INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Identificação automática de aves a partir de áudio

SILVESTRE DANIEL DIAS CARVALHO
Julho de 2020

POLITÉCNICO
DO PORTO

Automatic Bird Identification from Audio

Silvestre Daniel Dias Carvalho

Dissertation to obtain a Master's degree in Informatics Engineering, Area of Specialisation in Computer Systems

Supervisor: Elsa Ferreira Gomes

Porto, July 5, 2020

Abstract

Bird classification from audio is mainly useful for ornithologists and ecologists. With growing amounts of data, manual bird classification is time-consuming, which makes it a costly method.

Birds react quickly to environmental changes, which makes their analysis an important problem in ecology, as analyzing bird behaviour and population trends helps detect other organisms in the environment.

A reliable methodology that automatically identifies bird species from audio would be a valuable tool for the experts in the area.

The main purpose of this work is to propose a methodology able to identify a bird species by its chirp.

There are many techniques that can be used to process the audio data, and to classify the audio data. This thesis explores the deep learning techniques that are being used in this domain, such as using Convolutional Neural Networks and Recurrent Neural Networks to classify the data. Audio problems in deep learning are commonly approached by converting them into images using feature extraction techniques such as Mel Spectrograms and Mel Frequency Cepstral Coefficients.

Multiple deep learning and feature extraction combinations are used and compared in this thesis in order to find the most suitable approach to this problem.

Keywords: Bird Audio Classification, Deep Learning, Audio Feature Extraction

Resumo

Classificação de pássaros a partir de áudio é principalmente útil para ornitólogos e ecologistas. Com o aumento da quantidade de dados disponível, classificar a espécie dos pássaros manualmente acaba por consumir muito tempo.

Os pássaros reagem rapidamente às alterações climáticas, o que faz com que a análise de pássaros seja um problema interessante na ecologia, porque ao analisar o comportamento das aves e a tendência populacional, outros organismos podem ser detetados no meio ambiente.

Devido a estes factos, a criação de uma metodologia que identifique a espécie dos pássaros fiavelmente seria uma ferramenta bastante útil para os especialistas na área.

O objetivo principal do trabalho nesta dissertação é propor uma metodologia que identifique a espécie de uma ave através do seu canto.

Existem diversas técnicas que podem ser usadas para processar os dados sonoros que contêm os cantos das aves, e que podem ser usadas para classificar as espécies das aves. Esta dissertação explora as principais técnicas de *deep learning* que são usadas neste domínio, tais como as redes neuronais convolucionais e as redes neuronais recorrentes que são usadas para classificar os dados.

Os problemas relacionados com som no *deep learning*, são normalmente abordados por converter os dados sonoros em imagens utilizando técnicas de extração de atributos, para depois serem classificados utilizando modelos de *deep learning* tipicamente utilizados para classificar imagens. Dois exemplos destas técnicas de extração de atributos normalmente utilizadas são os *Espectrogramas de Mel* e os *Coeficientes Cepstrais da Frequência de Mel*.

Nesta dissertação, são feitas múltiplas combinações de técnicas de deep learning com técnicas de extração de atributos do som. Estas combinações são utilizadas para serem comparadas com o âmbito de encontrar a abordagem mais apropriada para o problema.

Contents

List of Figures

List of Tables

List of Source Code

List of Acronyms

AHP	Analytic Hierarchy Process.
API	Application Programming Interface.
AWS	Amazon Web Service.
BFCC	Bark Frequency Cepstral Coefficients.
CI	Consistency Index.
cmAP	Classification Mean Average Precision.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
CR	Consistency Ratio.
CRNN	Convolutional Recurrent Neural Network.
FFT	Fast Fourier Transform.
GFCC	Gammatone Frequency Cepstral Coefficients.
GPU	Graphics Processing Unit.
GRU	Gated Recurrent Unit Networks.
LP	Linear Predictive.
LSTM	Long Short Term Memory.
MFCC	Mel-Frequency Cepstral Coefficients.
NCD	New Concept Development.
PLP	Perceptual Linear Predictive Coefficients.
RAM	Random Access Memory.
ReLU	Rectified Linear Units.
RI	Random Index.
RNN	Recurrent Neural Network.
RPLP	Revised Perceptual Linear Prediction.
TF	TensorFlow.

Chapter 1

Introduction

This chapter presents some points to introduce this dissertation such as the context, the problem, the objective, the expected results, the value analysis the approach and the document structure.

1.1 Context

The *Xeno Canto* is an online community that has a database of bird sounds recordings from all over the world (*Xeno-Canto* 2020). The provided dataset has a subset of labeled recordings from California and Nevada, USA. The dataset has recordings of 91 species with 30 samples from each one, making it 2730 samples in total (Hiatt 2020).

1.2 Problem

With the continuing threat of climate change, the analysis of interactions between organisms and their environment is an important problem in ecology (Martinsson 2017). By analyzing bird behavior and population trends, other organisms in the environment can be detected because birds react quickly to environmental changes (Gavali et al. 2019).

Bird identification can be done manually by experts in the area, either though audio or images (Gavali et al. 2019), but the growing amounts of data makes this process tedious and time-consuming, meaning it requires a lot of human effort, making it a costly method (Martinsson 2017).

Therefore, a reliable methodology that automatically identifies birds from audio would be a valuable tool for experts in the area.

1.3 Objective

The objective of this thesis is to propose a methodology able to identify a bird species by its chirp.

In this thesis it is intended to explore deep learning techniques that are being applied in this domain in order to get the best results.

1.4 Expected results

The expected result developing this project is a reliable and efficient method of classifying birds though the sound of their chirps. To analyze the method reliability, the average *recall* rate for all species will be used.

1.5 Value Analysis

Some birdwatchers rely on visibly seeing the bird and knowing what a certain species looks like to be able to classify the bird species. Some may be able to identify a species by its chirp, but any new species they have to research through bird sound databases and find the correct bird with the same chirps. Most bird audio databases also rely on the users inputting the bird audio to correctly identify the bird species.

The method resulting from the project of this dissertation could be used to create a service to help birdwatchers and bird audio databases to correctly identify bird species.

There is a more in-depth approach to value analysis in section 2.2.

1.6 Approach

For the development of this project, related and relevant work previously developed was researched in chapter 3. With the information obtained, multiple approaches were presented to choose from.

The selected approach was deep learning using *TensorFlow*. Five deep learning models to test and compare, such as Convolutional Neural Network (CNN), two Recurrent Neural Network (RNN) variants and two Convolutional Recurrent Neural Network (CRNN) variants. Also, two feature extraction methods were selected, the Mel-Frequency Cepstral Coefficients (MFCC) and Mel Spectrogram.

These deep learning models and feature extraction methods are combined to then be optimized, experimented and compared.

The approach is analysed in-depth in chapter 4.

1.7 Document Structure

This thesis is divided by six chapters. After this introductory chapter, the second chapter is the Context. In that chapter, the problem will be contextualized and its value analysed. The third chapter is the State of the Art, it is where related and relevant work is documented, as well as some technologies that will be used. The fourth chapter is the Approach Evaluation, where the options researched in the third chapter are compared and selected to be used in chapter five. The fifth chapter is the Design of the Solution, it is where the design of the solution is presented. The sixth chapter is the Evaluation, which is where the solution is tested.

Chapter 2

Context

This chapter describes the dataset used for this dissertation and the value analysis.

2.1 Dataset

The dataset used for this work contains a subset of recordings, labeled by species, from California and Nevada, USA (dataset from Hiatt 2020).

It contains 91 species, with 30 sound sample files per species. In total there are 2730 samples, ranging from less than 1 second to 195 seconds. The sum of the duration of all samples from all of the species is 20 hours, 25 minutes and 8 seconds (73508 seconds).

Tables 2.1 and 2.2 list all the species in the dataset, and the percentage of the duration relative to the total duration of all species samples.

Table 2.1: Bird Species list and Duration Percentage (1)

Bird Species	Percentage	Bird Species	Percentage
Abert's Towhee	0.75%	Acorn Woodpecker	0.48%
American Bushtit	0.89%	American Crow	1.15%
American Dusky Flycatcher	1.66%	American Grey Flycatcher	1.67%
American Robin	2.27%	Anna's Hummingbird	0.57%
Ash-throated Flycatcher	1.55%	Bell's Sparrow	0.66%
Bell's Vireo	1.04%	Bewick's Wren	0.52%
Black Phoebe	0.71%	Black-chinned Sparrow	2.15%
Black-headed Grosbeak	1.46%	Black-tailed Gnatcatcher	0.78%
Blue-grey Gnatcatcher	1.28%	Brewer's Sparrow	2.12%
California Gnatcatcher	0.65%	California Quail	0.85%
California Scrub Jay	0.42%	California Thrasher	0.99%
California Towhee	0.63%	Canyon Wren	1.17%
Cassin's Finch	1.17%	Cassin's Vireo	1.83%
Chestnut-backed Chickadee	1.31%	Clark's Nutcracker	0.91%
Common Poorwill	0.82%	Common Yellowthroat	1.46%
Dark-eyed Junco	0.76%	Elegant Tern	0.48%

Table 2.2: Bird Species list and Duration Percentage (2)

Bird Species	Percentage	Bird Species	Percentage
Flammulated Owl	1.70%	Forster's Tern	0.95%
Great Horned Owl	1.47%	Green-tailed Towhee	1.18%
Grey Vireo	1.42%	Hermit Thrush	1.22%
Hermit Warbler	2.29%	House Finch	1.24%
House Wren	0.71%	Hutton's Vireo	0.82%
Juniper Titmouse	1.70%	Lark Sparrow	2.55%
Lazuli Bunting	1.90%	Lesser Goldfinch	1.21%
Lincoln's Sparrow	1.82%	Long-eared Owl	1.62%
MacGillivray's Warbler	1.01%	Marsh Wren	0.85%
Mountain Chickadee	0.51%	Mountain Quail	1.04%
Northern Flicker	0.93%	Northern Mockingbird	2.11%
Northern Pygmy Owl	0.93%	Northern Raven	0.64%
Northern Saw-whet Owl	0.41%	Nuttall's Woodpecker	0.33%
Nutting's Flycatcher	0.90%	Oak Titmouse	0.64%
Orange-crowned Warbler	0.92%	Pacific Wren	0.91%
Pacific-slope Flycatcher	0.64%	Phainopepla	1.32%
Pygmy Nuthatch	1.07%	Red Crossbill	0.25%
Red-winged Blackbird	1.54%	Ridgway's Rail	0.78%
Rock Wren	2.09%	Rufous-crowned Sparrow	1.13%
Slate-colored Fox Sparrow	1.90%	Snow Goose	0.78%
Song Sparrow	0.55%	Spotted Owl	0.72%
Spotted Towhee	0.50%	Steller's Jay	0.51%
Swainson's Thrush	1.93%	Thick-billed Fox Sparrow	0.85%
Tricolored Blackbird	0.32%	Verdin	0.77%
Warbling Vireo	1.44%	Western Meadowlark	0.99%
Western Screech Owl	0.99%	Western Wood Pewee	1.13%
White-breasted Nuthatch	0.62%	White-crowned Sparrow	1.04%
White-headed Woodpecker	1.28%	Wilson's Warbler	0.93%
Wrentit	0.58%	Yellow-billed Magpie	0.65%
Yellow-breasted Chat	1.64%		

The figure 2.1 presents the sum of the sample duration in seconds for each bird species in the dataset. It shows the variation of the available data from species to species, with the lowest being 185 seconds and the highest 1877 seconds.

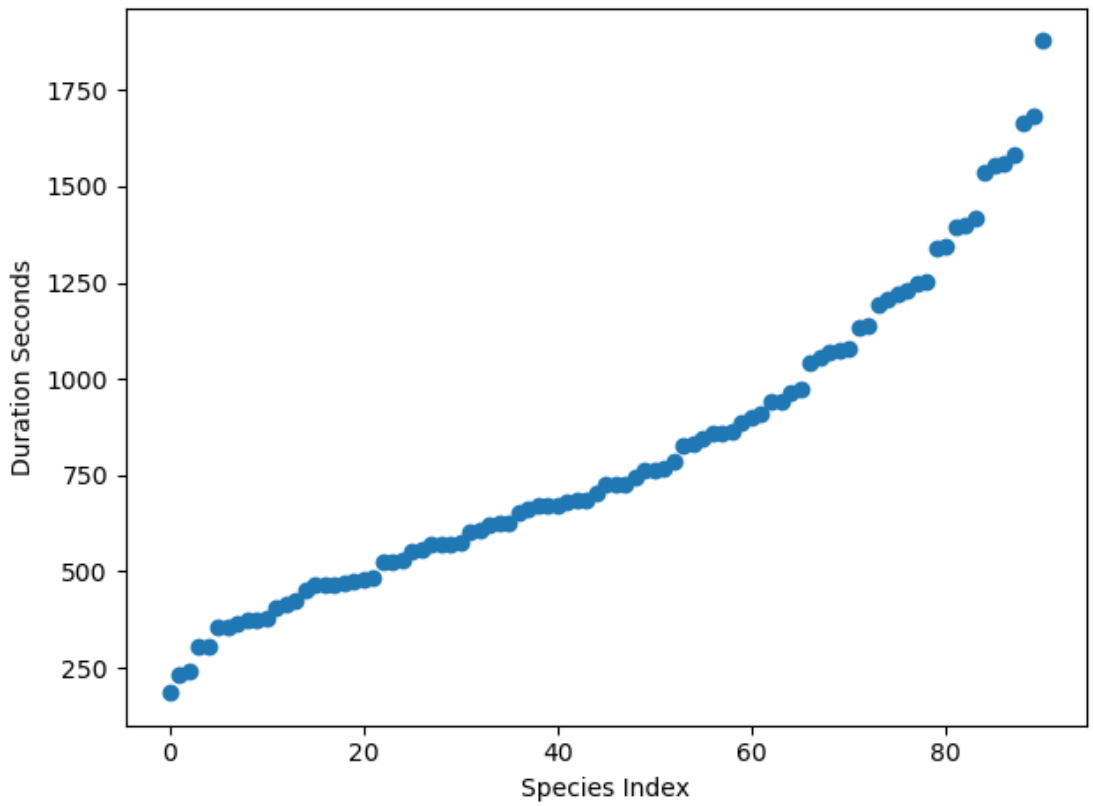


Figure 2.1: Scatter chart of the sample duration for each species

2.2 Value Analysis

2.2.1 New Concept Development (NCD)

Peter Koen (Koen et al. 2002) developed a model, the New Concept Development (NCD), in which there are 5 key elements: the identification and analysis of the opportunity, the generation, enrichment and selection of ideas, and the concept definition.

Each one of these elements are analysed below, following with the effective methods, tools and techniques (Koen et al. 2002) to analyze each element.

Opportunity Identification

A bird classification method could be useful for anyone who hears a bird singing and wonders what species it is. It could also be useful for bird sound databases (e.g. Xeno-canto) to automatically verify if an audio file of a bird species uploaded by one of its users was correctly classified.

The classification method could be implemented into a service or an application that these people could use to help identify a bird species by recording a sound clip of the bird singing.

Effective Methods, Tools and Techniques (Koen et al. 2002):

- Road-mapping;
- Technology trend analysis;
- Customer trend analysis;
- Competitive intelligence analysis;
- Market research;
- Scenario panning.

Opportunity Analysis

Recently, deep learning models normally used for visual recognition contexts have been used to classify sounds. It seems like it is a viable solution to the sound classification problem as it has outperformed other approaches (Piczak 2015).

This means that a bird classification method could use deep learning models.

Effective Methods, Tools and Techniques:

Many of same methods, tools and techniques used in the opportunity identification are used in this element as well, such as road-mapping, technology trend analysis, competitive intelligence analysis, customer trend analysis, and scenario planning. In opportunity identification, these tools were used to determine if an opportunity existed. In this element, considerably more resources are expended, providing more detail on the appropriateness and attractiveness of the selected opportunity (Koen et al. 2002).

Idea Generation and Enrichment

There are multiple deep learning models that can be applied to this audio classification problem, the most prominent should be selected to be researched and tested.

On the audio classification problem, the audio data that feeds the deep learning model should be pre-processed and have its relevant features extracted. Pre-processing and feature extraction methods should be researched and tested.

Effective Methods, Tools and Techniques (Koen et al. 2002):

- Discovering the archetype of the customer (Archetype research identifies the unstated "reptilian" or instinctive part of the brain);
- Identifying new technology solutions;
- An organizational culture that encourages employees to spend free time testing and validating their own and others' ideas.
- A variety of incentives to stimulate ideas;
- Inclusion of people with different cognitive styles on the idea enrichment team;

Idea Selection

After generating and enriching ideas, the ones that standout after researching should be selected for the development of a solution.

The selected technologies for the development of the solution are described throughout the document.

Effective Methods, Tools and Techniques (Koen et al. 2002):

- Portfolio methodologies based on multiple factors (not just financial justification) using anchored scales, such as technical success probability, commercial success probability, reward, strategic fit, strategic leverage;
- Formal idea selection process with prompt feedback to the idea submitters;
- Use of options theory to evaluate projects.

Concept Definition

The concept of the product to be developed is a method capable of classifying a bird species by its chirp sound. The development will be done using deep learning techniques, audio pre-processing and feature extraction.

The development of this method could help people or databases to correctly classify a bird species.

Effective Methods, Tools and Techniques (Koen et al. 2002):

- Carefully defining the project goals and outcomes;
- Setting criteria for the corporation that describe what an attractive (in terms of financial, market growth, market size, etc.) project looks like;
- Rapid evaluation of high-potential innovations;
- Early involvement of the customer in real product tests (even before the product is completed);
- Understanding and determining the performance capability limit of the technology;
- Pursue alternative scientific approaches.

2.2.2 Value

Any business activity is essentially about value exchange. Value exchange is defined as the delivery of a good or service, and having its value received and rewarded by a peer or customer, either within or outside one's company. Value has been defined in different theoretical contexts as need, desire, interest, standard, attitudes and preferences, which makes it very dependent on perception (Nicola, E. P. Ferreira, and J. J. P. Ferreira 2012).

Value for the Customer

Value for the customer is the personal perception of the the customer regarding the advantages arising out of the association with an organization's offering. These advantages can occur as a reduction in sacrifice, the presence of benefits, and the result of the weighed combination of sacrifices and benefits (Woodall 2003).

Perceived Value

Different customers perceive different value for the same products/services. In addition, organizations involved in the purchasing process can have different perceptions of customers' value delivery (Ulaga and Eggert 2006).

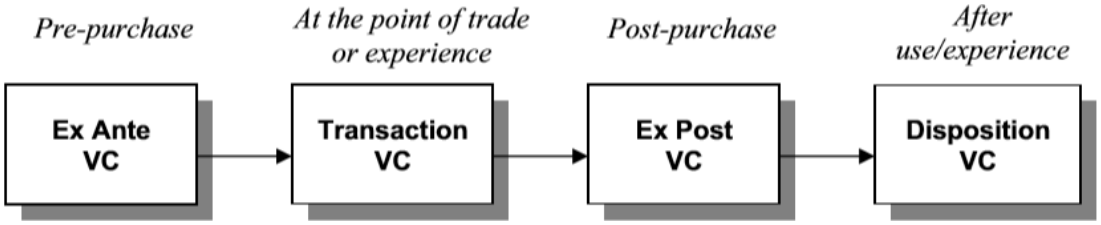


Figure 2.2: A Longitudinal Perspective on Value for the Customer (from Woodall 2003)

A longitudinal perspective with the benefits and sacrifices can be used to interpret the value for the customer. It contains four temporal values:

- Pre-purchase: desires and preconceived ideas of the customer regarding the value of the product;
- Transaction: customer value defined in the moment of the transaction, acquisition or exchange;
- Post-purchase: phase in which the customer evaluates his purchase;
- After use: reflects the usage experience and the period in which the sale or exchange is done by the customer.

The classification of bird species through audio could help people correctly assume a bird species, which in effect could remove outliers from bird databases.

It could also get more people interested in birdwatching, because birds could be easily recognized by recording a sample of them singing and the classification method would tell them the species.

Some rare birds could also be identified using the classification method, if the classification model is trained with the species.

Table 2.3 shows the benefits and sacrifices for each temporal value that customers of this method would experience.

Table 2.3: Longitudinal Value Perspective

	Benefits	**Sacrifices**
Pre-purchase	Convenience Customisation Knowledge	Price
Transaction		Acquisition costs
Post-purchase	Convenience Customisation	*Dataset* gathering time Deep learning model training time
After use	Convenience Customisation Knowledge	

2.2.3 Value Proposal

A value proposal describes the difference between a company's offer and those of its competitors, explaining why customers buy from the company (Lindic and Da Silva 2011).

A value proposition is about the customer but intended for the company's use, it defines exactly what the organization intends to provide to the customer's life (Lanning 2014). It defines how an organization works by focusing the activities on serving their customers on the best way possible, while maintaining profits (Barnes, Blake, and Pinder 2009).

The product intended to be developed is a method for the classification of bird species by audio. This method should have a high accuracy rate in order to present reliable classifications to its users.

The users of this method could either be any person that records a bird singing and wonders what species it is, or a bird sound database company looking to validate the classification of its own sounds.

There are existing bird species classification implementations available. This product will differentiate itself by testing multiple deep learning models, as well as explore pre-processing and feature extraction methods to raise the accuracy rate.

2.2.4 Canvas Business Model

A business model describes the logic behind of how a company creates, delivers and captures value (Osterwalder and Pigneur 2010).

A business idea that could be associated to the theme of this dissertation is the commercialization of a web service using the bird species classification method.

The users would upload an audio file of a bird singing and the service would return them the species of the bird.

The users could be the general public (birdwatchers) accessing the service through a web browser or an application. An Application Programming Interface (API) would also be available for bird sound databases to use.

The table 2.4 is a visualization of the proposed Canvas Model in order to create a business model with the theme of this dissertation.

Table 2.4: Canvas Business Model

<table>
<tr><th>Key Partners</th><th>Key Activities</th><th>Value Propositions</th><th>Customer Relationships</th><th>Customer Segments</th></tr>
<tr><td rowspan="3">Bird Sound Databases</td><td>Software development; Dataset collection</td><td rowspan="3">Bird species classification; Ease of use and convenience; High species classification accuracy</td><td>Feedback; Technical support</td><td rowspan="3">Birdwatchers; Bird Sound Databases</td></tr>
<tr><td>Key Resources</td><td>Channels</td></tr>
<tr><td>Know-how; Deep Learning Framework</td><td>Birdwatching Communities</td></tr>
<tr><td colspan="3">Cost Structure</td><td colspan="2">Revenue Streams</td></tr>
<tr><td colspan="3">Service development and maintenance; Hardware cost (Servers)</td><td colspan="2">Advertising revenue for the general public (birdwatchers); Subscription for Bird Sound Databases</td></tr>
</table>

2.2.5 Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP), created by T. L. Saaty 1988, is a multi-criteria decision method that uses numeric techniques to help decision makers to select an option out of multiple alternatives.

Developing the solution to this dissertation requires choosing or building a deep learning architecture.

In this section, three deep learning architectures will be analyzed: *LeNet* (LeCun et al. 1998), *AlexNet* (Krizhevsky, Sutskever, and Hinton 2012) and *GoogLeNet* (Szegedy et al. 2015).

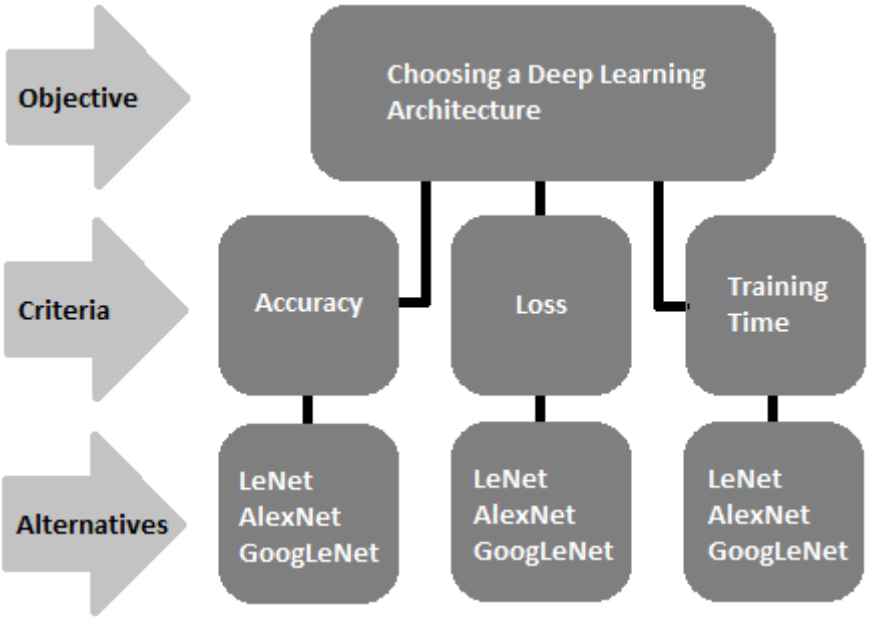


Figure 2.3: Hierarchical Decision Tree

Figure 2.3 shows the hierarchical decision tree. The objective is at the first hierarchical level, followed by three criteria at the second level and at the third level the three alternatives.

The data used for this analysis is from Shah, Bakarola, and Pati 2018, where research was done with the objective of getting the optimal approach for image recognition using a deep convolutional architecture". The specific test is testing the three architectures using the *MNIST* digit dataset on an Amazon Web Service (AWS) Graphics Processing Unit (GPU) instance with an Intel Xeon E5 Central Processing Unit (CPU) with 15GB of Random Access Memory (RAM), an NVIDIA K520 GPU with 3072 CUDA cores and 8GB GPU memory (Shah, Bakarola, and Pati 2018).

