Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Investigating Methods of Birdsong Recognition on Mobile Devices

Owen Burke

Masters in Computer Science (MCS)
Final Year Project, August 2021

Supervised by Dr. Fergal Shevlin

Submitted to the School of Computer Science and Statistics
O'Reilly Institute, University of Dublin, Trinity College, Dublin 2, Ireland

Declaration:

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

Owen Burke

Owen Burke

Summary

This final year project focused on the possible methods of classifying birdsong audio to their respective species, particularly in relation to mobile devices. A dataset of birdsong recordings was gathered from the website, xeno-canto.org, a public repository in which volunteers, ranging from amateur bird-watchers to research scientists from across the globe, can upload recordings of birdsong along with annotations and various forms of metadata.

A review of the current literature regarding content-based information retrieval with respect to audio was performed in order to establish the current state of the art in the field of audio classification, a lot of which pertains to music. This also included examining these approaches when applied specifically to birdsong, rather than their originally intended purpose.

From this analysis, a mobile application was developed to record audio from the user's environment, making use of a local convolutional neural network to generate predictions of which species of bird generated the incoming audio.

As part of this development, multiple approaches of generating audio data to train the neural network were examined as well as identifying the issues that are presented by making use of such technology and the possible approaches to mitigate such performance implications.

Abstract

Investigating Methods of Birdsong Recognition on Mobile Devices

Owen Burke, Masters in Computer Science

Supervised by Dr. Fergal Shevlin

August 2021

Conservation groups regularly survey bird populations in order to gauge the effects of human activity on species populations such as deforestation and global warming. There has also been a steady increase in amateur ornithology [1] over recent years. One problem that both these groups encounter is attempting to classify species without being able to physically see the animal. Therefore, they must rely on classification through audio which can be relatively error prone, especially for amateurs.

This paper examines the ability of audio classification approaches developed for music applications to classify birdsong and their performance. These approaches include audio-fingerprinting (sometimes referred to as audio identification) and audio-matching. It was discovered that these approaches were not sufficiently accurate to be useful in real world applications, although they did outperform a random baseline.

This led to the adoption of an approach previously used for speech-to-text applications. This involved generating a spectrogram for the incoming audio and running inference on a CNN, trained on other spectrograms of birdsong.

It was discovered that this approach greatly outperformed the musical approaches and led to the development of an Android application that was intended to provide the capability to users in the field. However, this approach showed itself to be relatively weak at differentiating between birdsong produced by similar sounding bird species and also demonstrated signs of overfitting due to the size of the dataset available.

Acknowledgements

Thank you to Dr. Fergal Shevlin for his willingness to supervise my research and development over the course of this project and for his crucial guidance, suggestions and feedback throughout. Also, I would like to thank the team at xeno-canto for making their vast array of birdsong recordings freely available to the public and to all those that participated in submitting their data, without which this project would not have been possible.

Table of Contents

List of Figures

List of Tables

List of Code Listings

Terminology

Bird Vocalisation: The sound made by the vibration of vocal folds within birds. Bird vocalisation includes both bird calls and bird songs.

Birdsong: The sounds produced by birds that have melodic patterns and resemble music to the human ear. Compared to bird calls, these vocalisations are relatively complex. The songs are generally longer in duration than bird calls and are more often related to the process of courtship and displays of territory [2].

Bird Calls: Simple and generally short sounds produced by birds that serve the function of communication between birds in the local neighbourhood or as alarms for incoming predators [3].

Information Retrieval: The process of obtaining information pertaining to a certain document, retrieving the individual document or retrieving related metadata from a collection of documents (these could be pieces of text, images, sounds, etc.) [4].

Audio-Fingerprinting: The process of generating a deterministic unique identifier from an audio signal [5].

Audio-Matching: An abstraction of audio-fingerprinting, which involves retrieving all documents that musically relate (i.e. through tempo, pitch, etc.) to a provided audio query [6].

Melody: A succession of musical tones / notes that the listener perceives as aurally appealing [7].

Tempo: In music, tempo is the speed or pace of a particular piece [8].

Pitch: This is the quality of audio signals that allow the listener to rank particular sounds as “high” or “low” (allows for ordering on a frequency related scale) [9].

Ambient Noise (Background noise): Any sound other than the sound attempting to be monitored or isolated and is therefore treated as a form of interference.

Machine Learning: A field of study of algorithms that improve over time through experience and recognizing patterns in data.

Convolutional Neural Network: A form of neural network composed of a series of layers and kernels that convolves over an input space (typically derived from an image).

Neural Network: A series of algorithms that aims to emulate the process of pattern recognition in data that regularly occurs in the human brain.

Computer Vision: A field of computer science that study methods and processes that can allow computers to interpret and understand images or video.

Spectrogram: A spectrogram is a visual representation of an audio signal by illustrating its spectrum of frequencies as time passes (i.e. time is on the x-axis with frequency on the y-axis with the amplitude/power represented by pixel colour/intensity).

Hertz: The hertz is a unit of frequency defined as one cycle per second and denoted as Hz (the human hearing range is typically 20-20,000 Hz) [10].

Decibel: Decibels are a measure of sound intensity, or amplitude, denoted as dB. It is used for expressing the ratio between two physical quantities. The intensity of a sound in decibels is 10 Log_{10} (S_1/S_2), where S_1 and S_2 are the intensity of the two sounds. Typically, when determining the dB level of a sound, the other sound used for the calculation is that of a sound just perceptible to the human ear [11].

Overfitting: A form of behaviour in data modelling in which a given model or implementation produces results that too closely model a previously seen set of data points. A simpler analogy would be a machine learning model that has too closely modelled its training data (due to, for example, overtraining) and therefore cannot generalise to unseen data. This is typically seen in the accuracy figures when a model has exceedingly high training accuracy, but much lower validation (unseen) accuracy.

Underfitting: This is the opposite of overfitting. Rather than too closely modelling the training data, an underfit model fails to capture the behaviour of the provided data and therefore is too general to generate useful predictions. This has both a detrimental effect on the training and validation accuracy.

1 Introduction

1.1 Background & Motivation

The following section will provide a brief description of the background to the project and the motivation behind developing such technologies to assist in solving the problem of classifying the species of a bird through its birdsong only.

Today, conservation groups regularly conduct surveys of particular bird species populations in order to gauge the possible effects of global warming and other human activities that can affect species populations, for example, by introducing forced migration into the environment due to deforestation. As an example of a conservation group, BirdWatch Ireland conducts population surveys of Ireland's countryside birds every summer [12]. This survey relies on over 200 volunteers across the country, NPWS (National Parks and Wildlife Service) rangers and BirdWatch Ireland staff. These surveys have indicated that while most species populations remain relatively stable, there is a selection of species that show either worrying declines, or drastic increases, in population. These include the Goldfinch and Blackcap which have increased substantially over the past 20 years, Grey Wagtails which have shown massive fluctuations in numbers especially during sustained harsh weather conditions, and Kestrels and Skylarks which have shown serious declines in numbers.

It is important to record these numbers as it can lead to the study of why these particular species are affected and can, subject to appropriate action, lead to the restoration of their original numbers. This is important, as a lack of one particular species or a huge increase in another can lead to an ecological imbalance [13] which can have knock on effects on the numbers of other animals, not just birds, due to a lack of food or an overabundance of a particular predator.

To quote BirdWatch Ireland's website, where they describe the process of conducting these population surveys, "*during the breeding season, CBS (countryside bird survey) counters record all birds seen and* ***heard*** *during two early morning walks in pre-assigned 1km grid squares...as many birds are detected first by sound (****and often not seen****) it is helpful to be able to identify birds by song and call*".

As they describe, when conducting these surveys, the birds often reside in the tree canopy or in the hedgerows. Therefore, a lot of the classification is done through audio only. However, as mentioned, these surveys are conducted by volunteers, not necessarily experienced professionals. For those without a sizable amount of experience, classification through audio can be relatively error prone, especially for total beginners. This is especially true when having to deal with ambient noise, such as passing cars, wind noise, other animals in the environment and even bird vocalisations from other species that are less prominent in the soundscape.

This project aims to develop a mobile solution to combat these issues and assist in identifying birdsong through audio only. This application is also designed to be run entirely locally on the mobile device and have no reliance on external servers in the cloud. The reasoning for this is that these surveys can occur in very remote/extreme locations with no internet access or mobile phone reception due to the distance to the nearest cell tower or obstruction from objects in the environment/the environment itself (i.e. trees, mountains, etc.).

1.2 Objectives

When discussing the objective of the project at the highest level, the aim is to develop a mobile application for Android devices that, ideally, is capable of matching a bird to its respective species through an audio sample of its birdsong. The method employed as part of the application to determine the species will be found through the analysis mentioned below.

At a lower level, the project aims to analyse the accuracy, performance and applicability of different approaches to solving the problem of birdsong classification. This will include:

- Comparing the different approaches against each other with an emphasis on the classification accuracy of the methods against a standardised dataset of birdsong samples as well as the overall performance of the approach (i.e. is it too slow to be considered useful on mobile applications?).
- Comparing some of the approaches to variations of themselves. For example, an approach will be compared to itself when trained on an altered/pre-processed version of the standardised dataset.
- Comparing the most successful methods found to those currently used in state of the art approaches.

1.3 Methodology

The below section will provide some additional detail regarding the methodology that was alluded to above.

- **Training and Validation Accuracy**

  Each method will be analysed in terms of its accuracy (the percentage of audio clips in the relevant dataset that it matches to the correct species) on both a training and validation dataset. The training dataset will contain the audio clips that were used to train the model, generate the relevant fingerprints / features, etc. (the data that allows the methods to generate predictions). The validation dataset will contain audio recordings that the model has not seen in training. This analysis allows us to make judgements about whether the model is better at classifying audio it has heard already or whether it is able to generalise to audio in real-world scenarios that it has not heard before.

- **Pre-processed / Altered Data Accuracy**

  Depending on the training and validation accuracy, some methods will be compared against themselves, but when trained on an altered / pre-processed version of the training dataset to analyse whether a method such as adding noise to some of the training audio can make a method more robust.

- **Performance and Applicability**

Through the process of gathering the training and validation accuracy figures, we can come to certain conclusions regarding the performance of the method (i.e. does one method take much longer to train than another? Is there a particular method that is simply too slow at classifying the audio that it cannot be considered useful in the real-world?). This then allows us to make certain conclusions around the applicability of the method in achieving the high-level objective of developing an Android application.

- **State of the Art**

Having gathered accuracy figures for our methods, we can then analyse them with respect to state of the art approaches that are not aimed specifically at mobile applications. This can serve as a yardstick against which we can measure our own methods to decide whether they can be labelled as ‘successful’.

2 State of the Art

The first stage of this research project, as mentioned above, involved an analysis of the current methodology in the audio classification space. This chapter will describe the current state of the art regarding content-based information retrieval in music (as these methods will be examined later when applied to birdsong) and the modern approaches specifically aimed at classifying birdsong.

2.1 Musical Approaches

Similar problems to birdsong classification have been tackled before, particularly in the area of music, and can be classified under the umbrella term of content-based information retrieval [14]. That is, given some form of data, make use of only the raw data present to retrieve some additional relating information.

This is an important distinction to make as many information retrieval approaches will make use of metadata (this could be any additional descriptive data such as a timestamp, location, hardware information, IP address, etc.). For our purposes, this metadata could theoretically allow us to determine the bird species with greater accuracy by using, for example, the time of the year/season to filter out species from our considerations that we know cannot (or, more specifically, are not known to usually) be present in the environment.

By removing any consideration of metadata in our approaches and focusing on the raw data only we are, arguably, developing a more versatile approach that would be more useful in real world applications. This is due to the fact that we could classify species that are new to an area due to changing migratory patterns.

The musical approaches that we will consider are both ‘query by example’ approaches. This means that, given a piece of query audio (the example), generate some form of query to retrieve the relevant information. These two approaches are audio-fingerprinting (also known as audio identification) and audio-matching.

2.1.1 Audio-Fingerprinting

The general idea behind audio-fingerprinting is to return exact matches to the query audio with a small acceptable degree of variation due to external sources such as ambient noise or some other forms of interference such as humming from musical speakers.

The most well-known user application based around this concept is Shazam. This is a mobile application that can record samples of musical tracks and identify the song. The parallels between this process and the problem this paper aims to address are clearly evident.

The underlying process is that the sample recording is used to generate ‘audio-fingerprints’. These fingerprints are then sent to a matching service. The matching service simply compares the incoming fingerprints against a database of fingerprints that was constructed beforehand. The process of matching fingerprints is the performance bottleneck in this approach and usually makes use of some form of database management system to efficiently compare the fingerprints. The audio-fingerprinting implementation that was examined throughout this project was Dejavu,

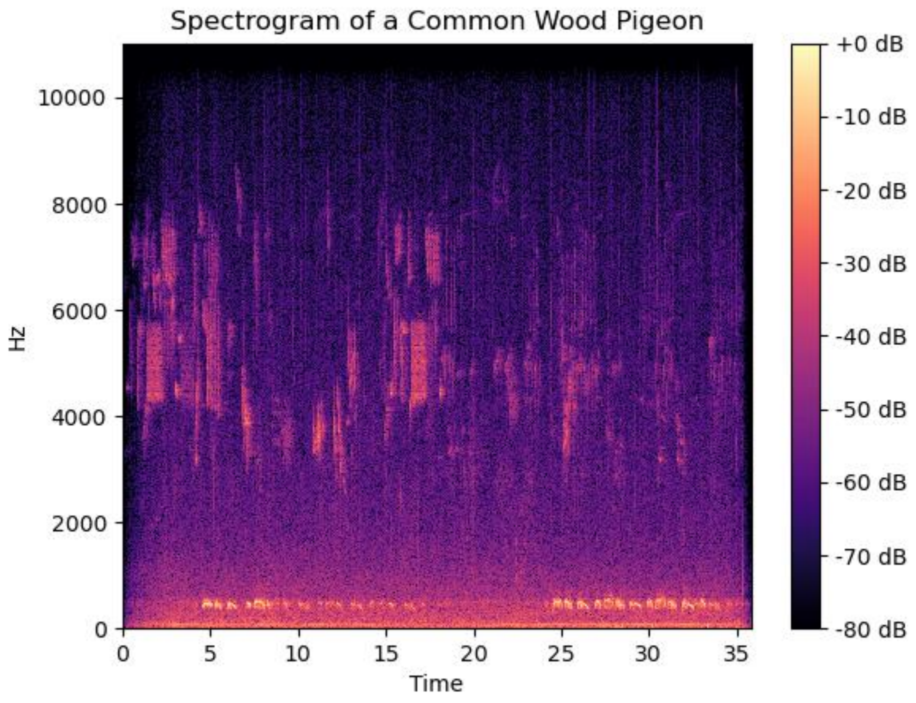


Fig. 1. A spectrogram of a Common Wood Pigeon’s call

an open-source Python library. The process works by iterating over a piece of audio in small windows, generating an individual spectrogram for each window (hence, multiple fingerprints for a single audio file). Fig. 1 is an example of a spectrogram.

Now that we have our spectrograms, in order to generate our fingerprints, Dejavu performs a process called ‘local peak finding’ by using image processing libraries to extract peaks from the spectrogram. The result is a series of time-frequency pairs, as the actual amplitude is not required as long as it is a local peak.

After calculating our time-frequency pairs for a window of time, we move on to calculating our fingerprints. This is done using a hash function. This is a one-way function as the original input cannot be deduced from the output values. The input to the hash function is the peak frequencies and the time difference between successive peaks. The results of the hash function is a fingerprint. In order to generate our database of fingerprints, we perform this process over all of our audio recordings and populate a locally running MySQL database.

The same process for generating fingerprints is repeated for any incoming query audio. These incoming fingerprints must then be matched against those in the database. This is done through a process called hash-alignment. When we originally populated the database, along with the individual fingerprints, each one is associated with a time offset from the start of its audio file (let's call this the database offset). This is also true for the query audio, but those fingerprints have offsets from the start of the query audio (the query offset). Dejavu aligns the hashes by reading all hashes and offsets from the database that have hashes equal to those we generated from the query audio, along with the audio name/id from the database. We then generate a list of pairs of audio names and database offsets minus the query offset. We can then return the matching audio name as the name in the most repetitive pair in the list.

