MSc DATA SCIENCE AND ANALYTICS DISSERTATION

HIERARCHICAL IDENTIFICATION OF BATS FROM CALL ACOUSTICS USING DEEP NEURAL NETWORKS

Author:
Japneet Singh

Supervisor:
Dr. Rafael DeAndrade Moral

A thesis submitted in fulfilment of the requirements for the degree of Masters in Data Science And Analytics 2019-2020

to the

Department of Mathematics & Statistics
Maynooth University

13th August 2020

ACKNOWLEDGEMENT

I would like to acknowledge and thank the following people who not only supported me during this dissertation but throughout the graduation.

I would first like to thank my dissertation supervisor Dr. Rafael DeAndrade Moral from the Mathematics and Statistics department at Maynooth University. His words of constant encouragement & meticulous reviews helped me to improve the quality of work in many aspects. He consistently allowed this dissertation to be my own work but steered me in the right direction whenever he thought I needed it. The availability and constant support from his end are highly appreciable.

Moreover, I would like to thank Dr. Catherine Hurley for the academic support and learning she offered throughout the year. I would also like to thank Dr. Barack Pearlmutter for lecturing the module on Machine Learning and Neural Networks that motivated me to pursue this study.

Last but not the least, without the unconditional and overwhelming support of my family back home in India, it would never have been possible.

Table of Contents

ABSTRACT

Tracing the biological hierarchy and narrowing down to the identification of the species of bats remain a historically challenging and dynamic topic of research in the domain of biology. Capturing the attention of the enthusiasts, the traces of research in this study involves the critical analysis of the most prominent trait of the bats – their calls – which is further blended with the concepts of Machine Learning and viewed as a supervised learning problem with the acoustic decomposition of the bat calls as the features. However, with the emergence of Deep Learning, the exploration of this multi-class classification problem in the context of Neural Networks is yet unexplored. Exploring through the lens of Deep Learning, the objective of this study revolves around the hierarchical classification of the family, genus, and species of bats using deep neural networks, taking the acoustic components of the bat calls as the input and simultaneously compare the model performances to the previously implemented counterparts. Delving into the nuances of deep learning in the context of this data, the study also attempts to break into the black-box mechanism of neural networks to explore the yet unexplored feature importance and feature ranking approaches. Finally, I took a little detour to explore another fascinating aspect of this study which investigates the association of the geographical location of the recorded bat call with its acoustic components. This is believed to be extremely beneficial for the biologists in identifying the geographic region in which a particular call was recorded, which may provide answers to a plethora of unanswered questions on the locational characteristics of the species.

Keywords: Deep Neural Networks, Backpropagation, weights, loss function, dropout, Feature Ranking, Keras, Sequential model, Multi-class confusion matrix.

Chapter 1

INTRODUCTION

The identification of bats from the acoustic characteristics of their calls has been a historic area of research for the biologists. However, the emergence of machine learning techniques in the computer science domain triggered a new revolution and transformed this identification of bats to a supervised learning problem. Motivated by this idea, a team of researchers in 2016 proposed a random forest classifier that opened the research possibilities in this domain than just providing an apparent solution to the problem (Zamora-Gutierrez, 2016). Driven by the same motivation with an aim to contribute further to this area, this study tends to apply the concepts of deep learning for the hierarchical classification of bat species from the acoustics, further exploring an additional aspect that identifies the geographical region in which a bat call was recorded.

Before we jump to the concepts of neural networks and their implementation in the context of this, it would be necessary to get accustomed to the layout of the bat-calls dataset and understand the major aspects that are necessary for this analysis.

1.1 Exploratory Data Analysis

The bat-calls dataset, as described by (Zamora-Gutierrez, 2016), is the numerically condensed version of the audio recordings of fifty-nine bat species gathered across ten different geographical locations. The biological hierarchy followed by the bats holds family at the top, having several genus at the second level and the species at the last level. In this data, there are 8 families of bats having a total of 32 different genus which in turn splits into 59 distinct species. The proportional breakdown of the recorded call acoustics across the hierarchy is visualized in Figure 1.

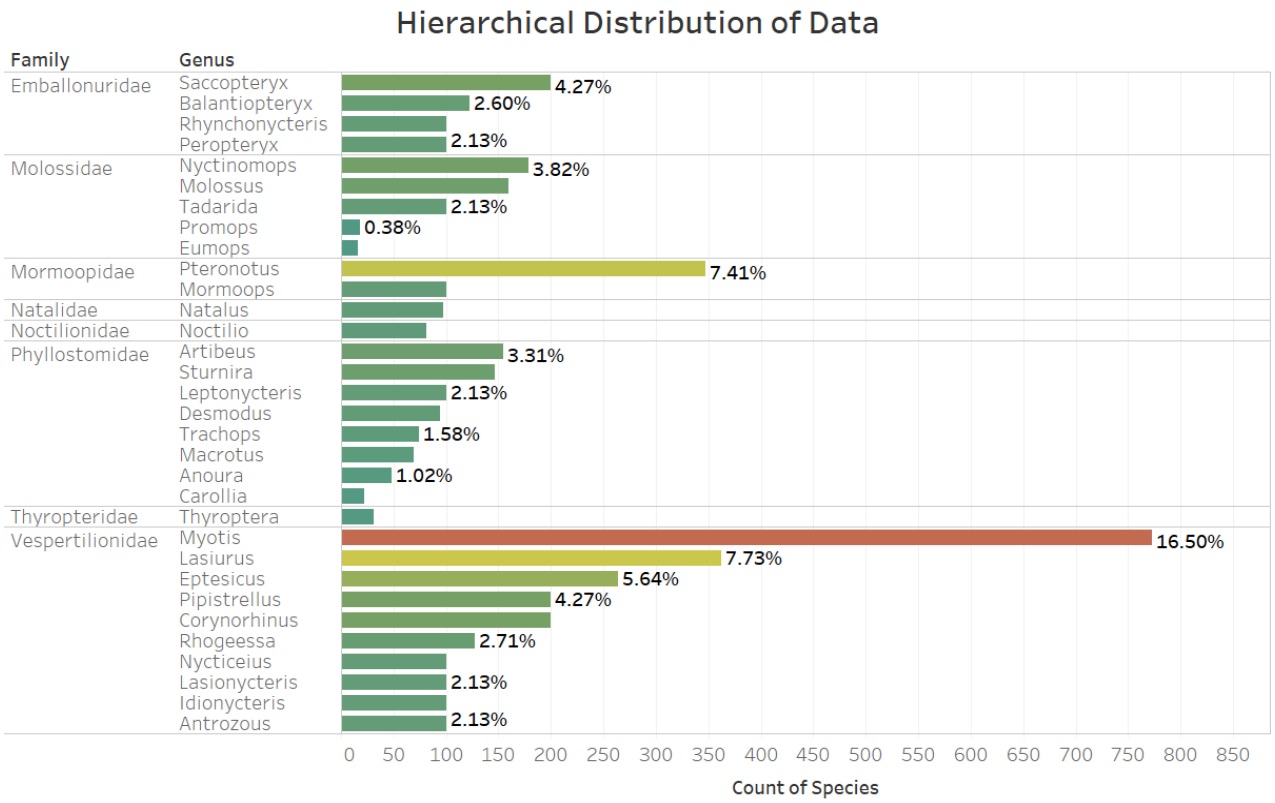


Figure 1: The biological hierarchy of bats indicating the proportion of observed Species within each Genus.

Illuminating the ecological characteristics of bats, there are six different guilds that differentiate the bats on their ecological traits. Another seemingly simple yet interesting features in the data that provide an insight into the data-gathering stage are file identifier that denotes the audio file from which the call acoustics are extracted and the owner which denotes the organization that recorded the bat call audios. Figure 2 describes the distribution of subject Families concerning the Owner giving a proportional breakdown of the data with respect to the Owner. From the figure, we see that the owner VZG collected the highest proportion of the data, with a majority of Observations coming from the *Vespertilionidae* family.

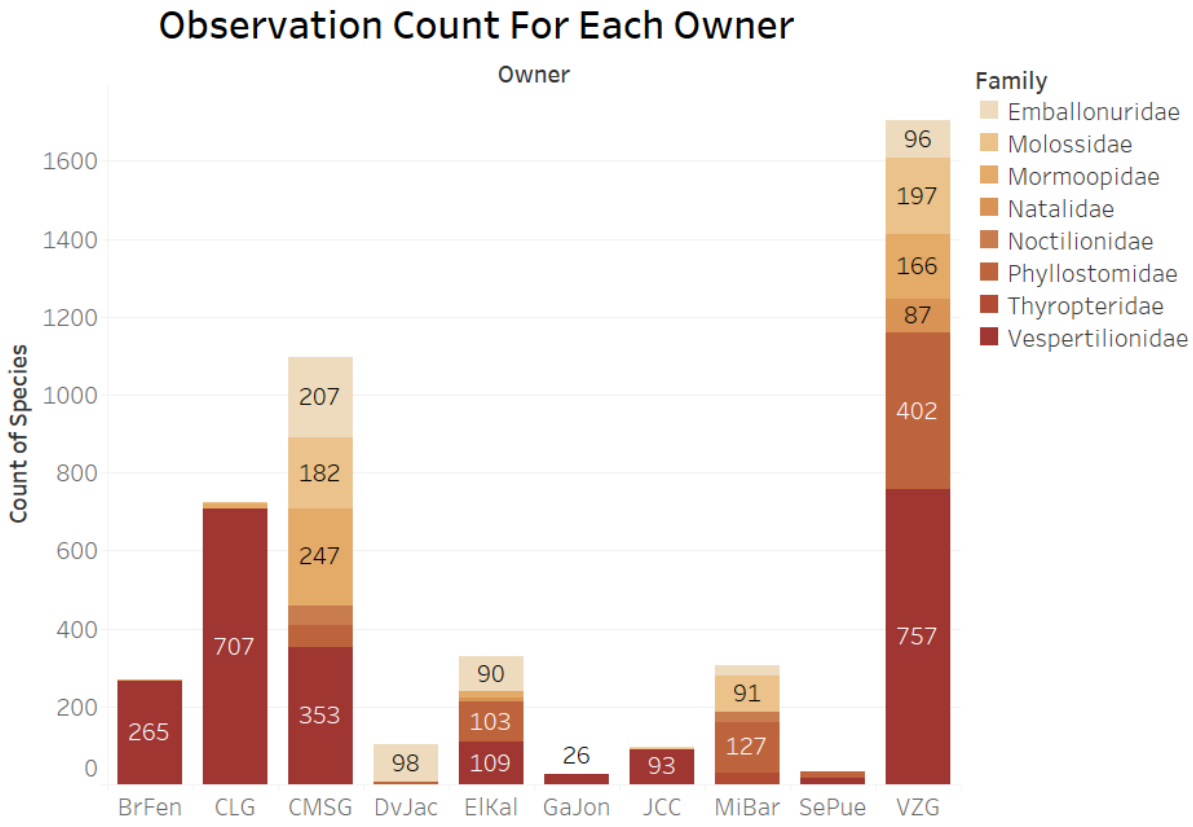


Figure 2: Number of distinct Species calls recorded by each Owner

Further, there are seventy-three numeric features that comprehensively describe the acoustic composition of an observed bat call and lastly, there is a column fold that states the number of folds used in cross-validation for random forest classifier deployed by Zamora-Gutierrez, which would be irrelevant for this study. For a glimpse of the initial 5 rows and some of the features, see Table 1.

File	Owner	Family	Genus	Species	Guild	Country	Quality	CallDuration
Anogeo-456-Maple_M00012	VZG	Phyllostomidae	Anoura	Anogeo	6	Mexico	0.434226	2.50367
Anogeo-456-Maple_M00012	VZG	Phyllostomidae	Anoura	Anogeo	6	Mexico	0.519223	2.547593
Anogeo-456-Maple_M00012	VZG	Phyllostomidae	Anoura	Anogeo	6	Mexico	0.631567	2.312634
Anogeo-456-Maple_M00012	VZG	Phyllostomidae	Anoura	Anogeo	6	Mexico	0.367486	2.233911
Anogeo-456-Maple_M00012	VZG	Phyllostomidae	Anoura	Anogeo	6	Mexico	0.590776	1.62519

Table 1: Glimpse of the data with the first five rows and seven categorical features along with two of the remaining seventy-three numeric features.

Additionally, on exploring the proportion of species according to the geographical location, we observe that around 80% of calls were recorded in the Mexican region and the breakdown of the remaining 20% of the observations is tabulated in Table 2.

Country	% of Observations
Mexico	80.58%
Canada	5.04%
FrenchGuiana	3.74%
CostaRica	3.59%
Panama	2.99%
Martinique	1.88%
Guadeloupe	0.90%
Brazil	0.77%
USA	0.30%
Unknown	0.23%

Table 2: Percentage of the total count of species broken down by country.

1.2 Missing and Unusual Values

In the dataset, we observe that there are 561 incomplete values in the data which capture our attention, with the feature Country being the most noticeable. From Table 2, we observe that for 0.23% subjects in the data, the geographical location of the recorded call is unknown which accounts for an observation count of 11. Exploring the possibility of the imputation of these unknown values using SAS studio, I began with looking for the similarity in feature *File* of these missing values however, all of the observations for a given File had a missing Country value. Further, I tried to look for the similarity in the feature Owner however the Owner to which these missing values belong to recorded the data in two countries, namely Mexico and Canada, which makes the imputation of these missing Countries even more complicated.