The decision criteria are:

- Accuracy: Percentage of correct predictions
- Loss: Metric used to rank and compare each training epoch (Lower is better)
- Training Time: Time that the architecture takes to fit the dataset

With the analysis presented in the appendix A, it was concluded that the selected alternative to be used as the deep learning architecture is "AlexNet", as it got the best result (57.78%) compared to the other alternatives, in this specific analysis with the chosen criteria and importance.

Chapter 3

State of the Art

This chapter presents the state of the art on the topics related to this thesis, such as the Deep Learning for Sound Classification, where multiple deep learning model architectures and the typical techniques used to evaluate these models are described. Some Deep Learning Frameworks are presented, as well as some of the most relevant Audio Processing techniques, which includes pre-processing and feature extraction techniques. Finally, some Existing Approaches are also overviewed.

3.1 Deep Learning for Sound Classification

Deep Learning is a sub-field of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks (Brownlee 2019c).

The typical deep learning approach for classification of sounds is to convert the audio file to an image, such as a spectrogram, and then use a neural network to process the image (Boddapati et al. 2017).

The following sub-sections are overviews over these deep learning classification models, and the evaluation methods typically used for classification models.

3.1.1 Convolutional Neural Network

An architecture that fits the audio classification problem is the Convolutional Neural Networks (CNN), which is a deep learning architecture that learns through images. It was inspired by the natural visual perception of the living creatures (Gu et al. 2015).

A CNN is a deep learning architecture which can take an input image, assign importance to multiple details in the image, such as weights and biases, and be able to differentiate one from another. Its architecture is analogous to the connectivity pattern of the neurons in the human brain, being inspired by the organization of the visual cortex (Saha 2018).

How it works

A CNN works by splitting its neurons into a three-dimensional structure. Each set of neuron analyzes a small region or feature of the image, specializing themselves in identifying one part of the image. CNN use the predictions from the layers to produce a final output of an

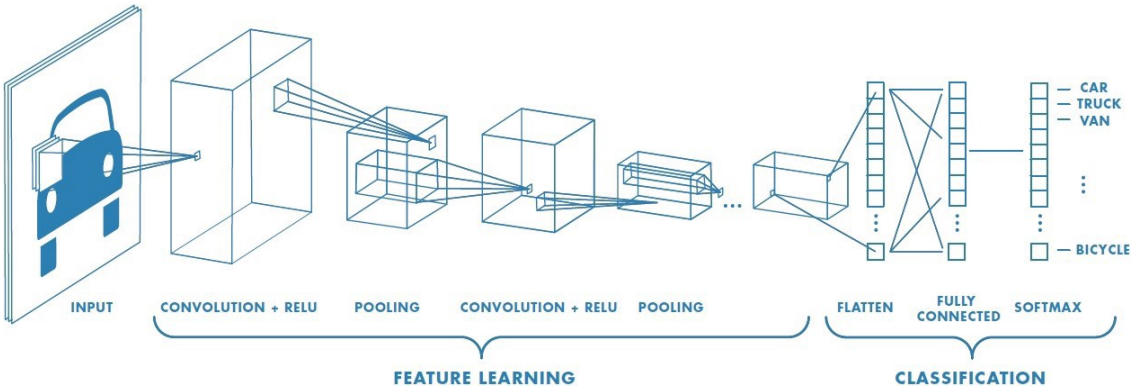


Figure 3.1: A Convolutional Neural Network representation (Chatterjee 2019)

array of probability scores that represent the likelihood that a specific feature in an image belongs to a certain class (MissingLink.ai 2019).

An image is an array of pixel values to a computer. These values are the only inputs available to the computer to learn how to perform image classification. The idea behind image classification using deep learning is that, given the array of values (pixels in an image), the deep learning model will output values describing the probability of the image being a certain class (Deshpande 2019). In the case of this thesis and as an example, when presented a spectrogram of a bird singing, it could result in something in the likes of 80% for Sage Sparrow species, 15% for Nuttall's Woodpecker species, 5% for Hermit Warbler species.

Layers

- Convolutional Layer

  The first layer in a CNN is a Convolutional Layer. The convolutional layer creates a feature map to predict the class probabilities for each feature. It does it by applying a filter, also known as kernel, which scans the whole image a few pixels at a time (MissingLink.ai 2019).

  The purpose of the convolution operation is to extract the high-level features from the input image. A CNN may have multiple convolutional layers. Typically, the first convolutional layer is responsible for capturing the low-level features such as edges, color, gradient orientation, etc. Adding more convolutional layers makes the network adapt to the high-level features, creating a network which has a deeper understanding of the images in the dataset, similar to how a human would by taking in the details of a picture (Saha 2018).

  Figure 3.2 is a representation of how a convolutional layer processes the input neurons (on the first convolutional layer these contain the pixel values from the input image). The 32 x 32 array represented in the figure has a 5 x 5 area selected, called the receptive field. What is selecting the area is usually called filter or kernel, it is an array of numbers containing weights or parameters. As the filter slides, or convolves, around the input array, it is multiplying the values in the filter with the original pixel values of the image. All of the multiplications in the filter are summed, creating values for the hidden layer. This process goes on for every position of the input volume, moving the

filter by 1 unit. After the filter has slid over all the locations, the resulting 28 x 28 array, the "first hidden layer", is the activation or filter map (Deshpande 2019).

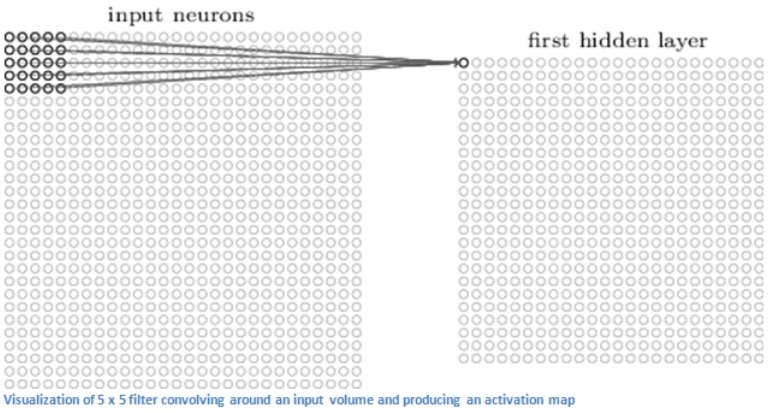


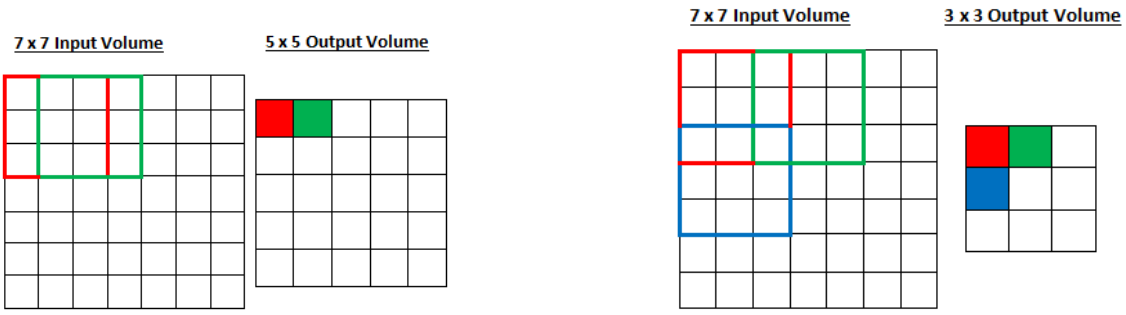
Figure 3.2: 5x5 Filter producing an Activation Map (Deshpande 2019)

– Stride

The stride controls how the filter convolves around the input volume. It is the amount by which the filter shifts, and is normally set in a way so that the output volume is an integer and not a fraction (Deshpande 2019).

Changing the stride has an effect on how the filter is applied to the image (input) and on the resulting size of the activation map (output) (Brownlee 2019a).

Higher stride values result in lower resolution outputs, also known as downsampling. Downsampling can be useful in cases where deeper knowledge of the filters used in the model or of the model architecture allow for some compression in the resulting feature maps (Brownlee 2019a).



(a) Stride value of 1 (b) Stride value of 2

Figure 3.3: Different values of Stride (Deshpande 2019)

Figures 3.3a and 3.3b both have a 7 x 7 input volume and a 3 x 3 filter (The third dimension was disregarded for simplicity). The figure 3.3a with the stride value of 1 has a resulting output volume of 5 x 5, while figure 3.3b with the stride value of 2 is only 3 x 3. This happens because the lower stride value has to process more values, as it has to step 1 by 1 both horizontally and vertically through the whole matrix. As mentioned before and as it can be seen in this example, higher stride value results in a downsampled output (Brownlee 2019a).

– Padding

As convolutional layers are applied, the size of the output volume will keep decreasing. In the early layers, the preservation of information about the original input volume is important so that low-level features can be extracted (Deshpande 2019).

Taking a 32 x 32 input volume and applying a 5 x 5 filter to it, using a stride value of 1, results in an output volume of 28 x 28. The spatial dimensions decrease (Deshpande 2019).

To keep spatial dimensions to remain the same as the input, *zero padding* can be applied. *Zero padding* means surrounding the input volume with zeros around the border (Deshpande 2019).

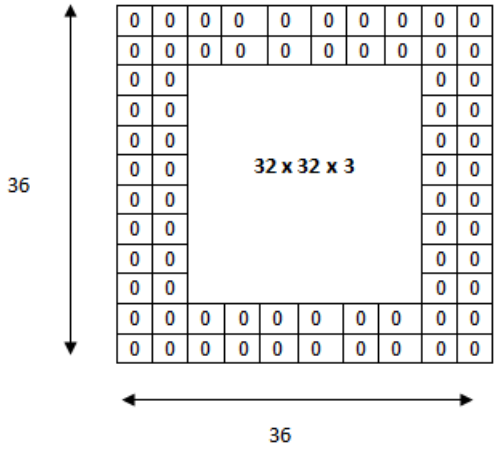


Figure 3.4: Padding value of 2 on 32 x 32 input (Deshpande 2019)

As it can be seen in figure 3.4, applying a padding of 2 on the original input volume of 32 x 32, results in an input volume of 36 x 36. After applying the 5 x 5 filter with the stride value of 1, at the end of the convolution the output volume will be 32 x 32 (Deshpande 2019).

- Activation Layer

  After each convolutional layer, it is usual to apply an activation layer. The purpose of this layer is to introduce nonlinearity to a system that computed linear operations during the convolutional layers (element multiplications and summations). It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers (Deshpande 2019).

  – Rectified Linear Units (ReLU)

    The ReLU applies the function f(x) = max(0, x) to all of the values in the input volume, which in basic terms means that it just changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the convolutional layer. Researches found that ReLU layers work better when compared to *tanh* or *sigmoid* because the network is able to train faster as it is more computationally efficient, without a significant difference to the accuracy (Deshpande 2019).

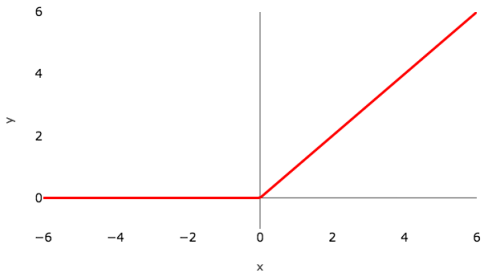


Figure 3.5: Rectified Linear Units Activation (Jain 2020)

There are also variations of the ReLU, such as *Leaky ReLU*, *Parametric ReLU* and *ReLU6*, which can solve some issues when using normal ReLU (Jain 2020).

– *Sigmoid*

The *sigmoid* activation applies the function f(x) = 1/(1+e $n^{(-x)}$). It is considered computationally expensive, and it causes vanishing gradient problem and it is not zero-centered (Jain 2020). The vanishing gradient problem occurs because the input in converted between 0 and 1, which makes their derivatives much smaller when compared to ReLU, which does not have a small derivative problem (Kumawat 2020).

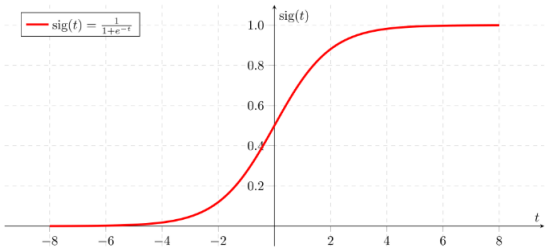


Figure 3.6: Sigmoid Activation (Jain 2020)

– *Hyperbolic Tangent Activation Function (Tanh)*

This activation applies the function f(x) = 2/(1+e $n^{(-2x)}$). It is a scaled *sigmoid* function, and solves non zero-centered problem as it ranges between -1 and 1 (Jain 2020).

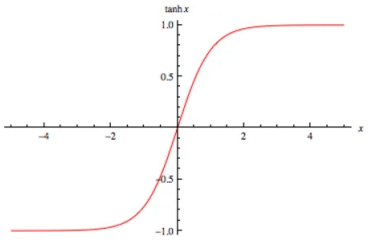


Figure 3.7: Tanh Activation (Jain 2020)

– *Softmax*

The softmax is used in binary and multi-class classification problems in the final layer, also known as the output layer (Jain 2020).

- Pooling Layer

  Pooling layers are referred to as a down-sampling layer (Deshpande 2019). The pooling layer is responsible for reducing the spatial size of the convolved feature. It is used to decrease the computational power required to process the data through dimensionality reduction (Brownlee 2019a).

  – Max Pooling: Returns the maximum value from the area of the image covered by the kernel. It also performs as a noise suppressant by discarding the noisy activations altogether, also denoising along with dimensionality reduction (Saha 2018).

  – Average Pooling: Returns the average of all of the values from the area of the image covered by the kernel. It only performs dismensionality reduction as a noise suppression mechanism (Saha 2018).

  Because of the noise suppression characteristics, max pooling performs better than average pooling (Brownlee 2019a).

  Figure 3.8 is a representation of the max pooling and average pooling layers.

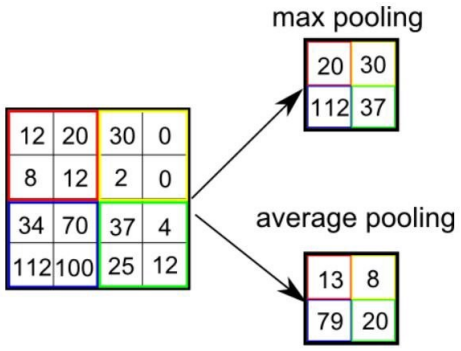


Figure 3.8: Types of pooling with a stride value of 2 (Brownlee 2019a)

- Dropout Layer

  The dropout layer *removes* a random set of activations in that layer by setting them to zero, forcing the network to be redundant, as the network should be able to provide the right classification even if some of the activations are removed (Deshpande 2019).

  The purpose of the dropout layer is to help alleviate the overfitting problem. This layer should only be used during training, and not during testing (Deshpande 2019).

- Fully Connected Layer

  The last layers in a CNN are fully connected layers. The input to the fully connected layer is the output from the final pooling or convolutional layer, which is flattened and then fed into the fully connected layer (Arunava 2018).

  – Flattening

    The output from the final (and any) pooling and convolutional layer is a 3-dimensional matrix. This layer needs to be flattened into a vector (1-dimension).

The flattened vector is then connected to some fully connected layers. These fully connected layers are the same as artificial neural networks and perform the same mathematical operations (Arunava 2018).

Figure 3.9 represents the flattening of an array into a vector.

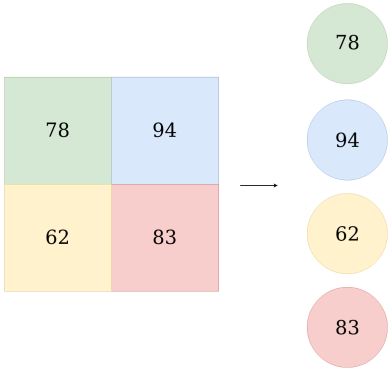


Figure 3.9: Flattening (Arunava 2018)

After the fully connected layers, Arunava 2018 uses the *softmax* activation function instead of ReLU in the final layer, obtaining the probabilities of the input belonging to each different class.

Figure 3.10 is a representation of a fully connected network.

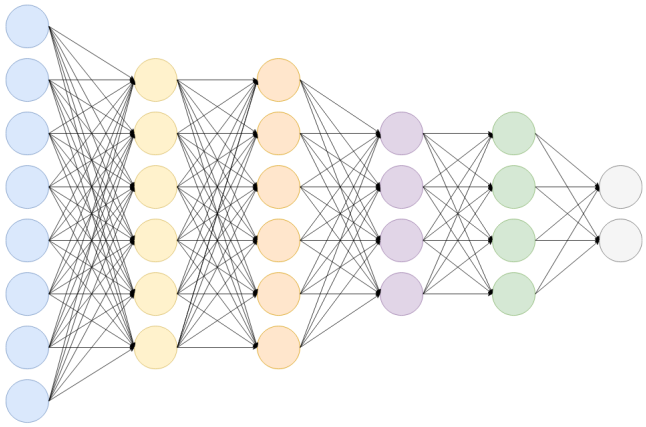


Figure 3.10: A fully connected network (Arunava 2018)

3.1.2 Recurrent Neural Network

RNN are designed to recognize patterns in sequences of data. The algorithms used in RNN take time and sequence into account, meaning they have a temporal dimension. RNN are applicable to images, which can be decomposed into a series of patches and treated as a sequence (Nicholson 2019).

RNN are networks with loops in them, which allows information to persist, and thus, they keep context of past network iterations in consideration (Olah 2019).

A recurrent network can be imagined as multiple copies of the same network, each one passing a message to its successor (Olah 2019).

It can be seen in the right side of figure 3.11, where the RNN loop is *unrolled* to help visualize what happens in these loops. On the left side of the figure it has an input 'x_t', an output 'h_t' and the RNN itself 'A' (Olah 2019).

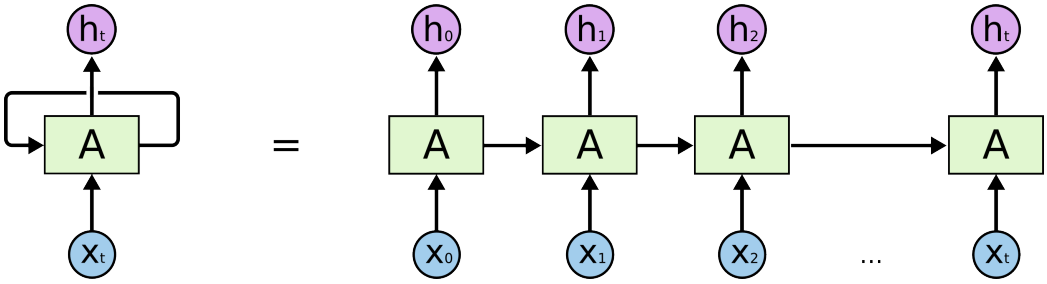


Figure 3.11: A recurrent neural network visualization (Olah 2019)

- Vanishing gradient problem (Short-term Memory)

  The vanishing gradient problem in RNN refers to the network suffering from short-term memory. When the sequence of recurrence is long enough, the network has trouble maintaining information from earlier time steps to the later ones. For example, processing an audio sample, RNN may leave out important information from the beginning of that sample (Nguyen 2019).

  During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. In RNN layers that get a small gradient update stop learning, usually the early layers. As these layers do not learn, the networks can forget what it learnt in longer sequences, thus having short-term memory (Nguyen 2019).

Long Short Term Memory Networks

Long Short Term Memory (LSTM) networks are a modified version of recurrent neural networks. These networks were created with the intent of fixing the short-term memory problem (Nguyen 2019).

Regular RNN have a vanishing gradient problem, where they *forget* information gradually. LSTM were designed with the intent to overcome these error back-flow problems (Hochreiter and Schmidhuber 1997).

LSTM are capable of learning long-term dependencies, retaining long-term temporal features in audio classification may help to improve the accuracy of the bird classification given that context is kept. LSTM remember information for long periods of time as their default behaviour as they are explicitly designed to avoid the long-term dependency problem (Olah 2019).

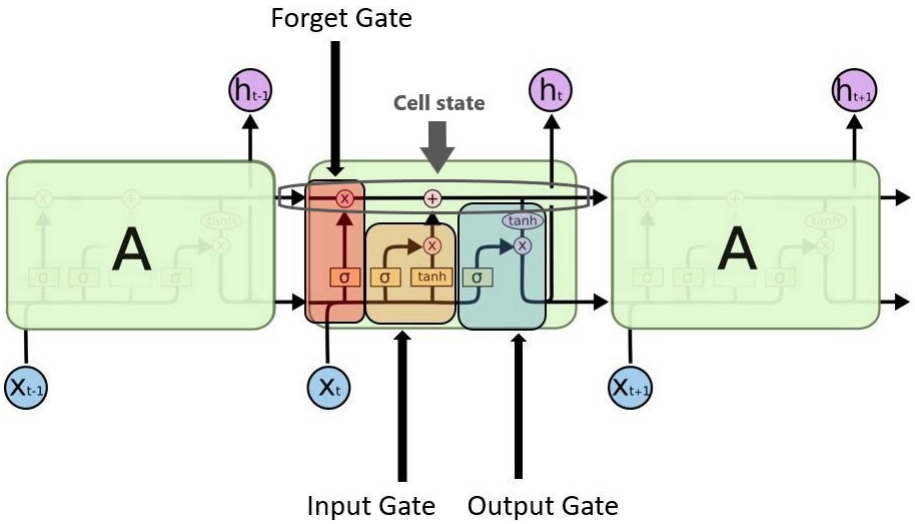


Figure 3.12: Long Short Term Memory (Mittal 2019)

- Gates
  - Input gate The input gate selects which value from the input should be used to modify the memory. The *sigmoid* function decides which values to let through, and the *tanh* function weights the values which are passed, deciding their level of importance raging from -1 to 1 (Mittal 2019).
  - Forget gate

    The forget gate selects which details are to be discarded from the volume. This is decided by the *sigmoid* function. Using the previous state (h_t-1) and the content input (X_t) it outputs a number between 0 (undo this) and 1 (keep this) for each number in the cell state (C_t-1) (Mittal 2019).
  - Output gate

    The input and the memory of the block is used to decide the output. The *sigmoid* function decides which values to let through to the next phase, and the *tanh* function gives weightage to the values which are passed deciding their level of importance ranging from -1 to 1 and multiplied with output of *sigmoid* function (Mittal 2019).
- Cell state

  The cell state can be seen on figure 3.12, it is the uppermost horizontal line.

  It is one of the core concepts of LSTM, along with its other gates. It acts as a transport highway that transfers relative information all the way through the sequence chain (Nguyen 2019).

  An analogy that can be used for it is that it is the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory (Nguyen 2019).

  A new cell state is created by first point-wise multiplying the cell state by the forget vector, which has a possibility of dropping values in the cell state if it gets multiplied

by values close to 0. Then the output is taken from the input gate and a point-wise addition is done, which updates the cell state to new values that the neural network finds relevant (Nguyen 2019).

Gated Recurrent Unit Networks

Introduced by Cho et al. 2014, Gated Recurrent Unit Networks (GRU) aim to solve the vanishing gradient problem which comes with a standard RNN. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce similar results (Kostadinov 2019).

To solve the vanishing gradient problem, GRU uses the update gate and reset gate. These vectors decide what information should be passed to the output, and can be trained to keep information from the early network, without forgetting them through time. (Kostadinov 2019).

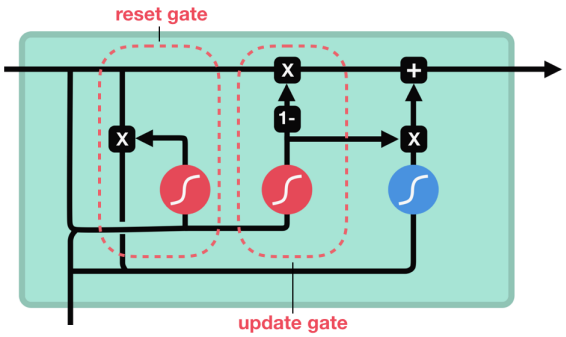


Figure 3.13: Gated Recurrent Unit Network (Nguyen 2019)

Compared to LSTM, GRU have a reduced number of parameters without compromising accuracy, which results in faster convergence and a more generalized model. This is the main reason behind the usage of GRU over LSTM in practical scenarios. (Perambai 2019).

- Gates
  - Update Gate

    The update gate determines how much of the past information needs to be passed along. This is powerful because the model can maintain all the information from the past, eliminating the risk of the vanishing gradient problem (Kostadinov 2019).
  - Reset Gate

    This gate is used for the model to decide how much information to forget (Kostadinov 2019).

3.1.3 Convolutional Recurrent Neural Network

A CRNN is a hybrid between CNN and RNN.

