day contributes could be the result of the performance improvement.

Another fold of interest, particularly with its performance with respect to its recall, is 9/10. Given its significantly different RMS_s distribution of `noClicks`, it is not surprising to see the difficulty that the LSTM had in classifying the negative class for that particular day as 95.5% was the lowest recall of all the other days. The CNN however did not struggle seeing a recall of 98.9%, tied for the highest recall of all other folds.

We should also point out the discrepancy between the 07/14 model (in which 07/14 was the test set for that model) in 8-LODO and the 7-LODO final held-out results (in which 07/14 was held-out entirely from the CV). Although the CNN achieves slightly better results in all four

metrics in the 8-LODO (< 0.5%), the LSTM in the 7-LODO outperforms the CNN by roughly 2% in accuracy and F_1 score, and 3% in precision. Since the held-out day in July from 7-LODO has a significantly different noise distribution than the rest of the days in September, this indicates that the CNN is not generalizing to real-world data as well as the LSTM.

In general, we find that both the CNN and LSTM have strengths and weaknesses on different days/folds. We have plotted the misclassification frequencies at each fold for both models in Figure 6.1.1.3.1. Each fold/day is split between its FP and FN, in red and blue, respectively. It is clear from these bar charts that both architectures struggle to classify `noClicks` since every day has some amount of FNs. The CNN, however, is perfect in not reporting any FP on 09/05, 09/07, 09/08 and 09/10. On the other hand, the LSTM does not classify any FNs on 09/08 and 09/12.

Uncharacteristically, however, we found a majority of the misclassifications on 07/14, for both architectures, are FPs. Since the 07/14 RMS_s distribution spreads to higher RMS_s values than the rest of the days, we believe there could be strong enough signals to raise false alarms. Taking a closer look at these FNs in particular, the CNN and LSTM turn out to misclassify different examples.

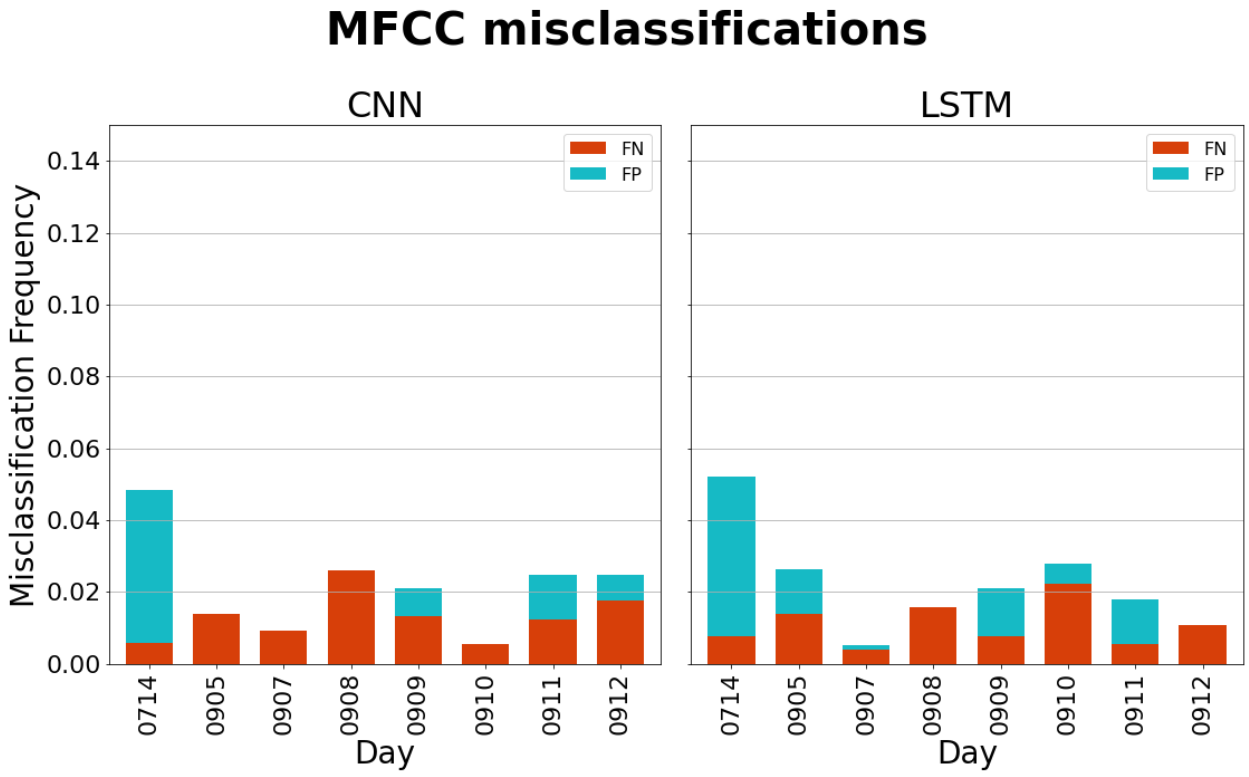

Figure 6.1.1.3.1. Comparison of MFCC misclassifications between the CNN (left) and LSTM (right). The y-axis corresponds to frequency of misclassification (n of misclassifications/n of total examples in that day). The x-axis are the corresponding test days of the misclassifications. Both architectures struggle in classifying `noClicks` on 07/14 with a total misclassification rate of 0.05.

Figure 6.1.1.3.2 shows the set differences and intersections of the misclassifications for the CNN and LSTM. The leftmost plot depicts the misclassifications that only the CNN made but the LSTM did not. The middle plot shows the set of misclassifications that either the CNN and LSTM made (i.e., the intersection of the misclassifications). The rightmost plot shows the misclassifications made by the LSTM but not the CNN. From inspection of these three plots, 07/14 shows a significant reduction of FP frequency when both architectures, in intersection, misclassify the same examples. Another significant reduction that should be noted is on 9/10, the day on which there is an RMS_s distribution covariate shift in the `noClicks`. The CNN has no misclassifications on 9/10 when we discount any example that the LSTM has also misclassified. In other words, the CNN never misclassified an example that LSTM didn't also misclassify. In comparing and observing the predictions of the CNN and LSTM such as this one, it is evident that both models provide their own forms of robustness throughout the eight days of 8-LODO.

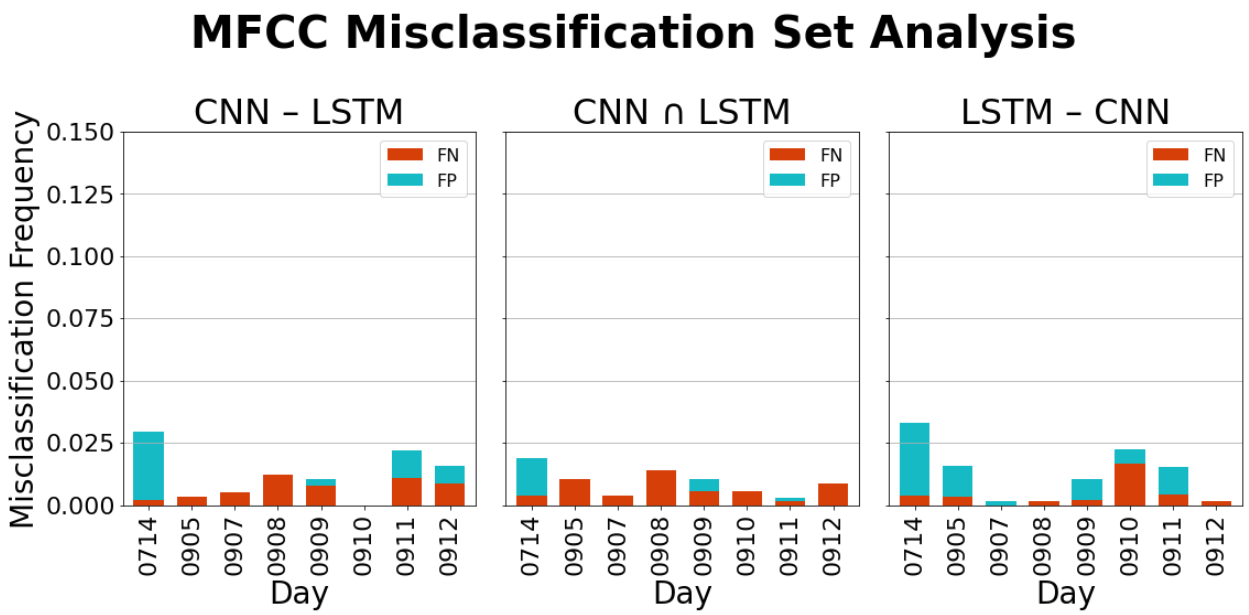

Figure 6.1.1.3.2. Set differences (left and right) and intersections (middle) of the MFCC misclassifications from the CNN and LSTM. The set differences present the unique misclassifications of each architecture while the intersections show misclassifications that both architectures had in common. The CNN has no unique misclassifications on 09/10. It should also be noted that both architectures have a significant number of unique misclassifications remaining in 07/14.

6.1.2. STFT

Similar to the organization of Section 6.1.1, this section is dedicated to the results and discussion of how the CNN and LSTM performed with STFT input. Overall, we observed the STFT to be the second-best preprocessing approach (after the MFCC) with an accuracy on the heldout 07/14 with 7-LODO of 85.8% and 88.7%, for the CNN and LSTM, respectively. The mean accuracy for CNN and LSTM on 8-LODO are 91.5% and 90.1%, respectively.

6.1.2.1. CNN

Figure 6.1.2.1.1 and Table 6.1.2.1.1 show the performance of the STFT input on the CNN on 8-LODO CV. Each confusion matrix in the figure results from a separate model trained and validated on all days except the day that it is named. For example, the bottom-right confusion

matrix, 09/12, is the model corresponding to the fold where 09/12 was left out for testing, and therefore the bottom-right confusion matrix is the result of the model training and validating on all days except 09/12. Table 4 then provides the metrics for each confusion matrix represented in figure 3.

