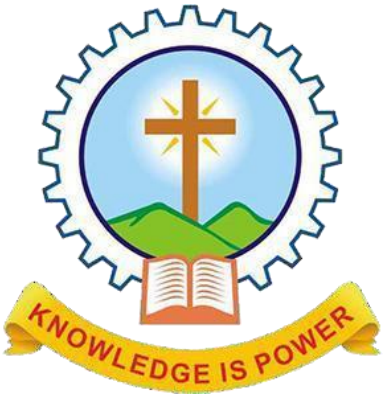
MAR ATHANASIUS COLLEGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University, TVM)

KOTHAMANGALAM



Department of Computer Applications

Main Project Report

BIRD SOUND CLASSIFICATION

Done by

Aneeta L R

Reg No: MAC21MCA-2004

Under the guidance of

Prof. Nisha Markose

2021-2023

MAR ATHANASIUS COLLEGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University, TVM)

KOTHAMANGALAM

CERTIFICATE

BIRD SOUND CLASSIFICATION

Certified that this is the bonafide record of Project work done by

Aneeta L R

Reg No: MAC21MCA-2004

During the academic fourth semester, in partial fulfillment of requirements for award of the degree,

Master of Computer Applications

of

APJ Abdul Kalam Technological University Thiruvananthapuram

Faculty Guide

Prof. Nisha Markose

Head of the Department

Prof. Biju Skaria

Project Coordinator

Prof. Biju Skaria

External Examiner

ACKNOWLEDGEMENT

First and foremost, I thank God Almighty for his divine grace and blessings in making all this possible. May he continue to lead me in the years to come. No words can express my humble gratitude to my beloved parents who have been guiding me in all walks of my journey.

I am also grateful to Prof. Biju Skaria, Head of Computer Applications Department and Project Coordinator for his valuable guidance and constant supervision as well as for providing necessary information regarding the Main project & also for his support.

I would like to express my special gratitude and thanks to my Project Guide Prof. Nisha Markose, Associate Professor, Department of Computer Applications for giving me such attention and time.

I profusely thank other Professors in the department and all other staff of MACE, for their guidance and inspirations throughout my course of study. My thanks and appreciation also go to my friends and people who have willingly helped me out with their abilities.

ABSTRACT

Topic: - Bird Sound Classification

Birds help a lot to shape the plant life we see around us. By recognizing bird songs researches and scientists can monitor wildlife and study the behaviour of birds. Every species of birds has their unique sounds. By using their sound, we can identify the birds to study about them. Recording sound is easier than finding or taking picture of bird.

The proposed system helps in identifying the species of birds from their sound. The architecture used to build the model is Resnet50 architecture. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Before passing the input to Resnet50, the inputted audio is pre-processed and converted to Mel-spectrogram.

The model identifies the audio as one of 5 classes (bird species), they are Blue Jay, Black-capped Chickadee, Mallard, Common Tern and American Redstart. It contains 1250 audio files, it’s an unbalanced dataset. By predicting the species from their sound, the researches go forward with their research on that bird.

LIST OF TABLES

Table 2.1. Literature summary……5

Table 3.1. Dimension table of the architecture…….31

LIST OF FIGURES

Fig 3.1. Mel-spectrogram of American redstart…….8

Fig 3.2. Mel-spectrogram of Black-capped chickadee……...9

Fig 3.3. Mel-spectrogram of Blue jay……...10

Fig 3.4. Mel-spectrogram of Common tern……..11

Fig 3.5. Mel-spectrogram of Mallard……... 12

Fig 3.6. Data visualization graph……..15

Fig 3.7. A residual block of deep residual network……..16

Fig 3.8. Bottleneck……17

Fig 3.9. Block diagram of architecture…….18

Fig 3.10. Skip connection…….19

Fig 3.11. Convolutional layer……...20

Fig 3.12. Process in max pooling layer…….21

Fig 3.13. Process in average pooling layer……...22

Fig 3.14. Convolutional block……..23

Fig 3.15. Identity block…….24

Fig 3.16. FC layer…….24

Fig 3.17. Convolution layer working with a stride of 1……26

Fig 3.18. Padding……..27

Fig 3.19. Working of convolutional layer…… 28

Fig 3.20. ReLU graph……...29

Fig 3.21. Softmax graph……...30

Fig 3.22. Development pipeline……33

Fig 3.23. Deployment pipeline……..34

Fig 4.1. Implementation code……...40

Fig 4.2. Model compile……41

Fig 4.3. Model training……42

Fig 4.4. Epoch 1 to 11……...43

Fig 4.5. Epoch 12 to 25……44

Fig 4.6. Epoch 26 to 31……45

Fig 4.7. Epoch 32 to 41……46

Fig 4.8. Epoch 42 to 52……47

Fig 4.9. Epoch 53 to 63……48

Fig 4.10. Epoch 64 to 73……...49

Fig 4.11. Epoch 74 to 83……...50

Fig 4.12. Epoch 84 to 93……...51

Fig 4.13. Epoch 94 to 100……52

Fig 4.14. Model evaluation ……..53

Fig 5.1. Accuracy graph……55

Fig 5.2. Loss graph……...55

Fig 5.3. Classification report……56

Fig 5.4. Confusion matrix……....56

Fig 6.1. Home page……...59

Fig 6.2. Home page when choosing file……...59

Fig 6.3. Result page input and output section……...60

Fig 6.4. Result page pre-processed data section……... 60

Fig 6.5. Result page model input section……..61

Fig 7.1. Git repository……...62

Fig 7.2. Git history……62

CONTENTS

1. Introduction **1**

2. Supporting Literature **2**

2.1. Literature Review……2

2.2. Findings and Proposals…….6

3. System Analysis **7**

3.1. Analysis of Dataset……7

3.1.1. About the dataset……7

3.1.2. Explore the Dataset…….7

3.2. Data Pre-processing……12

3.2.1. Data Cleaning……...12

3.2. 2. Analysis of Feature Variables……..13

3.2. 2. Analysis of Class Variables…….13

3.3. Data Visualization……...14

3.4. Analysis of Architecture……..15

3.4.1. Detailed study of Architecture……..15

3.5. Project pipeline……32

3.6. Feasibility Analysis ……34

3.6.1. Technical Feasibility…….35

3.6.2. Economic Feasibility……35

3.6.3. Operational Feasibility……..35

3.7. System Environment……36

3.7.1. Software Environment……..36

3.7.2. Hardware Environment…….39

4. System Design 40

4.1. Model Building……40

4.1.1 Implementation Code…….40

4.1.2 Model Planning……..41

4.1.3 Training……..41

4.1. 4 Testing……..53

5. Result and Discussion 54

6. Model Deployment 58

7. Git History 62

8. Conclusion 63

9. Future Work 64

10. Appendix 65

10.1. Minimum Software Requirements…….65

10.2. Minimum Hardware Requirements……...65

11. References 66

1. INTRODUCTION

There are some researchers and scientists who are studying about bird. This study is known as ornithology (a branch of zoology) and the researchers are known as ornithologist. Birds help a lot to shape the plant life we see around us. By recognizing bird songs ornithologist can monitor wildlife and study the behaviour of birds. Every species of birds has their unique sounds. But identifying a bird is a difficult method because we can't find birds physically. Bird tends to fly away from human presence. Birds tends to hide from humans also. So, seeing them and taking pictures of them for identification is difficult. But recording the sound of bird is easy and the recording can be useful in the future researches.

Their sound can be heard up to a certain distance and with the help of new technologies (smartphones, audio recorders, etc) we can record those sound. These recorded sounds can be used to find bird species because different bird species are having different sounds. Due to easily available recordings of bird sounds we can easily convert into spectrogram train the model with CNN architecture. People who are curious about wildlife can also use this system, all the have to do is record the bird sound and upload it into system.

The proposed system is a web application called bird sound classification. It helps in identifying the species of bird using their recorded sound. By identifying the bird helps researchers and scientist in ornithology study. This system uses the model built using Resnet50 architecture. Resnet50 is a convolutional neural network architecture. Resnet stand for Residual Network and 50 means it has 50 layers. The dataset used to train and evaluate the model is taken from a part of an online bird sound repository called xeno-canto. They are available as a dataset in Kaggle. It has many different bird species but here, we are using only 5 of them. They are Blue Jay, Black-capped Chickadee, Mallard, Common Tern and American Redstart. It is an unbalanced dataset having total 1250 audio files. By predicting the species from their sound, the researches go forward with their research on that bird.

2. SUPPORTING LITERATURE

2.1. Literature Review

Paper 1: Noumida, A., and Rajeev Rajan. "Deep learning-based automatic bird species identification from isolated recordings." 2021 8th International Conference on Smart Computing and Communications (ICSCC). IEEE, 2021.

The paper “Deep learning-based automatic bird species identification from isolated recordings” was published on 2021. Its authors are Noumida A and Rajeev Rajan. It was published by IEEE. It is a comparison between ResNet50, CNN, MFCC-DNN, InceptionResNetV2 and VGG-16. Conventional field techniques to identify and track distinct bird species have required much human effort. Deep learning methods can achieve higher detection accuracy on remote monitoring auditory data despite weather noise and a wide diversity of bird call kinds. This paper aims to develop an effective bird call classification for isolated recordings approach using various deep learning architectures, namely convolutional neural networks (CNN) and deep neural networks (DNN). The performance of models such as ResNet50, VGG-16, and InceptionResNetV2 has been compared to the acoustic MFCC-DNN methodology. In MFCC, the frequency bands are logarithmically positioned. Therefore, it approximates the response of the human auditory system more closely than any other system. But as per this paper mel-spectrogram gives more accuracy. This paper mainly focuses on single-label isolated audio recordings. Authors used 10 species with 1078 audio tracks from xeno-canto for this comparative study. Bird species identification has been done by converting audios to mel-spectrograms. Before converting into a mel-spectrogram the audios are trimmed. The trimming help in displaying the frequency beam to be clearer in mel-spectrogram. As per this paper ResNet50 is having 96.3%, CNN is having 93.7%, MFCC-DNN is having 87.7%, InceptionResNetV2 is having 87% and VGG-16 is having 91.9%. The challenge with automatic audio recordings is detecting the calls of interest species in lengthy recordings. The challenge with automatic audio recordings is detecting the calls of interest species in lengthy recordings. The results show that ResNet50, CNN, and VGG-16 outperforms the DNN and prove to be far better.

Paper 2: Saad, Aymen, Javed Ahmed, and Ahmed Elaraby. "Classification of Bird Sound Using High-and Low-Complexity Convolutional Neural Networks." Traitement du Signal 39.1 (2022): 187-193.

The paper “Classification of Bird Sound Using High-and Low-Complexity Convolutional Neural Networks” is also a comparative paper published on 2022. The authors of this paper are Aymen Saad, Javed Ahmed, Ahmed Elaraby. This paper compares between ResNet-50 and MobileNetV1. These two model’s inputs were converted to spectrogram using two algorithms. They are Short-Time Fourier Transform (STFT) and Mel Frequency Cepstral Coefficient (MFCC) algorithms. Short-Time Fourier Transform (STFT) and Mel Frequency Cepstral Coefficients (MFCC) are used to convert sound to image. STFT is applied to the audio signal by splitting the signal into separate overlapping frames, and then computing the DiscreteTime Fourier Transform. CNN architectures such as Inception-v3 and ResNet perform classification tasks based on the ability of the deep layers of neural network models to extract high-level features from the input images. The ResNet-50 architecture is a typical state-of-the-art CNN with a depth of 50 layers and MobileNetV1 has 28 layers. CNN models' deep layers are reduced to 10 layers (10 classes). The dataset used to build the models were Brazilian bird sound dataset from Xeno-canto bird sound repository. Here, ResNet-50 is benchmarked with a high-complexity CNN model. The models will identify 10 Brazilian birds. For rarer species, data augmentation is regularly performed to create synthetic samples. Prepare the data, Preparation of training and testing image datasets involves splitting the sets into training and validation data are the steps done on dataset. The audios are segmented before converting to spectrogram. The spectrogram images are resized to 224*224. Accuracy achieved by these models are MobileNet-v1(STFT) - 71.57%, MobileNet-v1 (MFCC) - 85.73%, ResNet-50 (STFT) - 90.40% and ResNet-50 (MFCC) - 90.56%. ResNet-50 is a high-complexity CNN while MobilenetV1 is a low-complexity CNN targeted for mobile applications. The high complexity model, which is ResNet-50, is observed to have higher accuracy, but requires a longer time to make a prediction, compared to the low complexity model. This is due to a large number of operations.

Paper 3: Yang, Fan, Ying Jiang, and Yue Xu. "Design of Bird Sound Recognition Model Based on Lightweight." IEEE Access 10 (2022): 85189-85198.

The paper “Design of Bird Sound Recognition Model Based on Lightweight” was published on 2022. Its authors are Fan Yang, Ying Jiang, and Yue Xu. In the traditional field of machine learning, we use Mel-frequency cepstral coefficients (MFCC) and inverted Mel-frequency cepstral coefficients (IMFCC) as sound features to recognize the sound of birds. Nowadays, the deep learning model is used to classify bird sound data with high classification accuracy. It uses Short Time Fourier transform (STFT) and other methods to convert birds sound into the spectrum and used convolutional neural network to classify bird sounds. However, the generalization ability of most existing bird sound recognition models is poor, and the complicated algorithm is applied to extract bird sound features. To address these problems, this paper proposes a lightweight bird sound recognition model is proposed. To build a lightweight feature extraction and recognition network with MobileNetV3 as the backbone. A data set of 264 kinds of birds from various bird recognition competitions of Kaggle were used here. The duration of each sample data is more than 10 seconds; therefore, this paper intercepts the sample data at 5 seconds interval so that the duration of each sample data is the same. The Mel spectrum, which is widely used in speech recognition systems, is selected as the feature of the bird audio signal. This paper also introduce PSA (Pyramid Split Attention) module [30] is introduced, which can fully capture the spatial information of different scales to enrich the feature space. This paper also does a comparative study between their proposed system, ResNet50, DenseNet, VGG, MobileNet, ShuffleNet and EfficientNet. The lightweight model consists of one Inception block, two Bnecks blocks and 17 Bneck blocks stacked together. In order to reduce the number of parameters and computations, this paper reduces the number of backbone layers of MobileNetV3, and the kernel size of the depthwise convolution is 3 × 3. Proposed architecture is less costly but ResNet50 has more accuracy than the paper proposed architecture. Accuracy achieved by the models as per paper are ResNet50 - 97%, proposed - 95%, MobileNet – 90% to 92%, ShuffleNet – 93%.

Literature Summary

Paper	Author	Published on	Source	Summary
Deep learning-based automatic bird species identification from isolated recordings	Noumida A, Rajeev Rajan	2021	IEEE Xplore	• Comparison between ResNet50(96.3%), CNN(93.7%), MFCC-DNN(87.7%), InceptionResNetV2(87%) and VGG-16(91.9%) • Ten species with 1078 audio tracks from xeno-canto. • The challenge with automatic audio recordings is detecting the calls of interest species in lengthy recordings
Classification of Bird Sound Using High-and Low-Complexity Convolutional Neural Networks	Aymen Saad, Javed Ahmed, Ahmed Elaraby	2022	Research Gate	• Comparison between ResNet-50 and MobileNetV1 • Short-Time Fourier Transform (STFT) and Mel Frequency Cepstral Coefficient (MFCC) algorithms • MobileNet-v1(STFT) - 71.57%, MobileNet-v1 (MFCC) - 85.73%, ResNet-50 (STFT) - 90.40% and ResNet-50 (MFCC) - 90.56%
Design of Bird Sound Recognition Model Based on Lightweight	Fan Yang, Ying Jiang, and Yue Xu	2022	IEEE Xplore	• ResNet50, DenseNet, VGG, MobileNet, ShuffleNet, EfficientNet and custom MobileNet(proposed) • ResNet50 has more accuracy than the paper proposed architecture but proposed architecture is less costly. • ResNet50(97%), proposed(95%)

Table 2.1: Literature summary

2.2. Findings And Proposals

The first paper is a comparative paper between ResNet50, CNN. MFCC-DNN, InceptionResNetV2 and VGG-16. The paper's authors use 10 species recordings as dataset for building the model. The performance of models such as ResNet50, VGG-16, and InceptionResNetV2 has been compared to the acoustic MFCC-DNN methodology. But as per the paper Mel-spectrogram gives more accuracy to these models than MFCC. As per this paper ResNet50 is having 96.3%, CNN is having 93.7%, MFCC-DNN is having 87.7%, InceptionResNetV2 is having 87% and VGG-16 is having 91.9%.

The second paper is also a comparative paper between ResNet-50 and MobileNetV1 with Short-Time Fourier Transform (STFT) and Mel Frequency Cepstral Coefficient (MFCC) algorithms. These algorithms are used to convert sound to image. The models introduced will identify 10 Brazilian birds whose audio recordings are taken from xeno-canto repository. The audios are segmented before converting to spectrogram. The spectrogram images are resized to 224*224. Accuracy achieved by these models are MobileNet-v1(STFT) - 71.57%, MobileNet-v1 (MFCC) - 85.73%, ResNet-50 (STFT) - 90.40% and ResNet-50 (MFCC) - 90.56%. From this paper we can conclude that Resnet50 has more accuracy but take more time to predict.

The third paper introduce a lightweight model to identify the bird using their sound and the authors has also done a comparative study with their new model and existing deep neural network models. They have used MobileNetV3 as the backbone. A data set of 264 kinds of birds from various bird recognition competitions of Kaggle were used here. They also give a detailed information on data or audio pre-processing for building the model. The lightweight model consists of one Inception block, two Bnecks blocks and 17 Bneck blocks stacked together. Proposed architecture is less costly but ResNet50 has more accuracy than the paper proposed architecture. Accuracy achieved by the models as per paper are ResNet50 - 97%, proposed - 95%, MobileNet – 90% to 92%, ShuffleNet – 93%. From all the reviewed papers we can conclude that most commonly used data are taken from xeno-canto and model that gives more accuracy is Resnet50.

