Birdsong Recognition
A Mobile Application

By

Suvineet Singh Kalsi

Final Year Project for
BSc (Hons) Computer Science and Mathematics
with Industrial Experience.

Department of Computer Science
Faculty of Science and Engineering
University of Manchester
May 2020

ABSTRACT

Automatic birdsong recognition is a challenging task and a sparsely researched subject. This project uses a deep learning approach to explore the possibility of making automatic birdsong recognition accessible to an every-day common user. This approach uses image recognition based on extracting characteristics of birdsongs from their spectrograms, via training a Convolutional Neural Network (CNN) with data for a number of European birds. An acceptable accuracy is realistically achievable with the possibility to scale to large number of bird species, limited by computational cost. The outcome of this project delivers a mobile application along with a deep learning model, which performs with an accuracy of 72.1% on 17,840 test samples, for 19 different bird species. The mobile application communicates with the model, which is on a server, via a REST API.

ACKNOWLEDGMENTS

Firstly, I would like to thank both my supervisors, Dr. Andrea Schalk and Dr. Tim Morris, for their constant support, guidance and feedback throughout the year. I feel grateful to Dr. Schalk for encouraging me and believing in me to pursue this project in the beginning. Despite unfortunate circumstances which meant Dr. Schalk had to take a leave, she was always available to assist me with any queries remotely. I would like to thank Dr. Morris, who was happy to step in for Dr. Schalk in this unusual scenario. He was always available for validating and improving my ideas which helped me to stay focused and deliver at a high quality.

I am indebted to my parents, my brother and my friends for their encouragement and support during difficult times. They are important to me and I am thankful to them for keeping me sane throughout the year.

Finally, I would like to thank all the members of staff and my peers at the University of Manchester for their assistance and expertise.

IMPACT OF COVID-19

The world faced a global pandemic by the name of COVID-19 during the latter stages of this project. The whole country experienced a lockdown for several weeks resulting in the closure of all public places. The University was majorly impacted as well having to shut down on-campus studies or communication. This had an impact on everyone involved and took a little while to settle in at first. The entire timetable was disrupted and assessment deadlines were extended. Although this did not affect the development of the project as the majority of the implementation was completed, my personal plan to deliver this report was disrupted. The disturbance in other assessments meant that I was unable to focus on completing the report in time. However, I am grateful to the department of Computer Science for understanding these concerns and extending the deadline for the report in these unprecedented times.

Contents

Chapter 1: Introduction

The ability to recognise birds by their sounds is extremely beneficial in the observation of these organisms in our ecosystem. The majority of birds vocalise to communicate with each other for territorial claims, mating and coordination efforts [6].

Bird vocalisations can be separated into songs and calls. A bird song is usually more complex and used for engagement and mating, whereas calls are relatively simple and often brief. Calls are commonly used for alarming other birds or to communicate information to members of a flock about the whereabouts of another [53].

An experienced listener may be able to distinguish many birds by their songs and calls. However, considering the fact that there are over 10,000 recorded bird species, this task becomes very impractical [26]. The use of tools such as employing sonograms has aided ornithologists in identifying birdsongs more swiftly and accurately, accelerating the scientific knowledge obtained into the behaviour of birds [33, 5].

Birdsong recognition is used in monitoring bird population, migration patterns and public applications for birdwatchers [42]. Possessing a fully automated system that could conduct this procedure would be extremely beneficial in large scale applications.

The project explores many complex areas of computer science, including audio signal processing, image recognition, machine learning and performance optimisation to provide a scalable solution for birdsong recognition that can be available to the general public.

1.1 Aims and Objectives

The aim of the project was to develop an automated system which can reliably classify a bird from a given birdsong recording. The goal was to have a mobile application which would be publicly available for users to record a birdsong, communicate it to a backend and obtain a classification. Furthermore, the backend system that performs the classification should perform the analysis with a high degree of accuracy.

The following objectives ensured success in meeting the overall goal for this project:

- Research on different approaches to conduct birdsong recognition.
- Design and implement a method which can be improved upon further by performing various experiments.
- Conduct performance analysis of the models and methods.
- Develop an iOS mobile application that allows users to record/upload a recording of a birdsong from their device, in order to get a classification of a bird. In addition, ensuring that the mobile application is intuitive, user-friendly and easy to use.

1.2 Existing Work

Research in automated birdsong recognition has increased over the past decade. A number of competitions have been held to encourage the development of solutions to this problem for different requirements.

The "MLSP 2013 Bird Classification Challenge" [27] has produced some promising approaches for a normalised data set of 645, ten second audio recordings for 19 different species of birds. The winning entry achieved an AUC of 95.4% by implementing a random forests approach using spectrogram cross-correlation and a binary relevance [18].

This was further expanded on a much larger data set of 14,027 audio files retrieved from Xeno-Canto [59], covering 501 individual species in the "LifeCLEF 2014 challenge" [22]. The winning solution achieved an AUC of 91.5% during cross-validation and a Mean Average Precision of 51.1% on the validation set [34]. The solution used a mixture of spectrogram cross-correlation, audio scene analysis techniques and a randomised trees classifier.

With regards to commercial solutions, there exist applications like Warblr [57], Chirpomatic [10] and BirdGenie [7] which indeed have a limited scope but perform the intended outcome desired. Bird Genie is able to classify about 200 species with an accuracy of 85% [30], whereas Warblr classifies a lower range of around 80 different species. However, it has a remarkable accuracy of 95% in optimum conditions [54].

This problem has also been attempted in 2016 and 2017 at the University of Manch-

ester by third year students. The first project used a dynamic warping algorithm approach to measure similarities between waveforms. This achieved an accuracy of 77%, tested with 5-fold cross validation for a relatively small amount of 4 different species [39]. The second approach involved image processing and cross-correlation mechanisms to extract characteristic song segments from spectrograms. These are used to identify similarities between birdsongs and classify any species present in recordings using a random forest classifier. This project was able to achieve an accuracy of 89% for 12 different bird species [45].

1.3 Proposed Solution

This paper presents a solution that uses image recognition conducted by deep learning methods. This makes use of the Xeno-Canto [59] dataset to retrieve audio samples which are used to train a Convolutional Neural Network (CNN). The audio is pre-processed and normalised into spectrogram images, which are used as input for the CNN. This pre-trained deep learning model is then utilised to classify new samples.

This is an approach that has proven to work well for Human Speech Emotion Recognition [4]. Although, human speech differs vastly from a birdsong and is much more easily recognisable, it is fair to assume that such an approach has the potential to produce promising results for birdsong recognition.

This model is then used to classify bulk data inputted into the system, but more importantly can be accessed by a REST API to predict individual samples in a request. Hence, the outcome of this project is an intuitive, user friendly iOS mobile application which allows users to communicate with the trained model to get classifications for real life data.

1.3.1 Process Overview

The project is divided into five separate parts, which enabled the entire integrated system to be built in a logical fashion. Each part is discussed in detail in further relevant sections, touching on the thought process behind each decision made.

1. **Data retrieval:** The data collected is recordings from uncontrolled environments. This helps factor in a real-world performance when training the model. Since, the amount of data required is huge, this process must be automated.

2. **Preprocessing:** The collected data required preprocessing. i.e. normalising the samples by audio metadata, duration and reducing noise before generating spectrogram images.

3. **Deep Learning model:** Using the preprocessed data, the implementation of a suitable Convolutional Neural Network (CNN) was needed to utilise this data to be trained.

4. **REST API:** Implementing a REST API that can make use of the trained CNN to attain predictions for given samples.

5. **iOS application:** Develop an iOS mobile application that allows users to record a sample from within the application and acquire a classification using the pre-trained model via the REST API.

1.3.2 Solution Overview

The system is split into two separate components. A client side mobile application and a containerised deep learning model that gathers, pre-processes and applies the training dataset to the deep learning model.

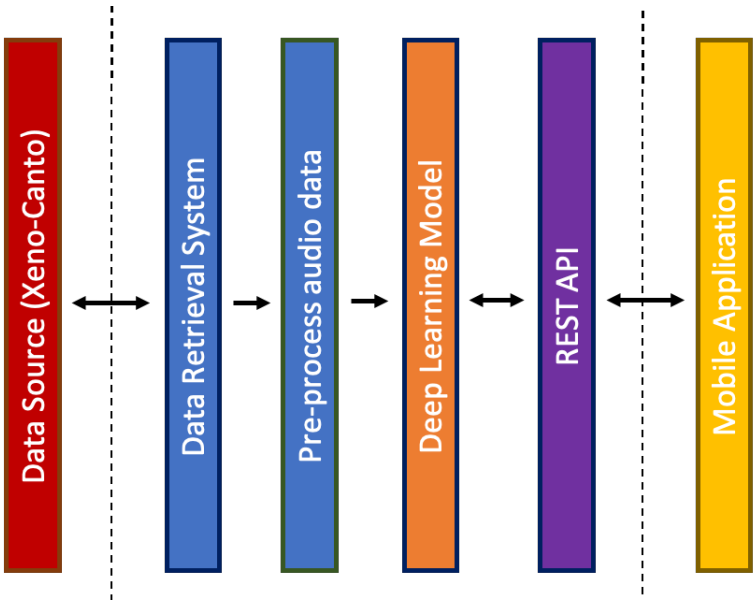

Figure 1.1: *Overview of the delivered solution.*

1.4 Report Structure

This report consists of 8 chapters:

- **Chapter 1: Introduction -** Problem outline describing the aims and objectives for the project along with a review of existing research to establish the proposed solution.
- **Chapter 2: Data Retrieval and Preprocessing -** A detailed dive into the methodology of collecting and preparing the samples utilised.
- **Chapter 3: Deep Learning -** Detailed description of the deep learning approach taken to solve the problem.
- **Chapter 4: Experimental Design -** Outlines the experiments performed to improve the performance of the model.
- **Chapter 5: Model Evaluation -** Description of evaluation methods applied to test the model and analyse results.
- **Chapter 6: Tools and Technologies -** Details of the tools and technologies used in the implementation.
- **Chapter 7: Final Product -** Details and presentation of the mobile application developed as the final outcome of the project.
- **Chapter 8: Conclusion -** Summary of the project's achievements, challenges faced and improvements for the future.