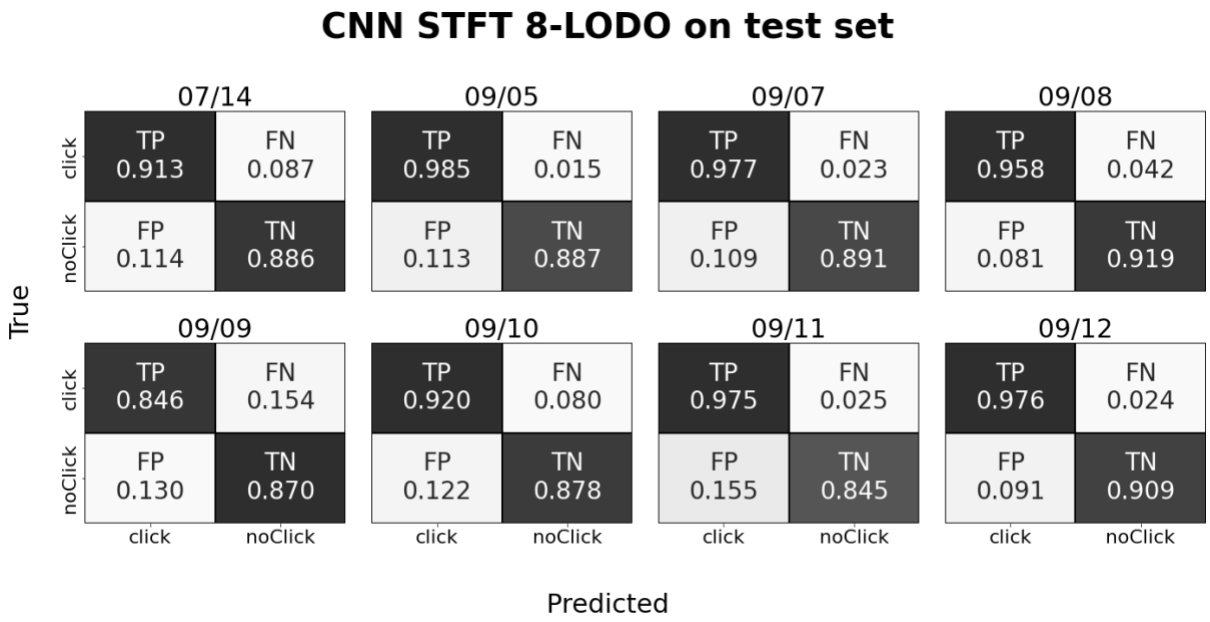

Figure 6.1.2.1.1. Confusion matrices over each fold of 8-LODO CV for the CNN with STFT input. Each model is named after its held-out test day. The model tested on 09/05 had the highest TP rate of 0.985 but had one of the lower TP rates of 0.887. The model tested on 09/08 showed the highest TN rate of 0.919.

	07/14	09/05	09/07	09/08	09/09	09/10	09/11	09/12	Mean
Accuracy	0.900	0.936	0.934	0.939	0.858	0.899	0.910	**0.942**	0.915
Precision	0.889	0.897	0.899	0.922	0.867	0.883	0.863	0.914	0.892
Recall	0.913	0.985	0.977	0.958	0.846	0.920	0.975	0.976	0.944
F1 Score	0.901	0.939	0.936	0.940	0.856	0.901	0.916	0.944	0.917

Table 6.1.2.1.1. Results metrics for each day for the CNN with STFT input. Each column corresponds to each model represented by the matrices in Figure 6.1.2.1.1. The mean over all the models for each metric is presented in the last column. As is consistent with the confusion matrices, 09/05 has the highest recall and 09/08 has the highest precision at 98.5% and 92.2%, respectively. The most accurate day, however, is 09/12 with an accuracy of 94.2% and F_1 score of 94.4%.

6.1.2.2. LSTM

The results for the LSTM are presented in the same manner as the CNN results in which a figure of confusion matrices accompanied with a table of metrics derived from those confusion matrices are shown below. Figure 6.1.2.2.1 and table 6.1.2.2.1 provide the results for each fold 8-LODO with STFT input into the LSTM.

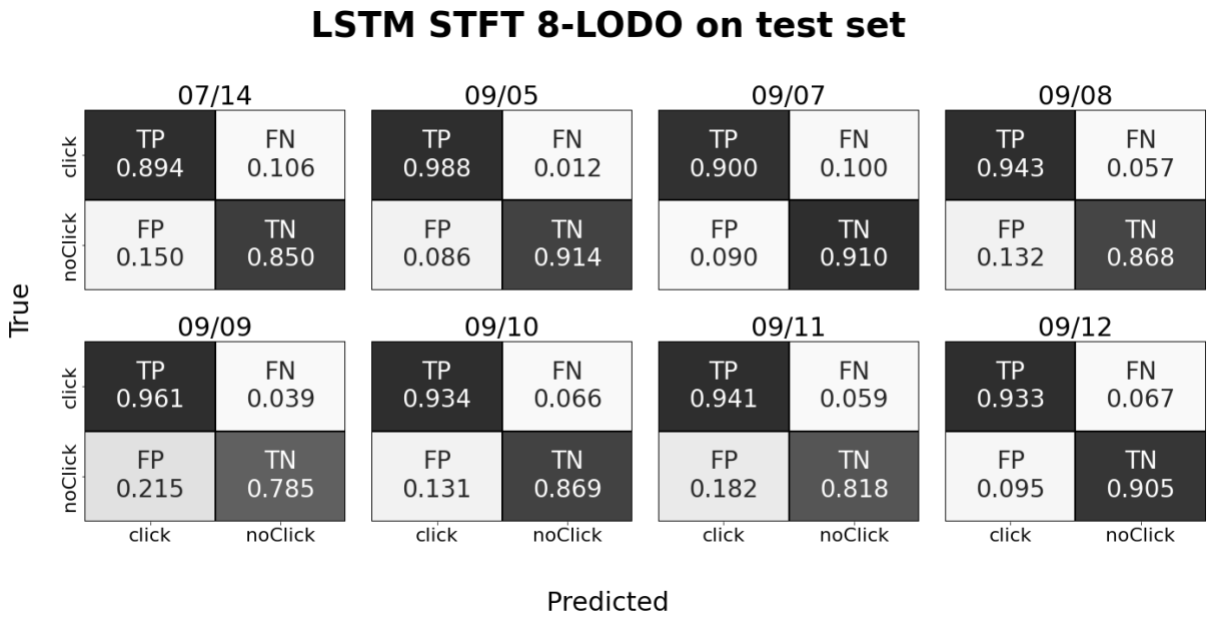

Figure 6.1.2.2.1 Confusion matrices over each fold of 8-LODO CV for the LSTM with STFT input. Each model is named after its held-out test day. The 09/05 fold had the significantly highest TP rate of 0.988. This same fold also had the highest TN rate of 0.919. The 09/09 struggled the most `noClicks` seeing a FP rate of 0.215.

	07/14	**09/05**	**09/07**	**09/08**	**09/09**	**09/10**	**09/11**	**09/12**	**Mean**
Accuracy	0.872	**0.951**	0.905	0.906	0.873	0.901	0.880	0.919	0.901
Precision	0.856	0.920	0.909	0.877	0.817	0.877	0.838	0.908	0.875
Recall	0.894	0.988	0.900	0.943	0.961	0.934	0.941	0.933	0.937
F1 Score	0.875	0.953	0.905	0.909	0.883	0.905	0.887	0.920	0.905

Table 6.1.2.2.1. Results metrics for each day for the LSTM with STFT input. Each column corresponds to each model represented by the matrices in Figure 6.1.2.2.1, so the date shown is the held-out day that was used for testing. The mean over all the models for each metric is presented in the last column. As is consistent with the confusion matrices, 09/05 has the highest recall and 09/09 has the lowest precision at 81.7%. The most accurate day by a significant margin is 09/05 with an accuracy 5% greater than the mean of all folds.

6.1.2.3. Discussion

Similar to the results we observed from MFCCs shown in Figure 6.1.1, the LSTM outperformed the CNN on the held-out 07/14 day, with 7-LODO, by 2.9%. 8-LODO has a contradicting result, however, in that the CNN, on average, is 1.4% more accurate than the LSTM. This suggests, again, that the CNN is not generalizing as well as the LSTM is.

Surprisingly, in 8-LODO we see the CNN struggle on 09/09 with an accuracy of 85.8% (*n* examples = 1662). This is not a day identified from our observations of RMS_s and visual inspections of the LTSAs that seemed particularly anomalous given it follows a similar distribution as the other days. 09/09 does contain examples with the lowest RMS_s seen throughout the entire data set (see Tables 1 – 3 in appendix A). This could indicate that the CNN on this particular fold overfit to the higher RMS_s values and thus associated less background noise with `noClick` classifications as we can see a significant increase in FN rate (15.4%, *n* clicks = 831). The LSTM on this same day performs at an accuracy 1.5% greater. Conversely, the CNN performs better on 07/14 (2.8% greater, *n* examples = 1848), the day that we did identify as challenging. This is counterintuitive, given that, as mentioned previously, the CNN performs worse on this same day when it is held out with 7-LODO. This further supports that the CNN is not generalizing well to unseen data.

Given that we see unique behaviors from the CNN and LSTM performances on the 8-LODO test days, Figure 6.1.2.3.1 shows the misclassification frequency of both models. We can more clearly see from Figure 6.1.2.3.1 that not only does the LSTM have a lower misclassification frequency on 09/09, it has significantly less FNs than its CNN counterpart on this day.

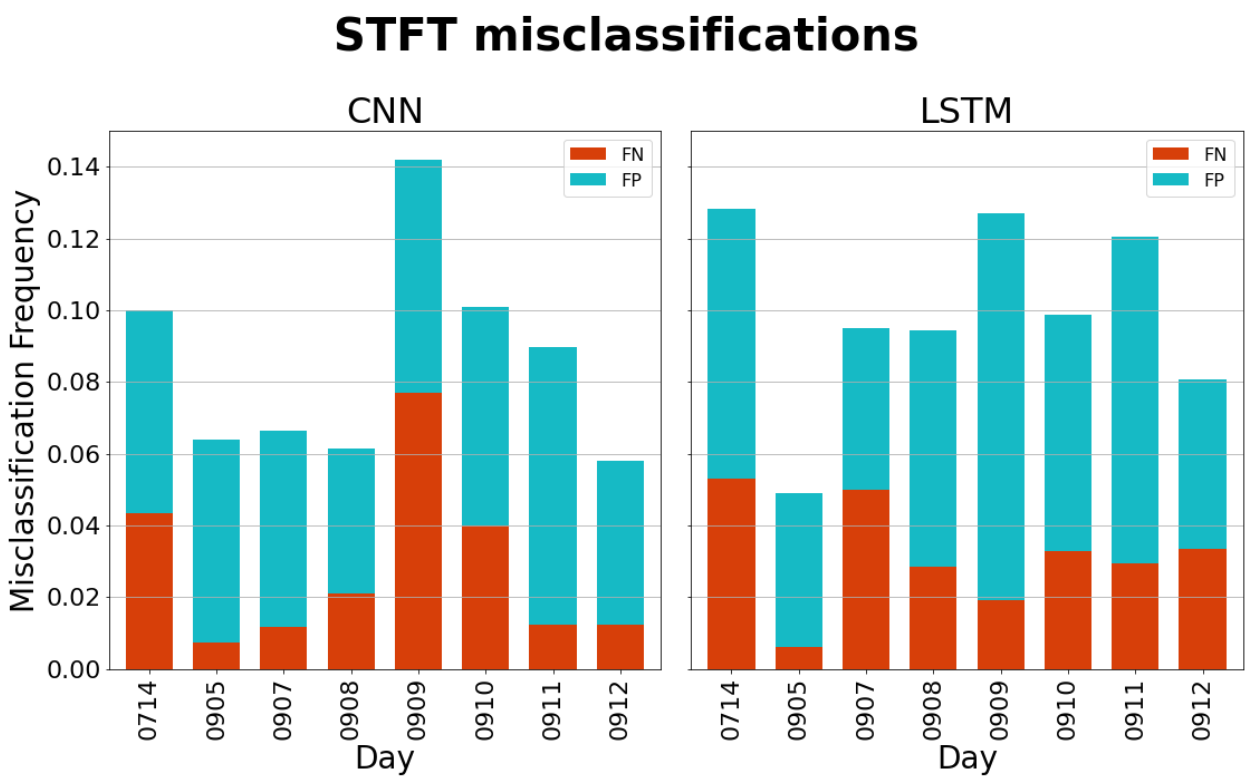

Figure 6.1.2.3.1. Comparison of STFT misclassifications between the CNN (left) and LSTM (right). The y-axis corresponds to frequency of misclassification (n of misclassifications/n of total examples in that day). The x-axis are the corresponding test days of the misclassifications. Both architectures struggle in classifying examples in 09/09, although the LSTM has a lower misclassification rate. The CNN, however, achieves a 0.025 lower misclassification frequency than the LSTM.