2.1.2 Audio-Matching

As well as considering audio-fingerprinting as a possible approach, audio-matching also presented itself as a consideration. The idea behind audio-matching is to be a more general approach to information retrieval than fingerprinting which is aimed at identifying exact matches to audio it has already encountered. Audio-matching aims to retrieve those recordings from a collection that musically correspond to an audio query. To compare this approach to fingerprinting with an example, if the query was a performance of Beethoven's 5th symphony by the American composer, Leonard Bernstein, the fingerprinting approach would return that it was Bernstein's version of the piece. However, the matching approach would return matches to other composers/performers versions of the piece as well.

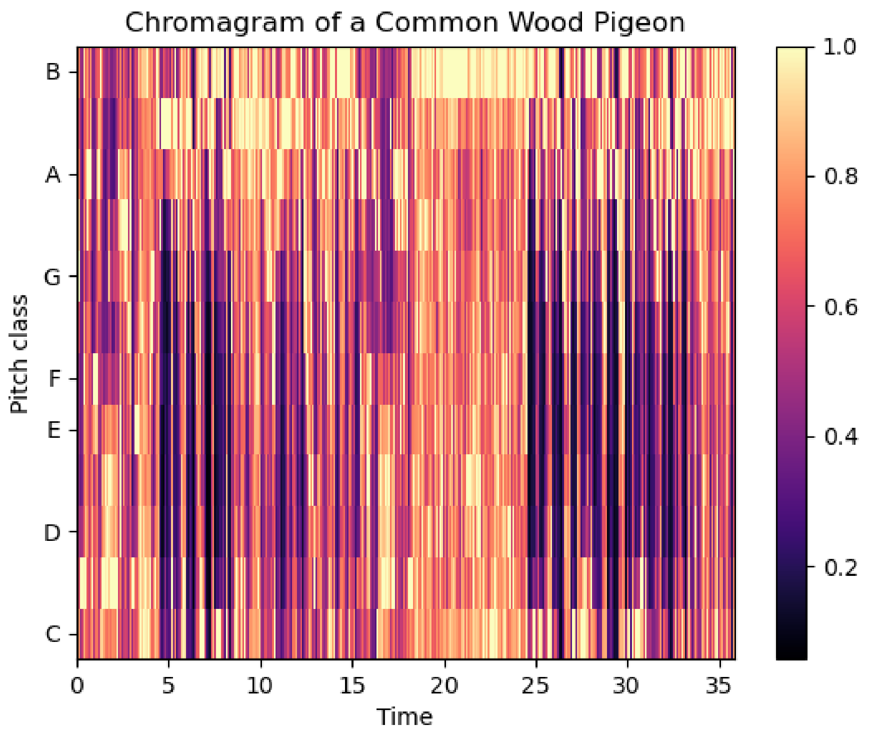


Fig. 2. Example of a chromagram of the same Wood Pigeon audio from fig. 1

This approach presented itself as a possible solution to solving the problem of classifying birdsong as it should be able to handle variations in the audio, as birdsong is stochastic [15] in nature (it is has no repeating pattern or rhythm structure and it is composed of short pieces of random sound). This is actually what makes it appealing to the human ear. Repetition in birdsong would eventually become irritating, but it changes very slightly, enough for it to be perceived as pleasing.

The aspect of audio-matching that allows it to generalise to a wider range of audio samples is the use of CENS (Chroma Energised Normalised Statistics) features. The idea behind using CENS is that similar pieces of music are based on the same basic note structure/material with deviations in characteristics such as tempo or articulations (this can denote how long or how softly a note should be played, i.e. Staccato, Legato, etc.). Chroma features, like CENS, indicate how much energy in a signal is distributed across the twelve pitch classes, or chroma bands, in western music, these being C, C#, D, D#, E, F, F#, G, G#, A, A# and B (the library used for fig. 2 shows, but does not label, the sharp (#) bands). By measuring these energy distributions over a given period of time, we can construct a chromagram (see fig. 2 - time on the x-axis, class on the y-axis and energy (relative to the other classes) represented by colour).

The concept behind CENS is that by calculating these energy distributions over longer periods of time, local deviations in, for example, tempo, as mentioned before, will be smoothed out giving us a set of features that will be similar to those from related pieces of music.

The first step in the process is to convert the query audio and database audio to CENS features. Following this, we find subsequences in the CENS features from the database that are similar to those from the query audio. This is done by computing a matching function using subsequence dynamic time warping [16]. Following this we identify local minima [17] in this matching function to compute the end timestamps of matching segments in the audio. We then apply a backtracking algorithm to compute the start timestamps for these segments. We can then quantify the degree to which the CENS features match by examining the amount of time between the start and end timestamps.

2.2 Convolutional Neural Network Approach

The final approach I examined during the methodology analysis stage, when considering the state of the art, was not related to musical applications but was more commonly used for speech-to-text methods. This represents a fundamentally different consideration of how to conceptually treat birdsong with this approach treating it more as individual speech (in the same sense that speech varies from person to person) rather than as examples of musical pieces.

This approach is centred on using a concept that has become fundamental to modern image processing, a convolutional neural network or CNN. CNNs are still relatively young in terms of the amount of research conducted in this space, especially around why they exhibit certain behaviour and how they can be specifically designed and tailored to certain applications or datasets. Most of the research in this area was conducted in the late 1990s and early 2000s but, due to hardware limitations at the time (lack of advanced GPUs when compared to those available today), the amount of research and use they garnered began to fall considerably.

However, they experienced a resurgence in the early 2010s (approx. 2012) when a CNN called AlexNet [18], designed by Alex Krizhevsky, achieved record setting accuracy in classifying images in the ImageNet [19] dataset (a dataset comprised of approximately 1 million images belonging to 1000 different classes - i.e. dogs, cats, cars, etc.). Since then, they have become widely adopted due to advancements in GPU design allowing them to be trained in a much more feasible time frame and have been applied to a wide variety of problems.

For our purposes, we will examine how we can utilise CNNs to classify audio recordings, knowing that they can successfully classify a wide variety of images. As an example of a state-of-the-art approach utilising CNNs to classify birdsong, I will refer to a submission to the 'LifeClef Bird Identification Task 2017', also referred to as BirdClef [20]. The specific submission I will refer to as a state-of-the-art approach is *'Audio Bird Classification with Inception-v4 extended with Time and Time-Frequency Attention Mechanisms'* by Antoine Sevilla and Hervé Glotin [21]. LifeClef is an annual competition, launched in 2003, under the Cross Language Evaluation Forum (CLEF) aimed to tackle a series of multimedia information retrieval tasks. BirdClef is one of those tasks, requiring competitors to submit papers detailing methods of birdsong classification. In the early years of the competition, competitors mainly focused on approaches similar to those detailed above (music focused). However, with the advent of CNNs and machine learning, those methods were quickly replaced by CNNs and the competition has since moved on to a more advanced task of identifying the species of birds that make up a soundscape, as opposed to just classifying the predominant species in the audio recording.

The consistent aspect of these different CNN approaches is that they all convert the audio to an image and, in the process, convert an audio classification problem into an image classification problem. The images that are generated are either standard spectrograms or variations on a spectrogram.

The aforementioned paper details how the first major step that is carried out is to convert the audio into three different log-spectrograms (a standard spectrogram with a logarithmic scale y-axis) using different window sizes (128, 512, 2048). Each of these is then treated as an individual channel in an RGB image (i.e. the 128-spectrogram will be treated as red values, and so on). Each of these channels are then aggregated into a single RGB image as figures 3 a), b), c) and d) show.

The image is then fed into an already trained version of InceptionV4 (a state-of-the-art CNN originally developed by Google) with some alterations made by Sevilla and Glotin.

The previously mentioned approaches regarding speech-to-text or word-to-text are commonly centred on a similar fundamental process to those in the BirdClef competition (audio to spectrogram conversion and CNN recognition).

The following chapters which detail the investigation and development as part of this project will describe, in more detail, how the CNN performs this image classification and the fundamental processes underlying this technology.

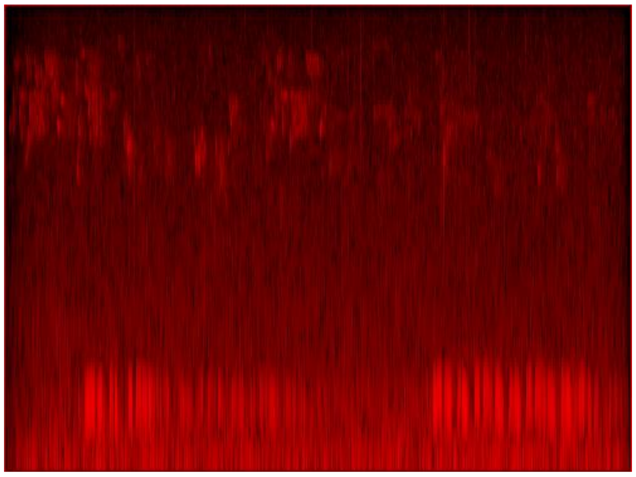


Fig. 3 a). Log-Spectrogram (128 sample window size) of the Common Wood Pigeon sample.

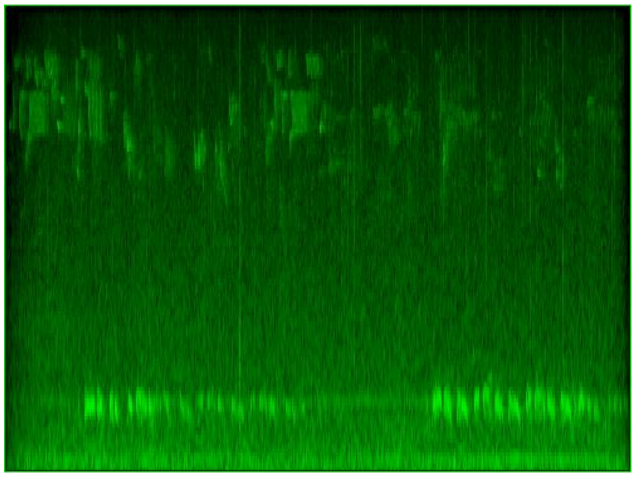


Fig. 3 b). Log-Spectrogram (512 sample window size) of the Common Wood Pigeon sample.

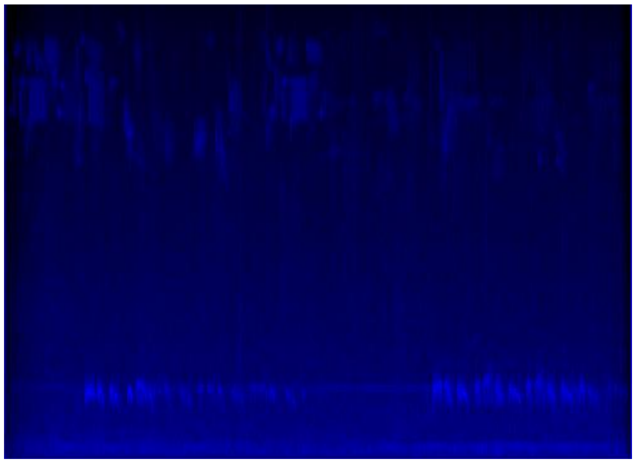


Fig. 3 c). Log-Spectrogram (2048 sample window size) of the Common Wood Pigeon sample.

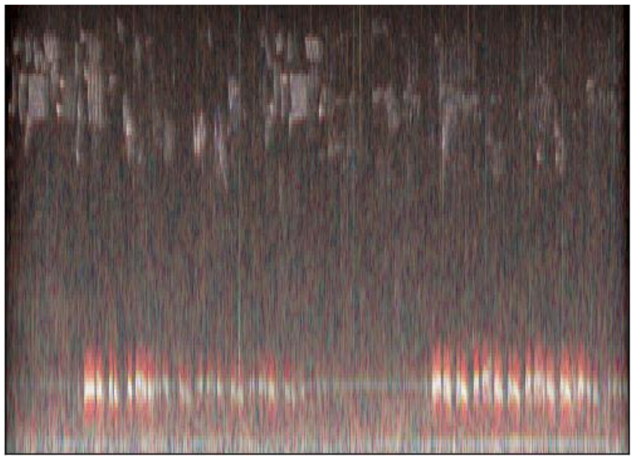


Fig. 3 d). Combined Log-Spectrogram of the three previous Common Wood Pigeon spectrograms as their individual colour channels (red, green and blue respectively).

3 Development & Investigation

This section will detail all development and methodology investigations carried out through the process of this project, from creating the dataset, to any preprocessing of the audio in that dataset (this turned out to be a very important aspect of the investigation), to investigating the different methods of birdsong recognition. It will be followed by sections discussing the implementation, results of this investigation and the conclusions that can be drawn from it.

3.1 Dataset Formation

In order to investigate any of the existing methods for birdsong recognition or those methods described in the state of the art when applied to this problem, we will need a collection of recordings of birdsong labelled with the correct species, in order to successfully test our methods and gauge their accuracy. Due to a lack of large datasets available online that meet this description and because I did not know, at the start of the investigation, how much data was required, I needed to generate the dataset.

This was done by relying on a citizen science project called Xeno-Canto and its website, xeno-canto.org. This is "*a website dedicated to sharing bird sounds from all over the world*". Xeno-Canto contains recordings submitted from a wide range of users, from research scientists to total amateurs and everyone in between. It also contains over 600,000 recordings from more than 10,000 species, totaling approximately 9,000 hours of audio recordings (although, some of these recordings have yet to be classified as Xeno-Canto also invites experienced users to attempt to classify unknown recordings).

It is very important to note that due to the fact that these recordings are submitted by a wide variety of users in a massive range of environments, and hence a wide variety of equipment is used to gather the recordings, the quality of these recordings can vary hugely. Some are surprisingly clear and some are extremely poor. As a result, when using this data as both training and validation data, we must reconcile with the fact that they may contain a lot of background noise, muffled audio, birdsong from species other than the main species in the recording (a soundscape), noise from animals other than birds, etc. To this end, a section will be devoted to the initial investigation of trying to clean the audio as much as possible (see section 3.2 'Dataset preprocessing methods'), although this decision was reversed when examining the CNN approach and developing the application, as will be discussed.

The dataset was gathered programmatically by scraping Xeno-Canto's API. The source code for this can be found in the source code zip/GitHub repository at *Tools/recordings_gatherer*. The API works by taking specific country names as parameters in the URL (these countries can be found in the *countries.txt* file). This results in a URL in the form of *https://www.xeno-canto.org/api/2/recordings?query=cnt:$Country*. This then returns a series of recordings with additional information (all responses are JSON formatted) denoting how many pages of recordings exist for that country (the first page is returned in this case). We then simply iterate over the total number of pages available for that country using URLs of the form https://www.xeno-canto.org/api/2/recordings?query=cnt:$Country&page=$Page_number. As part of this response, we receive a list of recordings redirecting to a link to download an mp3 file of the recording.

We then download the file and repeat the process for every recording in the list, for every page available for that country. We write the mp3 file and the returned json data to a sub-directory for the given species for that country.

As a forewarning, this script can take multiple days to fully gather all available data as the script waits for 3 seconds after every successive request to download a file, otherwise Xeno-Canto's servers will refuse connections from the users IP address as a rate limiting measure.

Due to time constraints while developing this project, the script was left to run through the night for approx. 2 days. This resulted in a dataset of slightly over 80 GB of data with 5262 species contained in the set. For practical purposes, when iterating over possible solutions to identifying birdsong, I will make use of a small subset of this entire dataset. Otherwise, the various methods would take too long to evaluate, especially considering the hardware limitations and that I do not have access to sufficient resources for a reasonable fee. However, with such a large dataset, it allows for a degree of engineering of the sub-dataset and allows us to compare wildly different bird species from across a huge geographical area (large biodiversity) as well as very similar species all present within a single country. In summation, while we will not make use of all of this data, it allows us to be extremely flexible in our investigation.

In terms of establishing the sub-dataset mentioned above, the script located in *Tools/gather_subdataset* can be used to create a restructured dataset by taking in a list of species and, optionally, a number beyond which no more recordings will be gathered if there are already that number in the sub-dataset. This will create a directory with a more useful structure, especially for using the tools in Tensorflow which will be discussed later, that will simply contain all the recordings for a certain species in a folder labelled with the species name.