3. SYSTEM ANALYSIS

3.1. Analysis of dataset

3.1.1. About the dataset

Dataset was collected from:

https://www.kaggle.com/datasets/rohanrao/xeno-canto-bird-recordings-extended-a-m

Dataset used in this project is xeno-canto bird recordings taken from Kaggle. This dataset contains sound recordings of 264 species. It is an unbalanced dataset containing 14.7K audio files. These are mp3 files. From this dataset only 5 species were selected for this project. They are Blue Jay, Black-capped Chickadee, Mallard, Common Tern and American Redstart. Audio files are having 5 sec or more duration. The number of files selected as data all together is 1250. These audio files are trimmed into 5 to 10 sec until the end of audio and converted to Mel-spectrogram. These converted Mel-spectrogram are saved as a dataset for building the model. These Mel-spectrogram are of size 432 x 432 and in PNG format. Each class excepted around 900 spectrograms except Common Tern and Common Tern has around 700 spectrograms. These PNG files are then pre-processed (converted to 224 x 224 size) to pass it to the architecture.

3.1.2. Explore the Dataset

Dataset taken for this project contains 1250 mp3 files. There are 5 bird species used in this project. They are Blue Jay, Black-capped Chickadee, Mallard, Common Tern and American Redstart. Audio files are having 5 sec or more duration for each. These audios are trimmed into max of 10 sec i.e., if the audios are more 10 sec, then they are trimmed from its start to its end in an interval of 10 sec. Or else if it's less than 10 sec then it's not trimmed. Before trimming 1 sec from the starting is avoided. The audios are trimmed because when converting to Mel-spectrogram the frequency beam displayed will be congested otherwise leading to complex spectrogram to process the feature. So, the audios are trimmed and then converted to Mel-spectrogram. The Mel-spectrogram is combination of two concepts, they are Mel scale and spectrogram. The Mel Scale is a logarithmic transformation of a signal's frequency. Spectrogram is the

visual representation of the spectrum of frequencies of the signal as it changes with time. Mel Spectrograms are spectrograms that visualize sounds on the Mel scale. The Mel scale closely mimics human perception, then it offers a good representation of the frequencies that humans typically hear. These Mel-spectrogram are PNG images. The audio in the dataset when converted to mel-spectrogram we get more than 4500 images in total (all 5 classes). They are used for building the model. Before building the model the Mel-spectrogram is rescaled into appropriate size for architecture used for building model. Mel-spectrogram is also used for data visualization in audio data. The sample of Mel-spectrogram for each class is shown below:

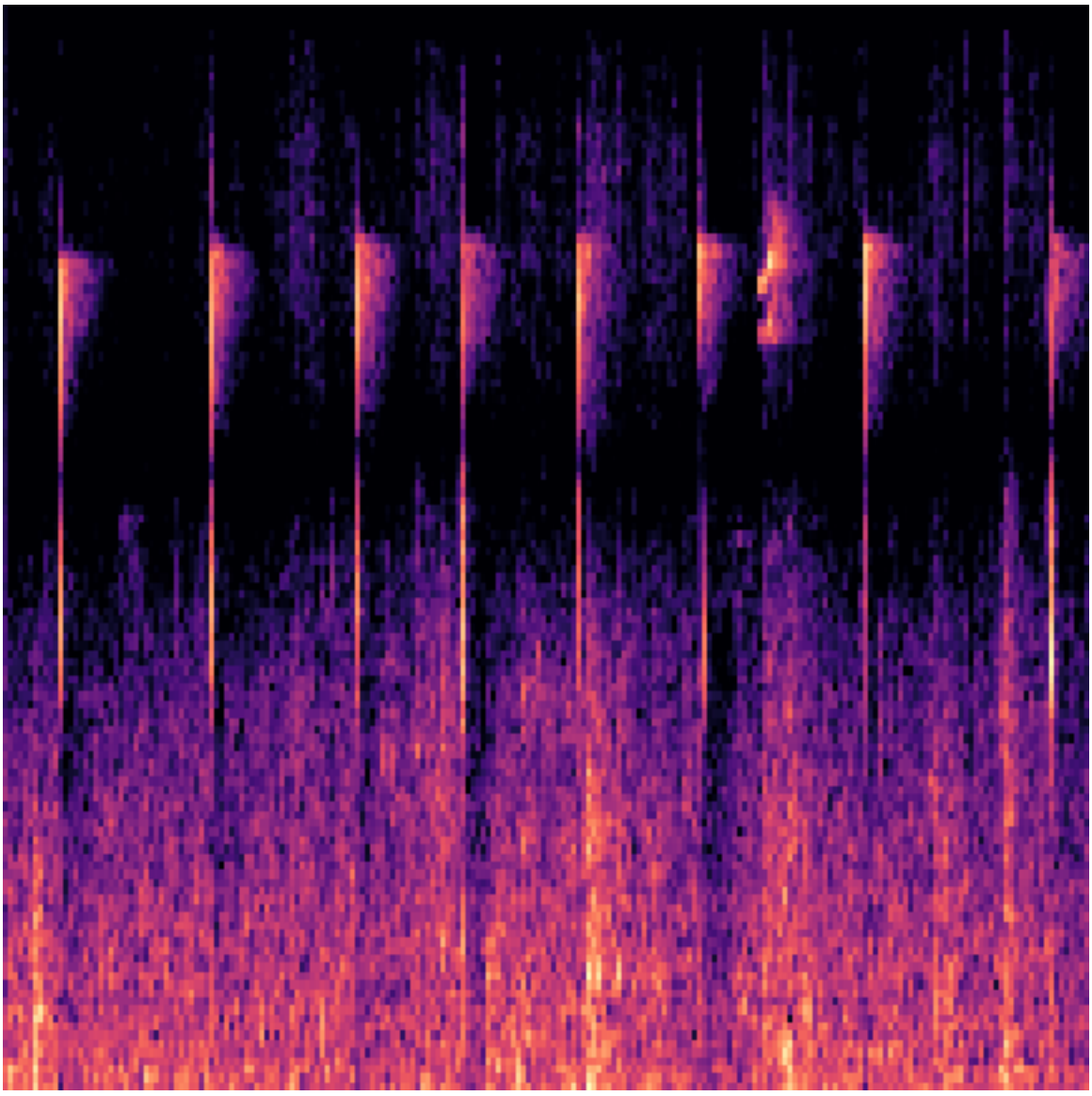


Fig 3.1: Mel-spectrogram of American redstart

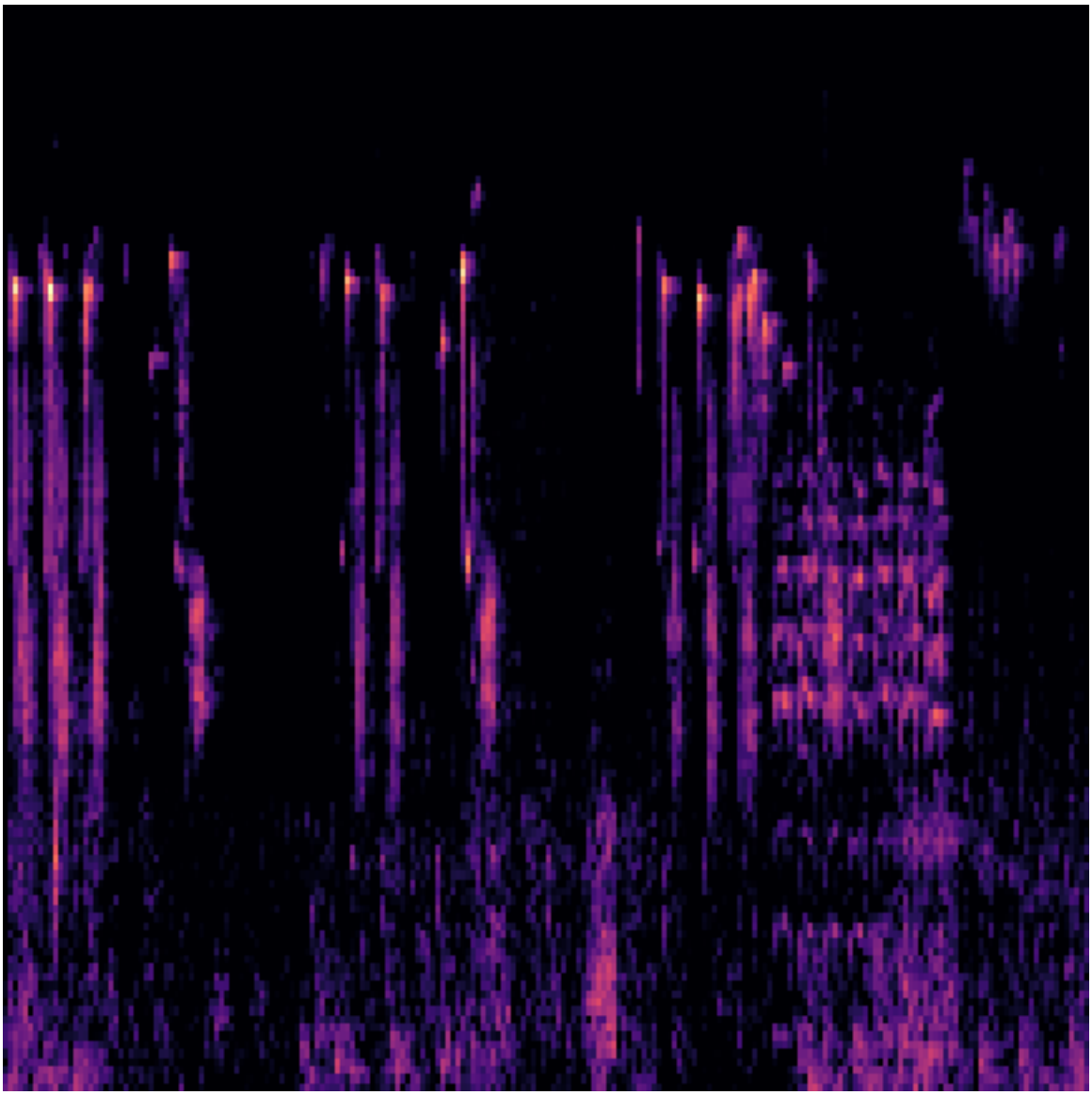


Fig 3.2: Mel-spectrogram of Black-capped chickadee

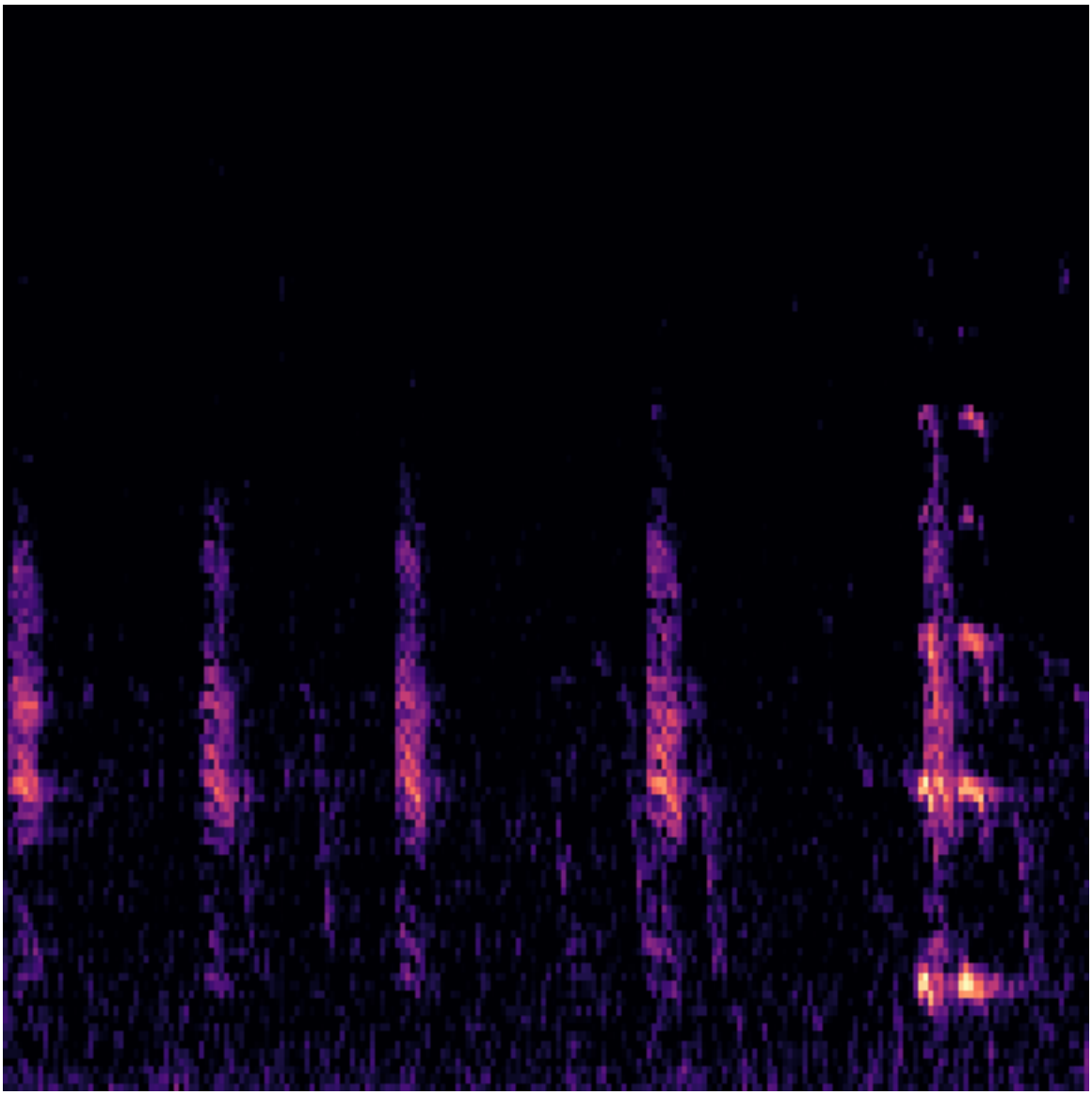


Fig 3.3: Mel-spectrogram of Blue jay

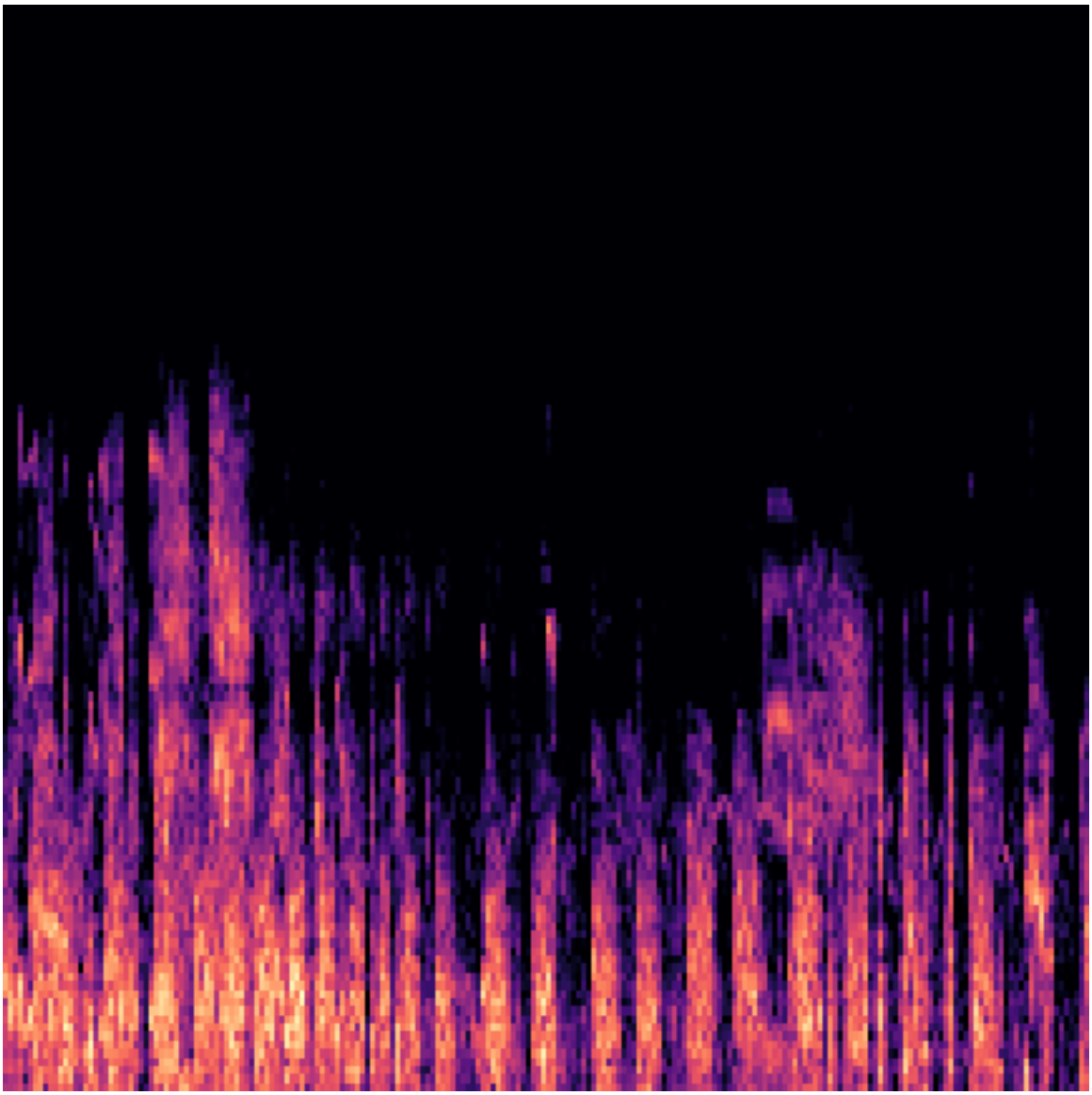


Fig 3.4: Mel-spectrogram of Common Tern

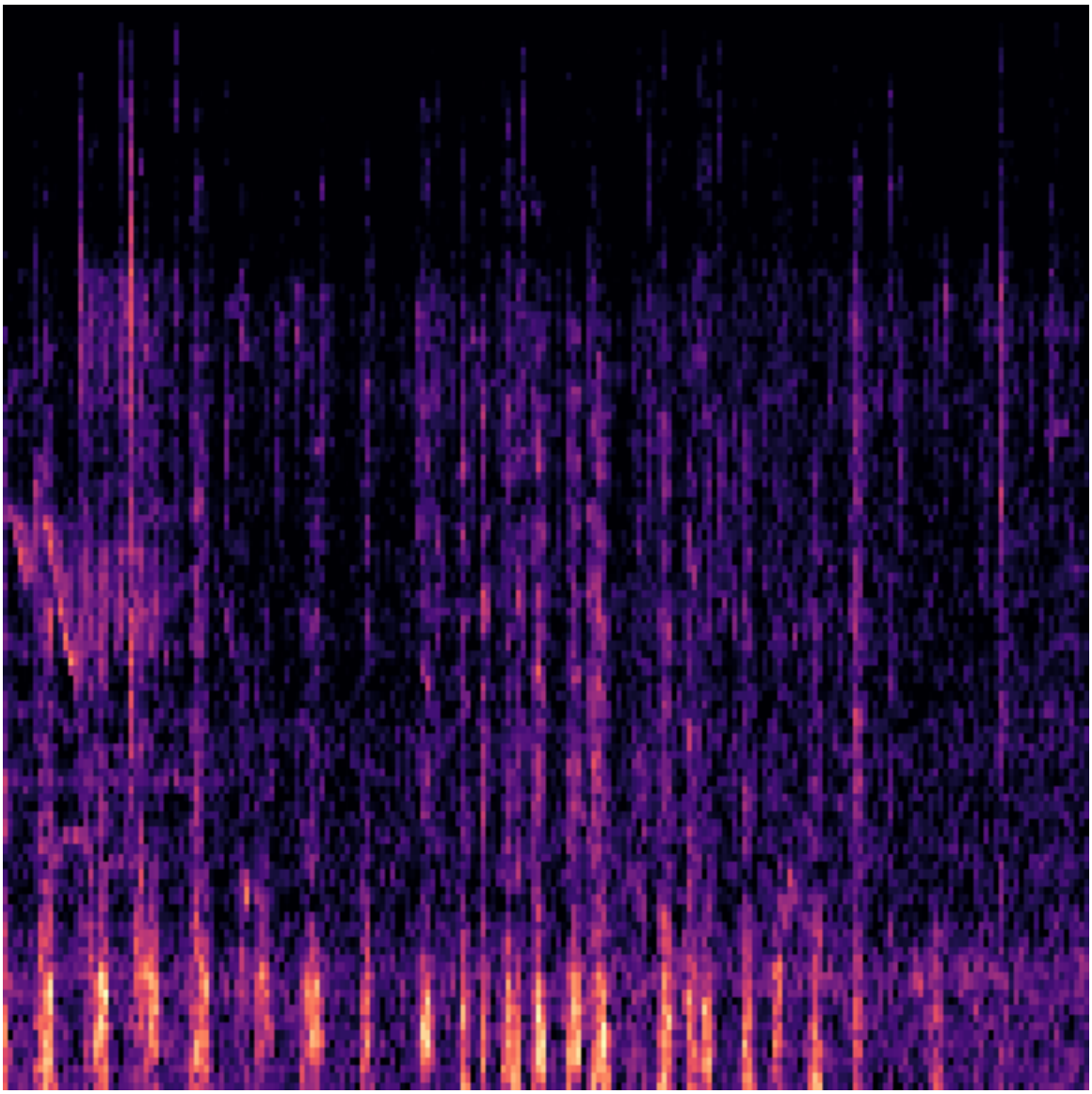


Fig 3.5: Mel-spectrogram of Mallard

3.2. Data Pre-processing

3.2.1 Data Cleaning

Audios in dataset are having different lengths or durations. We trim these audios for length of 10 sec to make it somewhat similar. But the main purpose of this is to make sure that each bandwidth or frequency line of the audios are clearer for feature extraction. That is, if a 20sec audio is converted to mel-spectrogram without trimming. The frequency bands shown in the mel-spectrogram will be extremely near to one another which may lead to difficult for the system to identify and learn the different between each class. So, audios are trimmed in an interval of 10 sec, by avoiding the 1

sec in the beginning of the audio. This 1 sec is ignored because most audios are having empty noise at their beginning. After pre-processing, they are converted to mel-spectrogram for feature extraction. Then these mel-spectrogram are again rescaled according to the size suitable for architecture.

3.2.2. Analysis of feature variables

Feature list is the attributes that describes the images and files in the dataset. The original dataset contains sound recordings of 264 species. It is an unbalanced dataset containing 14.7K audio files. These are mp3 files. From this dataset only 5 species were selected for this project about 1250 audio mp3 files. Audio files are 5 sec or more each. They are converted to Mel-spectrogram. Mel-spectrogram is an image having extension PNG. The image is having Width x Height x Depth - 432 x 432 x 3.

3.2.3. Analysis of class variables

The class variable in the proposed system handles five different bird species:

- Blue Jay
- Black-capped Chickadee
- Mallard
- Common Tern
- American Redstart

Each of these species has a unique set of vocalizations or bird songs, which can be analysed and used to classify the bird species based on its sound. Vocal sounds are made by a special organ only birds possess: the syrinx. The syrinx is located at the very top of the birds' windpipe. The air that comes in through the windpipe causes thin membranes to vibrate and produce sound. Their sound can be heard up to a certain distance. The Blue Jay (Cyanocitta cristata) is a large, colourful songbird found in North America. Their call is a loud and harsh. are known for their loud, clear whistled calls, which often sound like "jay-jay" or "queedle-queedle-queedle". They can also mimic the calls of other birds and animals. They also have a variety of other calls, including a soft, musical "whisper song" used during courtship. The Black-capped Chickadee (Poecile atricapillus) is a small, non-migratory bird found in North America. They have

a distinctive two-note "chick-a-dee-dee-dee" call, which they use to communicate with other birds in their flock. They also have a variety of other vocalizations, including songs and alarm calls. The Mallard (Anas platyrhynchos) is a common duck found in North America, Europe, and Asia. The male Mallard has a distinctive, raspy call that sounds like a nasal "quack", while the female has a softer, more muted quacking sound. Both males and females also make a variety of other vocalizations, including grunts and whistles.

The Common Tern (Sterna hirundo) is a migratory seabird found throughout the world. Their call is a high-pitched, shrill "kee-yah" sound, which they use to communicate with each other and to defend their nesting territories. They also have a variety of other calls, including a softer "churr" used during courtship. They also have a variety of other vocalizations for other calls. The American Redstart (Setophaga ruticilla) is a small, migratory songbird found in North America. They are warblers that have a variety of vocalizations, including songs and calls. Their songs are often described as "zee-zee-zee-zee" or "tsip-tsip-tsip-tsew". They also have a variety of calls, including alarm calls and begging calls. A softer "chip" is used during courtship. In summary, the class variable in the proposed system handles five different bird species, each with a unique set of vocalizations that can be analyzed to classify the species based on its sound.

3.3. Data visualization

One commonly used method for visualizing bird vocalizations is the spectrogram. A spectrogram is a visual representation of the frequency and intensity of sound over time. In a spectrogram, time is represented on the x-axis, frequency on the y-axis, and intensity is represented by color. By analyzing spectrograms of bird vocalizations, researchers can identify unique patterns and features that distinguish between different species. Here, we use mel-spectrogram to both visualize and build the model. The Mel-spectrogram is combination of two concepts, they are Mel scale and spectrogram. The Mel Scale is a logarithmic transformation of a signal’s frequency. The dataset we have selected here contains more than 4500 png images when converted to

mel-spectrogram. The graphical representation of how much mel-spectrogram is found in each class is shown below:

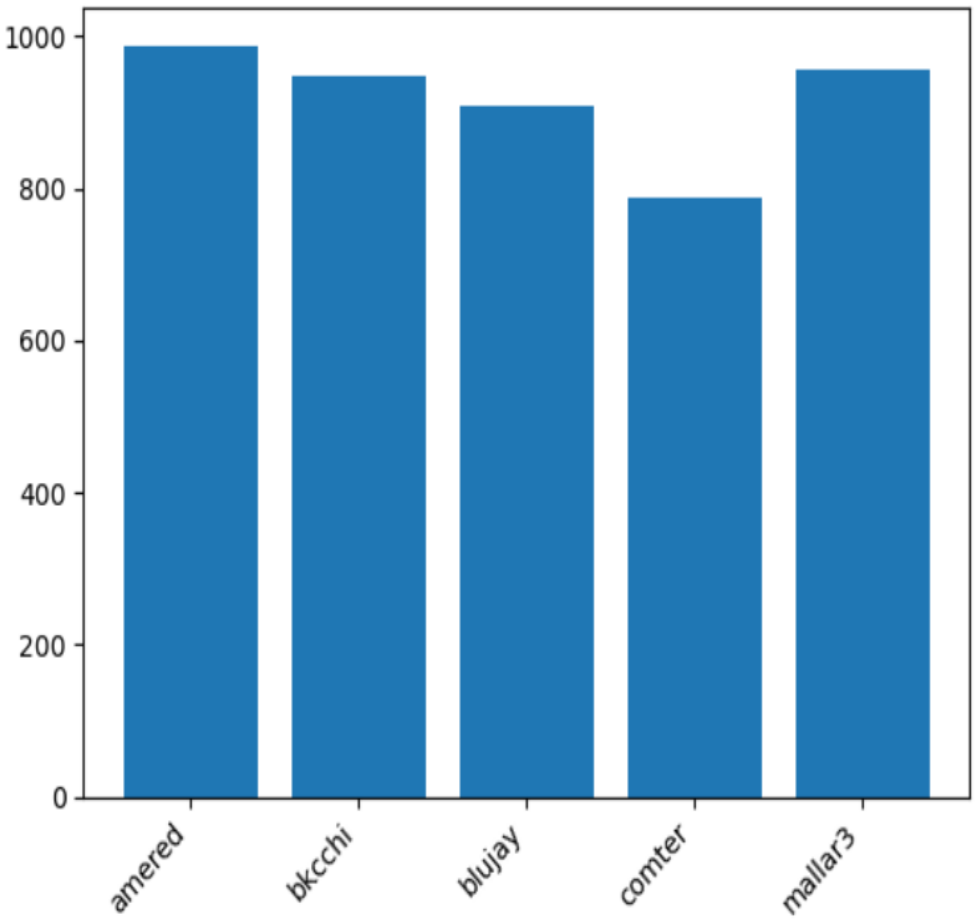


Fig 3.6: Data visualization graph

As per the graph American redstart has more mel-spectrogram than any other and common tern has the least number of mel-spectrograms.

3.4. Analysis of Architecture

3.4.1. Detailed study of Architecture

ResNet stands for Residual Network and is a specific type of convolutional neural network (CNN). ResNet is one of the most powerful deep neural networks which has achieved fantabulous performance results in the ILSVRC 2015 classification challenge. ResNet has achieved excellent generalization performance on other

recognition tasks and won the first place on ImageNet detection, ImageNet localization, COCO detection and COCO segmentation in ILSVRC and COCO 2015 competitions. There are many variants of ResNet architecture i.e., same concept but with a different number of layers. We have ResNet-18, ResNet-34, ResNet- 50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202, etc. The name ResNet followed by a two or more-digit number simply implies the ResNet architecture with a certain number of neural network layers.

Deep Residual Network is almost similar to the networks which have convolution, pooling, activation and fully-connected layers stacked one over the other. The only construction to the simple network to make it a residual network is the identity connection between the layers.

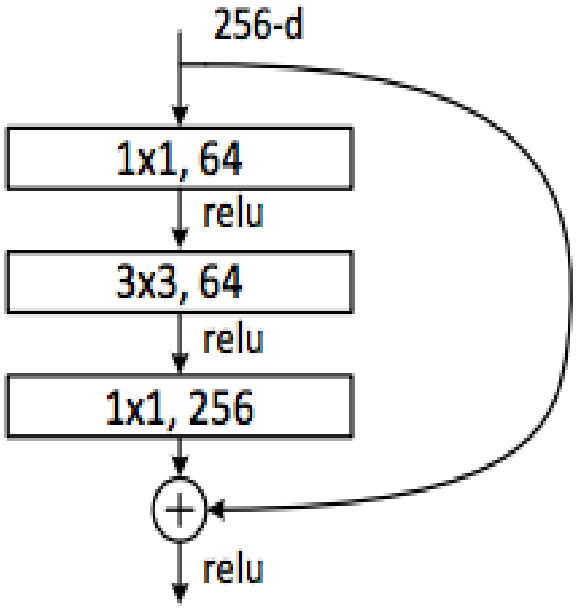


Fig 3.7: A residual block of deep residual network.

The architecture used in this project is ResNet-50. ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks. It is composed of multiple residual blocks, each of which contains a shortcut connection that allows the network

to learn residual functions. These shortcut connections help to mitigate the vanishing gradient problem and enable the network to learn more complex features from the input data. The network can take the input image having height and width as 224 x 224. In short, we can say that it has convolution layer, maxpool layer, convolutional blocks, identity blocks, average pool layer and fully connected layer.

Like every ResNet architecture this also performs the initial convolution and maxpooling using 7×7 and 3×3 kernel sizes respectively. For deeper networks like ResNet50, ResNet152, etc, bottleneck design is used. For each residual function F, 3 layers are stacked one over the other. Here we use the same as that of deeper networks. The three layers are 1×1, 3×3, 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions. Also, a convolutional block with 3 convolutional layers where first 2 layers are followed by batch normalization and activation function (ReLU). And the last layer is followed by an add function to add the inputted value of the block along with the block updated value and an activation function (ReLU).

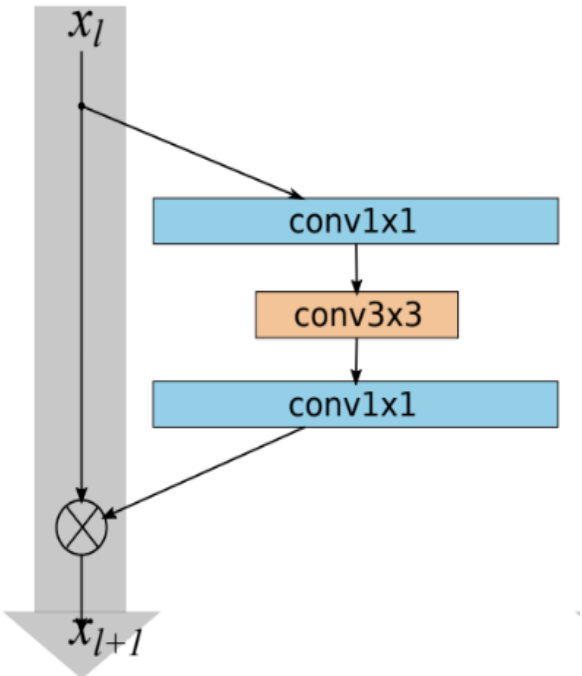


Fig 3.8: Bottleneck

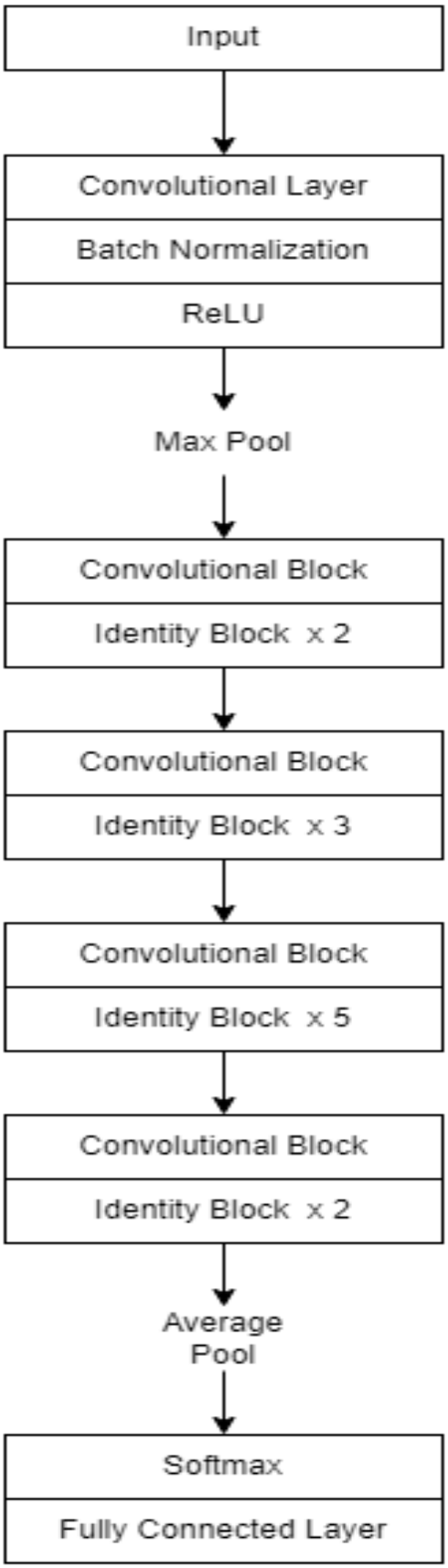


Fig 3.9: Block diagram of architecture

Skip connection

In ResNet architecture, a “shortcut” or a “skip connection” alleviate the issue of vanishing gradient by setting up an alternate shortcut for the gradient to pass through. This ensures that the higher layers of the model do not perform any worse than the lower layers. In other words, the skip connections add the outputs from previous layers to the outputs of stacked layers, making it possible to train much deeper networks than previously possible. The other important feature is its bottleneck residual block i.e.; 3 layers are stacked to form a residual block whose kernel size gives a bottleneck like structure. The three layers are 1×1, 3×3, 1×1 convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions.