Similar to Figure 6.1.1.3.2, Figure 6.1.2.3.2 shows the set differences and intersections of the misclassifications for the CNN and LSTM. The left and right plot correspond to the unique misclassifications of the CNN and LSTM, respectively. The middle plot is then the intersection of the misclassifications or in other words, the misclassifications that the CNN and LSTM both made.

We observed from these bar plots that the misclassifications on 09/09 that the LSTM and CNN make are mostly unique in that we see a much lower misclassification frequency when we perform the intersection of the CNN and LSTM misclassifications. In their intersection, we also see that not only are the misclassifications significantly reduced on the 09/10 folds, but that day (fold) no longer has any FPs.

In removing all of the common misclassifications that the LSTM had with the CNN, 07/14 and 09/09 see a significant reduction in misclassifications as well. In comparing Figure 6.1.2.3.1's CNN misclassifications (left) with the set difference CNN – LSTM in Figure 6.1.2.3.2 (left), there is an approximate reduction of 0.03 in the misclassification frequency.

Ultimately, both the CNN and LSTM individually perform better and worse on different days throughout 8-LODO. When we examine the unique examples that they misclassify, however, we observe significant reductions in their misclassification frequencies such as 07/14 and 09/09.

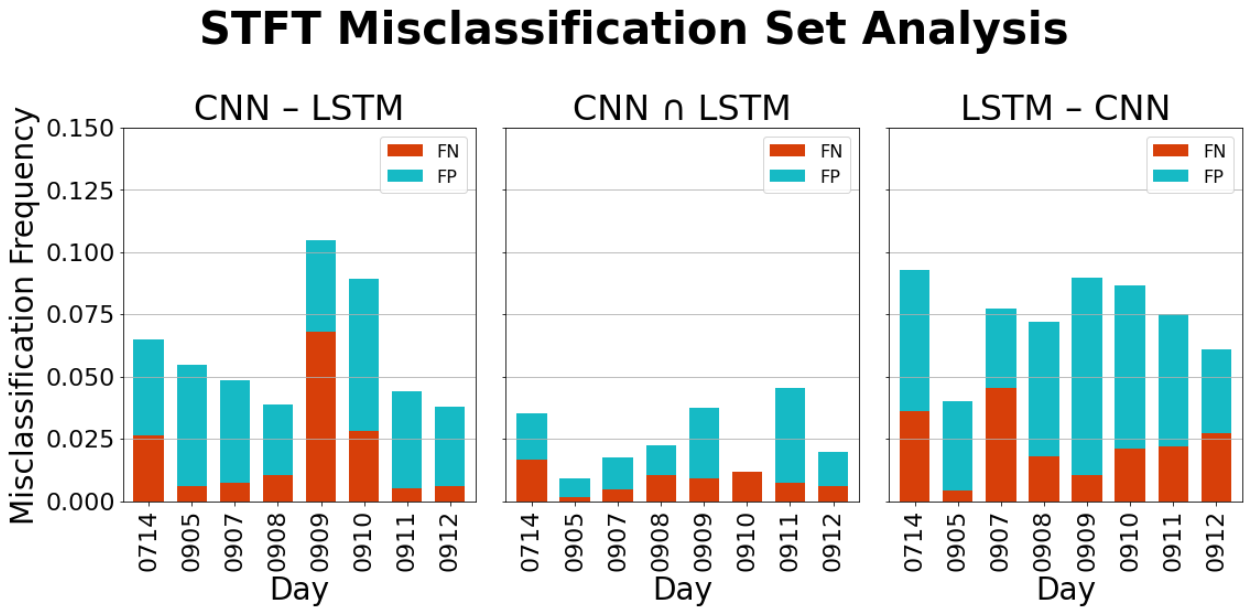

Figure 6.1.2.3.2. Set differences (left and right) and intersections (middle) of the STFT misclassifications from the CNN and LSTM. The set differences present the unique misclassifications of each architecture while the intersections show misclassifications that both architectures had in common. The set differences (left and right) of the misclassifications are all higher than their respective intersections. It should also be noted the intersection at 9/10 has FPs meaning that the LSTM and CNN did not have any FP misclassifications in common.

6.1.3. Heterodyned TKEO

Of the three preprocessing approaches, the TKEO was the least effective preprocessing method in classifying the held-out day, 07/14, on 7-LODO. Similar to the previous two preprocessing sections, 6.1.2 and 6.1.3, this section will be divided into three subsections. Section

6.1.3.1 and 6.1.3.2 serve to report the performance of the CNN and LSTM with the TKEO input on 8-LODO, respectively. Lastly, section 6.1.3.3 will discuss the meaning of the results and identify the differences between the DL architectures.

6.1.3.1. CNN

Figure 6.1.3.1.1 and Table 6.1.3.1.1 show the results after running 8-LODO on the TKEO input to the CNN. Figure 6.1.3.1.1 displays the confusion matrix of test results of each fold of 8-LODO such that every fold (day) is named after the day that is left out. Table 6.1.3.1.1 supports Figure 6.1.3.1.1 by providing the metrics (accuracy, precision, recall and F_1 score) for each fold.

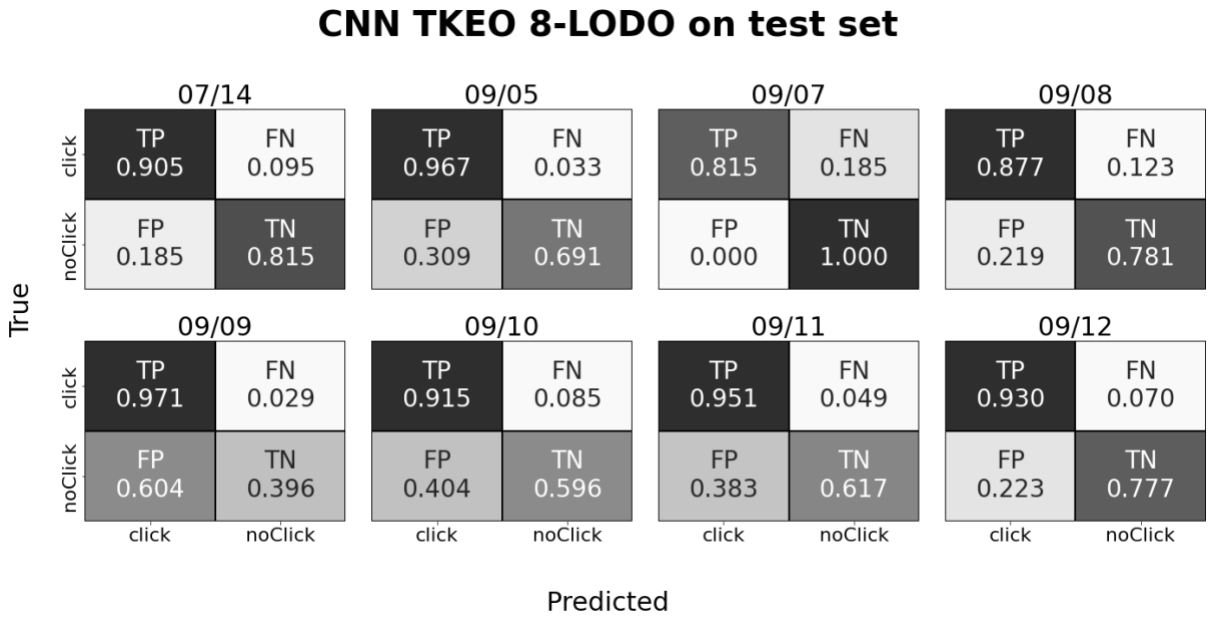

Figure 6.1.3.1.1. Confusion matrices over each fold of 8-LODO CV for the CNN with TKEO input. Each model is named after its held-out test day. The 09/09 fold has a concerning FP rate of 0.604 meaning that it misclassified `noClicks` more than it classified them correctly. This model however classified many of the examples as `clicks` which resulted in it having the highest TP rate of 0.971. The 09/07 fold, on the other hand, reported no FPs. The, however, comes at the cost of its TP rate which is the lowest at 0.815.

	07/14	09/05	09/07	09/08	09/09	09/10	09/11	09/12	Mean
Accuracy	0.860	0.829	**0.908**	0.829	0.684	0.756	0.784	0.854	0.813
Precision	0.830	0.758	1.000	0.801	0.617	0.694	0.713	0.807	0.778
Recall	0.905	0.967	0.815	0.877	0.971	0.915	0.951	0.930	0.916
F1 Score	0.866	0.850	0.898	0.837	0.754	0.789	0.815	0.864	0.834

Table 6.1.3.1.1. Results metrics for each day for the CNN with TKEO input. Each column corresponds to each model represented by the matrices in Figure 6.1.3.1.1, so the date shown is the held-out day that was used for testing. The mean over all the models for each metric is presented in the last column. As is consistent with the confusion matrices, 09/09 has the worst precision by a significant margin with precision 16.1% lower than the mean precision across all folds. Precision of the 09/07 fold, on the other hand, has the highest perfect precision and is 22.2% higher than mean. Although it has one of the lower recall of the days, 09/07 still has the highest accuracy of 90.8%.

6.1.3.2. LSTM

Following the same format as the previous section showing the results of the CNN, this section reports the results of TKEO input into the LSTM. Figure 6.1.3.2.1 and Table 6.1.3.2.1 show the confusion matrices and their corresponding metrics of each fold in 8-LODO.

