Efficient Detection for Acoustic Biodiversity Monitoring

Max Oswin

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

Acoustic monitoring has become an important field of research for the purposes of collecting data on the populations of various animal species. Bats are a particular focus, as they are extremely sensitive to changes in their environment, meaning changes in their populations can serve as an effective warning system for vulnerable ecosystems. As such, it is deeply important to have efficient, autonomous detection methods to collect timely information on these species.

This report introduces a new pipeline for the detection and classification of bat species via the structure of their echolocation calls, based upon previous work by Mac Aodha et al. [26]. The pipeline makes use of the Goertzel algorithm to reduce the amount of time spent on spectrogram generation, an expensive operation that is typically used in autonomous analysis of bat calls.

Convolutional neural network bat detectors are compared to decision tree detectors which implement the Goertzel algorithm. We further compare the relative species classification performance for these detection methods, as classification is based upon successful detections.

We find that our tree-based Goertzel algorithm detection methods lead to a decrease in the average amount of spectrogram generation by between 6.4 to 9.1 times, indicating an increase in efficiency. However, the decision tree methods lead to a small decrease in classification performance when compared to the standard convolutional method.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Max Oswin*)

Acknowledgements

Many thanks to my project supervisor, Oisin Mac Aodha, who was incredibly helpful and supportive throughout the entire process behind this report.

Another round of thanks go to my family, friends, and partner, who I all love dearly, and all of whom kept cheering for me throughout my final year of study at The University of Edinburgh.

Many thanks to Sebastien Piquemal for their open-source implementation of the Goertzel Algorithm in python.

Finally, thank you to my cat Muffin for being adorable!

Table of Contents

Chapter 1

Introduction

1.0.1 Acoustic Biodiversity Monitoring

As the climate crisis continues, it is becoming increasingly important to protect the planet's diverse range of wildlife [11] [6]. Measurement of the current populations of various species is key to understanding how to protect the most vulnerable animals within an ecosystem. Modern efforts on this front are becoming increasingly reliant on technology; one such example is acoustic biodiversity monitoring.

This practice utilises in-the-field microphones to listen for and record the activity of many different types of animal. Species are typically distinguished from background noise and between one another by their calls; as such, this method is particularly useful for monitoring species of birds, amphibians, and mammals. Bat species are of a particular interest, as they are often excellent indicator species [21]. Their populations tend to be particularly sensitive to changes in the environment - thus, by keeping a close eye on their numbers, researchers can understand if a habitat is under threat.

Rather than being compared as raw wave-forms, recordings are transformed into two-dimensional signals of time and frequency, also known as 'spectrograms'. When visualised (as in Figure 1.1), these signals display the combination of simple sine-wave frequencies that make up a given complex sound wave over time. Consequently, they can be used to analyse structural information of an individual call, and leave identifiable clues to the originating species. However, even with expert analysis, identification is not so simple; not all species can reliably be distinguished quantitatively, and qualitative analysis can be more of an "art" than "science", due to inherent subjectivity [39]. Furthermore, research programmes which use acoustic monitoring can produce hundreds of hours of audio per day over multiple recording devices [38]. Not only does the sheer quantity of data become expensive to store, it necessitates thousands of hours of careful human attention in order to produce accurate counts of species population. Considering Frick et al. [11] found that 988 bat species required conservation or research attention in 2019, it is clear that such a time-intensive process is incompatible with rapid environmental change. Responding quickly to species under existential threat is therefore crucial.

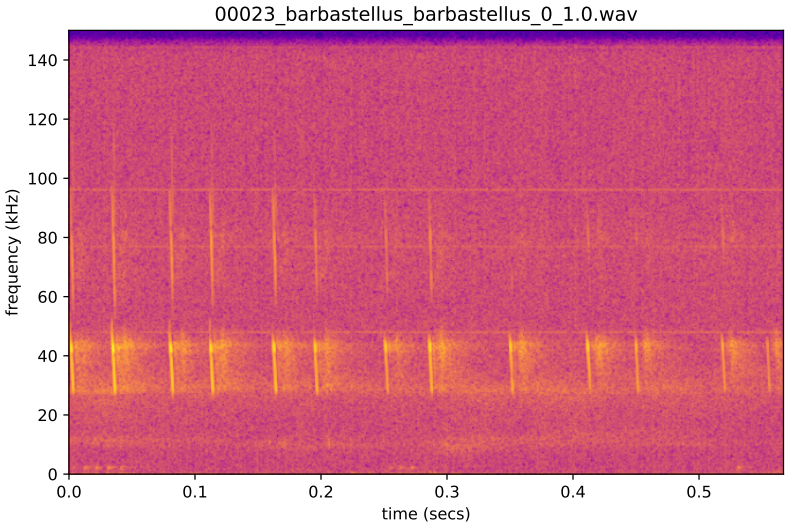

Figure 1.1: *An example time-frequency visualisation (i.e., a spectrogram) of a series of bat echolocation calls. The title of the plot refers to a file within the dataset, and contains the species making the calls, 'Barbastellus barbastellus'. The colour of each pixel refers to the amplitude of a frequency at a given time. Each orange diagonal line is an separate call made by a member of this species. The descending slope structure is found throughout many bat species. Given the structured repetition of the calls, it is most likely that only one individual is present in this recording.*

1.0.2 Acoustic Monitoring with Deep Learning

Deep learning acts to alleviate many of the aforementioned problems. Multilayered neural networks trained on expertly-labelled data can make accurate decisions regarding the detection and classification of recorded species, in lieu of human experts themselves. Furthermore, they can come to conclusions in a fraction of the time taken for human analysis. This makes deep learning methods a prime candidate for delivering high-quality data analysis within the crucial time-frame this research space requires.

However, the current deep learning method is not without its caveats. Computational generation and analysis of time-frequency diagrams is already resource intensive; Mac Aodha et al. [26] found that spectrogram generation was the most computationally expensive stage of their deep learning pipeline. The addition of neural networks only increases the resources required. For acoustic monitoring projects focused on a wide area, lowered budget, and the fastest results possible, immediate computational analysis onboard in-the-field listening devices is ideal. Such devices are typically of limited computing power, increasing the need for efficient deep learning algorithms and processes.

1.0.3 Project Overview and Findings

This project looks at increasing the efficiency of the deep learning acoustic detection and classification pipeline. Specifically, we investigate the addition of the Goertzel

algorithm with decision trees, in an effort to reduce unnecessary time-frequency diagram generation and save resources.

Our efforts resulted in a modified version of the pipeline introduced by Mac Aodha et al. [26] which optionally replaces DNN detection with one of many decision trees using the Goertzel algorithm. The Goertzel algorithm is able to detect the power-spectrum of single frequencies without generating a full spectrogram. Previous work has determined that this is more efficient than full spectrogram generation, if done for a small number of frequencies per input sequence [25]. By using decision trees to check for a specific few frequencies at regular intervals across an audio file, we aimed to drastically reduce the amount of spectrogram generation and increase pipeline efficiency. The pipeline is further adapted for the species classification task.

We found that our decision tree Goertzel algorithm methods lead to a decrease in the average amount of spectrogram generation by between 6.4 to 9.1 times, in comparison to the typical approach of processing all audio as spectrograms. However, this came at a decrease in classification recall at the genus level by up to 5%, and significant loss in detector performance depending on the input size and depth of the decision tree used. We also find that decision trees with small input sizes, equivalent to 1 temporal bin in a spectrogram, generally perform better than trees that take larger sequences of audio as input.

Chapter 2 gives an overview of the previous research done on bio-acoustic monitoring, covering both autonomous and manual human methods. Chapter 3 then takes a look into the specific algorithms used for processing audio within these systems, before pivoting to discuss the machine learning components used within our pipeline design, and concluding with a breakdown of the pipeline as a whole. Chapter 4 discusses the datasets used for training the aforementioned machine learning models, as well as the datasets used in chapter 5, where the different configurations of the pipeline are compared, and pipeline components are evaluated. Finally, chapter 6 provides some closing thoughts on the project, and discusses possible directions for future work.

Chapter 2

Background

2.1 Acoustic Detection and Analysis

2.1.1 Means of Classification

The characteristics of a species are key to determining how they may be detected and classified audibly. For example, bats make several types of call depending on their activity; for a sound comparison between species, calls of the same structure should be compared [5]. Search-phase calls are emitted when a bat is searching for prey, as well as general navigation, and are dominant choice for classification via audio [28]. This is due to their relative abundance when compared to other types of feeding call. They also feature enough inter-species variation to distinguish between species with high accuracy [32]. Some variation in calls exists within species due to age, sex, location, and environment [22] [15]. Physical phenomena, such as atmospheric pressure and the Doppler effect, introduce further variation, though often minimally so [28]. The extent of intraspecies variation is species specific; this makes certain species more easily confused with others. Likewise, for interspecies variation, outlier species that produce very low or high frequency calls are easily identifiable. However, the differences between the calls of more median bat species may prove more subtle. In general, bat species belonging to the same genus are more likely to have similar call structures than those of different genera. This means distinguishing between genera of bats is generally a simpler task than identifying individual species [5].

2.1.2 Acoustic Detection

Calls are detected with the use of ultrasonic acoustic monitoring devices. Commercial acoustic detection hardware, such as the Anabat II detector for detecting bat calls, has been used in research since the 1990s [32]. However, such hardware typically comes with high fees. Recently, more affordable and open source approaches have become available, such as AudioMoth, a small battery powered device developed by Open Acoustic Devices [18].

A common problem at this stage is the question of separation of target phenomena from

other ambient sounds found in the environment at night, such as human activity, fauna, and background noise. A simple approach is to utilise ‘threshold energy detection’, or a ‘noise gate’, such that only sounds above a certain decibel threshold are recorded [43]. Whilst this method does well to reduce the amount of data requiring analysis, it runs the risk of ignoring faint, distant, or muffled acoustic phenomena.

An alternative approach is to use software to automatically search through the data generated by these devices, as well as to visualise the results. Software packages such as Sonobat can greatly reduce the time required by a human classifier trying to find bat calls present in recorded audio [46]. However, these software packages are often closed source; as a result, it is difficult to understand the methods they use to detect such calls.

Deep learning methods can provide a great alternative to closed source detection methods. For example, they may mark small individual packets or samples of audio as either containing an acoustic phenomena of interest or not [26], or predict the location and size of calls continuously [27]. Many of these approaches are open source, allowing for a deeper understanding than the algorithms of closed-source software. However, beyond low complexity, deep learning models themselves are ’black box’; we can only know about their structure and training processes, not how they specifically make decisions [35].

2.1.3 Acoustic Analysis

2.1.3.1 Spectrogram Generation

Raw audio waveforms lack the information on frequency necessary to determine the species of animals such as bats. To remedy this, the short-time Fourier transform of the waveform can be taken. This process takes the audio signal into the frequency domain, separating the waveform into its constituent frequencies. A spectrogram is a matrix that contains the square-magnitude of these frequencies, given an input frequency and time [3]. Typically, spectrograms are visualised in two dimensions, with colour used to portray magnitudes.

Spectrograms are common throughout acoustic monitoring due to their simple generation process. However, they can be expensive to create - Mac Aodha et al. [26] found that spectrogram generation was the major bottleneck in their ’Bat Detect’ classification process when analysing live audio. This is an issue, as frequency analysis is a key part of species identification. Little work has been done on autonomously detecting species from raw audio. Fortunately, alternative methods for calculating frequency are possible; section 2.1.3.3 discusses one such method, the Goertzel algorithm, in more detail.

2.1.3.2 MFCs

The Mel-Frequency Cepstrum (MFC) of a waveform is an alternative approach to spectrogram representation. These show the ‘power spectrum’ of a signal, and require extra processing beyond the initial application of the fourier transform. Though the more complex generation process than spectrograms implies a greater bottleneck if applied to live data, MFCs shine at dimensionality reduction. Alipek et al. [2] found

that the use of MFCs reduced the size of their data by 98%, a reduction that allowed them to efficiently train their classification network on audio clips orders of magnitude longer than in previous work.

2.1.3.3 Goertzel Algorithm

The Goertzel algorithm, developed by Gerald Goertzel [14], excels at detecting the prominence of single frequencies within sound waves. Furthermore, for a small number of tones given the sample size, it is more efficient than generating a full spectrogram [25] or MFC. This makes it a perfect candidate for use in low-power efficient systems. Section 3.1.3 discusses the algorithm in more detail.

The Goertzel algorithm has also been applied to bio-acoustic monitoring in previous work. For example, Prince et al. [38] apply the algorithm to three different monitoring problems, including the detection of bat calls. They employed a 'sliding window' method, similar to Mac Aodha et al. [26], where the detector progressively iterated over each audio recording. At each step, the median output a Goertzel filter was compared to a pre-tuned threshold, and if the threshold was met, the sliding window would continue. If a sufficient number of thresholds were met in a row, then the audio was determined to contain a bat call.

Prince et al. [38]'s methods performed well, achieving an F_1 score of over 0.96 when detecting bats of the UK-native 'Pipistrellus Pygmaeus' species. However, this test evaluation featured no possible false positives, which throws the impressive results into question. Furthermore, they only tuned and tested their detection method for this single species. In chapter 3, we improve upon this by applying the Goertzel algorithm with decision trees, which allow for the comparison of audio across a dynamic path of frequencies and thresholds. This allows us to detect many different call structures and species with a single detection method.

2.1.3.4 Analysis and Classification

The classification of species calls is traditionally done via expert analysis. For bats, indicators of species include the duration, frequencies, and slope of the call [32]. Expert analysis is paramount in the creation of data sets used for many variations of deep learning classifiers.

Algorithmic classification methods are commonly applied to acoustic classification, but with mixed results. For example, Armitage and Ober [4] found that discriminant function analysis, a common technique for classification in statistics, performed poorly at bat species classification when compared to other supervised learning techniques, including neural networks.

2.2 Machine Learning Techniques

2.2.1 Classical Machine Learning

As briefly discussed in the previous section, there are several other machine learning techniques besides neural networks. Many of these techniques predate the rapid developments in deep learning within the last 15 years; we shall refer to these techniques as 'classical machine learning'.