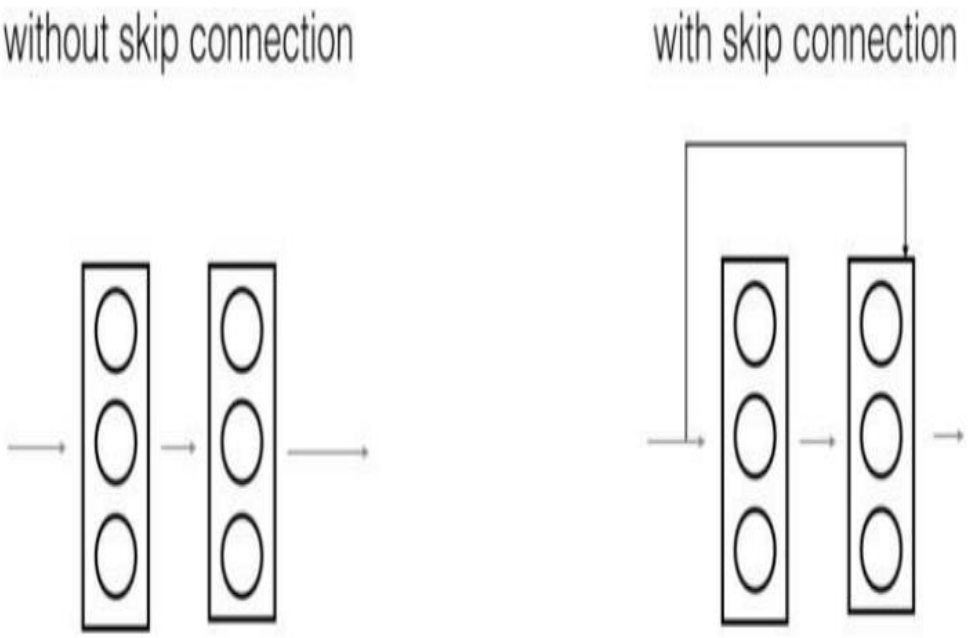


Fig 3.10: Skip Connection

Features and characteristics of Resnet

- A residual network is formed by stacking several residual blocks together.
- ResNet uses Batch Normalization at its core. The Batch Normalization adjusts the input layer to increase the performance of the network.

- ResNet makes use of the Skip Connection, which helps to protect the network from vanishing gradient problem.
- Deep Residual Network uses bottleneck residual block design to increase the performance of the network.

Diagrams and Details of Each Layer

1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size 7 x 7, having 64 kernels. By sliding (stride size = 2) the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (7 x 7). The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

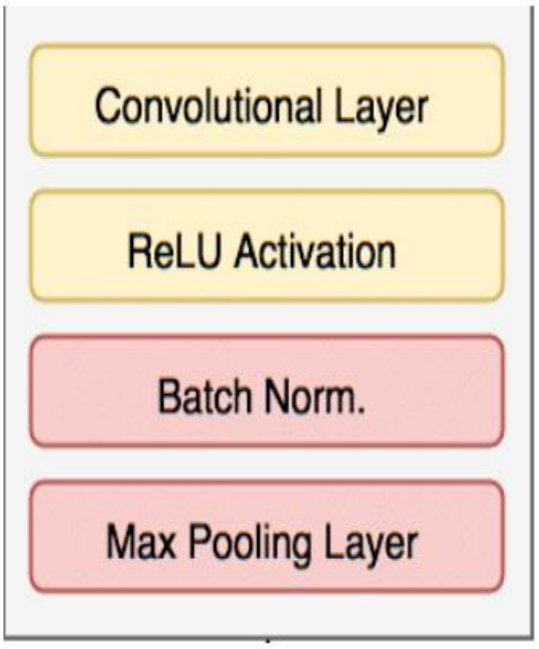


Fig 3.11: Convolutional Layer

2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling layer usually serves as a bridge in-between the Convolutional layers and also between Convolutional layer and the FC layer. In this architecture we use max pooling after convolutional layer and average pooling before FC layer.

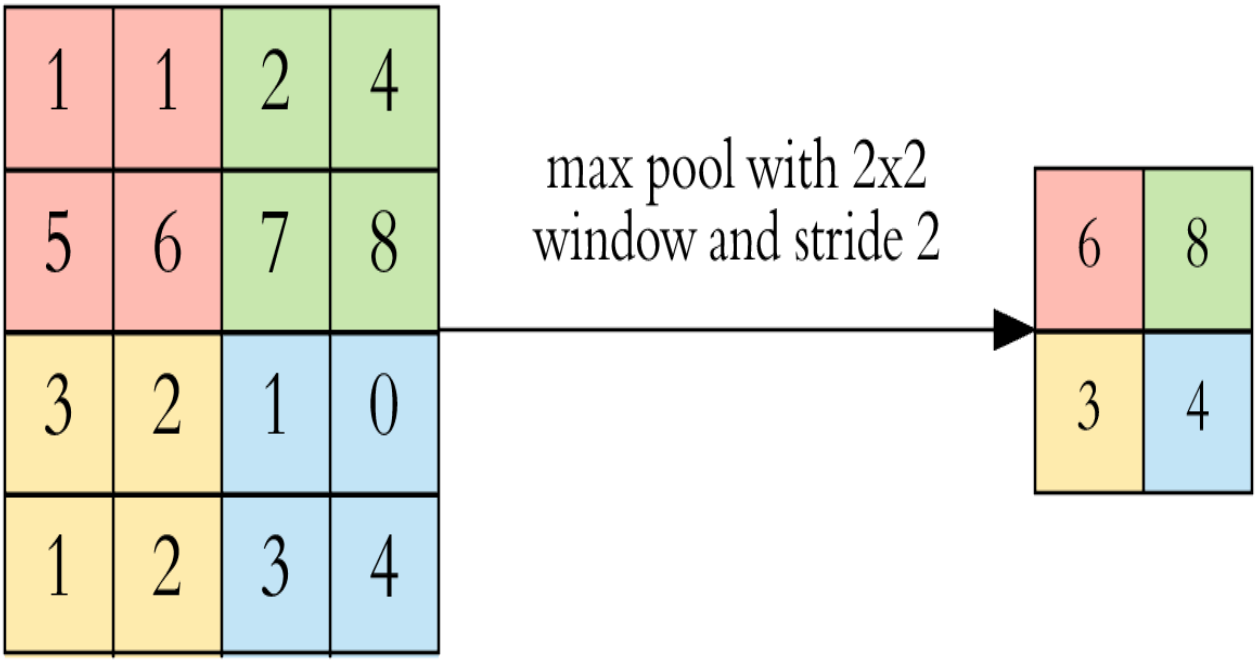


Fig 3.12: Process in max pooling layer

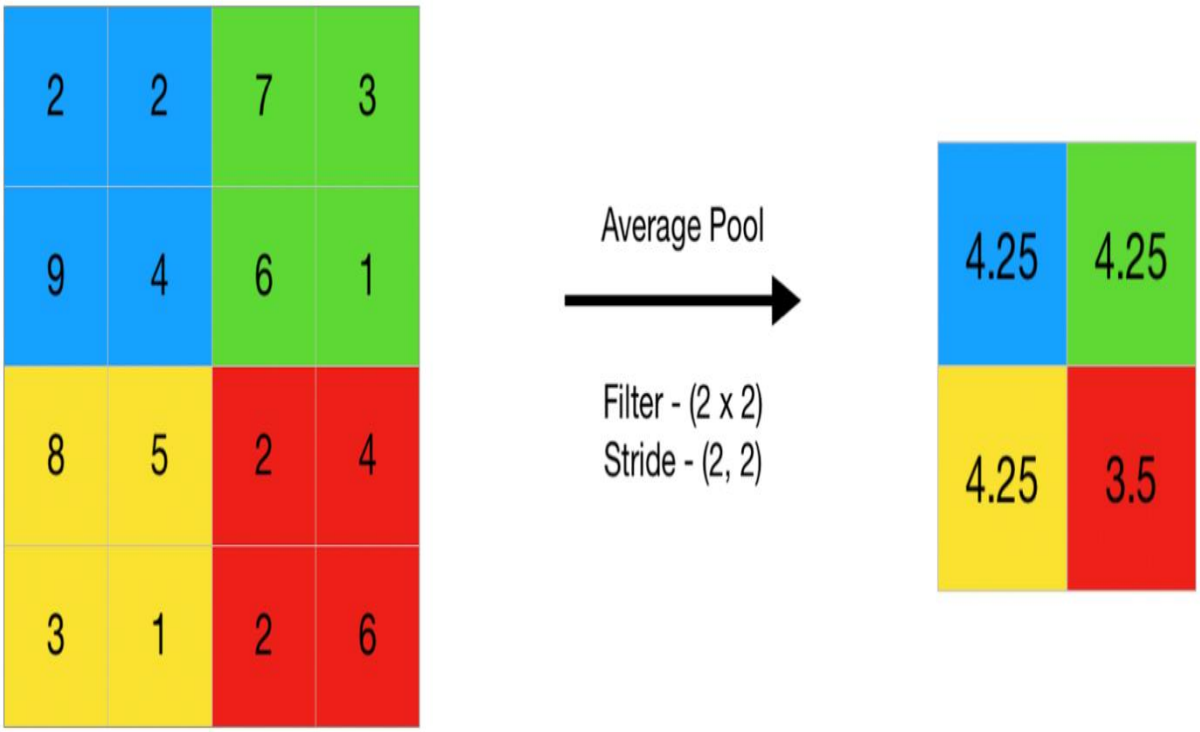


Fig 3.13: Process in average pooling layer

3. Convolutional Block

We can use this type of block when the input and output dimensions don't match up. For each first two convolutional layers followed by batch normalization and activation function (ReLU). The last convolutional layer passes the value to a add function to add the inputted value and updated value along with an activation function. The difference with the identity block and convolutional block is that there is no CONV layer in the shortcut path of convolutional block. Here the convolutional block is also a bottleneck block along with our shortcut CONV layer. In this architecture the first convolutional block is having 64, 64 and 256 number of filters and its repeating 3 times. Second is having 128, 128 and 512 number of filters and its repeating 4 times. The third is having 256, 256 and 1024 number of filters and its repeating 6 times. The fourth is having 512, 512 and 2048 number of filters and its repeating 3 times.

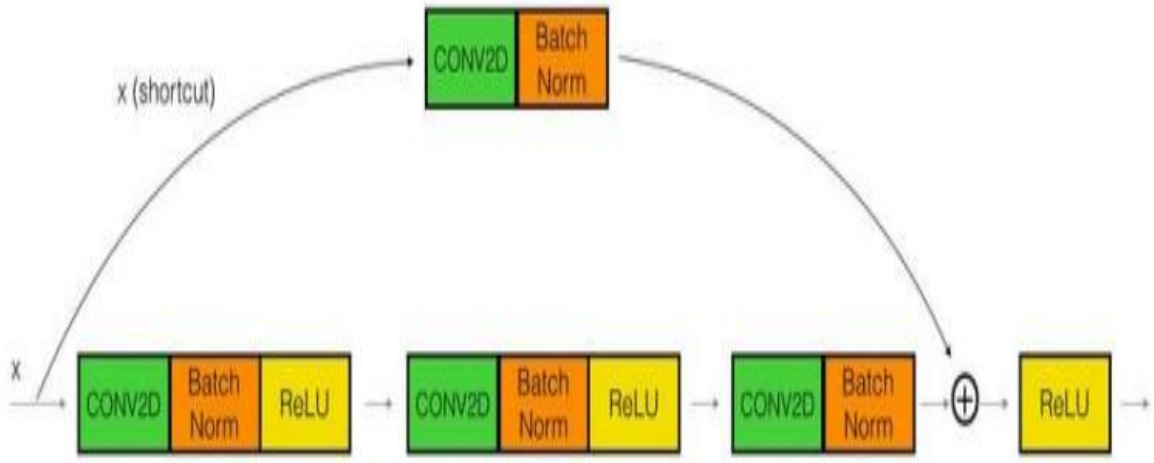


Fig 3.14: Convolutional block

4. Identity Block

The identity block is also called the residual block. It is the standard block used in ResNets and corresponds to the case where the input activation has the same dimension as the output activation. Here in this architecture each identity block is a bottleneck block. For each first two convolutional layers followed by batch normalization and activation function (ReLU). The last convolutional layer passes the value to a add function to add the inputted value and updated value along with an activation function. Here in this architecture the first identity block is having 64, 64 and 256 number of filters and it is repeating 2 times. Second is having 128, 128 and 512 number of filters and it is repeating 3 times. The third is having 256, 256 and 1024 number of filters and it is repeating 5 times. The fourth is having 512, 512 and 2048 number of filters and it is repeating 2 times.

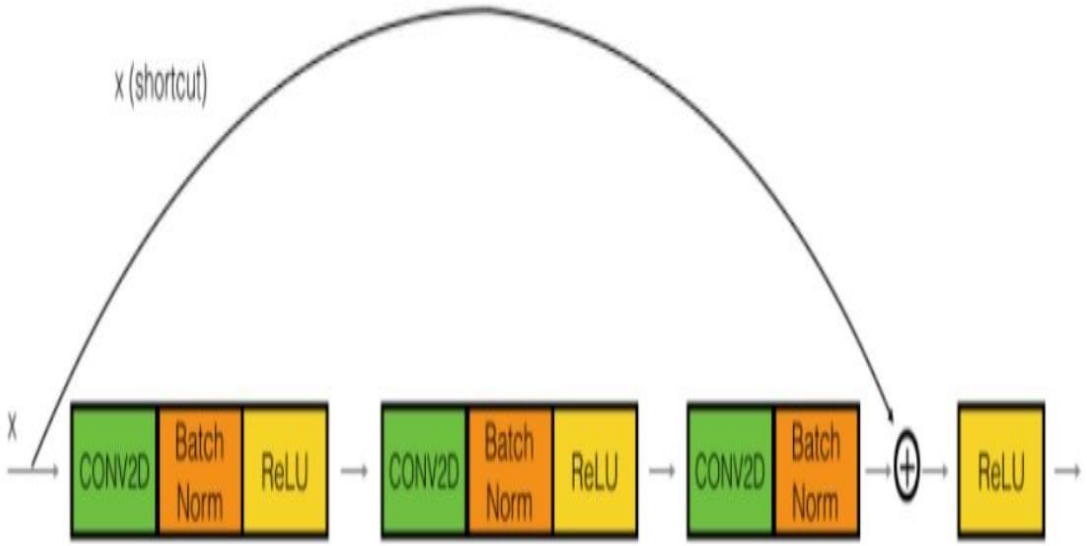


Fig 3.15: Identity block

5. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer. In this, the input image from the previous layers is flattened and fed to the FC layer. The flattened vector then undergoes few mathematical functions operations. In this stage, the classification process begins to take place. Fully connected layer always use softmax activation function in this architecture.

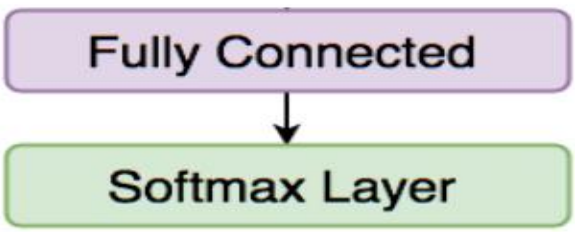


Fig 3.16: FC Layer

Activation Functions

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred for a multi-class classification, generally softmax is used. Before learning each layer, there are two parameters which are important in the working of convolutional neural network layers, stride and padding. Here we use ReLU activation function for layers and for FC layer we use softmax.

a. Stride

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and soon. Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

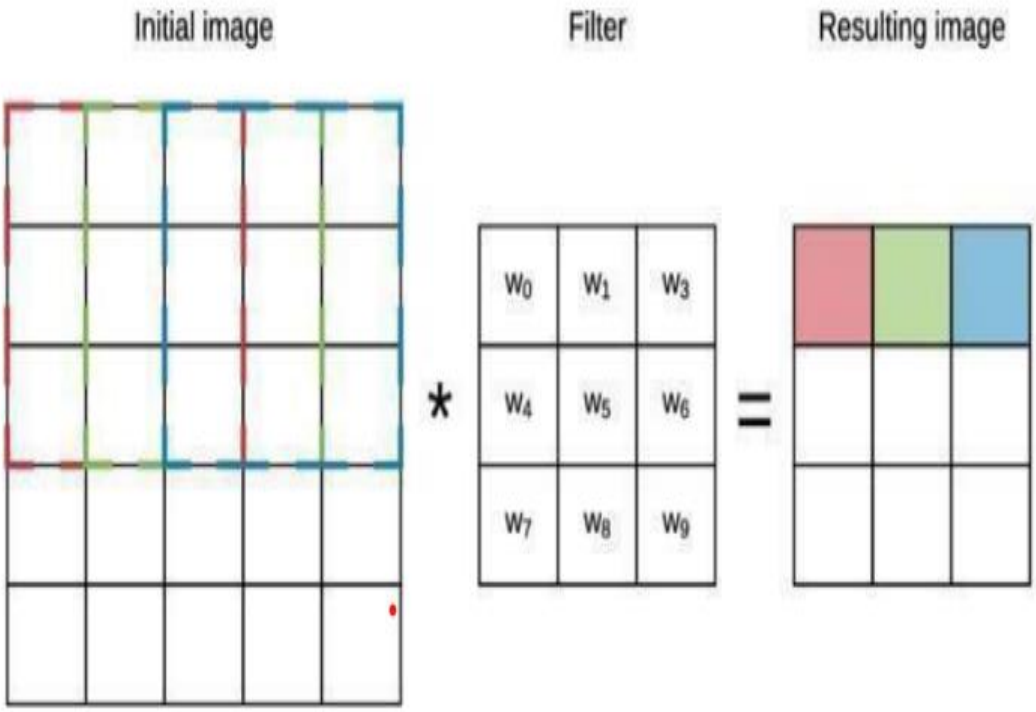


Fig 3.17: Convolution layer working with a stride of 1.