CNN have been combined with RNN, they are often used to model sequential data such as audio signals or word sequences. A CRNN is a modified CNN with the last convolutional

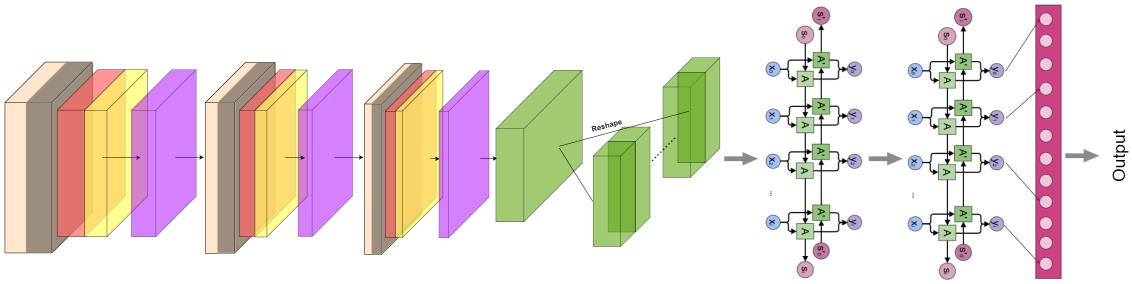


Figure 3.14: Convolutional Recurrent Neural Network (Chatterjee 2019)

layers replaced with a RNN. In CRNN, CNN and RNN play the roles of feature extractor and temporal summarizer, respectively. Adopting an RNN for feature aggregation enables the networks to take the global structure into account while local features are extracted by the remaining convolutional layers (Choi et al. 2016).

Choi et al. 2016 found that CRNN show a strong performance relative to the number of parameters and training time, which indicates the effectiveness of the hybrid structure in their music feature extraction and feature summarization experience.

3.1.4 Evaluating a Deep Leaning Classification Model

The evaluation of a deep learning model is a crucial part of this dissertation.

Evaluating a Machine Learning model is usually done like this:

- Using Hold-out

  Hold-out, also known as a train-test split, works by splitting the data into two parts. A training set and a validation set. The deep learning model is trained (or fit) using the training set, and predicting using the validation set, evaluating the accuracy of the predictions (Brownlee 2019b).

  In this thesis, each element of the training and validation set would contain a spectrogram (X variable, bird sound) and a label (Y variable, bird species). The training set will both variables to train the model. The validation set will only use the spectrogram to get a bird species prediction from the model, and then the label will be used to validate if the prediction is correct.

  Hold-out is a good approach to use when there is a lot of data or a slow model to train, although the resulting accuracy score for the model will be noisy because of randomness in the data, as the same model fit on different data will give different model accuracy scores (Brownlee 2019b).

- Using *k-Fold* Cross Validation

  The *k-Fold* cross validation splits the available data into *k-folds*, fitting the model on *k-1* folds, evaluating it on the held out fold, and repeating this process for each fold. The result is *k* different models that have k different sets of predictions, and in turn, *k* different accuracy scores (Brownlee 2019b).

This method generates a population of accuracy scores, which is useful as a mean can be taken to report the average expected performance of a model, as it makes it closer to the actual performance of the model in practice (Brownlee 2019b).

Deep learning models are stochastic, meaning that they use randomness while being fit on a dataset, such as random initial weights and random shuffling of data during each training epoch using stochastic gradient descent (Brownlee 2019b). Stochastic gradient descent is used in order to reduce the computational burden in the optimization of a deep learning model, instead of calculating a gradient for the entire dataset, a random subset of data is selected to create an estimate for the gradient (Srinivasan 2019).

This means that when a model is fit on the same dataset multiple times, it may give different predictions, affecting the accuracy scores. The randomness makes the model more flexible when learning, but makes the model less stable. (Brownlee 2019b).

- Fix the Random Seed

  Randomness can be set to be the same every time the model is fit. This is done by fixing the random number seed used by the model to generate weights and data shifts between epochs (Brownlee 2019b).

  This method is good when the same result is needed every time a model is run, although it is not a good practice as it is fragile and not a recommended method for evaluating models (Brownlee 2019b).

```
scores = list()
for i in k:
  train, test = split_old(data, i)
  model = fit(train.X, train.y)
  predictions = model.predict(test.X)
  skill = compare(test.y, predictions)
  scores.append(skill)
```

Listing 3.1: Fixing the Random Seed (Brownlee 2019b).

- Repeat Evaluation Experiments

  Repeating evaluation experiments consists in running the training and prediction phases repeatedly, usually 30 or more times according to Brownlee 2019b. This makes it so a standard error of the mean model accuracy can be calculated, which is how much the estimated mean of model accuracy varies from the actual mean model. This standard error indicated how wrong a mean accuracy of a new model might be (Brownlee 2019b).

  The standard error can be used to calculate a confidence interval for the mean accuracy. This assumes that the distribution of the results is Gaussian, which can be checked with an histogram, Q-Q plot, or using statistical tests on the collected scores (Brownlee 2019b).

```
for i in repeats:
  run_scores = list()
  for j in k:
    train, test = split_old(data, j)
    model = fit(train.X, train.y)
    predictions = model.predict(test.X)
    skill = compare(test.y, predictions)
    run_scores.append(skill)
  scores.append(mean(run_scores))

standard_error = standard_deviation / sqrt(count(scores))
interval = standard_error * 1.96
lower_interval = mean_skill - interval
upper_interval = mean_skill + interval
```

Listing 3.2: Repeat Evaluation Experiments (Brownlee 2019b).

Evaluation Metrics

This section presents the common evaluation metrics used for classification models.

- Confusion Matrix

  A confusion matrix is not exactly an evaluation metric, but it is important to get a general sense on how the model is doing.

  A confusion matrix is a tabular visualization of the model predictions versus the ground-truth labels. Each row of the confusion matrix represents the instances in a predicted class, and each column represents the instances in an actual class (Minaee 2019).

		Actual Class	
		Class A	Class B
Predicted Class	Class A	90	60
	Class B	10	940

Figure 3.15: A sample confusion matrix (edited, from: Minaee 2019)

  In the figure 3.15 can be seen, as example, a confusion matrix for 2 classes As it can be seen in figure 3.15, the diagonal elements of this matrix denote the correct prediction for each class, while the off-diagonal elements denote the samples which are misclassified (Minaee 2019).

  - True Positive: Represents the quantity of predictions from a class that were correctly classified as that class.
  - False Positive: Represents the quantity of predictions from a class that were incorrectly classified as that class.
  - False Negative: Represents the quantity of predictions from a class that were incorrectly classified as another class.

- Accuracy

The classification accuracy is defined as the number of correct predictions divided by the total number for predictions, multiplied by 100. When the class distribution of a dataset is imbalanced, classification accuracy is not a good indicator for a model's performance (Minaee 2019).

Classification accuracy is defined as:

$$Accuracy = \frac{CorrectPredictions}{TotalPredictions} * 100 \tag{3.1}$$

Using figure 3.15 as an example, there are 1000 Class B samples and 100 Class A samples. In this example, even if all the samples are predicted as the most frequent class, a high accuracy rate would be obtained. That is not accurate as it is not an indication that the model is actually learning, as it can be just predicting everything as the most common class (Minaee 2019).

- Precision

  Precision is a class specific performance metric. It is the fraction of samples from a class which are correctly predicted by a model, with the samples incorrectly classified as that class (Minaee 2019).

  Precision is defined as:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{3.2}$$

- Recall

  Recall, also known as Sensibility, is another class specific performance metric. It is the fraction of samples from a class which are correctly predicted by a model, with the samples incorrectly classified as another class (Minaee 2019):

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3.3}$$

- F1 Score

  F1 is a metric that combines Precision and Recall.

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.4}$$

3.2 Deep Learning Frameworks

A deep learning framework is an interface, library or tool which provides a clear and concise way to create deep learning models without having to get into the details of the underlying algorithms behind the deep learning models (Sharma 2019).

3.2.1 TensorFlow

TensorFlow (TF) was created by Google, with version 1.0 releasing in February 2017. It is arguably the most popular Deep Learning framework today. Gmail, Uber, Airbnb, Nvidia and lots of other prominent brands using it (Kharkovyna 2019).

Python is the language used to work with TF, although there are experimental interfaces for JavaScript, C++, Java, Go, C# and Julia. TF has the ability to work on computing clusters as well as running models on mobile platforms like iOS and Android (Kharkovyna 2019).

TensorBoard is TF's visualization toolkit, which enables tracking experiment metrics, visualizing the model graph and much more.

Starting with TF 2.0 Keras is the high level API for TF and it is extended so that all the advanced features of TF can be used directly from *tf.keras*. Eager execution is now the default, enabling debugging and prototyping faster than previous versions. One of the downsides of TF prior to version 2.0 was the use of a static graph as opposed to a dynamic graph, as competitors such as PyTorch use (Montantes 2019).

From *Keras - TensorFlow Core* 2020, three key advantages:

- User-friendly: Keras provides a simple and consistent interface optimized for common use cases, providing clear and actionable feedback for user errors.
- Modular and composable: Keras models are made by connecting configurable building blocks together, with few restrictions.
- Easy to extend: Custom building blocks can be written to express new ideas for research. New layers, metrics, loss functions and state-of-the-art models can be created.

TensorFlow allows deploying the deep learning models on one or more CPU, as well as on one or more GPU.

3.2.2 PyTorch

PyTorch was created by Facebook, with version 1.0 releasing in October 2018. It was developed for Facebook services but it is also used by companies such as Twitter and Salesforce. It is based on Torch, which is another deep learning framework based on Lua (Kharkovyna 2019). PyTorch is better suited to prototyping and small projects than TF, but when it comes to cross-platform solutions, TF is a more suitable choice (Kharkovyna 2019).

Standard debuggers are also available for PyTorch.

PyTorch is not on the supported backend list for Keras (from *Backend - Keras Documentation* 2019).

Simillar to TensorFlow, PyTorch allows the usage of multiple CPU and GPU.

3.2.3 Keras

Keras is an open-source neural-network library written in Python. It was initially released in March 2015, and its primary author and maintainer is François Chollet, a Google engineer.

It can be used as a high-level API on top of other popular lower level libraries such as TF. Keras is also known for its ease on prototyping as creating massive models of deep learning in Keras is reduced to single-line functions. But this strategy makes Keras a less configurable environment than low-level frameworks (Kharkovyna 2019).

Keras is ideal to learn and prototype simple concepts, as well as to understand the essence of the various deep learning models and their process of learning, as coding is much more readable and succint than other alternatives. Serialization and Deserialization API, callbacks and data streaming using Python generators is also very mature for Keras (Kharkovyna 2019).

Keras is not comparable to TF or PyTorch because it is on the higher level of abstraction, while the other two are on the lower level.

3.3 Audio Processing

3.3.1 Pre-processing

Signal pre-processing

The amplitude of each sound signal should be normalized to the same level to simplify further processing (Huang et al. 2009).

In the pre-processing of anuran calls done by J. G. Colonna et al. 2012, the first step was to standardize the signal, including the normalization of the signal and making it have zero mean (J. G. Colonna et al. 2012).

Kortas 2020 stated that since birds sing in high frequencies, a high pass filter could be applied to remove noise (Kortas 2020).

Syllable segmentation (J. Colonna et al. 2016)

The syllable segmentation algorithm was used by Fagerlund 2007 to classify bird songs, and by Huang et al. 2009 and J. G. Colonna et al. 2012 to classify frog sounds.

After signal pre-processing, the signal is padded with zeros with the width equal to the length of the half segment - β. The syllable segmentation is done by the following algorithm, where α is the threshold for the normalized signal (ranging between 0 and 1) and β is half of the length of the segment (J. Colonna et al. 2016).

The syllable segmentation algorithm has the following steps (from J. Colonna et al. 2016):

1. Find the maximum absolute value S(t) of the signal;
2. If S(t) < α go to step 5;
3. Select β seconds to the right and to the left of the peak, which gives us one syllable;
4. Extract the syllable from step 3 and replace the values in the signal with randomized small numbers to simulate noise. Go to step 1;
5. End.

The syllable extraction is illustrated in figure 3.16.

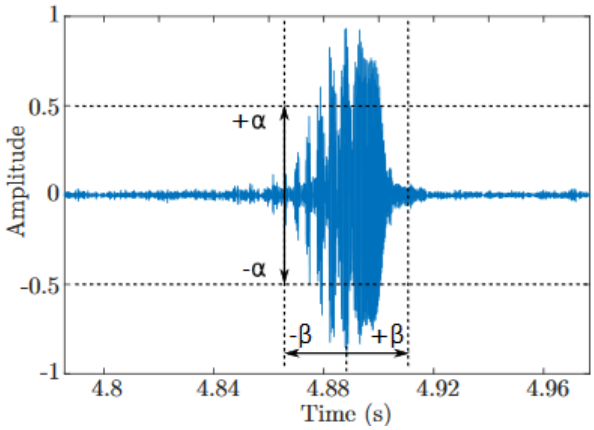


Figure 3.16: Example of syllable segmentation (J. Colonna et al. 2016)

According to Kortas 2020, the lengthier the audio from which a spectrogram is created, the more information is retained on the image, but the deep learning model could also become overfit[1]. If the data has a lot of noise or silence, there is a chance that short audio clips (up to 5 seconds) will not catch the needed information. Therefore, longer audio clips (up to 10 seconds) can be used, which in the Kortas 2020 experiment increased the model accuracy by 10% (Kortas 2020).

Therefore, if the Kortas 2020 method of segmenting data works with the syllable segmentation method, the β seconds from the J. Colonna et al. 2016 algorithm could be from 5 to 10 seconds.

3.3.2 Feature extraction

Choosing the appropriate signal features is crucial for the process of bird species recognition, as the acoustical environment of bird audio recordings contain noise in the signal, and there is a big diversity among bird species songs (Wielgat et al. 2007).

Mel Spectrogram

A Mel Spectrogram is a spectrogram where the frequencies are converted to the mel scale.

The mel scale was proposed by Stevens, Volkmann, and Newman 1937.

It is a perceptual scale of pitches such that equal distances in pitch sound equally distant to the listener. An example is that while humans can easily perceive the difference between 500hz and 1000hz, perceiving the difference between 10000hz and 10500hz is much harder, although the difference between the two pairs are the same (Roberts 2020).

[1] An overfit deep learning model is a model that does not generalize from training data to unseen data. It usually happens when the model memorizes the noise from the training data, resulting on predictions based on the noise. An underfit model is when too few features are learnt, making the model inflexible, usually resulting in wrong predictions (EliteDataScience.com 2017)

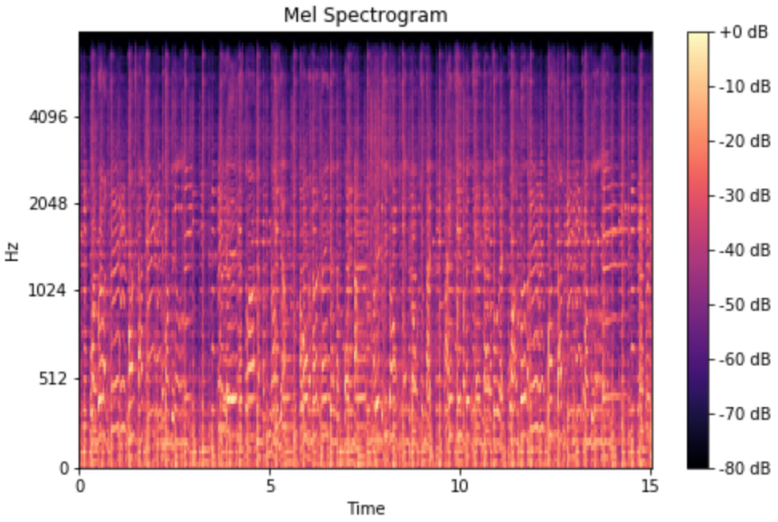


Figure 3.17: Example of Mel Spectrogram (Roberts 2020)

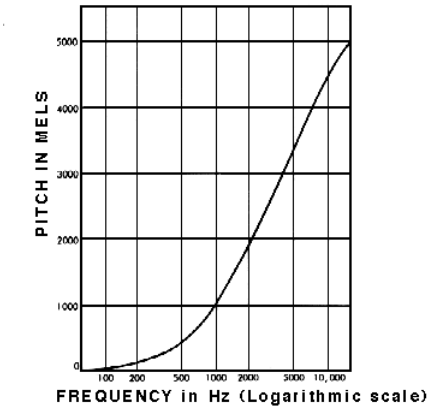


Figure 3.18: Example of Mel Scale (*Mel - Simon Fraser University* 2005)

Mel-Frequency Cepstral Coefficients (MFCC)

The MFCC are widely used in audio classification problems, due to its computational efficiency and noise robustness (Davis and Mermelstein 1980).

The MFCC reduce the amount of information needed to describe a signal for both periodic and aperiodic signals. For the extraction of MFCC, the audio signal is divided into short frames or windows with a given length. Then, a Fourier transform is taken from each frame, which results in the spectrum of the frame. The spectrum is then mapped onto the mel scale and the logs of the mel frequency spectrum are taken, followed by a direct cosine transform. The resulting spectrum amplitudes are MFCC (Davis and Mermelstein 1980).

The figure 3.19 represents the division of the audio signal to MFCC frames. A parameter named window step is how much the window moves before calculating the features for the next frame, meaning the frames can overlap each other. The amount of features to be calculated can be defined by the user, it is usually between 8 and 14 for the optimal amount of information (Loughran et al. 2009).

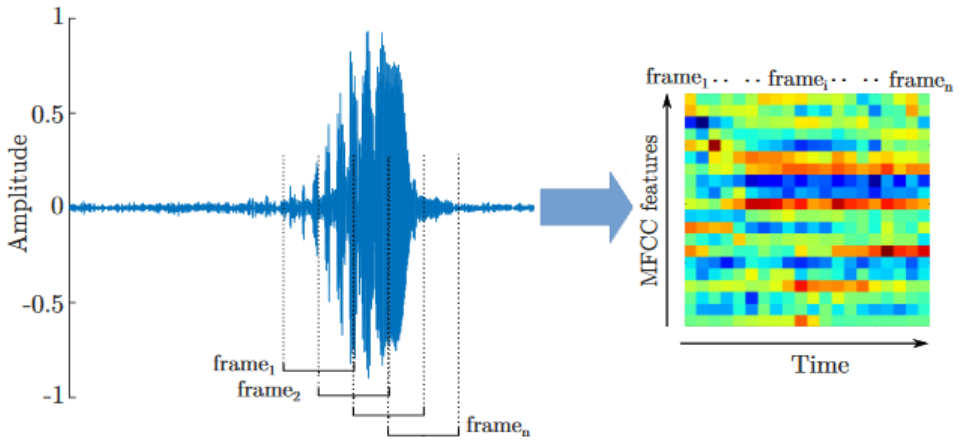


Figure 3.19: Example of MFCC extraction (edited, from: J. Colonna et al. 2016)

Kortas 2020 used MFCC on their sound based bird classification, and stated that the idea behind the mel scale is that it is connected with the way humans hear. When a spectrogram is connected to the mel scale, the result is a modified spectrogram, or a mel-frequency cepstrum. This result ignores the sounds humans do not hear and plots the most important parts (Kortas 2020).

Bark Frequency Cepstral Coefficients (BFCC)

Kaminska, Sapinski, and Pelikant 2013 used Bark Frequency Cepstral Coefficients (BFCC) to extract features for identifying emotional states from speech.

This algorithm is a combination of Perceptual Linear Predictive Coefficients (PLP) processing of the spectrum with the cosine transform to get the cepstral coefficients. Instead of a mel filter bank, bark filters are used to create the BFCC. The main difference between BFCC and MFCC is the power spectrum which is wrapped along its frequency axis onto the bark frequency (Kaminska, Sapinski, and Pelikant 2013).

Revised Perceptual Linear Prediction (RPLP)

Kumar et al. 2010 used Revised Perceptual Linear Prediction (RPLP) to extract features for the purpose of identifying a spoken language.

Instead of PLP coefficients, a mel filter bank is applied to create the RPLP. It is created by first segmenting the signal and processing the Fast Fourier Transform (FFT) spectrum with the mel scale filter bank. This results in a spectrum which is then converted to the cepstral coefficients using Linear Predictive (LP) analysis with a prediction order of 20 followed by cepstral analysis (Kaminska, Sapinski, and Pelikant 2013).

Gammatone Frequency Cepstral Coefficients (GFCC)

Gammatone Frequency Cepstral Coefficients (GFCC) was created by Patterson et al. 1987. Shao et al. 2009 found that GFCC features perform substantially better than conventional MFCC when applied to robust speech recognition.

They are created by first decomposing the input signal into the time-frequency domain using a bank of gammatone filters, followed a down-sampling operation of the filter-bank responses along the time dimension. Then, the magnitudes of the down-sampled filter-bank responses are compressed using cubic root operation, which is then decorrelated using a discrete cosine transform (Ayoub, Jamal, and Arsalane 2016).

3.4 Existing Approaches

3.4.1 BirdCLEF

The goal of the 2019 BirdCLEF challenge was to localize and identify all audible birds within the provided soundscape test set. Each soundscape was divided into segments of 5 seconds (Kahl et al. 2019).

The used evaluation metric was the Classification Mean Average Precision (cmAP). For each class 'c', all predictions classified as that class are extracted and ordered by decreasing probability in order to compute the average precision for that class. The mean for all classes is computed as the main evaluation metric (Kahl et al. 2019).

$$cmAP = \frac{\sum_{c=1}^{C} AveP(c)}{C} \tag{3.5}$$

In the equation 3.5 'C' is the number of classes (species) and 'AveP(c)' is the average precision for a given species (Kahl et al. 2019).

$$AveP(c) = \frac{\sum_{k=1}^{n_c} P(k) * rel(k)}{n_{rel}(c)} \tag{3.6}$$

In the equation 3.6 'k' is the position of an item in the list of the predicted segments for class 'c', 'n_c' is the total number of predicted segments for class 'c', 'P(k)' is the precision at cut-off 'k' in the list, 'rel(k)' is an indicator function equaling 1 if the segment at rank 'k' is a relevant one and 'n_{rel}(c)' is the total number of relevant segments for class 'c' (Kahl et al. 2019).

According to Kortas 2020, almost all of the winning solutions used CNN or CRNN, and even though many of the recordings contained noise, the CNN performed well without any additional noise removal, with many teams even claiming that noise reduction techniques did not help. Data augmentation techniques seemed to be widely used, especially techniques used in the audio processing phase such as time or frequency shift (Kortas 2020).

Best overall approach

Lasseck 2019 achieved top scores in most of the past editions in Bird-CLEF, as well as outperforming all other participating teams in the 2019 challenge (Kahl et al. 2019).

The best overall cmAP on the test set was 35.6%.

- Data Preparation

  All audio recordings were first high pass filtered at a frequency of 2 kHz (Q = 0.707) and then re-sampled to 22050 Hz (Lasseck 2019).

  Lasseck 2019 also created a "noise only" file from each soundscape by merging all parts without bird activity via concatenation. The file was later used together with other background recordings for noise augmentation (Lasseck 2019).

  This was the basic pipeline used according to Lasseck 2019:

  - Extract audio chunk from file with a duration of ca. 5 seconds
  - Apply short-time Fourier transform
  - Normalize and convert power spectrogram to decibel units (dB) via logarithm
  - Convert linear spectrogram to mel spectrogram
  - Remove low and high frequencies
  - Resize spectrogram to fit input dimension of the network
  - Convert grayscale image to RGB image

- Training Setup

  Lasseck 2019 used CNN pretrained on ImageNet (Deng et al. 2009), which were then fine-tuned with mel scaled spectrogram images representing short audio chunks. The models were trained using PyTorch, while using PySoundFile and LibROSA python packages for audio file reading and processing.

- Data Augmentation

  Lasseck 2019 applied the following methods in the time domain regarding the audio chunks:

  - Chunk extraction at random position in file
  - Duration jitter
  - Local time stretching and pitch shifting
  - Filter with random transfer function
  - Random cyclic shift
  - Adding audio chunks from files containing only background noise
  - Adding audio chunks from files belonging to the same bird species(single-label)
  - Adding audio chunks from files belonging to random bird species (multi-label)
  - Random signal amplitude of chunks before summation

– Time interval dropout

3.4.2 Avian Vocalizations (Hiatt 2019)

This solution is available in the dataset source, it was done by Hiatt 2019 as the Capstone Project for a Machine Learning Engineer Nanodegree, who is also the person that compiled the dataset used for this thesis (Hiatt 2020).

An interesting data pre-processing methodology was used where a data generator combines *Mel Spectrograms* with MFCC. This was done by concatenating the MFCC to the top of the *Mel Spectrograms* into a single 2-dimensional array, as it can be seen in the figure 3.20 (Hiatt 2019).

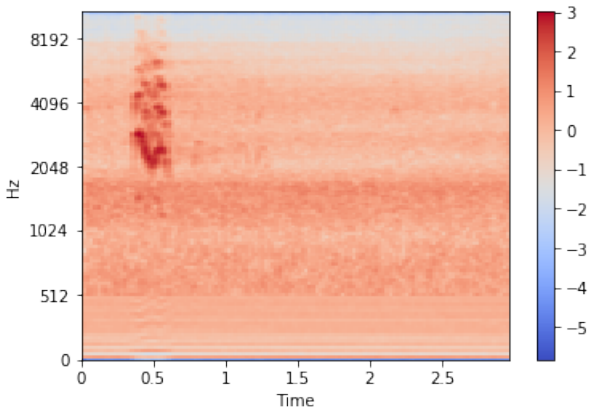


Figure 3.20: Feature Extraction (Hiatt 2019)

The deep learning model implemented is a Convolutional Neural Network with 3 stacks of *2-d Convolutions* using ReLU activation and *MaxPooling* layers followed by *Dropout* layers with a rate of 0.2. The model architecture finishes with a *Global Average Pooling* layer, which is followed by a fully-connected *Dense* layer using the *softmax* activation (Hiatt 2019).

The accuracy metric is used to evaluate the performance of the models. A hold-out is used, with a 77% and 33% ratio for the training and validation sets, respectively. The documented results were obtained using the trained model from the epoch with the minimum validation loss value. With a minimum validation loss of 3.6049, the model achieved a validation accuracy of 19.27%, and a training accuracy of 20.44% (Hiatt 2019).

The figures 3.21 and 3.22 present the percentage of correct classifications per species and a confusion matrix, respectively.