In this section, we shall touch upon a few common approaches that are used as common benchmarks of deep neural network ability. A sufficiently-trained deep learning network will almost always outperform the forthcoming techniques. However, the simpler structure of classical methods allows them to be deployed quickly in comparison. They also tend to be less computationally intensive.

2.2.1.1 Random Forests

Random forest classifiers work by generating k random decision tree classifiers T_k via a 'bootstrap sampling' (sampling with replacement) method. Each T_k is trained on a different bootstrap sample, by repeatedly picking m random variables from the sample and splitting nodes on the best variable, or split point, within m [16]. Training is supervised; along with bootstrap sample, trees are supplied the output class that the sample belongs to. Once fully trained, each tree in the forest will analyse input data-points, and cast a vote for the corresponding output class. The class with the most votes across the whole forest is deemed the final output class [7].

One advantage random forests hold over neural networks comes from the fact that they are made up of many estimators. Neural networks risk adjusting their weights to fit their training data too well; doing so decreases their performance on other data sets and disrupts their ability to generalise. While this risk remains for random forests, the law of large numbers means the chance of over-fitting does not increase as the number of estimators increases [7]. Since each random tree may only cast one vote, the effect of an individual tree's fit to the training data is minimal across the whole forest. Furthermore, by limiting the maximum depth that any tree in the forest can 'grow' to, they can be highly robust to noisy data [4].

2.2.1.2 Support Vector Machines

Similarly to neural networks, at the core of a support vector machine is an optimisation problem. For classification tasks, this is maximising the separation between data-points of binary classes with a hyperplane [17]. The hyperplane is constructed by support vectors between neighbouring data-points in a high dimensional feature-space, which the training data has been mapped to. All calculations are performed in input space, and applied to feature space via kernel mapping functions.

The variation in possible kernel functions allows for support vector machines to be adaptable to many different types of problem. For example, Armitage and Ober [4] found that a linear approach was not sufficient for their application of support vector

machines for multi-class bat species classification. Instead, they used a radial-basis function kernel, and classified inputs using a vote-based pairwise comparison method.

2.2.2 Deep Learning

Convolutional neural networks (CNNs) make up the vast majority of neural networks applied to acoustic monitoring research. In this subsection, we will cover the high-level concepts behind neural networks, followed by an explanation of how CNNs differ from other approaches, and why they excel within this domain. All approaches discussed will be through the lens of classification based tasks, as they are the focus of the bio-acoustic monitoring domain.

2.2.2.1 Neural Networks

Neural networks are reliant on layers of 'neurons' that take properties of a data-point or outputs of previous neurons as input, and perform some series of operations on that data before outputting. The details of these operations and outputs vary network to network, depending on the activation function used [42]. All neurons between two layers are connected to one another via a weight matrix X of floating point values, such that the value $X_{i,j}$ describes the strength of the connection between the ith neuron in the previous layer and the jth neuron in the next [45]. Weights can be any real number $X_{i,j} \in \mathbb{R}$. Layers of neurons between the input and output of a network are known as 'hidden layers'; beyond three total layers, networks are reffered to as 'deep neural networks' (DNNs), as these models have multiple 'hidden' layers [45]. Through training, the values of all weights in the network repeatedly update as to minimise a loss function, which describes the distance between the current output and the target output of the network. Popular loss functions for classification include cross-entropy and focal loss [49]. As training data is iterated through, the weights are attuned, the loss function is minimised, and the neural network becomes a more accurate classifier. As mentioned in the previous section, it is possible to 'overfit' to the training data with these machine learning techniques, decreasing the ability the model has to generalise, and thus reducing performance on non-training data sets [29].

Neural Networks can utilise supervised or unsupervised learning techniques. In supervised learning, the desired output value or class label is supplied along with the input, and the network is trained to match the output [34]. In unsupervised learning, no desired output is provided. Instead, the goal is to learn information solely from the properties of inputs, also known as the feature-space, in order to create meaningful classifications [44]. Both approaches have unique advantages when it comes to acoustic biodiversity monitoring. Supervised learning has been, and continues to be, the predominant method used in acoustic monitoring research. However, the creation of labelled data sets is a long and cumbersome task, often requiring experts to hand label data in the case of species classification [26] [2]. Unsupervised and semi-supervised learning allows for this lengthy process to be skipped, or partially skipped, at the risk of having a model not recognise the signals of interest within the data. While rare, such methods have seen some success in the field. For example, Ntalampiras and Potamitis [30] apply variational auto-encoders for the detection of atypical species within environments,

which can grant insight into the migratory patterns of a wide range of species.

2.2.2.2 Convolutional Neural Networks

Typical neural networks are fully connected between layers, as previously described. However, this approach can lead to huge size increases in weight matrices for inputs of a large size, such as images. Large fully-connected models become unwieldy to train, both in terms of time and risk of over-fitting, and also execute inefficiently. Additionally, this architecture does not account for the spatial structure of the input [29]. CNNs can overcome this issue via their application of convolutional filters. We discard the idea that every neuron is connected to every other neuron, and instead allow neurons to share weights. Upon exchanging information from one layer to the next, we take a small weight matrix, also known as a kernel, and systematically move it across an input matrix of neurons, such that the output matrix (commonly called a 'feature map') is the product of the kernel and input at each position of the kernel [34]. This process is convolution, and replaces the standard matrix multiplication step used to propagate information forward in standard fully-connected layers [29]. Crucially, this process reduces layer size, with the decrease dependant on kernel size and how it is moved across the input (determined by 'step' and 'padding' layer parameters). A common method of reducing size is via 'max pooling' convolutional layers, which replace regions of data with the maximum value within that region [41].

As such, CNNs perform well at spatially-critical tasks, such as computer vision and image analysis, due to this ability to efficiently distill information from a large input [29] [48]. CNNs are the dominant network type applied to acoustic monitoring because their visual strengths are applied to the analysis of time-frequency diagrams like spectrograms or MFCs.

Within passive acoustic monitoring research there tends to be two types of CNN that are focused upon; lightweight models that are fast and efficient, or more complex models that aim for a higher classification accuracy. Mac Aodha et al. trained two CNNs as part of their 'Bat Detective' pipeline, labelled 'fast' and 'full' respectively, which targeted these two niches [26]. The sequel pipeline, 'BatDetect 2', was cut to a single powerful model. This is potentially wasteful, as deploying lightweight models on-site can massively reduce the amount of audio information generated by monitoring systems. Gallacher et al. [12] applied this approach for their echobox project, where 17 call-detection boxes were placed throughout a park in London, and autonomously published their nightly classification results to a live website. However, as these devices only saved the audio of their identifications, it is difficult account for false negatives; nightly recordings cannot be analysed for calls the detectors missed. This is increasingly important for conservation, as rarer, endangered species are less likely to have sufficient data to train a successful classifier, rendering autonomous monitoring insufficient. Ultimately, both structures of network are important to the monitoring of animal species.

Chapter 3

Methods

This chapter begins with a deeper look at the mechanisms behind spectrogram generation, and defines the problem this project aims to tackle. The solution, a detection pipeline which implements the Goertzel algorithm, is proposed, followed by a more detailed look at the algorithm. The next subsection of the chapter is dedicated to discussing the components of the pipeline, focusing on the various deep learning and machine learning models it comprises. Finally, the structure of the pipeline as a whole is introduced and discussed in depth.

3.1 Problem definition; Spectrograms and Goertzel

3.1.1 Fourier Transforms and Spectrograms

3.1.1.1 Discrete Fourier Transform

The discrete Fourier transform (DFT) is a fundamental tool within signal processing [25]. Given a discrete, finite-length input sequence of samples x, where $x[n]$ refers to the n^{th} item in x and $|x| = N$, the discrete Fourier transform of x is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1 \tag{3.1}$$

This algorithm converts sequences of real-valued samples to sequences of complex-valued frequency. Alternatively, it is said to bring sequences into the 'frequency domain'. This operation is incredibly useful across many domains for a wide range of purposes. In the case of processing audio, a DFT converts a sound wave into its constituent sine-wave frequencies. This allows for the structural analysis of a sound.

An extremely important characteristic of the DFT is the discrete nature of the algorithm. The input sequence of samples must be finite, and are presumed to be some uniform distance apart from one another. For audio, this distance is the 'sample rate' of a recording, which determines how many values make up one second of audio. Similarly, the output sequence of frequencies is not continuous, and is determined by the number of input samples. This results in a 'binning' effect, where the frequencies in-between the

steps in the output sequence are grouped to the nearest sequence step. When discussing DFTs, a 'frequency bin' refers to one of these discrete steps in the sequence of output frequencies.

3.1.1.2 Spectrograms

Spectrograms portray DFT information as convenient, three dimensional functions. A spectrogram S of some given waveform x takes arguments of time and frequency simultaneously; given some time t and some frequency f, $S[t, f]$ is equal to the power-spectrum of f taken at t, as calculated by the discrete Fourier transform. Spectrograms can also be brought to resemble amplitude by converting the values to a logarithmic scale. Rather than calculating specific $S[t, f]$ values when needed, spectrograms must first be 'generated' as a whole from a given input sequence. As with individual DFTs, the values within a spectrogram are limited to a set number of time and frequency bins, determined by the input and terms of generation. The location of a single value within a spectrogram is a 'time-frequency' bin.

A spectrogram can be obtained by either analogue (sequential band pass filtering) or digital methods (the fast Fourier transform), though these do not always give the same results [3]. Although the analogue method predates the digital, generation by Fourier transform is vastly more commonplace due to the rise in digital technologies. Thus, all spectrograms discussed in this report are generated by methods incorportating the fast Fourier transform.

The fast Fourier transform (FFT) [9] is an alteration of the discrete Fourier transform which drastically reduces the computational complexity of the algorithm from $O(N^2)$ to $O(NlogN)$. The original implementation by Cooley and Tukey [9] is known as the radix-2 FFT, as it works optimally when $N = 2^t$ for some integer t. The optimisation is achieved by splitting the N-point input sequence x_m into two sequences, x_{2m} and x_{2m+1}. This process is repeated for both x_{2m} and x_{2m+1}, until the original sequence has been split into $t = log_2 N$ stages [31]. These residual sequences are then independently calculated as DFTs, as opposed a single DFT of one large N-point sequence. The breaking down of one sequence into many smaller sequences can also be known as 'windowing'. While alternative FFT methods and implementations exist, all have the same computational complexity.

In the case of spectrograms, a special type of fast Fourier transform is applied, known as the Short Time Fourier Transform (STFT). This method splits the input sequence N into many small overlapping sub-sequences, known as 'windows'. This process, similarly called 'windowing', is crucial for determining the variation in signal over time [24].

For all our spectrograms, we use Hanning windowing to partition the input audio into 1024 windows, with 75% overlap. For 20ms of audio, this results in each window being an $N = 20$ sequence. Then, we apply the 1 dimensional 'rfft' FFT method from the 'numpy' python package [33] to the set of all windows. The output of the complex series X is further computed as squared magnitude via $X = |X|^2$, in order to bring it to a spectrogram representation [24] [3]. Finally, all X are subsequently converted to 'amplitude' by converting values to a logarithmic scale.

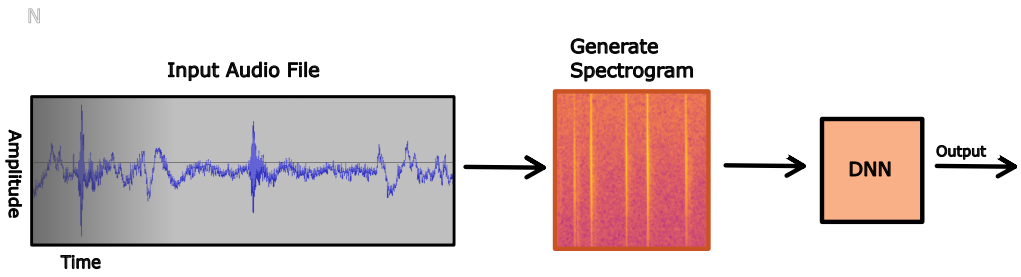

Figure 3.1: *A diagram depicting the structure of the neural network bat detection pipeline, as established by Mac Aodha et al. [26]. This pipeline architecture is entirely reliant on spectrograms for the analysis of bat calls, as it pre-generates them prior to applying its DNN. Reliance on unnecessary spectrogram generation can be deeply inefficient, especially on low-spec, in-the-field hardware. While the original pipeline utilised a DNN bat detector, a DNN bat species classifier could theoretically be applied in the same position, hence the generic 'DNN' label. This diagram excludes the spectrogram normalisation step, and does not portray the 'sliding' method of the DNN.*

3.1.2 The Problem

The issue with spectrogram generation arises when analysing audio data, particularly large volumes, for bio-acoustic phenomena. In the context of bats, though echolocation tends to occur in regular pulses [20], there is no way to know when an initial call may be detected. As observed by Vaughan et al. [47], the calls of British bat species are short, ranging from 70ms to just 0.3ms in duration. Thus, even in instances where portions of audio containing bat calls have been isolated (e.g: the datasets discussed in chapter 4), the individual calls are likely to only make up a small portion of the data. Thus, generating the entire recording as a spectrogram is highly likely to be wasteful; large portions of useless data will be processed, only to not contain calls, and be discarded. Furthermore, generating spectrograms on low-spec devices, such as those typically used in the field, is a large performance bottleneck [38]. These systems already have to There is therefore a priority to only generate spectrograms when they are needed. Typical deep-learning based pipelines, such as the one introduced by Mac Aodha et al. [26], are unable to tackle this problem as they are fully reliant on spectrograms for their analysis; Figure 3.1 gives an example of the structure of such a pipeline.

Saving on spectrogram generation can be achieved through other means of detection, such as in the form of a simple thresholding method based on amplitude or some other metric. However, such methods tend to be unreliable, as calls may be faint or partially obscured by other acoustic phenomena [43]. Ideally, we would be able to check for distinct features of bat calls without having to generate a full spectrogram. Here, we introduce the Goertzel Algorithm.