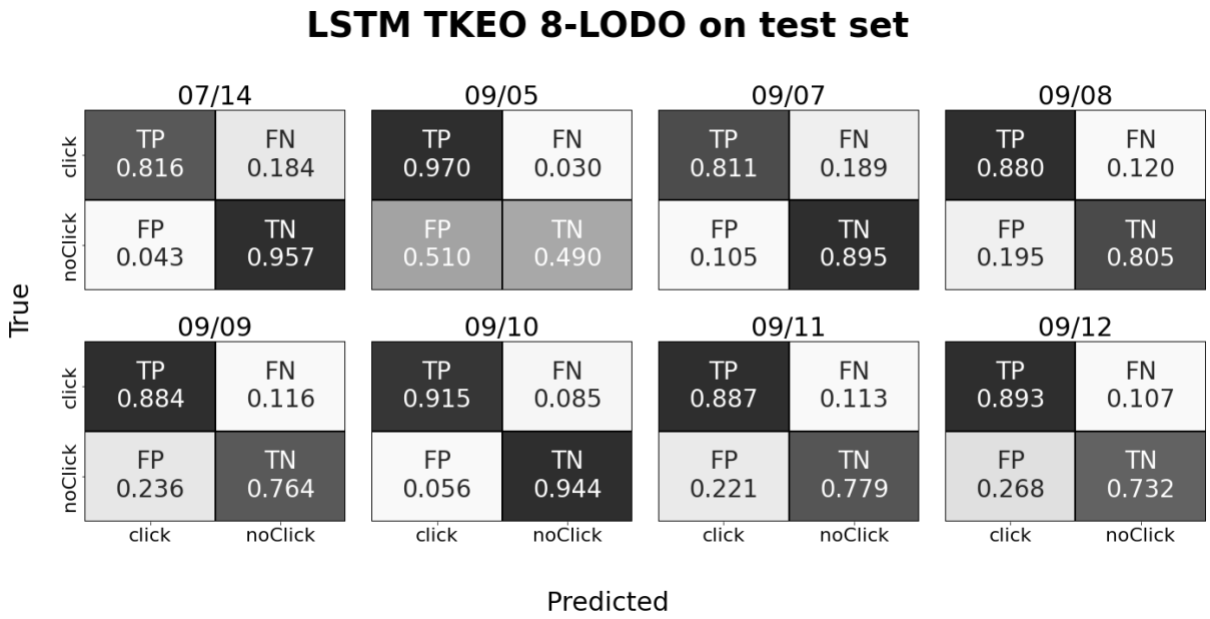

Figure 6.1.3.2.1. Confusion matrices over each fold of 8-LODO CV for the LSTM with TKEO input. Each model is named after its held-out test day. The 09/05 fold has a concerning FP rate of 0.510 meaning that it misclassified `noClicks` more than it classified them correctly. This model however classified many of the examples as `clicks` which resulted in it having the highest TP rate of 0.970. The 09/10 fold performs exceptionally well relative to other folds as it reports the highest TN rate of 0.944.

	07/14	**09/05**	**09/07**	**09/08**	**09/09**	**09/10**	**09/11**	**09/12**	**Mean**
Accuracy	0.886	0.730	0.853	0.843	0.824	**0.930**	0.833	0.812	0.839
Precision	0.950	0.655	0.885	0.819	0.789	0.942	0.800	0.769	0.826
Recall	0.816	0.970	0.811	0.880	0.884	0.915	0.887	0.893	0.882
F1 Score	0.878	0.782	0.847	0.848	0.834	0.929	0.841	0.827	0.848

Table 6.1.3.2.1. Results metrics for each day for the LSTM with TKEO input. Each column corresponds to each model represented by the matrices in Figure 6.1.3.2.1, so the date shown is the held-out day that was used for testing. The mean over all the models for each metric is presented in the last column. As is suggested from the confusion matrices, 09/05 has the worst precision by a significant margin with precision 17.1% lower than the mean precision across all folds. Accuracy, precision and F_1 score of the 09/10 fold, on the other hand, is the highest amongst all of the folds.

6.1.3.3. Discussion

On average, the LSTM is 2.6% more accurate (83.9%, *n* examples = 8,286) than its CNN counterpart (81.3%, *n* examples = 8,286) in our 8-LODO experiment. This conflicts with the results of performance in 7-LODO, where we see the CNN outperform the LSTM by 5.2% in accuracy (*n* examples = 1,848). Even though the accuracy is higher for the LSTM, the average recall reported for the CNN's 8-LODO, however, is 3.4% higher than that of the LSTM. The CNNs recall is also higher in the 7-LODO result as well. This means that the CNN has less FNs which means that it is better at not missing positive classifications (clicks).

In analyzing the particular folds of 8-LODO for both models, and contradicting our hypothesis that 07/14 would be a challenging day, TKEO does not particularly struggle with 07/14 relative to the other folds. 09/09 for the CNN, for example, was by far the most challenging fold of the eight with an accuracy of only 68.4% (*n* examples = 1662), a 12.9% deviation from the average accuracy of the folds. This means that CNN on this fold learned to have a low threshold for probability of the existence of a click resulting in a high false alarm rate. From Figure 4.3.3, we can see from the 09/07 LTSA that the examples are extracted from a time of day (between 2 am to 4 am) where the onset of noise is just beginning. Since this day was used explicitly for testing, the absence of examples (*n* examples = 1,662) with this type of noise in the training could have negatively affected the way in which the CNN learned. It also should be explicitly noted that although the RMS_s values for 09/09 are not on average the lowest, this particular day does still hold some of the lowest RMS_s values out of the entire dataset and this could have been a reason for this relatively poor performance.

The LSTM did not exhibit the same difficulties for this particular fold but rather struggled on 09/05 with an accuracy of 73.0% (*n* examples = 674), deviating from the mean accuracy by 10.9%. Similar to CNN's struggle with 09/09, however, the LSTM also had a high recall, meaning that it did not have many FNs reported. Table 4.3.3 shows that 09/05 has a mean RMS_s across all of its examples of 0.320, the lowest of all the days and 0.027 lower than 09/09. This indicates that the TKEO input influences the CNN and the LSTM to overfit to the background noise resulting in poor test performances across all of the folds.

Because both of these models have differing performances on different folds, a comparison of their misclassification frequencies is visualized in Figure 6.1.3.3.1. Besides 09/09 having the most misclassifications for both architectures, we also note that 09/07 for the CNN has no FPs. The LSTM on the other hand, not only has more FNs, it also has FPs. Both architectures are learning considerably different characteristics of the audio.

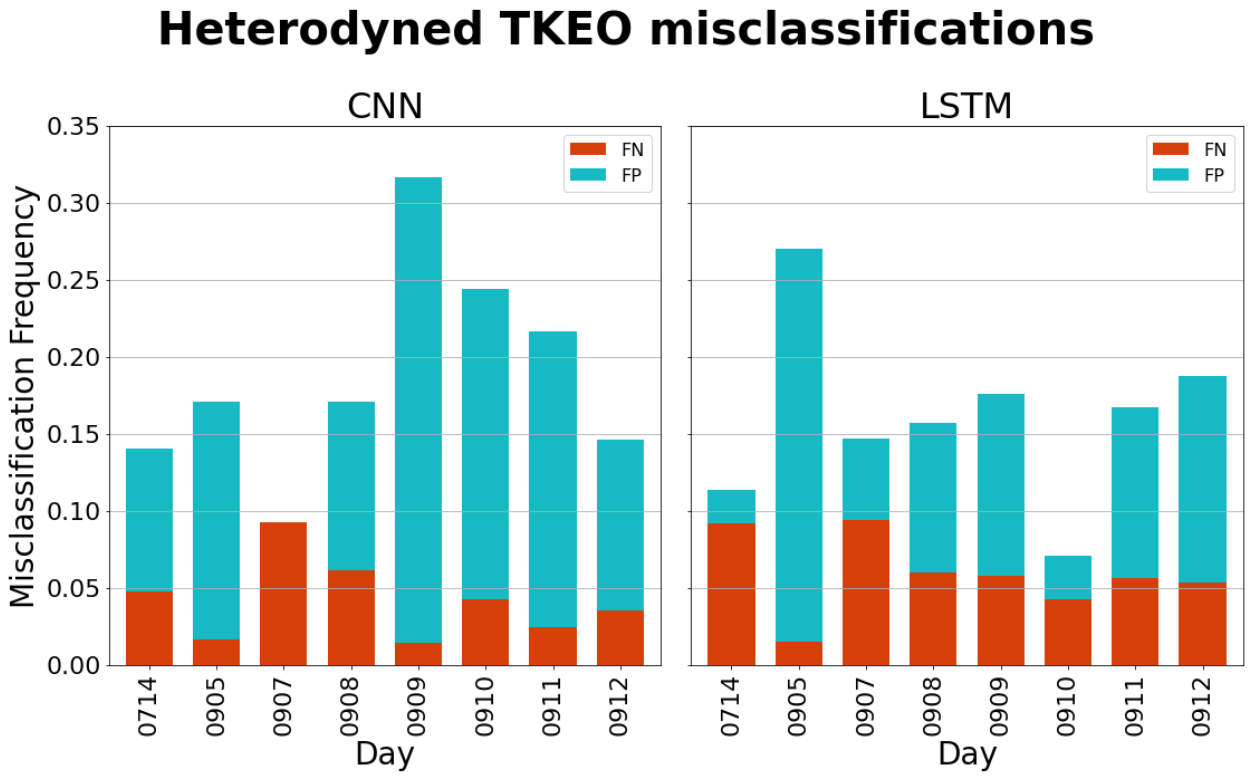

Figure 6.1.3.3.1. Comparison of TKEO misclassifications between the CNN (left) and LSTM (right). The y-axis corresponds to frequency of misclassification (n of misclassifications/n of total examples in that day). The x-axis are the corresponding test days of the misclassifications. The CNN misclassified significantly more on 09/09 than the LSTM did with a difference in frequency of approximately 0.16. The CNN, however, did struggle nearly as much on 09/05 as the LSTM with a difference in misclassification frequency of approximately 0.1. Also, the CNN did not make any errors on `click` examples on 09/07.

To further explore the characteristics of the audio that these architectures are learning, Figure 6.1.3.3.2 displays the set differences and intersections of the misclassifications for the CNN and LSTM. From left to right, the bar charts represent the unique CNN misclassifications (excluding the what LSTM misclassified), the misclassifications that LSTM and CNN both agreed to, and lastly the unique LSTM misclassifications (excluding what the CNN misclassified).