Chapter 2: Data Retrieval and Preprocessing

This chapter will focus on the procedure of gathering data that was used to train and test the deep learning model. This is an integral part of the process as any approach to classifying birdsongs requires existing data samples to decipher information about different birdsongs. Furthermore, once the raw data is collected, it is necessary to process it prior to using it, to construct a model. The diversity in the data samples indicates that a means of normalisation is crucial to obtain a reliable performance from the model [9]. To tackle this, several open source databases were explored that could provide useful samples.

Building on from the idea that large amounts of data samples were required, the volume of data available was analysed. A conscious choice of focusing on European bird species was made due to two main reasons. Firstly, this allowed me to test the final product in real-time without geographical restrictions. Secondly and more importantly, European birds had a high quantity of samples available to access.

2.1 Dataset: Xeno-Canto

The data was collected from Xeno-Canto [59] which provides over 527,990 recordings for 10,188 distinct bird species. The recordings are contributed by professional and amateur users from all around the world. This includes over 149,000 recordings for 714 individual European bird species from 51 different countries.

The duration of recordings can vary from a few seconds to over ten minutes. Some of these contain long durations of silence, or multiple species may be present in the background. However, the high quality samples reliably contain individually recognisable species. All the samples are provided in MP3 format and can be accessed using the simple public API provided by Xeno-Canto.

2.2 Metadata

The audio samples provided by Xeno-Canto contain relevant metadata which can be used to extract valuable information about each sample. The metadata includes the species in the recording, date, time and location of the recording and additionally, the presence of multiple species that may exist in the audio. For the proposed model, only the species in the recording was used. The location was not used in

this implementation as the main concern was regarding European birds. However, when expanding it further to more birds, the location could be useful to improve the accuracy of the model by filtering species by the location they are found in. This would add another part to the implementation where this extra information will impact the weights in the CNN. However, this was not explored within this project.

2.3 Data Retrieval Automation

The project required consideration of time efficiency in conducting the study. Downloading the data from Xeno-Canto manually was particularly tedious, especially due to the lack of a "bulk download" feature. To tackle this problem, an automated script was written that downloaded the relevant data required. The solution was enabled by Xeno-Canto's provision of a public API that could be used to retrieve all the information for a given species in a JSON format (See Appendix A for JSON schema).

Taking advantage of the API, an automated script was developed, written in `Python 3` which used the `urllib.requests` library to get a JSON response, containing the details of all the recordings for a given list of species. This JSON response included the file URLs for the recordings. The `wget` library enabled the download of all these recordings and save them in the appropriate locations automatically.

The automated downloader took an approximate total of 50 hours to download 42,081 recordings for 19 different species.

2.4 Preprocessing

This section explains the transformation of the raw data gathered from Xeno-Canto to the point where it could be used for training the Convolutional Neural Network.

2.4.1 Data Normalisation

As mentioned earlier, the data gathered from Xeno-Canto can vary a lot in terms of format, duration and quality. Therefore, before this data could be processed further, it had to be normalised, i.e. format the entire input data set in a way to make it consistent.

Firstly, all the recordings were normalised to the following format:

- **Format:** MP3
- **Bit depth:** 16-bits
- **Sample rate:** 22050Hz
- **Audio Channels:** Mono

The initial decision was to convert all audio to WAV format because it is a simple uncompressed format making it fairly easy to process and edit. However, due to the storage limitations it was not feasible to use WAV, which can be up to 10 times larger compared to MP3. Hence, the choice to use MP3 was made and all recordings were converted into this format. One variation in the audio samples was the number of channels. To deal with this issue, all the samples were converted to mono (one audio channel). To achieve this, for all stereo (2 audio channels) recordings, the right channel was removed. There is no clear evidence if the superposition of both channels is better than discarding one. Despite the fact that the superposition will retain all the information, it can increase noise in the data. However, all of this did not raise an issue because the number of stereo recordings were significantly low in the original data source. Only 9.3% of the 42,081 recordings were stereo.

Another key complication was the deviation in the duration of the audio between samples in the dataset. To overcome this complication, some analysis was performed regarding the variation in the duration of samples for different species. Upon examination it was decided that all the recordings would have a length of 15 seconds at most. Any recordings that were longer than 15 seconds were split into segments of 10 seconds. The 10 second segment length was purely an experimental decision. It seemed appropriate to pick this number after considering the distribution of durations in different species. Keeping in mind the requirement for efficiency mentioned earlier, an automation script was written to perform this. The script possessed the ability to allow the user to provide any desired interval so that the data could be normalised using a different length easily. This could have been explored further in detail for specific species, to find the optimum split intervals. However, this was not pursued due to time constraints. It is useful to note that this process produced a larger dataset as more separated samples were available to train the CNN.

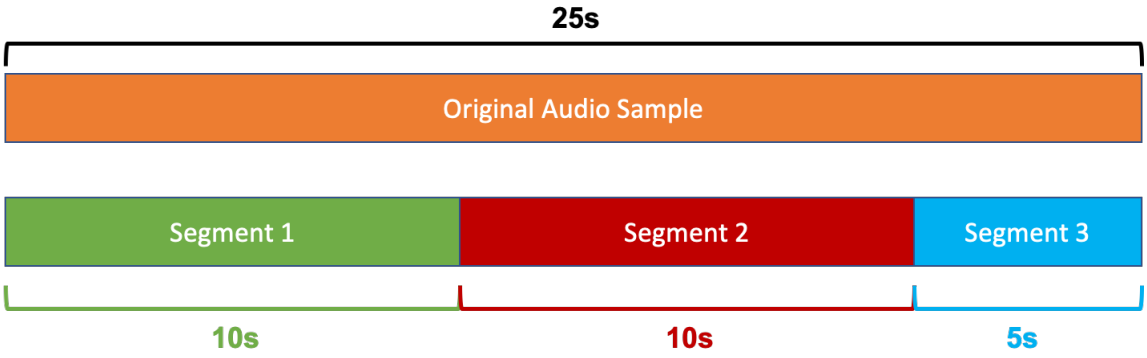

Figure 2.1: *Illustration of how audio samples longer than 15 seconds may be split into smaller segments.*

In addition, a further problem to consider was the audio files' quality. The majority of longer data samples contained a lot of silence, which also translated to the shorter normalised clips. Moreover, some samples were of extremely bad quality due to background noise. This meant the clarity of the birdsong was compromised. As a result, it had to be ensured that these useless data samples were filtered out from the dataset. To tackle this, the `pydub` python library was used to detect these silences and identify any unsatisfactory samples. This was done by using the maximum decibels relative to full scale (dBFS) in the audio, which is almost always the actual birdsong. A threshold from this maximum dBFS value was set, which was then treated as 'silence' in the sample. All the time intervals that were considered silent with the above criteria were extracted. If there existed only one such interval lasting at least 90% of the whole clip, then it was classed as an unwanted sample. However, if there were more than one such intervals, i.e. there was a periodic birdsong present in the audio, the sample was discarded if the cumulative total of 'silence' was more than 80% of the clip.

From the 42,081 raw recordings collected from Xeno-Canto, 387,780 samples were obtained after splitting them up into smaller samples. However, these were further filtered to a total of 220,696 samples for 19 individual bird species that were deemed useful as input data. It is important to note that all the pre-processing mentioned above was incrementally done by experimenting with multiple models to optimise the quality of the input samples. Chapter 4 discusses these experiments and decisions in detail.

2.4.2 Spectrogram Generation

A spectrogram is a visual representation of an audio signal representing the spectrum of frequencies varying over time. This gives an intuitive visualisation of sound as a 2-dimensional image. The time is kept across the x-axis with the frequency placed on the y-axis. The amplitude of the audio is represented as a heat colour map with a spectrum of colours ranging from low to high amplitudes [23]. Figure 2.2 below shows the preservation of amplitude knowledge from a waveform in a spectrogram along with how the frequencies are composed in the waveform.

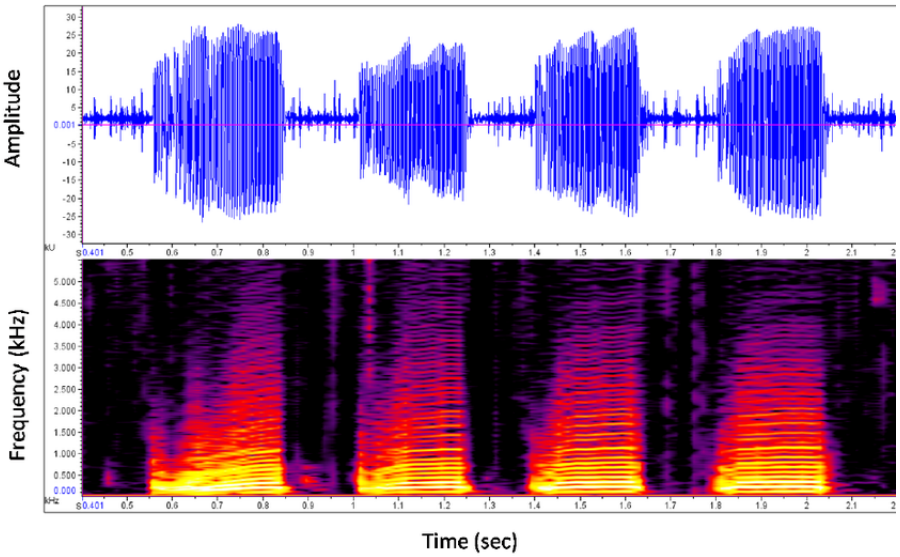

Figure 2.2: *An audio waveform and the corresponding spectrogram* [31].

To implement an image recognition approach for audio data, these spectrogram visualisations were used to distinguish characteristics of different birdsongs. This was accomplished by training a deep learning model with these spectrogram images as the input data. However, before this was possible, it was necessary to first generate these images from the audio data retrieved from Xeno-Canto.

To construct the spectrograms, the audio was loaded in its raw pulse-code modulation (PCM) format, which was used to generate the spectrogram using the windowed fast Fourier transform (FFT) [36]. This method was applied to all filtered recording samples and the spectrogram images were stored on disk. The Matplotlib library provides the `specgram` function which was used to generate these spectrograms. Fast Fourier Transform computation is handled efficiently by modern computers which

meant the process was reasonably quick. The following two important parameters were used for generating the spectrograms:

- **Window:** The Hamming window [52] of 512 points used to combine overlapping blocks.
- **Threshold:** `vmin` $= 20 \log_{10}(\text{maximum dBFS}) - 100$.
  This ignored any audio below the given threshold and plotted it as black on the spectrogram in order to reduce noise. The threshold uses the fact that the loudest birdsong recorded peaks at 125.4 decibals [20]. Hence, it was fair to assume a range of 0-100 decibels for the chosen samples.