3.1.3 The Goertzel Algorithm

The Goertzel algorithm, created by Goertzel [14], is an alternative signal processing algorithm to a discrete fourier transform, which excels at calculating the densities of single frequencies. Formally, the Goertzel algorithm acts to calculate a single frequency bin k of an N-point discrete Fourier transform over input sequence x, defined as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}kn} \tag{3.2}$$

where $X[k]$ is the frequency coefficient at time $n = N$ [25].

This is achieved through a process which resembles an infinite impulse response (IIR) filter. Given an initial input, such a filter never returns to an exact output of zero, as it partially relies on feedback (previous output). The Goertzel algorithm is specifically a two step process, defined by $W(n)$ followed by $Y(n)$ for some $n \in N$ such that $n \geq 0$:

$$W(n) = 2cos\left(\frac{2\pi m}{N}\right) W(n-1) - W(n-2) + x(n) \tag{3.3}$$

$$Y(n) = W(n) - e^{-i\frac{2\pi m}{N}} W(n-1) \tag{3.4}$$

where $w(n-1), w(n-2) = 0$ for $n \geq 2$, and where m is an integer in the range $[0, N)$ [25]. This keeps the parameter $\frac{2\pi m}{N}$ within the bounds $0 \leq \frac{2\pi m}{N} \leq 2\pi$, which is necessary to keep the algorithm equivalent to the discrete Fourier transform. Since m also corresponds to the target frequency, this has the additional effect of increasing the resolution of available frequencies as input size increases.

Prince et al. [38] report that the Goertzel algorithm's computational complexity is $O(N)$, which is less complex than the complexity of the FFT at $O(NlogN)$. However, this is only the case for single tone detection. For the general case, Lyons [25] states that the Goertzel algorithm is more efficient (leads to fewer multiplication operations) than an equivalent FFT when the following inequality holds:

$$M < \log_2 N$$

where M is the number of target frequencies, and N is the number of input samples. Such an inequality is crucial to keep in mind when designing efficient detection methods.

In further sections, when discussing to the output of the Goertzel algorithm, we refer specifically to the 'power spectrum' term. Equivalent to $|X(m)|^2$, or the raw magnitude in a typical spectrogram, this term is typically calculated using $W(n)$ values, skipping over additional complex calculations and further increasing the algorithm's efficiency:

$$|X(m)|^2 = W(n-1)^2 + W(n-2)^2 - W(n-1)W(n-2)2cos\left(\frac{2\pi m}{N}\right) \tag{3.5}$$

where $n = N$.

3.1.4 Solution: The Goertzel Pipeline

This project proposes a modified version of the bat detection pipeline initially developed by Mac Aodha et al. [26]. Our pipeline utilises the Goertzel algorithm in the detection of calls, in an effort to reduce the amount of unnecessary spectrogram generation. We use a classical machine learning method, a decision tree, to apply the Goertzel algotrithm. The pipeline is also adapted for the species classification task, bringing it in line with more recent work [27]. To distinguish between the original pipeline and our modified version, we refer to the modified pipeline as the 'Goertzel pipeline' throughout this report.

The Goertzel pipeline was designed to simulate real-time analysis of an incoming audio signal. Given an audio file input, it iteratively reads a small sequence of audio from the file, 1ms to 10ms in length depending on the decision tree used, and evaluates whether any bat calls occur within the input time frame. If a call is detected by the decision tree, then shortly after a spectrogram of the last 20ms of audio is generated. This is then optionally evaluated by the DNN detector to ensure a call is present, before the species of the bat is evaluated by the DNN classifier, and the results are logged. The pipeline then moves onto the next non-overlapping audio sequence. Thus, over the course of evaluation, the pipeline gradually 'slides' over the audio file.

Figure 3.2 gives an overview of the structure of the Goertzel Pipeline. Note that while this diagram shows a two step detection process, the decision tree detector or the DNN detector can both optionally be used as the sole method of detection. Section 5.1 explores which detection methods are the most efficient and the best performing when applied within the Goertzel pipeline.

3.2 Pipeline Components; Model Design, Training, and Evaluation

Before delving into the Goertzel pipeline as a whole, we should initially consider its most important base components. These predominantly take the form of Deep Learning Networks, with decision trees also playing a key role. This section covers each component in a vacuum; its design, the principles behind how it works, what role it performs within the Goertzel pipeline, and why it was chosen to fulfil such a role. Section 3.3 then covers the pipeline as a whole in detail.

3.2.1 Neural Networks

We use two separate neural networks for the task of call classification. Both networks take a small spectrogram as input. Specifically, each input spectrogram is generated from 20ms of audio, meaning each one consists of 20 time bins by 513 frequency bins. Chapter 4 discusses these spectrogram 'slices' in more detail. Both networks make use of convolution to distill information about the input spectrograms. It is thus important to note that while we refer to these networks as DNNs throughout the paper, they are CNNs in particular.

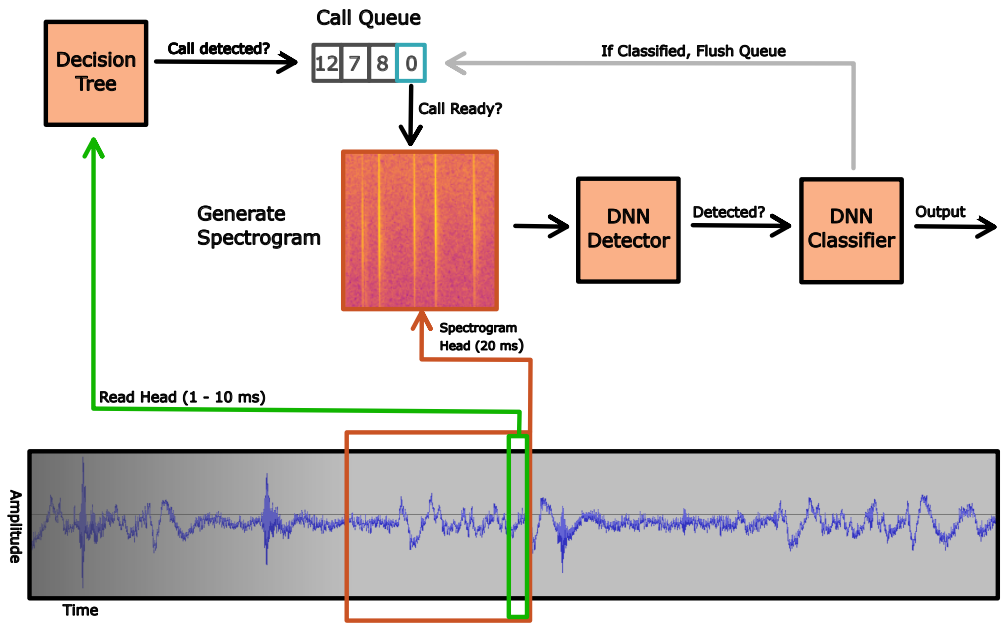

Figure 3.2: *A diagram depicting the structure of the complete Goertzel pipeline, with both the decision tree and the Detector DNN enabled. The decision tree detector or DNN detector may optionally be used as the sole method of detection. Note that the spectrogram depicted is purely a visualisation, and contains much more data than the true 20ms spectrograms that are generated by the pipeline. Likewise, the depicted bat audio has been severely amplified in order to make its structure more pronounced.*

The first network is a lightweight binary classifier, and is used to analyse whether an input contains a bat call or not. We call this network the 'detector' network throughout the paper. The other network, which is more robust, evaluates inputs that have been verified by the detector, and reports the most likely species that produced the input call; we refer to this network as the 'classifier' network. Both networks output the most probable class as an integer value, along with a confidence value for the prediction. It is important to note that neither network attempts to locate the position of the call within the input spectrogram, such as by drawing bounding boxes around acoustic phenomena. Instead, they tackle a 1 dimensional search for call presence and call species, respectively, and output a 1 dimensional class label.

The classification of species based on call structure is a difficult task, and this necessitates a more complex network which is slower to evaluate. It would thus be inefficient to run this network over an entire audio clip for the purposed of finding and classifying the calls of bats, considering calls take up the minority of each audio file. Similarly, the lightweight detector DNN is too simple in structure to classify the originating species of a call with much accuracy. Therefore, we use a lightweight detector network to detect the presence of bat calls within a spectrogram efficiently, and only apply the more complex classifier if it is required. The separation of DNN detector and DNN classifier also enabled the swapping to another detection method, such as a Goertzel

decision tree, in a modular fashion.

In general, the deep neural networks were designed for this project with two key principles in mind. The first was simplicity, which in turn allowed for rapid training and iterative development. The second, closely related to simplicity, was evaluation speed. As such, both networks are only a handful of layers deep. Training and evaluating the models consistently took under ten minutes using an Apple M1 laptop without GPU support. All deep learning methods were implemented in python, using the 'pytorch' and 'pytorch_vision' libraries [36].

3.2.1.1 Detector Network

As stated above, the detector network takes a 513 x 20 spectrogram as input. The network consists of two convolution layers followed by three fully connected layers. Both convolution layers utilise 5x5 filters and each is followed by a 2x2 max pooling layer, with the fully connected layers as the final layers of the network. All layers, bar the final one, make use of the ReLU non-linearity [13].

Larger filters were chosen in order to reduce the size of the input rapidly, within the span of just a few layers. Replacing larger filters for multiple smaller ones is suggested by Sze et al. [45] to help reduce computation complexity. However, for this model, such a practice resulted in issues in training due to vanishing gradients. Batch normalisation or residual connections may have helped to remedy this issue, but it was deemed more appropriate to simply keep the larger filters in use.

The detector was trained using Cross Entropy loss, and stochastic gradient descent with a learning rate of 0.0001 and momentum of 0.9. The model was trained for 10 epochs, with batch size 16. A greater number of epochs were experimented with, but did not lead to an improved performance on the test set.

3.2.1.2 Classifier Network

The classifier network also takes a takes a 513 x 20 spectrogram as input. This network is a similarly lightweight design, consisting of 3 convolution layers, three max pooling layers, and a single fully connected layer. In this network, the convolution layers each use smaller 2x2 filters. Additionally, each convolution layer utilises 18 channels, to better distinguish between the 18 possible classes. Similar to the detector, each convolution layer is followed by a 2x2 max pooling layer, with the fully connected layer being applied at the end. Each layer utilises the ReLU non-linearity once more, except for the final fully connected layer, which uses softmax.

This network also uses batch normalisation, which was introduced due to initial training problems. Batch normalisation was introduced by Ioffe and Szegedy [19], and serves to normalise the inputs between layers in a neural network, which aids the training process. In the classifier model, batch normalisation is applied between each layer, excluding the transition between convolution and max pooling layers.

The classifier network was trained using Cross Entropy loss, and the ADAM optimiser [23] with a learning rate of 0.001. Like the detector, the model was trained for 10

epochs, but with a reduced batch size of 8.

3.2.2 Decision Trees

Since the Goertzel algorithm outputs a single Fourier domain value per target frequency, an approach differing from a convolutional neural network was taken when incorporating it into the project. While a neural network would have still been a viable option for an alternative detector, doing so would have introduced further intensive computation into the efficient pipeline via the matrix operations that neural networks rely on. Instead, a simpler structure was decided upon; singular decision trees. Decision trees have added the benefit of hierarchically processing the data while not being black box methods.

A decision tree T consists of a set of sub trees and a branching condition that determines which branch is taken [40]. For the purposes of this project, all decision trees are binary. If the branch condition is met, then the left subtree t_l is selected, and otherwise the right branch is taken to subtree t_r. A branch condition is a boolean formula; for the trees that use the Goertzel algorithm, this is of the form $y \leq \theta$, where y is the power-spectrum output of some input x and selected frequency f, and θ is some threshold value. Upon reaching a leaf node, whether the input contains a call or not in returned. Subtrees are constructed through training, where values f and θ are gradually optimised to provide the best performance for all subtrees $t_s \in T$.

The decision trees were implemented using the scikit-learn python library [37], with the included 'DecisionTreeClassifier' class. The trees were varied by two key factors. One was the maximum possible depth for each branch, d; this corresponds to a maximum of d frequencies being evaluated by the Goertzel algorithm over the course of a single tree traversal.

The other parameter was the size of input, g_l. This produces differentiation between trees due to a quirk of the Goertzel algorithm - the larger the size of the input, the greater the number of select-able frequencies available. Thus, depending on the frequencies available, branches may be made on different frequencies with different threshold values, leading to trees of radically different structure.

It is imperative to note that the value of g_l corresponds to lengths of time within the audio clip. The sample rate of each audio clip s dictates the ratio of values within the file to seconds; each sample lasts for $\frac{1}{s}$ seconds. We try to define all g_l values, as well as general as sequences of audio, in form of x milliseconds (ms) during this section. This corresponds to input sequences of $\frac{s}{1000x}$ single values. However, for situations involving the Fourier transform, where size of the input sequence is important, we assume $s = 300,000$, meaning 1ms of audio is 300 samples long.

Values $d \in [5,10]$ and $g_l \in [1,2,5,10]$ were used, resulting in eight total decision trees trained overall. Throughout the report, trees with $d = 5$ will be referred to as 'shallow' trees, whereas $d = 10$ trees shall be called 'deep' trees.
However, not all of these possible trees are more efficient than an equivalent FFT. If we let $M = 10$ and $N = 1 \times 300$, we find that the inequality from Lyons [25] does not hold for the deep 1ms tree: $10 > log_2\ 300$. We find this case arises for the 2ms tree as well, where $N = 2 \times 300$. While we still evaluate the performance of these trees in the

experiments section, since they are less efficient than spectrogram generation, they are automatically not eligible to be used in the pipeline.

While decision trees were ultimately selected to be used in conjunction with the Goertzel algorithm, they were by no means the only option available. Another possibility could have been a 'boosting algorithm', which are a grouping of weak classifiers. Instead of the result being dependant on branching as in a tree, each weak classifier is applied in order, with the majority vote used as the outcome [10]. Comparison of the models that implement the Goertzel algorithm within the pipeline is a future research task.

3.2.3 Baselines - Random Forests

Two random forests were also trained during the initial explorations of the data, prior to the creation of the deep neural networks. As with the DNNs, one forest was trained to detect the calls, and another was trained to classify them. In this report, random forests are used as baselines of performance to compare against deep learning networks. All forests were created, trained, and evaluated using the python library scikit-learn [37].