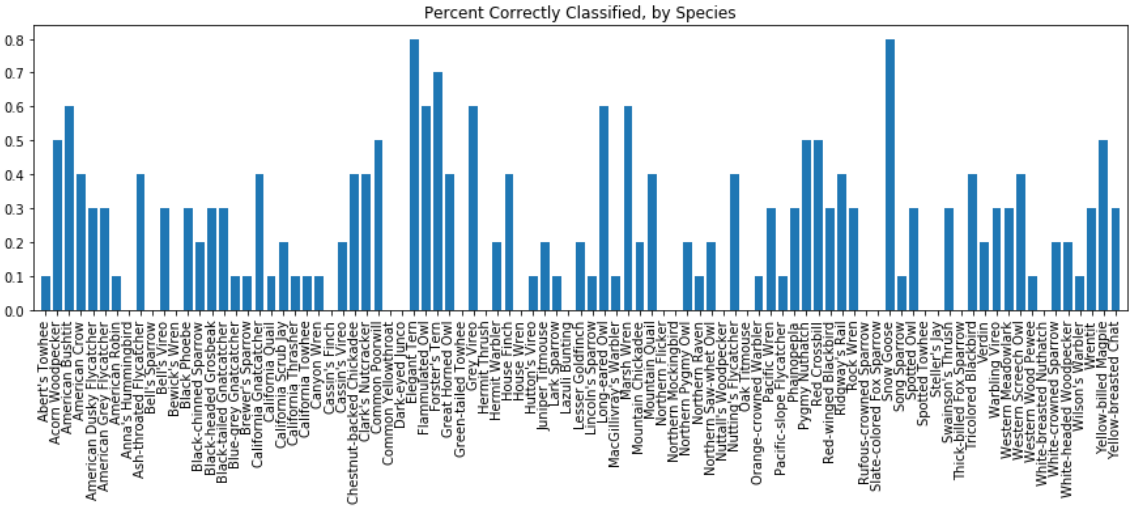


Figure 3.21: Bar Chart with the Percentage of Correct Classifications (Hiatt 2019)

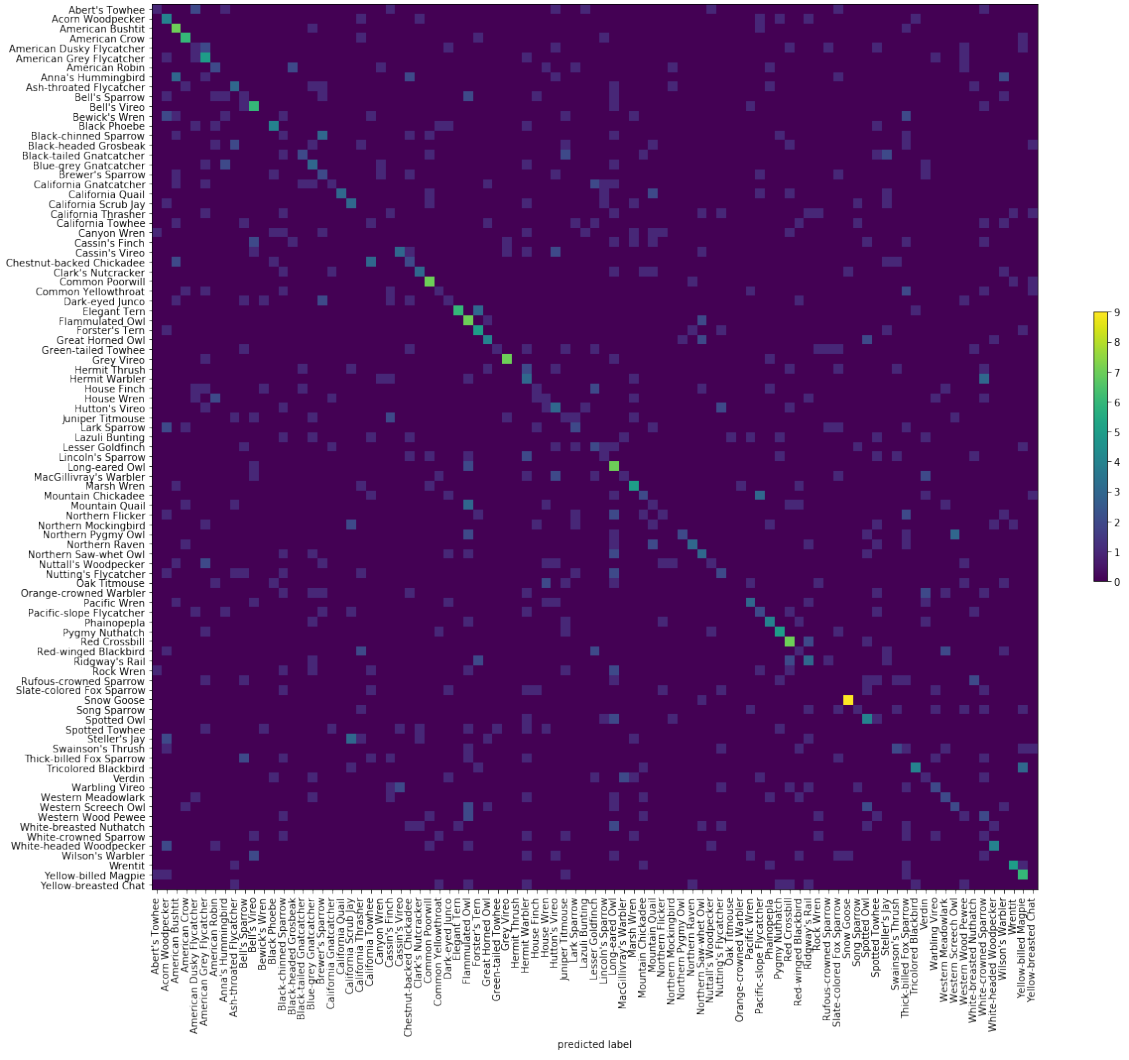


Figure 3.22: Confusion Matrix (Hiatt 2019)

Chapter 4

Adopted Approach

This chapter presents an overview on the adopted approach regarding the Deep Learning Framework, Deep Neural Networks, Evaluation Methodology, Pre-processing and Feature Extraction.

4.1 Deep Learning Framework

One of the major decisions on the development of the solution is the choosing a deep learning framework. The frameworks selected are PyTorch and TensorFlow. They are described in section 3.2.

The chosen framework to develop the solution is TensorFlow using Keras as the high-level API.

Both PyTorch and TensorFlow are a good choice for the deep learning framework. TensorFlow was chosen because of its integration with Keras, and the fact that there is also more content available online about TensorFlow and Keras than PyTorch, making it easier to solve any problem that may occur during development. Tensorflow, in terms of creating models for production, seems to be a better choice as deploying to a variety of devices and other languages is seamless.

4.2 Deep Neural Networks

Another major decision for the development of the solution is what classification architecture to use to classifying the bird sounds.

The studied neural networks on section 3.1 are:

- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
  - Long Short Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
- Convolutional Recurrent Neural Network (CRNN)
  - CNN-LSTM

  - CNN-GRU

During the implementation of the solution all of these neural networks are compared.

4.3 Evaluation Methodology

Evaluation methodologies were presented in section 3.1.4 of the state of the art.

The chosen methodology is repeating evaluation experiments using Hold-out.

The data is randomly split in two folds, the training set (with 80% of the data) and the validation set (with 20% of the data). The training set is used to train the deep learning model, while the validation set is used to evaluate the performance of the model with unseen data.

Repeating the evaluation experiments is useful because deep learning networks are stochastic. As mentioned in 3.1.4, they are stochastic because randomness is used while being fit in the dataset, such as random initial weights and random shuffling of data on each training epoch.

The number of times the Hold-out process is repeated will be defined later in this dissertation, after analysing the time a model takes to train.

- Evaluation Metrics

  As the objective in this dissertation is correctly identifying each bird species, the chosen evaluation metric is Recall.

  The equation 4.1 is an implementation of the recall evaluation metric to the problem:

$$Recall(c) = \frac{CorrectBirds(c)}{CorrectBirds(c) + IncorrectBirds(c)} \tag{4.1}$$

  '*Recall(c)*' is the percentage of birds from class 'c' that were correctly identified. '*CorrectBirds(c)*' is the amount of birds from class 'c' that were correctly identified. '*IncorrectBirds(c)*' is the amount of birds from class 'c' that were incorrectly identified as another species.

  The Weighted Arithmetic Mean of the recall for all the classes of birds (species) can be used as the main score to evaluate a model.

4.4 Pre-processing

Pre-processing methodologies were researched and documented in the section 3.3.1 of the state of the art.

- Standardizing the Signal

  The signal needs to be standardized before being used in any further processes, such as syllable segmentation or feature extraction.

  - Convert stereo audio to mono audio;

  - Normalize the sampling rate to 22kHz;
  - Normalize the bit-depth value range.

- Removing noise by applying a high pass filter
- Syllable segmentation

  The syllable segmentation algorithm proposed by Huang et al. 2009 and used by J. Colonna et al. 2016 is an interesting approach applicable to the bird species audio classification problem.

4.5 Feature Extraction

Multiple feature extractions are mentioned in the section 3.3.2.

The majority of implementations that achieved the best accuracy scores in the *BirdCLEF* challenge (section 3.4.1) used MFCC (Kahl et al. 2019). Fagerlund 2007 also used MFCC to classify bird sounds. Huang et al. 2009 and J. G. Colonna et al. 2012 also used them to classify anuran sounds.

This dissertation will experiment and compare the MFCC and Mel Spectrogram feature extractions. This choice was made due to the information and implementations available when compared to the other feature extraction methods.

Chapter 5

Design of the Solution

This chapter presents the design of the solution. This chapter starts by presenting the optimization and experiment methodologies. This chapter also describes the implementation and structure of the constructed solution.

5.1 Methodology

5.1.1 Optimization Methodology

Before running the experiments that will evaluate the extracted features with each deep learning model, the following will need to be optimized:

- Pre-processing methods;
- Feature extraction;
- Deep learning model.

The following figures 5.1 and 5.2 present two alternatives to how the deep learning models could be trained to visualize their results and improvements while the three items above are being manually optimized.

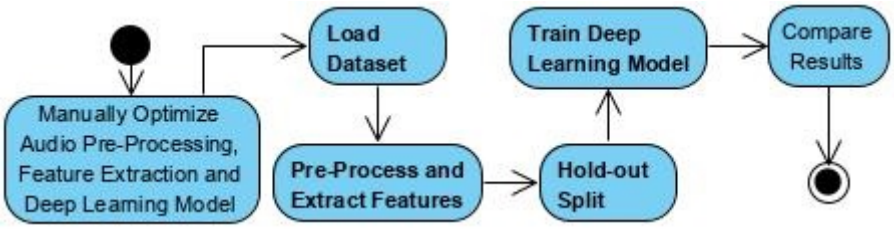
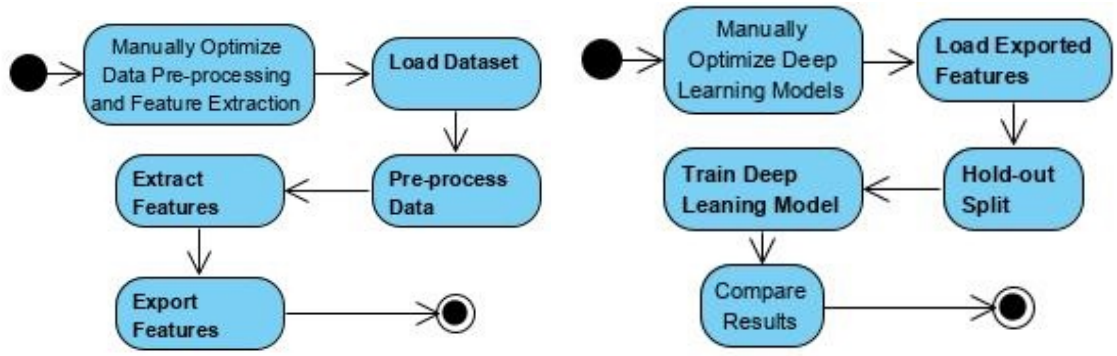


Figure 5.1: Optimization Alternative 1 - Linked Optimization

The alternative presented in the figure 5.1 consists in using the same flow of execution when optimizing the data pre-processing, feature extraction and deep learning architectures.

The alternative presented in the figure 5.2 consists in separating the data pre-processing and feature extraction optimization from the deep learning architecture optimization.

Having the process of data pre-processing and feature extraction optimization separated from the deep learning architecture separated allows generating and exporting multiple datasets. These datasets can use different data pre-processing to extract the features, and exporting



(a) Data pre-processing optimization (b) Deep learning model optimization

Figure 5.2: Optimization Alternative 2 - Separated Optimization

them means that multiple different types of exported features are available for the deep learning optimization process on 5.2b.

In terms of performance when testing different deep learning model architecture changes, the alternative on 5.2b does not need to load the original audio files, pre-process and extract the features like the alternative on 5.1 does.

Therefore, the alternative 2 presented on the figure 5.2 will be used.

5.1.2 Experiment Methodology

This section presents the methodology for the experiments that will be evaluated in the chapter 6.

Figure 5.3 is a visualization of the process used to gather the results of each combination of feature extraction and deep learning models, to then be evaluated and compared.

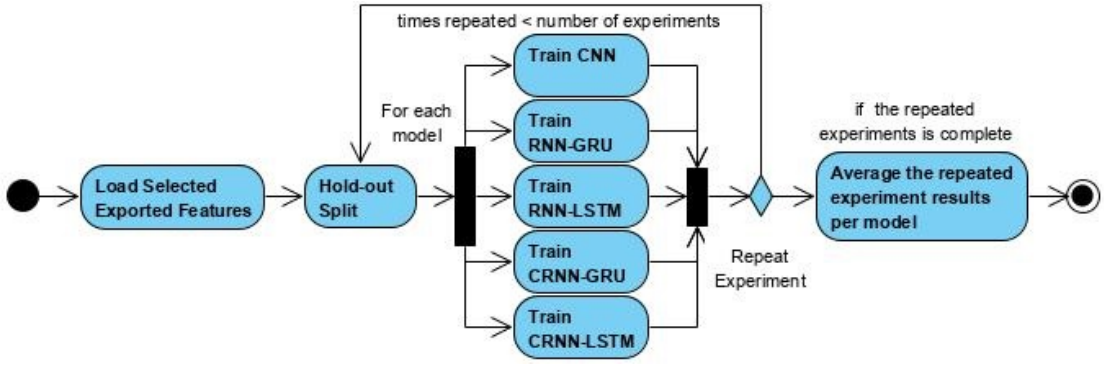


Figure 5.3: Experiment Methodology

The process starts by loading the selected exported extracted features dataset that were previously generated as shown in 5.2a. The available datasets consist on Mel Spectrograms and Mel Frequency Cepstrum Coefficient images generated from the data pre-processed from the original audio dataset.

While loading, the dataset is split using Hold-out, generating a training set which contains 80% of the data, and a validation set containing 20% of the data.

The five deep learning models will then run training and validation iteratively, using the same Hold-out split.

After the five models are done training, the experiment is repeated by making another hold-out split and running the training and validation process again. This is done to get a more accurate result, by averaging the results of each run.

5.2 Implementation

This section presents the implementation of the solution. It starts by explaining the data pre-processing methodology. Then the deep learning model architectures which will be used for the experiments are presented. At the end of this section, the constructed solution is presented.

5.2.1 Data Pre-processing

The data used for this solution was first described on section 2.1 of this document. It consists in 2730 audio files with the *MP3* file format, having different bit-rates and length.

The files are registered in an index file, which identifies the bird species of each file. The index file also provides other information such as the location of where the sample was taken, or the type of bird call, which is not used for the solution.

As stated in the Adopted Approach (section 4), features are extracted from the original audio to be used as the input data for the neural networks. So, the first step in the implementation of the solution was to generate the data that is input to the neural network.

Normalization

The first step taken in this implementation is the normalization of the audio.

Most of the original audio files were not normalized, being that the peak amplitudes were different in many files, resulting in some audio files being quiet relative to some that were loud.

During this normalization phase, the audio files were also converted from the *MP3* file format to the *WAV* file format. The selected sample-rate was 22050Hz.

The file size difference is a downside, because while the dataset using the *MP3* file format takes *1.4GB*, the *WAV* file format version takes *9GB*. But the performance increase when loading (depending on the storage performance), is an upside because the *WAV* files do not need to be decompressed when loaded, as opposed to *MP3* which is a compressed audio format.

The Python library *Pydub* was used to normalize the audio and export to the *WAV* file format.

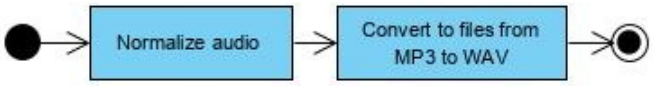


Figure 5.4: Normalization

Preparing for feature extraction

Before the feature extraction, the audio data can be further improved. This process was implemented with multi-threading support in order to speed it up.

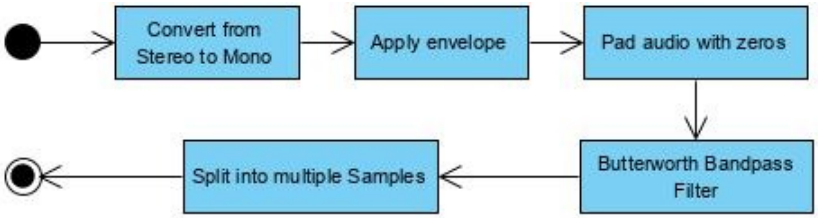


Figure 5.5: Removing noise and splitting into multiple samples

This process starts with converting the audio channels from Stereo to Mono. Then, an envelope filter can be applied to the audio data. This envelope basically removes parts of the audio that are below a certain threshold and can be considered background noise.

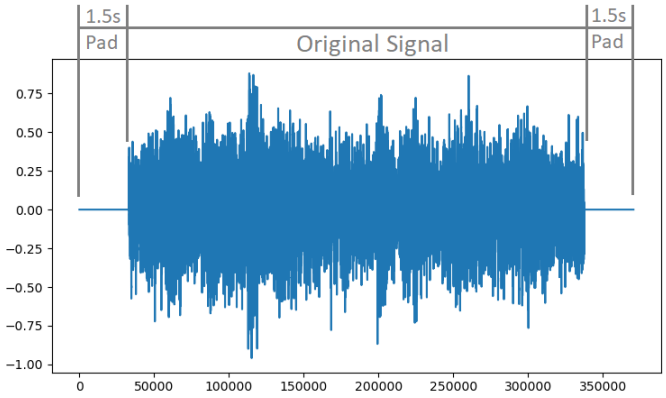


Figure 5.6: Signal padding of Black-tailed Gnatcatcher (ID: XC17806)

The audio is also padded with values close to zero at the beginning and the end of the data in order ease the process of splitting the audio into samples of a certain window length. The audio padding is half of the window length used for the samples at the beginning and the end of the audio.

The figure 5.6 presents the padding on a signal to be used on a split with a window length of 3 seconds. Half of the window length is added to the beginning and the end of the signal (1.5 seconds).

The Butterworth Bandpass filter (Butterworth et al. 1930) is then used to remove frequencies outside of the bird vocalization range. The value used for the lowest frequency cut is 1500hz, and 8000hz for the highest. While testing the pre-processing of the audio data, the Butterworth bandpass filter attenuates most of the noise, such as rain or wind, which

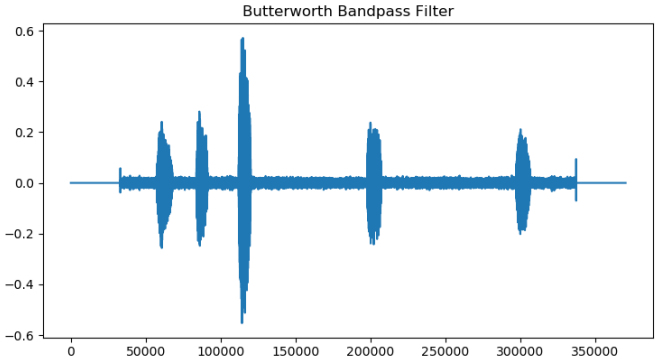


Figure 5.7: Butterworth Bandpass filter on Black-tailed Gnatcatcher (ID: XC17806)

in some cases it makes a big difference both visually and audibly regarding the clarity of the bird chirps.

The figure 5.7 represents the Butterworth Bandpass filter applied to the signal that was shown in the figure 5.6.

The final step is splitting the audio data into multiple samples. The process of splitting the audio data into samples begins by finding the peaks in the sound.

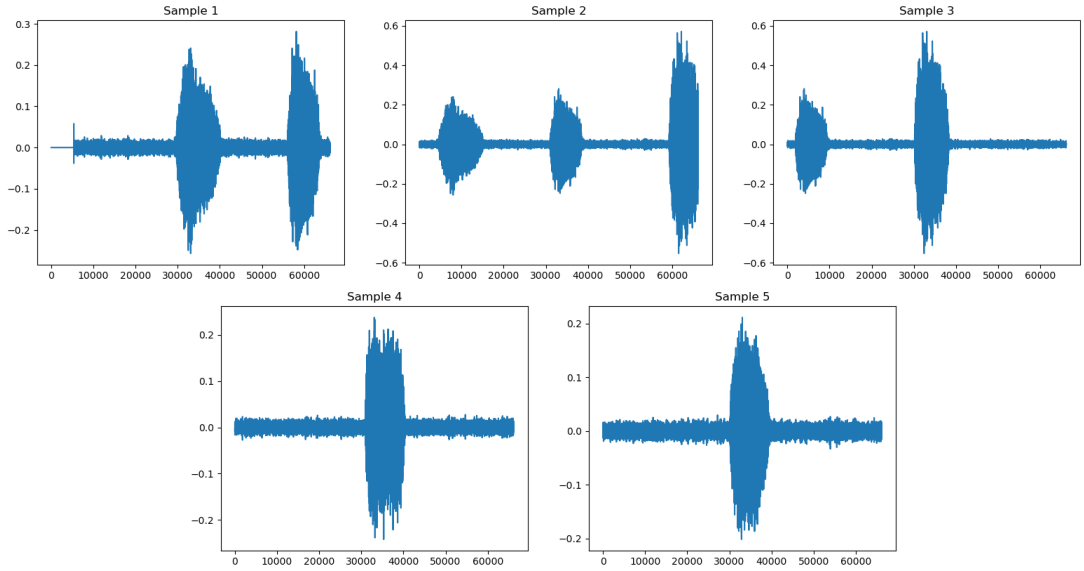


Figure 5.8: Split Samples of Black-tailed Gnatcatcher (ID: XC17806)

These peaks are obtained using a function from the Python library *SciPy*. This function has some parameters that can be defined, such as the peak threshold and the minimum distance between peaks.

After getting a list of peaks, the audio data is then split into multiple samples, centered on the peak, and all with the same length of 3 seconds in the example displayed on the figure 5.8.

As it can be seen on figure 5.8, the samples are split and centered on a certain peak. Even though some samples overlap with other peaks, it can act as data augmentation.

Extracting Features

After the samples were split into equal length, the data is ready for the feature extraction step. The feature extraction process was implemented with multi-threading support in order to speed it up.

The MFCC features were extracted using the Python library *python_speech_features* (Lyons et al. 2020). The listing 5.1 presents the MFCC extraction code.

```
from python_speech_features import mfcc

def get_mfccs(time_series, sample_rate):
    nfft = (round(sample_rate / 40))
    return mfcc(time_series, sample_rate, numcep=13, nfilt=26, nfft=nfft
    , lowfreq=1500, highfreq=8000).T
```

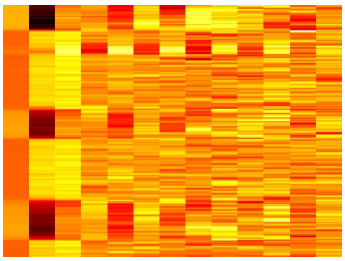
Listing 5.1: Mel Frequency Cepstral Coefficients Extraction Code

The *Mel Spectrograms* were extracted using the Python library *LibROSA* (McFee et al. 2020). The listing 5.2 presents the *Mel Spectrogram* extraction code.

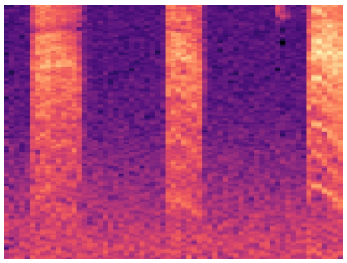
```
from librosa.feature import melspectrogram

def get_melspectrogram(time_series, sample_rate):
    return melspectrogram(y=time_series, sr=sample_rate, n_fft=1024,
    hop_length=1024, n_mels=128, htk=True, fmin=1500, fmax=8000)
```

Listing 5.2: Mel Spectrogram Extraction Code



(a) Extracted Mel Frequency Cepstrum Coefficients of Black-tailed Gnatcatcher (ID: XC17806, Sample 2)



(b) Extracted Mel Spectrogram of Black-tailed Gnatcatcher (ID: XC17806, Sample 2)

Figure 5.9: Mel Frequency Cepstrum Coefficients and Mel Spectrogram visualization

The figures 5.9a and 5.9b represent the extracted features from the Sample 2 of a bird species called *Black-tailed Gnatcatcher*, presented before in the figure 5.8.

The extracted features are then saved into *JPEG* files, sorted in folders by class name. The Python library *Matplotlib* (Hunter 2007) was used to plot and save the features.

5.2.2 Deep Learning Model Architectures

This section presents the architecture of the deep learning models that were selected to be used in this thesis on section 4.2.