From these plots, we can see that the intersection (middle plot) of the CNN and LSTM results in a reduced misclassification frequency across all days. The errors made in the more challenging days for the CNN and LSTM models are substantially reduced. Investigating the challenging days, 09/09 and 09/05 the CNN and LSTM, respectively, we see substantial reductions. 09/09 from the exclusive CNN misclassification rate (left plot) drops from 0.25 to 0.10 when intersected (middle plot) with the LSTM. The frequency from the exclusive LSTM misclassifications (right plot) for 09/05 model drops from 0.20 to 0.10 when intersected (middle plot) with the CNN. Additionally, 09/07 has no FPs when both models' misclassifications are intersected. Overall, although the CNN outperforms the LSTM on 7-LODO, our 8-LODO results provide a different perspective and show strengths and weaknesses of both models. Our observations of the misclassification frequencies at each fold provide insight into the benefits of comparing the outputs of both models such as the exclusively strong performances on 09/07 and 09/10 for the CNN and LSTM, respectively.

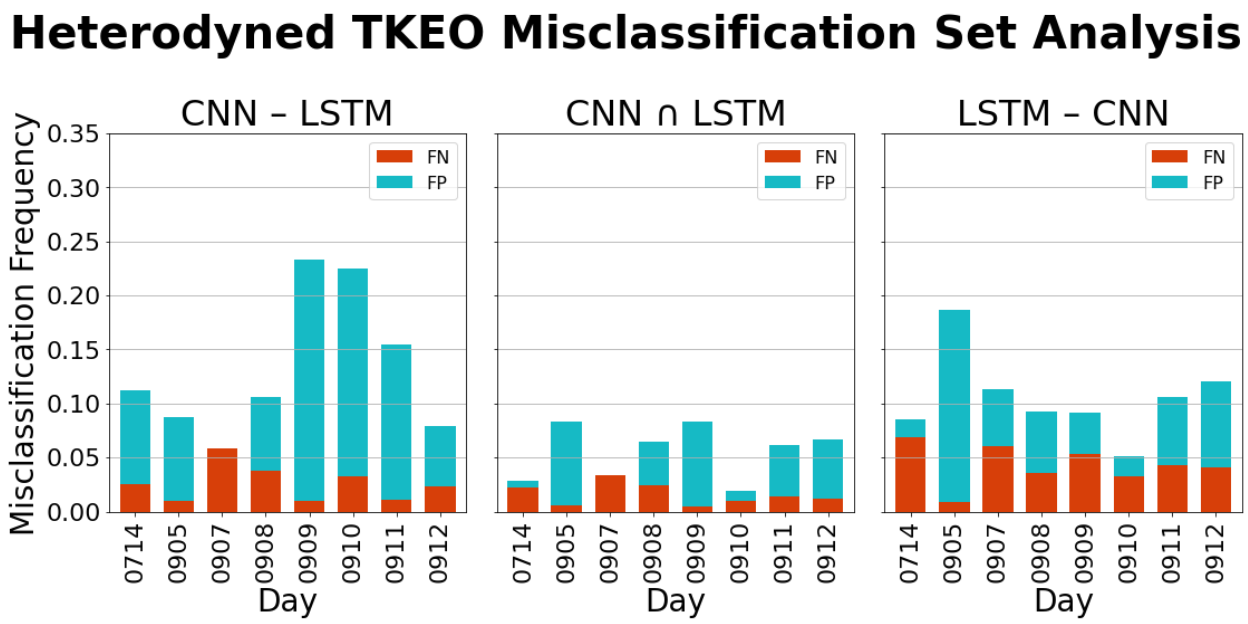

Figure 6.1.3.3.2. Set differences (left and right) and intersections (middle) of the TKEO misclassifications from the CNN and LSTM. The set differences present the unique misclassifications of each architecture while the intersections show misclassifications that both architectures had in common. The set differences (left and right) of the misclassifications are all higher than their respective intersections. 09/10 has a significantly low intersection rate (middle) meaning that the errors the CNN and LSTM make are mostly on different examples.

6.1.4. Discussion

After the analyses of results of using three different preprocessing approaches, MFCC, STFT and TKEO, as input into two different models, the CNN and LSTM models, we conclude that the MFCC performs the best on both 7-LODO and 8-LODO CVs. In particular, the LSTM-MFCC has the highest accuracy at 95.6%. With respect to the 7-LODO accuracies, the next best accuracy after LSTM-MFCC to be the LSTM-STFT at 88.7%. Table 6.1.4.1 shows the 7-LODO and 8-LODO accuracy results for all the preprocessing approaches.

Architecture	8-LODO mean accuracy across all folds	7-LODO accuracy on held-out day 07/14
MFCC		
CNN	0.978	0.935
LSTM	**0.978**	**0.956**
STFT		
CNN	0.915	0.858
LSTM	0.901	0.887
TKEO		
CNN	0.813	0.850
LSTM	0.839	0.798

Table 6.1.4.1. 7-LODO and 8-LODO mean accuracies for all preprocessing approaches. Notice that of the 8-LODO mean accuracies are greater than the 7-LODO held-out accuracy except for the CNN-TKEO experiment.

Given that an MFCC example requires the least amount of memory at 26 KB per example (see Section 5.2.3), it is surprising to see how successful MFCCs were with respect to other preprocessing methods. Its success over the STFT could be attributed to the fact that the STFT calculation alone is too simple. The MFCCs add additional steps after the STFT such as the conversion to the mel scale and the inverse DCT (see Section 2.2.2) that could be serving to capture harbor porpoise click characteristics.

The STFT and TKEO have sparser data than the MFCC data because the click signals are more finely embedded in the data. Given that our architectures are shallow, the MFCC's smaller size (and as a result less sparse) could then also contribute to better generalization with respect to the other models because there is less opportunity for the models to overfit the data. This successful performance in the MFCCs is consistent with the explorations discussed in Section 3.3 and further reinforces why MFCCs remain the state of the art preprocessing approach for DL bioacoustics classification.

The CNN results might mislead us to believe that the MFCC performs equivalently and the STFT outperforms the LSTM in 8-LODO. In 7-LODO, however, we see that the LSTM's seven folds generalize better due to the fact that the LSTM predicts the held-out 07/14 more accurately than the CNN. This is not the case for the TKEO however.

For TKEO, the CNN outperforms the LSTM by 5.2% on 7-LODO. This is surprising for two reasons. The first is that, for the first time, the LSTM has a higher accuracy in 8-LODO (2% increase) than the CNN. Secondly, given that the LSTM had recurrent feedback connections, it might be expected that it would perform better on sequential data. Since TKEO stays in the time domain (unlike the MFCC and STFT that transform to the time-frequency domain), the LSTM should especially be better equipped than the CNN to handle this data. Nonetheless, we see the CNN generalize better to 07/14 than the LSTM. More empirical hyperparameter tuning is needed on both the LSTM architecture and heterodyning to further investigate why the CNN is performing better with TKEO.

6.2. Ensemble Learning

After observing the strengths and weaknesses of the different variations of the models in Section 6.1, this section reports and discusses results of the stacked ensemble RF combiner that unites these models. Section 6.2.1 will report the results of the different RF combiners with confusion matrices and a table of metrics derived from them. Section 6.2.2 will then discuss these results.

6.2.1. Results

The predictions of models trained from every 7-LODO fold of the three different preprocessing approaches on the CNN and LSTM were used as the training data for the RF combiner. That means that each example is a feature vector of length 42 (3 preprocessing approaches x 2 deep learning models x 7 models for each fold). Figure 6.2.1.1 and Table 6.2.1.1 show the results after 7-LODO on 07/14 with the RF stacked ensemble combiner.

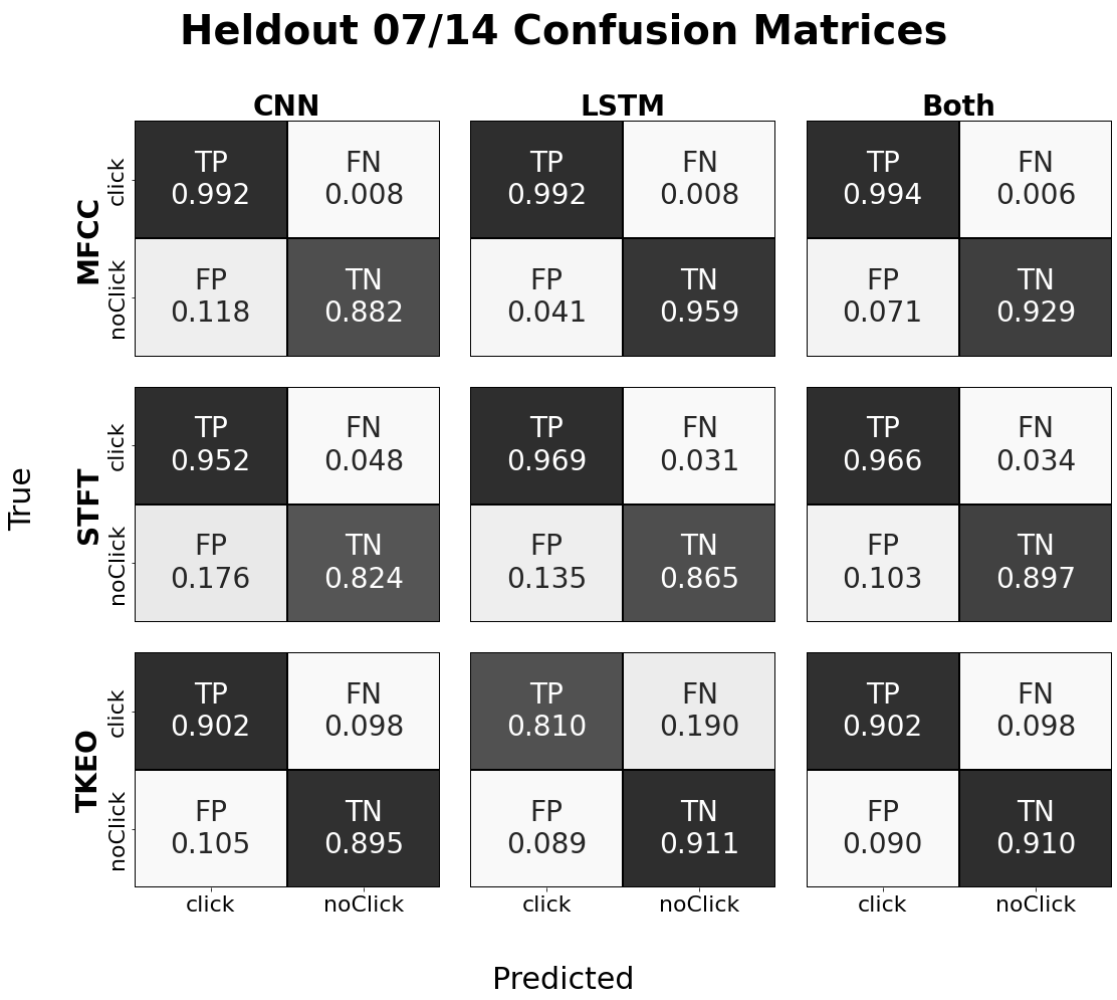

Figure 6.2.1.1. Confusion matrices of the RF *stacked* ensemble combiner on 7-LODO. The left and middle columns of the confusion matrices correspond to the results of the RF combiner trained on the CNN and LSTM family of models, respectively. Each family of models consists of the seven models corresponding to each fold of the 7-LODO. The last column to the right, labeled “Both”, represents the results of the RF combiner when trained on both the CNN and LSTM families of models (a total of 14 models).