Additionally, there are a couple of numeric features, for instance, *SlopeAtFc*, also that have some missing values. However, these features are the acoustic components of the bat calls, and thus imputing them might induce noise in the data that can lead the model to have a spike in the number inaccurate predictions for unseen data. Apart from the missing values, we also observe an unusually high value of 93 for the feature *bndwdth* which lies away from the distribution of data. Moreover, the univariate analysis of *TimeFromMaxToFc* yields two unusually high values of 23.56 and 24.43 for the observations 3546 and 3554, respectively. Though these unusual values are potential outliers, it would still not be justified to mark them as outliers as they might be simply some valid unusual observations. Taking the susceptibility of the sound data to the unwanted noise into the account, we would refrain from imputing the numeric features that correspond to the call acoustics as there is a high possibility of introduction of noise in the data even when imputing with group means. Moreover, the random noise in the data would deteriorate the quality of the predictions made by the models that would in turn depreciate the pragmatic implementation of the trained model on the unseen data.

Chapter 2

BACKGROUND

A study titled 'Acoustic identification of Mexican bats based on taxonomic and ecological constraints on call design' conducted in 2016 by Zamora-Gutierrez et. al, deployed random forest models for the hierarchical classification of bats taking the acoustics as the input with an objective to widen the horizon of the potential of research in bat acoustic monitoring. Further, they fitted the rf models for the family-level and genus-level classification and recorded the resultant classification accuracies to advocate the claim that the hierarchy untangled the identification of bats at different levels. Although the authors did not record significantly high classification accuracies, they illuminated the potential of research which motivates me to deploy the Deep Neural Networks (DNNs) for this classification problem and compare the results to the previously implemented techniques. Before diving into the implementation, we first get accustomed to the theoretical background of the neural networks and understand their working mechanism.

2.1 Neural Networks: An Overview

The basic idea behind neural networks is to simulate the dense structured network of brain cells in a computer to learn the associations, recognize patterns, and make decisions in a human-like way. The feature that makes neural networks so fascinating is that they learn and gradually improve on their own through training and retraining over the data *(Féraud, R. and Clérot, F., 2002).*

Now the question is why neural networks emerged at first? The advancement of machine learning techniques leads to the emergence of a plethora of algorithms depending on the characteristics of data encountered. However, there were still some complex relationships in the data that these algorithms failed to capture. One classic example of such problems comes from the domain of classification and is the XOR problem as described by *(Zhao Yanling, 2002).* To understand this problem, let us first look at the inputs and outputs as tabulated in Table 3 and further suppose that our data is scattered on a 2-D plane resembling the XOR plot

(see Figure 3) which makes it linearly inseparable and hence the existing machine learning classifiers that prefers the data to be linearly separable prove futile in this case.

Input 1 (x1)	Input 2 (x2)	Output
0	0	0
0	1	1
1	1	0
1	0	1

Table 3: XOR inputs and output

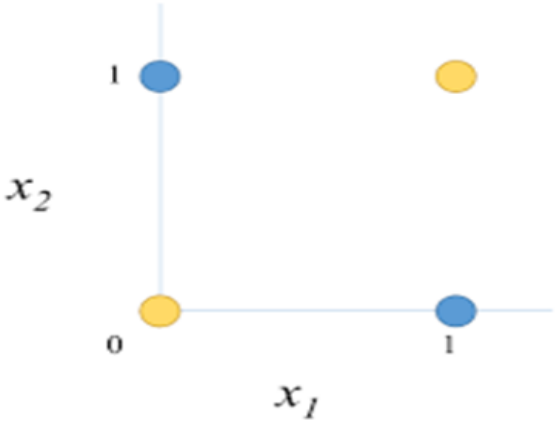


Figure 3: Visualizing XOR data

However, a multilayer perceptron or a neural network exhibits excellent performance in this non-linear classification problem is shown in Figure 4 and a visual of the decision boundary on the 2-D plane is shown in Figure 5.

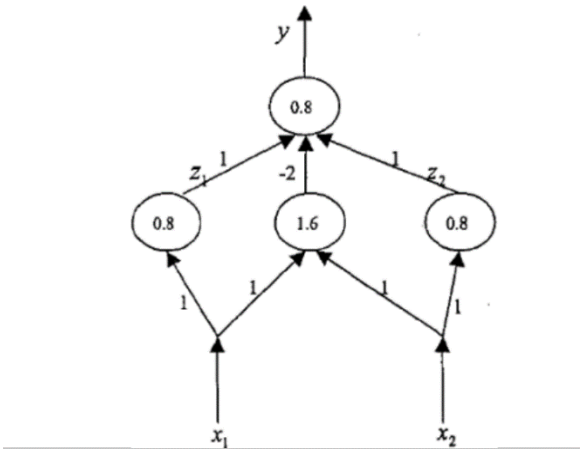


Figure 4: Neural network for the XOR problem *(Zhao Yanling, 2002)*. The numbers on the edges are the weights and the magnitudes inside the neurons are the threshold values that control the activation of the neurons. Z1 and Z2 are the outputs of the respective nodes and y is the resultant output.

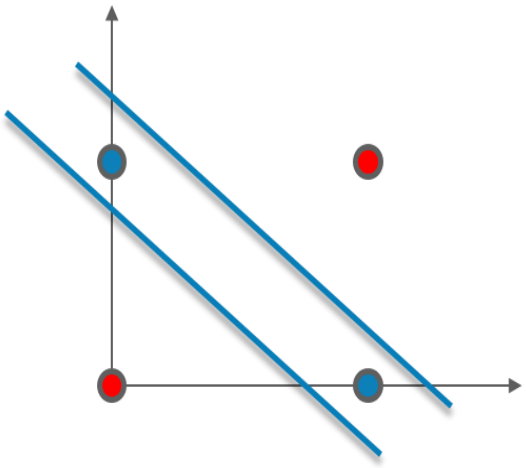


Figure 5: Visualizing decision boundary by neural network for XOR problem.

2.2 Perceptron Learning Rule

The building block of a neural network is a node, and a network with a single node is termed as a perceptron in deep learning jargon. A perceptron calculates the weighted sum of the input features and feeds it to the activation function which classifies the input into one of the output classes according to its flexibility of classification threshold, i.e. degree of *softness* (see figure 6). A connected network of multiple perceptrons spread over several layers is termed as a multilayer perceptron or a neural network.

Perceptron Model

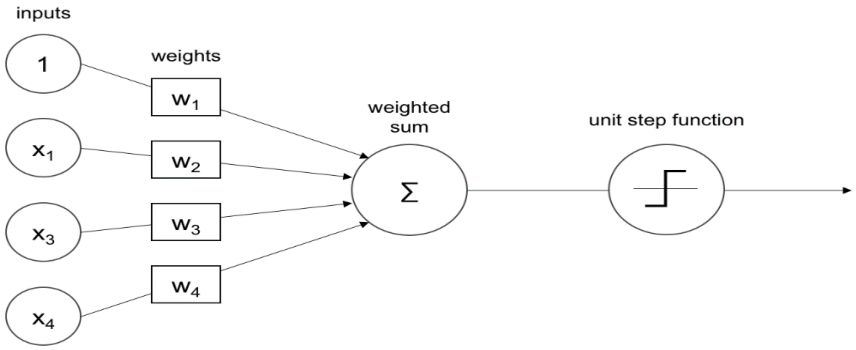


Figure 6: A diagrammatic description of the Perceptron Model where the weighted sum is the linear combination of the inputs and the weights which is further fed into an activation function (here, unit step function) to get the output class.

The working of neural networks revolves around recursively training over the labeled data thereby learning and updating the weights for the input signals for each node that collectively classify an unlabelled input into one of the output classes. This mechanism of recursively updating the weights to minimize the loss or the cost function using the labeled training data is termed as backpropagation.

2.2.1 Loss Function

Neural networks are trained using the Stochastic Gradient Descent (SGD) algorithm and require the selection of an appropriate loss function while specifying the model configurations. The objective behind the concept of the neural network is to gradually minimize the value of the loss function with each training epoch. It is pragmatically impossible to compute the ideal weights for a neural network as there are too many unknown parameters. Instead, the problem of learning is interpreted as an optimization problem and an algorithm is used to navigate the space of possible sets of weights the model may use in order to make good or good enough predictions.

Typically, a neural network model is trained using the SGD algorithm, and weights are updated using the backpropagation of error. A detailed working mechanism of the backpropagation is explained in section 2.3.

2.2.2 Activation function

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. Bias is a constant which empowers the neural network to fit the training data better. Mathematically, bias is the constant quantity that adds to the weighted sum of inputs and this quantity is fed to the activation function to produce the output of a node in the network. An activation function introduces non-linearity into the output of a neuron.

Neural networks update the weights and biases of the neurons on the basis of the error at the output through backpropagation. Activation functions make the backpropagation possible since the gradients are supplied along with the error to update the weights and biases.

A neural network without an activation function is interpreted as a linear regression model. The activation function does the non-linear transformation of the input making it capable to learn and perform more intricate tasks. Some Variants of activation function are listed below.

1. Linear Function

Irrespective of the number of layers in the network, if all the layers have a linear activation function, the activation function of the last layer could be formulated as a linear function of the first layer. The output of this function ranges from -inf to +inf. The linear activation function is always used in the output layer only. However, a linear function could be differentiated to induce non-linearity making the function constant with the result being independent of the input.

2. Sigmoid

   The Sigmoid function is a non-linear function that is represented by an S-shaped plot where the curve is quite steep indicating that a small change in the input x could yield major changes in the output y. This function is mostly used in the output layer when the response is binary and is compatible with levels 0 and 1 denoting the output classes as its output itself ranges between 0 and 1. Thus, a threshold, say 0.5, decides the output class of the network.

3. Tanh

   A mathematically shifted version of the sigmoid function is the Tangent Hyperbolic (tanh) function with its output ranging from -1 to 1. The tanh function is mostly used in the hidden layers of the neural network as the mean output of the layer is 0 (as the range is -1 to 1) which in turn centers the data to have the mean close to 0. This, further, eases the learning for the next layer significantly.

4. ReLU

   The most commonly used activation function is the Rectified Linear Unit, abbreviated as ReLU. ReLU is a non-linear function that makes it suitable for backpropagating the errors and update the weights and bias of the neural network. Its implementation is majorly found in the hidden layers of the network where it activates only a few neurons at a given instance of time, making the network sparse and hence reducing the computational complexity. This makes it computationally more efficient than sigmoid and tanh. It is a sensible option to use ReLU when you are clueless about the choice of the activation function.

5. Softmax

The softmax function is a variant of the sigmoid function and is generally more effective in multi-class classification problems. The softmax function shrinks the outputs for each class between 0 and 1 and further divides the squeezed values by the sum of the outputs. This function is ideally used in the output layer of the network when the objective is to compute the probability of each output class for a given input *(Specht, D.F., 1990).* The probability for each of the K output classes is computed using equation 1.

$$P(Y_i = 1) = (e^{b1.Xi})/ \sum_k e^{bk.Xi}$$

$$P(Y_i = 2) = (e^{b2.Xi})/ \sum_k e^{bk.Xi} \quad \ldots..$$

$$P(Y_i = K) = (e^{bK.Xi})/ \sum_k e^{bk.Xi} \qquad (1)$$

2.3 Backpropagation in Neural Networks

Back-propagation is the foundation of neural network training. It is the mechanism through which the neural networks tune their weights and bias based on the value of loss (or error) from the previous training iteration (Illustrated in Figure 7). The weights are tuned to minimize the loss improving the generalizability thereby making the model more reliable.

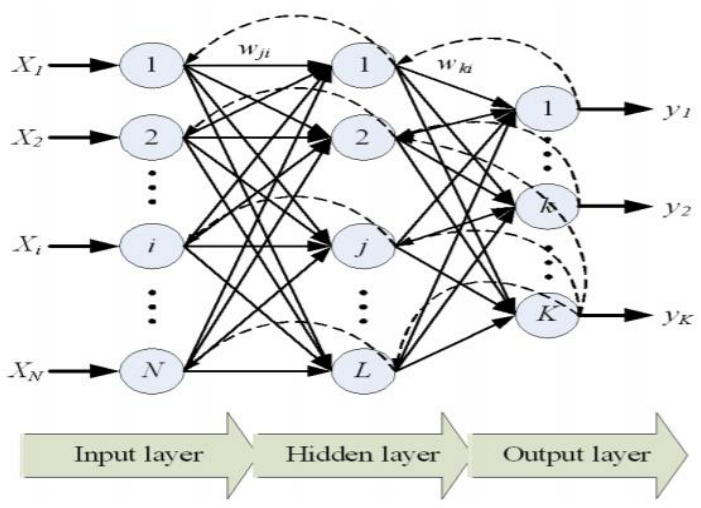


Figure 7: Backpropagation ANN (Che, Z.G., Chiang, 2011). The dotted edges show the path of the backpropagated gradient that is further used to update the weights of the previous layers.

The Backpropagation algorithm uses the delta rule or the gradient descent *(Hecht-Nielsen, R., 1992)* which is a first-order iterative optimization algorithm that searches for the minima of a function, to look for the minimum value of the loss function in weight space (see Figure 8). The vector of weights that minimizes the loss function is then considered to be a solution to the learning problem.

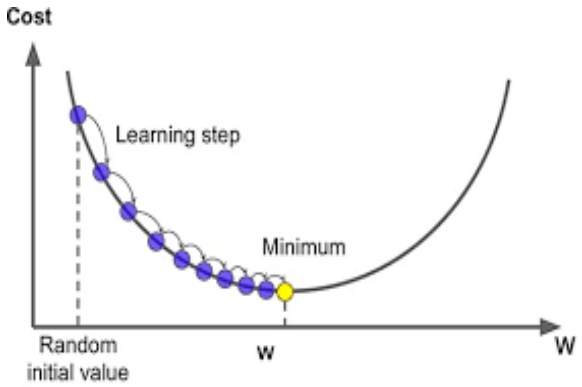


Figure 8: Gradient Descent algorithm for finding the minima of the loss (or cost) function.