The difference after adding the `vmin` threshold is depicted in Figure 2.3 below. It is clear that the sample on the right is much more refined by removing the background noise.

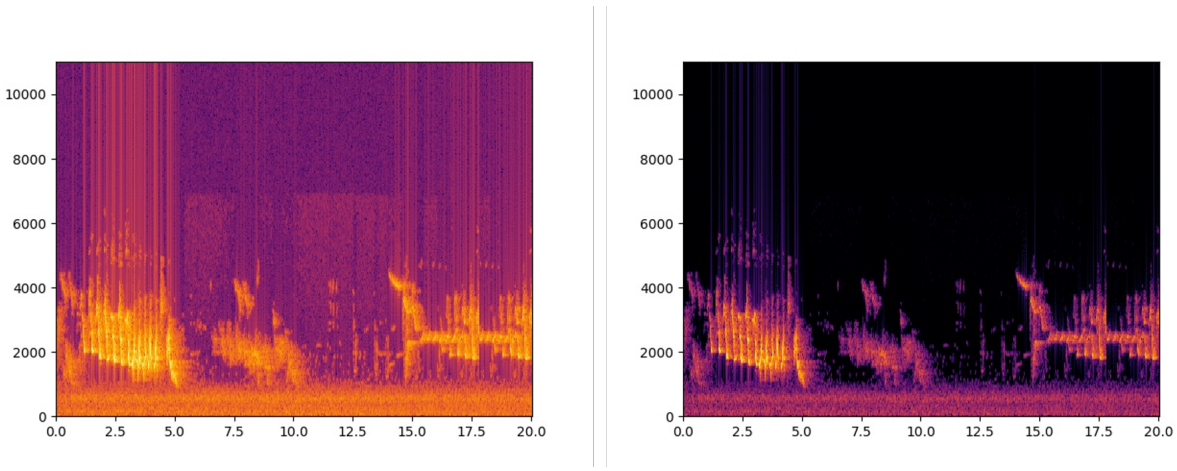

Figure 2.3: *Spectrograms generated with (right) and without (left) the* `vmin` *parameter.*

Chapter 3: Deep Learning

A lot of research was conducted regarding the approach to take to tackle this problem prior to developing the project. As seen in Section 1.2, the existing work majorly relies on having a Random Forests approach with a combination of other components. Although this approach has proven to be successful, this project was motivated to proceed from a different angle.

The method to use a Convolutional Neural Network was inspired from "Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network" [4], which has proven to work well. The method relied on a huge amount of input data samples which were available. A personal interest in learning about CNNs and how to implement them was also a factor in choosing this particular approach.

3.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network is a type of deep neural network which is most commonly used for image classification [13]. Importance of features is assigned for given input images to distinguish one image from another. A CNN is made up of multiple layers each performing a different task in the process. This variation of multilayer perceptrons design minimises the pre-processing required. CNNs are some of the most prestigious innovations in the field of computer vision. Details of how each layer works and is used in the proposed implementation is discussed below.

3.1.1 Birdsong Spectrogram CNN Architecture

Coloured images of size 256×256 pixels with three channels (RGB) were the inputs for the proposed CNN. These were represented by a $256 \times 256 \times 3$ array of pixel values. Various input image sizes were experimented to train and test different models. These are discussed in detail in Chapter 4.

The proposed model is constructed of 2 convolutional layers, 2 max-pooling layers and 2 fully connected layers. This choice was purely experimental in the beginning and was influenced from the deep learning project as part of the "Machine Learning A-Z: Hands-On Python & R In Data Science" course [16].

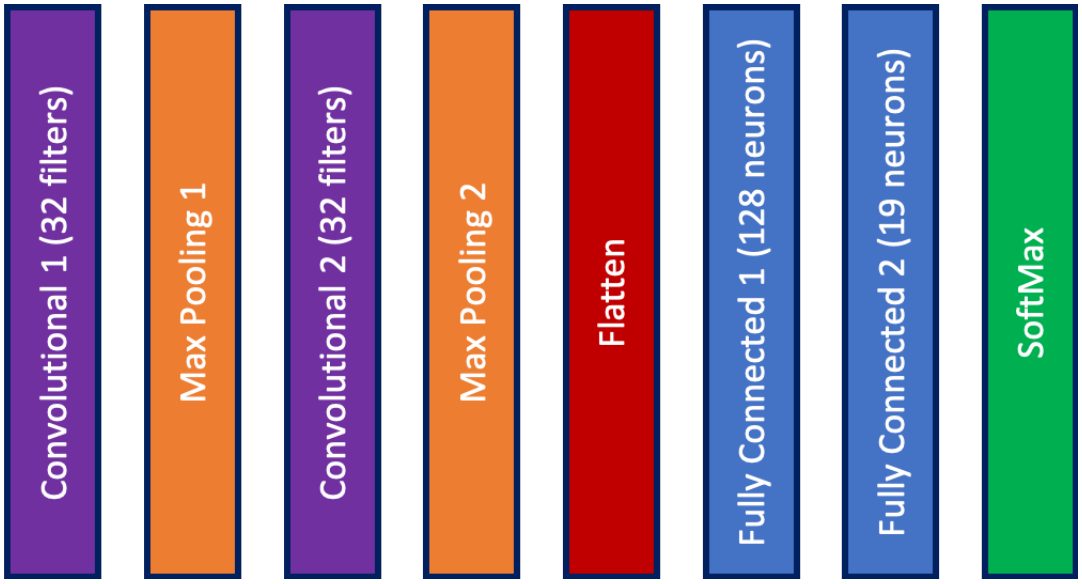

Figure 3.1: *Proposed CNN layers architecture.*

3.1.2 Convolutional Layer

As suggested by the name, convolutional layers are the core building blocks of a CNN. It consists of multiple feature detectors (also known as filters or kernels) which are spatially small, but expand over the full depth of the input volume. Each feature detector attempts to find certain characteristics such as straight edges, curves, simple colours, etc. in the image.

During the forward pass, each feature detector is convolved (slid) over the whole input image, computing the inner product between the matching positions of the filter and the input entries, producing an activation map. The collection of activation maps produced by each feature detector supplies the full output volume of the layer.

Figure 3.2 shows an example which consists of two feature detectors W0 and W1 of size 3×3, with a stride of 2 and a zero padding of 1. This illustrates how calculating the inner product and adding a bias b0 for the selected input (blue), with filter W1 (red) produces the output volume entry of 9 (highlighted green cell).

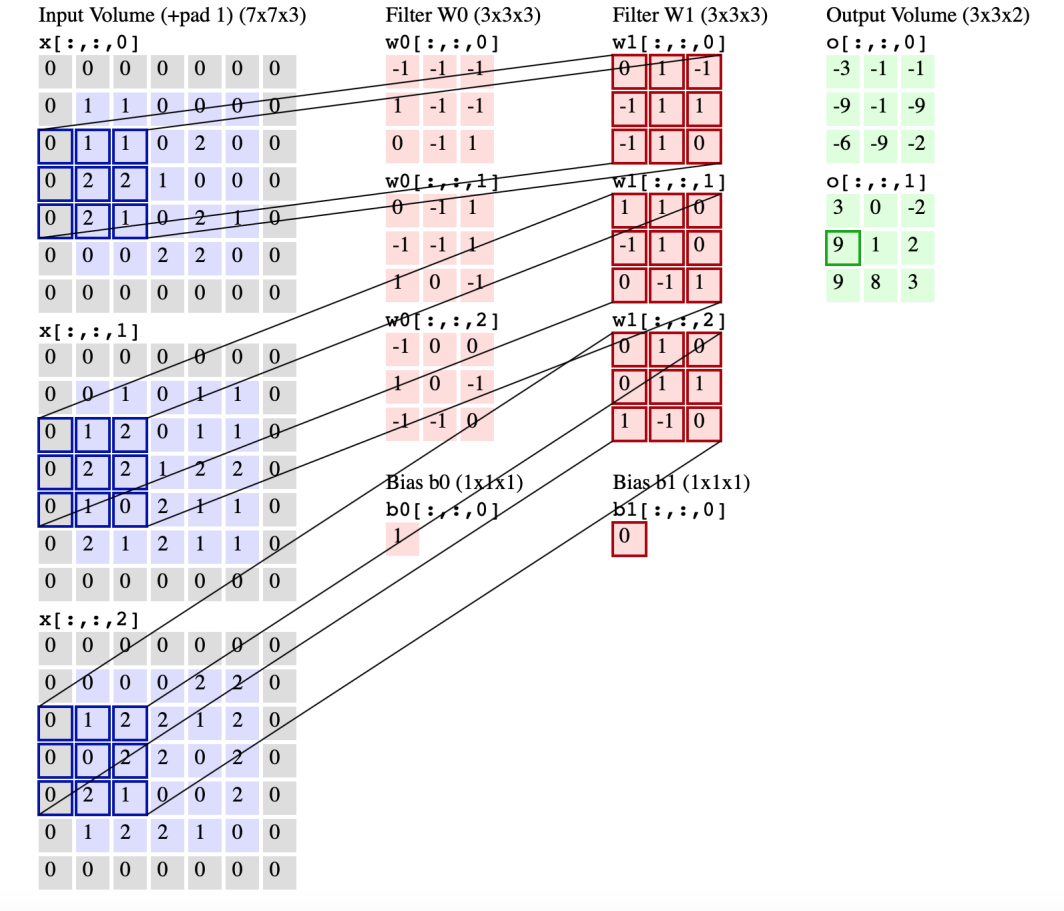

Figure 3.2: *Visualisation of a Convolutional Layer* [12].

The size of the output volume of the convolutional layer is determined by various hyperparameters such as the filter size, number of filters, and the stride (the amount the filter slides each time). For this implementation, the convolutional layers were constructed with 32 filters of size 3×3 and with a stride of 1. Similar numbers are commonly used in CNNs which influenced this decision [16, 48].

3.1.3 Activation Function

An activation function is used to attain the output of a node in a neural network. An inputted number is transformed non-linearly and passed onto the next layer as an input. There are many activation functions such as Sigmoid, Hyperbolic Tangent (tanh), Rectified Linear Unit (ReLU), etc. [49].