		CNN	LSTM	Both
MFCC	**Accuracy**	0.937	**0.975**	0.962
	Precision	0.894	0.960	0.934
	Recall	0.992	0.992	0.994
	F1 Score	0.941	0.976	0.963
		CNN	**LSTM**	**Both**
STFT	**Accuracy**	0.888	0.917	**0.932**
	Precision	0.844	0.878	0.904
	Recall	0.952	0.969	0.966
	F1 Score	0.895	0.922	0.934
		CNN	**LSTM**	**Both**
TKEO	**Accuracy**	0.898	0.861	**0.906**
	Precision	0.896	0.901	0.910
	Recall	0.902	0.810	0.902

Table 6.2.1.1. Results metrics for RF stacked ensemble combiner on 7-LODO. The left column specifies the base models' preprocessing approach and the three columns to the right specify the architectures of the base models. We observed the highest accuracy at 97.5% when the RF combiner was trained on the outputs of LSTM trained on the MFCC inputs. For the STFT base models, the RF combiner did the best when it was trained on the outputs of both architectures with an accuracy of 93.2%.

6.2.2. Discussion

Our RF combiner achieved the best accuracy on the held-out day, 07/14, when the RF it was trained on only the seven models of the LSTM MFCC family of models with an accuracy of 97.5%. This is a 1.9% improvement, or a 56.8% reduction in error, from its initial DL model described in section 6.1. For the CNN family of LSTM models, however, we see only a 0.2% improvement. In combining both the CNN and LSTM families of models, we surprisingly see a 1.3% reduction in accuracy with respect to the LSTM accuracy. Since the CNN performed 2.1%

worse in accuracy than the LSTM, the addition of the seven CNN model's predictions could have confused the RF. It should be noted, however, that although the accuracy decreases when we combine the LSTM and CNN, we see a 0.2% increase in the recall, meaning that we see less FNs.

For the STFTs, combining both the CNN and LSTM family of models results in a 4.5% increase in accuracy, or a 60.2% reduction in error, with respect to the initial DL LSTM STFT family of models. This contradicts the results of the RF with the MFCCs, in that combining the CNN and the LSTM family of models (88.8% and 91.7%, respectively) improves our accuracy to 93.2%.

Similarly for the TKEO, we also see that combining the CNN and LSTM family of models results in the best performance. The RF combiner improves the most accurate architecture, which in the case of TKEO is the CNN, by 5.6% or a 62.7% reduction in error.

Overall, every RF combiner that used both families of models in its training has a better accuracy than either of its original DL families of models that it is built from. This supports our hypothesis that the addition of an ensemble network of decision trees would help reduce the harbor porpoise misclassifications across the three preprocessing methods.

Furthermore, we observe that the misclassification frequencies observed in the set differences (left and right plots of Figures 6.1.1.3.2, 6.1.2.3.2 and 6.1.3.3.2) are proportional to the increases in accuracy observed by the RF combiner. In other words, when the models have more unique misclassifications (and consequently, less intersection amongst their misclassifications), the RF combiner is more likely to improve performance relative to its base models. Figure 6.2.3.1 shows a scatter plot with the y-axis corresponding to the 7-LODO accuracy of the stacked RF combiner models and the x-axis corresponding to 7-LODO of the original DL models (prior to stacking). These trends indicate that with the lower accuracy that the initial DL models had, a greater improvement was captured by the RF combiner.

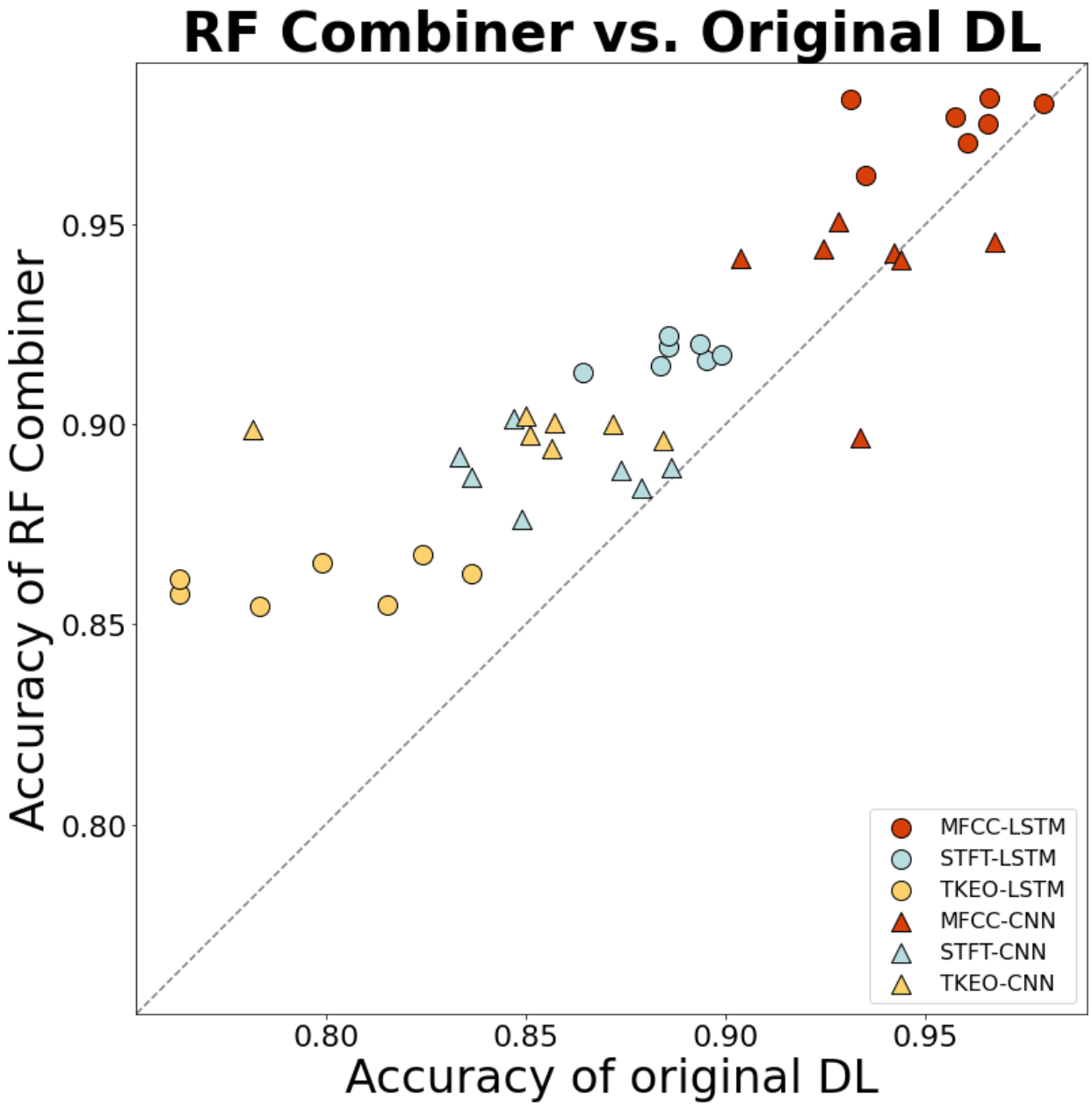

Figure 6.2.3.1. Scatter plot of the accuracies of the original DL and stacked RF. The y-axis represents the final accuracy of the stacked RF combiner and the x-axis corresponds to the accuracies of the base DL models. The CNN models are marked with triangles and the LSTM models are marked with circles. This plot indicates that the lower accuracy that the initial DL models had, the greater the improvement captured by the RF combiner.

7. Conclusion

In this study we implemented multiple ML approaches, all of which investigate the challenge of effectively predicting the presence of ultrasonic harbor porpoise clicks directly from noisy ultrasonic audio data. Our first experimentation of cross-validation strategies supports the existence of a covariate shift in our time-series data across the eight days of audio recording in the Bay of Fundy, Nova Scotia, Canada. Our implementation of LODO CV was the most accurate representation of how our models would perform in the real world that we can create from our limited set of human annotated observations. LODO CV was needed in order to address the covariate shift seen in our dataset. We observed only a 6% difference between the 7-LODO fold accuracy and the distinctive held-out test day (07/14) accuracy. Training without cross-validation and k-fold, in comparison, had a difference of 28.7% and 30.4%, respectively.

After establishing the need for LODO CV on our data, we compared six different models: an LSTM and CNN, each trained with MFCC, STFT and heterodyned-TKEO preprocessed features and methods. From this experiment, we found that the LSTM with MFCC ($LSTM_{MFCC}$ where the subscript is the preprocessing method) input was the most robust model achieving the highest accuracy on our held-out test day, 07/14. Other than the MFCCs, the next highest accuracies were achieved by the $LSTM_{STFT}$, CNN_{STFT} and CNN_{TKEO} models with accuracies of 88.7%, 85.8% and 85.0% on the held-out dataset, respectively.

In the analysis of these results with LODO CV, however, we also discovered that some models performed better on different days. This observation informed our last experiment that explored different methods of combining all the models in an ensemble, including each LODO fold's model, with the goal of leveraging the learned strengths of each individual model and ultimately further improving the overall accuracy of our system. Our hypothesis was correct in that the stacked ensemble improved the performance and led to almost perfect prediction of harbor porpoise clicks. We found that accuracies of the baseline models were inversely proportional to the improvement that the random forest (RF) combiner contributed. Consequently, the RF combiner trained on both the CNN_{TKEO} and $LSTM_{TKEO}$ model outputs showed the greatest improvement from the held-out test set, 07/14, with an increase in accuracy of 6% with respect to the CNN_{TKEO} base models. The RF combiner trained with CNN_{STFT} and $LSTM_{STFT}$ models showed

a 4.5% improvement in accuracy with respect to the CNN_{STFT} base models. The RF combiner trained with the CNN_{MFCC} model outputs resulted in a 1.9% improvement in accuracy.

The analysis of these results build upon related works discussed in Chapter 2. LODO is a common approach seen in time series data and our study proposed applying this cross-validation approach when training models on harbor porpoise clicks in order to potentially improve generalizability to real-world conditions. Heterodyning is a common processing method for bat call analysis and our study introduces this approach as an implementation for classification of marine mammals. Decision trees and RFs have previously been used in harbor porpoise classification and our investigation proposed implementing an ensemble of the outputs of 42 models as the inputs to a RF to improve the robustness of our classifier and increase potential generalization.