Imagine a convolutional neural network is taking an image and analysing the content. If the filter size is 3x3 pixels, the contained nine pixels will be converted down to 1 pixel in the output layer. Naturally, as the stride, or movement, is increased, the resulting output will be smaller. Stride is a parameter that works in conjunction with padding, the feature that adds blank, or empty pixels to the frame of the image to allow for a minimized reduction of size in the output layer. Roughly, it is a way of increasing the size of an image, to counter act the fact that stride reduces the size. Padding and stride are the foundational parameters of any convolutional neural network.

b. Padding

Sometimes filter does not perfectly fit the input image. we have two options to overcome this:

- Pad the picture with zeros (zero-padding) so that it fits.
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Padding is a term relevant to convolutional neural networks as it refers to the number of pixels added to an image when it is being processed by the kernel of a CNN. For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero. If, however, the zero padding is set to one, there will be a one-pixel border added to the image with a pixel value of zero.

Padding works by extending the area of which a convolutional neural network processes an image. The kernel is the neural networks filter which moves across the image, scanning each pixel and converting the data into a smaller, or sometimes larger, format. In order to assist the kernel with processing the image, padding is added to the frame of the image to allow for more space for the kernel to cover the image. Adding padding to an image processed by a CNN allows for more accurate analysis of images.

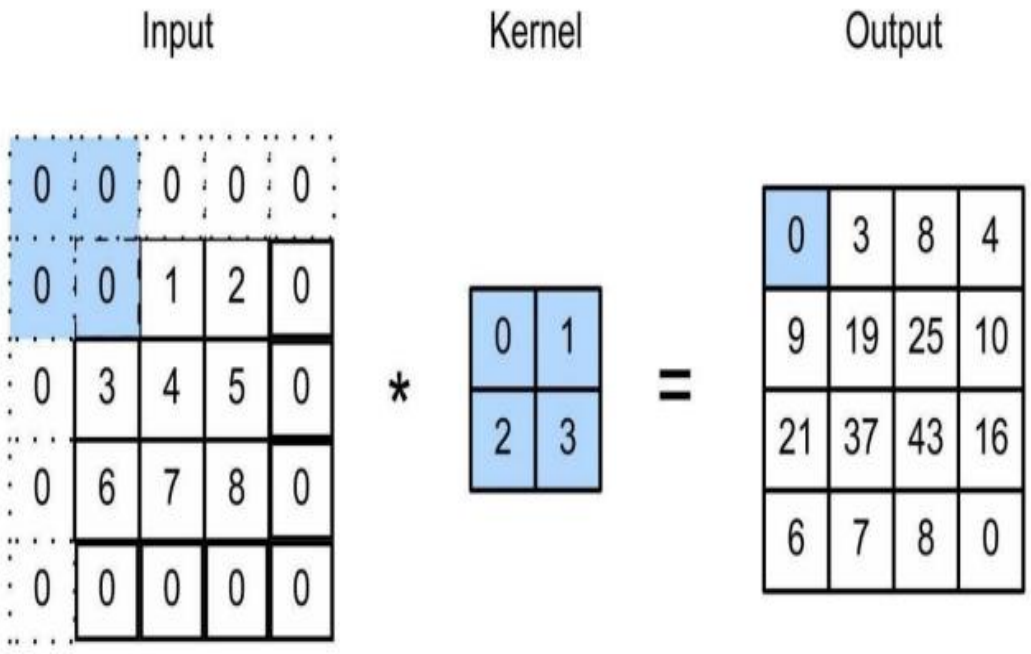


Fig 3.18: padding

The key building block in a convolutional neural network is the convolutional layer. We can visualize a convolutional layer as many small square templates, called convolutional kernels, which slide over the image and look for patterns. Where that part of the image matches the kernel's pattern, the kernel returns a large positive value, and when there is no match, the kernel returns zero or a smaller value.

Convolution layers used trainable kernels or filters to perform convolution operations, sometimes including an optional trainable bias for each kernel. These convolution operations involved moving the kernels over the input in steps called strides. Generally, the larger the stride was, the more spaces the kernels skipped between each convolution. This led to less overall convolutions and more miniature output size. For each placement of a given kernel, a multiplication operation was performed between the input section and the kernel, with the bias summed to the result. This produced a feature map containing the convolved result. The feature maps were typically passed through an activation function to provide input for the subsequent layer.

Size of the feature map = [(input_size − kernel_size + 2 × padding) / stride] + 1.

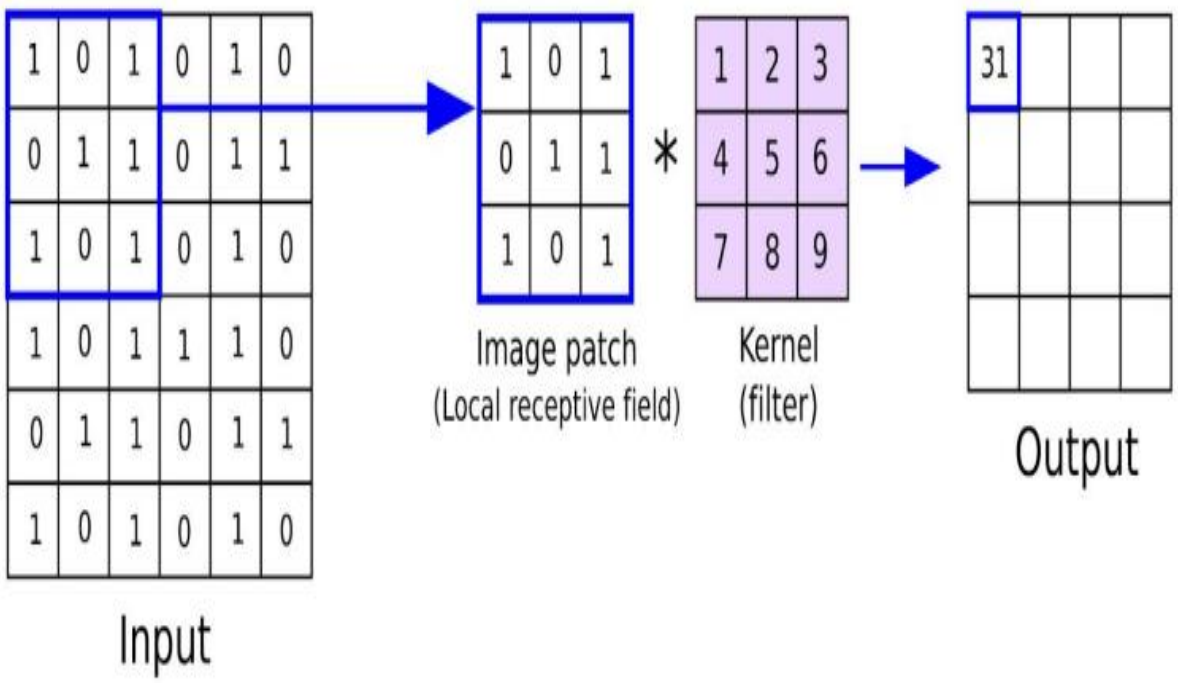


Fig 3.19: Working of convolutional layer.

- ReLU

The rectified linear activation function or ReLU for short is a linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

$$f(x) = max(0, x)$$

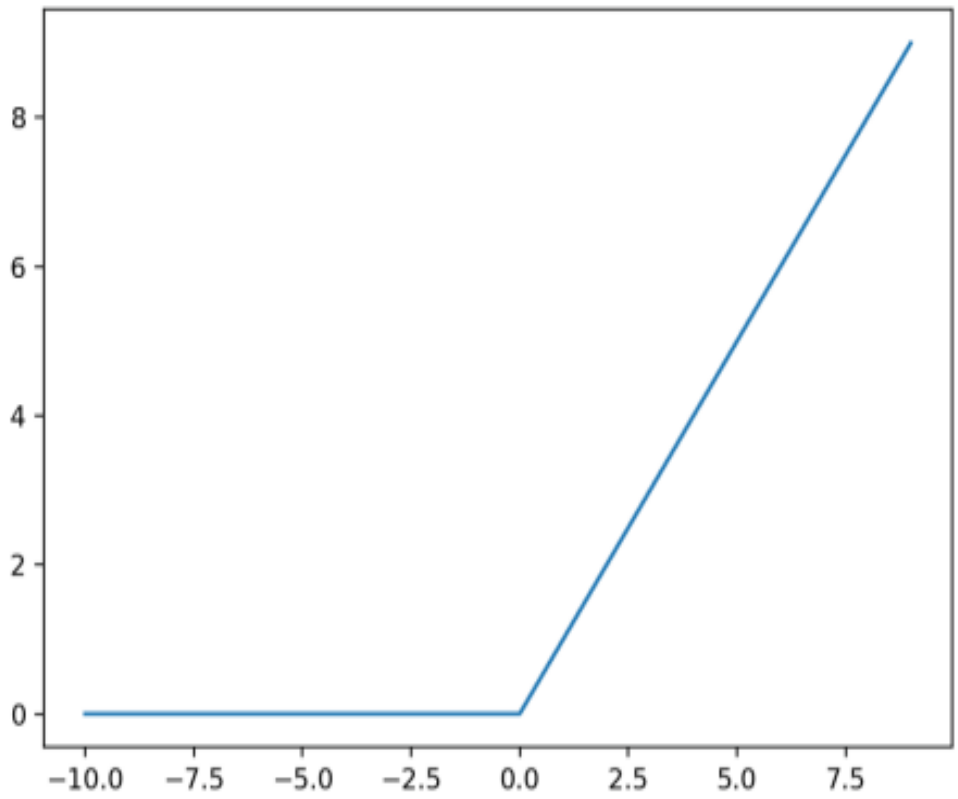


Fig 3.20: ReLU Graph

- Softmax

Softmax is often used as the activation for the last layer of a classification network because the result could be interpreted as a probability distribution.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

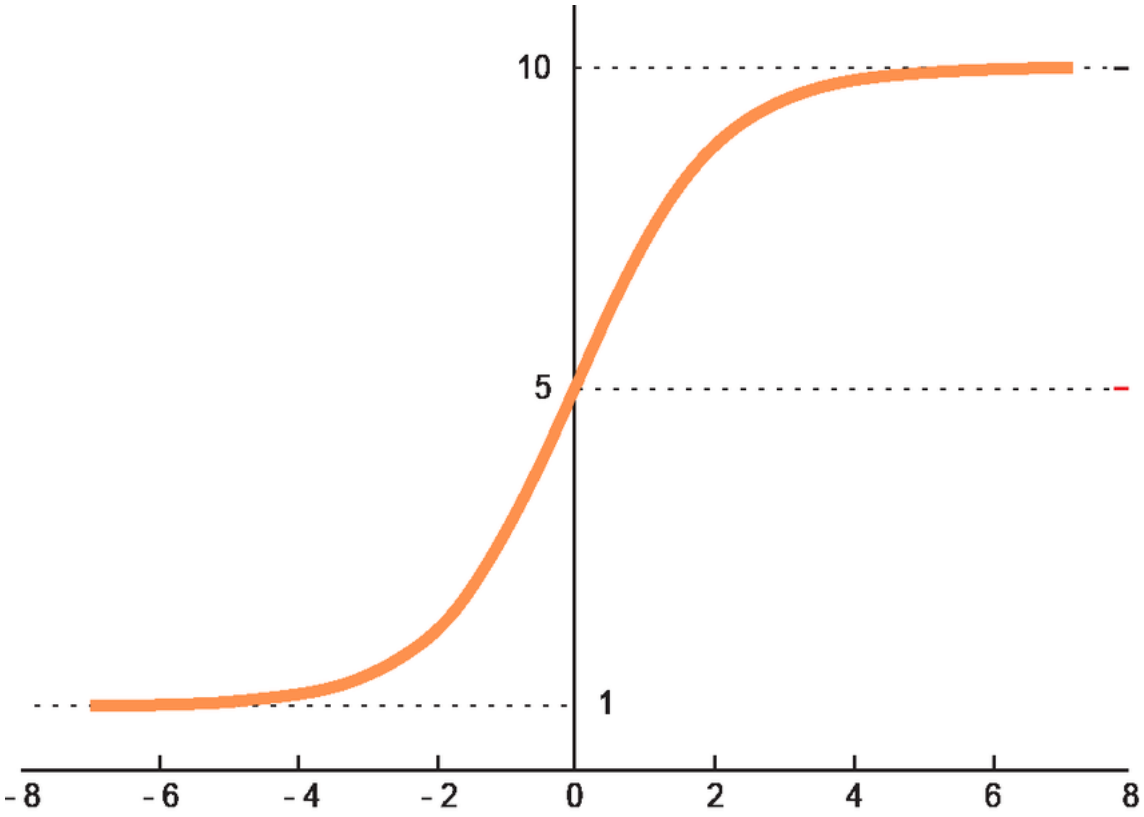


Fig 3.21: Softmax Graph

Dimension table of the architecture

Layer	Kernel size	Filters	Padding	Stride	Output size	Activation function
Input	-	-	-	-	224 x 224	-
Conv1	7 x 7	64	3 x 3	2	112 x 112	ReLU
Maxpool	3 x 3	-	1 x 1	2	-	-
Conv2_x	1 x 1 3 x 3 1 x 1 } x 3	64 64 256 } x 3	1 x 1	1	56 x 56	ReLU
Conv3_x	1 x 1 3 x 3 1 x 1 } x 4	128 128 512 } x 4	1 x 1	1	28 x 28	ReLU
Conv4_x	1 x 1 3 x 3 1 x 1 } x 6	256 256 1024 } x 6	1 x 1	1	14 x 14	ReLU
Conv5_x	1 x 1 3 x 3 1 x 1 } x 3	512 512 2048 } x 3	1 x 1	1	7 x 7	ReLU
Average pool	-	-	-	2	-	-
fc	-	-	-	-	5	Softmax

Table 2.1: Dimension table of the architecture

3.5. Project pipeline

- Development pipeline

Project pipeline explains the project flow. The below figure explains the project flow during the developing stage of the project. Dataset is taken and audio is pre-processed. Here pre-processing steps are trimming the audio by ignoring the 1 second duration audio from beginning. The audios are trimmed in an interval of 10 second until the end of that audio. Next step is to convert each pre-processed audio to mel-spectrogram. After this step, the mel-spectrogram is treated as the dataset that is using to build the model. The new converted dataset is then split into two. One for training the model and remaining for testing the model. 70% of mel-spectrogram is used for training and 30% used for testing. Using the split data the model is trained and tested several times. The models are saved each time it trains. The model having more accuracy and efficiency is taken for deployment.

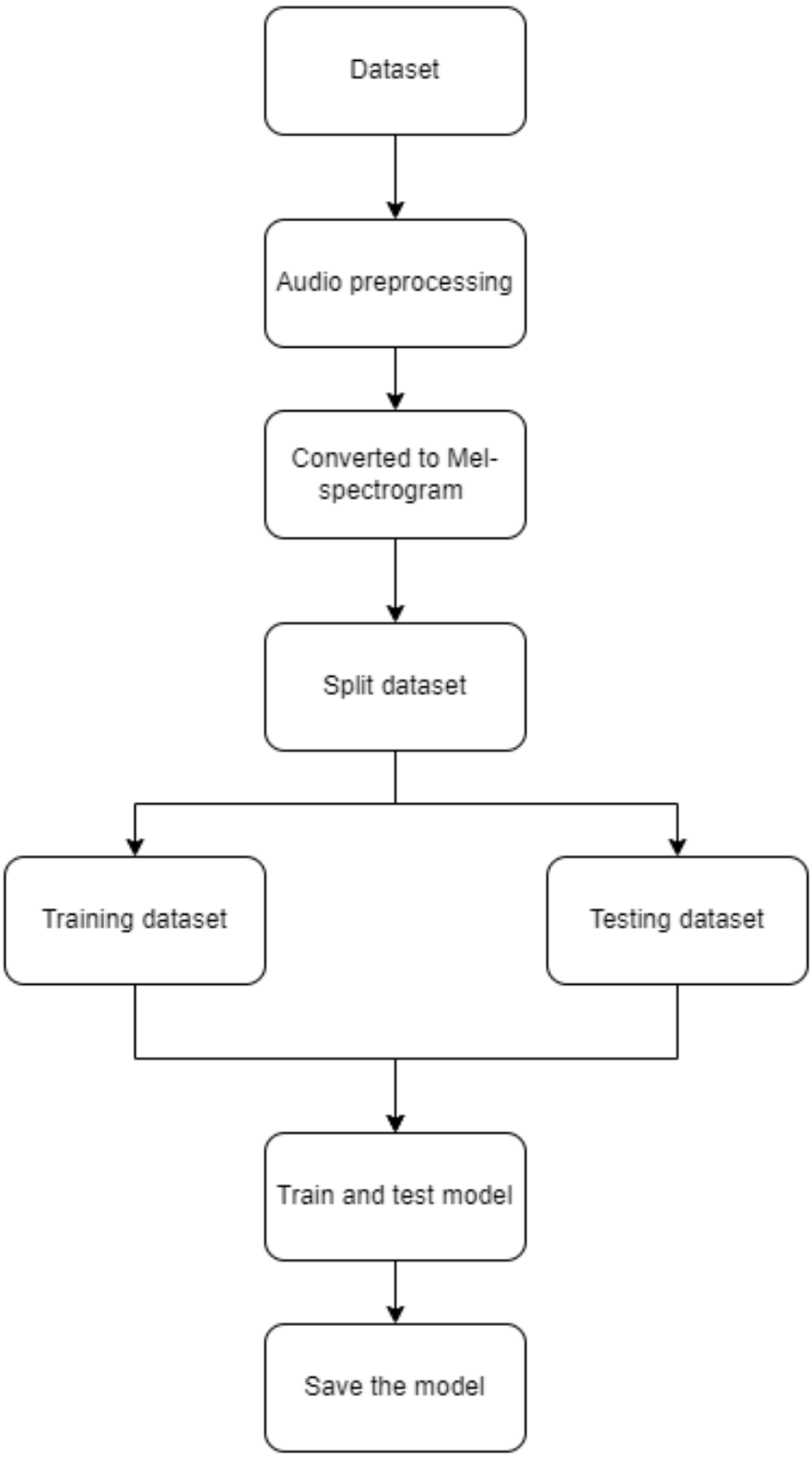


Fig 3.22: Development pipeline

- Deployment pipeline

The below figure explains the deploying stages of project. Here we are building a user interface for the system. Writing necessary functions to pre-process the audio and a function to convert the audio to mel-spectrogram. The model saved in development phase is loaded to predict the pre-processed data. A seperate user interface page will display the result along with pre-processed and inputted data.