The forests used in initial exploration were created with 20 tree estimators. This decision was rather arbitrary; after brief experimentation with number of estimators, a size of 20 trees was determined to provide a good performance comparative to the duration of training. However, due to the small number of trees, evaluation of these forests resulted in extremely quantized precision recall curves. As such, forests with larger numbers of estimators were trained. These were the random forests used as baselines in section 5.2.

The baseline detector random forest consists of 128 trees in total, and was trained on the same data as the detector DNN. Several other random forests were wrapped in an sci-kit learn 'OneVRest' classifier for the multi-class species task. While the random forest classifier we use supports multi-class tasks natively, we experienced difficulty in successfully training them for this purpose. The OneVRest classifier resolved this issue by training a binary random forest for each species class in the dataset. Given the 18 possible classes, forests of a reduced 32 estimators were used in order to reduce training time. This resulted in 576 trees across the whole OneVRest classifier, or a classifier forest 4.5 times larger than the detector forest.

The training and test data of the was also altered for the random forest classifier. For the same dataset used in the DNN, each integer class label was replaced with a one dimensional array of 18 entries in total. A single entry in each array was set to 1, while the rest were set to 0. The index of the positive value was used to determine the individual class.

3.3 Pipeline Design and Implementation

With the most complex components covered, we now discuss the Goertzel pipeline in detail. In section 3.3.1, we cover the analysis of an audio clip within the pipeline in a step by step process. We then take a look at how the performance of the pipeline is evaluated in section 3.3.2.

3.3.1 Step-by-step Explanation

3.3.1.1 Initialisation

Before beginning the step-by-step explanation, the setup of the pipeline will need to be covered. This is because the pipeline is highly flexible and can behave quite differently depending on these initial values.

As briefly discussed in section 3.1.4, there are many ways to run and evaluate this pipeline. The most important is that detection methods can be enabled or disabled at will; the pipeline can operate with a Goertzel-based decision tree detector, a DNN detector, both detectors together (tree first, DNN second), or none at all. In the case where no detector is used, a simple random method is used to generate a subsections of the call as spectrograms, which are immediately sent to the DNN classifier.

Over this explanation, we shall make use of both detection methods: a decision tree evaluation, followed by DNN detection. We shall use a Goertzel-based decision tree that takes 1ms of audio as input; let $g_l = 1$ refer to this value. The depth of the tree g_d is irrelevant. The 'read head' of the pipeline h is always the same size as decision tree input: $h = g_l$. The head shall iterate over the call in steps of 1ms. Let $j = 1$ reflect this step size. Let ρ refer to the current time within the audio file that the read head is point at; for $g_l > 1$, this ρ refers to the maximum time within the time range of the read head, or the 'right edge' of the reader. No positive thresholding will be applied; each positive detection by the decision tree shall be entered into the spectrogram queue. Other variables present throughout the steps shall be explained as they are encountered.

3.3.1.2 Loading Audio

The pipeline can try to predict the species of bat present within audio at either the file or folder level. For testing purposes, folder level is preferred, as the pipeline will automatically load the associated annotation file for a recognised folder of bat recordings. All recordings in the selected folder are subsequently iterated through and evaluated. This allows for comparison of the pipeline's detector and classifier to the expertly annotated calls within each audio clip - see chapter 5 for more details.

3.3.1.3 Goertzel Decision Tree

The 'read head' of the pipeline, at its current position within the given audio file, takes a g_l ms section of audio (in our case 1ms worth) and gives it as input to the decision tree. The tree uses the Goertzel algortihm to evaluate whether the small g_l ms slice of audio contains a bat or not.

Each node along the tree evaluates whether the power spectrum value at single target frequency f is less than or equal to a given threshold t, with both f and t dependant on training. If $f \leq t$, the left branch is taken, and the right branch otherwise. Each leaf node contains set probabilities of the input containing a call or not. For consistency, the maximal of these probabilities is always the output class.

If a call is predicted (ie; the positive class is output by the tree), the input time range is enqueued for spectrogram generation. Either case progresses on to queue evaluation.

3.3.1.4 Spectrogram Generation Queue

The spectrogram generation queue is a variable length queue that stores the time ranges of possible bat calls until they are generated as a spectrogram. The reason this queue exists is due to the difference in input size between the decision tree and the DNNs; $g_l < 20$ ms for all trained trees. Thus, we need to wait some φ ms for the detected call to 'come into view' for the 20ms range we generate as a spectrogram. In most instances of the pipeline, $\varphi = 15$ ms of audio. This duration is typically sufficient to bring a detected call to a position within the spectrogram such that it can be clearly identified by the classifier.

Each entry in the queue is a GenCount object, which stores an ordered set of counters $C = [c_1, ..., c_n]$, where each $c \in C$ is initialised to φ ms. The object also stores a of a set of times $T = [t_1, ..., t_{ng_l}]$ where a call was detected. Let each GenCount be written as $G_i = [C_i, T_i]$, and the queue $Q = [G_1, ..., G_m]$ If there a significant amount of time has passed with no positive call detections, then a new detection will be enqueued as a new GenCount object. Otherwise, the time range for the most recent detection will be appended to the most recent GenCount, G_m, along with a new φ ms counter. This is done to group several sequential call detections as single calls, mitigating the risk that individual calls are evaluated multiple times. Such a risk is especially potent for the 1ms decision tree; given a 10ms call, the tree would report a 1ms detection 10 times in a row, potentially leading to 10 spectrogram generations if this system were not in place.

At each read progression step, all counters are decremented by the step length of the read head j. In our example, this is just 1 ms. When the first counter $c_1 \in C_1$ within G_1 is ≤ 0, we remove all $c \in C_1$ where $c \leq 0$ from C_1. The time range $[t_1, ..., t_{g_l}] \subseteq T_1$ is also removed from T_1, though $[t_1, ..., t_{g_l}] \cup T_1$ is used for performance evaluation. A spectrogram S is then generated across the time range $\rho - 20, \rho$ ms, and we move on to the DNN Detector step. If $c_1 > 0$, then no attempt at call classification is made, and we move on to the read progression step.

3.3.1.5 DNN Detector

Once the spectrogram S is generated, the DNN detector will evaluate whether there is a call at that position. For the pipeline with both detectors enabled, this DNN verification allows for an increase in the pipeline's overall precision. However, since the DNN is always bounded by the decision tree's evaluation, detection of calls unseen by the tree is not possible. In other words, the recall of the double-detector pipeline is entirely dependant on the decision tree.

The DNN Detector can optionally be 'offset' from the DNN classifier by a factor of ω ms. This has the effect of delaying the DNN detector's position in the audio by ω ms, meaning the spectrograms generated by both methods will not perfectly overlap for $\omega > 0$. This significantly increases overall classification performance, at the cost of increasing the average spectrogram generation per call by ω temporal bins. In our experiments, we use values within the range $0 \leq \omega \leq 7$ for various pipeline configurations.

The DNN returns a binary label reflecting whether a call is present or not, as well as a

confidence value x_d. To count as a valid call, this confidence value must surpass some threshold k_d. Typically, $k_d = 0.85$. If the DNN detector verifies that a call is present, and $x_d \geq k_d$, then S passed on to the DNN classifier. Otherwise, S is discarded, and we skip ahead to read progression.

3.3.1.6 DNN Classifier

If the call has been detected, S is then evaluated by the DNN classifier, which will report the most likely originating species for the call within S along with the confidence of this decision x_c. For some small threshold value k_c, if $x_c \geq k_c$, then the output of the classifier is logged, and we progress to the Queue Flush step. Typically, $k_c = 0.075$, as the maximal confidence for any one class is observed to only be around 0.135. This is due to the fact that most (if not all) species are reported with some small probability by the DNN classifier.

If $x_c < k_c$, the output is instead dependant on G_1. If there exists some $c \in C_1$ such that $c > 0$, then it's possible the next spectrogram S_{+1} will result in an accurate classification. In such a case, we ignore the output of the classifier and move onto the read progession step. If no such c exists, then we know there are no more opportunities to confidently identify the species, and the call is instead assigned as the generic bat class. Now that the call has been assigned a generic label, we can progress to the Queue Flush step.

As Barclay [5] suggests, when determination between species is inaccurate, it is preferable to determine between groups of species. Thus, the classifier can optionally report output at the genus level instead of the species level. This is achieved simply by only taking note of the first word of each output label (which, as standard, are two words: genus, followed by species). It is important to note that while we only report the genus in this mode, the model still predicts at the species level. In other words, the model always predicts the most likely of 18 possible species labels, rather than a reduced set of 5 or 6 possible genera.

3.3.1.7 Queue Flush

If a call has been classified, G_1 is removed from the queue entirely, and all other queue entries $Q \setminus G_1$ have their indexes updated such that the new first member of Q is renamed G_1, and the last entry in the queue is G_{m-1}. This also occurs if $C_1 = \emptyset$ after the classification stage.

This is done to ensure we do not classify calls more than once; if a call object G_i remains in the queue after classification, and $C_i \neq \emptyset$, then we risk generating further spectrograms and potential classifications of this grouping in future cycles. It should be obvious that classifying one call as many calls would be detrimental to the pipeline's overall precision.

3.3.1.8 Read Progression

The read progression step is simple. We move the read head ρ (and by association, the spectrogram head) forward across the audio clip via incrementing it by j ms; $\rho = \rho + j$.

We then decrement all spectrogram generation counters $c_i \in C_i \in G_i$ by j, for all $G_i \in Q$. Finally, if there has been y ms since the last positive detection (not classification), a 'gap' in calls is established. This condition leads to the creation of a new $G \in Q$.

If the end of the loaded audio file has been reached, then we progress onto the performance evaluation step. Otherwise, we jump back to the goertzel decision tree detector step, in a new cycle of the pipeline.

3.3.2 Performance Evaluation

The performance of the pipeline is evaluated as it iterates over each file in the current dataset. Prior to initiating the pipeline, all annotations corresponding to the current recording are cached. An annotation is an expertly created store of bat-call data, including the start time, end time, and species, for a single call in the dataset. While annotations are grouped by audio files, it is important to reiterate that each annotation corresponds to a singular unique bat call; see section 4.1.1 for further information.

These annotations $A = [a_1, ..., a_j]$ are sorted by time, with a_1 occurring first in the file. Let $a_{latest} = a_1$. Additionally, an array A_{prev} is established, to store previous true-positive detections; a similar array is created for true-positive species classifications. These arrays are only used for evaluation.

Each time the detector method makes a detection, we either iterate through the current detection times in the spectrogram queue $t_i \in [t1, ..., t_{g_l}]$, or all times within the spectrogram itself $t_i \in [\rho - 20, ..., \rho]$, depending on whether a tree or neural network is used. We then iterate over all loaded annotations $a \in A$, evaluating time ranges $t_{start}, t_{end} \in a$ and try to find some annotation a such that $t_{start} \leq t_i < t_{end}$. If such an a exists for some t_i, we evaluate the detection as being a true positive, add a to A_{prev}, and move on to evaluating the classification. Alternatively, if no such t_i exists, we evaluate the detection as a false positive. A similar method is applied for the classifier, where true positives are assigned to the target class, and false positives are assigned to the incorrectly identified species.

False negatives are evaluated slightly differently. At the end of each time step, if no evaluation was made, the time $\rho - \varphi$ (aka; the time when detections leave the spectrogram queue) is evaluated, and compared to $t_{end} \in a_{latest}$. If $\rho - \varphi > t_{end}$, and $a \notin A_{prev}$, we know the annotation has been passed without detection. As such, the detection is declared a false negative. As before, this process occurs for both the detector and the classifier, meaning an undetected call hurts both the detector and the classifier's overall performance. This allows us to use the precision and recall of the classifier as an overall pipeline performance metric when evaluating different detection methods.

Chapter 4

Dataset

Having described the problem and pipeline in detail, we now turn to focus on the data that our methods were applied to. This section outlines the datasets used for implementation and experimentation, where they were sourced, and how they were modified over the course of the project.

4.1 The Raw Dataset

The data used for this project was sourced from Mac Aodha et al. [27], and consists of eight sub-datasets from a wide range of sources. This includes a subset of the Echobank [8] dataset, as well as recordings from bat conservation groups such as the Bat Conservation Trust and Bat Conservation Ireland. Each dataset consists of many audio clips containing bat echolocation calls, along with a singular JSON file containing the annotation data for each file.

4.1.1 Dataset Contents

Annotations label the bounds for individual calls, and are split into subsets, grouped by the audio file they annotate. Each annotation includes the start and end times of the related call, as well as the species that made it. Extra information is also present, such as the range of each calls' frequencies, but this is irrelevant for the Goertzel pipeline; such information is interpreted from the generated spectrograms by the DNN classifier. In total, there are 32929 annotated calls between the eight datasets.

Within each dataset, the majority of audio files share a common length. Almost all datasets consist of one second clips, with two exceptions: one dataset consists of half-second clips, and another dataset at two seconds per audio file. All recordings have the same sample rate of 300,000 samples per second. Instead of identifying the species via recording calls in the wild, the audio in these datasets was recorded from captured bats that had had their species physically verified. The individuals were recorded in a variety of different conditions, such as within the hands of the scientists, within nets or flight-cages, to various forms of free-flight [8].

4.1.2 Species and Species Distribution

In total, 17 unique bat species are present across all datasets, in addition to a generic 'Bat' class for species that were unclear, or otherwise not included within the data. This corresponds to the 17 bat species that reside in the UK. The distribution of species between datasets was highly variable, with Echobank and the Bat Conservation Trust being the only datasets to contain all 18 class labels for the bats. Other datasets, such as the Bat Conservation Ireland dataset, as well as some of the British Bat Calls subsets, were highly biased towards only a handful of species. The number of total calls per species can be seen in Table A.1 in Appendix A.

Most audio files contain only a single species of bat; the most prominent species per file is viewable for each file-level grouping of annotations. Excluding the generic 'Bat' class, only 3.89% of all annotations differ from the most prominent species for a given file, with 3.09% being members of a completely different genus.