3.2 Dataset preprocessing methods

In order to investigate the various methods in this section in any feasible time frame, a sub-dataset was generated using the recordings scraped from Xeno-Canto for the following six species: Atlantic Puffin, Eurasian Bittern, European Nightjar, Northern Raven, Red-Throated Loon and Tui. The reasons for choosing these species to generate the sub-dataset will be discussed later (from this point, I will refer to this sub-dataset simply as the dataset, unless otherwise explicitly stated). However, as long as all methods are compared against the same dataset, we should be able to accurately **compare them against each other** and the actual species in the dataset will not be important (however, as we will see later, it will affect the absolute accuracy of the chosen method).

In order to determine the performance of each method, particularly for the CNN method, when applied to real-world situations, the dataset will be split into a training dataset and a validation dataset. The purpose behind this is to produce reliable performance figures that you would expect to see in the real world. For this investigation, we will split the dataset randomly 80/20 into training/validation. This is a common split used in other machine learning applications and it would be a biased investigation if any method was given more training data (data to populate the database/train the model) and less validation data (previously unheard audio recordings that the method will attempt to successfully classify), or vice versa.

It is also important to note that the dataset was made to be evenly distributed across species in order to not place any emphasis on any particular species as it would lead to misleading results if, for example, the majority of training samples were from a Northern Raven.

To this end, the training dataset will have approximately 18 audio recordings per species while the validation dataset will contain approximately 5 audio recordings per species that we will attempt to classify.

It is also important to note that it is not the audio recordings themselves that are split 80/20, it is the number of audio recordings. For example, if we have 10 recordings, 8 of them will go towards training and 2 towards validation **regardless of the length of the audio files**. Therefore, we can end up in the situation where some of the recordings in the validation set are much longer or shorter than those in the training set and vice versa. However, this was a conscious decision as if we were to split the individual audio files up into sections we would see misleading performance figures if some sections from the same file were in both the training and validation set. Imagine we have a recording of some particular birdsong with some background ambient noise from its environment. If that recording was split in half and a person was played the first half, they could quite easily then match the second half due to, not necessarily the sound of the birdsong, but rather the sound of its environment (i.e. a busy road nearby, the sound of crashing waves, etc.). Therefore, by keeping the entire audio recordings whole then we should ideally be able to eliminate any classification of environmental noise rather than the birdsong itself while producing results that would reflect real world use.

This belief that by splitting the audio files themselves into sections, across training and validation, would lead to the successful classification of environmental noise was not simply a hypothesis. It was witnessed during the earlier phases of the investigation as it was the method I initially adopted to solve the problem of birdsong classification. To this end, section 4.1 ‘Audio Cleaning & Splitting’ will discuss the methods by which I initially pre-processed and, crucially, split up the audio recordings into chunks.

It is also worth noting that when generating the images for the CNN in pre-processing, it must be done in the same manner as the image is generated on the mobile device. To this end, the y-axis of the spectrogram denoting the frequency only goes up to 10 kHz (see section 4.1 and 4.3 for more detail on this). The images are also cropped to remove the additional information such as the axes labels, showing only the spectrogram itself. The reason for this is two-fold.

Firstly, it allows the maximum amount of information into the image that will be passed into the CNN and does not waste any pixels on a black background or axes labels that all images will share.

Secondly, in relation to the x-axis, it removes any likelihood that the CNN is relying on the x-axis (time) labels to help classify the image. This could occur if all recordings for a Common Blackbird were under 10 seconds in duration and all recordings for a Barn Swallow were over 10 seconds in duration, for example. The CNN could then easily differentiate between the two by extracting features from the x-axis, denoting time, such that if it recognised the duration was over 10 seconds then it could not possibly be a Common Blackbird. Clearly, this would result in unwanted behaviour as the recordings from users in the real-world could differ widely in duration. The code for the dataset preprocessing can be found in the *Preprocess.ipynb* file.

3.2.1 Repeating Pattern Extraction Technique (REPET) [25]

Another method that was examined when I was investigating the possible methods of cleaning the audio recordings was utilising a technique originally developed by Northwestern University, Illinois, US for separating the non-repeating vocals from the repeating backing musical instruments in music and distributed through the Nussl Python package.

The idea behind repurposing this technology was to examine whether this technique could utilise the stochastic nature of birdsong (birdsong is composed of short pieces of random sound) and treat it as the non-repeating foreground, isolating it from the repetitious background noise.

This process works by identifying repeating structures in a spectrogram and then isolating those repeated segments from the non-repeating segments. You can see the full details in the relevant paper and in the library documentation (see references).

The below audio clips illustrate how this process affects the audio recordings. Below is an unaltered recording of a Northern Raven, the isolated background from the recording and the foreground from the recording (the birdsong)[1].

Unaltered *Background* *Foreground*

Fig. 4 also provides a visual representation of the process, labelling the foreground and background as a spectrogram.

As you can see from fig. 4 and hear from the audio clips, this method excels when dealing with repetitive patterns, as you would expect. You can see and hear this from its ability to recognise the repeating white noise (occurs at approximately the same amplitude across the entire frequency spectrum), shown in orange. However, in the real world, the background audio of the birdsong recordings will not necessarily display this highly repetitive nature. You can see an example of this just before the six second mark in the audio recordings above when we capture what sounds like some wind noise or accidental vibration against the microphone (a short burst below 512Hz). Needless to say, even visually from the spectrogram, we can determine that this is not part of the birdsong as it has no resemblance to the other areas of the spectrogram in which

[1] **Note:** In order to play the embedded audio, a supporting PDF reader must be used (i.e. Acrobat Reader DC).

the calls from the Raven are self-evident. However, because it does not repeat itself through the audio and it only occurs once before the six second mark, the REPET algorithm cannot isolate it from the foreground and it is left untouched. This can apply to any other non-repeating sounds in an audio clip, such as a single car horn.

While this method has its uses in terms of removing white noise (which can simply be done using tools like FFmpeg) and repeating audio patterns (which aren't likely to occur in real-world environments), it proved to not be very useful for the purposes of isolating birdsong. As a result, a modification of the algorithm may prove to be useful for the purposes of this project but, in its current state, it is not.

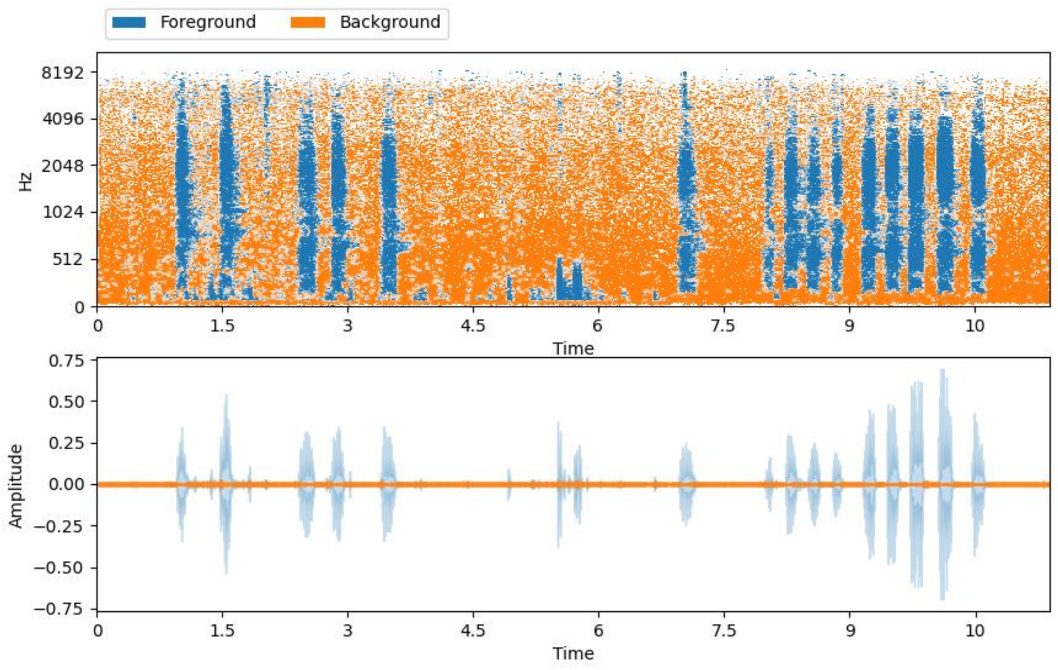


Fig. 4. Visual representation of the repeating pattern extraction technique on a Northern Raven sample.

3.2.2 Audiomentations

An entirely different approach to those taken above, aimed at cleaning the audio and removing unwanted noise/distortion, was also examined. This involved taking the opposite approach by purposely corrupting the audio with different forms of noise and distortion using a Python package called Audiomentations.

The idea behind this approach to dataset preprocessing is that if the approaches considered cannot learn to recognise birdsong successfully given a cleaned piece of audio, then can we use processes that can learn to 'ignore the noise' and then learn to recognise the birdsong from here? This approach to preprocessing the audio is intended for the CNN approach as this is effectively another form of data augmentation (a common process in machine learning that uses the already gathered data to generate more training data that is derived from the originals).

This functionality from the Audiomentations library was also combined with another self-implemented form of data augmentation in which the **training data recordings** were split up with a degree of randomness into a set of additional recordings. This involved taking a piece of training audio and making another random number of recordings between a minimum (set as 3) and maximum number (set as 7). The additional recordings also started from a random point in the original recording and had a duration of a random length between a minimum length (set as 15 seconds) and the maximum length possible given the starting time. As a result, the training data set contains the original recording as well as a set of recordings derived from it before any distortion, etc. was added (this prevents the issue mentioned in section 3.2 in which sub-recordings, shorter pieces of audio taken from a longer piece, from the same original recording were present in both the training and validation set which led to the issue of classifying environmental noise leading to a misleadingly high accuracy). Using the Audiomentations process, the changes in audio will be restricted to the training set only.

The idea behind generating sub-recordings of varying length was that the CNN could then learn to recognise birdsong regardless of the length of the audio, and hence the length of the x-axis in the spectrogram it was attempting to classify. This was an important decision as the images that are fed into the CNN must all be resized into the same dimensions (typically somewhere in the area of 224x224 pixels). Intuitively, if the spectrograms used in training were all for pieces of audio that were 15 seconds long and the CNN then tried to classify a piece of audio that was 60 seconds long, you can imagine how it would possibly struggle as there is essentially 4 times the amount of information in the same sized image space. This would result in the peaks and troughs that make the birdsong identifiable being more condensed and squashed together than any image in the training set.

The reason behind choosing a minimum duration of 15 seconds, as mentioned above, was due to the fact that if we had a recording that was, for example, 5 seconds long, the number of recordings that could be derived from it that did not show obvious repetition between each other while containing enough birdsong audio to serve as adequate training data, was limited. Hence, 15 seconds was chosen as a minimum duration as it would be long enough to capture the general audio behaviour of most species (i.e. any similar sounds that may repeat, the occasional signature chirp of a certain bird, etc.) while not being so short that we could not then augment our dataset. The reason behind choosing a random number of sub-recordings (between a minimum and maximum) was similar, in that the aim was to establish as much diversity as possible in the training dataset without making the dataset so large that it was impractical to train on due to time-constraints and not making enough sub-recordings that we were failing to sufficiently augment the training dataset.

Having augmented the training dataset with a series of sub-recordings, we then apply a series of audio operations provided by the Audiomentations library to each recording in the training dataset, both original recordings and sub-recordings. As a result, the training dataset will contain, for example, a file called Northern_Raven_1.wav or something similar. It will then also contain a file with the same audio that has had Gaussian noise added to it and another file that has clipping distortion added, and so on (i.e. the processes are applied individually to the same source audio, not one after another).

For a more detailed list regarding the implementation and specific details of the operations performed, see section 4.2 ‘Audiomentations Operations’.

3.3 Audio-Fingerprinting

Given the description of the audio-fingerprinting functionality provided in section 2.1.1, the audio-fingerprinting was implemented as follows.

Given a training and validation dataset created using a method of preprocessing described above, the implementation begins by cloning the Dejavu GitHub repository. This requires a MySQL server running locally (or in the Cloud, but the provided code uses a local server) with a database of a name of your choosing (the provided code assumes the database is called *birdsong* - specified in the *config* variable). If this database does not exist, it will need to be created manually. However, the tables within the database will be created by Dejavu, so it can be made as an empty database.

This then yields two tables within the database as described in the library documentation, one called *fingerprints* and another called *songs*. Dejavu is then used to fingerprint the entire training data directory. Following this, the validation dataset is iterated over and Dejavu is then used to recognise the validation audio as the provided FileRecognizer class can be used to recognise audio directly from files. We then sort the returned results by the confidence/matching scores, in decreasing order, and take the first one. Given that all audio files in our dataset follow the same naming convention, we can then simply check that the filename of the returned match contains the species name given by the sub-directory that the validation audio is placed in. This process continues and the final accuracy of this method can easily be calculated by keeping a running total of those validation audio files correctly matched versus the total number of validation audio files. See the provided code in *Test_Dejavu/dejavu/birdsong.py*.

3.4 Audio-Matching

The process of audio-matching is as described in section 2.1.2, but repurposed on our training and validation dataset. This code can be found in the Google Colab notebook / .ipynb file, *FMP_Birdsong.ipynb*.

This code follows a similar structure to that of the audio-fingerprinting. It is done by building a list of CENS features from the list of all training audio files. We then iterate over every file in the validation dataset, generate CENS features from these files and use the provided matching functionality given by the function *compute_plot_matching_function_DTW* that uses dynamic-time warping as discussed in section 2.1.2 by individually providing the CENS features from the training CENS list.

This then returns a list of start-time / end-time pairs of periods that match between the current validation CENS and training CENS. We simply take our matching file as the training file which provides the longest total duration of matching periods between the audio. We then make use of the naming convention as mentioned before in section 3.2 and 3.3 to keep a running sum of those matches that were correct. This allows us to quantify the methods classification accuracy.

3.5 Convolutional Neural Network (VGG19) [28]

This section will discuss in more detail how the classification of birdsong was implemented using a convolutional neural network.

As mentioned in section 2.2, when trying to choose between a top-of-the-line neural network architecture, it is best to look at those that have achieved the highest performance in classifying the ImageNet dataset. As a result, the main three CNN architectures worth considering are variations on VGG (Visual Geometry Group), ResNet (Residual Neural Network) and Inception (originally developed by Google).

Three CNN architectures based on those described above were examined when choosing a CNN to continue working with. These were InceptionV3, VGG19 and ResNet50 and are all freely and easily accessible through Tensorflow. Tensorflow is an open-source machine learning library available in a range of programming languages, although it is most commonly used in Python as that is the most popular language in the field of data analysis (we will use the Python package as this is the language used in Google Colab). In particular, Tensorflow places an emphasis on the training and inference (a conclusion reached on the basis of evidence - i.e. an input image) of deep neural networks. To see how these CNNs can be trained and deployed in relation to our needs, examine the Google Colab notebook, *BirdCall_Colab_VGG19.ipynb*.

The final CNN architecture that was chosen from the three mentioned above was VGG19. The reason for this was that it was slightly more accurate when classifying any of our validation datasets than both InceptionV3 and ResNet50. Although, the performance gains were marginal (the validation accuracy was typically in the range of 5% higher when using VGG19). However, an important lesson was learned when examining the CNN approach to classifying birdsong. That is that any choice of CNN from those mentioned above or any similarly designed neural network outperforms the audio-fingerprinting and audio-matching methods drastically, as will be discussed in the results section (section 5).

In order to adequately explain how this CNN approach works, I will discuss the basic concepts of how a neural network can be used to classify images (spectrograms in our case) and the specific underlying methods and layers that make up VGG19. I will then discuss and refer to the Google Colab notebook to illustrate how the network was trained using our dataset allowing us to classify previously unseen spectrograms.

3.5.1 CNN Basics & VGG19 [29]

A convolutional neural network is made up of an input and an output layer with multiple hidden layers between them. There can be over 100 hidden layers in some neural network architectures, hence the “deep” in deep neural networks. A crucial component of CNNs is that each layer takes the output of the previous layer as its input. There are also multiple different types of layers such as convolutional, pooling, fully connected, etc. and some of these will be discussed later in relation to VGG19.