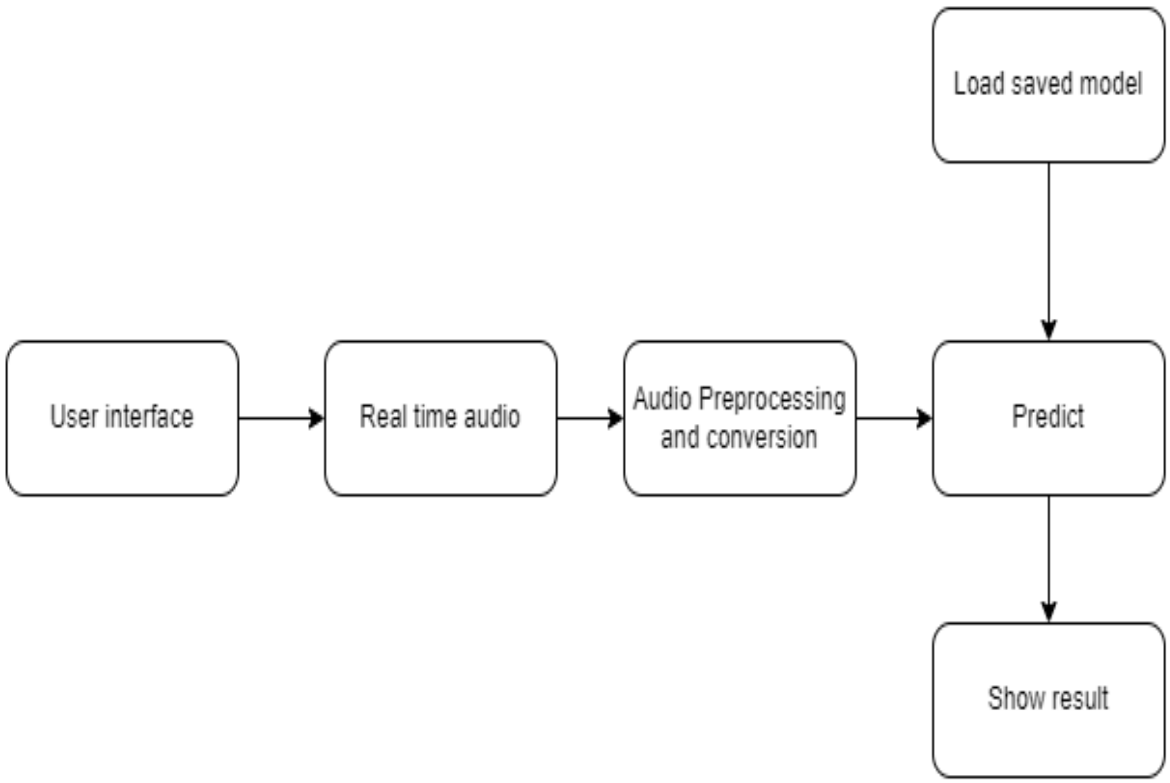


Fig 3.23: Development pipeline

3.6. Feasibility Analysis

A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing system or proposed system, opportunities and threats present in the natural environment, the resources required to carry through, and ultimately the prospects for success. Evaluated the feasibility of the system in terms of the following categories:

• Technical Feasibility

• Economic Feasibility

• Operational Feasibility

3.6.1. Technical Feasibility

Proposed system is technically feasible since all the required tools are easily available. Technical issues involved are the necessary technology existence, technical guarantees of accuracy, reliability, ease of access, data security, and aspects of future expansion. The application is technically feasible because all the technical resources required for the development and working of the application is easily available and reliable. The project is implemented in Python. Since Python supports a various libraries and packages that make the project development easier, the project was technically feasible. The codes are written in Kaggle, therefore all the libraries will be available, no need to install. These requirements are easily available, reliable, and will make the system more time saving and require less manpower.

3.6.2. Economic Feasibility

In our proposed system "Bird Sound Classification", the development cost of the application is optimum. The system requires only a computer for working. The code is working on Google Colab and Kaggle notebook. The Colab can consumes an amount of internet. The development of the system will not need a huge amount of money. It will be economically feasible. Kaggle provides immense computational power through their servers which make it more suitable than colab.

3.6.3. Operational Feasibility

Operational feasibility assesses the extent to which the required system performs a series of steps to solve business problems and user requirements. Operational feasibility is mainly concerned with issues like whether the system will be used if it is developed and implemented. The developed system is completely driven and user friendly. Since the code is written on Google Colab and Kaggle no need for worrying about importing or installing the libraries required. There is no need of skill for a new

user to open this application and use it. The interface contains only a file upload option and a submit button for home page and output display page contains the result, inputted audio, pre-processed audio, converted mel-spectrogram and model input. Users also need to be aware of the application initially. Then they can use it easily. So, it is feasible. But sometimes Colab has a GPU issues. If we use Kaggle notebook, they provide a higher computational power than Colab.

3.7. System Environment

System environment specifies the hardware and software configuration of the new system. Regardless of how the requirement phase proceeds, it ultimately ends with the software requirement specification. A good SRS contains all the system requirements to a level of detail sufficient to enable designers to design a system that satisfies those requirements. The system specified in the SRS will assist the potential users to determine if the system meets their needs or how the system must be modified to meet their needs.

3.7.1. Software Environment

Various software used for the development of this application are the following:

1. Python

Python is a high-level programming language that lets developers work quickly and integrate systems more efficiently. This model is developed by using many of the Python libraries and packages such as:

a. NumPy:

NumPy is a Python library used for working with arrays. NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. In this application, its used for handling arrays.

b. Matplotlib:

Matplotlib is a cross-platform, data visualization and graphical plotting library

for Python. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. In this application, its used for plotting the graph.

c. Tensorflow:

TensorFlow is an open-source library developed by Google primarily for deep learning applications. In this application, its used for creating and handling the model.

d. Keras:

Keras is a powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code. In this application, its used for creating and handling the model.

e. Scikit-learn:

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. In this application it is used to plot confusion matrix.

f. OS:

The OS module in Python provides functions for interacting with the operating system.OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. In this application, its used for saving the model.

2. Google Colab

Colab (short for Google Colaboratory) is a cloud-based platform that allows users to write, run, and share code in a Jupyter notebook environment. Colab is a free service offered by Google, and it provides access to powerful computing resources,

including GPUs and TPUs, which can be used to train deep learning models. Colab notebooks are hosted in the cloud and can be accessed from anywhere with an internet connection. We can write and execute code in Python. The notebooks can be saved to Google Drive, which makes them easy to access and share with others. Colab also supports popular data science libraries, such as NumPy, Pandas, Matplotlib, and TensorFlow, making it easy to analyze and visualize data. Colab supports many python libraries which can be easily loaded in the colab notebook. One of the key benefits of using Colab is the availability of a powerful GPU, which can significantly speed up the training of deep learning models.

3. Kaggle

A Kaggle Notebook is a free jupyter notebook server that can be GPU integrated. Just like Google Colab notebooks, it allows you to perform machine learning operations on cloud computers instead of doing it on your own computer. Each time you create a Kaggle Notebook, you can edit and run its content in the browser. There is no need to set up your own jupyter notebook environment, just enter Kaggle, create a notebook and start using it on the browser. Kaggle provides immense computational power through their servers which make it more suitable than colab. Here we use Kaggle to build the model.

4. Visual Studio Code

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE. Here we use vs code to create user interface.

5. HTML, CSS and Bootstrap

Hyper Text Markup Language is used for creating web pages.HTML describes the structure of the web page. Here, the user interface of my project is done using HTML. Cascading Style Sheet is used with HTML to style the web pages. Bootstrap is

a free, open source front-end development framework for the creation of websites and web apps. We have used it to make our user interface more interactive.

6. Github

Git is an open-source version control system that was started by Linus Torvalds. Git is similar to other version control systems Subversion, CVS, and Mercurial to name a few. Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute. Git is the preferred version control system of most developers, since it has multiple advantages over the other systems available. It stores file changes more efficiently and ensures file integrity better. The social networking aspect of GitHub is probably its most powerful feature, allowing projects to grow more than just about any of the other features offered. Project revisions can be discussed publicly, so a mass of experts can contribute knowledge and collaborate to advance a project forward.

3.7.2. Hardware Environment

Selection of hardware configuration is very important task related to the software development. The hardware configuration of project done system is:

Processor : Intel Core i5 preferred

Memory : 8 GB RAM or greater

Disk space : 40 GB or good internet connectivity

GPU : Nvidia K80 (Kaggle)

4. SYSTEM DESIGN

4.1 Model Building

Model building in deep learning refers to the process of designing and developing a neural network architecture that can accurately perform a specific task. Model Building involves Model Planning, Training and Testing.

4.1.1 Implementation Code

Resnet50 architecture is used to build the model here is a pretrained model. Model is created using built-in function of python libraries(keras).

```
model = models.Sequential()
model.add(tf.keras.applications.ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(224, 224, 3),
    pooling=None
))
model.add(layers.Flatten())
model.add(layers.Dense(256,activation="relu"))
model.add(layers.Dense(128,activation="relu"))
model.add(layers.Dense(5,activation="softmax"))
```

Fig 4.1: Implementation code

```
from tensorflow.keras.optimizers import Adam
model.compile(optimizer=Adam(learning_rate=1e-3),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Fig 4.2: Model compile

4.1.2 Model Planning

Model planning is a critical aspect of the deep learning process, which involves carefully designing a neural network architecture that can accurately solve a specific problem. This step mainly focusses on selecting architecture and dataset. The architecture used in the proposed system is Resnet50. Resnet50 is a convolutional neural network architecture. Resnet stand for Residual Network and 50 means it has 50 layers. It handles 5 classes, they are Blue Jay, Black-capped Chickadee, Mallard, Common Tern and American Redstart. The dataset would need to be pre-processed to extract features from the audio files. The audio files are trimmed and converted to mel-spectrogram to extract the features. The mel-spectrogram is taken for model training. The image(mel-spectrogram) dataset is splitted for training and testing. 3213 taken for training and 1374 images for testing. These images are pre-processed to make it suitable for the architecture.

4.1.3 Training

Model training is the process of teaching a deep learning model to recognize patterns in data and make accurate predictions. The goal of model training is to optimize the model's parameters so that it can accurately map input data to output predictions. Dataset split into two – training and testing dataset. Training dataset is the initial dataset we use to teach a deep learning application to recognize pattern. Training dataset is used to train our model, in order to get correct predictions by the model. The model was trained with 100 epochs with default Resnet50 pre-trained weight, where the data used for training is taken from the splitted mel-spectrogram dataset. 3213 images are taken

for training. We validate the result using validation dataset during training or each epoch. In this process, the highest validation accuracy we get for the saved model its 81.64%. The model reaches its highest training accuracy 88.07% at epoch 99.

```
steps_per_epoch = train_ds.n // train_ds.batch_size
validation_steps = valid_ds.n // valid_ds.batch_size

history = model.fit(x=train_ds,
                    validation_data=valid_ds,
                    epochs=100,
                    callbacks=callbacks,
                    class_weight=train_class_weights,
                    steps_per_epoch=steps_per_epoch,
                    validation_steps=validation_steps)
```

Fig 4.3: Model training

```
Epoch 1/100
25/25 [==============================] - ETA: 0s - loss: 5.2784 - accuracy: 0.3608
Epoch 1: val_loss improved from inf to 12474.40820, saving model to bird_resnet50.h5
25/25 [==============================] - 168s 5s/step - loss: 5.2784 - accuracy: 0.3608 - val_loss: 12474.4082 - val_accuracy: 0.2117 - lr: 0.0010
Epoch 2/100
25/25 [==============================] - ETA: 0s - loss: 1.1192 - accuracy: 0.5540
Epoch 2: val_loss improved from 12474.40820 to 215.64963, saving model to bird_resnet50.h5
25/25 [==============================] - 106s 4s/step - loss: 1.1192 - accuracy: 0.5540 - val_loss: 215.6496 - val_accuracy: 0.2094 - lr: 0.0010
Epoch 3/100
25/25 [==============================] - ETA: 0s - loss: 0.9186 - accuracy: 0.6506
Epoch 3: val_loss improved from 215.64963 to 1.73653, saving model to bird_resnet50.h5
25/25 [==============================] - 106s 4s/step - loss: 0.9186 - accuracy: 0.6506 - val_loss: 1.7365 - val_accuracy: 0.2094 - lr: 0.0010
Epoch 4/100
25/25 [==============================] - ETA: 0s - loss: 0.8183 - accuracy: 0.6976
Epoch 4: val_loss improved from 1.73653 to 1.72441, saving model to bird_resnet50.h5
25/25 [==============================] - 89s 4s/step - loss: 0.8183 - accuracy: 0.6976 - val_loss: 1.7244 - val_accuracy: 0.2094 - lr: 0.0010
Epoch 5/100
25/25 [==============================] - ETA: 0s - loss: 0.6776 - accuracy: 0.7569
Epoch 5: val_loss did not improve from 1.72441
25/25 [==============================] - 88s 4s/step - loss: 0.6776 - accuracy: 0.7569 - val_loss: 2.1202 - val_accuracy: 0.2086 - lr: 0.0010
Epoch 6/100
25/25 [==============================] - ETA: 0s - loss: 0.6393 - accuracy: 0.7783
Epoch 6: val_loss did not improve from 1.72441
25/25 [==============================] - 88s 4s/step - loss: 0.6393 - accuracy: 0.7783 - val_loss: 1.7548 - val_accuracy: 0.2039 - lr: 0.0010
Epoch 7/100
25/25 [==============================] - ETA: 0s - loss: 0.5391 - accuracy: 0.8107
Epoch 7: val_loss did not improve from 1.72441

Epoch 7: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
25/25 [==============================] - 103s 4s/step - loss: 0.5391 - accuracy: 0.8107 - val_loss: 1.7917 - val_accuracy: 0.2094 - lr: 0.0010
Epoch 8/100
25/25 [==============================] - ETA: 0s - loss: 0.4811 - accuracy: 0.8308
Epoch 8: val_loss did not improve from 1.72441
25/25 [==============================] - 103s 4s/step - loss: 0.4811 - accuracy: 0.8308 - val_loss: 1.9048 - val_accuracy: 0.2141 - lr: 2.0000e-04
Epoch 9/100
25/25 [==============================] - ETA: 0s - loss: 0.4406 - accuracy: 0.8460
Epoch 9: val_loss did not improve from 1.72441
25/25 [==============================] - 88s 4s/step - loss: 0.4406 - accuracy: 0.8460 - val_loss: 1.9305 - val_accuracy: 0.2109 - lr: 2.0000e-04
Epoch 10/100
25/25 [==============================] - ETA: 0s - loss: 0.3961 - accuracy: 0.8603
Epoch 10: val_loss did not improve from 1.72441

Epoch 10: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
25/25 [==============================] - 102s 4s/step - loss: 0.3961 - accuracy: 0.8603 - val_loss: 1.9792 - val_accuracy: 0.2102 - lr: 2.0000e-04
Epoch 11/100
25/25 [==============================] - ETA: 0s - loss: 0.3672 - accuracy: 0.8768
Epoch 11: val_loss did not improve from 1.72441
```

Fig 4.4: Epoch 1 to 11

```
Epoch 12/100
25/25 [==============================] - ETA: 0s - loss: 0.3511 - accuracy: 0.8733
Epoch 12: val_loss did not improve from 1.72441
25/25 [==============================] - 103s 4s/step - loss: 0.3511 - accuracy: 0.8733 - val_loss: 2.1378 - val_accuracy: 0.2070 - lr: 4.0000e-05
Epoch 13/100
25/25 [==============================] - ETA: 0s - loss: 0.3561 - accuracy: 0.8713
Epoch 13: val_loss did not improve from 1.72441