The inclusion of this ensemble method in the pipeline, however, is an extra step in processing and may not be as readily feasible in real-time inference due to processing time constraints. Future work should include further tuning of the preprocessing methods' parameters to observe whether a similar performance improvement can be attained without the implementation of an ensemble. Another limitation that should be noted is the limited number of examples collected in July. We collected a total of 1,848 examples in July while, in comparison, 6,438 examples were collected in September. Beyond the month of July, our models could be significantly more robust if they were trained on a more balanced dataset, representing multiple days and conditions throughout the entire year. Future research should include more annotations from different times of the year to get the best representation of the soundscape.

Future investigations should also examine the correlation between tide cycles in the Bay of Fundy and the general background noise in our audio. From the LTSAs in Figure 4.3.3, we see four distinct periods of noise (yellow humps) that we believe correspond to the inflow and outflow of the tides. Including weather and hydraulic information such as speed, direction, acceleration and water level could better inform our classifier to interpret the noise from any given example and could potentially lead to higher accuracies on signals with low signal-to-noise ratios.

The energy from these tides that we are observing in the form of noise in our signal (see LTSAs in Figure 4.3.3) is being harnessed by water turbines in the Bay of Fundy. These water turbines pose a serious threat to the well being of the harbor porpoises inhabiting the area. Not only can the blades of the turbines damage the porpoises passing by, the mechanical noise also

disrupts their echolocation and can prevent them from successfully navigating and foraging in this human-altered marine environment. The results of this research help build towards the larger goal of creating an effective classifier that can autonomously and quickly detect porpoise clicks in the soundscape. Approaches such as the classifier developed in this study could be incorporated into a real-time system that can send a halt signal to the water turbines whenever a porpoise click is detected. Such a system would enable the water turbines to safely harness tidal energy while greatly reducing the threats to the local harbor porpoise population.

Bibliography

[1] A. W. Archer, “World’s highest tides: Hypertidal coastal systems in North America, South America and Europe,” *Sediment. Geol.*, vol. 284–285, pp. 1–25, Feb. 2013, doi: 10.1016/j.sedgeo.2012.12.007.

[2] Z. Zhou, M. Benbouzid, J.F. Charpentier, F. Scuiller, and T. Tang, “Developments in large marine current turbine technologies – A review,” *Renew. Sustain. Energy Rev.*, vol. 71, pp. 852–858, May 2017, doi: 10.1016/j.rser.2016.12.113.

[3] L.A. Miller and M. Wahlberg, “Echolocation by the harbour porpoise: life in coastal waters,” *Front. Physiol.*, vol. 4, p. 52, Apr. 2013, doi: 10.3389/fphys.2013.00052.

[4] D. Tollit, R. Joy, J. Wood, A. Redden, and C. Booth, “Baseline presence of and effects of tidal turbine installation and operations on harbour porpoise in Minas Passage, Bay of Fundy, Canada” *J. Ocean Technol.*, vol. 14, p. 24, Aug. 2019.

[5] W. W. L. AU, “Echolocation in dolphins with a dolphin-bat comparison,” *Bioacoustics*, vol. 8, no. 1–2, pp. 137–162, Jan. 1997, doi: 10.1080/09524622.1997.9753357.

[6] W. W. L. Au, “Biosonar discrimination, recognition, and classification,” in *The Sonar of Dolphins*, W. W. L. Au, Ed. New York, NY: Springer, 1993, pp. 177–215. doi: 10.1007/978-1-4612-4356-4_9.

[7] NOAA Fisheries, “Harbor Porpoise | NOAA Fisheries,” *NOAA*, Apr. 20, 2022. Available: https://www.fisheries.noaa.gov/species/harbor-porpoise (accessed May 17, 2022).

[8] S. Andersen and M. Amundin, “Possible predator-related adaptation of sound production and hearing in the harbor porpoise (Phocoena phocoena),” *Aquatic Mamm*als, vol. 4, pp. 56–57, 1976.

[9] P. T. Madsen, D. Carder, K. Bedholm, and S. Ridgway, “Porpoise clicks from a sperm whale nose—Convergent evolution of 130 kHz pulses in toothed whale sonars?,” *Bioacoustics*, vol. 15, no. 2, pp. 195–206, 2005.

[10] P. M. Sørensen, D. M. Wisniewska, F. H. Jensen, M. Johnson, J. Teilmann, and P. T. Madsen, “Click communication in wild harbour porpoises (Phocoena phocoena),” *Scientific Reports*, vol. 8, no. 1, Art. no. 1, Jun. 2018, doi: 10.1038/s41598-018-28022-8.

[11] J. C. Steinberg, “Positions of stimulation in the cochlea by pure tones,” *Journal of the Acoustic Society of America*, vol. 8, no. 3, pp. 176–180, Jan. 1937, doi: 10.1121/1.1915891.

[12] J. F. Kaiser, “On a simple algorithm to calculate the ‘energy’ of a signal,” in *International Conference on Acoustics, Speech, and Signal Processing*, Apr. 1990, pp. 381–384 vol.1. doi: 10.1109/ICASSP.1990.115702.

[13] S. Solnik, P. Rider, K. Steinweg, P. DeVita, and T. Hortobágyi, “Teager–Kaiser energy operator signal conditioning improves EMG onset detection,” *European Journal of Applied Physiology.*, vol. 110, no. 3, pp. 489–498, 2010, doi: 10.1007/s00421-010-1521-8.

[14] V. Kandia and Y. Stylianou, “Detection of sperm whale clicks based on the Teager–Kaiser energy operator,” *Applied Acoustics*, vol. 67, no. 11, pp. 1144–1163, Nov. 2006, doi: 10.1016/j.apacoust.2006.05.007.

[15] W. Loh, “Fifty years of classification and regression trees,” *International Statistical Review.*, vol 82, no. 3, Decmeber 2014, doi: 10.1111/insr.12016.

[16] Y. Freund and R. E. Schapire, “Large margin classification using the perceptron algorithm,” *Machine Learning*, vol. 37, no. 3, pp. 277–296, Dec. 1999, doi: 10.1023/A:1007662407062.
[17] J. S. Cramer, “The origins of logistic regression,” Tinbergen Institute Working Paper, 02-119/4, Dec. 2002, doi: 10.2139/ssrn.360300.
[18] D. J. Hand and K. Yu, “Idiot’s Bayes—not so stupid after all?,” *International Statistics Review*, vol. 69, no. 3, pp. 385–398, 2001, doi: 10.1111/j.1751-5823.2001.tb00465.x.
[19] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Language*, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1023/A:1022627411411.
[20] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Aug. 1995, vol. 1, pp. 278–282 vol.1. doi: 10.1109/ICDAR.1995.598994.
[21] O. Sagi and L. Rokach, “Explainable decision forest: Transforming a decision forest into an interpretable tree,” *Information Fusion*, vol. 61, pp. 124–138, Sep. 2020, doi: 10.1016/j.inffus.2020.03.013.
[22] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, doi: 10.1016/j.neunet.2014.09.003.
[23] S. Amari, “Backpropagation and stochastic gradient descent method,” *Neurocomputing*, vol. 5, no. 4, pp. 185–196, Jun. 1993, doi: 10.1016/0925-2312(93)90006-O.
[24] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” In proc. Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, pp 315-323, January, 2010, doi:
[25] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–44, May 2015, doi: 10.1038/nature14539.
[26] S. Balaji, “Binary image classifier CNN using TensorFlow,” *medium.com*, Aug. 29, 2020. [Online]. Available: https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697 [accessed May 18, 2022].
[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Dec. 1998, doi: 10.1109/5.726791.
[28] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” International Conference on Learning Representations, Apr. 2015.
[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.
[30] C. Olah, “Understanding LSTM networks -- colah’s blog.” August, 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [accessed Apr. 05, 2022].
[31] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions Neural Networks.*, vol. 5, no. 2, pp. 157–166, Mar. 1994, doi: 10.1109/72.279181.
[32] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computing*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
[33] K. Fristrup and W. Watkins, “Characterizing acoustic features of marine animal sounds (Technical Report WHOI-92-04),” Jan. 1992, doi: 10.1575/1912/3055.

[34] M. P. Mcloughlin, R. Stewart, and A. G. McElligott, “Automated bioacoustics: methods in ecology and conservation and their potential for animal welfare monitoring,” *Journal of the Royal Society Interface*, vol. 16, no. 155, p. 20190225, Jun. 2019, doi: 10.1098/rsif.2019.0225.
[35] V. Kandia and Y. Stylianou, “Detection of creak clicks of sperm whales in low SNR conditions,” *European Oceans* , 2005, doi: 10.1109/OCEANSE.2005.1513203.
[36] M. Bittle and A. Duncan, “A review of current marine mammal detection and classification algorithms for use in automated passive acoustic monitoring,” Proceedings of Acoustics: Science, Technology and Amenity, pp. 1-8. Nov 17-20 2013.
[37] G. Mirzaei *et al.*, “The BIO-acoustic feature extraction and classification of bat echolocation calls,” *IEEE International Conference Electro Information Technology.*, pp. 1–4, May 2012, doi: 10.1109/EIT.2012.6220700.
[38] “Heterodyne and time-expansion methods for identification of bats in the field,” 2004. Department of Conservation Biology. July, 2004.
[39] R. Brigham, E. Kalko, G. Jones, S. Parsons, and H. Limpens, *Bat Echolocation Research. Tools, Techniques and Analysis*. Edited by Bat Conservation International, Austin, TX, 2004.
[40] K. M. Fristrup and W. A. Watkins, “Marine animal sound classification,” Woods Hole Oceanographic Institution, Technical Report, Oct. 1993. doi: 10.1575/1912/546.
[41] J. Ayala-Berdon *et al.*, “Random forest is the best species predictor for a community of insectivorous bats inhabiting a mountain ecosystem of central Mexico,” *Bioacoustics*, vol. 30, no. 5, pp. 608–628, Sep. 2021, doi: 10.1080/09524622.2020.1835539.
[42] J. Ross and P. Allen, “Random Forest for improved analysis efficiency in passive acoustic monitoring,” *Ecological Information.*, vol. 21, Jan. 2013, doi: 10.1016/j.ecoinf.2013.12.002.
[43] Y.-C. Tseng, B. N. I. Eskelson, K. Martin, and V. LeMay, “Automatic bird sound detection: logistic regression based acoustic occupancy model,” *Bioacoustics*, vol. 30, no. 3, pp. 324–340, May 2021, doi: 10.1080/09524622.2020.1730241.
[44] J. Noda, C. Travieso, D. Sanchez-Rodriguez, M. Dutta, and A. Singh, “Using bioacoustic signals and Support Vector Machine for automatic classification of insects,” Feb. 2016, pp. 656–659. doi: 10.1109/SPIN.2016.7566778.
[45] D. Chesmore, “Automated bioacoustic identification of species,” *Anais da Academia Brasileira*, vol. 76, pp. 436–40, Jul. 2004, doi: 10.1590/S0001-37652004000200037.
[46] F. Caruso *et al.*, “Monitoring of a nearshore small dolphin species using passive acoustic platforms and supervised machine learning techniques,” *Frontiers in Marine Science*, vol. 7, 2020, Accessed: Mar. 29, 2022. doi: 10.3389/fmars.2020.00267
[47] M. Cosentino, F. Guarato, J. Tougaard, D. Nairn, J. C. Jackson, and J. F. C. Windmill, “Porpoise click classifier (PorCC): A high-accuracy classifier to study harbour porpoises (Phocoena phocoena) in the wild,” *Journal of Acoustic Society of America*, vol. 145, no. 6, pp. 3427–3434, Jun. 2019, doi: 10.1121/1.5110908.
[48] D. Gillespie *et al.*, “PAMGUARD: Semiautomated, open source software for real-time acoustic detection and localization of cetaceans.,” *Journal of Acoustic Society of America*, vol. 125, no. 4, pp. 2547–2547, Apr. 2009, doi: 10.1121/1.4808713.
[49] Y. Shiu *et al.*, “Deep neural networks for automated detection of marine mammal species,” *Sci. Rep.*, vol. 10, no. 1, Art. no. 1, Jan. 2020, doi: 10.1038/s41598-020-57549-y.
[50] Y. LeCun *et al.*, “Backpropagation applied to handwritten zip code recognition,” *Neural Computing*, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.
[51] S. Kahl, C. M. Wood, M. Eibl, and H. Klinck, “BirdNET: A deep learning solution for avian