2.4 Optimizers

After passing every batch of input through the network and fetching the corresponding output, the neural network must have a plan to use the difference between the predicted and the actual results in order to alter the weights on each node such that the network progresses further towards the optimality. The algorithm that determines this progress is known as the optimization algorithm**.** There is a wide range of optimization algorithms some of which are described below.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD), is the classical optimization algorithm that computes the gradient of the network loss function concerning every individual weight in the network. Each forward pass through the network results in a certain parameterized loss function, and we use each of the gradients that were created for their corresponding weights, multiplied by a certain learning rate, to progress the weights in whatever direction their gradient is pointing. The new

weights generated after each epoch will always be strictly better than the ones from the previous epoch. A variant of SGD uses Nesterov momentum that improves optimization algorithm convergence speed. Nesterov momentum is that the technique that uses momentum. Momentum techniques are the ones that determine the current step taking the information from the previous steps into account.

Adagrad

A more advanced machine learning technique relative to SGD is Adagrad which performs gradient descent with a variable instead of a constant learning rate. The weights which earlier had large gradients are assigned large gradients, whereas the weights which had small gradients earlier are assigned small gradients. Thus, Adagrad is simply the SGD with an in-built learning rate updating mechanism for each node *(Ward, R., Wu, X. and Bottou, L., 2019)*. It is rather an improved version of SGD that assigns weights using the previously correct learning rates for every node instead of using a single pre-decided learning rate for all the nodes.

RMSprop

The RMSprop optimizer restricts the oscillations of the gradient in the vertical direction. This gives the flexibility to increase the learning rate and still converge faster to the minima taking larger steps in the horizontal direction instead of vertical, thus avoiding overshooting the minima. Similar to the other adaptive learning rate optimization algorithms, it is recommended to leave the hyperparameter of the RMSprop at their default settings.

Adam

Adam stands for Adaptive Moment Estimation. Adam stores exponentially decaying mean of past gradients (like momentum techniques) in addition to the exponentially decaying mean of past squared gradients. Adam is presently one of the most popular optimization algorithms, largely because it provides both the smart learning rate reduction and the momentum traits of the algorithms we have seen so far. The mathematical notation describing the weight update in the Adam optimizations algorithm is represented by equation 2. Here, W_{t+1} and W_t represent the current and the previous weight-vector. l is the weight decay hyperparameter and the

product of the learning rate (h) and the gradient penalizes the large weights *(Kingma, D.P. and Ba, J., 2014*).

$$W_{t+1} = (1 - l)W_t - h\nabla f_t(W_t) \quad (2)$$

AMSgrad

AMSgrad is a recently proposed improvement to Adam. The limitation of Adam is that it fails to converge to the globally optimal solution for certain datasets, whereas simpler algorithms like SGD perform well. A recently published paper proposed that the exponential weights on the terms in the algorithm pose the problem.

The hypothesis is that for certain datasets such as image data, there are a lot of small and less informative gradients punctuated by occasional large and more informative gradients. The inbuilt proclivity of Adam to give less priority to the more informative gradients because those gradients are quickly swallowed by the exponential weighting, causing the algorithm to overshoot the point of optimality without sufficiently exploring it.

2.4.1 Learning Rate

The learning rate, also termed as the step size, defines the magnitude of gradual decrement in gradient and hence regulates the convergence of the loss function. For a sequential model, the learning rate usually ranges from 0.0001 to 0.001. It is quite important to choose an optimal learning rate that is neither too large nor too small. A high learning rate would initially converge the gradient to the optimal loss function but would later oscillate around the optimal as the step size would be too big (visualized in Figure 9). However, if the learning rate is too low, the convergence would be quite slow and might require a large number of training epochs to converge to the optimal. Hence, an adaptive optimizer such as ADAM would be the ideal choice which starts with a larger value of the learning rate and gradually decreases its magnitude as it nears the optimal value for the loss function.

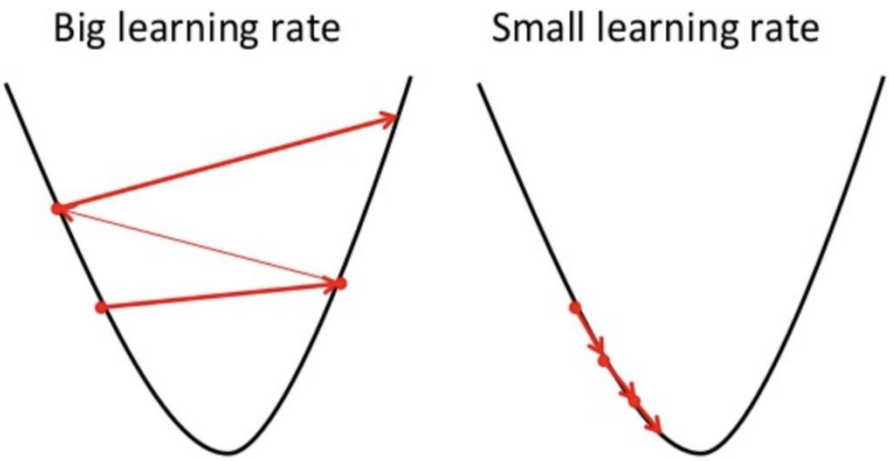


Figure 9: Visual comparison of learning rates

2.5 Regularization

Regularization is a technique that modifies the learning algorithm in such a way that the model refrains from overfitting the training data and hence allows the model to generalize better. This further improves the performance of the model on the unseen test data as well. Analogous to regularization in machine learning where the coefficients are penalized, the regularization techniques in deep learning penalizes the weight matrices of the nodes which reduces the overfitting thereby enabling the model to generalize better. Some of the commonly used regularization techniques are L1 and L2 regularization, Dropout, Data Augmentation, and Early Stopping.

2.5.1 L2 and L1 Regularization

The L1 and L2 are the most conventional regularization techniques used in the neural networks. In these methods, the cost (or the loss) function is updated by adding a penalty term.

Cost function = Loss (say, sparse binary cross entropy) + Penalty term

The addition of this penalty term shrinks the magnitudes of weight matrices in accordance with the assumption that a neural network with smaller weight matrices generates simpler models that in turn generalizes better, reducing the overfitting on the training data to quite an extent.

However, the factor that distinguishes the L1 and L2 techniques is the penalty term.

In L2, the squared value of weights is penalized (as seen in Equation 3):

$$Cost\ function = Loss + \lambda/(2m) * \sum ||w||^2 \quad (3)$$

Here, the hyperparameter lambda (λ) is the regularization parameter whose value is optimized to improve the results of the model. Since the L2 regularization forces the weights to shrink towards zero, it is also referred to as 'weight decay'.

Whereas in L1 Regularization, the absolute value of weights is penalized (see Equation 4):

$$Cost\ function = Loss + \lambda/(2m) * \sum ||w|| \quad (4)$$

In L1 regularization, the weights may be shrunk to zero. Hence, it is sometimes viewed as a method that implicitly performs feature reduction thereby compressing the model.

Using the *keras* library, regularization could be applied to any layer of the neural network directly using the regularizers module of the library.

2.5.2 Dropout

Dropout is one of the most interesting regularization techniques that produce quite impressive results and is consequently the most repeatedly used technique in the deep learning domain. The technique works in a pretty straight-forward fashion where a specific proportion of nodes is selected randomly and removed from the network along with all the incoming and outgoing connections. This pruning of nodes and the corresponding connections is best illustrated in Figure 10.

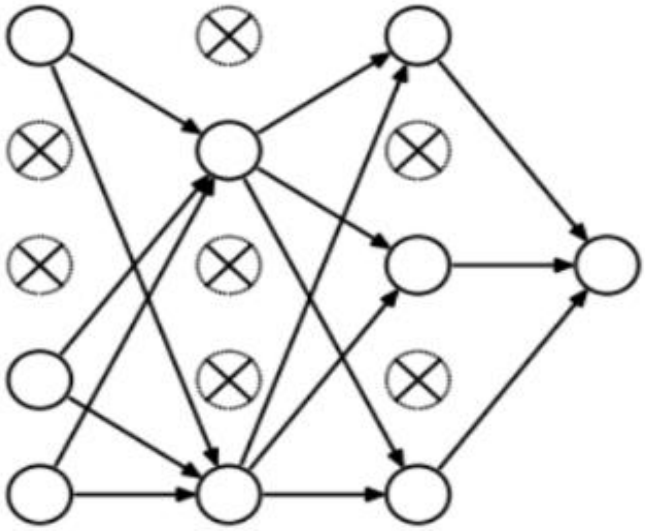


Figure 10: A rough sketch of a neural network with a dropout regularization.

Consequently, each epoch has a different set of nodes and connections to pass the input through and this results in the model to learn on the training data with a different set of nodes generating different sets of outputs, preventing the network to overfit the training data. It can also be fairly compared to the ensemble technique in machine learning. Ensemble models capture more randomness and hence give a better performance as compared to a single classifier. Similarly, a deep learning model with dropout also enhances the performance of a regular neural network model. The probability of selecting the number of nodes that should be dropped in an epoch is given as the hyperparameter to the dropout function. Moreover, Dropout founds its application in both, the hidden layers as well as the input layers.

2.5.3 Data Augmentation

Introducing generalization and thus mitigating the overfitting of a neural network could simply be accomplished by increasing the size of the training data. In machine learning, it is mostly not possible to increase the size of training data as the labeled data is too costly. However, when dealing with image data, the training data could be easily expanded by performing elementary operations such as rotation, flipping, scaling, etc on the image and adding the resultant image to the data. This technique is referred to as Data Augmentation. This usually leads to significant improvement in the validation accuracy of the model.

2.5.4 Early Stopping

Early stopping is a method that implements the cross-validation technique, reserving a part of the training data as the validation set. The underlying concept is to immediately halt the training of the model when decay in the performance of the model on the validation data is observed. This is known as early stopping. Illustrating this concept in Figure 11, the network will stop training at the dotted line as the model clearly starts to overfits the training data.

Figure 11: Illustration of the Early Stopping regularization technique

In *keras*, we find the implementation of the early stopping in the *callback* function. The function takes two parameters – Monitor and Patience. Monitor is used to specify the performance metric that must be monitored, for instance, a value 'val_err' assigned to the Monitor parameter specifies that validation error should be monitored for early stopping of the training of the network. On the other hand, Patience represents the number of epochs after which the training must be stopped when no further improvement is observed.

2.6 Types of neural networks

1. Feedforward Neural Network

One of the simplest types of Artificial Neural Networks, a feedforward neural network propagates the input through some or all of the nodes until it reaches the output layer. In other words, data flows in a unidirectional fashion from the first layer until it reaches the output node. Unlike in more complex types of neural networks, there is no backpropagation and data move unidirectionally only. A feedforward neural network might or might not have hidden layers and deals quite well with the noisy data.

2. Recurrent Neural Network (RNN) – Long Short Term Memory (LSTM)

A Recurrent Neural Network is a variant of artificial neural network which saves the output of a particular layer in the network and feeds it back to the input layer recursively. The first layer is similar to that in the feedforward network with the linear combination of the features and their corresponding weights as an input. The recurrent process is carried out in the subsequent layers *(Aggarwal C.C. (2018))*.

In an RNN, each node works as a memory unit while computing and carrying out operations. The training of the network begins with the forward propagation as usual but learns and retains the information on the output that it may need to use later. If the prediction is not correct, the network learns itself and works towards rectifying the prediction during the backpropagation step. Long Short Term Memory (LSTMs) are very effective in the NLP branch of deep learning and finds wide application in text-to-speech conversion.

3. Convolutional Neural Network (CNN)

A convolutional neural network (CNN) uses a variant of the multilayer perceptron (MLP) and contains one or more convolutional layers that can either be fully interconnected or pooled otherwise. A convolutional layer carries out a convolutional operation on the input before propagating the result to the next layer. Because of this convolutional operation, the network can be much deeper but with much fewer parameters and this enables them to show impressive results in image and video recognition, recommender systems, and Natural Language Processing. Convolutional neural networks also perform well in semantic parsing and paraphrase detection which further finds its application in plagiarism detection. CNNs are widely applied in image classification and signal processing.

The above network types along with some others are portrayed in Figure 12.