The Rectified Linear Unit activation function was implemented in the proposed

CNN model. The function is defined as $f(x) = \max(0, x)$, for all input values. It is clear to see that the function maps all negative input values to zero. This results in computational efficiency due to the fewer number of neurons getting activated each time.

The convergence of the stochastic gradient descent is sped up by a factor of six with ReLU compared to sigmoid and tanh activation functions [32]. ReLU is used in almost all the CNNs or deep learning models, making it the most used activation function. Hence, this was employed in the proposed model.

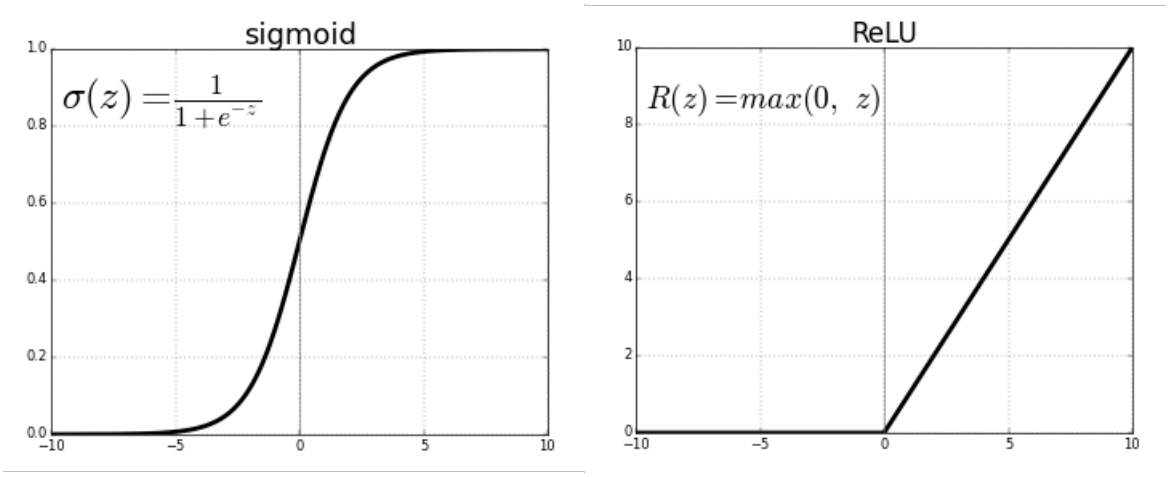

Figure 3.3: *Sigmoid vs ReLU function* [49].

3.1.4 Pooling Layer

A pooling layer is occasionally positioned between convolutional layers to spatially downsample the input volume. This aids in reducing the number of parameters and computation in the network, controlling overfitting, whilst preserving the most important features. It is also useful to remove the spatial invariance in the image. For example, two images of the same class may have different lighting, orientation, direction, etc. However, these variables should not affect the features extracted from the image. The reduction of less important data in the input performed by the pooling layer assists with this issue.

There are several types of pooling layers, but this project concentrated on Max-Pooling because it is the most used non-linear pooling function [12]. The implementation consisted of 2 max-pooling layers which applied a 2×2 filter with a stride of

2 to all depth pieces in the input. For each region of the input covered by the filter, the maximum value is chosen to generate a spatially smaller map.

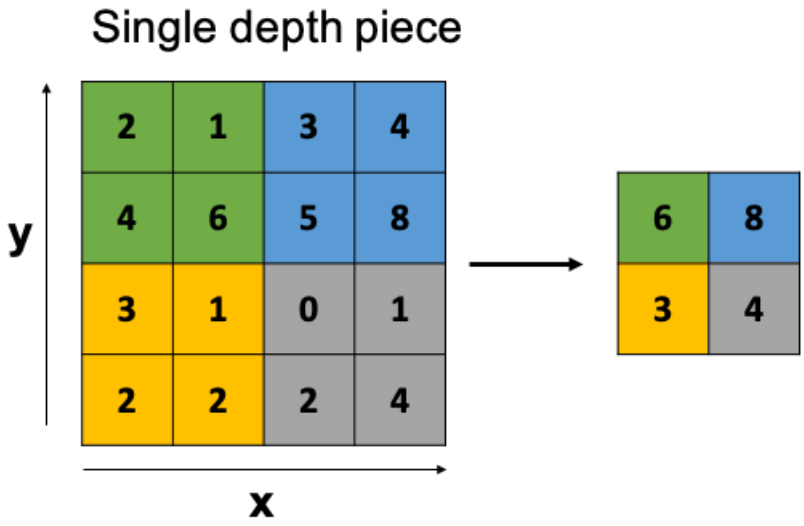

Figure 3.4: *Max Pooling with a* 2×2 *filter and stride* 2.

3.1.5 Fully Connected Layer

The final collection of layers are the fully connected layers which take as input the output values of the layers before them, and output an n-dimensional vector, where n is the number of class labels to be classified. The output of size $a \times b$ from the final max-pooling layer in the CNN is first flattened into an $(a.b)$-dimensional vector, to be used as the input for the first fully connected layer.

The proposed CNN has 2 fully connected layers with 128 neurons for the first layer and 19 neurons in the second layer, each corresponding to a bird species.

3.1.6 SoftMax Classifier

The final fully connected layer (i.e. the output layer) generates the class predictions for a given input, via a classification function. There are many classification functions such as SoftMax and Sigmoid which may be more suitable than the other for certain scenarios.

Sigmoid is generally used for binary classifications [49]. Since this particular problem required mutli-class classification, the SoftMax classifier was chosen which produces

a categorical probability distribution, i.e. the likelihood of each class is given in the range of 0 and 1, with the cumulative total equalling one. The class assigned with the highest probability is chosen as the prediction to validate the classification.

3.1.7 Prediction Result

To validate the accuracy of the model, the class with the highest probability is chosen as the classification by the SoftMax classifier. However, as far as the result for the user was concerned, this was altered to present a more useful metric.

The probability distribution for the classes did not always yield a clear winner for the predicted class. For example, there existed cases where the the probability distribution was as follows:

$$\texttt{Bird1} = 0.54, \quad \texttt{Bird2} = 0.45, \quad \ldots, \quad \texttt{Bird19} = 0.0001.$$

In a case like this, the model was unable to make a confident prediction for one individual class. Hence, rather than returning only `Bird1` to the user with a 55% certainty, it was sensible to return both `Bird1` and `Bird2`, indicating to the user the confusion in the prediction.

This was implemented such that any predictions that were at most 0.15 away from the highest probability, were included in the result.

Chapter 4: Experimental Design

This chapter will outline the different experimentation performed during the process of building the CNN. The aim was to build a model that was reliable and accomplished a high level of accuracy. However, the autonomous nature of CNNs meant that several experiments had to be carried out by adjusting the various hyperparameters involved, and evaluate their results to attain the optimum model.

4.1 Approach Validation

At the beginning of the implementation, a model with only 10 species was trained with minimal pre-processing of the samples, in order to validate that the method works and has the potential to be scaled further. As discussed in Chapter 2, the data samples were collected from Xeno-Canto and pre-processed into smaller segments. However for the first model, no further audio cleaning was performed on the data samples. This was a conscious decision made to explore the capability of the chosen approach.

An input image size of 64×64 pixels was chosen to train the initial model. The input size was optimised further, which is evaluated in later sections. The model was trained with 50,000 samples for the following 10 species, each containing 1000 samples. This model will be referred as *Model 1* for the remainder of this paper.

Table 4.1: *Bird species for training Model 1.*

Coal Tit	Eurasian Tree Sparrow
Common Blackbird	Great Spotted Woodpecker
Common Chaffinch	Great Tit
Common Chiffchaff	Song Thrush
Eurasian Blue Tit	Yellow Hammer

These specific samples were chosen based on the highest number of samples available. The training took approximately 24 hours with 25 epochs. The number of epochs are the number of complete passes through the training set. The correctness of the approach was validated by testing it against a validation data set. The model showed great promise by performing with an accuracy of 81.2% on 10,000 test samples.

4.2 Training

This section explores the progressive improvements made to train different models to obtain the final model. Before evaluating the steps taken to improve these models, the following two important components of the CNN are discussed, which are crucial in order to optimise the training of the neural network. Neural Networks are most commonly optimised using a Gradient Descent algorithm [41].

4.2.1 Loss Function

A loss function is a vital component of a neural network. It indicates the difference between actual values compared to the predicted values during the training of a neural network. Choosing the most appropriate loss function was important in order to optimise the output of the proposed neural network. There are different types of loss functions which include 'regressive' and 'classification' loss functions [1]. The problem in question required a classification output for multiple class labels. Therefore, the choice to use a classification loss function called the Cross Entropy loss [46] was made.

The categorical cross entropy loss function (also known as 'log loss') is used to measure the performance of a classification model which has outputs as probabilities between 0 and 1 [1]. Recalling the SoftMax activation function in Section 3.1.3, the classification result yielded probabilities between 0 and 1, which made this an appropriate loss function to use.

4.2.2 Optimiser

As mentioned above, this approach seeks to optimise the weights of the neural network by using a Gradient Descent optimisation algorithm in the back-propagation step of the training. There is a wide range of gradient descent optimisation algorithms available to use [47]. However, the aim was to choose an optimiser that was computationally efficient, had relatively small memory requirements and was suitable for a solution involving large amounts of data. To achieve this, the Adaptive Moment Estimation (Adam) optimiser was chosen, which is a method that computes adaptive learning rates for each parameter [29, 47]. This is an optimiser which satisfies the criteria of efficiency required and presents substantial performance benefits in the model's training speed [8].

4.2.3 Model Tuning

Once it was determined that the approach to use a CNN for birdsong recognition worked on ten bird species, this could be extended to more bird species. However, a complication arose regarding availability of data, causing a restriction on the number of bird species that could be added. As mentioned in Section 4.1, 5000 samples for each species were used to train the model along with a 1000 each to validate the model. However, there were only 9 more species which provided a similar number of samples. To tackle this, a compromise was necessary to restrict the model to have 19 species. Therefore, the following nine more species were gathered, for which at least 5000 samples could be retrieved.

Table 4.2: *Bird species added to the list in* Table 4.1 *to train Model 2.*

Common Nightingale	Eurasian Wren
Common Whitethroat	European Robin
Corn Bunting	Red Crossbill
Eurasian Blackcap	Tawny Owl
Eurasian Skylark	-

A new model was trained with these 19 species with minimal pre-processing similar to *Model 1*. It was intuitive to expect that the overall accuracy of the model would decrease as more species were added, which was exactly the outcome that was obtained. The model's accuracy decreased from 81.2% with 10 species to 61.1% after adding 9 more species. Although the performance was expected to deteriorate, the decline in the accuracy was unsatisfactory. This model will be referred as *Model 2* for the rest of the paper.