Epoch 13: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
25/25 [==============================] - 103s 4s/step - loss: 0.3561 - accuracy: 0.8713 - val_loss: 2.1620 - val_accuracy: 0.2047 - lr: 4.0000e-05
Epoch 14/100
25/25 [==============================] - ETA: 0s - loss: 0.3403 - accuracy: 0.8765
Epoch 14: val_loss did not improve from 1.72441
25/25 [==============================] - 88s 4s/step - loss: 0.3403 - accuracy: 0.8765 - val_loss: 2.1916 - val_accuracy: 0.2156 - lr: 8.0000e-06
Epoch 15/100
25/25 [==============================] - ETA: 0s - loss: 0.3390 - accuracy: 0.8817
Epoch 15: val_loss did not improve from 1.72441
25/25 [==============================] - 87s 3s/step - loss: 0.3390 - accuracy: 0.8817 - val_loss: 2.2583 - val_accuracy: 0.2023 - lr: 8.0000e-06
Epoch 16/100
25/25 [==============================] - ETA: 0s - loss: 0.3593 - accuracy: 0.8697
Epoch 16: val_loss did not improve from 1.72441

Epoch 16: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
25/25 [==============================] - 89s 4s/step - loss: 0.3593 - accuracy: 0.8697 - val_loss: 2.2763 - val_accuracy: 0.2086 - lr: 8.0000e-06
Epoch 17/100
25/25 [==============================] - ETA: 0s - loss: 0.3486 - accuracy: 0.8814
Epoch 17: val_loss did not improve from 1.72441
25/25 [==============================] - 103s 4s/step - loss: 0.3486 - accuracy: 0.8814 - val_loss: 2.3224 - val_accuracy: 0.2047 - lr: 1.6000e-06
Epoch 18/100
25/25 [==============================] - ETA: 0s - loss: 0.3389 - accuracy: 0.8830
Epoch 18: val_loss did not improve from 1.72441
25/25 [==============================] - 88s 4s/step - loss: 0.3389 - accuracy: 0.8830 - val_loss: 2.3215 - val_accuracy: 0.2070 - lr: 1.6000e-06
Epoch 19/100
25/25 [==============================] - ETA: 0s - loss: 0.3352 - accuracy: 0.8791
Epoch 19: val_loss did not improve from 1.72441

Epoch 19: ReduceLROnPlateau reducing learning rate to 3.200000264769187e-07.
25/25 [==============================] - 103s 4s/step - loss: 0.3352 - accuracy: 0.8791 - val_loss: 2.3424 - val_accuracy: 0.2047 - lr: 1.6000e-06
Epoch 20/100
25/25 [==============================] - ETA: 0s - loss: 0.3411 - accuracy: 0.8697
Epoch 20: val_loss did not improve from 1.72441
25/25 [==============================] - 104s 4s/step - loss: 0.3411 - accuracy: 0.8697 - val_loss: 2.3399 - val_accuracy: 0.2062 - lr: 3.2000e-07
Epoch 21/100
25/25 [==============================] - ETA: 0s - loss: 0.3226 - accuracy: 0.8875
Epoch 21: val_loss did not improve from 1.72441
25/25 [==============================] - 103s 4s/step - loss: 0.3226 - accuracy: 0.8875 - val_loss: 2.3466 - val_accuracy: 0.2031 - lr: 3.2000e-07
```

Fig 4.5: Epoch 12 to 25

```
Epoch 22/100
25/25 [==============================] - ETA: 0s - loss: 0.3320 - accuracy: 0.8830
Epoch 22: val_loss did not improve from 1.72441

Epoch 22: ReduceLROnPlateau reducing learning rate to 6.400000529538374e-08.
25/25 [==============================] - 104s 4s/step - loss: 0.3320 - accuracy: 0.8830 - val_loss: 2.3234 - val_accuracy: 0.2125 - lr: 3.2000e-07
Epoch 23/100
25/25 [==============================] - ETA: 0s - loss: 0.3387 - accuracy: 0.8742
Epoch 23: val_loss did not improve from 1.72441
25/25 [==============================] - 104s 4s/step - loss: 0.3387 - accuracy: 0.8742 - val_loss: 2.3428 - val_accuracy: 0.2047 - lr: 6.4000e-08
Epoch 24/100
25/25 [==============================] - ETA: 0s - loss: 0.3287 - accuracy: 0.8817
Epoch 24: val_loss did not improve from 1.72441
25/25 [==============================] - 103s 4s/step - loss: 0.3287 - accuracy: 0.8817 - val_loss: 2.3426 - val_accuracy: 0.2078 - lr: 6.4000e-08
Epoch 25/100
25/25 [==============================] - ETA: 0s - loss: 0.3131 - accuracy: 0.8921
Epoch 25: val_loss did not improve from 1.72441

Epoch 25: ReduceLROnPlateau reducing learning rate to 1.2800001059076749e-08.
25/25 [==============================] - 87s 3s/step - loss: 0.3131 - accuracy: 0.8921 - val_loss: 2.3242 - val_accuracy: 0.2102 - lr: 6.4000e-08
Epoch 26/100
25/25 [==============================] - ETA: 0s - loss: 0.3502 - accuracy: 0.8768
Epoch 26: val_loss did not improve from 1.72441
25/25 [==============================] - 105s 4s/step - loss: 0.3502 - accuracy: 0.8768 - val_loss: 2.3128 - val_accuracy: 0.2109 - lr: 1.2800e-08
Epoch 27/100
25/25 [==============================] - ETA: 0s - loss: 0.3402 - accuracy: 0.8791
Epoch 27: val_loss did not improve from 1.72441
25/25 [==============================] - 104s 4s/step - loss: 0.3402 - accuracy: 0.8791 - val_loss: 2.2980 - val_accuracy: 0.2117 - lr: 1.2800e-08
Epoch 28/100
25/25 [==============================] - ETA: 0s - loss: 0.3396 - accuracy: 0.8836
Epoch 28: val_loss did not improve from 1.72441

Epoch 28: ReduceLROnPlateau reducing learning rate to 2.5600002118153498e-09.
25/25 [==============================] - 87s 4s/step - loss: 0.3396 - accuracy: 0.8836 - val_loss: 2.3009 - val_accuracy: 0.2086 - lr: 1.2800e-08
Epoch 29/100
25/25 [==============================] - ETA: 0s - loss: 0.3389 - accuracy: 0.8781
Epoch 29: val_loss did not improve from 1.72441
25/25 [==============================] - 104s 4s/step - loss: 0.3389 - accuracy: 0.8781 - val_loss: 2.2802 - val_accuracy: 0.2086 - lr: 2.5600e-09
Epoch 30/100
25/25 [==============================] - ETA: 0s - loss: 0.3640 - accuracy: 0.8690
Epoch 30: val_loss did not improve from 1.72441
25/25 [==============================] - 103s 4s/step - loss: 0.3640 - accuracy: 0.8690 - val_loss: 2.2395 - val_accuracy: 0.2141 - lr: 2.5600e-09
Epoch 31/100
25/25 [==============================] - ETA: 0s - loss: 0.3409 - accuracy: 0.8778
Epoch 31: val_loss did not improve from 1.72441
```

Fig 4.6: Epoch 26 to 31

```
Epoch 31: ReduceLROnPlateau reducing learning rate to 5.1200004236307e-10.
25/25 [==============================] - 104s 4s/step - loss: 0.3409 - accuracy: 0.8778 - val_loss: 2.2331 - val_accuracy: 0.2102 - lr: 2.5600e-09
Epoch 32/100
25/25 [==============================] - ETA: 0s - loss: 0.3348 - accuracy: 0.8814
Epoch 32: val_loss did not improve from 1.72441
25/25 [==============================] - 104s 4s/step - loss: 0.3348 - accuracy: 0.8814 - val_loss: 2.1769 - val_accuracy: 0.2086 - lr: 5.1200e-10
Epoch 33/100
25/25 [==============================] - ETA: 0s - loss: 0.3170 - accuracy: 0.8856
Epoch 33: val_loss did not improve from 1.72441
25/25 [==============================] - 103s 4s/step - loss: 0.3170 - accuracy: 0.8856 - val_loss: 2.1501 - val_accuracy: 0.2055 - lr: 5.1200e-10
Epoch 34/100
25/25 [==============================] - ETA: 0s - loss: 0.3295 - accuracy: 0.8814
Epoch 34: val_loss did not improve from 1.72441

Epoch 34: ReduceLROnPlateau reducing learning rate to 1.0240001069306004e-10.
25/25 [==============================] - 104s 4s/step - loss: 0.3295 - accuracy: 0.8814 - val_loss: 2.1032 - val_accuracy: 0.2062 - lr: 5.1200e-10
Epoch 35/100
25/25 [==============================] - ETA: 0s - loss: 0.3427 - accuracy: 0.8794
Epoch 35: val_loss did not improve from 1.72441
25/25 [==============================] - 87s 3s/step - loss: 0.3427 - accuracy: 0.8794 - val_loss: 2.0071 - val_accuracy: 0.2188 - lr: 1.0240e-10
Epoch 36/100
25/25 [==============================] - ETA: 0s - loss: 0.3311 - accuracy: 0.8788
Epoch 36: val_loss did not improve from 1.72441
25/25 [==============================] - 104s 4s/step - loss: 0.3311 - accuracy: 0.8788 - val_loss: 1.8951 - val_accuracy: 0.2320 - lr: 1.0240e-10
Epoch 37/100
25/25 [==============================] - ETA: 0s - loss: 0.3370 - accuracy: 0.8814
Epoch 37: val_loss did not improve from 1.72441

Epoch 37: ReduceLROnPlateau reducing learning rate to 2.0480002416167767e-11.
25/25 [==============================] - 104s 4s/step - loss: 0.3370 - accuracy: 0.8814 - val_loss: 1.7566 - val_accuracy: 0.2773 - lr: 1.0240e-10
Epoch 38/100
25/25 [==============================] - ETA: 0s - loss: 0.3541 - accuracy: 0.8720
Epoch 38: val_loss improved from 1.72441 to 1.60409, saving model to bird_resnet50.h5
25/25 [==============================] - 88s 4s/step - loss: 0.3541 - accuracy: 0.8720 - val_loss: 1.6041 - val_accuracy: 0.3539 - lr: 2.0480e-11
Epoch 39/100
25/25 [==============================] - ETA: 0s - loss: 0.3215 - accuracy: 0.8856
Epoch 39: val_loss improved from 1.60409 to 1.42757, saving model to bird_resnet50.h5
25/25 [==============================] - 106s 4s/step - loss: 0.3215 - accuracy: 0.8856 - val_loss: 1.4276 - val_accuracy: 0.4289 - lr: 2.0480e-11
Epoch 40/100
25/25 [==============================] - ETA: 0s - loss: 0.3535 - accuracy: 0.8775
Epoch 40: val_loss improved from 1.42757 to 1.27522, saving model to bird_resnet50.h5
25/25 [==============================] - 90s 4s/step - loss: 0.3535 - accuracy: 0.8775 - val_loss: 1.2752 - val_accuracy: 0.5031 - lr: 2.0480e-11
Epoch 41/100
25/25 [==============================] - ETA: 0s - loss: 0.3533 - accuracy: 0.8771
Epoch 41: val_loss improved from 1.27522 to 1.19147, saving model to bird_resnet50.h5
25/25 [==============================] - 107s 4s/step - loss: 0.3533 - accuracy: 0.8771 - val_loss: 1.1915 - val_accuracy: 0.5586 - lr: 2.0480e-11
```

Fig 4.7: Epoch 32 to 41

```
Epoch 42/100
25/25 [==============================] - ETA: 0s - loss: 0.3365 - accuracy: 0.8784
Epoch 42: val_loss improved from 1.19147 to 1.04219, saving model to bird_resnet50.h5
25/25 [==============================] - 107s 4s/step - loss: 0.3365 - accuracy: 0.8784 - val_loss: 1.0422 - val_accuracy: 0.6250 - lr: 2.0480e-11
Epoch 43/100
25/25 [==============================] - ETA: 0s - loss: 0.3428 - accuracy: 0.8784
Epoch 43: val_loss improved from 1.04219 to 0.96163, saving model to bird_resnet50.h5
25/25 [==============================] - 106s 4s/step - loss: 0.3428 - accuracy: 0.8784 - val_loss: 0.9616 - val_accuracy: 0.6547 - lr: 2.0480e-11
Epoch 44/100
25/25 [==============================] - ETA: 0s - loss: 0.3229 - accuracy: 0.8775
Epoch 44: val_loss improved from 0.96163 to 0.91164, saving model to bird_resnet50.h5
25/25 [==============================] - 107s 4s/step - loss: 0.3229 - accuracy: 0.8775 - val_loss: 0.9116 - val_accuracy: 0.6812 - lr: 2.0480e-11
Epoch 45/100
25/25 [==============================] - ETA: 0s - loss: 0.3331 - accuracy: 0.8830
Epoch 45: val_loss improved from 0.91164 to 0.81258, saving model to bird_resnet50.h5
25/25 [==============================] - 107s 4s/step - loss: 0.3331 - accuracy: 0.8830 - val_loss: 0.8126 - val_accuracy: 0.7219 - lr: 2.0480e-11
Epoch 46/100
25/25 [==============================] - ETA: 0s - loss: 0.3353 - accuracy: 0.8849
Epoch 46: val_loss improved from 0.81258 to 0.74280, saving model to bird_resnet50.h5
25/25 [==============================] - 106s 4s/step - loss: 0.3353 - accuracy: 0.8849 - val_loss: 0.7428 - val_accuracy: 0.7445 - lr: 2.0480e-11
Epoch 47/100
25/25 [==============================] - ETA: 0s - loss: 0.3388 - accuracy: 0.8843
Epoch 47: val_loss improved from 0.74280 to 0.74087, saving model to bird_resnet50.h5
25/25 [==============================] - 91s 4s/step - loss: 0.3388 - accuracy: 0.8843 - val_loss: 0.7409 - val_accuracy: 0.7398 - lr: 2.0480e-11
Epoch 48/100
25/25 [==============================] - ETA: 0s - loss: 0.3450 - accuracy: 0.8720
Epoch 48: val_loss improved from 0.74087 to 0.70857, saving model to bird_resnet50.h5
25/25 [==============================] - 106s 4s/step - loss: 0.3450 - accuracy: 0.8720 - val_loss: 0.7086 - val_accuracy: 0.7469 - lr: 2.0480e-11
Epoch 49/100
25/25 [==============================] - ETA: 0s - loss: 0.3341 - accuracy: 0.8872
Epoch 49: val_loss improved from 0.70857 to 0.66149, saving model to bird_resnet50.h5
25/25 [==============================] - 106s 4s/step - loss: 0.3341 - accuracy: 0.8872 - val_loss: 0.6615 - val_accuracy: 0.7719 - lr: 2.0480e-11
Epoch 50/100
25/25 [==============================] - ETA: 0s - loss: 0.3362 - accuracy: 0.8804
Epoch 50: val_loss improved from 0.66149 to 0.64098, saving model to bird_resnet50.h5
25/25 [==============================] - 91s 4s/step - loss: 0.3362 - accuracy: 0.8804 - val_loss: 0.6410 - val_accuracy: 0.7844 - lr: 2.0480e-11
Epoch 51/100
25/25 [==============================] - ETA: 0s - loss: 0.3343 - accuracy: 0.8830
Epoch 51: val_loss did not improve from 0.64098
25/25 [==============================] - 88s 4s/step - loss: 0.3343 - accuracy: 0.8830 - val_loss: 0.6640 - val_accuracy: 0.7828 - lr: 2.0480e-11
Epoch 52/100
25/25 [==============================] - ETA: 0s - loss: 0.3352 - accuracy: 0.8759
Epoch 52: val_loss improved from 0.64098 to 0.59710, saving model to bird_resnet50.h5
25/25 [==============================] - 105s 4s/step - loss: 0.3352 - accuracy: 0.8759 - val_loss: 0.5971 - val_accuracy: 0.8055 - lr: 2.0480e-11
```

Fig 4.8: Epoch 42 to 52

```
Epoch 53/100
25/25 [==============================] - ETA: 0s - loss: 0.3382 - accuracy: 0.8788
Epoch 53: val_loss did not improve from 0.59710
25/25 [==============================] - 103s 4s/step - loss: 0.3382 - accuracy: 0.8788 - val_loss: 0.6001 - val_accuracy: 0.8125 - lr: 2.0480e-11
Epoch 54/100
25/25 [==============================] - ETA: 0s - loss: 0.3233 - accuracy: 0.8853
Epoch 54: val_loss did not improve from 0.59710
25/25 [==============================] - 102s 4s/step - loss: 0.3233 - accuracy: 0.8853 - val_loss: 0.6010 - val_accuracy: 0.7977 - lr: 2.0480e-11
Epoch 55/100
25/25 [==============================] - ETA: 0s - loss: 0.3390 - accuracy: 0.8843
Epoch 55: val_loss improved from 0.59710 to 0.55946, saving model to bird_resnet50.h5
25/25 [==============================] - 90s 4s/step - loss: 0.3390 - accuracy: 0.8843 - val_loss: 0.5595 - val_accuracy: 0.8211 - lr: 2.0480e-11
Epoch 56/100
25/25 [==============================] - ETA: 0s - loss: 0.3481 - accuracy: 0.8723
Epoch 56: val_loss did not improve from 0.55946
25/25 [==============================] - 103s 4s/step - loss: 0.3481 - accuracy: 0.8723 - val_loss: 0.5905 - val_accuracy: 0.8102 - lr: 2.0480e-11
Epoch 57/100
25/25 [==============================] - ETA: 0s - loss: 0.3334 - accuracy: 0.8827
Epoch 57: val_loss did not improve from 0.55946
25/25 [==============================] - 104s 4s/step - loss: 0.3334 - accuracy: 0.8827 - val_loss: 0.5626 - val_accuracy: 0.8062 - lr: 2.0480e-11
Epoch 58/100
25/25 [==============================] - ETA: 0s - loss: 0.3523 - accuracy: 0.8765
Epoch 58: val_loss improved from 0.55946 to 0.54158, saving model to bird_resnet50.h5
25/25 [==============================] - 105s 4s/step - loss: 0.3523 - accuracy: 0.8765 - val_loss: 0.5416 - val_accuracy: 0.8211 - lr: 2.0480e-11
Epoch 59/100
25/25 [==============================] - ETA: 0s - loss: 0.3346 - accuracy: 0.8810
Epoch 59: val_loss did not improve from 0.54158
25/25 [==============================] - 103s 4s/step - loss: 0.3346 - accuracy: 0.8810 - val_loss: 0.5811 - val_accuracy: 0.8070 - lr: 2.0480e-11
Epoch 60/100
25/25 [==============================] - ETA: 0s - loss: 0.3399 - accuracy: 0.8827
Epoch 60: val_loss did not improve from 0.54158
25/25 [==============================] - 87s 4s/step - loss: 0.3399 - accuracy: 0.8827 - val_loss: 0.5706 - val_accuracy: 0.8070 - lr: 2.0480e-11
Epoch 61/100
25/25 [==============================] - ETA: 0s - loss: 0.3417 - accuracy: 0.8807
Epoch 61: val_loss did not improve from 0.54158