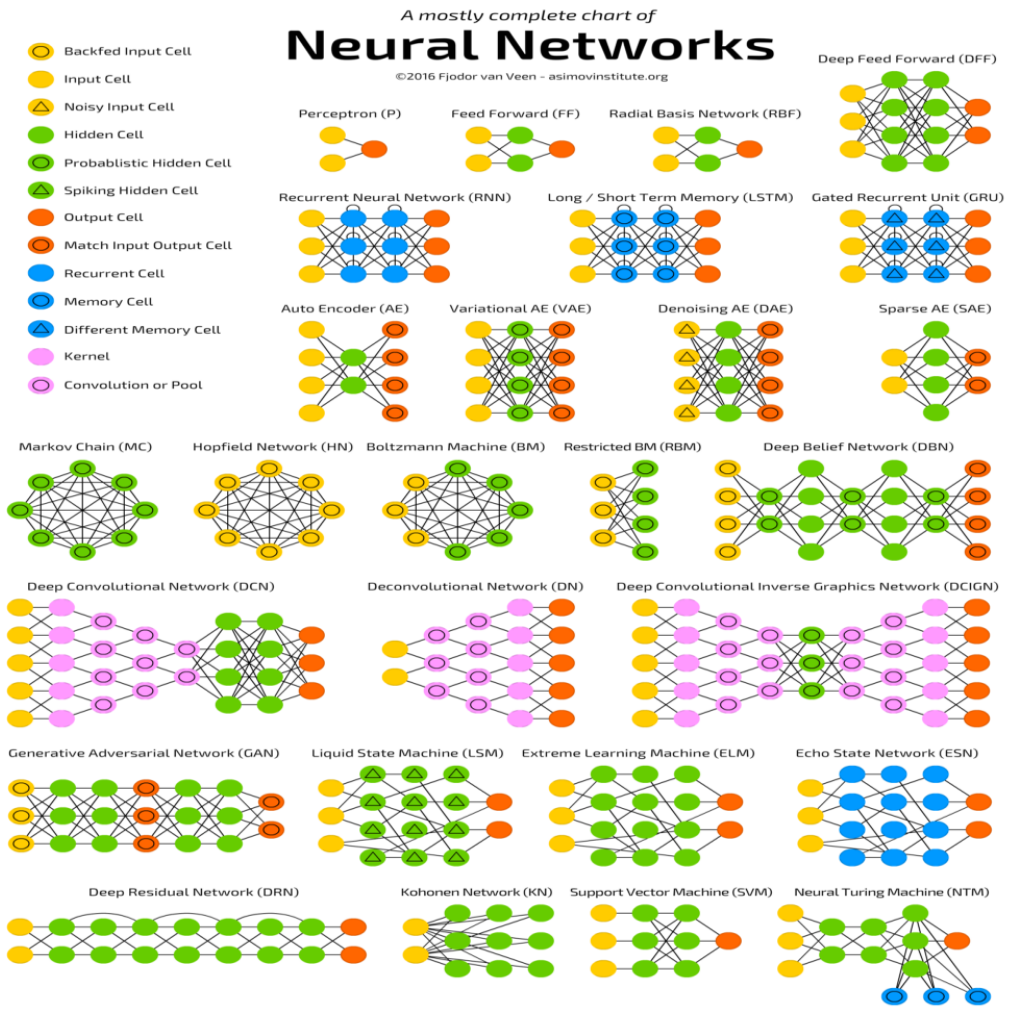


Figure 12: Types of Neural Networks *(Fjodor Van Veen, 2016).*

Chapter 3

IMPLEMENTATION AND INFERENCES

3.1 Data Cleaning and Pre-Processing

An Artificial Neural Network put a type constraint over the features and require all the features to be numeric. However, categorical features can also be incorporated for training the network by encoding them into the integers. For this encoding, the *pandas* library offers a couple of encoding techniques such as one-hot encoding, ordinal encoding and embedding. Thus, for our data, we encode all the categorical columns - family, genus, species, guild, and country. We also observe some missing values in the data which accounts for 12% of the data approximately. Hence, we obtain our initial results by omitting these observations and look for possible imputations later in the analysis.

Neural networks are highly sensitive to the varying scale of the features and generally do not performs well on unscaled data. Consequently, we might observe poor training accuracy and some issue in the convergence of the loss function to the optimal minimum. Exactly for this reason, the *sklearn* library contains the implementation of a couple of scaling techniques, for instance, Standard scaling and Min-Max Scaling, reducing the efforts of defining a function to implement them. Thus, we use the *StandardScaler* from the preprocessing module of the *sklearn* library to scale our data to have a mean 0 and a unit standard deviation. As expected, we observe a massive improvement in the convergence which is reflected by the improvement in the training accuracy in the models discussed below.

Finally, we split the data into training and validation sets to train the model on labeled observations and hence evaluate the model on unseen data. Here, in this study, we split the data into a ratio of 70:30 with 70% of data partitioned into a training set and the remaining 30% into the validation set. For this, we use the train_test_split() function defined in the preprocessing module of the sklearn library.

3.2 Classifying the Species

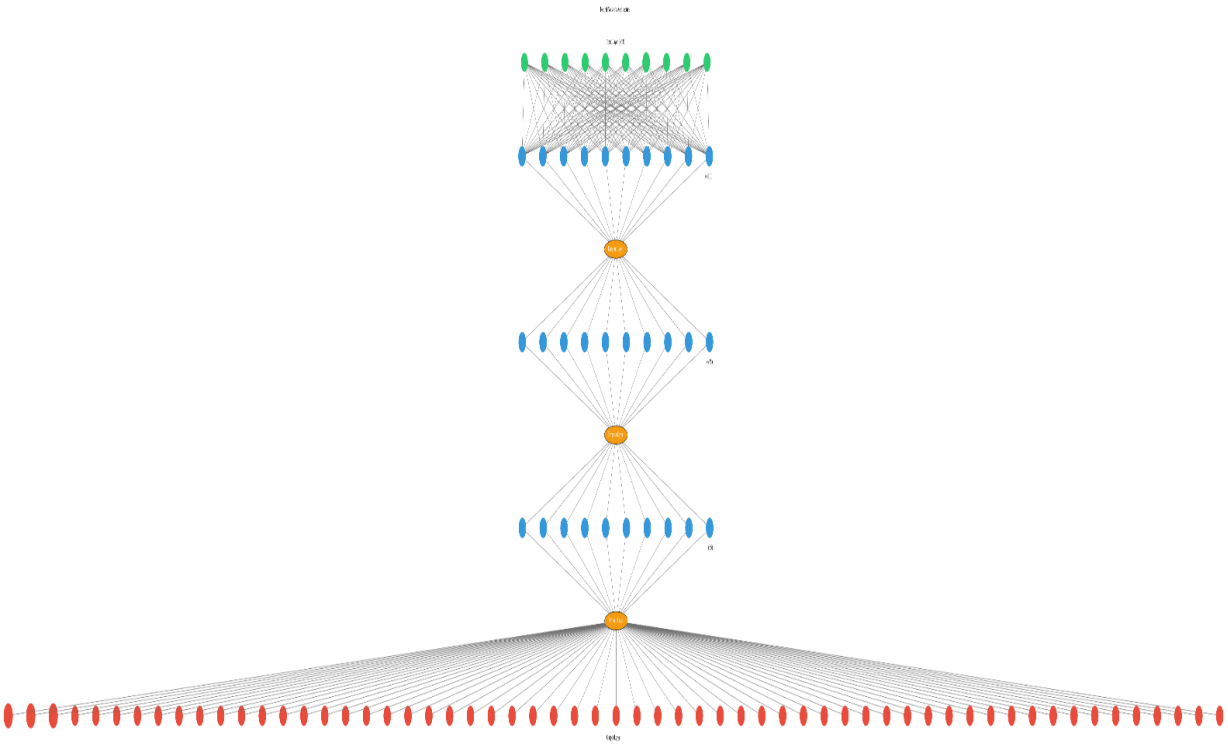
The idea here is to train a network on the designated training data keeping the species as a response having 59 classes. Initially, all the numeric features that represent the acoustics of the bat calls were fed to the model, leaving out all the categorical features, namely, guild, country, and file. In the pre-processing stage, all the categorical features were integer-encoded to conform with the compatibility criteria of neural networks. Implementing an Artificial Neural Network (ANN) using a sequential model in *keras (Ketkar N. (2017)* with a *sparse_categorical_cross_entropy* as the loss function and an adam optimizer, I started with a model with a single hidden layer having five nodes at first which gave an accuracy of nearly 62% on validation data which improved gradually with addition of additional hidden layers and altering the number of neurons in each layer. After trying numerous network architectures, the validation accuracy on the final structure was recorded as 81.79% which improved further to somewhere around 83% on tweaking the hyperparameters. The final architecture is explained in section 3.2.1.

3.2.1 Network Architecture and the Hyperparameters

Deciding a suitable architecture for Deep Learning models is a trial and error based process where we start by training the simplest possible network structure - a perceptron model – on our training data and gradually increase the complexity of the architecture by introducing more layers and altering the count of neurons within each layer, thereby recording the training and validation performances of the network. However, it is important to be cautious while making the network structure more intricate as an overly complex structure might steer the boat in the opposite direction and the network, on the contrary, might give a worse overall performance instead of showing any improvement.

Trying around 20 different structures and recording their performances on training and validation data, I narrowed down to a nearly optimal, if not a theoretically optimal, network architecture with three hidden layers (see Figure 13(a) and 13(b)). Subsequently, an issue encountered after deciding the network structure was that though the network had a high classification accuracy on the training data, it gave a rather unsatisfactory performance on the

validation set which indicates that the model is overfitting the training data. The reason behind this overfitting behavior was that the class predictions made by the network possibly rely entirely on a few nodes and connections, thereby following the same path to reach to the output layer, rendering the remaining neurons and connections ideal for most of the time. This introduced the need of introducing a regularizer in the network which keeps a check on this. Hence, I added 3 Dropout layers in the network after each of the three hidden layers specifying the proportion of nodes that should freeze at random after each training epoch.



(a). Illustration of the final network architecture. Here, the layers with neurons in green and red are the input and output layers respectively. The three layers in blue are the hidden layers with the orange circles indicating the dropout layers.