The convolutional layer is one of the most important layers to consider when discussing CNNs. The basic premise in a convolutional layer is that they make use of kernels/filters. At their basic level, these kernels are masks that are applied over the input to the current layer. Take for

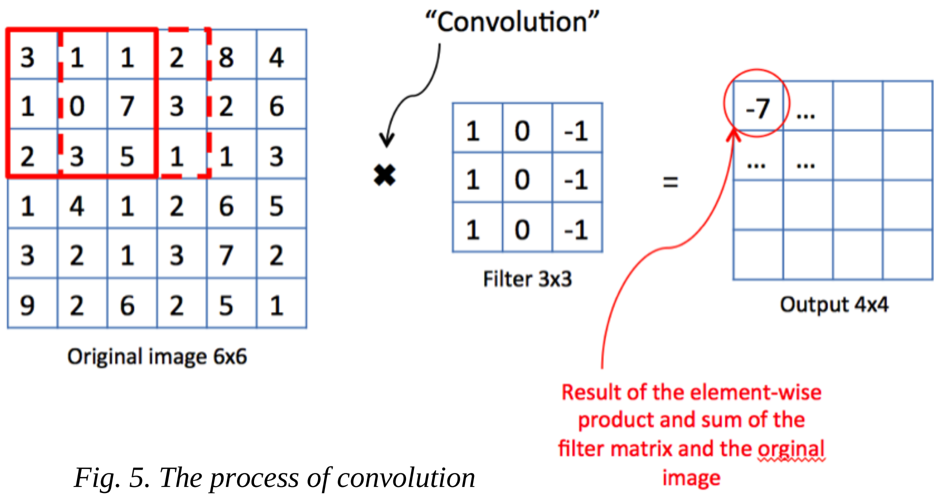


Fig. 5. The process of convolution

example, a grayscale image. We know that a grayscale image is simply a 2-dimensional array, also referred to as a matrix, with values in each cell that denote the pixel intensity (i.e. a value of 0 is totally black and a value of 255 (the maximum value that can be denoted by a byte) is completely white). We can then *convolve* over this matrix using our kernel to produce a certain output matrix. Fig. 5 illustrates the process of convolution.

The result of -7 is given by the following expression, showing the convolution operation (typically denoted by a * in CNN literature, not to be confused with matrix multiplication):

$$(3 \times 1) + (1 \times 0) + (1 \times -1) + (1 \times 1) + (0 \times 0) + (7 \times -1) + (2 \times 1) + (3 \times 0) + (5 \times -1) = -7$$

Fig. 5 also illustrates how the kernel is overlaid on top of the input (given by the solid red line) and then moved over by one space (shown by the dotted red line) with the process repeating, giving the result for the cell to the right of the previous one. This is then repeated downwards throughout the image. As you can see in fig. 5, given our original image, the output has been downsized from 6x6 to 4x4 as the kernel works from top-left to bottom-right, up until the point at which it can go no further as the kernel would stretch beyond the bounds of the image. This is where we can apply padding to prevent downsizing.

Padding is simply done by adding extra rows and columns of 0s around the input image in order to extend the bounds of the image allowing the kernel to fully convolve over all the values that were present in the original image without downsizing the output matrix. Convolutions can also be ‘strided’, that is, rather than moving the kernel over by one each time, the *stride* can be increased to, for example, 2 or 3.Intuively this will decrease the size of the output matrix as we are discarding a certain level of data from the original image.

On a multi-channel image (i.e. a colour image - three channels for red, green and blue), we define a stack of 3 kernels, one for each channel, and convolve over each of these separately. As a result, the output from this process will be 3 separate matrices. In order to obtain a single output matrix, we simply add the corresponding value in each matrix together, i.e. the value at (1, 1) in matrix one is added to the value at (1, 1) in matrix two and so on. The final result is then the value at (1, 1) in the resulting output matrix. (Typically in the CNN literature, the stack of kernels is assumed, such that if a kernel is defined as 3x3, it is actually a set of three 3x3 kernels).

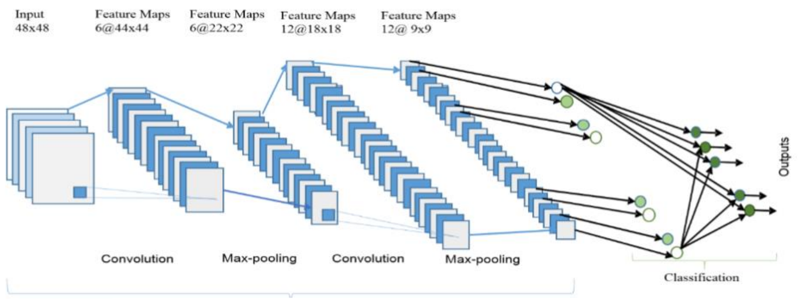


Fig. 6. Illustration of hidden layers in a CNN

We can also apply multiple different kernels to the same input and stack the final outputs together. This is useful because some kernels may be specifically tailored for identifying a certain feature in an image (i.e. the presence of fur on an animal - this is an oversimplification, but it should illustrate the idea) while other kernels are aimed at identifying totally different features. The fact that we are using multiple sets of kernels on our input space, results in the following kind of behaviour shown in fig. 6, in which we end up with an ever increasing array of input images into the following layers.

The crucial aspect of these kernels is their ability to extract features. Take, for example, the

1	0	-1
1	0	-1
1	0	-1

kernel shown here with a collection of 1’s, 0’s and -1’s. This kernel can be used to extract vertical edges in an image. By extension, we should be able to learn certain values to place in our kernels that can extract certain features relevant to our task (i.e. identify certain peaks in our spectrograms). As we continue through our network, as shown above, our kernels perform this feature extraction on the features previously extracted by layers before, such that we are finding features of features. This is an important aspect to highlight in that by learning which weights to place in our kernels in order to successfully classify an image, we cannot know for certain which kernels relate to which specific feature (i.e. we cannot look at a kernel and conclude that it is for identifying a certain signature peak in the call of a Common Chaffinch). The obvious question to ask next is “how are these kernel weights learned?”. This is done through a process called back-propagation (given a spectrogram and knowing that is a Chaffinch, what changes need to be made to the weights to correctly classify it) and using algorithms such as gradient descent. However, this is beyond the scope of this project and paper as it is not a case study in the underlying processes of convolutional neural networks but is more focused on the **applications** of CNNs.

In order to obtain a useful piece of data at the end of our network, we cannot simply propagate matrices that are the same size as our original input image as it is of no use to have a vast array of 224x224 matrices when we do not know what they signify. Therefore, we must introduce an element of downsizing. In VGG19, this is where max-pooling layers come in. A max-pooling layer is relatively simple. It simply takes in an input matrix and, using a certain block of a given size, will extract the largest value from its block and write that value to the output matrix. See fig. 7 for a max-pooling layer with a block size of 2x2.

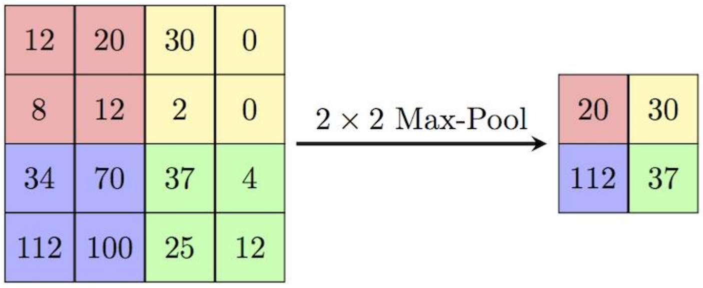


Fig. 7. 2x2 Max-pooling layer

The final layer to discuss in relation to VGG19 is a fully-connected layer. This layer takes, as its input, a matrix and flattens it into a 1-dimensional array. The output is then a function of a weighted sum (these weights must also be learned) of the flattened array. This function is non-linear, which allows the network to model more complex, non-linear behaviour. A common choice for this function is the softmax function [30]. Again, to discuss this further would require delving into the finer details of neural network design and behaviour. The important takeaway from this is that the softmax function returns a confidence score for its corresponding class (bird species) given an input matrix.

The below image illustrates the architecture of VGG19.

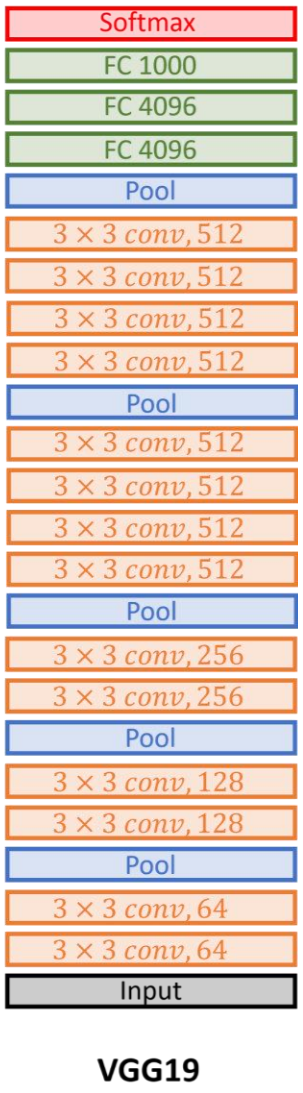


Fig. 8. VGG19 neural network architecture

In fig. 8, you can see that, following the input image, we have two convolutional layers each with 64 3x3 kernels (remember that 3x3 kernels are actually a stack of three 3x3 kernels). This is then followed by a max-pooling layer to downsize the matrices and the process continues with two more layers each with 128 3x3 kernels, and so on. The important point to get across here, is the sheer number of kernel weights in the network. Modern CNNs can very easily have millions of kernel weights that must be learned in order to classify an image. When CNNs are being designed, the kernel weights are typically initialised randomly across the network and are then learned through training to a more suitable set of values. This is why CNNs notoriously require such high performance hardware to train the network (especially when dealing with deeper networks and larger dataset) and also take such a long time to train (this was also why Google Colab was chosen as it provides access to high performance servers with much more RAM and performant GPUs than I would have had access to otherwise). This leads into the concept of transfer learning.

The concept of transfer learning is what made this project possible. The process is summarised as re-training a CNN, which has already been trained on a certain dataset, with a new dataset. In relation to classifying birdsong, this was done by taking the VGG19 CNN that has already been trained on the ImageNet dataset. The concept is that, as the CNN now has the weights in its kernels to classify a wide variety of images and has proven to be very successful at addressing the task of image classification, we can now alter those weights slightly to address a similar task (in our case, classifying spectrograms). Given that we are not starting from a set of random weights, we should require less time to train the model to achieve a suitable level of accuracy given that the weights are already suited for image classification.

See section 4.3 for a discussion of the CNN in relation to its implementation for this project and the process of training the network.

4 Implementation

This section will provide the implementation details regarding how the audio dataset was initially pre-processed and the audio was split into smaller recordings as well as how the audio dataset was augmented. This section will also discuss the process of training the CNN and implementing the network in the Android application.

4.1 Audio Cleaning & Splitting

The following sections describe the step-by-step methods that were initially used to pre-process the audio. These processes were all performed using FFmpeg.

1. Loudness Normalisation

This FFmpeg filter allows the audio recordings to be normalised with a certain decibel (a logarithmic unit used to measure the sound level) range according to the EBU R 128 standard. The default parameters in the FFmpeg tool were used for this and normalised the audio to the following specifications:

- The target average decibel value that the normalised audio will have = -24 LUFS
- The final loudness range of the audio after normalisation = 20 LUFS
- The maximum true peak value of the final audio = -2 LUFS

LUFS in an abbreviation for Loudness Units Full Scale. 1 LUFS can be understood as 1 dB.

The reasoning behind normalising the audio will become apparent after the following steps are described. The below command is used to normalise the audio

```
ffmpeg -i "incoming.wav" -af loudnorm "normalised.wav”
```

Code Listing. 1. Loudness normalisation using FFmpeg

2. High-pass and Low-pass filter

A high-pass filter allows us to attenuate all components of an audio signal below a certain frequency, letting high frequencies ‘pass through’. Originally this was an electronic filter used in digital circuitry but can be applied to any waveform signal.

A low pass filter has the opposite effect, cutting off frequency components above a certain frequency value letting lower frequency components pass through.

By applying some domain knowledge in the area of ornithology, we should be able to make it easier to classify audio recordings by ignoring the frequency components that we know cannot be produced by a bird. To this end, we know that the species that produces birdsong with the lowest frequency component is a Southern Cassowary, a large flightless bird with black feathering native to Australia that has been recorded producing audio signals as low as 24 Hz [22] which is at the lower extreme of the human hearing

Fig. 9. Southern Cassowary

Fig. 10. Prothonotary Warbler

range (humans can detect frequencies from about 20 Hz to 20 kHz [23]).

On the other extreme end, the highest frequency species can produce sounds that sit around 8000Hz (8 kHz) [24] and slightly above. These can include different varieties of warblers, sparrows, waxwings, etc.

The below command is used to apply a high-pass and low-pass filter at certain frequency levels (high-pass at 20Hz and low-pass at 9500Hz). These values were chosen as they are close to those maximum and minimum frequencies mentioned above while allowing for some margin.

```
ffmpeg -i "birdsong.wav" -af "highpass=f=20, lowpass=f=9500"
"filtered.wav"
```

Code Listing. 2. High-pass & Low-pass filters using FFmpeg

3. White Noise removal

This step is simply to try and improve the audio quality of the recordings by reducing any white noise in the recordings. This is done using the Fast Fourier Transform and, by default, will reduce the noise by 12 dB. The below command is used to de-noise the audio.

```
ffmpeg -i "noisy.wav" -af afftdn "denoise.wav"
```

Code Listing. 3. White noise removal using FFmpeg

4. Silence Removal

The next step is to remove periods of silence before splitting the audio into chunks in order to avoid having pieces of audio with no occurrence of birdsong in them as this should be impossible to classify (assuming our method doesn't inadvertently classify environmental noise).

This is why the loudness normalisation is a crucial step and must be done before this step. This is because we must use a hardcoded decibel value for which any quieter sounds will be treated as silence. Therefore, if we did not normalise the audio files to the same decibel levels then we would be treating birdsong as silence in some cases assuming the bird was far away enough from the recording device that it was considered quiet enough to be silent.

The decibel value was chosen manually by listening to a series of normalised audio files of birdsong recordings from the dataset and choosing a value that was sufficient to remove periods of silence but not remove **any** of the birdsong. This was chosen as -46 dB. However, we do not want to remove all periods of silence as that would eliminate some of the pacing and rest intervals that are unique to some bird species. However, as we are generating 3 second chunks, silence that occurs for longer than a period of 1 second was selected to be removed. Fair questions can arise from all this, those being, why 3 seconds or why 1 second, etc.? As mentioned, this process of cleaning the audio was only examined during the initial phase and, as it turned out, was not beneficial to the results of our methods so it was not required to attempt to find values for these hyper-parameters that would maximise the accuracy, as an alternative approach out-performed this preprocessing approach.

Also, the choice to normalise the audio was two-fold. Firstly, for the reason mentioned above. Secondly, because by normalising the audio, if our process is capable of classifying birdsong, it should eliminate the need to have recordings of the same species at different distances (and hence volume levels) from the recording device, i.e. it should not have to learn what a Common Pigeon sounds like when it is up close, in the middle distance and when it is far away. It should simply have to recognise a Common Pigeon always with the same decibel range.