As a result of the decline in performance, the task of improving this as much as possible arose. A systematic and progressive approach was taken to achieve this objective. The remainder of this chapter discusses this method in detail.

4.2.3.1 Refining Spectrograms

The first intuitive step in an attempt to improve the model was to pre-process the input data further to make it more reliable. Researching audio characteristics of birdsongs, enabled the refining of spectrograms and reduced the background noise in them. As mentioned in Section 2.4.2, a threshold for the decibels to be plotted was

added when generating the spectrograms. To implement this, the `vmin` parameter of the `specram` function provided by `Matplotlib` was utilised. The choice of this value was justified in Section 2.4.2. This process removed any data below the threshold and plotted it in black, making the spectrogram much more refined (as shown in Figure 2.3). Applying this refinement to the input data was done in the hope that the model would be able to emit redundant features in the images. As a result, the overall accuracy would be expected to increase.

In contrast to the hypothesis above, the evidence showed that the refinement to the spectrograms did not improve the results significantly. A new model (*Model 3*) was trained with the same but refined audio samples used in *Model 2*. However, the accuracy of *Model 3* was only 61.7%, compared to the 61.1% in *Model 2* on the same validation set. It was slightly surprising to see a negligible increase of less than 1%, which meant that further investigation was required to improve upon the input data.

4.2.3.2 Manual Audio Analysis

The unflattering results obtained from *Model 3* raised suspicions regarding the reliability of the input samples used for training. To explore this further, some manual analysis was performed on the spectrograms that were used as part of the training set. A considerable amount of time was spent manually looking at spectrograms and listening to their corresponding audio samples to examine the data. As a result, it was discovered there was a significant problem in the data samples. A substantial amount of spectrograms seemed to have mostly silence in them. This silence was introduced as a side effect from splitting longer samples into smaller segments as explained in Section 2.4.1. Some of these samples can be seen in Figure 4.1 below.

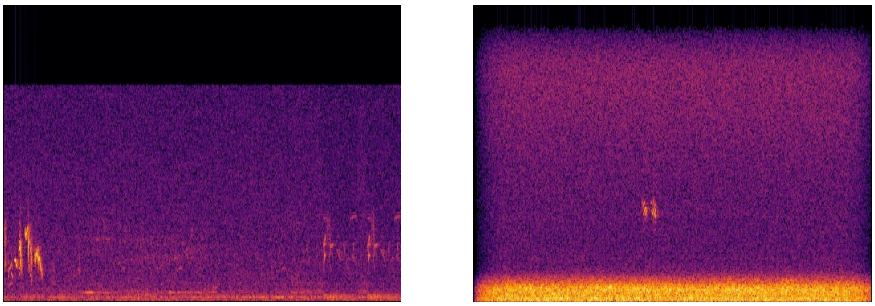

Figure 4.1: *Defective samples highlighting the absence of birdsong.*

To understand the depth of this problem, an automated script was written to detect silence in the samples. This was implemented using the `silence` module in the the `pydub` library. The criteria used to determine the silence in a given sample was explained in Section 2.4.1. This analysis was performed on the whole training set, obtaining the percentage of samples used that were classed as inadequate due to the silence.

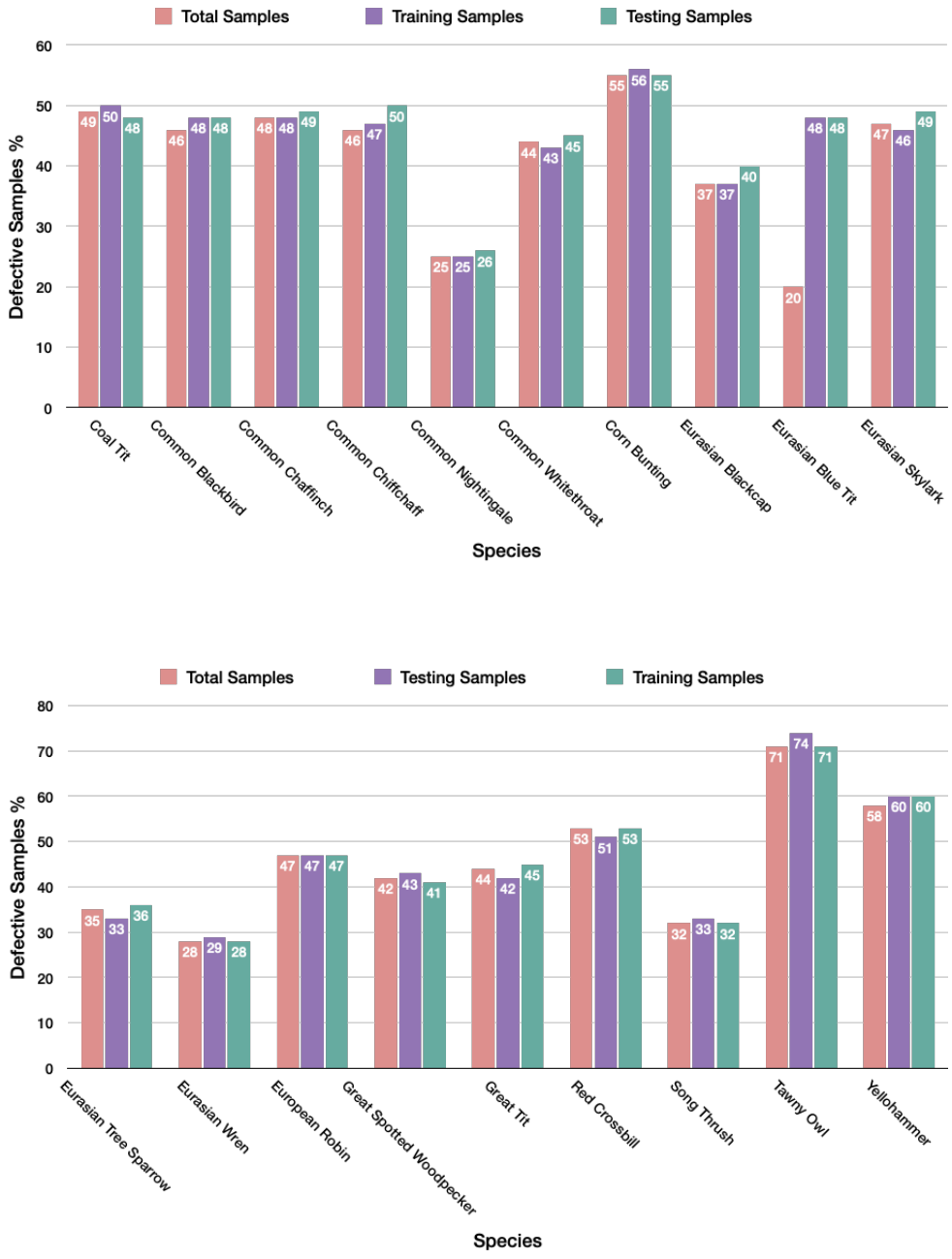

Figure 4.2: *Percentage of samples used in Model 3 classed as defective.*

The evidence in Figure 4.2 highlights that the training dataset in *Model 2* and *Model 3* contained a significant amount of unreliable samples. An average of 43% of all the samples in possession were labelled defective. From this, the randomly generated training set contained an average of 45% undependable samples. This analysis was a helpful exercise to determine a crucial flaw in the input data, allowing an improvement of sample quality for the training set.

Drawing upon the knowledge obtained on the quality of the samples, a new training set was generated by removing any unreliable samples described above. Since the number of filtered samples for a few species was not enough to have a training set of 5000 and a validation set of 1000, the training set was chosen to be of 4000 samples from each species (a total of 76,000). A new model (*Model 4*) was trained with this data which was expected to perform better than *Model 3* due to the lack of defective samples.

In contrast to before, on this occasion the hypothesis was supported by the results. *Model 4* performed with an accuracy of 65.8%, an improvement of 4.1% from *Model 3* on a validation set of 17,840 samples. This was a significant improvement to the model, which was justified and achieved because of the analysis performed on the input data. This demonstrates the importance of pre-processing the data at a high standard before it can be used by the deep neural network.

4.2.3.3 Hyperparameter Experiments

This section demonstrates the hyperparameter optimisation performed on several hyperparameters to enhance the performance of the model. The following models all use the same samples for training and testing. Firstly, the performance of the model was examined based on the input image size. Models 1-4 discussed earlier were trained with input image sizes of 64×64 pixels. The aim was to deduce if the performance would increase with a larger image size, due to more data being present which could be extracted from the image. The results of this experiment are summarised in Table 4.3.

Table 4.3: *Performance of models trained with different image sizes.*

Input Size (pixels)	Accuracy (%) 17840 samples	Training time (hours)
64×64	65.7	29
128×128	67.3	38
256×256	**72.1**	**48**
512×512	72.3	57

As evident in Table 4.3, the accuracy of the model improved with larger image sizes. However, a computational cost was added as a result of this. It is clear to see that the model trained with input images of size 256×256 pixels performed second best behind 512×512 pixels, by 0.2%. Nevertheless, the time taken to train was significantly higher for the larger input size. Therefore, it was concluded that the optimum image size to use was 256×256 pixels by considering the overall performance. Consequently, all further optimisation experiments used 256×256 pixels as the input image size.

Another hyperparameter that was inspected was the batch size. The batch size is the number of images chosen from the training set that are propagated through the network in one pass. Usually a larger batch size results in a lower test accuracy [50]. All models discussed above used a batch size of 32. The aim of this experiment was to explore if the accuracy could be significantly improved with a smaller batch size or if the training time could be decreased considerably, whilst preserving the accuracy by increasing the batch size. Models with the following batch sizes were trained and their performances are summarised in Table 4.4

Table 4.4: *Performance of models trained with different batch sizes.*

Batch Size	Accuracy (%) 17840 samples	Training time (hours)
16	72.9	68
32	**72.1**	**57**
64	67.7	48
128	64.3	34

As illustrated by the evidence, the training time was significantly lowered as the batch size increased. However, the accuracy of the model was compromised. It is also clear that the model performed marginally better with a batch size of 16 compared to 32. However, the improvement of 0.8% was not worthwhile due to the

11 hours of added time to the training. Hence, a batch size of 32 was concluded to be optimal when training the convolutional neural network.