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 120)               9120
_________________________________________________________________
dropout_31 (Dropout)         (None, 120)               0
_________________________________________________________________
dense_42 (Dense)             (None, 85)                10285
_________________________________________________________________
dropout_32 (Dropout)         (None, 85)                0
_________________________________________________________________
dense_43 (Dense)             (None, 65)                5590
_________________________________________________________________
dropout_33 (Dropout)         (None, 65)                0
_________________________________________________________________
dense_44 (Dense)             (None, 59)                3894
=================================================================
Total params: 28,889
Trainable params: 28,889
Non-trainable params: 0
_________________________________________________________________
```

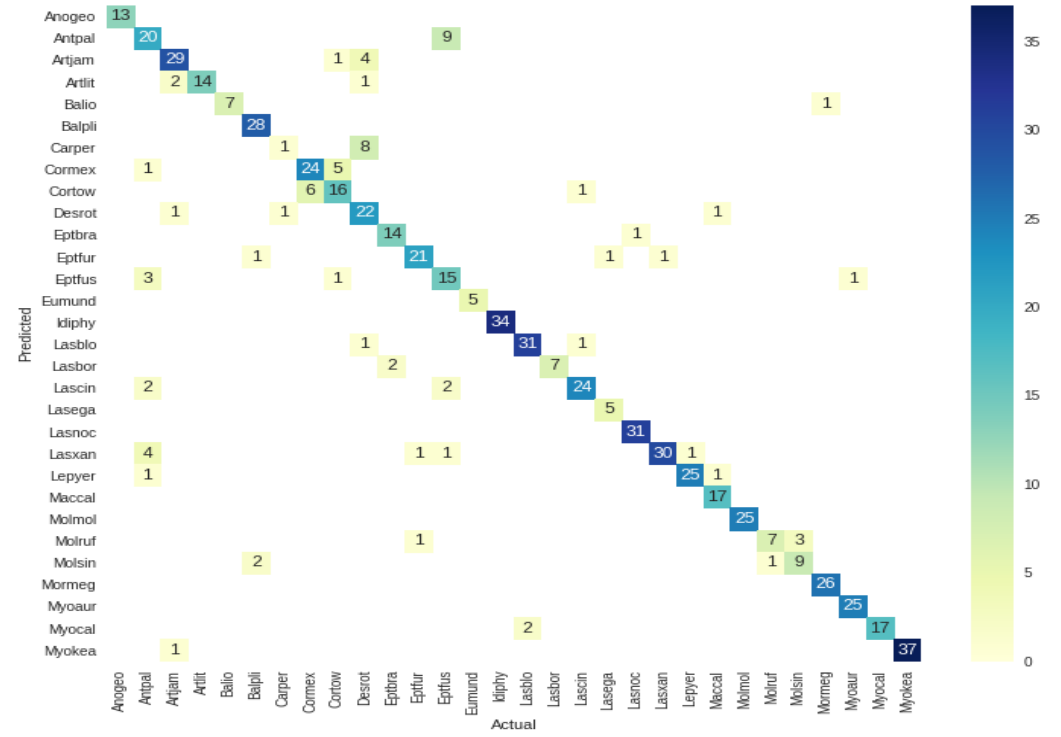
(b). Final network architecture with a detailed description of each layer.

Figure 13: Summarizing the network structure

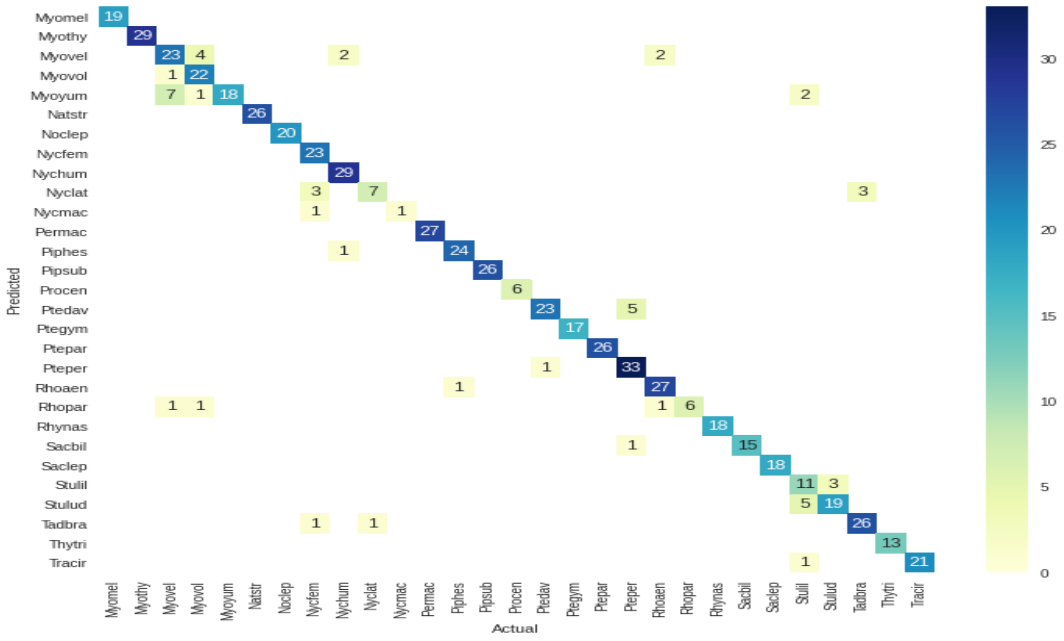
The performance of this model when only call acoustics (all the numeric features) is known is indicated by a training and validation accuracy of 89.6% and 80.2% with a noticeable issue in the convergence of the validation loss. To counter this, we add the feature Country to the list of features and retrain the model. The motivation behind this is that the knowledge of the country catalyzes the identification of the species. As expected, we observe that both the training and the validation losses are nearly equal throughout the training, which indicates an improvement in the convergence of the loss function, and a consequent improvement in the validation accuracy is observed, leaping up to 84.6%.

A deeper insight into the performance of this model on each specie is highlighted by the multi-class confusion matrix in Figure 14 where all the non-diagonal elements for a given row or column sums up to give the classification error for that particular specie. Overall, the models perform well in classifying the individual species with a few instances of high misclassifications.

Confusion Matrix for Species Model



(a)



(b)

Figure 14: Confusion matrix divided into two parts. All the non-diagonal elements of the matrix indicate the number of misclassifications for a given specie.

3.2.2 Decrypting the Neural Network to Assess Feature Importance

Neural Network is a black box algorithm where extracting the information on feature importance is not straight-forward. However, an ANN model with backpropagation updates its weights that decides the importance of an input connected to a given node in a particular layer. Thus, the methodology here is that the recursive updating of the weights propagated back to the first layer of the neural network could be used to rank the features in order of their importance. Exploring this technique further, I extracted the standardized weights from the first layer of the network and traced back to further extract the weights of each input feature. Mapping this data down to a bar plot in Figure 15, the identification of significant features of the Species model becomes quite interpretable.

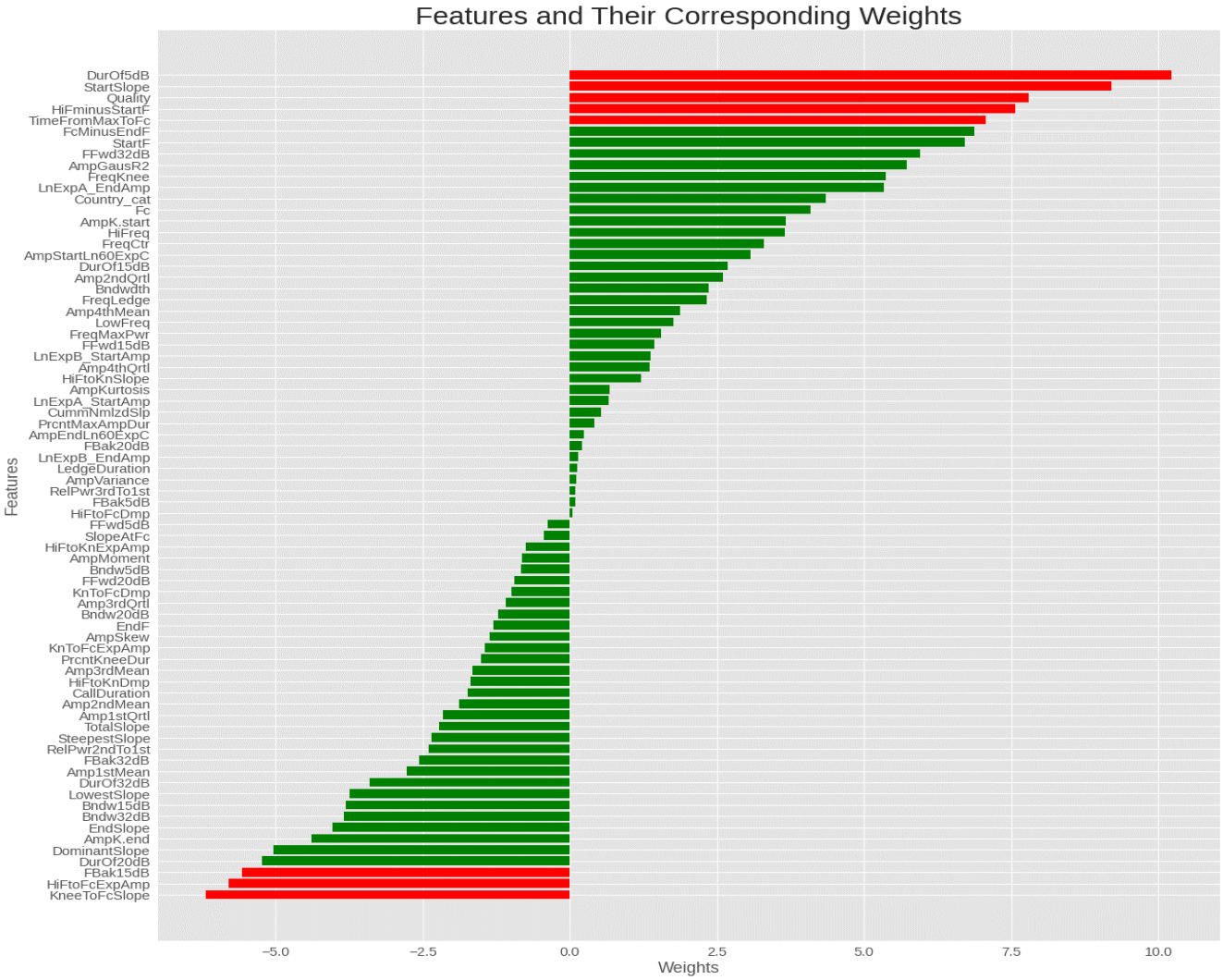


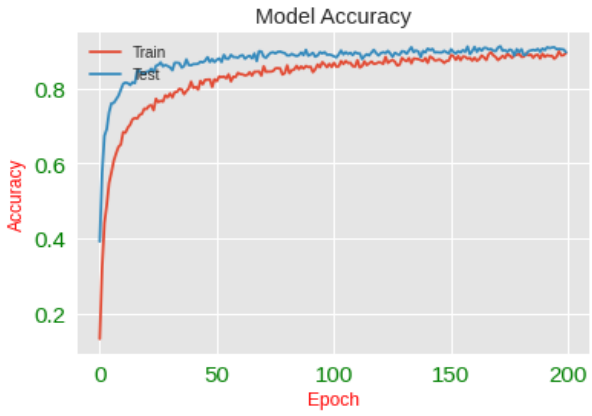
Figure 15: Feature ranking using weights where the eight most important features are colored red.

3.2.3 Hierarchical Modelling

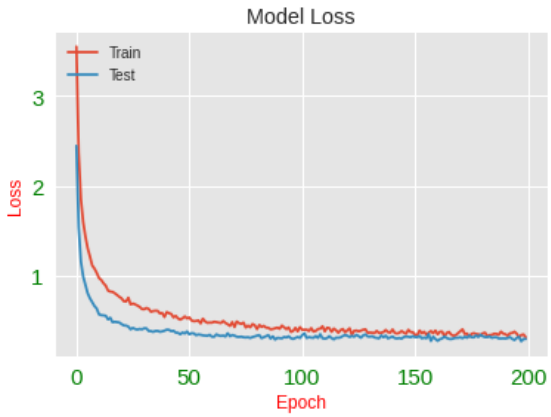
The next phase of my work revolves around the biological hierarchy of the bats which explores the improvement in the predictive power of the neural network that predicts the species when the family is also known along with the existing features. This work is further extended to the hypothetical situation when we also know the genus of the bat and hence add it to the list of features.

Species within a family

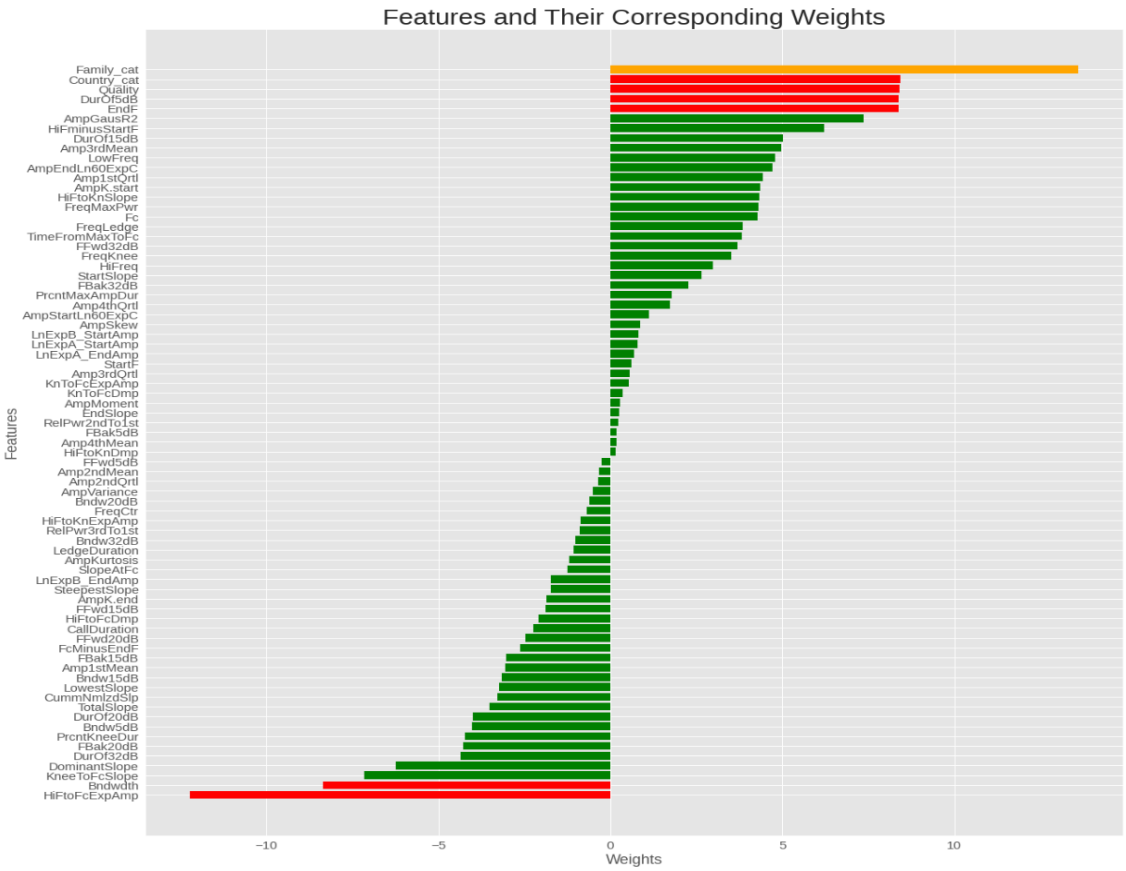
Adding family to the list of features, we observe an improvement in the accuracy of the model as expected. This model simulates the situation when apart from the acoustic characteristics of the bat call, the biologist also knows the family of the subject. Here, we observe a significant improvement in the validation accuracy (see Figure 16(a)) and conclude that Family is an important predictor, which is quite intuitive. From the Features vs Weights plot in Figure 16(c), we observe that the Family is the most important feature in this the prediction of the Species of the subject.



(a) Training and Validation accuracy



(b) Training and Validation loss



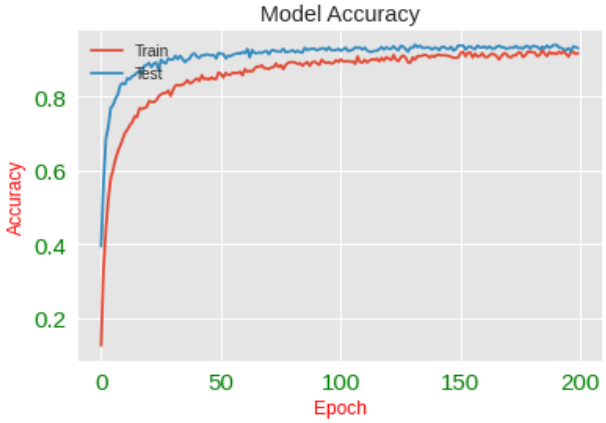
(c) Eight most important features are highlighted in red with Family as the most important predictor highlighted in orange.

Figure 16: Performance of hierarchical model when Family is known

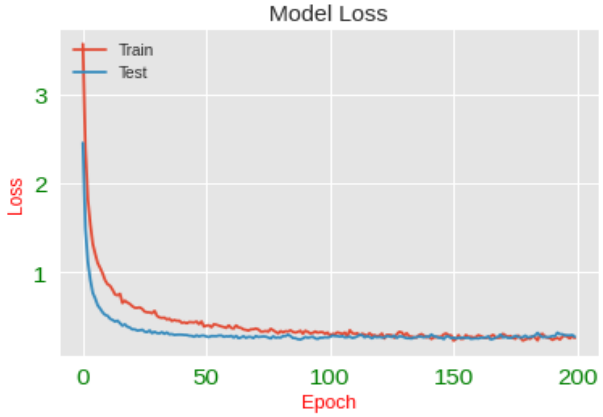
The above Accuracy and Loss plots (Figures 16(b)) are evident of the improvement in the convergence of the gradient towards optimal. The validation accuracy recorded after the final training epoch is 85.9% as compared to a 72% accuracy achieved by the random forest model fitted by *(Zamora-Gutierrez, 2016).*

Species within a Genus

Similar to the Family, with Genus added to the feature list, we observe an improved convergence of the loss function and a corresponding increase in validation accuracy. This infers that the knowledge of the Genus of a subject along with the call acoustics improves the predictive power of the neural network that trains on only acoustic characteristics.

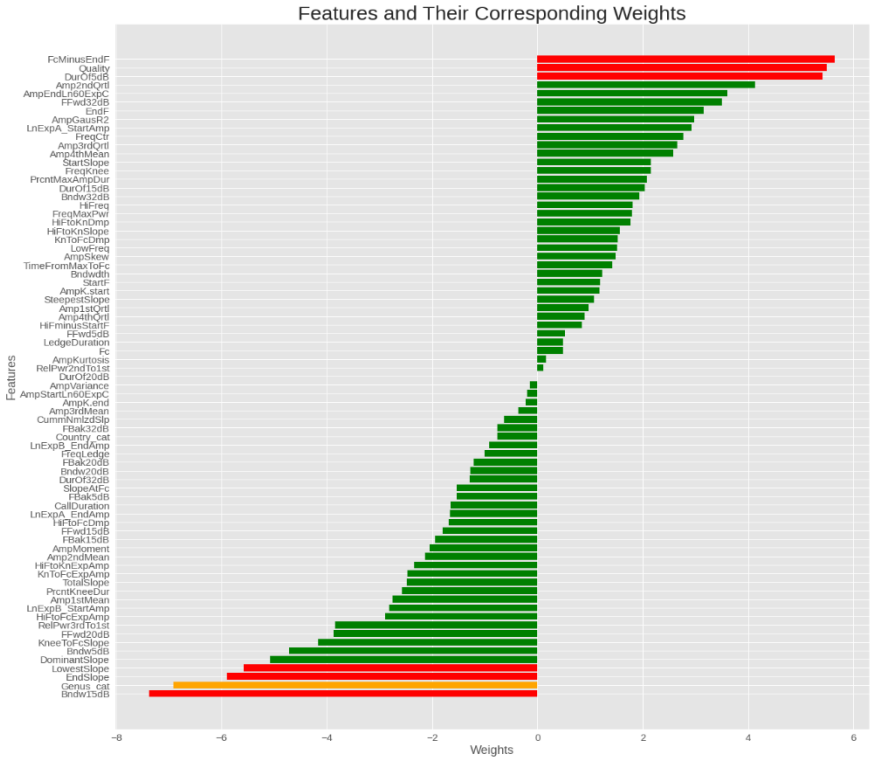


(a) Training and Validation Accuracy



(b) Training and Validation Loss

Screenshot of the Final Training Epochs



(c) Eight most important features are highlighted in red with Genus as the second most weighted feature highlighted in orange.