The below command is used to remove the periods of silence as mentioned.

```
ffmpeg -i "cleaned.wav" -af silenceremove=stop_periods=-
1:stop_duration=1:stop_threshold=-46dB "no_silence.wav"
```

Code Listing. 4. Silence removal using FFmpeg

5. Audio Splitting

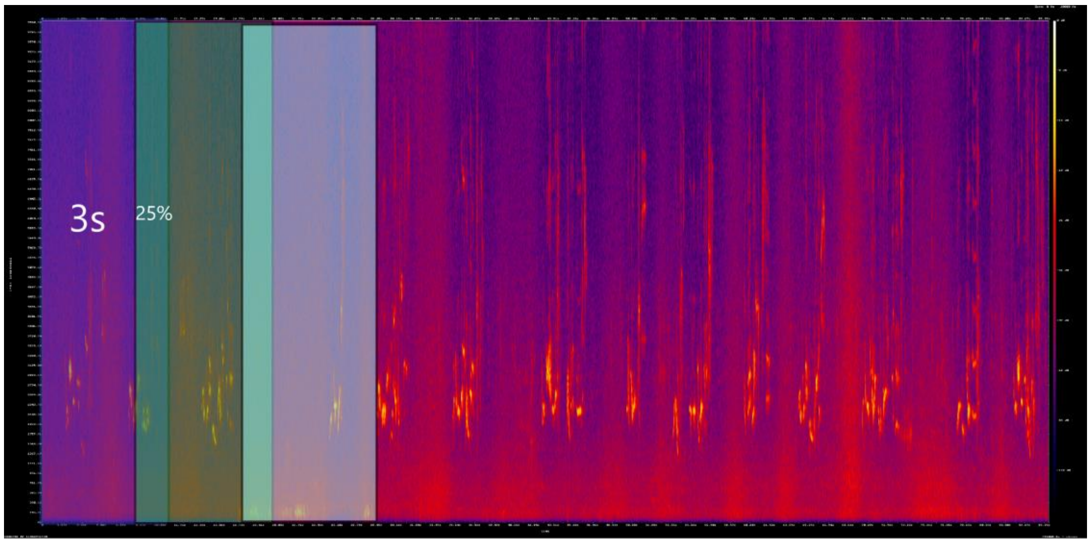


Fig. 11. Illustration of audio splitting with overlap

The next step is to take the cleaned audio and to split it up into chunks. This was initially done by splitting up the cleaned recordings into 3 second chunks with a 25% overlap. Fig. 11 illustrates how this works with the coloured blocks representing 3 second chunks and how they overlap with each other. This is repeated across the whole original audio file. If the final chunk cannot be made into a 3 second audio clip, then it is ignored.

4.2 Audiomentations Operations

This section will provide specific implementation details regarding the operations that were performed using the Audiomentations library in order to augment the audio dataset.

1. Gaussian Noise

This operation simply adds a level of Gaussian Noise to the audio with parameters that can be specified for the minimum and maximum amplitude of the noise, given by the AddGaussianNoise class.

Gaussian Noise is statistical noise that has a probability density function equal to the normal distribution (also known as a Gaussian distribution, named after German mathematician and physicist Carl Freidrich Gauss, 1777-1855).

A probability density function defines a probability distribution for a discrete random variable and can give the probability that a random variable will fall between a certain range of values by calculating the area under the curve between those values (the integral). The probability density function of a Gaussian variable x is given as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean and σ is the standard deviation. You can see the effect visually in fig. 12 a) and fig. 12 b) (the original audio in figure a, the audio with added noise in figure b).

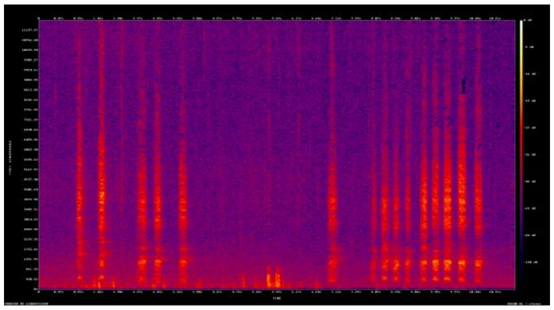


Fig. 12 a). Unedited Spectrogram

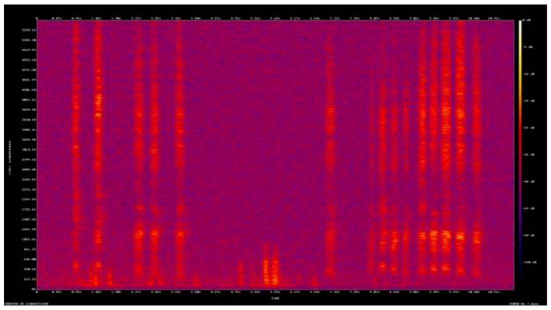


Fig. 12 b). Spectrogram with Gaussian Noise

2. Time Shift

This operation is denoted by the *Shift* class in Audiomentations. This operation is relatively self-descriptive as it simply randomly shifts the start time of the audio to another point in the recordings (adjusting the end time accordingly). The default behaviour was used that enables rollover (that is, if the start time is moved to just before the original end point, then it ‘rolls over’, wrapping around back to the original start point and continuing from there).

It is worth noting that, with the rollover option enabled, the resulting audio is of the same duration as the input audio (this is desired - see section 3.2.2 discussing the length of augmented recordings).

3. Background Noise

This operation is provided by the AddBackgroundNoise class and relies on a collection of audio files containing samples of background noise. The files used can be found here [26] and here [27].

This operation mixes the background noise into the input signal. This is an especially useful operation as it is able to replicate birdsong in the presence of, for example, passing cars which is particularly useful for training a CNN to classify audio in a wide variety of real world environments.

For this operation, we must specify a directory containing the downloaded background noise samples. In the case that these background samples have a duration shorter than the duration of the input signal, then the background noise will be repeated throughout the signal.

The volume of the background noise to be added is relative to the input signal. This ensures that the background noise does not drown out certain pieces of audio.

This operation defines a minimum SNR (Signal to Noise Ratio) and a maximum SNR in dB. The Signal to Noise Ratio is the ratio of the power of the desired signal to the power of the noise (such that a higher SNR is more desirable while a low SNR will result in poor quality, noisy audio). The SNR can be expressed in decibels as follows using the expression SNR = S - N, where S is the power of the desired signal and N is the power of the noise. As an example, you measure a radio signal with a strength of -10 dB and a noise signal of -50 dB. The resulting SNR is 40dB (i.e. the strength of the desired signal is 40dB higher than that of the noise - hence a low SNR in dB is less desirable).

This operation chooses a random SNR value between the minimum and maximum (by default this is 3dB and 30dB respectively). The choice of background noise sample is also chosen randomly. The below audio files illustrate the addition of street traffic to the call of a Northern Raven.

Original

With added noise

4. **Background Short Noise**

This operation is essentially the same as that described above except for the fact that instead of adding the noise sample to the entire input audio, it mixes in various bursts of overlapping sounds with random pauses in between bursts and is provided by the AddShortNoises class.

The default minimum and maximum SNR dB values are also slightly different with 0dB and 24dB respectively instead of 3dB and 30dB.

5. **Clipping Distortion**

This functionality is provided by the ClippingDistortion class. In electronics, clipping distortion occurs when an amplifier delivers an output voltage or current beyond its maximum capacity. This results in the waveform being ‘clipped’ shown by fig. 13 with values above and below a certain range being cut off resulting in a more square-like waveform. The process is similar for audio in this operation where a minimum percentile and a maximum percentile are specified as arguments. A random value is chosen between these arguments resulting in the percentage of points that will be clipped. If, for example, the value chosen was 40% then the points are clipped if they are below the 20th or above the 80th amplitude percentile.

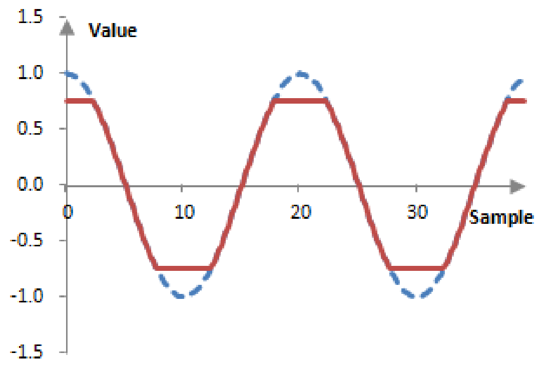


Fig. 13. Effect of clipping distortion

The values chosen for these min and max arguments were the default values of 0% and 40% respectively. The audio clips below exhibit the effects of clipping distortion with a value of 50% (clipping those below the 25th and above the 75th percentile).

Original

With Distortion

6. Gain Adjustment

This operation is provided by the Gain class and multiplies the input signal by a random amplitude factor, thereby increasing or decreasing the volume of the audio.

The purpose of this augmentation process is somewhat related to the loudness normalization discussed in section 4.1. The idea being that by generating training samples of varying volume levels that the model could be trained to become invariant to the distance at which the song was recorded from the actual bird, such that the model has seen what an Arctic Warbler sounds like close up as well as far away.

As per the previous operations, the gain levels are specified through a minimum and maximum gain parameter after which a random value is chosen between the min and max, in decibels. The minimum is set as -12dB (reduce the volume) and the maximum is set as 12dB (increase the volume).

7. Mp3 Compression

Finally, this operation uses an mp3 encoder to compress the input audio file, thereby reducing the audio quality, as mp3 files are compressed using lossy compression (uses approximations and partial data discarding to reduce the file size).

This operation chooses a random bitrate for the mp3 compression from a list of supported bitrates, ranging from 8 Kbps (Kilobits Per Second) to 64 Kbps, although this can be increased to 320 Kbps if desired (intuitively, the higher the bitrate, the more bits are encoded per second of audio resulting in a higher quality audio compression with a larger file size, and vice versa for a lower bit rate).

The idea behind this augmentation process is to train the model on audio samples of poor quality that could result from lower quality hardware in the recording device, such as microphones in older mobile devices or less expensive phones in order to make the application more versatile and applicable to a wider range of real-world users.

The below audio clips demonstrate the mp3 compression process with a bitrate of 16 Kbps.

Original

Compressed

The Audiomentations library does offer more audio processing capabilities such as shifting the pitch of the audio or stretching out the duration of the audio. However, these were not selected as operations for augmenting the training dataset as they would fundamentally change the birdsong to a point at which it would no longer resemble the original birdsong (such as if you shifted the pitch of a certain species higher or lower, it may then sound like another species).

The purpose of these chosen operations is not to alter the behaviour of the birdsong itself, but to apply real-world effects that could degrade the quality of the user's recordings.

4.3 CNN Training

The following section will reference the Google Colab notebook *BirdCall_Colab_VGG19.ipynb* to discuss the process of training the VGG19 network using transfer learning.

The first step involved in using Google Colab is to mount your Google Drive account to the Colab server and install your dependencies (see the section titled *Mount Google Drive Account* in the notebook). Following this, Tensorflow must be installed. In the section *Install Tensorflow*, the random seeds are set in order to ensure the most repeatability as possible when training the model and gathering accuracy statistics.

The notebook assumes that the training and validation image dataset can be found in a folder called FYP in your Google Drive (*FYP/Consistent_dataset/train* and *FYP/Consistent_dataset/validation* respectively). Note that the audio file dataset can be converted into an image dataset for the CNN by using the below FFmpeg command on each audio file

```
ffmpeg -i "audio.wav" -lavfi showspectrumpic=stop=10000 "spectrogram.png"
```

Code Listing. 5. Spectrogram generation with FFmpeg

Note that the maximum frequency displayed on the y-axis is given by the *stop* parameter set at 10,000 as we know that 10 kHz is the absolute top of the frequency range that birdsong can reach.

In the *Generate training and validation data* section of the *BirdCall_Colab_VGG19.ipynb* notebook, the image size is set as 224x224 pixels as this is the expected input image size into the VGG19 network and is generally the standard image input size you will see in most cutting edge CNNs. The batch size is also set as 32.

The batch size defines the number of images used to train the network in batches. In this case, the CNN will be trained on the first 32 images and adjust its kernel weights. It will then take the next 32 images and repeat the process for every image in the training dataset. The batch size controls the accuracy of the estimate of the error gradient when training a neural network and, in turn, a lower batch size will provide a less accurate estimate and can lead to unstable training that negatively affects the result. In turn, we would ideally like to use a large batch size. However, each image in the batch must be loaded in-memory and fed into the CNN rather than being read from the disk. As a result, for a large batch size, we need a large amount of RAM. If training this model locally or using a standard Google Colab account, you may need to lower this batch size (the investigation of this project was done with a Google Colab Pro account, providing access to higher RAM servers).

After establishing the training and validation dataset, the VGG19 network is downloaded in the *Create Model* section. This is where we must make some alteration to the network to meet our requirements. The VGG19 model that is downloaded is trained on the ImageNet dataset which has approximately 1000 classes. Therefore, the model currently will generate predictions for those 1000 classes. This needs to be changed to predict the number of species that we have in our training dataset.

This is done by setting the *include_top* parameter to *False*. This removes the fully connected layer at the end of the network which provides the confidence values for the classes. As a result we must now define our own final layers to generate predictions for the number of species we are dealing with. This is done with the following snippet:

```python
model = tf.keras.Sequential([
  base_model,
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.GlobalAveragePooling2D(),
  tf.keras.layers.Dense(number_of_classes, activation='softmax')
])
```

Code Listing. 6. Final CNN definition using VGG19 base model, in Python

This defines a model as the VGG19 network with the fully-connected layer removed, followed by another convolutional layer, a dropout layer [31] (this is a method of combating overfitting - see terminology), followed by an average pooling layer [32] (also to combat overfitting) and then a dense, or fully-connected, layer that is set to the number of species we have in our dataset (number_of_classes). The choice of these layers to add to the network is a go-to solution when using transfer learning.

The next step is to train the model. This was done in two steps. Firstly, the VGG19 component of the model was frozen such that the weights in the kernels for that component could not be changed. Hence, the only training/changing of weights in this step is those defined in the layers we added to the end ourselves. Given that there are a relatively small number of weights, this step is quick to perform and is intended to bring the weights of the final layers into the approximate range of values for successfully classifying the images. This is continued in step two.

Step two is to perform the process of transfer learning. This is done by un-freezing the VGG19 component and training the entire model again on our dataset. You will see the epochs are set to 20 and 30 respectively for each step. One epoch is when the entire dataset is passed through to train the model. Hence, for the first step, the model is re-trained 20 times on the same data and 30 times for step 2. The effect of increasing or decreasing the epoch is that, when increased, the model can begin to suffer from overfitting (such that it has been trained too much on the training data that it can no longer sufficiently recognise images it has never seen before as it is too specific to those that it has seen). Alternatively, when decreasing the number of epochs, the model can suffer from underfitting (in that it has not seen enough training data to learn the correct kernel weights to accurately classify our images). Therefore, choosing the number of epochs is somewhat of a balancing act. However, Tensorflow provides us with a means to restore the weights of the model back to those that gave the highest validation accuracy (this accuracy is the most meaningful as it represents real-world accuracy/accuracy on recordings we have not seen in training). This is done using the *restore_best_weights* argument in the *EarlyStopping* object passed into the training method as a callback.

Following the training, we then simply download the model. This is done by converting the CNN model to a Tensorflow Lite file. This is a process provided by Tensorflow aimed at allowing developers to deploy neural networks on mobile devices (exactly what is needed here). Following the running of the notebook, you will see a model.tflite file being downloaded in your web browser.

The next section will discuss how this tflite file is deployed into an Android application that can be used to classify birdsong in a real-world environment.

4.4 Android Mobile Application

This section discusses how the Android mobile application was developed and how the CNN is deployed locally on the device. The source code for this can be found in the relevant zip file/GitHub repository.

In order to import a tflite file into Android Studio it must be populated with certain metadata. This includes the name of the model, the model version, the image input width and height, the number of classes, etc. In order to populate the file with metadata, Tensorflow has provided a Python script. This can be found in the codebase at *Tools/Model_Metadata_Maker/main.py*. This script requires the path to the tflite file to be specified as well as a text file that specifies the class labels for the model (the bird species that the model can predict).

In order to run this script, the arguments must be specified as below:

```
    python main.py --model_file=model.tflite --label_file=labels.txt --
export_directory=model_with_metadata
```

Code Listing. 7. TensorflowLite model metadata generation using the Python script

This will save a model.tflite file and a model.json file in a folder called *model_with_metadata*. The tflite is the file we need for the Android application.