These models are implemented using the *TensorFlow* (Abadi et al. 2015) framework with the *Keras* (Chollet et al. 2015) back-end on the Python language.

These models are the result of analysing and testing multiple existing deep learning model architectures available online, such as the implementations by Adams 2020. The models were adapted and optimized to the use case of this dissertation, some of the experiments that were done are documented at section 6.2.1.

Convolutional Neural Network

The chosen Convolutional Neural Network architecture is composed by 6 convolutional layers (*Conv2D*), and always followed by a pooling layer (*MaxPooling2D*).

Table 5.1: Convolutional Neural Network - Model Architecture

	Type	**Kernel**	**Kernel Size**	**Notes**	**Input Shape**
1	*Conv2D*	8	7x7	Activation = *relu*	224 x 224 x 3
2	*MaxPooling2D*		2x2		224 x 224 x 8
3	*BatchNormalization*				112 x 112 x 8
4	*Conv2D*	16	5x5	Activation = *relu*	112 x 112 x 8
5	*MaxPooling2D*		2x2		112 x 112 x 16
6	*BatchNormalization*				56 x 56 x 16
7	*Conv2D*	32	3x3	Activation = *relu*	56 x 56 x 16
8	*MaxPooling2D*		2x2		56 x 56 x 32
9	*BatchNormalization*				28 x 28 x 32
10	*Conv2D*	64	3x3	Activation = *relu*	28 x 28 x 32
11	*MaxPooling2D*		2x2		28 x 28 x 64
12	*BatchNormalization*				14 x 14 x 64
13	*Conv2D*	128	3x3	Activation = *relu*	14 x 14 x 64
14	*MaxPooling2D*		2x2		14 x 14 x 128
15	*BatchNormalization*				7 x 7 x 128
16	*Conv2D*	256	3x3	Activation = *relu*	7 x 7 x 128
17	*Dropout*			Rate = 0.5	7 x 7 x 256
18	*MaxPooling2D*		2x2		7 x 7 x 256
19	*Flatten*				4 x 4 x 256
20	*Dropout*			Rate = 0.5	4096
21	*Dense*	128		Activation = *relu*	4096
22	*Dense*	91		Activation = *softmax*	128

Between each convolutional and pooling layer pair, a batch normalization layer is added. These layers reduce the amount of shift on the values of the hidden layers, increasing the learning speed and reducing over-fitting (D. 2019).

After a *Flatten* layer, a *Dropout* layer with a rate of 50% is added to reduce over-fitting.

The last layer is the fully connected layer, which is a *Dense* layer with the *softmax* activation.

In total, this model has 933211 trainable parameters.

The table 5.1 represents the implemented Convolutional Neural Network model architecture.

Recurrent Neural Network - Long Short Term Memory

The chosen Long Short Term Memory model architecture uses a bidirectional *LSTM* layer.

The first layer reshapes the image data in order to have the correct dimensions for the *LSTM* layer.

The table 5.1 represents the implemented Long Short Term Memory model architecture.

Table 5.2: Long Short Term Memory - Model Architecture

	Type	**Kernel**	**Notes**	**Input Shape**
1	*Reshape*		*TimeDistributed*; *Target* = -1	224 x 224 x 3
2	*Dense*	64	*TimeDistributed*; Activation = *tanh*	224 x 672
3	*LSTM*	128	*BiDirectional*	224 x 64
4	*Dense*	64	Activation = *relu*	224 x 256
5	*MaxPooling1D*			224 x 64
6	*Dense*	32	Activation = *relu*	112 x 64
7	*Flatten*			112 x 32
8	*Dropout*		Rate = 0.5	3584
9	*Dense*	32	Activation = *relu*	3584
10	*Dense*	91	Activation = *softmax*	32

In total, this model has 376955 trainable parameters.

Recurrent Neural Network - Gated Recurrent Unit

The Gated Recurrent Unit model architecture is similar to the presented Long Short Term Memory model architecture but with the *LSTM* layer switched to the *GRU* layer and different units on the *Dense* layers after the *MaxPooling1D* layer.

The table 5.3 represents the implemented Gated Recurrent Unit model architecture.

Table 5.3: Gated Recurrent Unit - Model Architecture

	Type	**Kernel**	**Notes**	**Input Shape**
1	*Reshape*		*TimeDistributed*; *Target* = -1	224 x 224 x 3
2	*Dense*	64	*TimeDistributed*; Activation = *tanh*	224 x 672
3	*GRU*	128	*BiDirectional*	224 x 64
4	*Dense*	128	Activation = *relu*	224 x 256
5	*MaxPooling1D*			224 x 128
6	*Dense*	64	Activation = *relu*	112 x 128
7	*Flatten*			112 x 64
8	*Dropout*		Rate = 0.5	7168
9	*Dense*	32	Activation = *relu*	7168
10	*Dense*	91	Activation = *softmax*	32

In total, this model has 465627 trainable parameters.

Convolutional Recurrent Neural Network - Long Short Term Memory

The Convolutional Recurrent Neural Network using Long Short Term Memory model architecture is based on merging the Convolutional Neural Network model architecture with the Long Short Term Memory model architecture.

Layers have been added, removed and modified in order to get better results.

A dropout layer is added in between the Convolutional layers and the Long Short Term Memory layers in order to reduce over-fitting.

The table 5.4 represents the implemented Convolutional Recurrent Neural Network - Long Short Term Memory model architecture.

Table 5.4: Convolutional Recurrent Neural Network - Long Short Term Memory - Model Architecture

	Type	**Kernel**	**Notes**	**Input Shape**
1	*Conv2D*	32	Activation = *relu*	224 x 224 x 3
2	*MaxPooling2D*			224 x 224 x 32
3	*Conv2D*	64	Activation = *relu*	112 x 112 x 32
4	*MaxPooling2D*			112 x 112 x 64
5	*Conv2D*	128	Activation = *relu*	56 x 56 x 64
6	*MaxPooling2D*			56 x 56 x 128
7	*Conv2D*	256	Activation = *relu*	28 x 28 x 128
8	*MaxPooling2D*			28 x 28 x 256
9	*Dropout*		Rate = 0.5	14 x 14 x 256
10	*Reshape*		*TimeDistributed*; *Target* = -1	14 x 14 x 256
11	*LSTM*	256	*BiDirectional*	14 x 3584
12	*MaxPooling1D*			7 x 512
13	*Dropout*		Rate = 0.5	7 x 512
14	*LSTM*	128	*BiDirectional*	7 x 512
15	*MaxPooling1D*			7 x 256
16	*Flatten*			3 x 256
17	*Dropout*		Rate = 0.5	768
18	*Dense*	32	Activation = *relu*	768
19	*Dense*	91	Activation = *softmax*	64

In total, this model has 8981307 trainable parameters.

Convolutional Recurrent Neural Network - Gated Recurrent Unit

The Convolutional Recurrent Neural Network - Gated Recurrent Unit model architecture is based on the presented Convolutional Recurrent Neural Network - Long Short Term Memory but with the *LSTM* layer switched to the *GRU* layer, with double the amount of kernel units. The second to last *Dense* layer also has double the amount of kernel units.

The table 5.5 represents the implemented Convolutional Recurrent Neural Network - Gated Recurrent Unit model architecture.

Table 5.5: Convolutional Recurrent Neural Network - Gated Recurrent Unit - Model Architecture

	Type	**Kernel**	**Notes**	**Input Shape**
1	*Conv2D*	32	Activation = *relu*	224 x 224 x 3
2	*MaxPooling2D*			224 x 224 x 32
3	*Conv2D*	64	Activation = *relu*	112 x 112 x 32
4	*MaxPooling2D*			112 x 112 x 64
5	*Conv2D*	128	Activation = *relu*	56 x 56 x 64
6	*MaxPooling2D*			56 x 56 x 128
7	*Conv2D*	256	Activation = *relu*	28 x 28 x 128
8	*MaxPooling2D*			28 x 28 x 256
9	*Dropout*		Rate = 0.5	14 x 14 x 256
10	*Reshape*		*TimeDistributed*; *Target* = -1	14 x 14 x 256
11	*GRU*	512	*BiDirectional*	14 x 3584
12	*MaxPooling1D*			14 x 1024
13	*Dropout*		Rate = 0.5	7 x 1024
14	*GRU*	256	*BiDirectional*	7 x 1024
15	*MaxPooling1D*			7 x 512
16	*Flatten*			3 x 512
17	*Dropout*		Rate = 0.5	1536
18	*Dense*	64	Activation = *relu*	1536
19	*Dense*	91	Activation = *softmax*	64

In total, this model has 15065915 trainable parameters.

5.2.3 Constructed Solution

This section presents the constructed solution by providing an overview on the used Python files, classes and libraries.

The figure 5.10 is a visualization of the constructed solution. Some functions, function arguments and variables have been omitted in order to simplify the diagram.

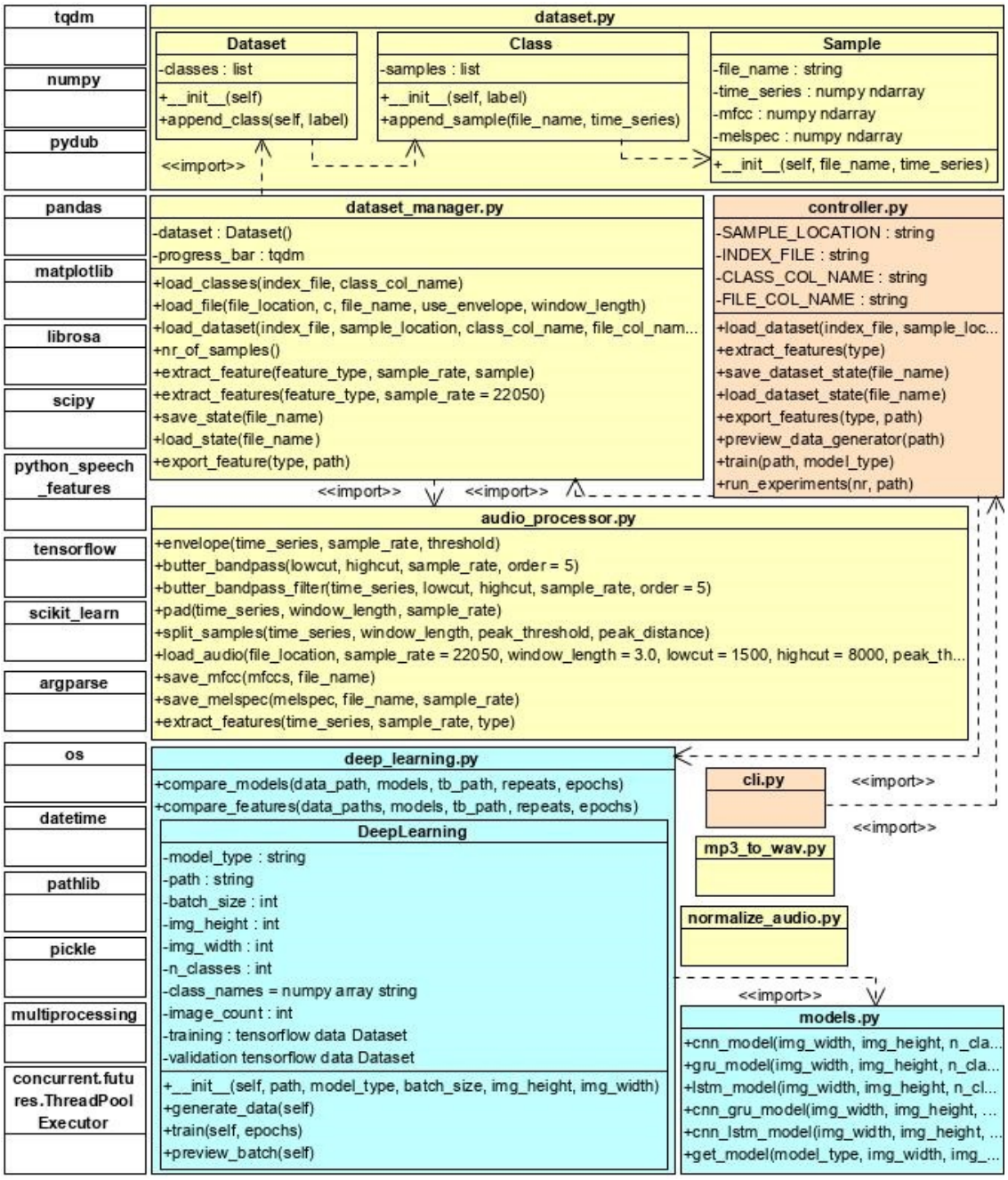


Figure 5.10: Class and Library diagram

The solution is split into 2 independent parts:

- Data Pre-processing;
- Deep Learning.

These 2 parts can be ran independently, but are also linked by the file *controller.py* which acts as an interface for the file *cli.py*, which is a text based menu that interacts will both parts.

In total, the solution can be ran in 3 different ways:

- Through the *cli.py* file, mainly used to explore the dataset
  - Allows the user to load a dataset, which automatically pre-processes the audio files
  - Extract features: *Mel Frequency Cepstrum Coefficients* and *Mel Spectrogram*
  - Export features
  - Save and Load the dataset state
  - Explore the dataset
  - Preview deep learning batch generator
  - Train deep learning model
  - Run the experiment methodologies
- Through the *dataset_manager.py* file, used to quickly execute data pre-processing, feature extraction and exporting.
  - Loads dataset, pre-processes data, extracts and exports features.
- Through the *deep_learning.py* file, used to quickly test deep learning model architectures and run the experiment methodologies.
  - Run the experiment methodologies
  - Test deep learning model

Data Pre-processing

This part of the solution is composed by these Python files:

- *dataset_manager.py*

  This Python file manages all of the dataset operations.

  It uses a *Thread Pool* when using the functions on *audio_processor.py* to load the dataset files, extract and export features in order to speed up the process.

  It also can save and load the state of the *Dataset* class using the Python library *pickle*, with serializes the class and its contents to a file. This proved useful when optimizing the feature extraction, as the dataset files did not need to be loaded and be pre-processed each time a change was done to the feature extraction, speeding up the process.

- *audio_preprocessor.py*

  This Python file handles the loading of the audio files, the pre-processing, such as the envelope, the Butterworth bandpass filter, padding the audio, and splitting into samples.

  It also handles the feature extraction and exports the features.

- *dataset.py*

  The Python file *dataset.py* contains 3 classes:

  - Dataset: contains a list of Class (list of Bird Species)
  - Class: contains a list of Samples, and the name of the Bird Species
  - Sample: contains the original file name and the sample number, the audio time series, the *Mel Frequency Cepstrum Coefficients* and the *Mel Spectrogram* extracted features

- *mp3_to_wav.py* and *normalize_audio.py*

  These two Python files were used early on the development of this solution.

  The original audio files are encoded with the *MP3* format and were converted to the uncompressed *WAV* format using the *mp3_to_wav.py* file.

  The audio amplitudes were normalized using the *normalize_audio.py* file.

  Both of these files use the library *PyDub* to handle the audio.

Deep Learning

This part of the solution contained in the *deep_learning.py* and *models.py* Python files.

The *deep_learning.py* file contains a class named *DeepLearning*, which provides the following methods:

- *generate_data()*: Generates the training and validation sets (80% and 20% split, respectively as a default, but can be changed), using the *ImageDataGenerator* from the back-end *Keras* of the *TensorFlow* framework.
- *train()*: Fits the model with the training set and validates with the validation set. The method gets the model from the *models.py* Python file.

The *deep_learning.py* file also provides the functions *compare_models()* and *compare_features()* which automatically execute the experiment methodologies as presented on section 5.1.2.

The *models.py* file provides all of the deep learning models used in the experiments. The alternatives used for the experiments in the section 6.2.1 are also in this file, but are omitted in the figure 5.10.

Chapter 6

Evaluation

This chapter presents the evaluation of the solution. The section 6.1 overviews the multiple options in the solution the be evaluated, the metrics that will be used to evaluate the options and the hypothesis. The section 6.2 contains the multiple experiments of this dissertation. The section 6.3 describes the statistical test that evaluates the hypothesis.

6.1 Methodology

This section presents the hypothesis and combinations that will be evaluated, as well as the metrics used.

6.1.1 Combinations

The main objective in this dissertation is to propose a methodology able to identify a bird species by its chirp using deep learning techniques. Multiple combinations of feature extraction methods and deep learning models will be tested.

In total, there are 10 combinations to be tested, as stated in chapter 4 5 deep learning models and 2 feature extraction methods were selected to be tested. These combinations are:

- CNN with: MFCC and Mel Spectrogram
- RNN LSTM with: MFCC and Mel Spectrogram
- RNN GRU with: MFCC and Mel Spectrogram
- CRNN LSTM with: MFCC and Mel Spectrogram
- CRNN GRU with: MFCC and Mel Spectrogram

6.1.2 Metrics

As stated in chapter 4, hold-out split and repeating experiments will be used to get reliable results out of the experiments.

The main metric used to compare the deep learning models is the average recall per bird species. Other metrics are also documented on the experiments, such as accuracy, loss and time.

6.1.3 Hypothesis

The null hypothesis claims that all the combinations of deep learning architectures with feature extraction methods perform equally. The rejection of the null hypothesis means that there exists at least a pair of classifiers with significantly different performances.

The Friedman test followed by the Nemenyi test are used as the statistical tests to evaluate the hypothesis.

The Friedman test is a non-parametric statistical test for comparing more than two samples. When the Friedman test leads to significant results, at least one of the samples is different from the other samples (Corder and Foreman 2009). Then, the Nemenyi test can be used to post-hoc pairwise test for multiple comparisons.

6.2 Experiments

The upcoming sections document how the deep learning model architectures were chosen and optimized, how different feature extractions perform when combined with different deep learning models, a result comparison and the statistical test to answer the hypothesis.

The experiments were conducted in the following environment:

- OS: Ubuntu 20.04 LTS (Focal Fossa)
- Python: version 3.6
- TensorFlow: version 2.2
- Processor: AMD Ryzen 9 3900X
- RAM: 32GB
- Graphics Card: AMD RX 580X 8GB (*RadeonOpenCompute* 3.5.0)

6.2.1 Deep Learning Model Architectures

During the process of constructing the solution, several deep learning model architectures were tested.

The following sections present 3 architecture variations for each model. The variations for each model type were selected out of a large group of networks that were built and tested. Only 3 variations are presented in order to simplify the dissertation.

This experience elapsed a maximum of 50 epochs for each alternative. This was decided because while pre-testing the alternatives it was found that, on the validation set, the models stop improving, or barely improve after 50 epochs and their loss values increase, indicating that the model is over-fitting. In order to save time between each experiment, 50 epochs was chosen.

The experiment was repeated 3 times and the values used to evaluate the models were averaged. On each experiment, the same training and validation data was used per model architecture alternative. The input data used were MFCC extracted features with sample lengths of 3 seconds.

The metrics registered in this experiment were:

- At the maximum recall value:
  - Maximum recall value (average);
  - Epoch of maximum recall value (average);
  - Time to maximum recall value (average);
  - Loss at maximum recall value (average);
  - Accuracy at maximum recall value (average).

The models which performed the best in these experiments are the ones detailed in section 5.2.2.

Convolutional Neural Network

The alternatives presented on table 6.1 only differ on the second to last Dense layer.

These alternatives were chosen because they performed the best compared to other layer configurations.

Table 6.1: Convolutional Neural Network alternatives

(a) Alternative 1

1	Conv2D (8)
2	MaxPooling2D
3	BatchNormalization
4	Conv2D (16)
5	MaxPooling2D
6	BatchNormalization
7	Conv2D (32)
8	MaxPooling2D
9	BatchNormalization
10	Conv2D (64)
11	MaxPooling2D
12	BatchNormalization
13	Conv2D (128)
14	MaxPooling2D
15	BatchNormalization
16	Conv2D (256)
17	Dropout (0.2)
18	MaxPooling2D
19	Flatten
20	Dropout (0.2)
21	Dense (64)
22	Dense (91)

(b) Alternative 2

1	Conv2D (8)
2	MaxPooling2D
3	BatchNormalization
4	Conv2D (16)
5	MaxPooling2D
6	BatchNormalization
7	Conv2D (32)
8	MaxPooling2D
9	BatchNormalization
10	Conv2D (64)
11	MaxPooling2D
12	BatchNormalization
13	Conv2D (128)
14	MaxPooling2D
15	BatchNormalization
16	Conv2D (256)
17	Dropout (0.2)
18	MaxPooling2D
19	Flatten
20	Dropout (0.2)
21	Dense (256)
22	Dense (91)

(c) Alternative 3

1	Conv2D (8)
2	MaxPooling2D
3	BatchNormalization
4	Conv2D (16)
5	MaxPooling2D
6	BatchNormalization
7	Conv2D (32)
8	MaxPooling2D
9	BatchNormalization
10	Conv2D (64)
11	MaxPooling2D
12	BatchNormalization
13	Conv2D (128)
14	MaxPooling2D
15	BatchNormalization
16	Conv2D (256)
17	Dropout (0.2)
18	MaxPooling2D
19	Flatten
20	Dropout (0.2)
21	Dense (128)
22	Dense (91)

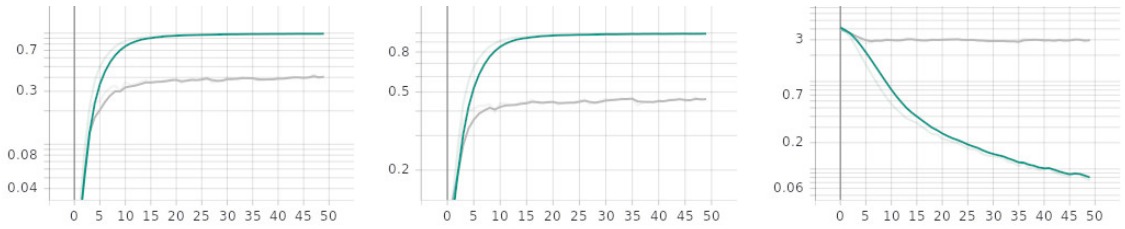
The table 6.2 compares the 3 alternatives presented in the table 6.1. The values are averaged from the result of 3 repeated experiments.

Alt.	At Maximum Recall (Validation)							
	Validation Set			Epoch	Time	Training Set		
	Recall	Accuracy	Loss			Recall	Accuracy	Loss
1	40.55%	44.99%	3.204	45	52m06s	97.91%	98.52%	0.1104
2	39.85%	45.49%	2.898	43	51m09s	99.08%	99.35%	0.0633
3	**41.70%**	46.52%	2.956	42	45m31s	98.68%	99.10%	0.0839

Table 6.2: Convolutional Neural Network alternative comparison

The alternative 3 presented on the table 6.1c is considered the best in this experiment due to its recall on validation when compared to the others.

Recall, accuracy and loss graphs from the best run of the alternative 3 are presented in the figures 6.1a, 6.1b and 6.1c respectively. The training line is displayed in a dark teal color and the validation line is displayed in a gray color.



(a) Recall per Epoch (b) Accuracy per Epoch (c) Loss per Epoch

Figure 6.1: Recall, Accuracy and Loss per Epoch - Convolutional Neural Network

Recurrent Neural Network - Long Short Term Memory

The alternatives are presented on table 6.3.

The alternative 1 (table 6.3a) differs from the alternative 2 (table 6.3b) on the *Dense* layer in between the *Bidirectional LSTM* layer and *MaxPooling1D* layer.

The alternative 3 (table 6.3c) only differs in the *LSTM* layer, where in the alternatives 1 and 2 *Bidirectional LSTM* is used.

These alternatives were chosen because they performed the best compared to other layer configurations.

Table 6.3: Recurrent Neural Network - Long Short Term Memory alternatives

(a) Alternative 1

1	Reshape
2	TimeD. Dense (64)
3	Bidir. LSTM (128)
4	Dense (128)
5	MaxPooling1D
6	Dense (64)
7	Flatten
8	Dropout (0.5)
9	Dense (32)
10	Dense (91)

(b) Alternative 2

1	Reshape
2	TimeD. Dense (64)
3	Bidir. LSTM (128)
4	Dense (128)
5	MaxPooling1D
6	Dense (32)
7	Flatten
8	Dropout (0.5)
9	Dense (32)
10	Dense (91)

(c) Alternative 3

1	Reshape
2	TimeD. Dense (64)
3	LSTM (128)
4	Dense (128)
5	MaxPooling1D
6	Dense (64)
7	Flatten
8	Dropout (0.5)
9	Dense (32)
10	Dense (91)

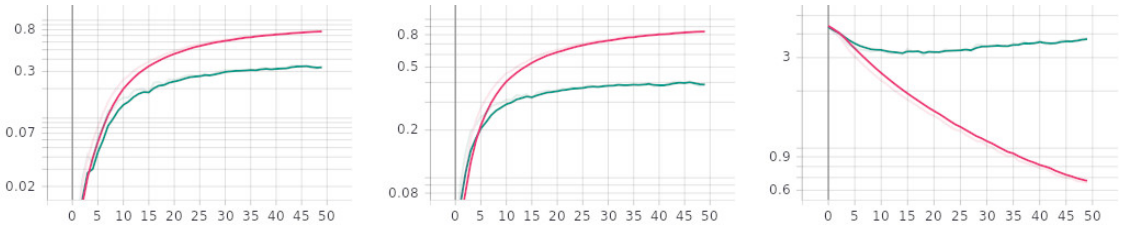
The table 6.4 compares the 3 alternatives presented in the table 6.3. The values are averaged from the result of 3 repeated experiments.