Figure 17: Performance of hierarchical model when Genus is known

Here, we get an even better accuracy of 90.23% as compared to a 71.2% accuracy achieved by the model in the previous study *(Zamora-Gutierrez, 2016).* The fact model does not overfit the data is evident from the nearly equal values of training and validation loss (and correspondingly, accuracies) as seen in figure 17(a) and figure 17(b).

3.3 Classifying the Genus

In this section, we train a model with Genus as the target variable with acoustic characteristics and country as features. The underlying idea again deals with a practical situation that biologists usually encounter while identifying the bats from their call acoustics. The variable of interest is Genus, which lies at the second level of the biological hierarchy of bats. There are 32 distinct Genus in this data which means we have 32 output classes.

Genus	**% of Total Count of Genus**
Myotis	16.50%
Lasiurus	7.73%
Pteronotus	7.41%
Eptesicus	5.64%
Saccopteryx	4.27%
Pipistrellus	4.27%
Corynorhinus	4.27%
Nyctinomops	3.82%
Molossus	3.42%
Artibeus	3.31%
Sturnira	3.12%
Rhogeessa	2.71%
Balantiopteryx	2.60%
Tadarida	2.13%
Rhynchonycteris	2.13%
Peropteryx	2.13%
Nycticeius	2.13%

Mormoops	2.13%
Leptonycteris	2.13%
Lasionycteris	2.13%
Idionycteris	2.13%
Antrozous	2.13%
Natalus	2.07%
Desmodus	2.01%
Noctilio	1.73%
Trachops	1.58%
Macrotus	1.47%
Anoura	1.02%
Thyroptera	0.66%
Carollia	0.47%
Promops	0.38%
Eumops	0.34%

Table 4: Proportion of observations in each Genus.

From table 4, we see that the data is a bit unbalanced with 16.5% observations from *Myotis* genus and *Eumops* genus having 0.34% of the data. Despite this unbalance, the neural network performs quite well in predicting genus from call acoustics as the ANN apparently deals quite well with unbalance in the data which is evident from table 5 ahead. The classification accuracy recorded on the validation data in the final training epoch is nearly 88% as compared to a 77.8% validation accuracy achieved by the random forest model by *(Zamora-Gutierrez, 2016).* The feature ranking according to updated weights after the training of the model is portrayed by the plot in figure 18.

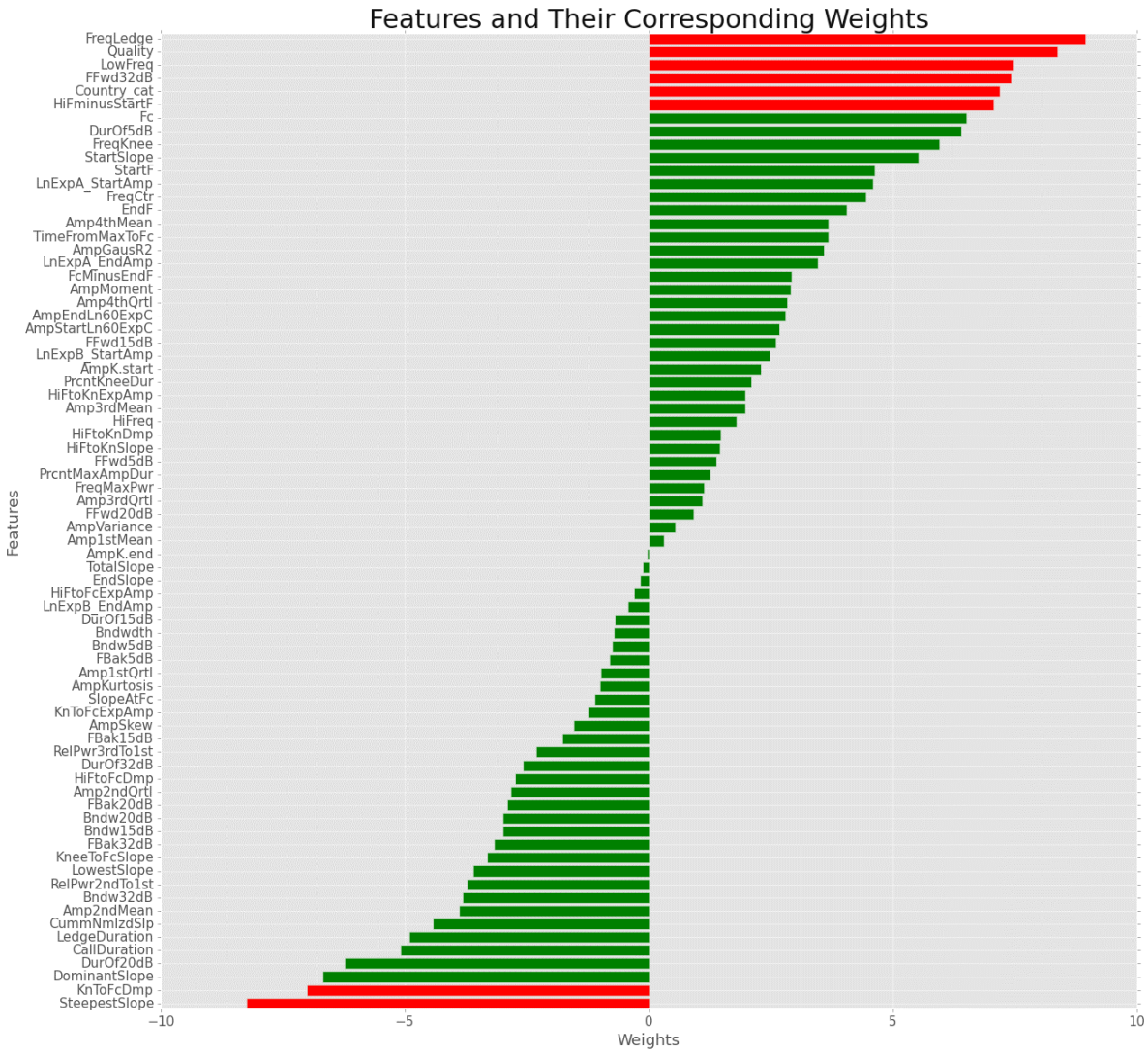


Figure 18: Feature ranking using weights where the eight most important features are colored red.

The model performance on each Genus is evaluated by quantifying the classification results through a confusion matrix in Figure 19. These results are further condensed into a classification report as tabulated in Table 5 which displays the precision, recall, and f1 score on each of the Genus.

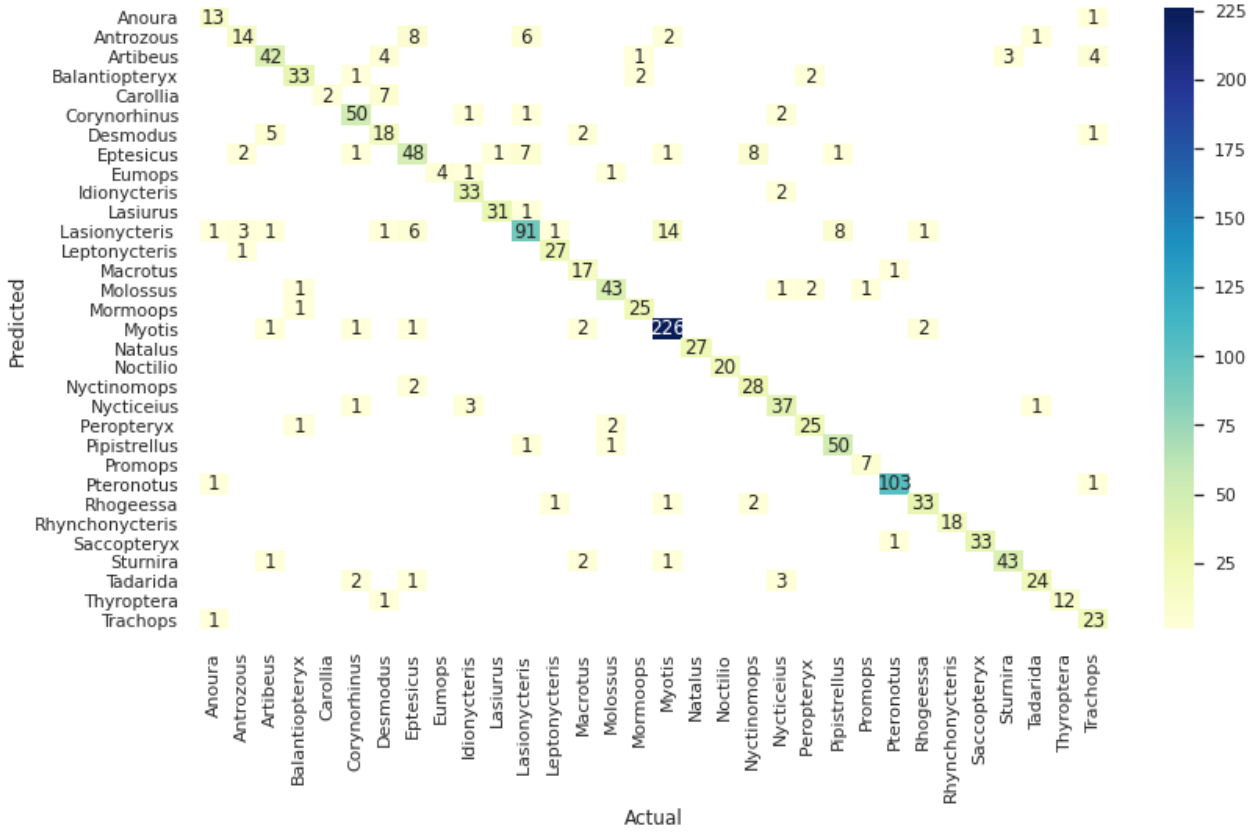


Figure 19: Confusion matrix for the model on Genus classification. The non-diagonal elements indicate the number of misclassified instances of a genus in the validation set.

Genus	Precision	Recall	f1-score
Anoura	0.81	0.93	0.87
Antrozous	0.7	0.45	0.55
Artibeus	0.84	0.78	0.81
Balantiopteryx	0.92	0.87	0.89
Carollia	1	0.22	0.36
Corynorhinus	0.89	0.93	0.91
Desmodus	0.58	0.69	0.63
Eptesicus	0.73	0.70	0.71
Eumops	1	0.67	0.8
Idionycteris	0.87	0.94	0.90
Lasiurus	0.97	0.97	0.97
Lasionycteris	0.85	0.72	0.78

Leptonycteris	0.93	0.96	0.95
Macrotus	0.74	0.94	0.83
Molossus	0.92	0.90	0.91
Mormoops	0.89	0.96	0.93
Myotis	0.92	0.97	0.95
Natalus	1	1	1
Noctilio	1	1	1
Nyctinomops	0.74	0.93	0.82
Nycticeius	0.82	0.88	0.85
Peropteryx	0.86	0.89	0.88
Pipistrellus	0.85	0.96	0.90
Promops	0.88	1	0.93
Pteronotus	0.98	0.98	0.98
Rhogeessa	0.92	0.89	0.90
Rhynchonycteris	1	1	1
Saccopteryx	1	0.97	0.99
Sturnira	0.93	0.91	0.92
Tadarida	0.92	0.8	0.86
Thyroptera	1	0.92	0.96
Trachops	0.77	0.96	0.85
ACCURACY	0.88		

Table 5: Classification report for the Genus classification model.

In the above classification report, we observe high values for all the three metrics across all the Genus, which indicates that the model performs quite well in identifying all the individual genus.

3.4 Classifying the Family

Another aspect of this study deals with the predictions of the Family of the subject whose call acoustics, country, and source file are known. Family, being at the topmost level of the

hierarchy, is probably the first characteristic to be identified during the study of any animal species. Before jumping on training the model with Family as the target variable directly, it is important to observe the proportion of observations in each of the class of the feature Family. Conclusively, we observe an unbalance across Families is tabulated below in Table 6 for easy interpretation. Here we see that approximately 50% of the observations come from the Vespertillionidae and only 0.66% of acoustic calls are there from the Thyropteridae.

Family	% of Total Count of Family
Vespertilionidae	49.65%
Phyllostomidae	15.11%
Emballonuridae	11.14%
Molossidae	10.1%
Mormoopidae	9.54%
Natalidae	2.07%
Noctilionidae	1.73%
Thyropteridae	0.66%

Table 6: Proportion of Observations in each Family.

Though this unbalance is worth addressing, it is still not an intricate issue in training the model as the neural network quite efficiently handle the mild imbalance in the data here. The accuracy of this model for the classification of the Family on the validation set turns out to be a whopping 97.13% (as compared to a 91.7% accuracy for the random forest counterpart (Zamora-Gutierrez, 2016)), which is what we expected as it is much easier to make an accurate prediction of the Family of a bat as compared to the Species and the Genus as it is less diverse with only eight output classes due to its position at the top level of the biological hierarchy. We observe the classification performance on each of the Family through the multi-class confusion matrix in Figure 20 and condense it to a classification report in Table 7 that examines the precision, recall, and f1-score on each Family class.

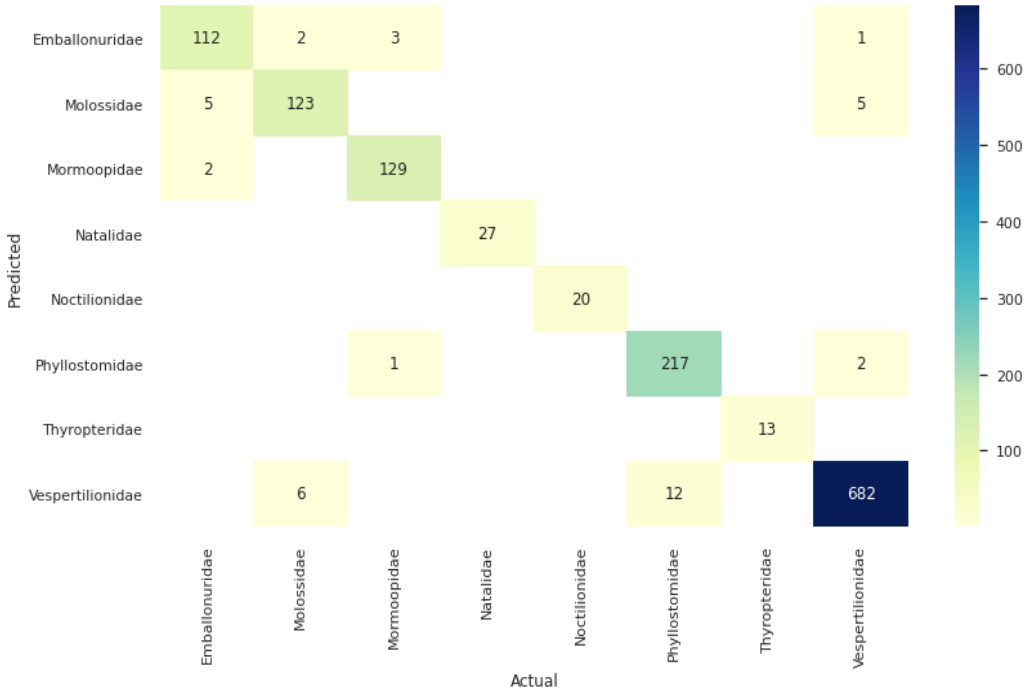


Figure 20: Confusion Matrix for the Model on Family Classification.