4.2 Dataset Processing

4.2.1 Spectrogram Data

4.2.1.1 Spectrogram Generation and Slicing

Spectrograms of the calls were generated by sequentially iterating through all annotation files. Each audio file associated to a group of annotations was generated as a spectrogram via the short-time Fourier transform. Some slight post-processing was done to convert the square-magnitude spectrogram values to a logarithmic scale of amplitude. All spectrograms generated in this way consisted of 513 frequency bins, corresponding to an evenly-spaced range of 513 frequencies between 0Hz and 150kHz, with length dependant on the audio file's duration. 1 second files resulted in spectrograms 1168 temporal bins wide; 0.5 second and 2 second clips resulted in spectrograms roughly half or double the width respectively.

After generation, spectrograms were then sliced into small strips containing the annotated calls. Spectrogram slices clearly portray structure of an individual call; Figure 4.1 contains illustrative example slices for 9 of the 18 classes in the dataset. Figure A.1, sourced from Mac Aodha et al. [27], gives a more detailed look at the varied structure of the bat species present across the datasets.

The height of each strip remains the same as in the full spectrograms, as 513 frequency bins, with the width being set at 20 temporal bins. A width of 20 bins was chosen as a compromise between the shortest and longest bat calls. Across all annotation files, the mean width of bat calls is underneath 20 bins after being generated as a spectrogram. Additionally, a shorter width lowered the risk of any overlap between slices. Calls wider than a slice were simply truncated to the first 20 bins. While the 'head' of each call still contains sizable information for identification, this decision nevertheless introduces some bias towards the start of calls into the detection and classification models.

This bias is compounded by the limitations of the spectrogram generation process. While the annotated calls can give start and end times at the precision of microseconds,

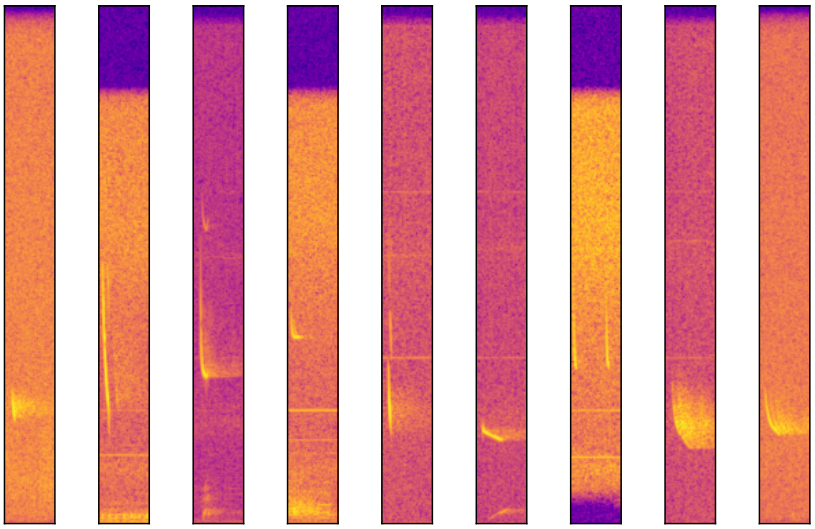

Figure 4.1: *Spectrograms of nine different bat species generated from the dataset. Each spectrogram is 50ms of audio long (for improved visualisation), and goes from 0 to 150kHz in frequency. The distinct, angled streaks in each spectrogram are the calls of a bat. From left to right, the species are 'Barbastellus barbastellus', 'Myotis daubentonii', 'Pipistrellus pipistrellus', 'Pipistrellus pygmaeus', 'Plecotus austriacus', 'Nyctalus leisleri', 'Pipistrellus nathusii', 'Nyctalus noctula', and 'Eptesicus serotinus'.*

the spectrogram of an audio clip is only a few thousand temporal bins in length at most. Consequently, slicing the spectrogram at exactly the start of an annotation can result in partial calls. To remedy this, an 'offset' value ω is used, such that the slice begins ω bins before the call start time. Such a practice granted increased call visibility within the slices, and lead to increased model performance when compared with models trained and tested on data with no offset. However, in ensuring the start of each call is captured correctly, we increase the chance that the tails of longer calls are cut off, further increasing bias.

Some calls occur too close to the beginning or the end of the sound files, such that the standard 20 temporal bin slice crosses the bounds of the spectrogram array and cannot generate. Such calls were unfortunately dropped from the dataset. Such occurrences were rare; only 138 annotations resulted in dropped calls across all datasets, or 0.419% of all labelled calls. Individually, some datasets had higher dropout rates than others, ranging from less than 0.3% to 1.5%.

4.2.1.2 Detector Network Data

Slices used in the detector network follow a binary labelling system; either a slice contains a bat call, or it does not. So far, the generated slices always contain calls. Thus, the detector dataset required additional slices to be generated where bats are not present.

After an audio clip had all of its annotations generated as slices, the spaces between each annotation are used to create empty slices. The exact position between the calls was randomly sampled. We attempted to generate a single empty slice between each pair of calls, resulting in a 1:1 ratio of empty slices to call-containing slices. However, an essential condition of generating these empty slices was that the gap between calls was sufficiently wide enough for a 20ms slice. Otherwise, the empty slices risked overlapping into the following call. This condition significantly reduced the total number of possible generations, bringing the total down to 25739 empty slices. This increased the size of the detector dataset by 1.782 times, to 58668 slices total.

Finally, the slices were split by the original dataset they belonged to, creating eight output datasets of varying sizes. Four of these datasets were used to train the detector network, with another for validation and two more for network testing. One dataset was completely set aside to be used when testing the pipeline.

4.2.1.3 Classification Network Data

There are many more labels within the classification dataset than the detection one. Each call is assigned a number corresponding to its originating species, resulting in 18 different labels in total. Unlike in the detection dataset, no further slices were generated; we assume that if a slice reaches the classifier, a bat call is present within it.

As with the detector datasets, each annotation file was generated and saved as a separate dataset of call slices. However, additional datasets were also created in attempts to better train the classifier.

One alternate group of datasets was filtered for the generic 'Bat' class, to check if the DNN classifier saw increased accuracy. Unsurprisingly, this resulted in increased dropout across all dataset files; some annotation files contained very few instances of this class, whereas others were over 10% generic bats in total. Some of these generic-less datasets were used to train a new classifier, which was validated on data with the generic bats reintroduced. No performance difference was measured between classification models that were trained on the generic-less data compared to the original datasets.

One problem with the per-annotation datasets was the wild difference in species distribution between them, as previously mentioned. The effect of this imbalance was much more pronounced in the classifier than the detector. The species present in the test datasets were rare in the training data, and vice versa, leading to poorly performing models. To alleviate this problem, another alternate dataset was created that combined 6 of the 8 datasets and output three specific train, validation, and test files. This aggregate set consisted of 28427 slices, and was distributed between train, validation, and test files with a 90% : 5% : 5% split. This greater collection of bat calls with a more even species distribution made for much better training data for the classifier. One of the two remaining datasets were used as extra test data, which was also its role in the detector network. The other dataset was reserved for use in pipeline evaluation.

4.2.1.4 Final steps

Some final pre-processing was carried out across all spectrogram datasets by transforming the numpy array representations of slices into a form that more closely represents pytorch tensors. This allowed for quick loading into custom dataloaders for network training and testing.

Similarly, all datasets were saved alongside with additional information regarding the mean amplitude value across all slices, as well as the standard deviation of the collection. These factors were used to normalise each slice before using it in network testing or training.

4.2.2 Goertzel Data

Several more datasets were generated with the Goertzel algorithm for the purpose of training decision trees. Each Goertzel dataset used every audio file procured by the Bat Conservation Trust, as this was the smaller of the two datasets that contained every species of bat. Four datasets were generated in total, varying by the size of input: 1ms, 2ms, 5ms, and 10ms slices of audio were used. To keep the description of the generation process general, let $n \in [1, 2, 5, 10]$ refer to the variable input size.

Each audio clip was loaded in order of appearance within the BCT's dataset annotation file. The data was then generated by taking $300n$ sample sequences from the loaded audio. Since the sample rate for each recording is 300,000 Hz, $300n$ samples is always equivalent to n milliseconds. These sequences did not overlap with one-another.

The n ms sequences were then used as input to the Goertzel algorithm, with a frequency range of 0 to 150kHz - the same range as in the spectrograms. The result was $150n$ power-spectrum values, taken at intervals of $\frac{1}{n}$kHz. Recall that the number of frequency bins present in the Goertzel algorithm is dependent on the length of input; sampling longer periods of audio resulted in a higher precision output.

Each slice was assigned a binary label as to whether or not it contained a bat call. This was done by simply checking if the sampled time period was within the bounds of any call annotation for the given audio clip. Each slice was designated as containing a call if it was contained within an annotation, and as empty otherwise. This method introduced some bias against larger inputs, as it only took a small overlap with an annotation to label a slice as non-empty. This was much more likely to result in mostly empty slices being positively labelled for longer input sequences than for smaller ones.

Given that the variable-length slices were all generated on the same data, each dataset contains roughly the same amount of information. However, some slices are much smaller than others; the 1ms dataset is 200359 slices big whereas the 5ms one is 39912 slices long, and the 10ms set contains 19855 in total. Note that the 1ms dataset is more than ten times as big as the 10ms one; this is due to a small percentage of audio files within the Bat Conservation Trust dataset being less than one second in length. As such, towards the ends of audio clips, situations arise where a longer sample period cannot be taken, but several smaller ones can, leading to disparity between database sizes.

Chapter 5

Experiments

This chapter details the evaluation of the methods discussed in chapter 3. We start with evaluating the pipeline by calculating the percentage of spectrograms generated for various configurations, and comparing performance to random baselines with equivalent spectrogram generation amounts. We then cover the evaluation of specific pipeline components, which influenced the tested configurations.

5.1 Pipeline Evaluation

All experiments in this section were run using different configurations of the main pipeline, using the same test dataset for performance evaluation.

5.1.1 Spectrogram Generation Evaluation

As spectrogram generation was the most expensive part of the Mac Aodha et al. [26] pipeline, the proportion of spectrograms generated by the Goertzel Pipeline serves as a good indicator of performance efficiency. In this experiment, we compare spectrogram generation proportion to detector F_1 score (a common metric for general network performance) for five configurations of the pipeline.

We evaluate the amount of audio generated as spectrograms on a file-by-file basis, by calculating the percentage of generation across each file. All audio files in the test set were 1 second in length, corresponding to 1000 total possible DFT bins. Therefore, each 1ms sequence of audio generated as a spectrogram contributes 0.1% to the reported generation percentage. Since spectrograms are generated in response to a predicted call when using decision trees, we normalise the amount of generation by the number of calls present in each audio file.

Two decision trees were tested; the shallow 1ms tree, and the deep 5ms tree. The reasoning for why these two trees were selected is presented in section 5.2.1.1. Each tree gives unique results when combined with the DNN detector, resulting in five configurations total. The DNN detector uses a DNN detector offset of 7, as this value was found to optimise classifier performance. However, this offset increases the effective

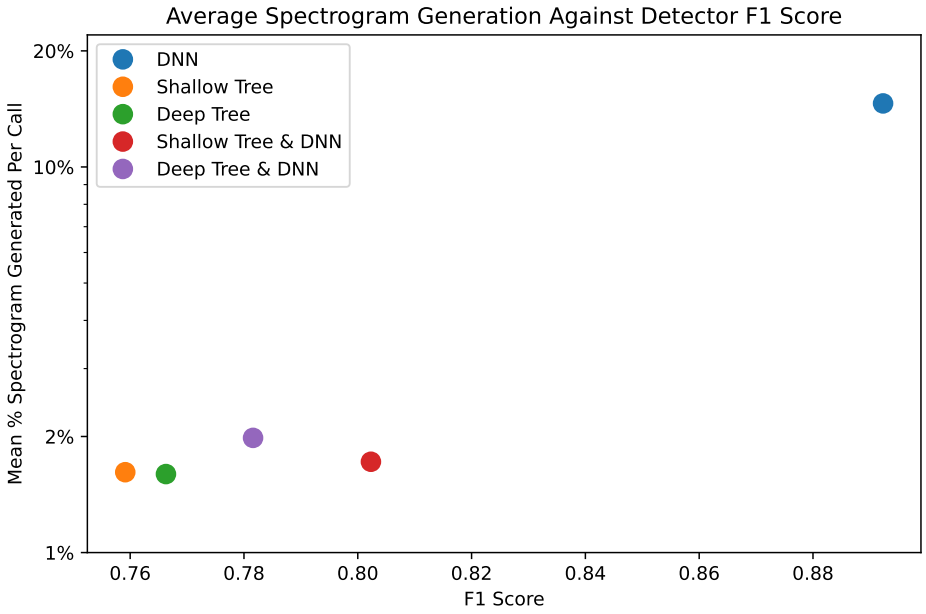

Figure 5.1: *A comparison of spectrogram generation against F_1 score, for five different pipeline configurations. The Y axis denotes the mean percentage of an input audio file generated as a full spectrogram, normalised for a per-call basis. The figure also uses a logarithmic y-scale for ease of comparison. The 'Tree & DNN' methods are a two-stage detection process where the tree is applied first, followed by the DNN detector. F_1 scores come from the mean detector precision-recall for each configuration, as seen in Table 5.1.*

spectrogram size for a single call to 27ms. The combined methods use DNN offsets equivalent to the utilised tree's input size; 1 for the shallow tree & DNN, and 5 for the deep tree & DNN. These offsets were similarly found to optimise detector performance.

Figure 5.1 shows the the average percentage of spectrograms generated per call for Decision Tree, DNN Detector, and combined pipeline configurations against each method's F_1 score on the test dataset. In the combined pipelines ('Tree & DNN' in Figure 5.1, decision trees detect calls first. Then, after a detected call leaves the spectrogram queue, the DNN evaluates the generated spectrogram as a secondary detector. In the lone tree configurations, the DNN is skipped, and decision tree detection are always classified after leaving the spectrogram generation queue. Similarly, the DNN detector is not gated by the trees, nor the spectrogram queue, when used alone. F_1 score was calculated from the 'Detector Performance' section of Table 5.1; performance is discussed in more detail in the following experiment.