4.2.4 Summary

Summarising the above experiments, it is clear to see that the accuracy of the model improved significantly compared to *Model 2*, as demonstrated above. An accuracy of 61.1% in *Model 2* was increased to an accuracy of 72.1% in the proposed model. A remarkable elevation of 11% concluded that the time consuming process of experimentation was valuable to the solution. A synopsis of the hyperparameters used in the final proposed model can be seen in Table 4.5 below.

Table 4.5: *Hyperparameters and their corresponding values used by the proposed model.*

Hyperparameter	Value	Description
Input Size	256×256	The size of images in the training set.
Batch Size	32	Amount of training samples to be propagated through the network at once.
Number of epochs	25	The number of complete passes through the training set.
Fully Connected Layer 1 neurons	128	Number of neurons in first fully connected layer.
Fully Connected Layer 2 neurons	19	Number of neurons in second fully connected layer.
Convolutional filter size	3×3	Size of filter in each convolutional layer.
Max Pooling filter size	2×2	Size of filter in each max pooling layer.
Max Pooling strides	2	Filter stride in each max pooling layer.
Optimiser	Adam	Optimisation algorithm.

Chapter 5: Model Evaluation

This chapter evaluates the final deep learning model that was delivered in the project. As illustrated in Chapter 4, careful optimisation of hyperparameters and methods to pre-process training data were applied to improve the model. The hyperparameters of the proposed model can be seen in Table 4.5.

The proposed model was trained on 19 species which are listed in Table 5.1. A total of 76,000 samples were used to train the model. The model performed with an accuracy of 72.1% on a validation set of 17,840 samples. This was a relatively good accuracy and demonstrated that the approach could be successfully scaled to more species. Although the accuracy of the model was deemed desirable, it was important to evaluate how well it worked in detail, in order to improve this in the future.

Firstly, the performance of the model per species was assessed. Table 5.1 shows the average accuracy for each species. The model was validated with 1000 samples for each species except for the Eurasian Skylark (59 samples) and the Great Spotted Woodpecker (781 samples). Although Table 5.1 suggests that the model operates best for the Eurasian Skylark, this cannot be concluded because the accuracy was based on a relatively smaller test set. Hence, it was concluded that the model performs best on the Corn Bunting.

Table 5.1: *Validation Accuracy per species for 1000 samples each (∗ := less than 1000 samples).*

Species	Accuracy (%)	Species	Accuracy (%)
Eurasian Skylark*	96.6	Common Nightingale	73.9
Corn Bunting	84.5	Eurasian Tree Sparrow	71.9
Eurasian Blackcap	84.2	Great Spotted Woodpecker*	71.3
Tawny Owl	80.7	Coal Tit	68.8
Common Blackbird	80.0	Red Crossbill	67.9
Yellohammer	78.7	European Robin	62.8
Eurasian Wren	78.0	Great Tit	62.4
Common Chiffchaff	76.1	Common Chaffinch	58.5
Common Whitethroat	74.4	Song Thrush	48.2
Eurasian Blue Tit	73.9	-	-

Note that the number of samples used for the Eurasian Skylark were significantly low due to the unavailability in the dataset. Validation samples are desirably unique from the training samples, hence, training samples were not chosen to increase the size of the validation set for the Eurasian Skylark.

It is evident that the worst performing species was the Song Thrush with the lowest accuracy by over 10%. Further analysis was performed to detect any correlations with this result. Figure 5.1 depicts the confusion matrix for the results obtained.

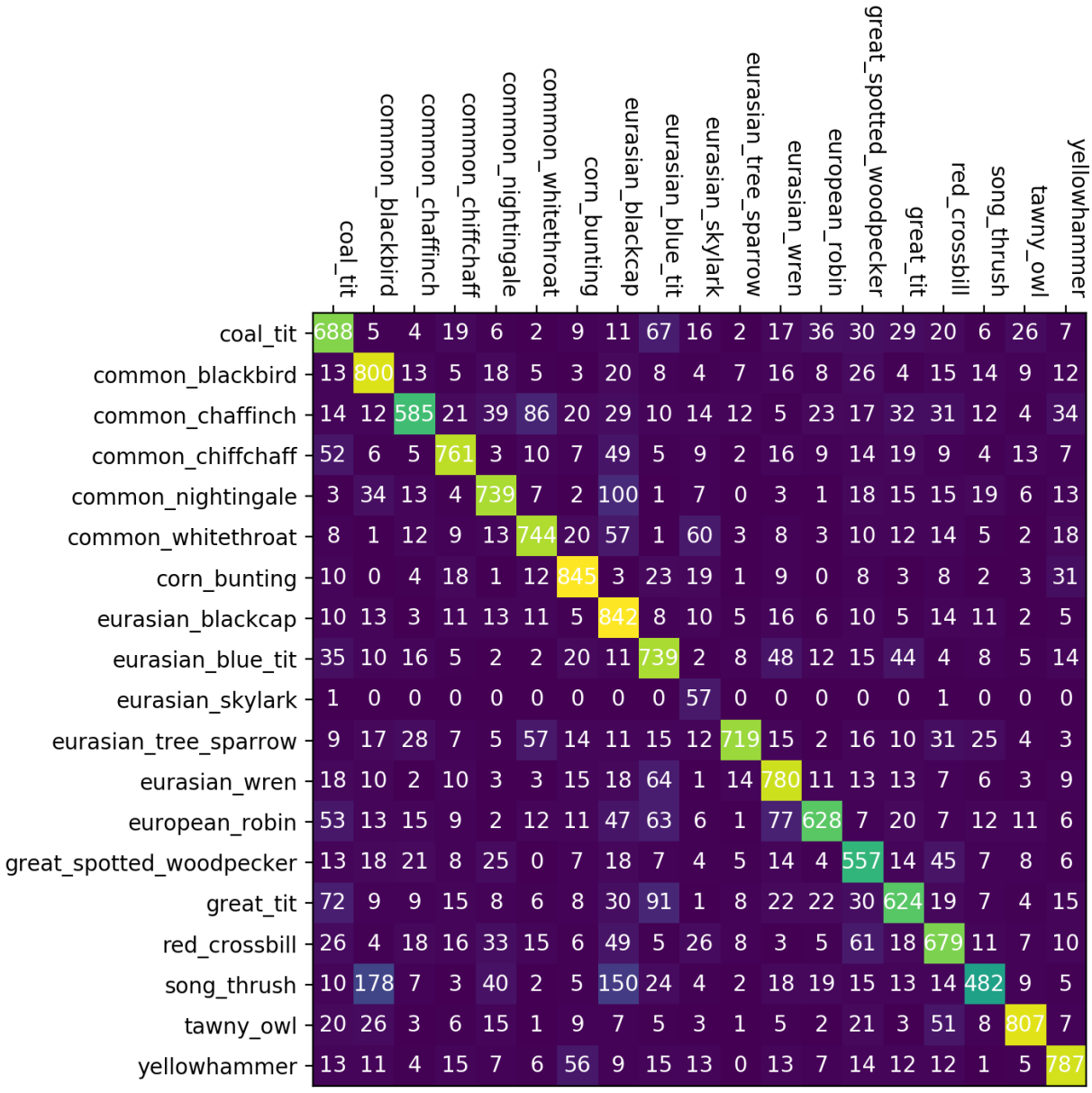

Figure 5.1: *Confusion Matrix for proposed model on* 17,840 *test samples.*

Observing the Song Thrush in the confusion matrix shows that it is mostly classified incorrectly as a Common Blackbird (178 times) and an Eurasian Blackcap (150). Moreover, the only other species that was incorrectly classified as one specific species at least a 100 times was the Common Nightingale. Despite the average accuracy of 73.9% for Common Nightingale, it was classified as an Eurasian Blackcap 10% of the time. The impact of the Eurasian Blackcap in incorrect classifications is interesting as 12.4% of the incorrect classifications were classified as Eurasian Blackcap. From this, 40.4% of samples were actually either a Song Thrush or Common Nightingale.

These results imply similarities between birdsongs for the Eurasian Blackcap, Song Thrush and Common Nightingale. This would be something to explore in further detail, although not pursued for this specific report. On the other hand, it could also insinuate that the model was unable to decipher distinct features for the Eurasian Blackcap, resulting in this ratio of incorrect classifications.

The above results can be investigated in further detail to analyse these specific birdsongs to determine the cause of this error. However, this was outside the scope of this project due to time constraints.

Chapter 6: Tools and Technologies

This chapter outlines the technologies and tools used in the development process of this project. A project of this magnitude requires integration of several different tools to achieve a successful outcome. The majority of the project was developed in Python 3 because of the diverse range of frameworks and libraries available.

1. **Deep Learning Library:** The CNN has been written using Keras [28]. Keras is a high level API built on top of TensorFlow [56] which made it very easy to implement the deep learning models with minimal amount of code. It is a pythonic library and made the building, training and testing of the neural network relatively simple.

2. **REST API service:** The REST API that allows communication with the trained model was developed in Python using the Flask framework [17].

3. **iOS Mobile Application:** The iOS application was written in Swift using XCode 11 [58].

4. **Containerisation:** The backend model and REST API was containerised using Docker [14].

5. **Libraries:**

   - **Audio Manipulation:** Pydub [44].
   - **Spectrogram Manipulation:** Matplotlib [25].
   - **Data handling:** NumPy [37] and Pandas [40].
   - **Mobile Application:** AVFoundation, RxSwift, RxAlamofire, SwiftUI.
   - Other various libraries for smaller tasks.

6. **Project Management:**

   - **Source Control:** The project was managed by two separate GitHub [21] repositories for the model and the mobile application.
   - **Development Lifecycle:** The project was managed in a Kanban [2] style with weekly sprints tracked using Ora.pm [38].

Chapter 7: Final Product

This chapter discusses the final product that was delivered as part of this project. So far this report has discussed in detail, the theory and implementation of the deep learning approach to solving automated birdsong recognition. However, the overall goal was to make this available to the general public. Therefore, the final product that was delivered as part of this project was an intuitive, user-friendly and easy to use iOS mobile application which can perform birdsong recognition. The application is completely self-contained with the ability to record, save and manage birdsongs from within the application.