Alt.	At Maximum Recall (Validation Set)							
	Validation Set			Epoch	Time	Training Set		
	Recall	Accuracy	Loss			Recall	Accuracy	Loss
1	32.54%	37.25%	4.026	44	86m02s	84.65%	89.53%	0.4833
2	**33.62**%	39.20%	3.768	45	91m47s	80.05%	86.24%	0.5925
3	30.85%	37.05%	3.789	48	63m58s	71.64%	79.88%	0.7964

Table 6.4: Long Short Term Memory alternative comparison

The alternative 2 presented on the table 6.3b is considered the best in this experiment due to its recall on validation when compared to the others.

Recall, accuracy and loss graphs from the best run of the alternative 2 are presented in the figures 6.2a, 6.2b and 6.2c respectively. The training line is displayed in a pink color and the validation line is displayed in a dark teal color.



(a) Recall per Epoch (b) Accuracy per Epoch (c) Loss per Epoch

Figure 6.2: Recall, Accuracy and Loss per Epoch - Long Short Term Memory

Recurrent Neural Network - Gated Recurrent Unit

The alternatives are presented on table 6.5. The Gated Recurrent Unit model architecture implementations are very similar to the Long Short Term Memory model architecture implementations presented on the table 6.3, having the *LSTM* layers switched to the *GRU* layers.

Table 6.5: Recurrent Neural Network - Gated Recurrent Unit alternatives

(a) Alternative 1

1	Reshape
2	TimeD. Dense (64)
3	Bidir. GRU (128)
4	Dense (128)
5	MaxPooling1D
6	Dense (64)
7	Flatten
8	Dropout (0.5)
9	Dense (32)
10	Dense (91)

(b) Alternative 2

1	Reshape
2	TimeD. Dense (64)
3	Bidir. GRU (128)
4	Dense (128)
5	MaxPooling1D
6	Dense (32)
7	Flatten
8	Dropout (0.5)
9	Dense (32)
10	Dense (91)

(c) Alternative 3

1	Reshape
2	TimeD. Dense (64)
3	GRU (128)
4	Dense (128)
5	MaxPooling1D
6	Dense (64)
7	Flatten
8	Dropout (0.5)
9	Dense (32)
10	Dense (91)

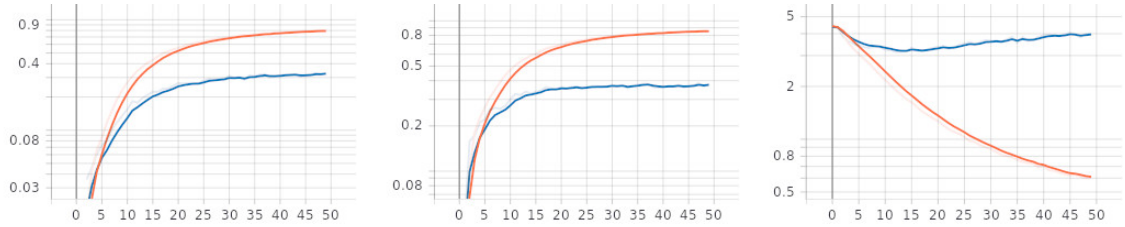
The table 6.6 compares the 3 alternatives presented in the table 6.5. The values are averaged from the result of 3 repeated experiments.

Alt.	At Maximum Recall (Validation Set)							
	Validation Set			Epoch	Time	Training Set		
	Recall	Accuracy	Loss			Recall	Accuracy	Loss
1	**33.59**%	38.24%	3.964	44	105m14s	82.51%	87.62%	0.5431
2	33.39%	38.25%	3.931	47	109m58s	61.30%	85.09%	0.6130
3	31.27%	36.91%	3.861	47	71m29s	72.85%	80.28%	0.7777

Table 6.6: Gated Recurrent Unit alternative comparison

The alternative 1 presented on the table 6.5a is considered the best in this experiment due to its recall on validation when compared to the others.

Recall, accuracy and loss graphs from the best run of the alternative 1 are presented in the figures 6.3a, 6.3b and 6.3c respectively. The training line is displayed in an orange color and the validation line is displayed in a blue color.



(a) Recall per Epoch (b) Accuracy per Epoch (c) Loss per Epoch

Figure 6.3: Recall, Accuracy and Loss per Epoch - Gated Recurrent Unit

Convolutional Recurrent Neural Network - Long Short Term Memory

The alternatives are presented on table 6.7.

The alternative 1 presented on the table 6.7a is similar to the alternative 3 presented on the table 6.7c, differing only on the *LSTM* layers. These 2 alternatives were chosen because they performed the best compared to other layer configurations.

The alternative 2 presented on 6.7b was the first one created. It is based on joining a Convolutional Neural Network alternative with a Long Short Term Memory alternative.

Table 6.7: Convolutional Recurrent Neural Network - Long Short Term Memory alternatives

(a) Alternative 1

1	Conv2D (32)
2	MaxPooling2D
3	Conv2D (64)
4	MaxPooling2D
5	Conv2D (128)
6	MaxPooling2D
7	Conv2D (256)
8	MaxPooling2D
9	Dropout (0.5)
10	TimeDist. Reshape
11	Bidir. LSTM (256)
12	MaxPooling1D
13	Dropout (0.5)
14	Bidir. LSTM (128)
15	MaxPooling1D
16	Flatten
17	Dropout (0.5)
18	Dense (64)
19	Dense (91)

(b) Alternative 2

1	Conv2D (8)
2	MaxPooling2D
3	BatchNormalization
4	Conv2D (16)
5	MaxPooling2D
6	BatchNormalization
7	Conv2D (32)
8	MaxPooling2D
9	BatchNormalization
10	Conv2D (64)
11	MaxPooling2D
12	BatchNormalization
13	Conv2D (128)
14	MaxPooling2D
15	Reshape
16	TimeD. Dense (64)
17	Bidir. LSTM (128)
18	Dense (64)
19	MaxPooling1D
20	Dense (32)
21	Flatten
22	Dropout (0.5)
23	Dense (32)
24	Dense (91)

(c) Alternative 3

1	Conv2D (32)
2	MaxPooling2D
3	Conv2D (64)
4	MaxPooling2D
5	Conv2D (128)
6	MaxPooling2D
7	Conv2D (256)
8	MaxPooling2D
9	Dropout (0.5)
10	TimeDist. Reshape
11	Bidir. LSTM (512)
12	MaxPooling1D
13	Dropout (0.5)
14	Bidir. LSTM (256)
15	MaxPooling1D
16	Flatten
17	Dropout (0.5)
18	Dense (64)
19	Dense (91)

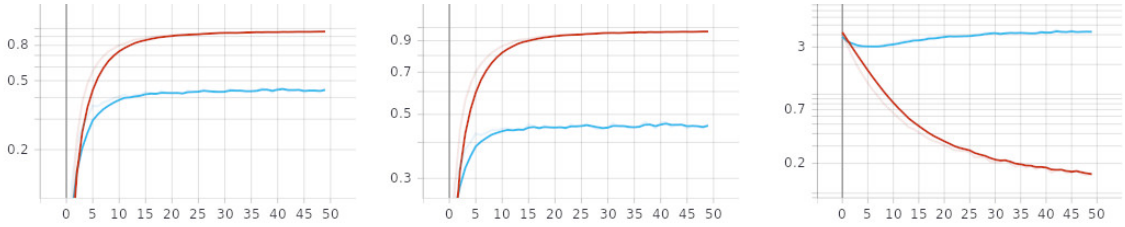
The table 6.8 compares the 3 alternatives presented in the table 6.3. The values are averaged from the result of 3 repeated experiments.

Alt.	At Maximum Recall (Validation Set)							
	Validation Set			Epoch	Time	Training Set		
	Recall	Accuracy	Loss			Recall	Accuracy	Loss
1	**45.12**%	46.65%	4.193	41	74m26s	94.83%	96.64%	0.1914
2	32.25%	39.22%	3.042	46	54m28s	53.48%	65.79%	1.346
3	44.69%	46.52%	4.069	38	75m35s	95.60%	96.56%	1.807

Table 6.8: Convolutional Recurrent Neural Network - Long Short Term Memory alternative comparison

The alternative 1 presented on the table 6.7a is considered the best in this experiment due to its recall on validation when compared to the others.

Recall, accuracy and loss graphs from the best run of the alternative 1 are presented in the figures 6.4a, 6.4b and 6.4c respectively. The training line is displayed in a red color and the validation line is displayed in a blue color.



(a) Recall per Epoch (b) Accuracy per Epoch (c) Loss per Epoch

Figure 6.4: Recall, Accuracy and Loss per Epoch - Convolutional Recurrent Neural Network - Long Short Term Memory

Convolutional Recurrent Neural Network - Gated Recurrent Unit

The alternatives are presented on table 6.9.

The Convolutional Recurrent Neural Network - Gated Recurrent Unit model architecture implementations are very similar to the Convolutional Recurrent Neural Network - Long Short Term Memory model architecture implementations presented on the table 6.7, having the *LSTM* layers switched to the *GRU* layers.

Table 6.9: Convolutional Recurrent Neural Network - Gated Recurrent Unit alternatives

(a) Alternative 1

1	Conv2D (32)
2	MaxPooling2D
3	Conv2D (64)
4	MaxPooling2D
5	Conv2D (128)
6	MaxPooling2D
7	Conv2D (256)
8	MaxPooling2D
9	Dropout (0.5)
10	TimeDist. Reshape
11	Bidir. GRU (256)
12	MaxPooling1D
13	Dropout (0.5)
14	Bidir. GRU (128)
15	MaxPooling1D
16	Flatten
17	Dropout (0.5)
18	Dense (64)
19	Dense (91)

(b) Alternative 2

1	Conv2D (8)
2	MaxPooling2D
3	BatchNormalization
4	Conv2D (16)
5	MaxPooling2D
6	BatchNormalization
7	Conv2D (32)
8	MaxPooling2D
9	BatchNormalization
10	Conv2D (64)
11	MaxPooling2D
12	BatchNormalization
13	Conv2D (128)
14	MaxPooling2D
15	Reshape
16	TimeD. Dense (64)
17	Bidir. GRU (128)
18	Dense (64)
19	MaxPooling1D
20	Dense (32)
21	Flatten
22	Dropout (0.5)
23	Dense (32)
24	Dense (91)

(c) Alternative 3

1	Conv2D (32)
2	MaxPooling2D
3	Conv2D (64)
4	MaxPooling2D
5	Conv2D (128)
6	MaxPooling2D
7	Conv2D (256)
8	MaxPooling2D
9	Dropout (0.5)
10	TimeDist. Reshape
11	Bidir. GRU (512)
12	MaxPooling1D
13	Dropout (0.5)
14	Bidir. GRU (256)
15	MaxPooling1D
16	Flatten
17	Dropout (0.5)
18	Dense (64)
19	Dense (91)

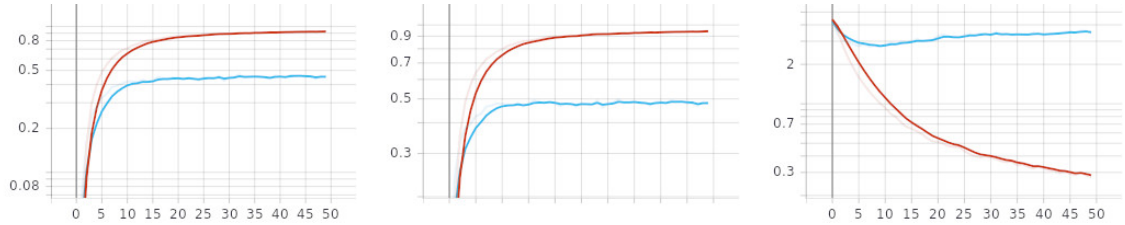
The table 6.10 compares the 3 alternatives presented in the table 6.9. The values are averaged from the result of 3 repeated experiments.

Alt.	At Maximum Recall (Validation Set)							
	Validation Set			Epoch	Time	Training Set		
	Recall	Accuracy	Loss			Recall	Accuracy	Loss
1	44.47%	47.05%	3.570	47	86m37s	91.73%	93.56%	0.3046
2	28.61%	37.61%	3.085	46	54m40s	40.77%	55.47%	1.751
3	**46.27**%	48.94%	3.414	41	82m53s	92.14%	94.12%	0.3007

Table 6.10: Convolutional Recurrent Neural Network - Gated Recurrent Unit alternative comparison

The alternative 3 presented on the table 6.9c is considered the best in this experiment due to its recall on validation when compared to the others.

Recall, accuracy and loss graphs from the best run of the alternative 3 are presented in the figures 6.5a, 6.5b and 6.5c respectively. The training line is displayed in a red color and the validation line is displayed in a blue color.



(a) Recall per Epoch (b) Accuracy per Epoch (c) Loss per Epoch

Figure 6.5: Recall, Accuracy and Loss per Epoch - Convolutional Recurrent Neural Network - Gated Recurrent Unit

6.2.2 Combinations

This experiment intends to evaluate the performance of each type of extracted feature with each of the best performing model architectures from the experiment documented on section 6.2.1.

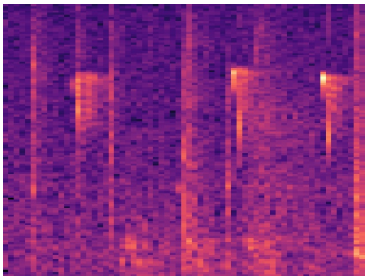
The extracted features that will be tested are these:

- MFCC with 3 second sample lengths;
- MFCC with 1.5 second sample lengths;
- Mel Spectrograms with 3 second sample lengths;
- Mel Spectrograms with 1.5 second sample lengths.

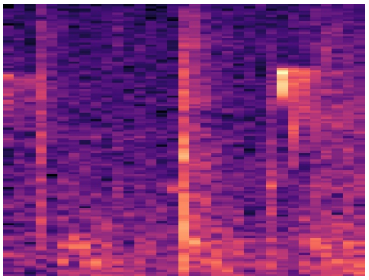
Examples of these extracted features can be seen in the figures 6.6 and 6.7.

(a) Extracted MFCC (3 seconds) (b) Extracted MFCC (1.5 seconds)

Figure 6.6: Extracted Mel Frequency Cepstrum Coefficients (American Crow, ID: XC110263, Sample 4)



(a) Extracted Mel Spectrogram (3 seconds)



(b) Extracted Mel Spectrogram (1.5 seconds)

Figure 6.7: Extracted Mel Spectrogram (American Crow, ID: XC110263, Sample 4)

The deep learning model architectures are described on section 5.2.2.

In total the experiment will test 5 deep learning model architectures with 4 different extracted features, which means 20 different combinations to be compared.

In order to get a consistent result, the experiments will be repeated 5 times and averaged.

The results of this experiment are presented in the table 6.11.

Table 6.11: Extracted Features and Deep Learning Model Architecture Combination Comparison

Comb.		At Maximum Recall (Validation Set)							
		Validation Set			Epoch	Time	Training Set		
F.	Models	Recall	Acc.	Loss			Recall	Acc.	Loss
MFCCs 3s	CNN	40.50%	45.41%	3.040	47	57m	98.82%	99.13%	0.0787
	LSTM	32.44%	38.01%	3.778	47	91m	77.86%	84.65%	0.6491
	GRU	35.50%	39.53%	3.979	46	107m	85.69%	89.77%	0.4622
	CRNN LSTM	**44.76**%	46.10%	4.233	42	74m	96.17%	97.01%	0.1556
	CRNN GRU	44.40%	47.81%	3.470	41	88m	90.80%	92.79%	0.3498
MFCCs 1.5s	CNN	38.82%	44.44%	3.134	45	52m	98.19%	98.76%	0.1116
	LSTM	27.27%	33.65%	3.909	45	89m	71.99%	80.79%	0.7931
	GRU	31.36%	36.23%	4.097	46	111m	79.56%	85.26%	0.6354
	CRNN LSTM	42.36%	43.95%	4.404	43	76m	91.97%	93.81%	0.2796
	CRNN GRU	**43.35**%	46.47%	3.539	40	82m	90.82%	92.64%	0.3476
Mel Spectrogram 3s	CNN	47.51%	43.05%	2.574	40	68m	99.05%	99.37%	0.0761
	LSTM	26.27%	33.32%	3.832	44	95m	65.85%	75.80%	0.9608
	GRU	28.23%	33.48%	4.271	43	110m	75.96%	83.36%	0.7034
	CRNN LSTM	47.95%	50.30%	3.475	48	93m	94.40%	95.60%	0.2140
	CRNN GRU	**50.17**%	53.13%	3.003	45	98m	90.64%	92.70%	0.3659
Mel Spectrogram 1.5s	CNN	44.21%	49.28%	2.957	41	68m	98.76%	99.14%	0.0813
	LSTM	24.62%	32.51%	3.790	47	101m	62.68%	73.29%	1.051
	GRU	27.56%	32.51%	4.071	46	115m	79.56%	85.46%	0.6354
	CRNN LSTM	46.46%	48.48%	3.763	40	78m	91.30%	93.39%	0.3109
	CRNN GRU	**47.92**%	50.71%	3.286	46	102m	89.74%	91.95%	0.3935

Overall, both Mel Frequency Cepstrum Coefficients and Mel Spectrogram performed better with a sample length of 3 seconds when compared to the 1.5 second sample length versions, as it can be seen in the table 6.11

The best combination on this experiment is using the Mel Spectrogram with 3 second sample length as the extracted features, and the Convolutional Recurrent Neural Network - Gated Recurrent Unit as the deep learning model. This combination achieved an average recall of 50.17% on epoch 45. The figures 6.8, 6.9 and 6.10 present the recall, accuracy

and loss of the Convolutional Recurrent Neural Network - Gated Recurrent Unit using Mel Spectrograms. In the figures, the training line is orange and the validation line is blue.

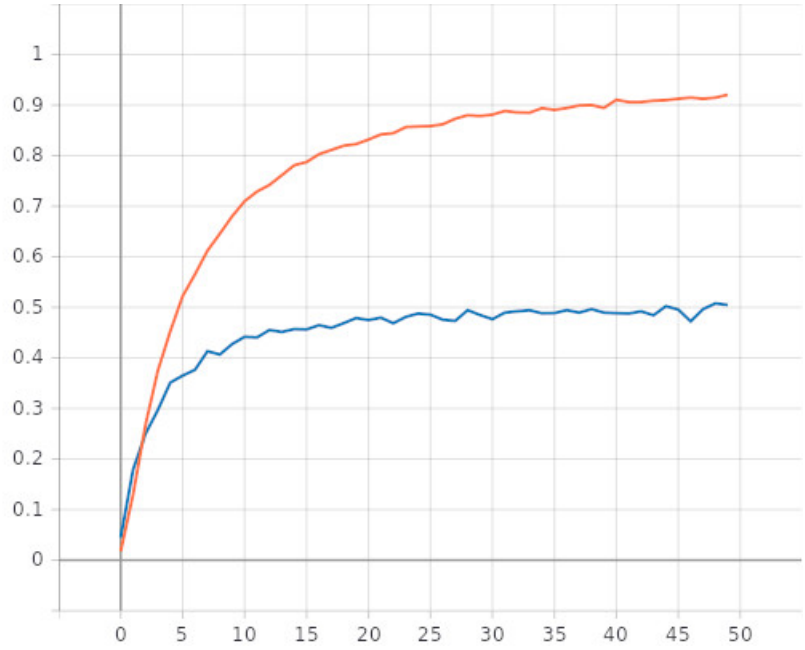


Figure 6.8: Recall per epoch of CRNN-GRU using Mel Spectrogram

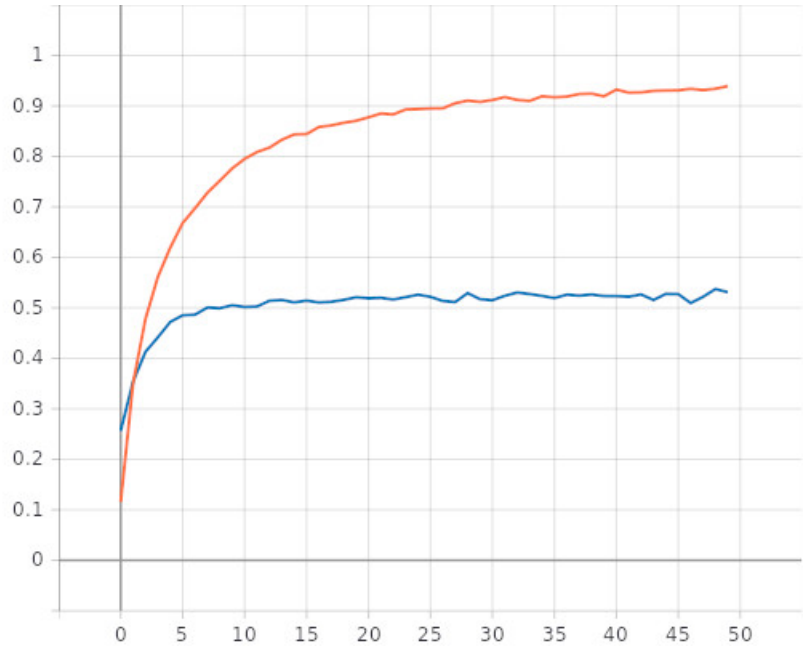


Figure 6.9: Accuracy per epoch of CRNN-GRU using Mel Spectrogram

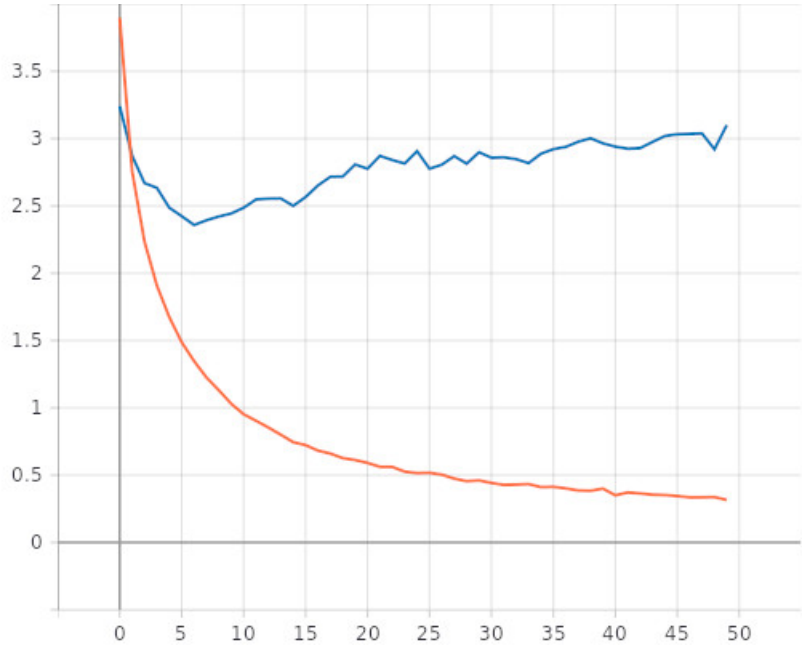
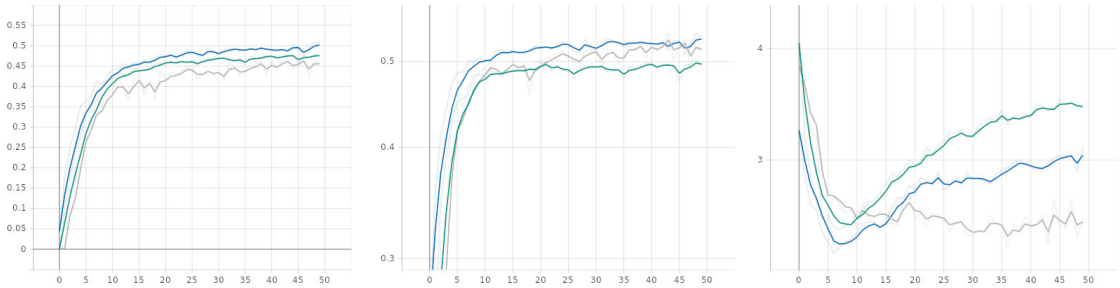


Figure 6.10: Loss per epoch of CRNN-GRU using Mel Spectrogram

The Convolutional Neural Network and the Convolutional Recurrent Neural Networks performed better than the 2 tested Recurrent Neural Network models.

The figures 6.11a, 6.11b and 6.11c compare the validation performance of the 3 models in recall, accuracy and loss respectively. The Convolutional Neural Network is represented in a Gray color, the Convolutional Recurrent Neural Network - Long Short Term Memory is represented in a Blue color, and the Convolutional Recurrent Neural Network - Gated Recurrent Unit is represented in a Green color.



(a) Recall per Epoch (b) Accuracy per Epoch (c) Loss per Epoch

Figure 6.11: Recall, Accuracy and Loss per Epoch - Top 3 models using mel spectrogram

The figure 6.11c shows that the 2 Convolutional Recurrent Neural Networks have a similar loss curve. It also shows that the Convolutional Neural Network becomes less over-fit per epoch than the others, as its loss values seem to decrease or stay constant, as opposed to the 2 other models.

Best Models

This section documents an experiment where the top 3 models of the last experiment are tested for 100 epochs, as the previous tests were only conducted up to 50 epochs.

The top 3 models were the Convolutional Neural Network and the 2 Convolutional Recurrent Neural Networks, and using the Mel Spectrogram with a 3 second sample length. The experiment results are present on the table 6.12.

Model	At Maximum Recall (Validation)							
	Validation			Epoch	Time	Training		
	Recall	Accuracy	Loss			Recall	Accuracy	Loss
CNN	46.67%	51.56%	2.857	89	201m	99.40%	99.52%	0.0426
CRNN LSTM	47.92%	49.93%	3.878	84	209m	96.35%	96.57%	0.144
CRNN GRU	**51.36**%	53.75%	3.020	82	210m	93.76%	95.11%	0.2431

Table 6.12: Top 3 model comparison

Training the models for 50 more epochs only improved performance of the Convolutional Recurrent Neural Network - Gated Recurrent Unit model, resulting in an increase of the recall by 1.19%, accuracy also increased by 0.62%, with the loss only increasing by a negligible 0.017, meaning that model did not over-fit by training for more epochs.