We find that any method using the Goertzel algorithm leads to a reduction in the proportion of spectrograms generated by a range of 6.4 to 9.1 times. The DNN detector generates 100% of each recording as a spectrogram, as expected; for the test dataset, this works out to an average of 14.61% of an audio file per call. In comparison, the solo tree methods only generate an average 1.61% of audio as spectrograms per call. The combined methods generate slightly more, at an average of 1.85% of the input audio

file per call.

We also observe that in both the solo and combined pipeline configurations, the shallow 1ms trees lead to more spectrogram generation than the deep 5ms trees. Between the solo configurations, this difference is minuscule at just 0.0179%; far less than a single spectrorgam bin. The difference is more pronounced between the combined DNN and tree configurations, where the shallow 1ms tree generates 0.264% of audio per call more than the deep 5ms tree. Such a difference is mostly due to the increased DNN offset within the deep combined configuration, though additional generations may arise due to the input size and detection accuracy of the respective models.

In general, the F_1 score for configurations applying the Goertzel algorithm were significantly lower than for the DNN detector. This is to be expected, as the lone DNN detector has access to far more DFT information (513 x 20 values) than the decision trees, which only evaluate up to ten frequency values per detection. Considering this 1000x reduction of information, the lone trees in particular perform extremely competitively. The combined methods see some increase in overall performance, at the cost of slightly increased spectrogram generation. Interestingly, the shallow combined pipeline outperfroms the deep combined configuration, when the lone shallow tree under-performs in comparison to the deep tree. This large performance increase is most likely due the imprecisions of the 1ms tree being mitigated by the DNN detector. It is debatable if the increase in evaluation time that comes from the two-step detection process is worth the increase in performance. Pipeline performance is discussed in greater detail in section 5.1.2.

It is important to note that the differences in generation proportion do not account for an six to nine-fold reduction in the computation done by Goertzel methods, as input still has to be evaluated using the Goertzel algorithm. According to the efficiency inequality in section 3.1.3, we know the number of Goertzel iterations each tree performs is more efficient than an equivalent spectrogram generation for that tree's input size. We assume that this also holds true when accounting for the additional computation that comes with traversing each tree. Such an assumption is reflected in the observed execution times of goertzel methods, which evaluate faster than the DNN method (especially with using smaller tree sizes). However, we do not report execution times between methods, as such a measure is implementation dependant. Similarly, we do not take into account the performance difference between using a decision tree versus a DNN.

5.1.2 Precision Recall Evaluation

In this experiment, we report the performance of both the detector, as well as the performance of the DNN classifier, in order to determine overall pipeline performance. This is because classifier performance is directly linked to call detection; if an annotation goes undetected, it cannot be classified. Therefore, such an occurrence is not just a false negative for the detector, but also for the classifier. Classifier performance is measured at the genus level. For a more readable performance evaluation, the true positive, false positive, and false negative counts are summed across all genera, and used to calculate the mean precision and recall values displayed in Table 5.1.

Pipeline Configuration	Detector Performance		Classifier Performance	
	Precision	*Recall*	*Precision*	*Recall*
DNN Detector	87.64 %	**90.88** %	58.97 %	**56.06** %
Shallow Tree	85.80 %	68.07 %	64.04 %	53.38 %
Deep Tree	87.66 %	68.06 %	56.97 %	47.13 %
Random (1.16%)	21.85 %	14.73 %	39.19 %	5.84 %
Shallow Tree & DNN	**99.42** %	67.24 %	**75.73** %	52.76 %
Deep Tree & DNN	99.16 %	64.50 %	66.15 %	46.10 %
Random (1.85%)	21.79 %	16.40 %	39.49 %	6.61 %

Table 5.1: *The performance of the detector and classifier for each pipeline configuration. Configurations are grouped by the proportion of spectrograms they generate. Values highlighted in bold are the top results for each column. The 'Tree & DNN' methods are a two-stage detection process where the tree is applied first, followed by the DNN detector. The random methods use a unique classifier, trained to detect calls at any position within the spectrogram. This alternate classifier increased classification performance by around 1% for precision and recall for the random models. The unique classifier performs worse than the typical classifier for all other models, hence why it is only mentioned in passing detail.*

We additionally establish a random detection method to use as a baseline to our Goertzel methods. Each random process is initialised with the proportion of temporal bins it can generate as a spectrogram per call in each audio file. As usual, each bin is equivalent to 1ms of audio. After the current audio clip is loaded, a random 20ms range within the current audio clip to generate a spectrogram, which is compared against the time ranges in the annotation files to determine whether a call has been 'detected' or not. If the spectrogram intersects with an annotation by at least 3ms, then it will be passed along to a unique DNN classifier, which was trained to classify calls from random positions within the spectrogram. This process is repeated until the total number of generated bins is greater than or equal to the allotted bins per call.

We tested our methods against two random baselines; one with 1.61% spectrogram generation per call, and another with 1.85% generation per call. These proportions were obtained from the average per-call spectrogram generation proportions for the lone tree methods and the combined methods respectively, as discussed in section 5.1.1. Each random method was evaluated on the test data 200 times, with the average precision recall of the detector and the classifier on display in Table 5.1. All other models were as described in the previous experiment (see section 5.1.1), and were evaluated a single time each on the test dataset due to their deterministic nature.

We find that the DNN detector, which processes all audio into spectrograms, performed with the highest recall. This detection method additionally resulted in the best recall performance within the classifier. However, though the detector itself performed much worse, the lone shallow tree offers competitive classifier recall for large reduction in computation.

This is rather unexpected, due to the large disparity in detector recall between models. The cause of this closeness could be different queue systems used by the DNN detector

versus the decision trees. The tree method of spectrogram generation is inherently more consistent; spectrograms will only be passed to the classifier if audio has been detected and then waited for some constant time in the queue. In comparison, the DNN detector generates a spectrogram at every iteration of the pipeline, meaning that the position of the detected call within the spectrogram is highly variable in comparison. While the DNN offset was introduced to mitigate this variability, a dynamic offset may have increased performance on the classifier. Alternatively, the classifier itself may be to blame for this closeness; perhaps, given that the test data consists of species less common in the training data (see the 'Reserve' column in Table A.1), the classifier struggles to achieve more than 56% mean recall. However, the classifier tends to exceed this performance when evaluated outside of the pipeline (see section 5.2.2.2), so this possibility seems less likely.

We find that the combined methods of a decision tree followed by the DNN detector result in extremely high detection precision, in exchange for slightly decreased recall compared to the lone tree methods. Such a split is expected, as the two step detection process is very effective at filtering the false positive detections of the decision trees. The decision tree similarly filters false positives that the DNN detector may have judged to be true calls. However, due to their different training methods, some accurate detections by the decision trees are denied by the DNN. This is especially likely for calls that are detected 'late' by the decision tree, as the DNN was only trained on spectrograms of the first 20ms of all calls. Therefore, if a long call is detected beyond this limit, the DNN will almost certainly not recognise it. This results in a lower recall for the combined methods for the combined methods in comparison to the lone trees, though this situation may have been possible to remedy with additional training data.

The classifier consistently performed worse for configurations involving the deep tree over the shallow tree, despite both methods having similar detector recall across both lone and combined configurations. This was almost certainly due to the larger input size of the deep tree, at 5ms of audio or 1500 samples. While the increased size helped with precision when compared to the lone shallow tree, it also severely limits the possible positions of detected calls within the spectrogram. If the deep tree misses the first chance to detect a call, the next chance it gets is in another 5ms of audio, which may result in the read head moving far enough to hinder classifier performance. In comparison, the 1ms movement of the shallow pipeline is minimally likely to affect the position of the output call within the spectrogram, unless many detections are missed.

Finally, both random baselines were easily out-performed by the solo tree detectors and the combined detector pipeline configurations. This is to be expected, as while each method within a group of generates a similar small amount of full spectrograms, the Goertzel algorithm yields far much more information about the current state of the audio than random chance. Unsurprisingly, the random baseline that generates a greater amount of spectrorams lead to a greater performance in terms of detector and classifier metrics.

5.1.3 Pipeline Evaluation Conclusion

Overall, the DNN detector is the best performing pipeline configuration, meaning maximal spectrogram generation remains the most preferable option when resources are not an issue. The shallow decision tree provides competitive classification performances, though this may be due to other limitations. The two-step shallow tree & DNN configuration offers a successful middle-ground between both methods, and currently achieves the highest overall classifier F_1 score. However, it is debatable whether the increase in spectrogram generation and evaluation time is worth the performance increase when compared to the lone shallow tree. While these exact results may not hold true for a more robust pipeline with a dynamic DNN offset, decision trees that use the Goertzel algorithm nevertheless present a viable alternative detector method for efficiency driven bio-acoustic monitoring devices.

The deeper tree methods are unfortunately difficult to justify using within the Goertzel pipeline. While the lone deep tree achieves a slight advantage over the shallow tree for detector precision, its larger input size leads to a consistently worse classification performance. Furthermore, the slight increase in precision is hardly worth a doubled count of Goertzel evaluations. The precision advantage also vanishes when both trees are used in two-step detection processes with the DNN.

5.2 Component Evaluation

This section details the evaluation of the individual components of the pipeline. The section begins with a breakdown of the performance of the eight trained decision trees; the best performing of these influenced which pipeline configurations were tested in section 5.1. The top-performing trees are further evaluated against a simple peak-amplitude detection method. We then follow with an evaluation of our deep learning methods, compared against random forest baselines.

5.2.1 Evaluation of Decision Trees

5.2.1.1 Comparison of Tree Performance

As discussed previously, eight Goertzel decision trees were implemented in total, varying by depth and input size. In this experiment we evaluate the performance of all eight models. The results determined which trees were selected for the pipeline evaluation experiments in section 5.1.

Trees were evaluated for performance on the same set of testing data, both within and outside of the pipeline. For the pipeline tests, a gap of 5ms is used for all trees. The read head was of variable size, and always set to match the required number of input samples each tree was trained on. Queue timers of 15ms are used for all decision trees. This choice would lead to decreased classifier accuracy for trees with larger inputs, but minimally affects detection performance.

Figure 5.2 displays the precision recall performance of the eight trees, split into two graphs by maximum depth, outside of the pipeline. Here, precision and recall is

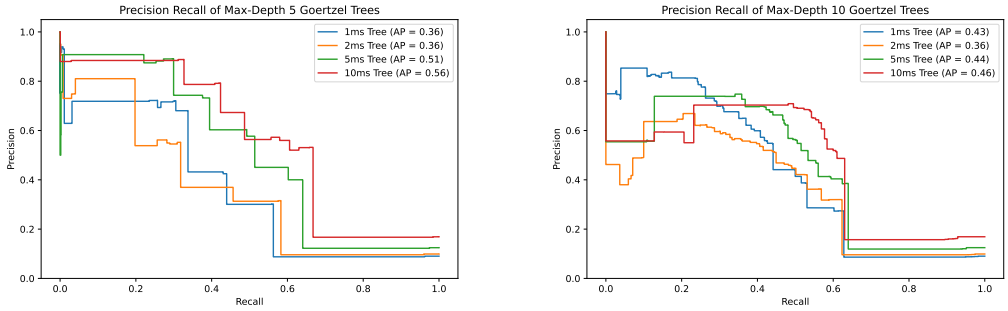

Figure 5.2: *The comparative precision recall of all trained decision trees, split into two graphs by tree depth. These curves pertain to raw performance on sequential input audio sequences. In practice, various measures within the pipeline help to improve the reliability of the trees. In both graphs, the blue curve is a 1ms input tree, the orange is a 2ms tree, the green curve is a 5ms tree, and the red curve is a 10ms tree.*

evaluated on whether a given x ms input slice of audio contains any part of a call. Table 5.2 displays the mean precision, recall, and F_1 score of the trees when tested within the pipeline. For this set of values, only detections that go on to be generated as spectrograms are evaluated. As false positives may be flushed from the queue, low precision tends to be less important than low recall for this method of evaluation. However, the evaluation criteria remains as the same as in Figure 5.2. Table 5.2 also displays the mean execution time for each tree, measured separately from the pipeline. This data was gathered by evaluating 1,900,000 ms of audio per tree, measuring the average time taken using the 'timeit' python module.

We found that increased depth had a varied effect on tree performance both within and outside of the pipeline. The deep 2ms, 5ms, and 10ms trees all saw a decreased raw precision, particularly at low recall levels, when compared to their shallow counterparts. Within the pipeline, we observed a similar drop in mean precision across all models, in exchange for increased mean recall. All but the 1ms tree saw an increase in F_1 score, with the deep 5ms tree obtaining the greatest of all evaluated trees. Such a result is likely due to the spectrogram queue within the pipeline, which rewards imprecise trees over inaccurate ones through the flushing of false positives.

For raw scores, trees with a larger input size generally outperformed trees which take less audio as input. The 5ms and 10ms trees always see a greater average precision than the 1ms and 2ms trees, with 10ms typically being the best performing overall. The exception to this is for low recall deep trees, where the 1ms tree achieves the best precision. Additionally, we observe that 2ms trees perform poorly, almost always giving similar or worse results than the 1ms trees at all depths.

Within the pipeline, we typically observe an increase in precision and a decrease in recall. Trees that take an even number of milliseconds of audio as input were consistently outperformed by smaller trees with an odd input size. For the 2ms detectors, this may be partially due to pipeline variables, which do not fully divide by the stride of the read head. 10ms trees perform very poorly. This was most likely due to the training process; the data, as detailed in 4.2.2, was more likely to be inappropriately labelled for larger

Tree	Mean Precision	Mean Recall	F_1 score	Time per MS of Audio (ms)
5d-1w	85.07 %	71.81 %	0.779	0.286
5d-2w	84.63 %	61.59 %	0.713	0.280
5d-5w	95.36 %	61.97 %	0.751	0.281
5d-10w	95.74 %	56.10 %	0.707	0.344
10d-1w	73.88 %	77.55 %	0.757	0.572
10d-2w	78.72 %	73.94 %	0.763	0.559
10d-5w	91.04 %	70.30 %	0.793	0.609
10d-10w	88.88 %	58.76 %	0.708	0.747

Table 5.2: *The performance of each trained decision tree. Precison, recall, and F_1 score were calculated within the Goertzel pipeline, whereas mean execution time was calculated separately. The values in the 'Tree' column denote the tree evaluated; a xd-yw label denotes a tree with Max-Depth=x and input length y ms.*

input widths. Such problems maybe have been prevented with a more robust training system.