As mentioned previously in section 3.2, the spectrogram that is generated on the mobile device must be generated in the same manner as the spectrogram that was used to train the neural network. To this end, the spectrogram image is made within the application using the mobile-ffmpeg Android library using the follow piece of Java code, given a recording from the device microphone saved in a wav file:

```
FFmpeg.execute("-i " + wavFile.getAbsolutePath() + " -y -lavfi
showspectrumpic=s=600x960:stop=10000 " + directory.getAbsolutePath()+"/" +
imageFilename +".png");
```

Code Listing. 8. Spectrogram generation using the mobile-ffmpeg Android library

The reason behind choosing the maximum frequency of 10 kHz for the spectrogram has been discussed before in section 4.1. The reason for setting the size of the spectrogram as 600x960 pixels has not been discussed. The reason for this was simply a matter of performance. By default, mobile-ffmpeg and the regular ffmpeg command line tool generate an image of 4096x2048 pixels. This is unnecessarily large, having higher resolution than a 4k image which would also be unnecessarily large.

In order to make the application as efficient as possible the image should be as small as possible without causing any issues (an unforeseen issue arose with smaller sizes in that ffmpeg would begin to crop pixels out of the spectrogram causing information to be lost). Therefore 600x960 was the smallest image that could be made without causing unwanted cropping.

The resulting spectrogram then has the axes and any additional border pixels (always the same dimensions) cropped from it. This was done using OpenCV as shown by the function below.

```
private void cropImage(File imageFile)
```

```
{
   Mat src = Imgcodecs.imread(imageFile.getAbsolutePath(),
Imgcodecs.IMREAD_COLOR);
   int borderWidth = 120;
   int borderHeigth = 60;
   Rect crop = new Rect(borderWidth, borderHeigth, src.width()-
(borderWidth*2), src.height()-(borderHeigth*2));
   Mat croppedImage = new Mat(src, crop);
   imageFile.delete();
   imwrite(imageFile.getAbsolutePath(), croppedImage);
}
```

Code Listing. 9. Function to crop an image file within the Android application

The reason why the dimensions from which the image was cropped in the Android application differ from the dimensions used to crop the image when generating the training data is that there are some behavioural differences between ffmpeg and mobile-ffmpeg as it is not an exact port of the library for mobile devices.

Now that we have our cropped spectrogram, it is simply a matter of running the inference over the image using the tflite model. This can easily be done using the following code:

```
try
{
   Model = Model.newInstance(context);
   TensorImage image = TensorImage.fromBitmap(myBitmap);
   Model.Outputs outputs = model.process(image);
   List<Category> probability =
outputs.getProbabilityAsCategoryList();
   model.close();
} catch (IOException e) {//Handle exception }
```

Code Listing. 10. Inference using the TensorflowLite model file within the Android application

We can then simply iterate over the probability list and update the user interface accordingly for the species that it has classified most confidently. Once the user records the next piece of audio, the spectrogram generated from the previous recording is deleted in order to limit the amount of storage space that is required by the application on the user's device.

Fig. 14 a) and 14 b) illustrate the Android application, showing the application before it has recorded any audio (this is done using the “record” button. The audio can then also be played back using the “play” button) and the application after it has classified a piece of audio (see the best guess at the top of the screen, followed by all possible species and the confidence values from the CNN in decreasing order).

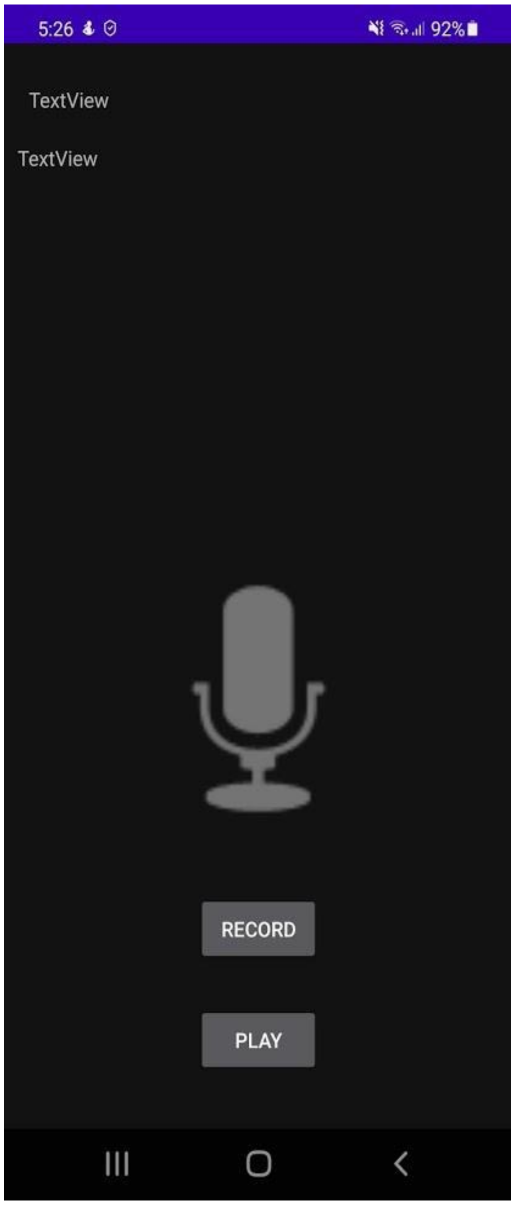
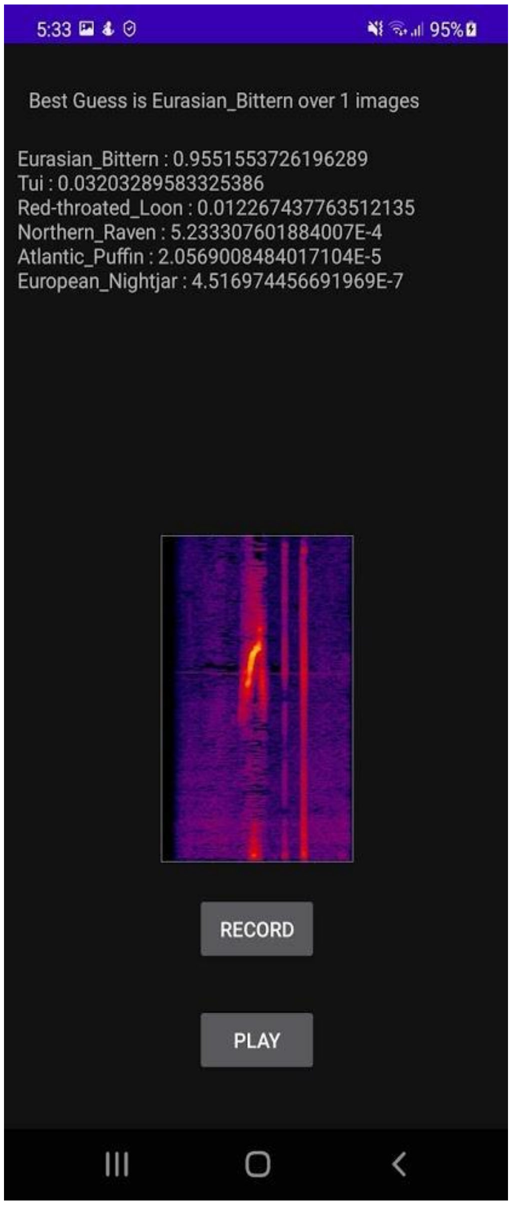


Fig. 14 a). Mobile application empty screen



Best Guess is Eurasian_Bittern over 1 images

Eurasian_Bittern : 0.9551553726196289
Tui : 0.03203289583325386
Red-throated_Loon : 0.012267437763512135
Northern_Raven : 5.233307601884007E-4
Atlantic_Puffin : 2.0569008484017104E-5
European_Nightjar : 4.516974456691969E-7

Fig. 14 b). Mobile application classification screen

5 Results

This section will discuss the results of investigating the previously described methods in section 3 when applied to birdsong recognition.

5.1 Six-Species Dataset (No preprocessing) [2]

This dataset, as discussed in section 3.2, contains recordings for the following six species: Atlantic Puffin, Eurasian Bittern, European Nightjar, Northern Raven, Red-Throated Loon and Tui. The audio recordings in this dataset are not preprocessed in any of the manners discussed in section 3.2. They are simply the audio recordings scraped from Xeno-Canto and are used directly with no attempt to clean the audio or augment the dataset.

Table 1 illustrates both the training and validation accuracy of the audio-fingerprinting approach, the audio-matching approach and a random baseline (chooses a species at random) at recognising the spectrograms of the audio recordings.

Method	Training Accuracy	Validation Accuracy
Audio-Fingerprinting	100 %	30.8 %
Audio-Matching	19.2 %	19.2 %
Random Baseline	16.6 %	16.6 %

Table 1. Training and validation accuracy for the six-species dataset

Fig. 15 a) shows the training and validation accuracies for the CNN across a series of 10 training runs (i.e. for each run, the CNN is reset, trained again and validated against the validation dataset). The reason for doing this was to provide a more meaningful set of results rather than a single accuracy figure such as those in table 1 which could be a misleading result (i.e. an usually high outlier from typical accuracies). The minimum, maximum and accuracy range is also shown for training and validation as well as the average accuracy for both across the 10 runs.

The reason for choosing the six species in this dataset was because their birdsongs sound quite distinct from each other (this was decided through a manual process by listening to a selection of online videos found here [33] and here [34] that highlight unique sounding birds and taking a selection of species for which I had gathered the most recordings from Xeno-Canto). This was in an effort to ensure the CNN was capable of distinguishing between any birdsong at all, never mind birdsong from different species that sound quite similar.

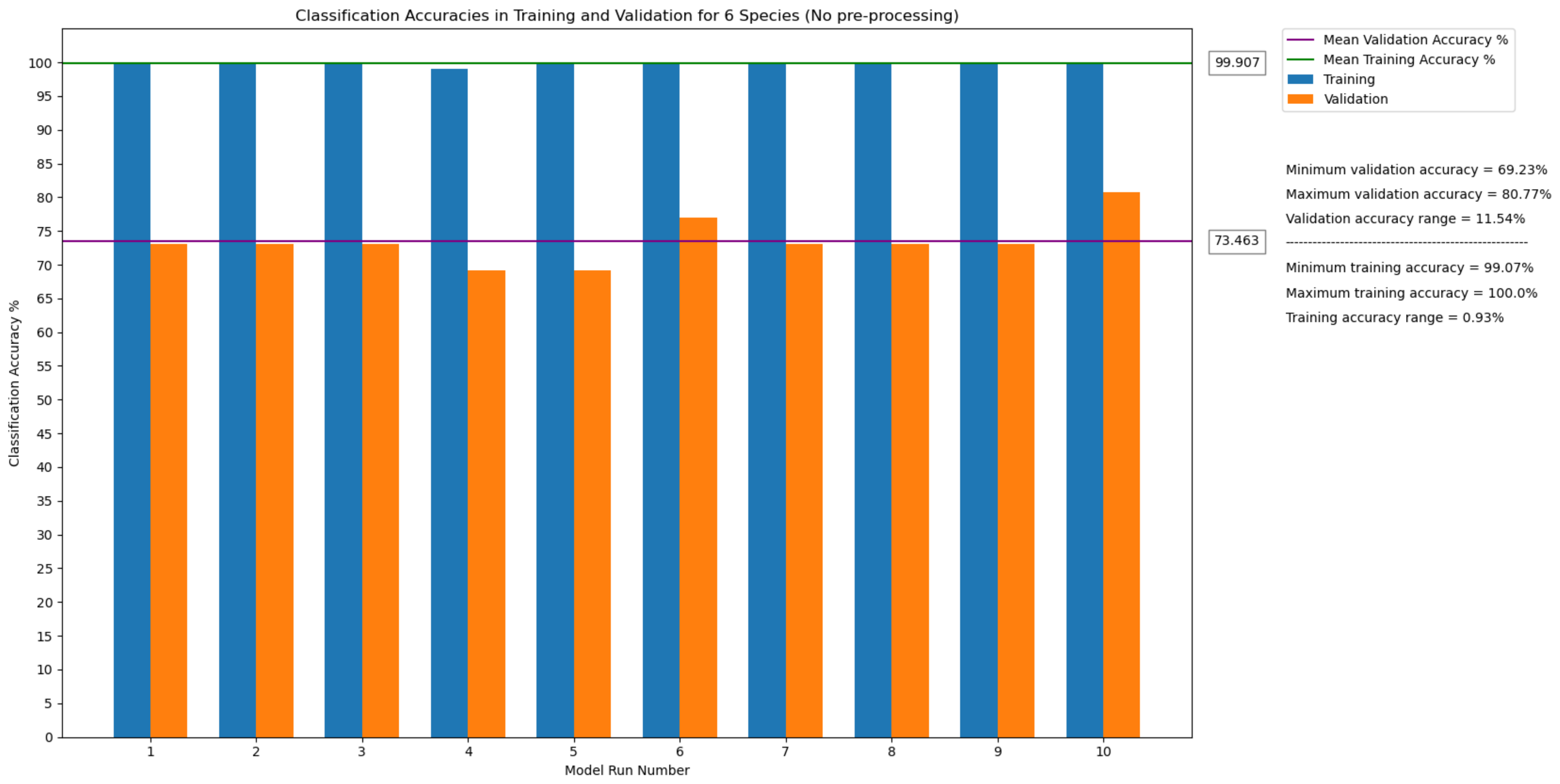


Fig. 15 a). CNN accuracies visualisation for 6 species dataset (no pre-processing)

5.2 Top Seven Species Dataset (No preprocessing) [2]

This section provides the results of the three methods examined but when applied to a dataset of the top seven species in the entire dataset. These seven species are: Common Blackbird, Common Chaffinch, Common Chiffchaff, Eurasian Blackcap, Great Tit, House Wren and Red Crossbill. The CNN results are provided with a bar-chart visualisation, as before.

The reasons for examining these methods on this dataset are as follows:

- There is a much larger number of recordings for each species than the number of recordings available for the six-species dataset mentioned above. For example, there are 23 audio recordings available in the entire dataset for the Atlantic Puffin (this is the species for which I have the lowest number of recordings of all six). However, for the House Wren (also the lowest number of recordings for the top seven) I have 221 audio recordings available. Therefore, I have almost 10 times as many recordings for these top seven species. The methods were examined on this dataset in order to determine if the availability of training data was the major contributing factor to the validation accuracy figure.

- This dataset was also chosen as the top seven species just happen to sound quite similar (at least in comparison to the six-species dataset which was intentionally engineered to contain distinct sounding birdsong). This allowed for an investigation into whether it was the biodiversity of the dataset that was the key component to a higher or lower validation accuracy.

Table 2 illustrates both the training and validation accuracy of the audio-matching and the baseline approach. Note, I have not included the figures for the audio-fingerprinting method as we can see in section 5.1 that the CNN vastly outperforms it with regards to the validation accuracy and given that all the development for the audio-fingerprinting method was done locally, I lack the hardware resources to efficiently generate fingerprints for every audio recording in the top seven species as it is a very computationally intensive process (note: If you attempt this locally, your machine may likely freeze).

Method	Training Accuracy	Validation Accuracy
Audio-Matching	21 %	22.4 %
Random Baseline	14.3 %	14.3 %

Table 2. Training and validation accuracy for the top seven species dataset

[2] The results for the methods presented in the tables within this section are not repeated across runs, similar to the CNN, as the methods are either deterministic or will generate the same result across runs. Hence, the bar-charts would all be the same. This would not provide any additional useful information. Therefore, a single accuracy figure is given in the table.