The Convolutional Recurrent Neural Network - Long Short Term Memory model did not improve as expected, as the model was already becoming over-fit on 50 epochs, as shown by its loss curve in the figure 6.11c.

The Convolutional Neural Network, which was the most promising to improve because of its loss curve in the figure 6.11c, also did not improve its recall value, although its accuracy improved by 8.51%.

The recall, accuracy and loss graphs on the validation set of this experiment is presented on the figures 6.12, 6.13 and 6.14.

The lines in the graphs have smoothing applied to them in order to improve readability. They were generated in *TensorBoard* with a 0.5 smoothing value. The original lines without smoothing are slightly visible as shadows behind the smoothed lines.

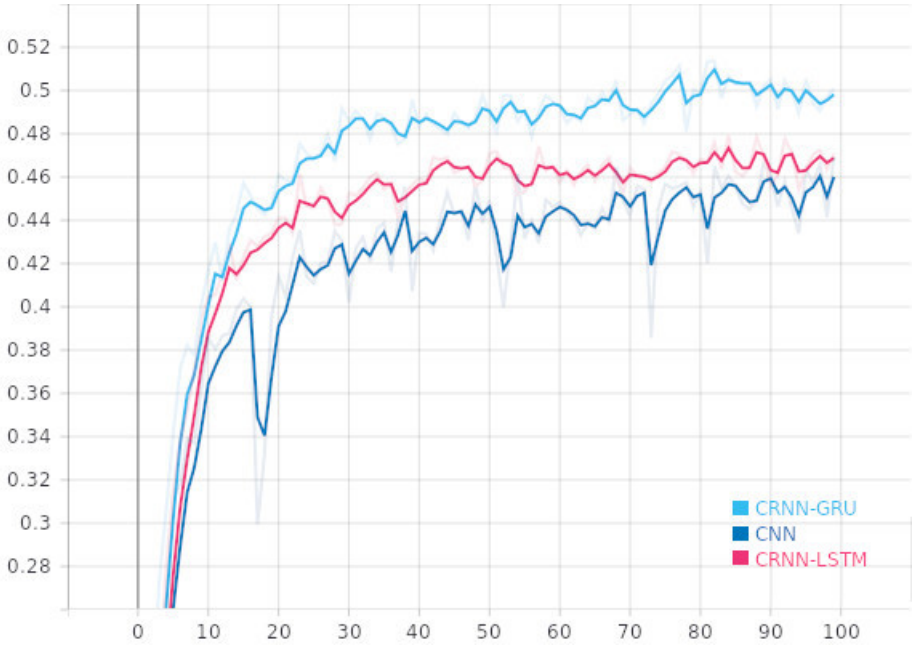


Figure 6.12: Recall per epoch of CNN, CRNN-LSTM and CRNN-GRU using Mel Spectrogram

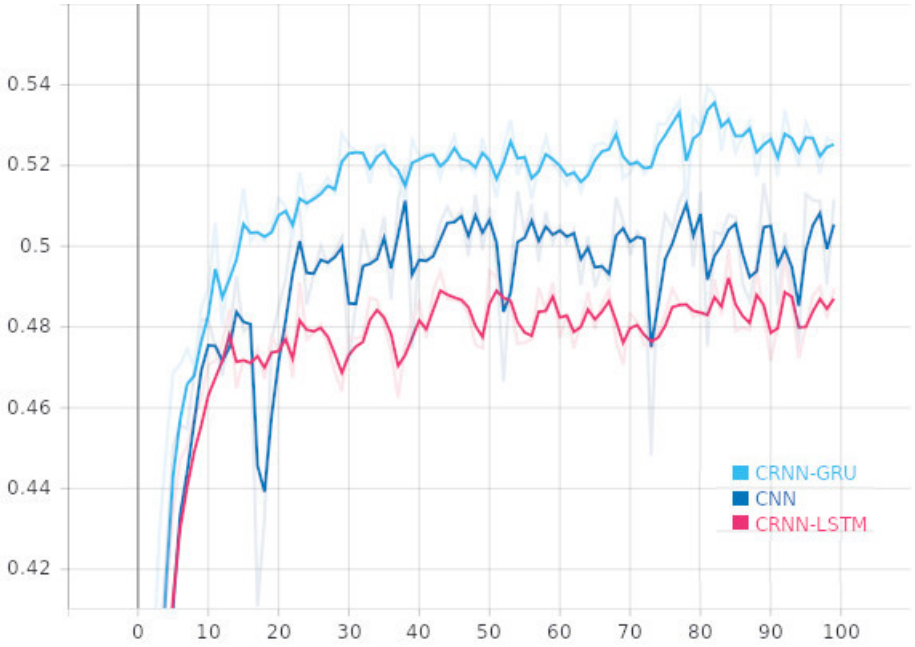


Figure 6.13: Accuracy per epoch of CNN, CRNN-LSTM and CRNN-GRU using Mel Spectrogram

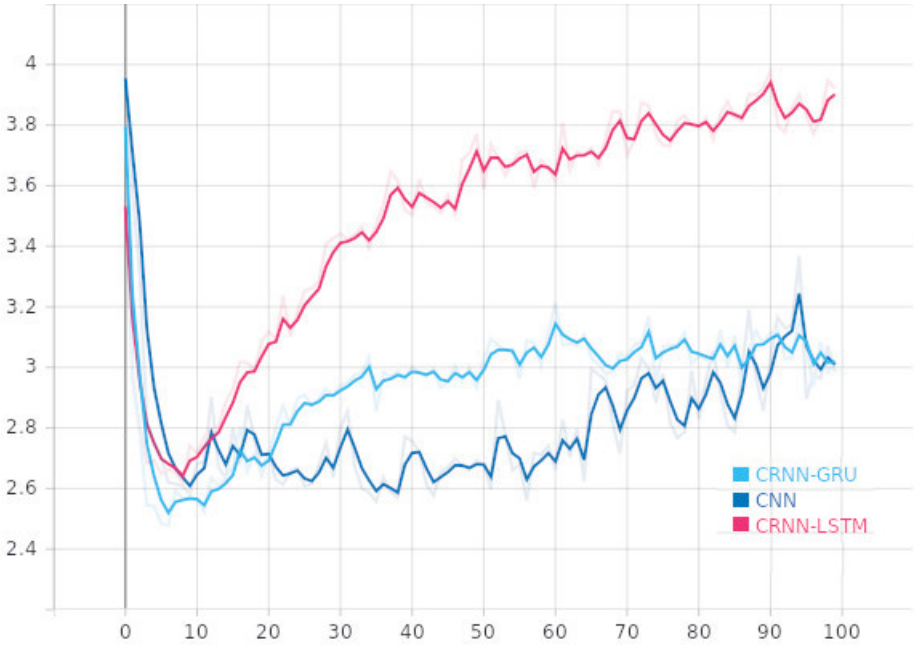


Figure 6.14: Loss per epoch of CNN, CRNN-LSTM and CRNN-GRU using Mel Spectrogram

From all the models tested in this dissertation, the Convolutional Recurrent Neural Network - Gated Recurrent Unit described in the section 5.2.2 is considered the best model architecture for this dissertation.

When used with the Mel Spectrogram with a sample length of 3 seconds as the feature extraction method, the model performed the better than the others, achieving a recall of 51.36% and an accuracy of 53.75% after training for 82 epochs.

Tables 6.13 and 6.14 list all the species in the dataset, and the percentage of the duration relative to the total duration of all species samples on one experiment.

Table 6.13: Percentage of Correct Classifications per Bird Species (1)

Bird Species	Percentage	Bird Species	Percentage
Abert's Towhee	59.56%	Acorn Woodpecker	48.64%
American Bushtit	90.00%	American Crow	44.44%
American Dusky Flycatcher	72.22%	American Grey Flycatcher	65.31%
American Robin	64.29%	Anna's Hummingbird	71.11%
Ash-throated Flycatcher	5.00%	Bell's Sparrow	21.67%
Bell's Vireo	74.70%	Bewick's Wren	27.27%
Black Phoebe	61.36%	Black-chinned Sparrow	10.00%
Black-headed Grosbeak	60.00%	Black-tailed Gnatcatcher	76.92%
Blue-grey Gnatcatcher	15.79%	Brewer's Sparrow	54.72%
California Gnatcatcher	0%	California Quail	64.29%
California Scrub Jay	18.87%	California Thrasher	47.87%
California Towhee	37.50%	Canyon Wren	21.05%

Table 6.14: Percentage of Correct Classifications per Bird Species (2)

Bird Species	Percentage	Bird Species	Percentage
Cassin's Finch	27.78%	Cassin's Vireo	61.54%
Chestnut-backed Chickadee	58.82%	Clark's Nutcracker	76.92%
Common Poorwill	33.33%	Common Yellowthroat	65.31%
Dark-eyed Junco	56.20%	Elegant Tern	77.78%
Flammulated Owl	47.14%	Forster's Tern	51.82%
Great Horned Owl	20.59%	Green-tailed Towhee	68.60%
Grey Vireo	89.36%	Hermit Thrush	44.44%
Hermit Warbler	82.35%	House Finch	50.91%
House Wren	93.22%	Hutton's Vireo	19.23%
Juniper Titmouse	38.46%	Lark Sparrow	54.55%
Lazuli Bunting	65.12%	Lesser Goldfinch	23.88%
Lincoln's Sparrow	87.29%	Long-eared Owl	51.11%
MacGillivray's Warbler	44.23%	Marsh Wren	66.67%
Mountain Chickadee	44.62%	Mountain Quail	64.00%
Northern Flicker	0%	Northern Mockingbird	50.00%
Northern Pygmy Owl	13.10%	Northern Raven	65.22%
Northern Saw-whet Owl	22.92%	Nuttall's Woodpecker	67.92%
Nutting's Flycatcher	44.07%	Oak Titmouse	50.00%
Orange-crowned Warbler	38.61%	Pacific Wren	34.48%
Pacific-slope Flycatcher	46.15%	Phainopepla	18.92%
Pygmy Nuthatch	53.33%	Red Crossbill	77.78%
Red-winged Blackbird	31.03%	Ridgway's Rail	47.78%
Rock Wren	100%	Rufous-crowned Sparrow	21.74%
Slate-colored Fox Sparrow	57.14%	Snow Goose	78.95%
Song Sparrow	35.53%	Spotted Owl	45.45%
Spotted Towhee	47.66%	Steller's Jay	48.15%
Swainson's Thrush	69.09%	Thick-billed Fox Sparrow	71.43%
Tricolored Blackbird	26.67%	Verdin	56.60%
Warbling Vireo	30.23%	Western Meadowlark	58.82%
Western Screech Owl	78.26%	Western Wood Pewee	36.36%
White-breasted Nuthatch	68.15%	White-crowned Sparrow	63.41%
White-headed Woodpecker	38.18%	Wilson's Warbler	59.38%
Wrentit	38.57%	Yellow-billed Magpie	52.94%
Yellow-breasted Chat	28.57%		

6.2.3 Result Comparison

From the experiments in the section 6.2.2, the feature extraction *Mel Spectrogram* with 3 second sample lengths was concluded to be better than the other implementations in the solution, as it performed better than the others overall. The best model architecture that

was implemented in the solution is considered the Convolutional Recurrent Neural Network using Gated Recurrent Unit layers, as it achieved the best results when compared to the other implementations.

This implementation can be compared to the implementation created by Hiatt 2019 (see section 3.4.2). The results from that implementation uses the accuracy at the minimum validation loss epoch, and a 77% training and 33% validation hold-out split.

The following results use the same Mel Spectrogram with the 3 second length and the Convolutional Recurrent Neural Network using Gated Recurrent Units that performed better in this thesis, but to compare with the Hiatt 2019 implementation, the results are taken from the minimum validation loss epoch, and a 77% training and 33% validation hold-out split was also used.

The table 6.15 compares the best implementation in this thesis with the Hiatt 2019 implementation.

Implementation	At Minimum Loss (Validation Set)			
	Validation Set		Epoch	Training Set
	Accuracy	Loss		Accuracy
This Thesis	44.26%	2.657	6	70.74%
Hiatt 2019	19.27%	3.605	85	20.44%

Table 6.15: Result Comparison

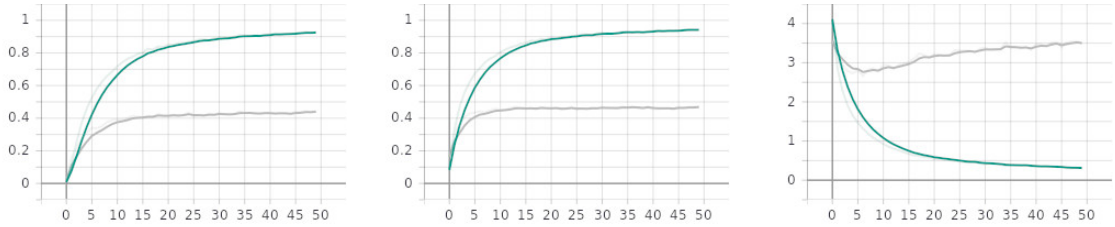
The results present on the table 6.15 show that the best implementation in this dissertation achieved a better accuracy score than the Hiatt 2019 implementation.

For consistency with the rest of this thesis, the table 6.16 adds the usual information documented on the other experiments of this thesis, using the same experiment from the table 6.15.

At Minimum Loss (Validation Set)							
Validation Set			Epoch	Time	Training Set		
Recall	Accuracy	Loss			Recall	Accuracy	Loss
33.84%	44.26%	2.657	6	13m16s	58.11%	70.74%	1.315

Table 6.16: Full results from the Result Comparison experiment

Recall, accuracy and loss graphs from the experiment in this section are presented in the figures 6.15a, 6.15b and 6.15c respectively. The training line is displayed in a dark teal color and the validation line is displayed in a gray color.



(a) Recall per Epoch (b) Accuracy per Epoch (c) Loss per Epoch

Figure 6.15: Recall, Accuracy and Loss per Epoch - Comparing Results experiment

The figures 6.16 and 6.17 present a confusion matrix and the percentage of correct classifications per species, respectively. It uses the model trained to the maximum validation recall value instead of the minimum validation loss value. The implementation used to generate these graphs is based on the Hiatt 2019 implementation (see section 3.4.2 for comparison).

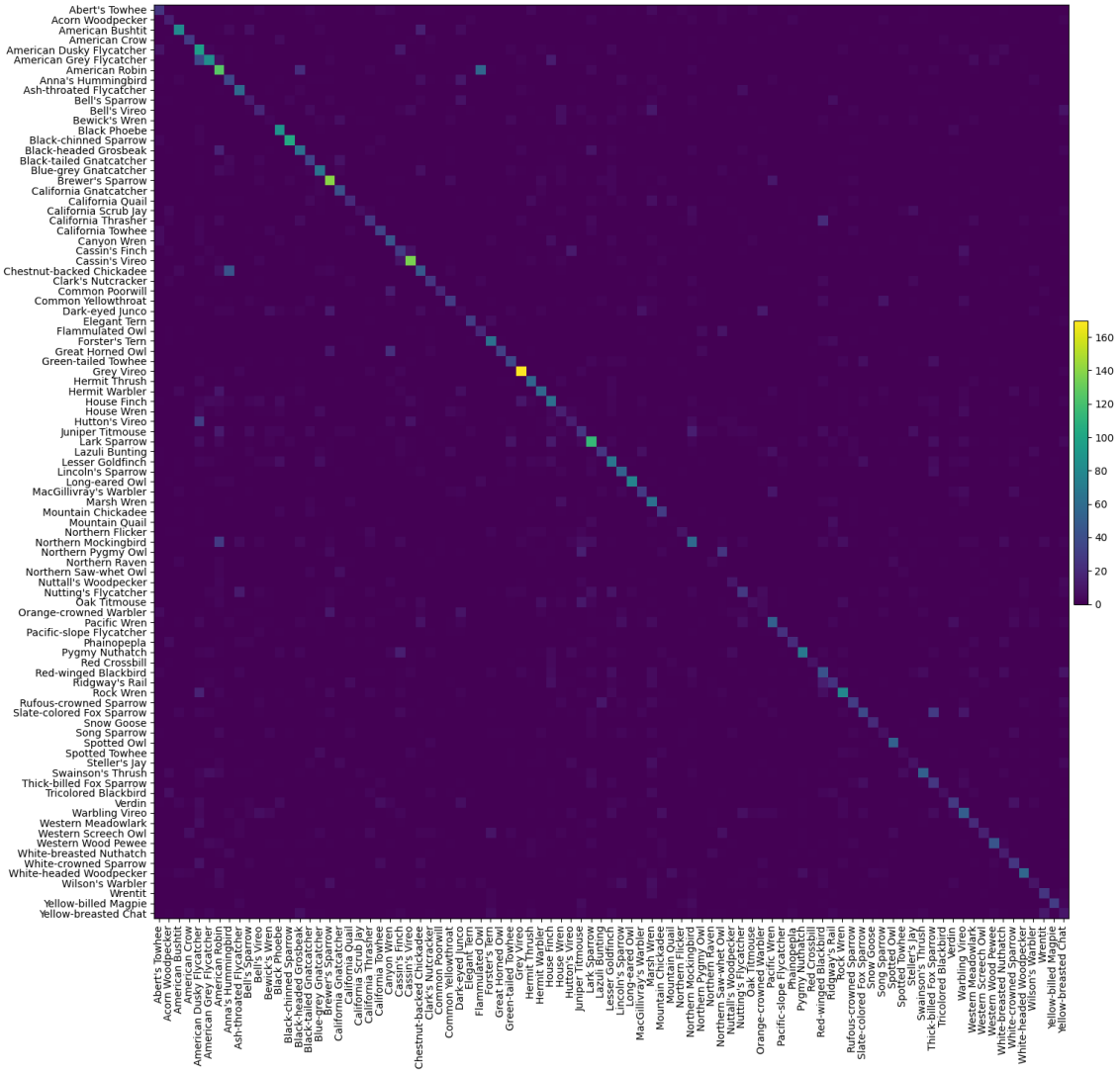


Figure 6.16: Confusion Matrix

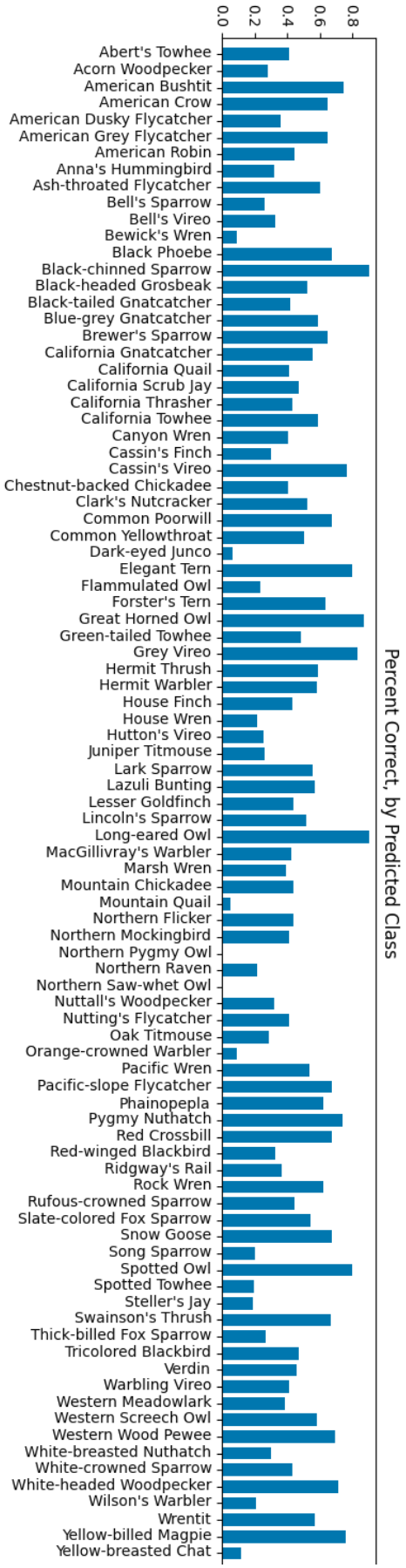


Figure 6.17: Bar Chart with the Percentage of Correct Classifications (Rotated)

6.3 Statistical Test

This section documents the statistical tests to answer the hypothesis presented in the section 6.1.3.

The combinations used for this test are the ones that were compared in the section 6.2.2.

The python package *Autorank* (Herbold 2020) was used to perform the statistical test.

The package only needs a *dataframe* with a minimum of 5 results for each combination to perform the statistical tests, generate a textual description of the results, as well as the plot of the results.

The values used to compare the combinations are the average recall per bird species that was generated from each repetition on the experiment of section 6.2.2 that was used to calculate the mean recall for each combination that is registered in the table 6.11.

The statistical analysis was conducted for 20 populations with 5 paired samples, which are the 20 combinations of feature extractions with the deep learning models, and the 5 mean recall per species values of each combination, respectively. The family-wise significance level of the tests is $\alpha = 0.05$.

The first step is verifying if all populations are normal with the Shapiro-Wilk test (Shapiro and Wilk 1965). The null hypothesis that the population is normally distributed failed to be rejected as the minimal observed probability value (p-value) is 0.16 which is higher than 0.05, therefore, it can be assumed that all populations are normal.

The second step is verifying if all populations have the same variance. The Bartlett's test for homogeneity (Snedecor and Cochran 1967) is applied and the null hypothesis that the data is homoscedastic is rejected, which means that the data can be assumed to be heteroscedastic, and thus, the populations have different variances.

The third step is verifying if at least one of the samples if different from the other samples. The non-parametric Friedman test (Friedman 1940) is used to determine if there are any significant differences between the mean values of the populations. The null hypothesis of the Friedman test that there is no difference in the central tendency of the populations is rejected, therefore it can be assumed that there is a statistically significant difference median values of the populations.

The last step is using the Nemenyi test (Nemenyi 1962) as the post-hoc pairwise test for multiple comparisons to infer which differences are significant. The differences between populations are significant if the difference of the mean rank is greater than the Critical Distance ($CD = 13.26$) of the Nemenyi test.

The results of this test are presented in the table 6.17 and figure 6.18, which ranks the Mel Spectrograms with 3 second sample lengths combined with the Convolutional Recurrent Neural Network using Gated Recurrent Unit layers (3s MELS CRNN-GRU) as the best combination (population). The combinations with the Mean Rank from 1 to 14.2 (see table 6.17) are considered to not be significantly different, as they are within the Critical Distance of 13.26.

The Mean Rank (MR), Mean value (M), Standard Deviation (SD), Confidence Intervals (CI), Effect Size (ES) and the Magnitude of the ES is reported for each combination in the table 6.17.