For execution speed, all deep trees saw increased evaluation times per millisecond of audio. This is to be expected; each deep tree evaluates up to 10 frequencies, whereas the shallow trees always evaluate 5. It is hard to argue that the increase to performance is worth the time considering that shallow trees achieve a higher F_1 score than 3 of the 4 deep trees. Intriguingly, input size did not have a consistent effect on execution speed. 1ms, 2ms, and 5ms trees always performed at roughly the same rate, with 2ms trees always being the quickest per milisecond of audio. 10ms trees were consistently much slower than any of their counterparts.

Overall, there are two standout trees of the eight tested. 5d-1w achieves a high recall with minimal execution time, whilst 10d-5w achieves a similar recall with higher precision, albeit much more slowly. Crucially, both of these trees are more efficient than generating a spectrogram at their respective input sizes, as per Lyons [25]; the same can not be said for the deep trees with small input windows, 10d-1w and 10d-2w, which were determined inefficient in section 3.2.2. The remaining four trees are also theoretically more efficient than an FFT, but struggle to perform as well as the standout trees. Therefore, the shallow 1ms tree and the deep 5ms tree were selected for evaluation in the pipeline experiments.

5.2.1.2 Evaluation Against Baseline

The two 'best' decision trees, as specified in section 5.2.1.1, were further compared against a peak amplitude detector, based upon a baseline used in Skowronski and Harris [43]. This detector takes the normalised absolute amplitudes of 1ms of audio as input, and compares them to a threshold value t. If the maximum, or 'peak', amplitude within the input is $\geq t$, then the detector predicts a call. t was determined by iterating through the training audio for the decision trees in 1ms slices, and calculating the mean peak amplitude of all annotated calls. The peak amplitude detector was then implemented into the pipeline, optionally replacing the role of the decision trees. All three methods were re-evaluated on the same test data used in 5.2.1.1.

Method	Mean Precision	Mean Recall	F_1 score
Amplitude Detector	67.50 %	72.00 %	0.697
Shallow Tree (1ms input)	85.07 %	71.81 %	0.779
Deep Tree (5ms input)	95.74 %	56.10 %	0.707

Table 5.3: *The performance of two selected goertzel-algorithm decision trees, compared to the performance of a baseline amplitude detector. 'Shallow Tree' is the same model as '5d-1w' in Table 5.2, and 'Deep Tree' is equivalent to '10d-5w'.*

The peak amplitude detector was much more imprecise than the decision trees using the Goertzel algorithm, which is to be expected, given the simplicity of the metric. However, this imprecision allowed for a very high recall, beating out both decision trees. While the closeness of F_1 scores of all models may give the impression that the Goertzel algorithm only slightly outperforms a much simpler metric, it should be noted that the test dataset only contains bat calls; therefore, any loud phenomena is likely to be a call. In the field, this is highly unlikely to be the case, as loud acoustic phenomena may arise from the weather, other wildlife, or man-made sources. Due to the targeting of specific frequencies, the Goertzel methods are more robust to such false positive detections. Comparatively, a simple peak amplitude detector has no recourse against non-bat acoustic phenomena. Thus the decision trees are the more preferable method.

5.2.2 Evaluation of Neural Networks

5.2.2.1 DNN Detector Evaluation

We evaluated the performance of the DNN detector against the two datasets which were not used in its training or validation, nor reserved for the pipeline evaluation.

Figure 5.3 shows the performance of the detector network in comparison to a baseline random forest of 128 trees. We find that both models are equally precise for low recall values. As recall increases, the DNN maintains a strong performance, while the random forest begins to decline in precision.

The deep neural network handily outperforms the random forest. This matches the results found by Mac Aodha et al. [26], where their convolutional neural networks similarly outperformed random forests. However, we find that the random forest provides much stronger competition in our implementation. This is most likely due to the increase in training data we have available; while Mac Aodha et al. [26] were limited to four thousand training bat calls, we have many tens of thousands.

5.2.2.2 DNN Classifier Evaluation

We evaluated classifier performance on two different datasets. First, we tested it on the aggregated test set, which is described in the classifier section of the dataset chapter (section 4.2.1.3). We then further tested it on one of the two datasets that were not incorporated into the aggregate set. This was done in order to test the full capabilities of the classifier. The aggregated test data contains every species, but has been edited; the non-aggregate data is unedited, but contains only half of all species. In testing on

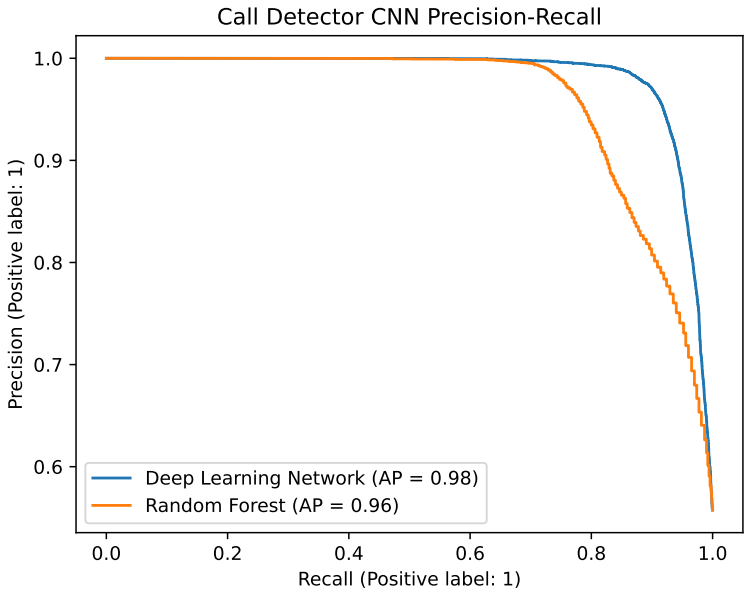

Figure 5.3: *The performance of the Detector DNN compared to a baseline random forest with 128 trees. Note the partial scale on the y axis; this is to highlight the differences between the models. The 128 trees of the random forest result in a set* 128^2 *possible output probabilities, leading to a slightly quantised curve.*

both datasets, we hope to understand how the model performs across all species, and for unfamiliar distributions of species.

Figure 5.4 contains the micro-averaged performance for both datasets, compared to a baseline random forest. We calculate results at both the species and the genus level. As elsewhere in this paper, the genus level results are the maximal probabilities across all species of the same genus; regardless of the level we report, each classifier is always making decisions at the species level. The performance has been micro-averaged, meaning the average precision recall across all classes is displayed.

Overall, we find that the DNN classifier outperforms the random forest baseline across both datasets. It does so at both the species and the genus level, though this is expected, as both are fundamentally the same classifications. The random forest methods tend to be more precise than the DNNs for low-recall data, but become more and more outmatched as recall increases. However, the difference in low recall precision between models is lower for the non-aggregate set, suggesting that the random forest struggles at generalisation compared to the DNN.

We also find that across both datasets, genus level evaluation is more effective than species level. Therefore, when evaluating pipeline methods, we choose to evaluate at the genus level. Both the DNN classifier and the random forest classifier performed better on the aggregated test set compared to the non-aggregated raw test set. Once more, this is expected, as the non-aggregated test set is atypical of the training data.

Figure 5.5 displays the precision recall for the top five species across both test datasets

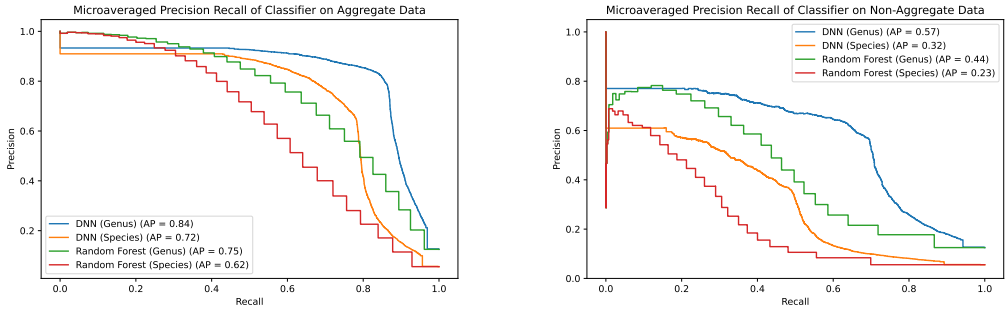

Figure 5.4: *The performance of the DNN classifier against the Random Forest baseline, for both the aggregate (left) and non-aggregate (right) test data. The baseline random forest curves are jagged due to their small size, at 32 trees each.*

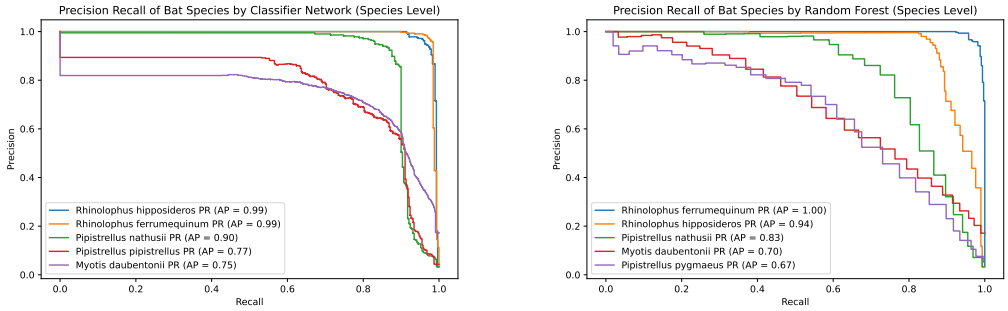

Figure 5.5: *Precision recall of species which achieved top-5 average precision for the DNN classifier (left) and the random forest baseline (right). Here, results display performance across the aggregate and non-aggregate test sets combined. The baseline random forest curves are jagged due to their small size, at 32 trees each.*

for the DNN classifier, in comparison to the top five for the random forest. We find that both models excelled at detecting similar species, with the Rhinolophus genus achieving excellent results in both models. The remaining three species in each graph have some variation, with Pipistrellus Pipistrellus being present in the top 5 DNN species, and Pipistrellus Pygmaeus appearing in the random forest top 5. However, the same genuses appear in either top five, reinforcing the model's strengths at genus level classification.

Finally, figure 5.6 shows the confusion matrix for the DNN classifier. We see that the model excells at classifying the genera of Rhinolophus, Barbastellus, and Pipistrellus, as well as select species in the Myotis family, such as Myotis Daubentonii and Myotis Nattereri. However, the classifier fails to predict the Plecotus genera well across both species in the dataset. Plecotus Austriacus is commonly confused for Myotis Nattereri, whereas Pletocus Auritus is mistaken for a wide range of other species, the most common being Eptesicus Serotinus. This makes some sense considering the train-test split of classifier data, observable in Table A.1, as over half of all Plecotus bats are included in the reserve data.

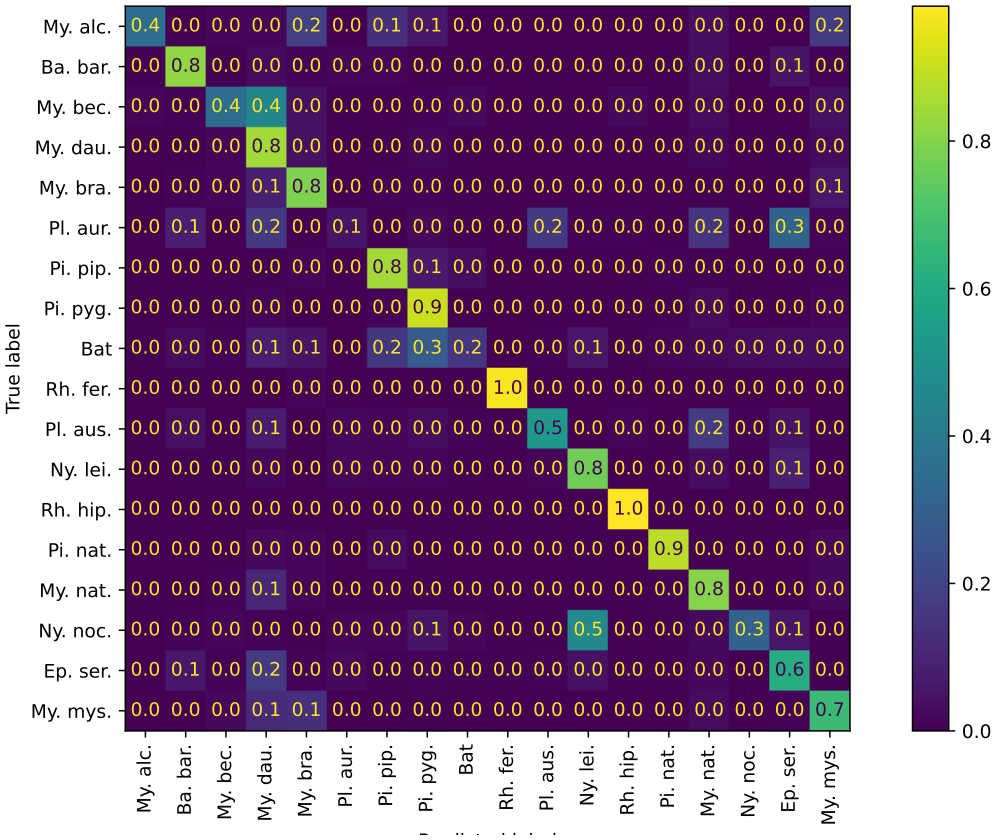

Figure 5.6: *The confusion matrix for the DNN classifier across both test sets. Values are normalised by the total of each class per row. Class labels had to be shortened to fit within the bounds of the figure. We observe that the model struggles to correctly identify the entire Plecotus genera, as well as Nyctalus Nocta in particular.*

The model also fails to predict the generic 'Bat' class, but this is understandable, as this is not a real bat species, and likely comprises of the calls of multiple different species. Finally, we observe that while certain species of bat are often misclassified, such as Myotis Alcathoe, Myotis Bechsteinii, and Nyctalus Nocta, they are almost always mixed up with other species of their respective genera. Therefore, the rise in performance when genus evaluation is used is mostly due to eliminating species-level inaccuracies in the Myotis and Nyctalus families.