7.1 Mobile Application

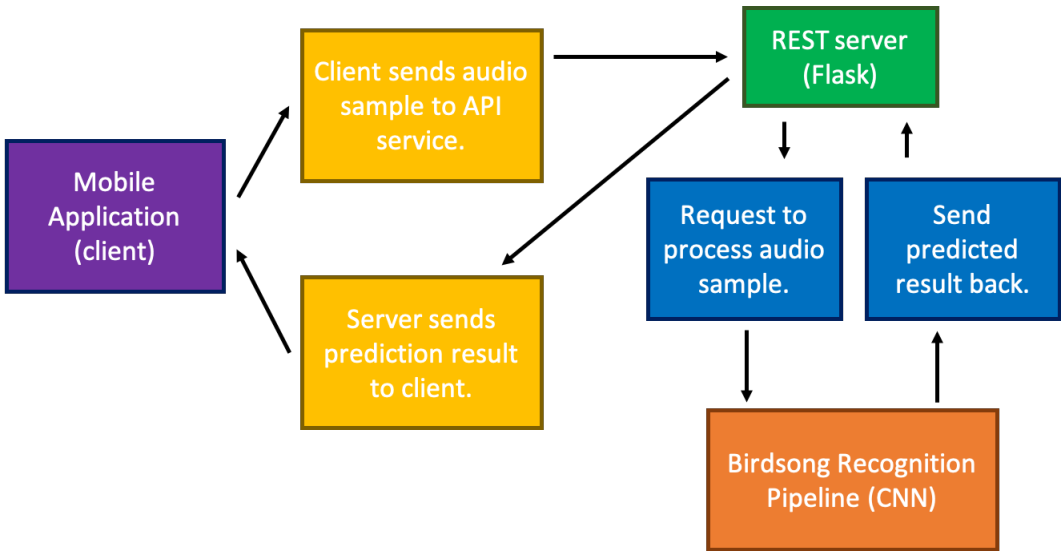

Figure 7.1: *Data flow diagram for the mobile application.*

7.1.1 Backend

The backend consists of three components: a web server, the birdsong recognition pipeline and the mobile application backend.

7.1.1.1 REST Service

The responsibility of the REST service was to allow the client to communicate with the birdsong recognition pipeline. The service was developed using the Flask

framework in Python 3. The requests consist of a Multi-part dataform [35] and the response is in a straight-forward JSON format, communicated between the client and server over HTTP.

The server receives a sample recorded on the client's device in M4A format, which is converted into MP3 and stored in a temporary location on the server, with an assigned UUID. The sample is fed into the birdsong recognition module, which generates a spectrogram image for the CNN to make a prediction. The result of the prediction is returned to the client in a JSON format (See Appendix A).

7.1.1.2 Birdsong Recognition Pipeline

(I) Image Processing:
When a model was trained, the specifications of the spectrogram parameters were stored in a file on the server. These parameters are loaded for a specific model and used to generate the spectrogram for the new audio sample which is sent from the client. This facilitated the ability to have multiple models that were trained with different hyperparameters, enabling to make use of the relevant 'active' model. Moreover, this ensured that the input from the client was in the same format as the samples used to train the model.

The spectrogram for the new samples is generated and passed onto the pre-trained CNN model.

(II) CNN Model:
Training a deep learning model can take a significant amount of time, which makes it necessary to be able to store the trained model to disk, in order to make predictions in the future.

All the models discussed in Chapter 4 were stored on the server, each in it's own directory. A trained and evaluated model was represented by a JSON file, along with the optimised weights which were stored in a HDF5 file [24]. The best model was chosen as the 'active model' which was stored in an environment variable on the server. The REST service accesses this variable to make use of the correct model for making predictions.

The chosen model is loaded from it's directory and is compiled in the CNN module. This model is used to make a prediction for the input image given by the image

processing module. This generates a JSON representing the prediction probabilities for each class label and feeds it back to the REST service. The module can return multiple predictions if the probabilities are close for multiple class labels as described in Section 3.1.7.

7.1.1.3 Mobile backend

The mobile application backend was written entirely in Swift 5 using XCode 11.

(I) Audio Recorder:
The application comes with an inbuilt audio recording feature which was developed using the AVFoundation library [3]. The application allows the recordings to be stored on the device which can later be accessed. This overcame concerns with connectivity issues when trying to predict a birdsong. The user may not have internet connection when they make a recording in the wild, hence the ability to store these samples in the application makes it convenient for the user to be able to classify these at a later time.

(II) Classifications:
As stated above, the classification is done on the server and not on the device itself. Using the RxSwift and RxAlamofire frameworks, the application can make requests to the REST service and handle the results in a Reactive fashion [43]. These libraries made handling asynchronous requests much simpler with tidier code.

7.1.2 Frontend

The frontend was developed using SwiftUI [55] in a 'declarative' manner. It made use of the Combine library [11] to manage data flow along with rendering the user interface. The application interface was designed using Sketch [51] before coding the interface with SwiftUI widgets.

7.1.2.1 User Interface

The main screen presents a list of recordings that have been made by the user. The user can listen to them and attain a classification for each of these. There is a record button which makes it accessible to record new samples very quickly and easily. Unwanted recordings can be deleted by simply swiping left on them. Figure

7.2 represents the home screen witnessed whilst employing the application.

A recording can be classified by tapping the search button on the recording tab. A request to the server is made and the application presents a 'Predictions' screen. This shows a list of all the predicted species along with a percentage of certainty (See Figure 7.3). To see further information about a species the user can tap on the name to view it's image, useful facts and a sample birdsong recording. This added a user-oriented feature to make the application more practical for public use (See Figure 7.4).

The 'Birds' page can be accessed from the top left of the screen which displays the full list of bird species that the application is able to classify (See Figure 7.4). Similarly to the predictions page, detailed information about each species can be viewed from here.

7.1.2.2 Application Screenshots

Below are screenshots from the iOS mobile application. A detailed video demonstration of the application can be accessed in the screencast.

Figure 7.2: *Application Home screen overview.*

Figure 7.3: *Prediction results for two samples. Single (left) and multiple (right) predictions.*

Figure 7.4: *Page displaying known birds (left) and detailed information page (right).*

Chapter 8: Conclusion

8.1 Summary

In this dissertation, the use of a deep neural network for birdsong recognition was examined. The existing methods to relevant problems were used as an inspiration to implement this approach. The task was reduced to an image recognition problem, using spectrograms generated from processed segments of recordings, sourced from Xeno-Canto. An automated process to retrieve and pre-process the data was implemented to accommodate efficiency concerns. Various techniques were explored to process the audio samples to suitably ready them for training input.

A considerable amount of research and experimentation in deep learning methods was carried out to achieve the final outcome of the project. Considerate decisions were made throughout the project to choose the most appropriate techniques and technologies to implement the convolutional neural network. Hyperparameters optimisation was carried out to improve the quality of results produced by the CNN.

The final model was trained on 19 European bird species (See Table 5.1) with 76,000 training samples. Experimental results demonstrate that the stated system achieves an accuracy of 72.1% on a validation set of 17,840 samples.

As an outcome of this project, an iOS Mobile Application with an intuitive and user-friendly interface was delivered and available for public use. The mobile application was self-contained and allowed the users to record and manage new samples from within the application.

Overall, this project was successful because it fulfilled the original objectives set out at the beginning of the project. In addition, it also presented a proof of concept for a scalable birdsong recognition solution, that could be made available to the general public. Even though the accuracy achieved may not have been considered remarkable compared to some existing approaches, the project made use of a unique technique to tackle the problem whilst making a meaningful contribution to the field of birdsong recognition.

8.2 Project Reflection

Despite the project being successful, the final solution posed some issues with both the model and the mobile application, preventing it from being an industry ready solution. For this to be ready for public use, the performance of the model would need to be improved. The results in Chapter 4 show that experiments were performed to improve the model by tuning several hyperparameters in the CNN. However, the structure of the layers in the CNN was unchanged due to time restrictions. All the trained models had 2 convolutional layers, 2 max pooling layers and 2 fully connected layers, yet the chosen layer structure was not necessarily optimal. Given more time, it would be worthwhile to analyse the performance with more convolutional layers along with a higher number of feature detectors. This is because a higher value yielded a greater accuracy, as reported by "Group Emotion Recognition with Machine Learning" [19]. Due to the computational time restrictions, this was not explored in the project.

In an automated birdsong recognition system, the accuracy and the number of classifiable species are the most important factors. It is clear that the proposed approach yielded auspicious results with somewhat reliable accuracy. However, the model had the ability to recognise only 19 European bird species. As has been shown in Section 1.2, the number of species was fairly low compared to some of the existing commercial applications. This could be improved by gathering more data samples for other species. Moreover, contrary to the decision to use an equal number of samples for each species, this is not necessary. Hence, more species could be added despite a lower number of samples available for them.

This project was also slightly restricted by the hardware available. The machine used to train the CNN did not possess a GPU, which would have been extremely helpful to accelerate the training process. The specification of the machine used can be seen in Appendix B. Therefore, if redoing this project, the deep learning model should be trained on a machine with a GPU. Currently, the trained model exists on a standalone machine which is made accessible to the mobile application via SSH. However, this is an issue that needs tackling in order to scale the solution to be industry ready.

Additionally, the mobile application developed in this project is rather a prototype. Although it provides basic functionality to serve the purpose of this project, it

requires feature enhancements before it can be publicly available. For example, currently the application allows the user to store the recordings within the application and get a classification for a selected recording. However, once the classification is presented, the result is not cached on the device. This means that the user requires connectivity to get a prediction for this recording again in the future.

8.2.1 Future Work

As discussed above, there are a lot of improvements that could be made in the future to make this an industry ready solution. The approach taken here has shown great promise for further development in terms of scalability. These enhancements apply to both the deep learning model and the mobile application. Further to the analytical investigations previously suggested, the following major improvements can be crucial to improve the outcome of this project.

Firstly, a rewarding enhancement would be to explore the optimal segment lengths for the audio samples. As discussed in Section 2.4.1, an experimental decision was taken to split all audio samples to make them at most 15 seconds long. If there was no time constraint, it would be immensely useful to perform experiments to find the optimum segment lengths for each bird species in the pre-processing step.

In addition to the existing approach, a major and non-trivial augmentation to the model would be to implement a hybrid machine learning system that would incorporate the deep neural network. As discussed in Section 2.2, the recordings gathered from Xeno-Canto contain the location information. Using this contextual metadata along with the additional data for average segment durations for each species, a similar approach can be applied to the one implemented in "Improving plankton image classification using context metadata" [15], to further improve the model. This is a practical approach to have for the mobile application; the GPS location data from the device can be used for new recordings made by users when predicting new samples. It is important to note the magnitude of this enhancement which could potentially be an entire project in itself.