diversity monitoring," *Ecological Information*, vol. 61, p. 101236, Mar. 2021, doi: 10.1016/j.ecoinf.2021.101236.
[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," IEEE Conference on Computer Vision and Pattern Recognition, Dec. 2015. doi: 10.48550/arXiv.1512.03385.
[53] P. C. Bermant, M. M. Bronstein, R. J. Wood, S. Gero, and D. F. Gruber, "Deep machine learning techniques for the detection and classification of sperm whale bioacoustics," *Scientific Reports*, vol. 9, no. 1, Art. no. 1, Aug. 2019, doi: 10.1038/s41598-019-48909-4.
[54] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," IEEE Conference on Computer Vision and Pattern Recognition, Jan. 2018. doi: 10.48550/arXiv.1608.06993.
[55] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv, arXiv:1704.04861, Apr. 2017. doi: 10.48550/arXiv.1704.04861.
[56] M. Thomas, B. Martin, K. Kowarski, B. Gaudet, and S. Matwin, "Marine mammal species classification using convolutional neural networks and a novel acoustic representation," *Machine Learning and Knowledge Discovery in Databases*, Cham, 2020, pp. 290–305. doi: 10.1007/978-3-030-46133-1_18.
[57] D. Duan *et al.*, "Real-time identification of marine mammal calls based on convolutional neural networks," *Applied Acoustics*, vol. 192, p. 108755, Apr. 2022, doi: 10.1016/j.apacoust.2022.108755.
[58] J. Li, M. Abdulrahman, G. Zweig, and Y. Gong, "LSTM time and frequency recurrence for automatic speech recognition," *2015 IEEE Workshop Automatic Speech Recognition and Understing*, Dec. 2015, doi: 10.1109/ASRU.2015.7404793.
[59] C.-S. A. Gong, C.-H. S. Su, K.-W. Chao, Y.-C. Chao, C.-K. Su, and W.-H. Chiu, "Exploiting deep neural network and long short-term memory method-ologies in bioacoustic classification of LPC-based features," *PLOS ONE*, vol. 16, no. 12, p. e0259140, Dec. 2021, doi: 10.1371/journal.pone.0259140.
[60] F. Weninger and B. Schuller, "Audio recognition in the wild: Static and dynamic classification on a real-world database of animal vocalizations," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 337–340. doi: 10.1109/ICASSP.2011.5946409.
[61] D. Duan, "Detection method for echolocation clicks based on LSTM networks," *Mobile Information Systems*, vol. 2022, p. e4466037, Mar. 2022, doi: 10.1155/2022/4466037.
[62] Daniel Hasselman, Tyler Boucher, and Jessica Douglas, "Data Analysis Component of Comparative Passive Acoustic Monitoring (PAM) Technology Assessment," Fundy Ocean Research Center of Energy, Halifax, NS, Tech. Report. 1 June, 2020.
[63] D. K. Mellinger, "Acoustic feature extraction and classification in Ishmael," *Journal of . Acoustic Society of America.*, vol. 134, no. 5, pp. 3986–3986, Nov. 2013, doi: 10.1121/1.4830526.
[64] "SoX - Sound eXchange | HomePage." [online]. Availabe: http://sox.sourceforge.net/. [accessed May 18, 2022].
[65] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," arXiv, arXiv:1309.0238, Sep. 2013. doi: 10.48550/arXiv.1309.0238.
[66] Y. Sasaki, "The truth of the F-measure," *Teach Tutor Mater*, Jan. 2007.

[67] V. Velardo, *musikalkemist/DeepLearningForAudioWithPython*. 2022. Accessed: Apr. 19, 2022. [Online]. Available: https://github.com/musikalkemist/DeepLearningForAudioWithPython/blob/master/16-%20How%20to%20implement%20a%20CNN%20for%20music%20genre%20classification/code/cnn_genre_classifier.py

[68] L. Roeder, *Netron, Visualizer for neural network, deep learning, and machine learning models*. 2017. doi: 10.5281/zenodo.5854962.

[69] V. Velardo, *musikalkemist/DeepLearningForAudioWithPython*. 2022. Accessed: Apr. 19, 2022. [Online]. Available: https://github.com/musikalkemist/DeepLearningForAudioWithPython/blob/master/19-%20How%20to%20implement%20an%20RNN-LSTM%20for%20music%20genre%20classification/code/19-%20How%20to%20implement%20an%20RNN-LSTM%20for%20music%20genre%20classification.py

[70] Decumanus at English Wikipedia, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=11123462

[71] Mellinger, D. (2022) heterodyne (https://www.mathworks.com/matlabcentral/fileexchange/112100-heterodyne), MATLAB Central File Exchange. Retrieved May 23, 2022.

Appendices

A. Dataset tables

timestamp	count	mean	std	min	50%	max
07/14						
144050	313	0.623	0.081	0.421	0.637	0.796
144250	326	0.670	0.096	0.323	0.676	0.843
184650	285	0.343	0.048	0.225	0.343	0.479
09/05						
222446	314	0.308	0.040	0.181	0.310	0.416
233646	23	0.274	0.058	0.193	0.272	0.447
09/07						
112847	212	0.534	0.072	0.247	0.556	0.625
113047	557	0.482	0.147	0.058	0.550	0.628
09/08						
115047	21	0.370	0.039	0.321	0.359	0.435
142847	313	0.328	0.068	0.070	0.340	0.454

09/09						
013247	522	0.351	0.047	0.249	0.348	0.462
013447	25	0.388	0.056	0.273	0.396	0.481
020847	240	0.341	0.054	0.196	0.337	0.480
021047	44	0.327	0.049	0.244	0.331	0.442
09/10						
022648	213	0.346	0.044	0.216	0.345	0.456
09/11						
030849	147	0.466	0.067	0.346	0.459	0.638
031249	214	0.369	0.040	0.232	0.371	0.466
132049	46	0.332	0.043	0.273	0.325	0.416
09/12						
091449	17	0.530	0.223	0.273	0.677	0.819
092649	50	0.745	0.064	0.601	0.766	0.822
094449	22	0.621	0.075	0.452	0.601	0.714
100249	93	0.381	0.047	0.290	0.386	0.479
100449	12	0.371	0.064	0.260	0.397	0.443

162049	60	0.427	0.040	0.353	0.425	0.505
162449	24	0.371	0.091	0.202	0.394	0.498
163249	50	0.365	0.053	0.287	0.352	0.497

Table 1. Click dataset profile of the average RMS_s statistics for each segment of each day. The 50% is the 50th percentile is equivalent to the median of RMS_s in that that given segment.

	count	**mean**	**std**	**min**	**50%**	**max**
07/14						
144450	313	0.609	0.101	0.391	0.582	0.819
144850	326	0.687	0.079	0.273	0.701	0.826
185550	285	0.349	0.050	0.217	0.343	0.475
09/05						
222246	314	0.333	0.047	0.160	0.332	0.451
233446	23	0.335	0.062	0.228	0.332	0.459
09/07						
110247	212	0.409	0.061	0.346	0.421	0.488
111847	557	0.485	0.035	0.391	0.483	0.539
09/08						

115247	21	0.330	0.065	0.158	0.350	0.394
143247	313	0.351	0.044	0.103	0.354	0.456
09/09						
012847	522	0.353	0.049	0.224	0.356	0.452
013847	25	0.313	0.111	0.042	0.339	0.436
020647	240	0.338	0.065	0.039	0.346	0.465
021447	44	0.340	0.077	0.089	0.349	0.465
09/10						
021048	213	0.557	0.080	0.338	0.605	0.630
09/11						
025049	147	0.668	0.140	0.099	0.726	0.792
033849	214	0.344	0.063	0.050	0.345	0.456
132249	46	0.343	0.075	0.092	0.350	0.458
09/12						
075249	17	0.294	0.104	0.045	0.329	0.436
080249	50	0.347	0.066	0.195	0.348	0.443
080849	22	0.355	0.058	0.192	0.373	0.437

110449	93	0.347	0.056	0.066	0.350	0.455
111049	12	0.286	0.108	0.052	0.325	0.409
153049	60	0.385	0.069	0.235	0.378	0.514
154049	24	0.455	0.061	0.367	0.471	0.601
173049	50	0.332	0.046	0.250	0.327	0.426

Table 2. `noClick` dataset profile of the average RMS_s statistics for each segment of each day. The 50% is the 50th percentile is equivalent to the median of RMS_s in that given segment.

Day	count	mean	std	min	50%	max
07/14	1848	0.555	0.163	0.217	0.587	0.843
09/05	674	0.320	0.047	0.160	0.320	0.459
09/07	1538	0.480	0.103	0.058	0.502	0.628
09/08	668	0.340	0.058	0.070	0.350	0.456
09/09	1662	0.347	0.055	0.039	0.347	0.481
09/10	426	0.452	0.124	0.216	0.417	0.629
09/11	814	0.430	0.143	0.050	0.389	0.792
09/12	656	0.410	0.133	0.045	0.382	0.822

Table 3. Dataset profile of the entire dataset, per day. This includes `clicks` and `noClicks`.