Chapter 6

Conclusions

In conclusion, we find that implementation of the Goertzel algorithm within bio-acoustic monitoring devices leads to a significant reduction in the amount of full spectrogram generation. This represents an overall increase in model efficiency. However, the tested detection methods which implement the Goertzel algorithm do not perform as well as the standard convolutional neural network method proposed by Mac Aodha et al. [26], resulting in a slight loss of classification accuracy when determining species.

One limitation of the project was the lack of a dynamic queue for the DNN detector, based upon the call location within the spectrogram. This may have limited the classifier accuracy for the detector DNN, and give a false impression of the difference in performance for the simple decision tree methods. Additionally, we did not apply any denoising methods to our generated spectrograms. A method akin to the one presented in Aide et al. [1] could have feasibly increased performance across the board for our spectrogram-based detectors.

While this project investigated the performance of simple detectors using the Goertzel algorithm against standard convolutional neural networks, it does not consider more cutting edge architectures, such as the transformer network provided in Mac Aodha et al. [27]. Future work should be done in investigating whether the Goertzel algorithm can be applied to pipelines involving such architectures. Alternatively, research could be conducted into incorporating the Goertzel algorithm into the neural networks themselves, which may be able to learn deeper, more accurate embeddings than the decision tree methods tested in this report.

Finally, the bat detection and classification pipeline proposed in this report was only tested in simulation. Further research should be conducted as to whether or not the efficiencies enabled by this architecture are meaningful when implemented on hardware and used in the field.

Bibliography

[1] T Mitchell Aide, Carlos Corrada-Bravo, Marconi Campos-Cerqueira, Carlos Milan, Giovany Vega, and Rafael Alvarez. Real-time bioacoustics monitoring and automated species identification. *PeerJ*, 1:e103, 2013.

[2] Sercan Alipek, Moritz Maelzer, Yannick Paumen, Horst Schauer-Weisshahn, and Jochen Moll. An efficient neural network design incorporating autoencoders for the classification of bat echolocation sounds. *Animals*, 13(16):2560, 2023.

[3] Richard A Altes. Detection, estimation, and classification with spectrograms. *The Journal of the Acoustical Society of America*, 67(4):1232–1246, 1980.

[4] David W Armitage and Holly K Ober. A comparison of supervised learning techniques in the classification of bat echolocation calls. *Ecological Informatics*, 5(6):465–473, 2010.

[5] Robert MR Barclay. Bats are not birds—a cautionary note on using echolocation calls to identify bats: a comment. *Journal of Mammalogy*, 80(1):290–296, 1999.

[6] Anna Berthinussen, Olivia C Richardson, and John D Altringham. *Bat conservation: global evidence for the effects of interventions*, volume 5. Pelagic Publishing Ltd, 2014.

[7] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[8] AL Collen. *The evolution of echolocation in bats: a comparative approach*. PhD thesis, UCL (University College London), 2012.

[9] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[10] Artur J Ferreira and Mário AT Figueiredo. Boosting algorithms: A review of methods, theory, and applications. *Ensemble machine learning: Methods and applications*, pages 35–85, 2012.

[11] Winifred F Frick, Tigga Kingston, and Jon Flanders. A review of the major threats and challenges to global bat conservation. *Annals of the new York Academy of Sciences*, 1469(1):5–25, 2020.

[12] Sarah Gallacher, Duncan Wilson, Alison Fairbrass, Daniyar Turmukhambetov, Michael Firman, Stefan Kreitmayer, Oisin Mac Aodha, Gabriel Brostow, and Kate

Jones. Shazam for bats: Internet of things for continuous real-time biodiversity monitoring. *IET Smart Cities*, 3(3):171–183, 2021.

[13] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[14] Gerald Goertzel. An algorithm for the evaluation of finite trigonometric series. *American Math. Monthly*, 65:34–35, 1958.

[15] Guillén, B Juste, and Ibáñez. Variation in the frequency of the echolocation calls of hipposideros ruber in the gulf of guinea: an exploration of the adaptive meaning of the constant frequency value in rhinolophoid cf bats. *Journal of Evolutionary Biology*, 13(1):70–80, 1999.

[16] Trevor Hastie, Robert Tibshirani, Jerome Friedman, Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Random forests. *The elements of statistical learning: Data mining, inference, and prediction*, pages 587–604, 2009.

[17] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[18] Andrew P Hill, Peter Prince, Evelyn Piña Covarrubias, C Patrick Doncaster, Jake L Snaddon, and Alex Rogers. Audiomoth: Evaluation of a smart open acoustic device for monitoring biodiversity and the environment. *Methods in Ecology and Evolution*, 9(5):1199–1211, 2018.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[20] Gareth Jones and Marc W Holderied. Bat echolocation calls: adaptation and convergent evolution. *Proceedings of the Royal Society B: Biological Sciences*, 274(1612):905–912, 2007.

[21] Gareth Jones, David S Jacobs, Thomas H Kunz, Michael R Willig, and Paul A Racey. Carpe noctem: the importance of bats as bioindicators. *Endangered species research*, 8(1-2):93–115, 2009.

[22] Gordon Jones, T Gordon, and J Nightingale. Sex and age differences in the echolocation calls of the lesser horseshoe bat, rhinolophus hipposideros. *Mammalia*, 56 (2):189–194, 1992.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Wen-kai Lu and Qiang Zhang. Deconvolutive short-time fourier transform spectrogram. *IEEE Signal Processing Letters*, 16(7):576–579, 2009.

[25] Richard G Lyons. *Understanding digital signal processing, 3/E*. Pearson Education India, 1997.

[26] Oisin Mac Aodha, Rory Gibb, Kate E Barlow, Ella Browning, Michael Firman, Robin Freeman, Briana Harder, Libby Kinsey, Gary R Mead, Stuart E Newson, et al. Bat detective—deep learning tools for bat acoustic signal detection. *PLoS computational biology*, 14(3):e1005995, 2018.

[27] Oisin Mac Aodha, Santiago Martinez Balvanera, Elise Damstra, Martyn Cooke, Philip Eichinski, Ella Browning, Michel Barataud, Katherine Boughey, Roger Coles, Giada Giacomini, et al. Towards a general approach for bat echolocation detection and classification. *bioRxiv*, pages 2022–12, 2022.

[28] Kevin L Murray, Eric R Britzke, and Lynn W Robbins. Variation in search-phase calls of bats. *Journal of Mammalogy*, 82(3):728–737, 2001.

[29] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[30] Stavros Ntalampiras and Ilyas Potamitis. Acoustic detection of unknown bird species and individuals. *CAAI Transactions on Intelligence Technology*, 6(3): 291–300, 2021.

[31] Henri J Nussbaumer. *Fast Fourier transform and convolution algorithms*. Springer, 1982.

[32] Michael J O'Farrell, Bruce W Miller, and William L Gannon. Qualitative identification of free-flying bats using the anabat detector. *Journal of Mammalogy*, 80(1): 11–23, 1999.

[33] Travis E Oliphant et al. *Guide to numpy*, volume 1. Trelgol Publishing USA, 2006.

[34] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[35] JoséM Paruelo and Fernando Tomasel. Prediction of functional characteristics of ecosystems: a comparison of artificial neural networks and regression models. *Ecological Modelling*, 98(2-3):173–186, 1997.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[37] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[38] Peter Prince, Andrew Hill, Evelyn Piña Covarrubias, Patrick Doncaster, Jake L Snaddon, and Alex Rogers. Deploying acoustic detection algorithms on low-cost, open-source acoustic sensors for environmental monitoring. *Sensors*, 19(3):553, 2019.

[39] Danilo Russo and Christian C Voigt. The use of automated identification of bat echolocation calls in acoustic monitoring: A cautionary note for a sound analysis. *Ecological Indicators*, 66:598–602, 2016.

[40] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[41] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[42] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

[43] Mark D Skowronski and John G Harris. Acoustic detection and classification of microchiroptera using machine learning: lessons learned from automatic speech recognition. *The Journal of the Acoustical Society of America*, 119(3):1817–1833, 2006.

[44] Dan Stowell and Mark D Plumbley. Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning. *PeerJ*, 2:e488, 2014.

[45] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12): 2295–2329, 2017.

[46] JM Szewczak. Sonobat (4.3)[computer software], 2019.

[47] Nancy Vaughan, Gareth Jones, and Stephen Harris. Identification of british bat species by multivariate analysis of echolocation call parameters. *Bioacoustics*, 7 (3):189–207, 1997.

[48] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[49] Jie Xie, Kai Hu, Ya Guo, Qibin Zhu, and Jinghu Yu. On loss functions and cnns for improved bioacoustic signal classification. *Ecological Informatics*, 64:101331, 2021.

Appendix A

Supplimentary Dataset Information

This section contains supplementary figures which were unable to fit in the main body of the report.

Table A.1 portrays the number of calls of each bat species across the entire dataset, as well as whether they were used as training, test, or validation data for the classifier-training dataset.

Figure A.1 portrays the varied call structures of all bat species present within the data used in this project. The figure originates from Mac Aodha et al. [27].

Species	Train	Validation	Test	Reserve	**Total**
Myotis alcathoe	331	18	18	504	**871**
Barbastellus barbastellus	892	49	49	2	**992**
Myotis bechsteinii	663	36	36	149	**884**
Myotis daubentonii	5389	299	299	336	**6353**
Myotis brandtii	1749	97	97	1	**1944**
Plecotus auritus	511	28	28	556	**1123**
Pipistrellus pipistrellus	1710	95	95	1	**1901**
Pipistrellus pygmaeus	2121	117	117	2	**2357**
Rhinolophus ferrumequinum	1477	82	82	1	**1642**
Plecotus austriacus	330	18	18	480	**846**
Nyctalus leisleri	1044	58	58	0	**1160**
Rhinolophus hipposideros	1317	73	73	1	**1464**
Pipistrellus nathusii	1305	72	72	1	**1450**
Myotis nattereri	2371	131	131	2	**2635**
Nyctalus noctula	182	10	10	204	**406**
Eptesicus serotinus	972	54	54	1512	**2592**
Myotis mystacinus	2126	118	118	409	**2771**
‘Bat’ (Generic)	1087	60	60	193	**1400**

Table A.1: *The total calls for each bat species present across all datasets. This table also includes the ‘train / validation / test’ split used for the classifier training data, as discussed in section 4.2.1.3. The 2 datasets not used in the classifier set are included in the ‘Reserve’ section, as these calls were reserved for pipeline and component evaluation in chapter 5.*

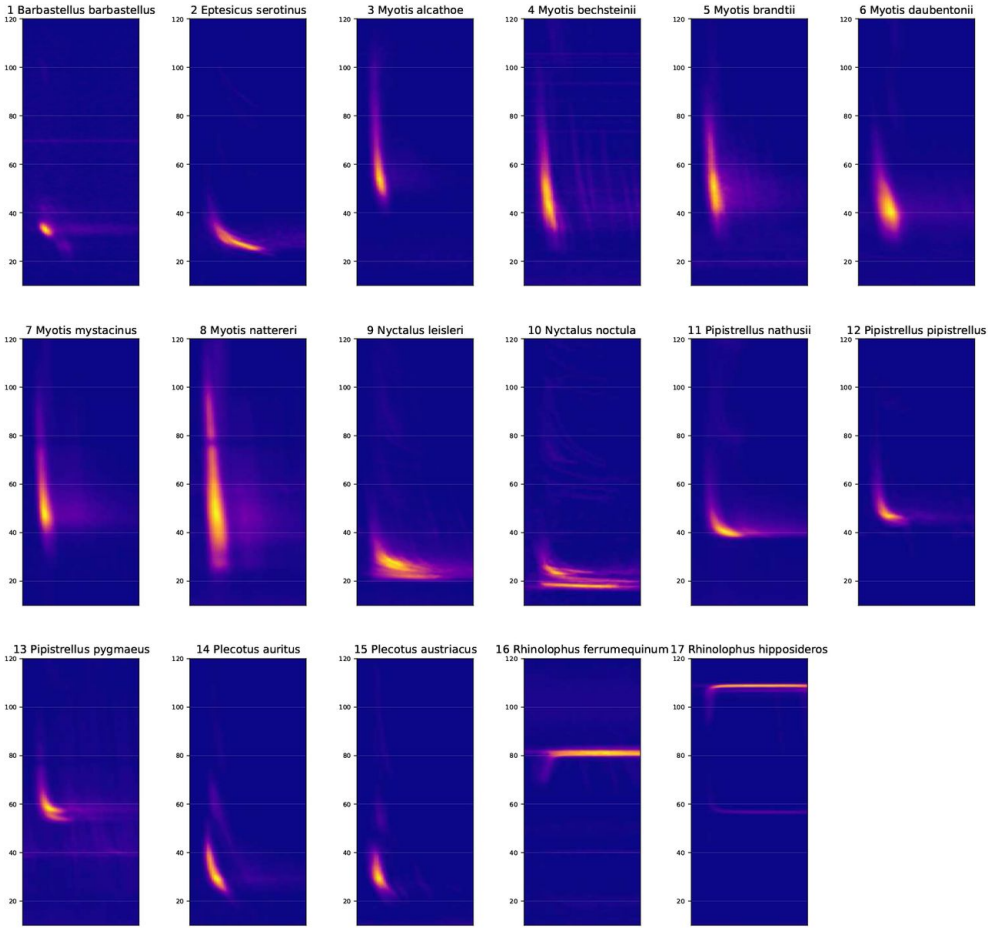

Figure A.1: *The varied call structures of the 17 unique bat species included in the dataset. Each image is a spectrogram of a call made by the labelled species. As the generic 'Bat' class from the annotations is not a true bat species, its associated spectrograms are excluded. This figure is sourced from Mac Aodha et al. [27].*