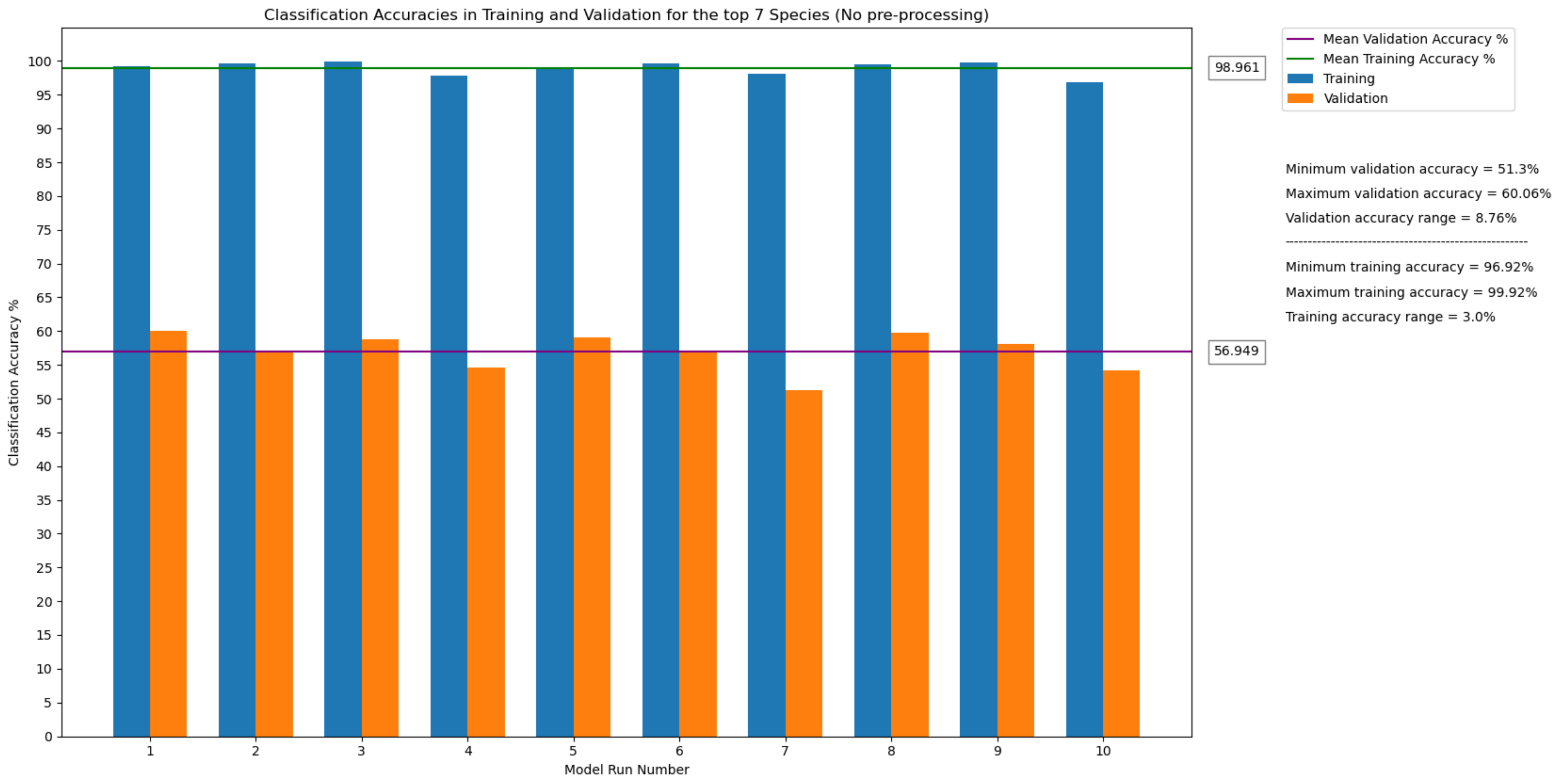


Fig. 15 b). CNN accuracies visualisation for top 7 species dataset (no pre-processing)

5.3 CNN with audio cleaning & splitting

Given that in the previous section we have shown the CNN to be the most accurate method, in this section I will provide the classification accuracy results for the data preprocessing methods described in section 3.2 and 4.1 for the CNN approach only.

The below visualisation of results (see fig. 15 c)) are the accuracy figures on both the training and validation dataset for the six-species dataset also used in section 5.1. For this, the training dataset was preprocessed as described in section 4.1 (loudness normalisation, high-pass/low-pass filters, etc.) and split into 3 second chunks with a 25% overlap. The validation dataset also underwent the same process, such that any 3 second chunks stay within the same dataset that the original audio file was placed in (i.e. there will be no pieces of audio in the validation set that came from the same clip as pieces in the training set - this prevents issues of environmental noise classification as described before in section 3.2). See the accuracy visualisation in fig. 15 c).

It is worth noting that the concept of splitting up the audio into equal sized chunks presents its own issues when implementing this process on mobile devices. These will be discussed in section 6.3.2.

5.4 CNN with Audiomentations

In contrast to section 5.3 discussing the results of the CNN on cleaned audio, this section will provide the accuracy figures for the CNN when used on the augmented training dataset of the six-species dataset, described in section 3.2.2.

The corresponding visualisation for this section follows the visualisation for section 5.3 on the following pages (see fig. 15 d)).

See section 6 for the conclusions that can be drawn from these results.

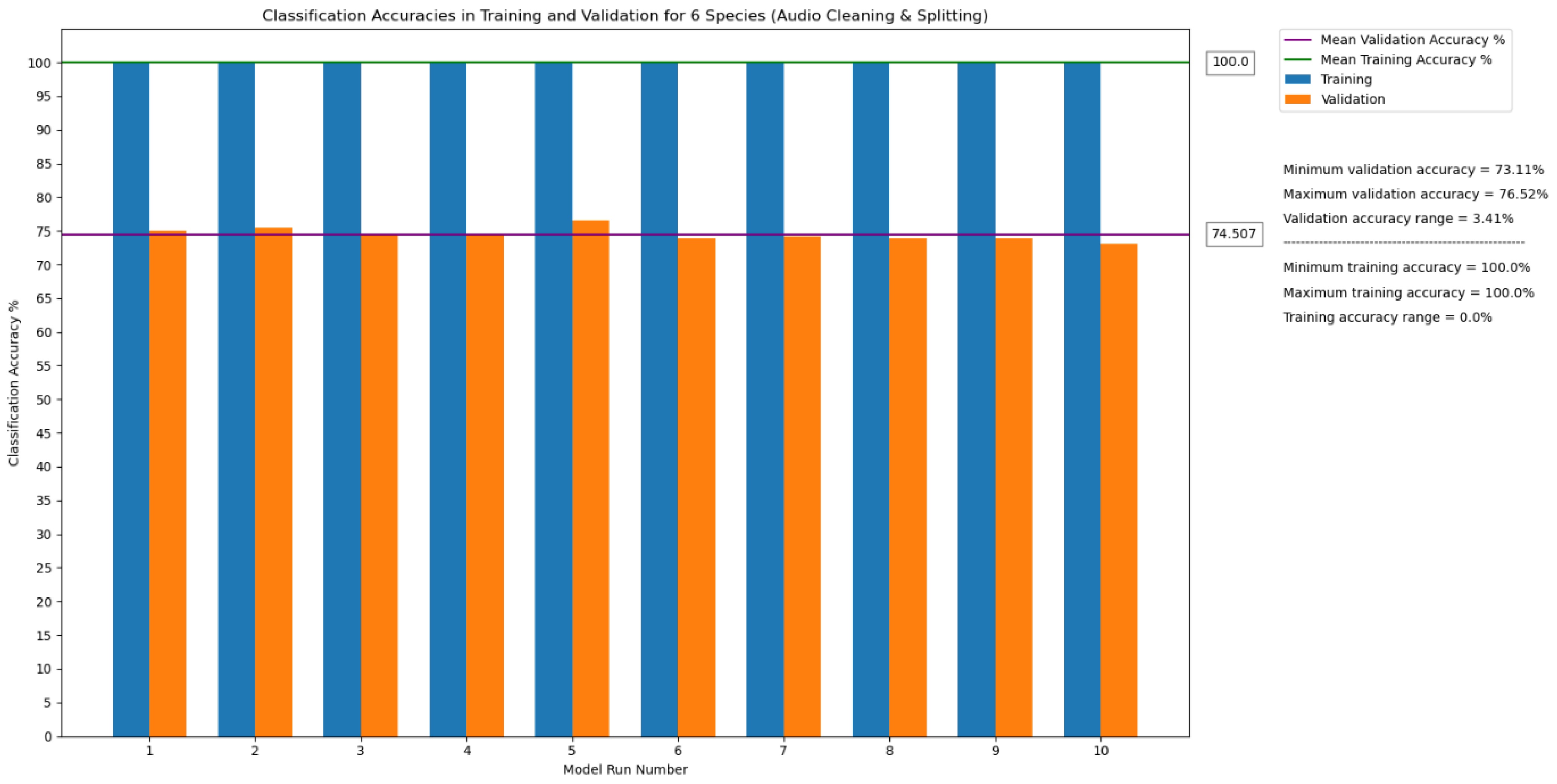


Fig. 15 c). CNN accuracies visualisation for 6 species dataset (audio cleaning & splitting)

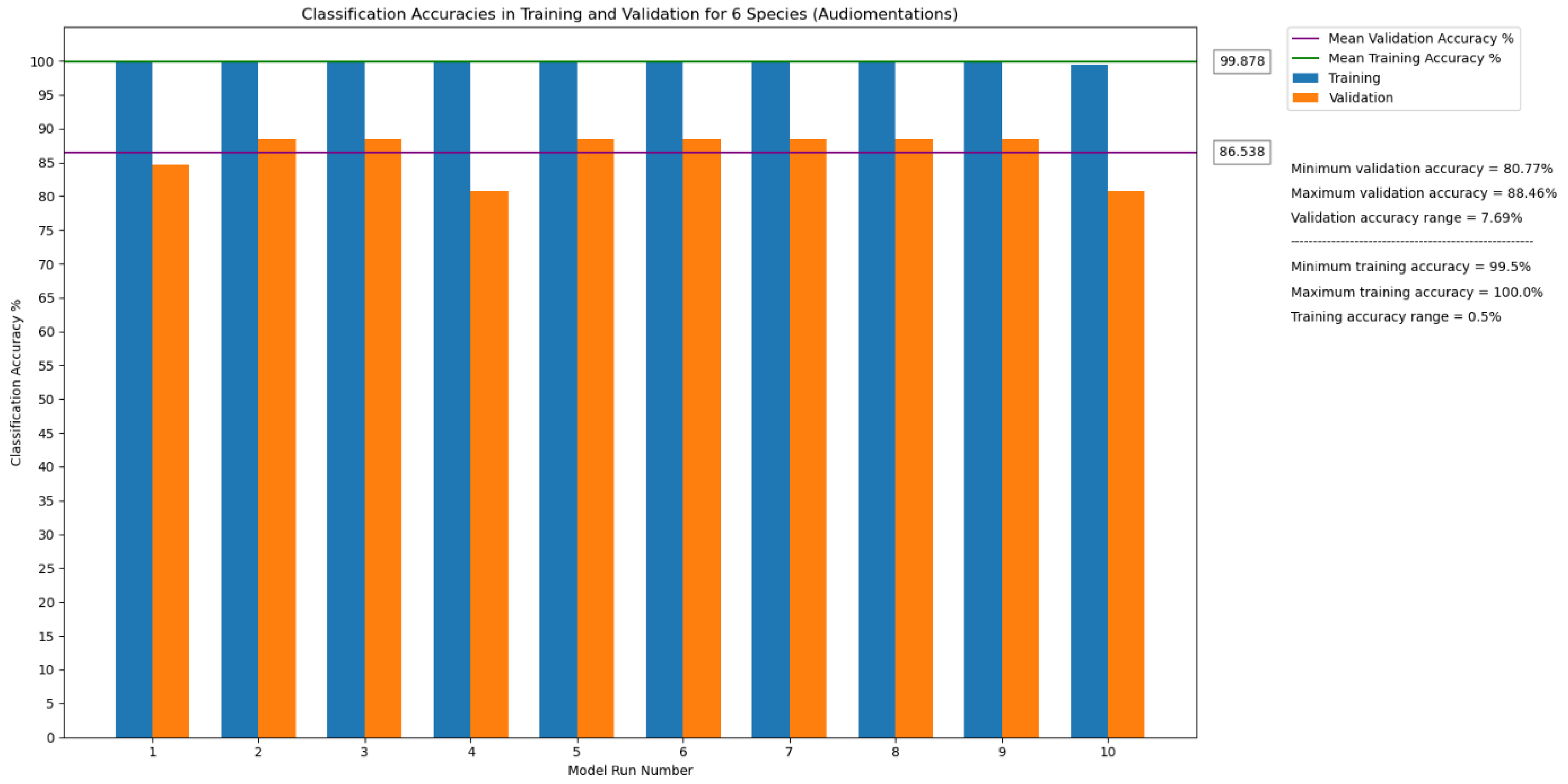


Fig. 15 d). CNN accuracies visualisation for 6 species dataset (audiomentations)

6 Conclusions and Future Work

This section will discuss the conclusions that can be drawn from the results presented above in section 5 and what we can say about the different methods for classifying birdsong and how they can be provided through mobile devices.

It is clear from examining the results above, for any of the datasets used, that the approach of using a CNN trained on spectrograms of birdsong is the way to go about solving the problem of birdsong classification. More ‘traditional’ methods of audio recognition such as audio-fingerprinting and audio-matching simply cannot match the performance of the CNN in multiple respects. Had I known this sooner, a deeper investigation into methods to improve the CNN accuracy could have been carried out.

6.1 Audio-Fingerprinting Conclusions

Firstly, let's discuss the results of the audio-fingerprinting method. By looking at the training and validation accuracy on the six-species dataset of 100% and 30.8% respectively, we can arrive at a couple of conclusions regarding its suitability to the problem we are trying to address. By simply considering the difference between the training and validation accuracy, we can conclude that this method overfits. That is, it fails to adequately generalise to previously unheard audio. However, it is excellent (flawless on the dataset used) in classifying audio it has already seen in ‘training’. Realistically, this behaviour is to be expected when applying this method to the problem of classifying birdsong recordings captured in outside environments.

This is due to the fact that the process was developed for recognising music, to be used in mobile applications like Shazam. From the perspective of classifying musical tracks, there is the obvious advantage that, once a piece of music is published, it does not change in any manner whatsoever. The only changes in the audio that the application will have to deal with, between the user recorded audio and the track that was recorded and released by the artist, is a certain level of background noise. However, when dealing with birdsong that was recorded in the wild, our ‘training’ audio will already contain a certain level of noise (not something we have to worry about in music). Our birdsong recordings also vary greatly by length, volume, etc. We also have to deal with multiple birds of the same species singing at the same time or even different, but less prominent, birds in the same environment. In short, the process of audio-fingerprinting is too specific to the ‘training’ audio provided as the hashes that are generated from additional samples of birdsong of the same species just aren’t likely to equal those generated in training.

Now, one method to combat this overfitting, and the go-to method in the field of machine learning, is to simply use more training data. However, this presents a couple of problems. One being that we don’t have more training data for those species in the dataset and in order to bring the validation accuracy into line with that produced by the CNN, we would need a vast amount of training data, more than would be realistic. For example, imagine all the possible variations in audio that users can submit of a certain species (different levels of wind noise, cars passing by, people talking, the sound of leaves under-foot, etc.). In order to successfully classify these recordings, we would also need to have fingerprinted extremely similar pieces of audio in training. To do this for every species simply is not realistically possible.

We then also have the problem that with more data, comes increased query times. Even when using an efficient database management system, as we are (MySQL), we still need to iterate over all the training data used in order to check if it matches the query audio or not. So, even if we did manage to gather enough training data to sufficiently increase the validation accuracy, the query times would be unsuitable and the database would be too large to run locally on a device without the need for servers in the cloud (this was a requirement stated in the introduction).

6.2 Audio-Matching Conclusions

In contrast to the results of the audio-fingerprinting approach, we have the audio-matching method. Rather than exhibiting behaviour of overfitting, audio-matching is an example of an approach that suffers from underfitting. This means the approach fails to correctly classify either the 'training' audio or previously unheard audio. It fails to capture the behaviour of a species' birdsong sufficiently, resulting in poor training and validation accuracy. From this we can conclude that the approach is too general, as opposed to fingerprinting which is too specific.

This is not necessarily unexpected behaviour, as this method was also intended to be applied to music, being able to recognise different versions of the same piece of music. However, when considering birdsong, it is an unsuitable assumption that birdsong from different species will be unique enough that we will not misclassify a species but that the method will also be able to handle variations in background noise for recordings of the same species. While many species of bird have unique birdsong, many of them are just too similar for such a general method as audio-matching to distinguish between them.

This leads us to the conclusions that using features similar to CENS (Chroma Energised Normalised Statistics) that aim to smooth deviations in tempo, etc. are unsuitable for the problem of classifying birdsong.