	precision	**recall**	**f1-score**
Emballonuridae	0.94	0.95	0.95
Molossidae	0.94	0.92	0.93
Mormoopidae	0.97	0.98	0.98
Natalidae	1	1	1
Noctilionidae	1	1	1
Phyllostomidae	0.95	0.99	0.97
Thyropteridae	1	1	1
Vespertilionidae	0.99	0.97	0.98
accuracy	0.97		

Table 7: Condensed Classification Report on each Family

From table 7, we see that precision and recall for each of the eight Families are very high, which is evident that the model performs quite well in classifying each class of the family. Consequently, we observe high f1-scores which illuminate and further strengthens the same inference.

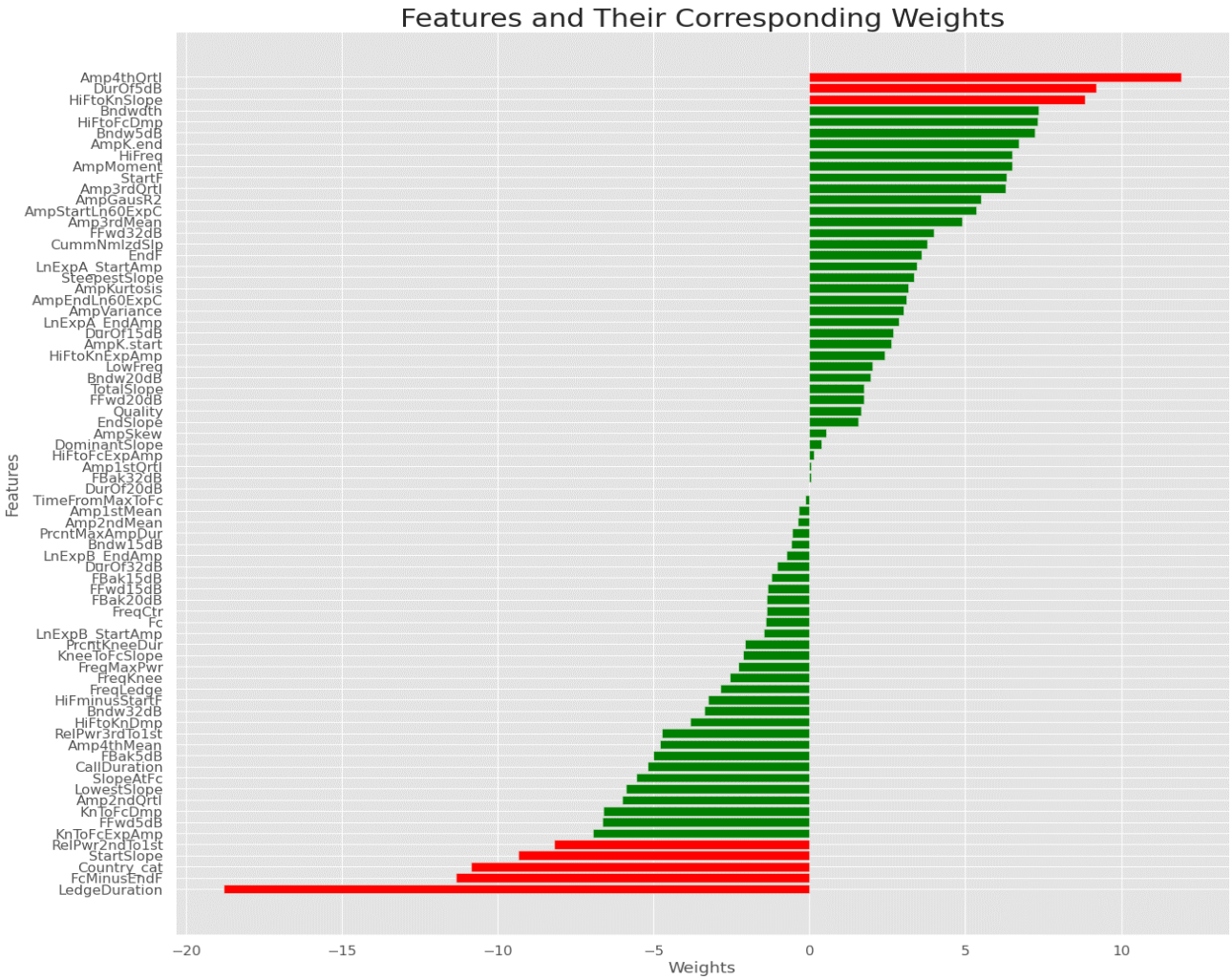


Figure 21: Feature ranking using weights where the eight most important features are colored red.

Again, we observe Country among the eight most important predictors (see figure 21) which means that the neural network gives a high weight to the feature *Country_cat* indicating that Country is an important predictor in identifying the family of a subject whose acoustics are known.

3.5 Identifying the geographic location of the subject when call acoustics are known

The objective here, as the heading says, is to identify the location of the subject whose recorded call acoustics, in essence, the acoustic constituents of the call are known. This, I believe, would be highly useful for the biologists in carrying the study on the ecological traits of the bats.

Moreover, the knowledge of the location of an unknown specie coupled with the knowledge of exotic characteristics of the other species in the region would be highly informative in identifying the similarity in behaviors which, if not the exact species, might yield the top levels of the hierarchy (Family or Genus).

Unexpectedly and quite surprisingly, the model performs quite well in identifying the location of the unlabelled data despite a massive imbalance among the classes of the response, i.e. Country. To assure that the model does not overfit the training data, I increased the dropout to 0.40, 0.35, and 0.35 after each hidden layer and observed a validation accuracy of 91.7%. The network structure and the evaluation metrics are illustrated in figure 22 and figure 23 respectively.

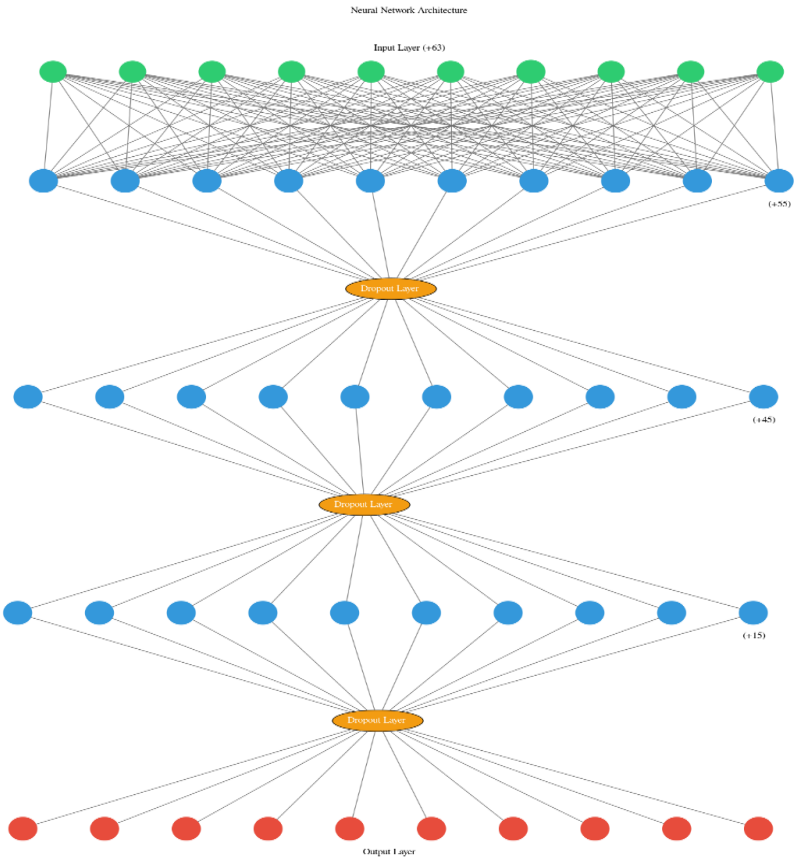
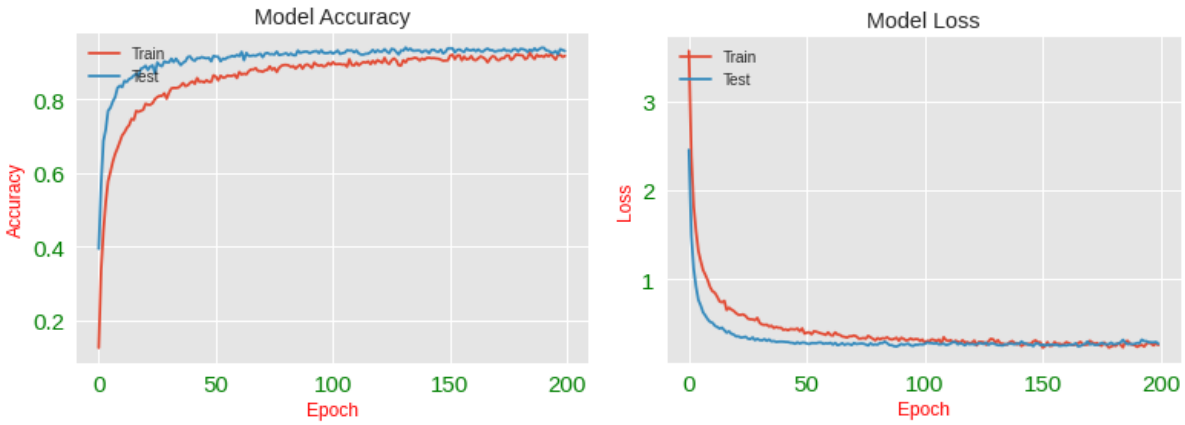


Figure 22: Network Structure for the model on identifying the location.



(a) Training and Validation Accuracy (b) Training and Validation Loss

Figure 23: Model Performance for Identifying the Location of the recorded call.

The convergence of the training and validation loss evident from figure 24(b) assures that the model is not overfitting the training data. Furthermore, the classification performance on each Country is reflected by the multi-class confusion matrix in figure 24 which is further tabulated into a classification report in Table 8. To evaluate the classification performance on imbalanced data, f1-score is a more reliable metric as it is the weighted average of precision and recall.

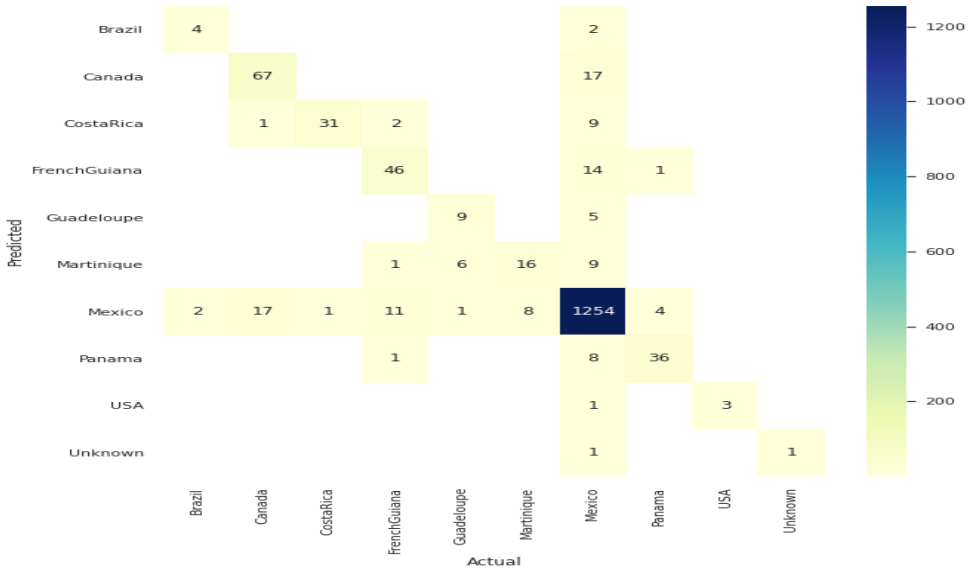


Figure 24: Confusion Matrix for the Model Identifying the Location of the recorded observation.

	precision	recall	f1-score	support
Brazil	0.67	0.67	0.67	6
Canada	0.79	0.80	0.79	84
CostaRica	0.97	0.72	0.83	43
FrenchGuiana	0.75	0.75	0.75	61
Guadeloupe	0.56	0.64	0.6	14
Martinique	0.67	0.5	0.57	32
Mexico	0.95	0.97	0.96	1298
Panama	0.88	0.8	0.84	45
USA	1	0.75	0.86	4
Unknown	1	0.5	0.67	2
accuracy	0.92			

Table 8: Classification report precision, recall, and f1-score on each Country.

The precision, recall, and f1-score altogether indicate that the model exhibits a decent overall performance with an issue with the countries Martinique, Guadeloupe, and Brazil having a low f1-score. Support indicates the number of observations from a particular Country in the test set. Despite having low support, the model yields a high f1-score for the USA however the performance on the country Martinique is unsatisfactory despite having higher support. The model performs the best on Mexico as it has the maximum support count. The performance of the network for this classification problem could be further improved on underrepresented countries by up-sampling the data from the locations other than Mexico which would be a future endeavor of this study *(Johnson, J.M. and Khoshgoftaar, T.M., 2019)*.

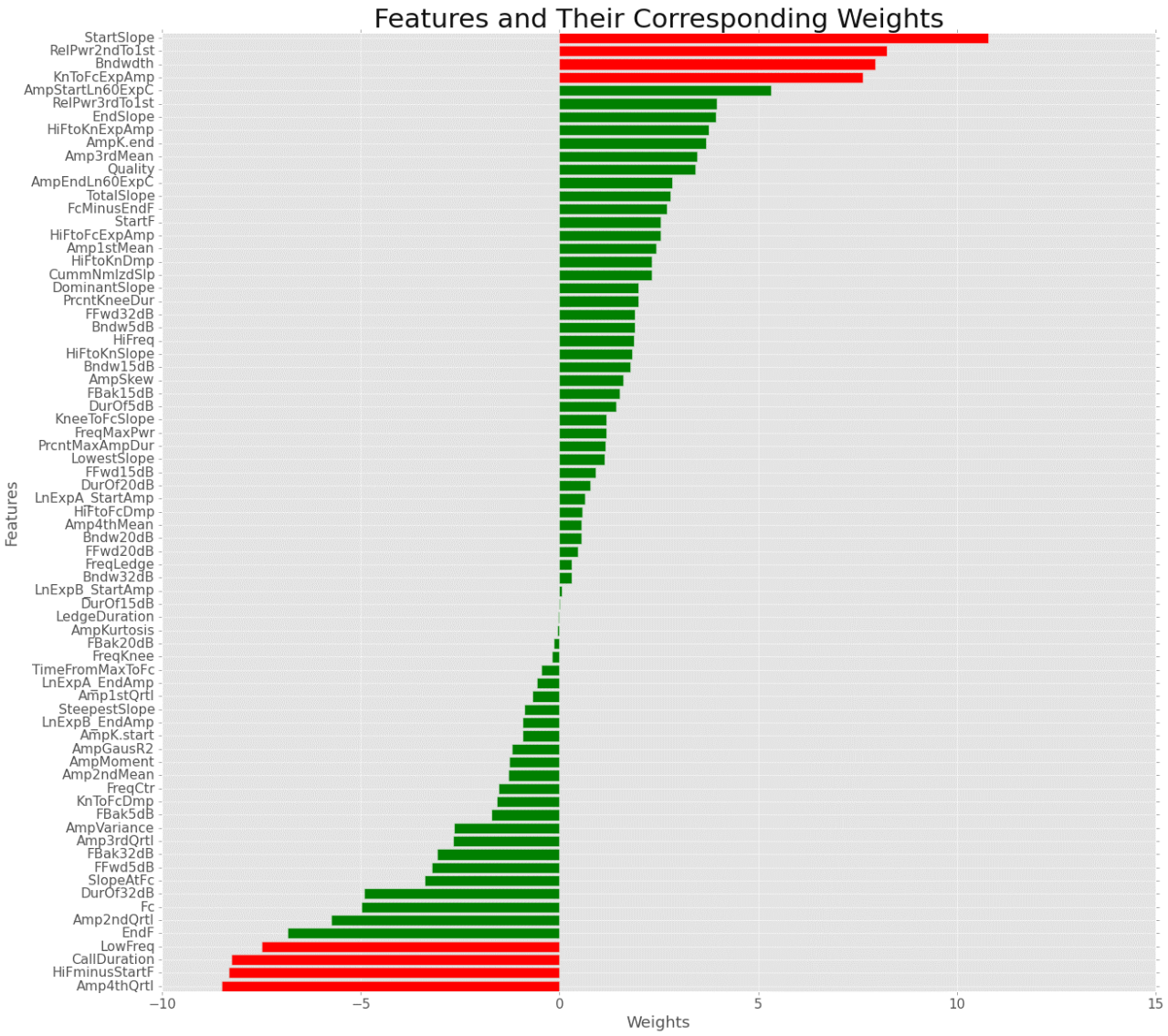


Figure 25: Feature ranking using weights where the eight most important features are colored red.

From the above Weights vs Features plot (see figure 25), we observe that the feature *CallDuration* is among the most significant features. Thus, it might not be incorrect to say that the duration of calls is a factor that varies among species from different geographic locations. Fascinated by this part of the study, I intend to pursue a focussed exploration and analysis of this association to a further nuanced level in my future work.

Evaluating the Results

As we expected, the DNN classifier gives an improved accuracy across all the classifiers as compared to Zamora-Gutierrez's random forest classifiers. The comparison of the accuracies of both the models is summarized in table 9 for a quick evaluation of the results.

Evaluating performances across models

Classifier	Validation Accuracies (%)	
	Random Forest	Neural Network
Species Classification	66%	84.6%
Species within Family	72%	85.9%
Species within Genus	71.2%	90.23%
Genus classification	77.8%	88%
Family classification	91.3%	97.13%
Country Identification	Unexplored	91.7%

Table 9: A comparison of the accuracies obtained by the Random Forest models proposed by (Zamora-Guiterrez, 2016) and the DNN classifiers in this study.

CONCLUSION