Combination	MR	M	SD	CI	ES	Magnitude
3s MELS CRNN-GRU	1.000	0.502	0.005	[0.480, 0.523]	0.000	negligible
1.5s MELS CRNN-GRU	2.600	0.479	0.001	[0.477, 0.482]	6.710	large
3s MELS CRNN-LSTM	2.800	0.479	0.005	[0.457, 0.502]	4.635	large
3s MELS CNN	3.600	0.475	0.004	[0.458, 0.492]	6.284	large
1.5s MELS CRNN-LSTM	5.000	0.465	0.001	[0.459, 0.470]	10.824	large
3ms MFCC CRNN-LSTM	6.600	0.448	0.008	[0.412, 0.483]	8.544	large
3s MFCC CRNN-GRU	7.000	0.444	0.004	[0.425, 0.463]	13.126	large
1.5s MELS CNN	7.400	0.442	0.006	[0.413, 0.471]	10.709	large
1.5s MFCC CRNN-GRU	9.000	0.433	0.002	[0.422, 0.445]	18.235	large
1.5s MFCC CRNN-LSTM	10.000	0.424	0.004	[0.403, 0.444]	17.032	large
3s MFCC CNN	11.000	0.405	0.005	[0.382, 0.428]	19.846	large
1.5s MFCC CNN	12.000	0.388	0.004	[0.372, 0.405]	27.187	large
3s MFCC GRU	13.000	0.355	0.005	[0.332, 0.378]	30.355	large
3s MFCC LSTM	14.200	0.324	0.017	[0.244, 0.404]	13.942	large
1.5s MFCC GRU	14.800	0.314	0.004	[0.296, 0.331]	43.559	large
3s MELS GRU	16.400	0.282	0.007	[0.251, 0.314]	37.154	large
1.5s MELS GRU	17.000	0.276	0.011	[0.225, 0.326]	26.826	large
1.5s MFCC LSTM	18.000	0.273	0.003	[0.257, 0.289]	55.243	large
3s MELS LSTM	18.600	0.263	0.009	[0.221, 0.304]	33.262	large
1.5s MELS LSTM	20.000	0.246	0.013	[0.185, 0.307]	25.699	large

Table 6.17: Summary of populations

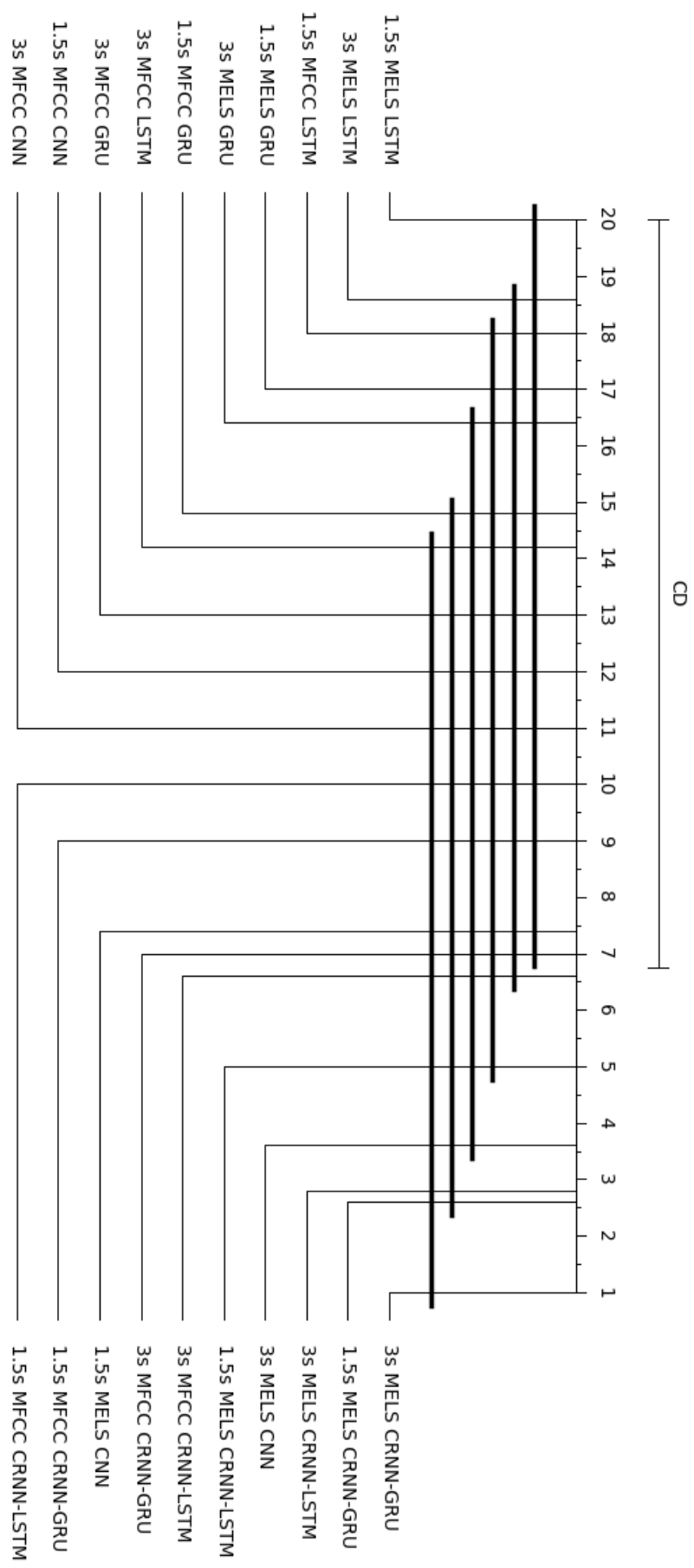


Figure 6.18: Critical Distance Diagram (Rotated)

Chapter 7

Conclusion

The main objective of this dissertation is to contribute to the development of a reliable methodology that classifies bird species by their chirp, using deep learning techniques. If a methodology was always correct on predicting the species of a bird, it could change the way specialists of the domain work, as it would save them the time of manually identifying the bird species.

In this document, an audio pre-processing methodology was presented that normalized the audio, removed noise outside of the bird vocalization range and split an audio file into multiple equal length samples by detecting syllables or peaks. The audio samples then have their features extracted in one of two methods. These are the Mel Spectrograms and the Mel Frequency Cepstrum Coefficients.

The classification methodology for this solution uses one of the two types of extracted features to train a Deep Learning model, which can then be used to predict the species of a bird, by first pre-processing the audio file using the same audio pre-processing methodology that was used on the dataset used to train the model.

An optimization methodology was created with the intent of testing and refining both the pre-processing methodology and the Deep Learning models. During the experiments conducted using this methodology, multiple alternative Deep Learning model architectures were tested, and three alternatives of each model are documented in the section 6.2.1. The deep learning models are different alternatives of Convolutional Neural Networks, Recurrent Neural Networks and Convolutional Recurrent Neural Networks. The alternatives which performed the best of each deep learning model type are present in the section 5.2.2, and are used for the following experiments.

The second experiment methodology of this dissertation uses one alternative of each Deep Learning model type that scored the best in the optimization experiments. These models are the ones in the section 5.2.2. The methodology also tests four different types of input data, which are two versions of the Mel Spectrogram and the Mel Frequency Cepstrum Coefficients, one with a 3 second sample length and the other with 1.5 second sample length. Each of the 4 types of input data is combined with the 5 deep learning models, resulting in 20 different combinations for this experiment. This experiment concluded that the Deep Learning models perform better with the Mel Spectrogram feature extraction with a 3 second sample length. These experiments are documented on the section 6.2.2.

The third experiment uses the best models of the last experiment, which were the Convolutional Recurrent Neural Network using Gated Recurrent Unit layers, the Convolutional Recurrent Neural Network using Long Short Term Memory layers and the Convolutional Neural Network. This experiment is conducted for 100 epochs instead of the 50 used for

the other experiments, and used the Mel Spectrogram feature extraction with the 3 second sample length. These experiments are documented on the section 6.2.2.

The third experiment concluded that the deep learning model that performed better is the Convolutional Recurrent Neural Network using Gated Recurrent Unit layers (see section 5.2.2, table 6.12), which, on the validation set (20%), obtained a recall of 51.36% and an accuracy of 53.75%.

The final experiment is comparing the best methodology obtained in this thesis with another implementation (Hiatt 2019), while using the same experiment environment. The methodology from thesis performed better, achieving an Accuracy value on the validation set of 44.26%, while the other obtained 19.27%. This comparison is documented on the section 6.2.3.

7.1 Further Improvements

In a perspective of continuing this work, the following points could be considered:

- Improving the audio pre-processing methodology: Other audio pre-processing and feature extraction techniques could be tested, some of these are mentioned in the State of the Art such as:
  - Data Preparation and Data Augmentation techniques, like the ones used by Lasseck 2019 (see section 3.4.1);
  - Feature Extraction methods (see section 3.3.2).
- Use further optimized Deep Learning models: Although multiple variations of the models are presented along this document, there can be better implementations with different architectures and hyper-parameters. Transfer learning and techniques such as Lowering the Learning Rate could also be used, as well as training the models for more epochs.
- Conduct more experiments using different types of bird sound datasets.

Bibliography

Abadi, Martin et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. url: https://www.tensorflow.org/.

Adams, Seth (2020). *Audio-Classification*. [Online; accessed 7. Jun. 2020]. url: https://github.com/seth814/Audio-Classification/tree/2f0032d81dcfa3d662cab1c1c4e7e30520f7edd6.

Arunava (2018). "Convolutional Neural Network". In: *Medium*. issn: 1777-7605. url: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05 (visited on 12/27/2019).

Ayoub, Bouziane, Kharroubi Jamal, and Zarghili Arsalane (2016). "Gammatone frequency cepstral coefficients for speaker identification over VoIP networks". In: *2016 International Conference on Information Technology for Organizations Development (IT4OD)*. doi: 10.1109/IT4OD.2016.7479293.

Backend - Keras Documentation (2019). [Online; accessed 19. Jan. 2020]. url: https://keras.io/backend.

Barnes, Cindy, Helen Blake, and David Pinder (2009). *Creating and Delivering Your Value Proposition: Managing Customer Experience for Profit*. Kogan Page Publishers. isbn: 978-074945859-1. url: https://books.google.pt/books/about/Creating_and_Delivering_Your_Value_Propo.html?id=8d73CPt_khwC&redir_esc=y.

Boddapati, Venkatesh et al. (2017). "Classifying environmental sounds using image recognition networks". In: *Procedia Comput. Sci.* 112, pp. 2048–2056. issn: 1877-0509. doi: 10.1016/j.procs.2017.08.250.

Brownlee, Jason (2019a). *A Gentle Introduction to Padding and Stride for Convolutional Neural Networks*. url: https://machinelearningmastery.com/what-is-deep-learning/ (visited on 12/24/2019).

– (2019b). *How to Evaluate the Skill of Deep Learning Models*. url: https://machinelearningmastery.com/evaluate-skill-deep-learning-models/ (visited on 01/25/2019).

– (2019c). *What is Deep Learning?* url: https://machinelearningmastery.com/what-is-deep-learning/ (visited on 12/21/2019).

Butterworth, Stephen et al. (1930). "On the theory of filter amplifiers". In: *Wireless Engineer* 7.6, pp. 536–541.

Chatterjee, Chandra Churh (2019). "An Approach Towards Convolutional Recurrent Neural Networks". In: *Medium*. url: https://towardsdatascience.com/an-approach-towards-convolutional-recurrent-neural-networks-a2e6ce722b19 (visited on 12/28/2019).

Cho, Kyunghyun et al. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv: 1406.1078 [cs.CL].

Choi, Keunwoo et al. (2016). *Convolutional Recurrent Neural Networks for Music Classification*. arXiv: 1609.04243 [cs.NE].

Chollet, François et al. (2015). *Keras*. https://keras.io.

Colonna, Juan Gabriel et al. (2012). "Feature subset selection for automatically classifying anuran calls using sensor networks". In: *The. 2012 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. issn: 2161-4393. doi: 10.1109/IJCNN.2012.6252794.

Colonna, Juan et al. (2016). "Automatic Classification of Anuran Sounds Using Convolutional Neural Networks". In: *ResearchGate*, pp. 73–78. doi: 10.1145/2948992.2949016.

Corder, Gregory and Dale Foreman (2009). "Nonparametric Statistics for Non-Statisticians: A Step-By-Step Approach". In: pp. 79–80. doi: 10.1002/9781118165881.

D., F. (2019). "Batch normalization in Neural Networks - Towards Data Science". In: *Medium*. url: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c.

Davis, S. and P. Mermelstein (1980). "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: pp. 357–366. issn: 0096-3518. doi: 10.1109/TASSP.1980.1163420.

Deng, Jia et al. (2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. issn: 1063-6919. doi: 10.1109/CVPR.2009.5206848.

Deshpande, Adit (2019). *A Beginner's Guide To Understanding Convolutional Neural Networks*. url: https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/ (visited on 12/23/2019).

EliteDataScience.com (2017). *Overfitting in Machine Learning*. [Online; accessed 15. Feb. 2020]. url: https://elitedatascience.com/overfitting-in-machine-learning.

Fagerlund, Seppo (2007). "Bird Species Recognition Using Support Vector Machines". In: *EURASIP J. Adv. Signal Process.* 2007.1, pp. 1–8. issn: 1687-6180. doi: 10.1155/2007/38637.

Friedman, Milton (1940). "A Comparison of Alternative Tests of Significance for the Problem of m Rankings". In: *Ann. Math. Statist.* 11.1, pp. 86–92. doi: 10.1214/aoms/1177731944. url: https://doi.org/10.1214/aoms/1177731944.

Gavali, Prof. Pralhad et al. (2019). "Bird Species Identification using Deep Learning". In: *International Journal of Engineering Research & Technology* 8.4. issn: 2278-0181. url: https://www.ijert.org/bird-species-identification-using-deep-learning.

Gu, Jiuxiang et al. (2015). *Recent Advances in Convolutional Neural Networks*. arXiv: 1512.07108 [cs.CV].

Herbold, Steffen (2020). "Autorank: A Python package for automated ranking of classifiers". In: *Journal of Open Source Software* 5, p. 2173. doi: 10.21105/joss.02173.

Hiatt, Sam (2019). "Avian Vocalizations - Report". In: *Kaggle*. url: https://www.kaggle.com/samhiatt/avian-vocalizations-report.

– (2020). *Avian Vocalizations from CA and NV, USA - Kaggle*. [Online; accessed 1. Jul. 2020]. url: https://www.kaggle.com/samhiatt/xenocanto-avian-vocalizations-canv-usa.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Comput.* 9.8, pp. 1735–1780. issn: 0899-7667. doi: 10.1162/neco.1997.9.8.1735. url: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Huang, Chenn-Jung et al. (2009). "Frog classification using machine learning techniques". In: *Expert Syst. Appl.* 36.2, Part 2, pp. 3737–3743. issn: 0957-4174. doi: 10.1016/j.eswa.2008.02.059.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95. doi: 10.1109/MCSE.2007.55.

Jain, Vandit (2020). "Everything you need to know about "Activation Functions" in Deep learning models". In: *Medium*. issn: 8498-2253. url: https://towardsdatascience.

com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253.

Kahl, Stefan et al. (2019). "Overview of BirdCLEF 2019: Large-Scale Bird Recognition in Soundscapes". In: *CLEF*.

Kaminska, Dorota, Tomasz Sapinski, and Adam Pelikant (2013). "Comparison of perceptual features efficiency for automatic identification of emotional states from speech". In: *2013 6th International Conference on Human System Interactions (HSI)*, pp. 210–213. issn: 2158-2254. doi: 10.1109/HSI.2013.6577824.

Keras - TensorFlow Core (2020). [Online; accessed 27. Jan. 2020]. url: https://www.tensorflow.org/guide/keras.

Kharkovyna, Oleksii (2019). "Top 10 Best Deep Learning Frameworks in 2019". In: *Medium*. url: https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de.

Koen, Peter A et al. (2002). "Fuzzy front end: effective methods, tools, and techniques". In: *The PDMA toolbook 1 for new product development*.

Kortas, Magdalena (2020). "Sound-Based Bird Classification". In: *Medium*. url: https://towardsdatascience.com/sound-based-bird-classification-965d0ecacb2b.

Kostadinov, Simeon (2019). "Understanding GRU Networks". In: *Medium*. url: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be (visited on 12/28/2019).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Kumar, Pawan et al. (2010). "Spoken Language Identification Using Hybrid Feature Extraction Methods". In: *ArXiv* abs/1003.5623.

Kumawat, Dinesh (2020). *7 Types of Activation Functions in Neural Network - Analytics Steps*. [Online; accessed 1. Jul. 2020]. url: https://www.analyticssteps.com/blogs/7-types-activation-functions-neural-network.

Lanning, Michael J. (2014). "Delivering Profitable Value A Revolutionary Framework to Accelerate Growth, Generate Wealth, and Rediscover the Heart of Business". In:

Lasseck, Mario (2019). "Bird Species Identification in Soundscapes". In: *CLEF*.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lindic, Jaka and Carlos Marques Da Silva (2011). "Value proposition as a catalyst for a customer focused innovation". In: *Management Decision*. doi: 10.1108/00251741111183834.

Loughran, Róisín et al. (2009). "The Use of Mel-frequency Cepstral Coefficients in Musical Instrument Identification". In: *International Computer Music Conference Proceedings* 2008. issn: 2223-3881.

Lyons, James et al. (2020). "jameslyons/python_speech_features: release v0.6.1". In: *Zenodo*. doi: 10.5281/zenodo.3607820.

Martinsson, John (2017). "Bird Species Identification using Convolutional Neural Networks". PhD thesis. url: https://odr.chalmers.se/handle/20.500.12380/249467.

McFee, Brian et al. (2020). "librosa/librosa: 0.7.2". In: doi: 10.5281/zenodo.3606573.

Mel - Simon Fraser University (2005). [Online; accessed 5. Jun. 2020]. url: https://www.sfu.ca/sonic-studio-webdav/handbook/Mel.html.

Minaee, Shervin (2019). "20 Popular Machine Learning Metrics. Part 1: Classification & Regression Evaluation Metrics". In: *Medium*. url: https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce.

MissingLink.ai (2019). *Convolutional Neural Network Architecture: Forging Pathways to the Future*. url: https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-architecture-forging-pathways-future (visited on 12/23/2019).

Mittal, Aditi (2019). "Understanding RNN and LSTM". In: *Medium*. url: https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e (visited on 12/27/2019).

Montantes, James (2019). "TensorFlow vs PyTorch vs Keras for NLP - Exxact". In: *Medium*. url: https://towardsdatascience.com/tensorflow-vs-pytorch-vs-keras-for-nlp-exxact-8e51dd13c3f5.

Nemenyi, Peter (1962). "Distribution-free multiple comparisons". In: *Biometrics*. Vol. 18. 2. International Biometric Soc 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, p. 263.

Nguyen, Michael (2019). "Illustrated Guide to LSTM's and GRU's: A step by step explanation". In: *Medium*. url: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (visited on 12/28/2019).

Nicholson, Chris (2019). *A Beginner's Guide to LSTMs and Recurrent Neural Networks*. url: https://pathmind.com/wiki/lstm (visited on 12/27/2019).

Nicola, Susana, Eduarda Pinto Ferreira, and J. J. Pinto Ferreira (2012). "A NOVEL FRAMEWORK FOR MODELING VALUE FOR THE CUSTOMER, AN ESSAY ON NEGOTIATION". In: *Int. J. Info. Tech. Dec. Mak.* 11.03, pp. 661–703. issn: 0219-6220. doi: 10.1142/S0219622012500162.

Olah, Christopher (2019). *Understanding LSTM Networks*. url: https://colah.github.io/posts/2015-08-Understanding-LSTMs (visited on 12/27/2019).

Osterwalder, Alexander and Yves Pigneur (2010). *Business model generation: a handbook for visionaries, game changers, and challengers*. John Wiley & Sons.

Patterson, Roy D et al. (1987). "An efficient auditory filterbank based on the gammatone function". In: *a meeting of the IOC Speech Group on Auditory Modelling at RSRE*. Vol. 2. 7.

Perambai, Abhishek (2019). "A deep dive into the world of gated Recurrent Neural Networks: LSTM and GRU". In: *Medium*. url: https://medium.com/analytics-vidhya/lstm-and-gru-a-step-further-into-the-world-of-gated-rnns-99d07dac6b91 (visited on 12/28/2019).

Piczak, Karol J. (2015). *Environmental sound classification with convolutional neural networks*. IEEE. doi: 10.1109/MLSP.2015.7324337.

Roberts, Leland (2020). "Understanding the Mel Spectrogram - Analytics Vidhya - Medium". In: *Medium*. url: https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53.

Saaty, R. W. (1987). "The analytic hierarchy process—what it is and how it is used". In: *Mathematical Modelling* 9.3, pp. 161–176. issn: 0270-0255. doi: 10.1016/0270-0255(87)90473-8.

Saaty, Thomas L (1988). "What is the analytic hierarchy process?" In: *Mathematical models for decision support*. Springer, pp. 109–121.

Saha, Sumit (2018). "A Comprehensive Guide to Convolutional Neural Networks the ELI5 way". In: *Medium*. issn: 3211-6453. url: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 12/23/2019).

Shah, Parth, Vishvajit Bakarola, and Supriya Pati (2018). "Optimal Approach for Image Recognition using Deep Convolutional Architecture". In: *ArXiv* abs/1904.11187.

Shao, Yang et al. (2009). "An auditory-based feature for robust speech recognition". In: pp. 4625–4628. doi: 10.1109/ICASSP.2009.4960661.

Shapiro, S. S. and M. B. Wilk (1965). "An Analysis of Variance Test for Normality (Complete Samples)". In: *Biometrika* 52.3/4, pp. 591–611. issn: 00063444. url: http://www.jstor.org/stable/2333709.

Sharma, Pulkit (2019). *Top 5 Deep Learning Frameworks, their Applications, and Comparisons!* [Online; accessed 27. Jan. 2020]. url: https://www.analyticsvidhya.com/blog/2019/03/deep-learning-frameworks-comparison.

Snedecor, George Waddel and William Gemmell Cochran (1967). *Statistical methods*. Iowa state university press.

Srinivasan, Aishwarya V. (2019). "Stochastic Gradient Descent—Clearly Explained !!" In: *Medium*. url: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.

Stevens, Stanley Smith, John Volkmann, and Edwin B Newman (1937). "A scale for the measurement of the psychological magnitude pitch". In: *The Journal of the Acoustical Society of America* 8.3, pp. 185–190.

Szegedy, Christian et al. (2015). "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

Ulaga, Wolfgang and Andreas Eggert (2006). "Value-Based Differentiation in Business Relationships: Gaining and Sustaining Key Supplier Status". In: *Journal of Marketing - J MARKETING* 70, pp. 119–136. doi: 10.1509/jmkg.2006.70.1.119.

Wielgat, Robert et al. (2007). "HFCC based recognition of bird species". In: *Signal Processing Algorithms, Architectures, Arrangements, and Applications SPA 2007*, pp. 129–134. issn: 2326-0319. doi: 10.1109/SPA.2007.5903313.

Woodall, Tony (2003). "Conceptualising 'Value for the Customer': An Attributional, Structural and Dispositional Analysis". In: *Academy of Marketing Science Review* 12.

Xeno-Canto (2020). [Online; accessed 11. Feb. 2020]. url: https://www.xeno-canto.org.

Appendix A

Calculations for the Analytic Hierarchy Process

The fundamental scale defined by R. W. Saaty 1987 was used to obtain the values in the table A.1.

Table A.1: Comparison Matrix

Criteria	**Accuracy**	**Loss**	**Training Time**
Accuracy	1	3	5
Loss	1/3	1	3
Training Time	1/5	1/3	1
Column Total	23/15	13/3	9

The accuracy was established to be the most important criteria, the loss being the second, and the least important the training time.

The table A.2 presents the normalized comparison matrix with the priority vector, which is the weight of the criteria.

Table A.2: Normalized comparison matrix with estimated weights

Criteria	**Accuracy**	**Loss**	**Training Time**	**Weights**
Accuracy	0.6522	0.6923	0.5556	0.7235
Loss	0.2174	0.2308	0.3333	0.2605
Training Time	0.1304	0.0769	0.1111	0.1061
				1.000

The normalized comparison matrix with the priority vector shows that the most important criteria is the accuracy (0.7231), followed by the loss (0.2157) and finally the training time (0.0612).

The Consistency Ratio (CR) can be used to verify the consistency of the values from the matrix. To calculate it, λ_{max} which is the maximum Eigen value must be first obtained.

It can be obtained by first multiplying the comparison matrix with the priority vector, which originates a new vector, as shown in table A.3. Each element of this new vector can then

be divided by the each respective priority vector element, and the average of the values is the Eigen value λ_{max}.

Table A.3: Calculating the new vector

1	3	5		0.7235		2.0355
1/3	1	1	x	3	=	0.8200
1/5	1/3	1		0.1061		0.3376

$$\lambda_{max} = average(\frac{2.0355}{0.7235}, \frac{0.8200}{0.2605}, \frac{0.3376}{0.1061}) = 3.0477 \quad (A.1)$$

Now the Consistency Index (CI) can be obtained with the following formula, where 'n' is the matrix order:

$$CI = \frac{\lambda_{max} - n}{n-1} = \frac{3.0477 - 3}{3-1} = 0.0239 \quad (A.2)$$

Then, to complete the consistency verification, the CR needs to be calculated. It is obtained by dividing the CI by the Random Index (RI), which can be obtained from table A.4.

Table A.4: Random Index values for n order squared matrices

1	2	3	4	5	6	7	8	9	10	11
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51

$$CR = \frac{CI}{RI} = \frac{3.0477 - 3}{3-1} = 0.0412 \quad (A.3)$$

As CR is lower than 0.1, the values of the relative priorities are consistent.

Now the peer comparison matrix can be constructed, using each of the selected alternatives. The matrix is also normalized in order to obtain the priority vector.

After the priority vector values (or weights) for all of the alternatives are obtained the alternative priority can be calculated. To calculate it, a matrix with the values of the priority vector values is created. This matrix is then multiplied by the criteria priority vector. The result is the alternative priority vector.

Table A.5: Peer comparison matrix for the criteria accuracy

Accuracy	**LeNet**	**AlexNet**	**GoogLeNet**
LeNet	1	1/2	2
AlexNet	2	1	3
GoogLeNet	1/2	1/3	1
Total	3.5	1.8333	6

Table A.6: Normalized peer comparison matrix for the criteria accuracy with priority vector (weight)

Accuracy	**LeNet**	**AlexNet**	**GoogLeNet**	**Weight**
LeNet	0.2857	0.2727	0.3333	0.2972
AlexNet	0.5714	0.5455	0.5	0.5390
GoogLeNet	0.1429	0.1818	0.1667	0.1638
				1.0000

Table A.7: Peer comparison matrix for the criteria loss

Loss	**LeNet**	**AlexNet**	**GoogLeNet**
LeNet	1	1/3	3
AlexNet	3	1	5
GoogLeNet	1/3	1/5	1
Total	4.3333	1.5333	9

Table A.8: Normalized peer comparison matrix for the criteria loss with priority vector (weight)

Loss	**LeNet**	**AlexNet**	**GoogLeNet**	**Weight**
LeNet	0.2308	0.2174	0.3333	0.2605
AlexNet	0.6923	0.6522	0.5555	0.6333
GoogLeNet	0.0769	0.1304	0.1111	0.1061
				1.0000

Table A.9: Peer comparison matrix for the criteria training time

Training Time	**LeNet**	**AlexNet**	**GoogLeNet**
LeNet	1	5	9
AlexNet	1/5	1	5
GoogLeNet	1/9	1/5	1
Total	1.3111	6.2	15

Table A.10: Normalized peer comparison matrix for the criteria training time with priority vector (weight)

Training Time	**LeNet**	**AlexNet**	**GoogLeNet**	**Weight**
LeNet	0.7627	0.8065	0.6	0.7231
AlexNet	0.1525	0.1613	0.3333	0.2157
GoogLeNet	0.0847	0.0323	0.0667	0.0612
				1.0000

Table A.11: Alternative priority calculation

	Accuracy	**Loss**	**Training Time**		**Criteria Priority**		**Alternative Priority**
LeNet	0.2972	0.2605	0.7231		0.7235		0.3596
AlexNet	0.5390	0.6333	0.2157	x	0.2605	=	**0.5778**
GoogLeNet	0.1638	0.1061	0.0612		0.1061		0.1526