6.3 Convolutional Neural Network Conclusions

Given that we know audio-fingerprinting and audio-matching exhibit overfitting and underfitting respectively (however, it is worth noting that both methods still outperformed a random baseline), we need a method that can strike a middle ground. A method that can sufficiently capture the behaviour of a species' birdsong, but is also general enough to classify unseen data. From our results, we can conclude that using a convolutional neural network is an example of such a method.

We can see that the average training accuracy of the CNN is roughly equivalent to that of the audio-fingerprinting approach on the six species dataset with no pre-processing while the validation accuracy is almost 2.5 times higher than the validation accuracy for audio-fingerprinting. This presents a much more usable method for classifying birdsong in the real world as the model will never have seen the spectrogram, which a user of the application has generated, in training. So the validation accuracy for the CNN is the accuracy you should expect for the final application as all the recordings in the validation dataset are real-world, unedited recordings.

6.3.1 Problems with the CNN approach

However, the CNN method is not without its problems. One issue, which is highlighted by the drop in the average validation accuracy between the six-species dataset and the top seven species dataset, is how the biodiversity of the dataset can affect the performance of the model. That is, how distinct the examples of birdsong are, between species, in the dataset. As mentioned, this was why the performance was investigated on the top-seven species dataset, as the species in that collection sound quite similar in comparison to the species in the six-species dataset.

You can see how the decrease in biodiversity negatively affects the performance as the average validation accuracy dropped to slightly over 50%. This was found to be the major contributing factor to the performance of the model.

When considering the performance of a single model at classifying similar birdsong, we see a similar issue to the audio-fingerprinting approach. That is, overfitting. You can see that the average training accuracy for the top seven species is still extremely high, at 98%, compared to the validation accuracy. This leads to a similar conclusion as before. In order to successfully classify similar sounding birdsong, we would need a lot more training data. However, given that these seven species have the largest number of recordings available, this implies that the amount of training data would be very substantial, to the point at which dedicated hardware would need to be used to train the model and much more time would need to be invested. For example, the top performing model in the BirdClef competition, which was mentioned before in section 2.2, was trained for over 100 hours. This amount of time was beyond what was feasible for this project.

Another issue that is present relates to the nature of the dataset itself. Given that, in order to train the model, we are relying on data submitted by people from around the world, some species are simply going to be recorded more than others, as they are more common. This leads to a massive data imbalance in the dataset, where 50% of all recordings I had gathered were in the top 630 species out of 5262. This results in some species simply not having enough data available to train the models effectively. Also, the fact that there is such a large number of species in the dataset means that the performance of the CNN will degrade after a certain point. For instance, VGG19 was designed to recognise 1000 different everyday objects from the ImageNet dataset, not 5262 different classes.

6.3.2 Audio preprocessing and augmentation conclusions

It is also worth noting that the validation accuracy can be increased somewhat by appropriately pre-processing the audio. As you can see from the results in section 5, the method of cleaning the audio and splitting it up into equal sized chunks (see section 4.1) was investigated as well as augmenting the audio files (see section 4.2).

From these results we can conclude that attempting to clean the audio through processes like loudness normalisation, high-pass filters, etc. has no real effect on either the training or validation accuracy and only serves to make the method more complex and less efficient as the process must also be repeated on the mobile application. The fact that we have to repeat the process of cleaning and splitting the audio within the mobile application not only makes the application less efficient, but we then must devise a method by which we generate a final prediction from a collection of predictions (one for each chunk). The method I originally investigated was simply taking the average predictions for each species. However, due to the issues mentioned above such as a lack of training data, etc. the CNN can produce highly varied confidence values for incorrect classifications (i.e. it can give an incorrect classification a confidence of 90%, for example). Naturally, this can wreak havoc on the averages.

A better, alternative method to data preprocessing, that does not require any cleaning or splitting of the audio, is using the Audiomentations library. This increased the average validation accuracy on the six-species dataset by 13%. There are two reasons for this. Firstly, by augmenting the audio recordings we are generating more training data, which backs up the points made above, that more training data is necessary to increase the accuracy. Secondly, it trains the CNN to recognise, for example, an Atlantic Puffin, in the presence of a variety of different noisy environments or when dealing with distorted audio. This increases the robustness of the application as the model can then learn to ignore different forms of ambient noise (as the noise, distortion, etc. is present across the training dataset in various forms).

6.4 Future Work

Regarding how to combat the decrease in validation accuracy due to the biodiversity, this would be an interesting area of future investigation. In order to successfully classify birdsong, ideally we want to refrain from fundamentally altering the audio as this can then result in the model confusing different species. To this end, an interesting topic that could yield successful results would be an area of machine learning called ‘ensemble learning’. This involves, rather than just using one model to generate predictions, the input being passed into a collection or ‘ensemble’ of models that are all trained on different datasets. Applying this to birdsong classification, it may be possible to generate a series of datasets with the maximum possible biodiversity without the need to fundamentally change the audio through pre-processing and then train a collection of neural networks with each network being trained on one of the datasets in the series. This should ensure each model has the highest possible validation accuracy for the species it was trained on and the correct prediction would be from the model that outputs the highest confidence.

However, it is extremely difficult to quantify the biodiversity of a dataset. For instance, the six-species dataset was generated manually by listening to a collection of different species and deciding which of them sounded most distinct to the human ear. Therefore, in order to investigate the feasibility of ensemble learning, a metric would need to be constructed to correctly generate the datasets.

Ensemble models may also help in combatting the effects of having so many classes to predict because, as mentioned in section 6.3.1, the fact that there is such a large number of species in the dataset means that the performance of the CNN will degrade after a certain point. Having a collection of CNNs will reduce the number of classes per model.

Regarding future development, it would also be beneficial to develop more advanced digital signal processing libraries for Android platforms as mobile-ffmpeg is quite limited in its customizability (i.e. it does not allow for additional parameters to be specified when generating the spectrogram, as the leading solutions from the BirdClef competition do (see section 2.2), such as specifying the window size for the Fast Fourier Transform).

7 References

[1] [Connor21] D. Connor, "*Birdwatchers flying high as lockdown piques interest*", https://www.rte.ie/news/coronavirus/2021/0127/1193485-birdwatching/, retrieved Apr 2021.

[2] [Holub13a] J. Holub, A. Divito, "*Why do birds sing?"* New York: Puffin Young Readers.

[3] [Ehrlich88a] P. Ehrlich, D. Dobkin, D. Wheye. *""Bird Voices" and "Vocal Development" from Birds of Stanford essays"*, https://web.stanford.edu/group/stanfordbirds/text/uessays/uBird_Voices.html, https://web.stanford.edu/group/stanfordbirds/text/uessays/uVocal_Development.html, retrieved Apr 2021.

[4] [Kalra13a] P. Kalra, T. Mathur, T. Gupta, *"Survey paper on information retrieval algorithms and personalized information retrieval concept"* in *International Journal of Computer Applications*, pp.66.

[5] [Müller15] M. Müller, *"Audio Identification"* in *Fundamentals of Music Processing*, International Audio Laboratories Erlangen, https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S1_AudioIdentification.html, retrieved Apr 2021.

[6] [Müller15] M. Müller, "Audio Matching" in *Fundamentals of Music Processing*, International Audio Laboratories Erlangen, https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S2_AudioMatching.html, retrieved Apr 2021

[7] [Forte74] A. Forte, *"Tonal harmony in concept and practice"*, Holt, Rinehart and Winston.

[8] [Fraisse82] P. Fraisse, *"Rhythm and tempo*" in *The psychology of music*, *1*, pp.149-180.

[9] [Klapuri06] A. Klapuri, "*Introduction to Music Transcription*" in *Signal Processing Methods for Music Transcription*, edited by Anssi Klapuri and Manuel Davy, 1–20 (New York: Springer, 2006): p. 8. ISBN 978-0-387-30667-4.

[10] "hertz". (1992). *American Heritage Dictionary of the English Language* (3rd ed.), Boston: Houghton Mifflin.

[11] [Gregersen10], E. Gregersen, *The Britannica Guide to sound and light*. Britannica Educational Publishing.

[12] [BirdWatch Ireland], "*Countryside Bird Survey (CBS)"*, https://birdwatchireland.ie/our-work/surveys-research/research-surveys/countryside-bird-survey/, retrieved Apr 2021.

[13] [Sullivan01a] Å. Sullivan, C. Edlund, C.E. Nord, C.E., *"Effect of antimicrobial agents on the ecological balance of human microflora*" in *The Lancet infectious diseases*, *1*(2), pp.101-114.

[14] [Müller15] M. Müller, *"Content-Based Audio Retrieval"* in *Fundamentals of Music Processing*, International Audio Laboratories Erlangen, https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7.html, retrieved Apr 2021.

[15] [Winterman13] D. Winterman, *"The surprising uses for birdsong",* BBC News Magazine, 8 May 2013, https://www.bbc.com/news/magazine-22298779, retrieved Apr 2021.

[16] [Müller15] M. Müller*, "Subsequence DTW"* in *Fundamentals of Music Processing*, International Audio Laboratories Erlangen, https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S2_SubsequenceDTW.html, retrieved Apr 2021.

[17] [Müller15] M. Müller, *"Diagonal Matching"* in *Fundamentals of Music Processing*, International Audio Laboratories Erlangen, https://www.audiolabs-erlangen.de/resources/MIR/FMP/C7/C7S2_DiagonalMatching.html, retrieved Apr 2021.

[18] [Iandola16a], F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, *"SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". arXiv preprint arXiv:1602.07360*.

[19] [Image-net.org17] *ImageNet*, http://www.image-net.org/.

[20] [CLEF17] www.imageclef.org, *BirdCLEF 2017 | ImageCLEF / LifeCLEF - Multimedia Retrieval in CLEF*, https://www.imageclef.org/lifeclef/2017/bird, retrieved Apr. 2021.

[21] [Sevilla17a] A. Sevilla, H. Glotin, *"Audio Bird Classification with Inception-v4 extended with Time and Time-Frequency Attention Mechanisms*" in *CLEF (Working Notes)*.

[22] [Wild Ambience17] "*The Incredible Call of the Southern Cassowary"* in *Wild Ambience*, https://wildambience.com/wildlife-sounds/southern-cassowary/.

[23] [Purves01a] D. Purves, G.J Augustine, D. Fitzpatrick, *"The Audible Spectrum"* in *Neuroscience. 2nd edition*. Sunderland (MA): Sinauer Associates, https://www.ncbi.nlm.nih.gov/books/NBK10924/

[24] [The Cornell Lab09] "*Do bird songs have frequencies higher than humans can hear?"* in *All About Birds*, https://www.allaboutbirds.org/news/do-bird-songs-have-frequencies-higher-than-humans-can-hear/.

[25] [Rafii12a], Z. Rafii, B. Pardo, *"Repeating pattern extraction technique (REPET): A simple method for music/voice separation"* in *IEEE transactions on audio, speech, and language processing*, *21*(1), pp.73-84.

[26] [PremiumBeat16] "*15 Free Ambient Background Tracks"* in *The Beat: A Blog by PremiumBeat*, https://www.premiumbeat.com/blog/free-ambient-background-tracks/, retrieved Apr. 2021.

[27] [Columbia] www.ee.columbia.edu. "*Sound examples: Noise"*, https://www.ee.columbia.edu/~dpwe/sounds/noise/, retrieved Apr. 2021.

[28] [Simonyan15a] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition" at ICLR (International Conference on Learning Representations) 2015, pp. 1 – 14, Visual Geometry Group, Department of Engineering Science, University of Oxford, April 2015.

[29] [Brownlee19] J. Brownlee, *"How Do Convolutional Layers Work in Deep Learning Neural Networks?"* in *Machine Learning Mastery*, *17*.

[30] [Wood] T. Wood, *"What is the softmax function"* in DeepAI, https://deepai.org/machine-learning-glossary-and-terms/softmax-layer, retrieved Apr 2021.

[31] [Brownlee18], J. Brownlee, *"A gentle introduction to dropout for regularizing deep neural networks*" in *Machine Learning Mastery*, *3*.

[32] [Brownlee19], J. Brownlee, *"A gentle introduction to pooling layers for convolutional neural networks*" in *Machine Learning Mastery*, *22*.

[33] [www.youtube.com] *"World's Weirdest Bird Sounds - Part One"*, https://www.youtube.com/watch?v=Ray5GGBlZHk, retrieved Apr. 2021.

[34] [www.youtube.com] *"World's Weirdest Bird Sounds - Part Two"*, https://www.youtube.com/watch?v=x3luv_nZDzY&t=3s, retrieved Apr. 2021.

[35] [Drevo20] W. Drevo, Dejavu (Version e56a4a2) https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/ [Source Code]. https://github.com/worldveil/dejavu.

[36] [Müller15] M. Müller, *"FMP Notebooks"* in *Fundamentals of Music Processing*, International Audio Laboratories Erlangen, https://www.audiolabs-erlangen.de/resources/MIR/FMP/C0/C0.html, retrieved Apr 2021.

[37] [FFmpeg team21] FFmpeg (Version 4.3.2) http://ffmpeg.org/ [Source Code]. https://github.com/FFmpeg/FFmpeg

[38] [tensorflow21] Tensorflow (Version 2.4.1) https://www.tensorflow.org/ [Source Code]. https://github.com/tensorflow/tensorflow

[39] [nussl21] Nussl (Version 1.1.9) https://interactiveaudiolab.github.io/project/nussl.html [Source Code]. https://github.com/nussl/nussl

[40] [Sener21] T. Sener, mobile-ffmpeg (Version 4.4) https://tanersener.github.io/mobile-ffmpeg/ [Source Code]. https://github.com/tanersener/mobile-ffmpeg

[41] [Jordal21], I. Jordal, Audiomentations (Version 0.16.0) https://pypi.org/project/audiomentations/ [Source Code] https://github.com/iver56/audiomentations

[42] [opencv21] OpenCV (Version 4.5.2) https://opencv.org/ [Source Code]. https://github.com/opencv/opencv

8 Appendix

8.1 List of Technologies Used

Dejavu: A Python library for generating audio fingerprints, aimed at musical applications, which can then store these in a database and match incoming audio queries to those existing fingerprints. This was analysed as a possible solution for classifying birdsong by treating it as musical [35].

Libfmp: A Python package that was distributed as part of the 'Fundamentals of Music Processing' by Meinard Müller, a collection of materials and Jupyter notebooks for learning/teaching music audio processing. This package contains an audio-matching implementation that was also examined as a possible solution [36].

FFmpeg: An open-source command line tool used for the processing of audio, video and other multimedia files. This was used for preprocessing the audio files and for generating spectrograms [37].

Android Platform: A mobile devices operating system based on a version of the Linux kernel, developed by Google. This was the operating system for which the final application was developed.

Google Colab: A cloud hosting service for executing Python notebooks (.ipynb files), providing access to performant GPUs and high-ram machines. This service is mainly aimed at machine learning applications, but was also used to preprocess audio as it can be mounted to a user's Google Drive account.

Tensorflow: An open-source library for developing machine learning applications and training deep neural networks. Also provides access to pre-trained convolutional neural networks. It is available in a range of programming languages (Python was used for this project, on Google Colab). Also allows the users to save the trained models as TensorflowLite files for deployment in Android applications [38].

REPET / Nussl: REPET (Repeating Pattern Extraction Technique), a technique developed by Northwestern University, Illinois, US, to extract the vocals from music tracks into its own audio file and the backing instruments into another file. This was examined as a possible method of isolating the birdsong from its environment, as it is distributed through the Nussl Python package [39].

Mobile-ffmpeg: An Android mobile port of the tool described above. This was used for generating the spectrogram and processing the audio within the Android application [40].

Java: An object oriented programming language that was used to develop the mobile application, as Java and Kotlin are the two languages intended for use in Android Studio. However, Kotlin is still in its infancy compared to Java.

Python: A high-level, multi-paradigm, open-source programming language commonly used in data science and scientific computation. Due to its relative simplicity and ease of use, there are numerous libraries available across a huge range of applications, including signal processing which are of use for this project.

Audiomentations: A Python library used for augmenting audio datasets by taking the original audio and adding effects such as clipping distortion, mp3 compression, adjusting the gain, adding background noise, adding Gaussian noise, etc. [41]

OpenCV: A computer vision library originally developed by Intel, to be used for computer vision applications. It is available in a range of languages (a Python (used in Google Colab) and java version (used in the Android application) were used for this project) [42].