The study shows that the intricacy involved in the classification of bat species, which the existing machine learning algorithms could not tackle, shows a massive improvement with the deployment of the deep neural networks and hence opens the doors for further research in this field. We observe impressive results on each specie along with high overall model accuracy for the prediction of Species from the call acoustics as compared to the low accuracies for the random forest model fitted in the previous study. Extending to the hierarchical classification modeling, we observed an even better performance when the family or genus is known. The new aspect of identifying the geographical location covered in this study gave unexpectedly good results despite the huge unbalance among the classes, even on the underrepresented countries which had low support. This area of study, I believe, have a great potential for future research in the biological domain and can find its application in investigating the location-oriented ecological characteristics.

Chapter 4

PROSPECTIVE WORK

4.1 Yellowbrick Visualization

During the course of my thesis, a challenging yet exciting part was to create informative visualizations that describe the performance and inferences driven by the neural networks. Throughout this voyage, I felt that in comparison to machine learning algorithms, the neural networks have limited options of visualizations. Exploring the options, I came across a wonderful module of the sklearn library called YellowBrick that is useful for generating visualization for machine learning models *(Bengfort, B. and Bilbro, R., 2019)*, however, the module has some compatibility issues with the keras deep learning models and hence cannot be used for creating visualizations for keras neural networks. Driven by fascination towards visualization in deep learning, I wish to pursue my future work in the direction of making this module available for keras models as well.

4.2 LIME for decrypting neural networks

Neural Networks being a black box technique, makes it hard to decompose it to extract the information on feature ranking and feature elimination. While reading on this subject and exploring the possibilities, I came across this extremely simple method of interpreting the performances of classifiers called LIME. LIME (Local Interpretable Model-agnostic Explanations) is a novel explanation technique that explains the prediction of any classifier in an interpretable and faithful manner by learning an interpretable model locally around the prediction *(Melis, D.A. and Jaakkola, T., 2018)*. My future endeavors revolve around making these resources available for keras neural networks and improve their interpretability.

4.3 A Deep Learning Model with the actual audio files

Deep Learning is getting popular nowadays due to the flexibility of deep learning models in dealing with a vast variety of data which include image, video, and audio data as well. Thus, I

wish to combine the concepts of deep learning and Independent Component Analysis (proposed by *(Hyvärinen, A. and Oja, E., 2000)*) to classify the bat species from the gaussian mixture of their call audios. This, I believe, would yield even better and extremely reliable results as the collected data would be in its purest form and hence the possible induction of error or noise while decomposing the acoustic components of the call would be avoided. Additionally, I would even like to explore the dropout-based feature selection approach, proposed by *(Chang, C.H., 2017)*, to further penetrate the black box of the neural network.

References

[1] Aggarwal C.C. (2018) Recurrent Neural Networks. In: Neural Networks and Deep Learning. Springer, Cham. https://doi.org/10.1007/978-3-319-94463-0_7

[2] Chang, C.H., Rampasek, L. and Goldenberg, A., 2017. Dropout feature ranking for deep learning models. *arXiv preprint arXiv:1712.08645*.

[3] Zamora-Gutierrez, V., Lopez-Gonzalez, C., MacSwiney Gonzalez, M.C., Fenton, B., Jones, G., Kalko, E.K., Puechmaille, S.J., Stathopoulos, V. and Jones, K.E., 2016. Acoustic identification of Mexican bats based on taxonomic and ecological constraints on call design. *Methods in Ecology and Evolution*, *7*(9), pp.1082-1091.

[4] Zhao Yanling, Deng Bimin and Wang Zhanrong, "Analysis and study of perceptron to solve XOR problem," The 2nd International Workshop on Autonomous Decentralized System, 2002., Beijing, China, 2002, pp. 168-173, doi: 10.1109/IWADS.2002.1194667.

[5] Bengfort, B. and Bilbro, R., 2019. Yellowbrick: Visualizing the scikit-learn model selection process. *Journal of Open Source Software*, *4*(35), p.1075.

[6] Ketkar N. (2017) Introduction to Keras. In: Deep Learning with Python. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-2766-4_7

[7] Melis, D.A. and Jaakkola, T., 2018. Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems* (pp. 7775-7784).

[8] Hyvärinen, A. and Oja, E., 2000. Independent component analysis: algorithms and applications. *Neural networks*, *13*(4-5), pp.411-430.

[9] Féraud, R. and Clérot, F., 2002. A methodology to explain neural network classification. *Neural networks*, *15*(2), pp.237-246.

[10] Bhardwaj, A., Tiwari, A., Bhardwaj, H. and Bhardwaj, A., 2016. A genetically optimized neural network model for multi-class classification. *Expert Systems with Applications*, *60*, pp.211-221.

[11] Hecht-Nielsen, R., 1992. Theory of the backpropagation neural network. In *Neural networks for perception* (pp. 65-93). Academic Press.

[12] Che, Z.G., Chiang, T.A. and Che, Z.H., 2011. Feed-forward neural networks training: a comparison between genetic algorithm and back-propagation learning algorithm. *International journal of innovative computing, information and control*, *7*(10), pp.5839-5850.

[13] Specht, D.F., 1990. Probabilistic neural networks. *Neural networks*, *3*(1), pp.109-118.

[14] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[15] Ward, R., Wu, X. and Bottou, L., 2019, May. Adagrad stepsizes: sharp convergence over nonconvex landscapes. In *International Conference on Machine Learning* (pp. 6677-6686).

[16] Johnson, J.M. and Khoshgoftaar, T.M., 2019. Survey on deep learning with class imbalance. *Journal of Big Data*, *6*(1), p.27.