Moreover, scaling the existing solution to a publicly available application requires the model to communicate with multiple clients (mobile application). In order to achieve this, it would be wise to perform the model computation and host the model on the cloud, via Amazon Web Services (AWS) or Google Cloud Platform (GCP).

Apart from the future improvements in the birdsong recognition model, it would also be appropriate to make enhancements to the mobile application to make it more useful and practical. Hence, it would be a convenient feature to cache the results from the server on the device. Furthermore, management of recordings can be improved by facilitating the ability to create custom folders. The user could then arrange recordings in separate folders in a more meaningful manner, whilst getting a more personalised experience. Finally, given more time, the application should be made cross-platform for both iOS and Android to increase accessibility for the general public.

8.3 Personal Reflection

This project was an extraordinary learning experience for me. I progressed from having very basic knowledge of Deep Learning methods in the beginning, to implementing a fully functioning CNN which gives me great pride. In addition to learning about the specifics of implementing neural networks, I was also able to explore a variety of technical areas including audio signal processing, image processing, building a web server and developing an iOS application. This project allowed me to acquire a range of skills which can be easily applied to a vast breadth of research projects or in the industry.

I believe this project helped me develop as a Computer Scientist by constantly raising challenges throughout its development. I learnt to stay resilient when conducting myself to overcome these obstacles. A project of this scale was always likely to pose various technical challenges, which I was able to tackle with research and the aid of several people with technical expertise.

This project has taught me many important lessons. The biggest lesson I have learnt is to always be curious. In a research based problem such as automated birdsong recognition, curiosity is a key component to be able to design a sophisticated system. Furthermore, time management and working under pressure are two valuable lessons gained from this project, that I will be able to apply both in my personal and professional life.

Overall, I have really enjoyed building this project and am extremely proud to have presented a fully functional prototype of my primary goal.

References

[1] Apoorva Agrawal. *Loss Functions and Optimization Algorithms. Demystified.* 2017. URL: https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c.

[2] Atlassian. *What is Kanban?* URL: https://www.atlassian.com/agile/kanban.

[3] *AVFoundation.* URL: https://developer.apple.com/av-foundation/.

[4] Abdul Badshah et al. "Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network". In: Feb. 2017, pp. 1–5. DOI: 10.1109/PlatCon.2017.7883728.

[5] Joseph C. Beaver. *Sonograms as Aids in Bird Identification.* URL: https://sora.unm.edu/sites/default/files/journals/nab/v030n05/p00899-p00904.pdf.

[6] *Bird Songs | Find out about Bird Calls in the UK – The RSPB.* URL: www.rspb.org.uk/birds-and-wildlife/bird-songs/.

[7] *BirdGenie - ID Birds by their songs.* URL: https://www.birdgenie.com.

[8] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.* 2017. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[9] Jason Brownlee. *How to use Data Scaling Improve Deep Learning Model Stability and Performance.* 2019. URL: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/.

[10] *Chirpomatic - Automatic birdsong identifier app.* URL: www.chirpomatic.com.

[11] *Combine.* URL: https://developer.apple.com/documentation/combine.

[12] *Convolutional Neural Networks (CNNs/ConvNets).* URL: https://cs231n.github.io/convolutional-networks/.

[13] Adit Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks.* 2016. URL: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/.

[14] *Docker.* URL: www.docker.com.

[15] Jeffrey S. Ellen, Casey A. Graff, and Mark D. Ohman. *Improving plankton image classification using context metadata.* 2019. URL: https://doi.org/10.1002/lom3.10324.

[16] Kirill Eremenko, Hadelin de Ponteves, and SuperDataScience Team. *Machine Learning A-Z: Hands-On Python and R In Data Science*. URL: https://www.udemy.com/course/machinelearning/.

[17] *Flask*. URL: https://flask.palletsprojects.com/en/1.1.x/.

[18] Fodor. *Fodor, Gábor (2013). "The Ninth Annual MLSP Competition: First Place".*

[19] Samanyou Garg. *"Group Emotion Recognition with Machine Learning", Third year project report. School of Computer Science, University of Manchester.* 2019.

[20] Cara Giaimo. *The Loudest Bird in the World Has a Song Like A Pile Driver*. 2019. URL: https://www.nytimes.com/2019/10/21/science/loudest-bird-bellbird.html.

[21] *Github*. URL: https://github.com.

[22] Hervé Goëau et al. *LifeCLEF Bird Identification Task 2014. CLEF: Conference and Labs of the Evaluation Forum*. 2014. URL: https://hal.inria.fr/hal-01088829/document.

[23] Greg Green. *Seeing sound: What is a Spectrogram?* 2018. URL: https://blogs.bl.uk/sound-and-vision/2018/09/seeing-sound-what-is-a-spectrogram.html.

[24] *HDF Group*. URL: www.hdfgroup.org.

[25] John Hunter et al. *Matplotlib*. 2012. URL: www.matplotlib.org.

[26] BirdLife International. *Birds are found almost everywhere in the world, from the poles to the equator*. 2013. URL: http://datazone.birdlife.org/sowb/casestudy/birds-are-found-almost-everywhere-in-the-world-from-the-poles-to-the-equator.

[27] Kaggle. *MLSP 2013 Bird Classification Challenge | Kaggle*. URL: www.kaggle.com/c/mlsp-2013-birds/.

[28] *Keras*. URL: www.keras.io.

[29] Diederik P. Kingma and Jimmy Lei Ba. *Adam: A Method for Stochastic Optimization*. 2015. URL: https://arxiv.org/pdf/1412.6980.pdf.

[30] Carolyn Kormann. *The Real Bird Genie*. 2015. URL: www.newyorker.com/tech/annals-of-technology/the-real-bird-genie.

[31] Kathryn Kovitvongsa and Phillip S. Lobel. *Convenient Fish Acoustic Data Collection in the Digital Age - Scientific Figure on ResearchGate.* URL: https://www.researchgate.net/figure/Spectrograms-and-Oscillograms-This-is-an-oscillogram-and-spectrogram-of-the-boatwhistle_fig2_267827408.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012), p. 3. DOI: 10.1145/3065386.

[33] Gregory J. Kunkel. *About sonagrams.* 2007. URL: http://www.bio.umass.edu/biology/kunkel/gjk/songrm.htm.

[34] Mario Lasseck. *Large-scale identification of birds in audio recordings - Notes on the winning solution of the LifeCLEF 2014 Bird Task.* 2014. URL: http://ceur-ws.org/Vol-1180/CLEF2014wn-Life-Lasseck2014.pdf.

[35] *MDN web docs.* URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST.

[36] Brian Moyer and Griffin Brown. *Understanding Spectrograms.* 2020. URL: https://www.izotope.com/en/learn/understanding-spectrograms.html.

[37] *NumPy.* URL: www.numpy.org.

[38] Ora. *Ora - Project Management System.* URL: https://ora.pm/home.

[39] Jake Palmer. *"Bird Song Recognition". 3rd year project report. School of Computer Science, University of Manchester.* 2016.

[40] *Pandas.* URL: www.pandas.pydata.org.

[41] Parul Pandey. *Understanding the Mathematics behind Gradient Descent.* 2019. URL: https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e.

[42] C. John Ralph et al. *Monitoring Bird Populations by Point Counts.* 1995. URL: https://www.fs.fed.us/psw/publications/documents/psw_gtr149/psw_gtr149.pdf.

[43] *ReactiveX.* URL: www.reactivex.io.

[44] James Robert. *Pydub.* 2019. URL: www.pypi.org/project/pydub/.

[45] Victor Rodrigues. *"Automatic Song Recognition". Third year project report. School of Computer Science, University of Manchester.* 2017.

[46] Stacey Ronaghan. *Deep Learning: Which Loss and Activation Functions should I use?* 2018. URL: https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8.

[47] Sebastian Ruder. *An overview of gradient descent optimization algorithms.* 2016. URL: https://ruder.io/optimizing-gradient-descent/.

[48] Sabyasachi Sahoo. *Deciding optimal kernel size for CNN.* 2018. URL: https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363.

[49] Sagar Sharma. *Activation Functions in Neural Networks.* 2017. URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[50] Kevin Shen. *Effect of batch size on training dynamics.* 2018. URL: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e.

[51] *Sketch.* URL: www.sketch.com.

[52] Julius Smith. *Spectral Audio Signal Processing.* Jan. 2008.

[53] *Songs and Calls.* URL: https://ornithology.com/ornithology-lectures/songs-calls/.

[54] Dan Stowell and Mark D. Plumbley. *Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning.* 2014. URL: https://peerj.com/articles/488/.

[55] *SwiftUI.* URL: https://developer.apple.com/documentation/swiftui.

[56] *TensorFlow.* URL: www.tensorflow.org.

[57] Warblr. *Warblr - The Birdsong Recognition app.* URL: www.warblr.co.uk.

[58] *XCode.* URL: https://developer.apple.com/xcode/.

[59] Xeno-Canto. *Xeno-Canto - Sharing bird sounds from around the world.* URL: www.xeno-canto.org.

Appendix A: JSON Schemas

Listing A.1: *Xeno-Canto API response example.*

```
{
    "numRecordings": "1",
    "numSpecies": "1",
    "page": 1,
    "numPages": 1,
    "recordings": [
        ...,
        Array of Recording objects (see below),
        ...
    ]
}
```

Listing A.2: *Recording Object Xeno-Canto.*

```
{
    "id": "477551",
    "en": "Eurasian Wren",
    "cnt": "Poland",
    "lat": "52.6907",
    "lng": "23.6035",
    "alt": "160",
    ...,
    "url": "\/\/www.xeno-canto.org\/477551",
    "file": "\/\/www.xeno-canto.org\/477551\/download",
    ...,
    "time": "08:00",
    "date": "2019-05-03",
    "uploaded": "2019-05-29",
    ...,
}
```

Listing A.3: *JSON response format returned from the server.*

```
{
    "result": {
        "birdA": 54,
        "birdB": 45,
    }
}
```

Appendix B: Machine Specifications

CPU Model	Intel® Core™ i5-2400
CPU Architecture	x84_64
CPU MHz	3400
CPU Cores	4
RAM Type	DDR3
RAM GB	8
OS	Ubuntu 18.04.4 LTS