Epoch 61: ReduceLROnPlateau reducing learning rate to 4.096000622011431e-12.
25/25 [==============================] - 104s 4s/step - loss: 0.3417 - accuracy: 0.8807 - val_loss: 0.5587 - val_accuracy: 0.8148 - lr: 2.0480e-11
Epoch 62/100
25/25 [==============================] - ETA: 0s - loss: 0.3363 - accuracy: 0.8833
Epoch 62: val_loss did not improve from 0.54158
25/25 [==============================] - 104s 4s/step - loss: 0.3363 - accuracy: 0.8833 - val_loss: 0.5612 - val_accuracy: 0.8203 - lr: 4.0960e-12
Epoch 63/100
25/25 [==============================] - ETA: 0s - loss: 0.3288 - accuracy: 0.8781
Epoch 63: val_loss did not improve from 0.54158
25/25 [==============================] - 88s 4s/step - loss: 0.3288 - accuracy: 0.8781 - val_loss: 0.5467 - val_accuracy: 0.8195 - lr: 4.0960e-12
```

Fig 4.9: Epoch 53 to 63

```
Epoch 64/100
25/25 [==============================] - ETA: 0s - loss: 0.3467 - accuracy: 0.8755
Epoch 64: val_loss did not improve from 0.54158

Epoch 64: ReduceLROnPlateau reducing learning rate to 8.192000897078167e-13.
25/25 [==============================] - 88s 4s/step - loss: 0.3467 - accuracy: 0.8755 - val_loss: 0.5672 - val_accuracy: 0.8164 - lr: 4.0960e-12
Epoch 65/100
25/25 [==============================] - ETA: 0s - loss: 0.3273 - accuracy: 0.8820
Epoch 65: val_loss did not improve from 0.54158
25/25 [==============================] - 104s 4s/step - loss: 0.3273 - accuracy: 0.8820 - val_loss: 0.5635 - val_accuracy: 0.8164 - lr: 8.1920e-13
Epoch 66/100
25/25 [==============================] - ETA: 0s - loss: 0.3463 - accuracy: 0.8775
Epoch 66: val_loss did not improve from 0.54158
25/25 [==============================] - 87s 3s/step - loss: 0.3463 - accuracy: 0.8775 - val_loss: 0.5584 - val_accuracy: 0.8125 - lr: 8.1920e-13
Epoch 67/100
25/25 [==============================] - ETA: 0s - loss: 0.3233 - accuracy: 0.8833
Epoch 67: val_loss did not improve from 0.54158

Epoch 67: ReduceLROnPlateau reducing learning rate to 1.6384001360475466e-13.
25/25 [==============================] - 87s 3s/step - loss: 0.3233 - accuracy: 0.8833 - val_loss: 0.5576 - val_accuracy: 0.8148 - lr: 8.1920e-13
Epoch 68/100
25/25 [==============================] - ETA: 0s - loss: 0.3251 - accuracy: 0.8917
Epoch 68: val_loss did not improve from 0.54158
25/25 [==============================] - 104s 4s/step - loss: 0.3251 - accuracy: 0.8917 - val_loss: 0.5570 - val_accuracy: 0.8164 - lr: 1.6384e-13
Epoch 69/100
25/25 [==============================] - ETA: 0s - loss: 0.3297 - accuracy: 0.8836
Epoch 69: val_loss did not improve from 0.54158
25/25 [==============================] - 104s 4s/step - loss: 0.3297 - accuracy: 0.8836 - val_loss: 0.5515 - val_accuracy: 0.8188 - lr: 1.6384e-13
Epoch 70/100
25/25 [==============================] - ETA: 0s - loss: 0.3291 - accuracy: 0.8794
Epoch 70: val_loss improved from 0.54158 to 0.52632, saving model to bird_resnet50.h5
25/25 [==============================] - 105s 4s/step - loss: 0.3291 - accuracy: 0.8794 - val_loss: 0.5263 - val_accuracy: 0.8273 - lr: 1.6384e-13
Epoch 71/100
25/25 [==============================] - ETA: 0s - loss: 0.3332 - accuracy: 0.8791
Epoch 71: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3332 - accuracy: 0.8791 - val_loss: 0.5691 - val_accuracy: 0.8172 - lr: 1.6384e-13
Epoch 72/100
25/25 [==============================] - ETA: 0s - loss: 0.3362 - accuracy: 0.8791
Epoch 72: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3362 - accuracy: 0.8791 - val_loss: 0.5787 - val_accuracy: 0.8117 - lr: 1.6384e-13
Epoch 73/100
25/25 [==============================] - ETA: 0s - loss: 0.3367 - accuracy: 0.8768
Epoch 73: val_loss did not improve from 0.52632

Epoch 73: ReduceLROnPlateau reducing learning rate to 3.2768002178849846e-14.
25/25 [==============================] - 104s 4s/step - loss: 0.3367 - accuracy: 0.8768 - val_loss: 0.5563 - val_accuracy: 0.8227 - lr: 1.6384e-13
```

Fig 4.10: Epoch 64 to 73

```
Epoch 74/100
25/25 [==============================] - ETA: 0s - loss: 0.3241 - accuracy: 0.8872
Epoch 74: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3241 - accuracy: 0.8872 - val_loss: 0.5472 - val_accuracy: 0.8195 - lr: 3.2768e-14
Epoch 75/100
25/25 [==============================] - ETA: 0s - loss: 0.3349 - accuracy: 0.8830
Epoch 75: val_loss did not improve from 0.52632
25/25 [==============================] - 103s 4s/step - loss: 0.3349 - accuracy: 0.8830 - val_loss: 0.5841 - val_accuracy: 0.8180 - lr: 3.2768e-14
Epoch 76/100
25/25 [==============================] - ETA: 0s - loss: 0.3343 - accuracy: 0.8827
Epoch 76: val_loss did not improve from 0.52632

Epoch 76: ReduceLROnPlateau reducing learning rate to 6.553600300244697e-15.
25/25 [==============================] - 104s 4s/step - loss: 0.3343 - accuracy: 0.8827 - val_loss: 0.5712 - val_accuracy: 0.8125 - lr: 3.2768e-14
Epoch 77/100
25/25 [==============================] - ETA: 0s - loss: 0.3382 - accuracy: 0.8810
Epoch 77: val_loss did not improve from 0.52632
25/25 [==============================] - 88s 4s/step - loss: 0.3382 - accuracy: 0.8810 - val_loss: 0.5354 - val_accuracy: 0.8227 - lr: 6.5536e-15
Epoch 78/100
25/25 [==============================] - ETA: 0s - loss: 0.3438 - accuracy: 0.8794
Epoch 78: val_loss did not improve from 0.52632
25/25 [==============================] - 88s 4s/step - loss: 0.3438 - accuracy: 0.8794 - val_loss: 0.5731 - val_accuracy: 0.8156 - lr: 6.5536e-15
Epoch 79/100
25/25 [==============================] - ETA: 0s - loss: 0.3485 - accuracy: 0.8765
Epoch 79: val_loss did not improve from 0.52632

Epoch 79: ReduceLROnPlateau reducing learning rate to 1.3107200431082805e-15.
25/25 [==============================] - 87s 3s/step - loss: 0.3485 - accuracy: 0.8765 - val_loss: 0.5794 - val_accuracy: 0.8062 - lr: 6.5536e-15
Epoch 80/100
25/25 [==============================] - ETA: 0s - loss: 0.3278 - accuracy: 0.8810
Epoch 80: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3278 - accuracy: 0.8810 - val_loss: 0.5693 - val_accuracy: 0.8102 - lr: 1.3107e-15
Epoch 81/100
25/25 [==============================] - ETA: 0s - loss: 0.3403 - accuracy: 0.8791
Epoch 81: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3403 - accuracy: 0.8791 - val_loss: 0.5608 - val_accuracy: 0.8141 - lr: 1.3107e-15
Epoch 82/100
25/25 [==============================] - ETA: 0s - loss: 0.3379 - accuracy: 0.8781
Epoch 82: val_loss did not improve from 0.52632

Epoch 82: ReduceLROnPlateau reducing learning rate to 2.621440128568208e-16.
25/25 [==============================] - 88s 4s/step - loss: 0.3379 - accuracy: 0.8781 - val_loss: 0.5564 - val_accuracy: 0.8203 - lr: 1.3107e-15
Epoch 83/100
25/25 [==============================] - ETA: 0s - loss: 0.3387 - accuracy: 0.8788
Epoch 83: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3387 - accuracy: 0.8788 - val_loss: 0.5778 - val_accuracy: 0.8094 - lr: 2.6214e-16
```

Fig 4.11: Epoch 74 to 83

```
Epoch 84/100
25/25 [==============================] - ETA: 0s - loss: 0.3400 - accuracy: 0.8810
Epoch 84: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3400 - accuracy: 0.8810 - val_loss: 0.5268 - val_accuracy: 0.8211 - lr: 2.6214e-16
Epoch 85/100
25/25 [==============================] - ETA: 0s - loss: 0.3561 - accuracy: 0.8768
Epoch 85: val_loss did not improve from 0.52632

Epoch 85: ReduceLROnPlateau reducing learning rate to 5.2428803630155353e-17.
25/25 [==============================] - 104s 4s/step - loss: 0.3561 - accuracy: 0.8768 - val_loss: 0.5677 - val_accuracy: 0.8047 - lr: 2.6214e-16
Epoch 86/100
25/25 [==============================] - ETA: 0s - loss: 0.3400 - accuracy: 0.8820
Epoch 86: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3400 - accuracy: 0.8820 - val_loss: 0.5525 - val_accuracy: 0.8070 - lr: 5.2429e-17
Epoch 87/100
25/25 [==============================] - ETA: 0s - loss: 0.3235 - accuracy: 0.8846
Epoch 87: val_loss did not improve from 0.52632
25/25 [==============================] - 103s 4s/step - loss: 0.3235 - accuracy: 0.8846 - val_loss: 0.5737 - val_accuracy: 0.8203 - lr: 5.2429e-17
Epoch 88/100
25/25 [==============================] - ETA: 0s - loss: 0.3408 - accuracy: 0.8801
Epoch 88: val_loss did not improve from 0.52632

Epoch 88: ReduceLROnPlateau reducing learning rate to 1.0485760990728867e-17.
25/25 [==============================] - 103s 4s/step - loss: 0.3408 - accuracy: 0.8801 - val_loss: 0.5948 - val_accuracy: 0.8109 - lr: 5.2429e-17
Epoch 89/100
25/25 [==============================] - ETA: 0s - loss: 0.3278 - accuracy: 0.8823
Epoch 89: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3278 - accuracy: 0.8823 - val_loss: 0.5436 - val_accuracy: 0.8305 - lr: 1.0486e-17
Epoch 90/100
25/25 [==============================] - ETA: 0s - loss: 0.3368 - accuracy: 0.8778
Epoch 90: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3368 - accuracy: 0.8778 - val_loss: 0.5479 - val_accuracy: 0.8266 - lr: 1.0486e-17
Epoch 91/100
25/25 [==============================] - ETA: 0s - loss: 0.3315 - accuracy: 0.8804
Epoch 91: val_loss did not improve from 0.52632

Epoch 91: ReduceLROnPlateau reducing learning rate to 2.097152165058549e-18.
25/25 [==============================] - 103s 4s/step - loss: 0.3315 - accuracy: 0.8804 - val_loss: 0.5872 - val_accuracy: 0.8062 - lr: 1.0486e-17
Epoch 92/100
25/25 [==============================] - ETA: 0s - loss: 0.3375 - accuracy: 0.8781
Epoch 92: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3375 - accuracy: 0.8781 - val_loss: 0.5892 - val_accuracy: 0.8117 - lr: 2.0972e-18
Epoch 93/100
25/25 [==============================] - ETA: 0s - loss: 0.3475 - accuracy: 0.8762
Epoch 93: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3475 - accuracy: 0.8762 - val_loss: 0.5509 - val_accuracy: 0.8203 - lr: 2.0972e-18
```

Fig 4.12: Epoch 84 to 93

```
Epoch 94/100
25/25 [==============================] - ETA: 0s - loss: 0.3345 - accuracy: 0.8830
Epoch 94: val_loss did not improve from 0.52632

Epoch 94: ReduceLROnPlateau reducing learning rate to 4.19430449555322e-19.
25/25 [==============================] - 103s 4s/step - loss: 0.3345 - accuracy: 0.8830 - val_loss: 0.5517 - val_accuracy: 0.8328 - lr:
2.0972e-18
Epoch 95/100
25/25 [==============================] - ETA: 0s - loss: 0.3355 - accuracy: 0.8843
Epoch 95: val_loss did not improve from 0.52632
25/25 [==============================] - 104s 4s/step - loss: 0.3355 - accuracy: 0.8843 - val_loss: 0.5558 - val_accuracy: 0.8203 - lr:
4.1943e-19
Epoch 96/100
25/25 [==============================] - ETA: 0s - loss: 0.3298 - accuracy: 0.8827
Epoch 96: val_loss did not improve from 0.52632
25/25 [==============================] - 87s 3s/step - loss: 0.3298 - accuracy: 0.8827 - val_loss: 0.5524 - val_accuracy: 0.8305 - lr:
4.1943e-19
Epoch 97/100
25/25 [==============================] - ETA: 0s - loss: 0.3363 - accuracy: 0.8794
Epoch 97: val_loss did not improve from 0.52632

Epoch 97: ReduceLROnPlateau reducing learning rate to 8.388609197901593e-20.
25/25 [==============================] - 104s 4s/step - loss: 0.3363 - accuracy: 0.8794 - val_loss: 0.5624 - val_accuracy: 0.8172 - lr:
4.1943e-19
Epoch 98/100
25/25 [==============================] - ETA: 0s - loss: 0.3315 - accuracy: 0.8797
Epoch 98: val_loss did not improve from 0.52632
25/25 [==============================] - 103s 4s/step - loss: 0.3315 - accuracy: 0.8797 - val_loss: 0.5980 - val_accuracy: 0.8148 - lr:
8.3886e-20
Epoch 99/100
25/25 [==============================] - ETA: 0s - loss: 0.3431 - accuracy: 0.8807
Epoch 99: val_loss improved from 0.52632 to 0.52415, saving model to bird_resnet50.h5
25/25 [==============================] - 105s 4s/step - loss: 0.3431 - accuracy: 0.8807 - val_loss: 0.5242 - val_accuracy: 0.8164 - lr:
8.3886e-20
Epoch 100/100
25/25 [==============================] - ETA: 0s - loss: 0.3404 - accuracy: 0.8797
Epoch 100: val_loss did not improve from 0.52415
25/25 [==============================] - 103s 4s/step - loss: 0.3404 - accuracy: 0.8797 - val_loss: 0.5388 - val_accuracy: 0.8203 - lr:
8.3886e-20
```

Fig 4.13: Epoch 94 to 100

4.1.4 Testing

Model testing is referred to as the process where the performance of a fully trained model is evaluated on a testing set. Testing help us to checks (tests) whether this built model works correctly or not. Once training is complete, the final performance of the model is evaluated on the testing set to assess its ability to generalize to new data. The trained model can be used to classify new bird sound recordings if it achieves an optimum accuracy. The test accuracy got after evaluating the model is 81.73%.

```
27]:
model.evaluate(
    x=valid_ds,
    batch_size=None,
    verbose='auto',
    sample_weight=None,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
    return_dict=False
)

11/11 [==============================] - 27s 2s/step - loss: 0.5380 - accuracy: 0.8173
27]: [0.5379570126533508, 0.8173217177391052]
```

Fig 4.14: Model evaluation

5. RESULTS AND DISCUSSION

The aim of this project was to build a system which helps to identify if the bird is Blue Jay, Black-capped Chickadee, Mallard, Common Tern or American Redstart by their sound. The project use Resnet50 architecture to build the model. The project was successfully completed with a training accuracy of 88.07%. The accuracy is the metrics used in the training of the dataset. Accuracy is a measurement of observational error. It defines how close or far off a given set of measurement are to their true value. This model showed better accuracy with increase in each epoch during training. It reached 88.07% accuracy at 99th epoch of training with validation accuracy of 81.64%. This shows that by increasing the number of epochs in training, we can improve the quality of prediction. But may not be possible all the time. While evaluating the model with the test dataset it achieved 81.73%. This shows that even at high training accuracy the model may not provide the same accuracy while predicting a new unknown data. To evaluating and summarizing the performance of the model we use different techniques like accuracy graph, loss graph, classification report and confusion matrix. Loss graph shows how poorly or well a model behaves after each iteration of optimization. An accuracy graph is used to measure the model's performance. It is draws as a comparative graph line between training and validation. A classification report is a performance evaluation metric. It is used to show the precision, recall, F1 Score, and support of your trained classification model. A confusion matrix presents a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes. All these accuracy graph, loss graph, classification report and confusion matrix of the proposed system is given below:

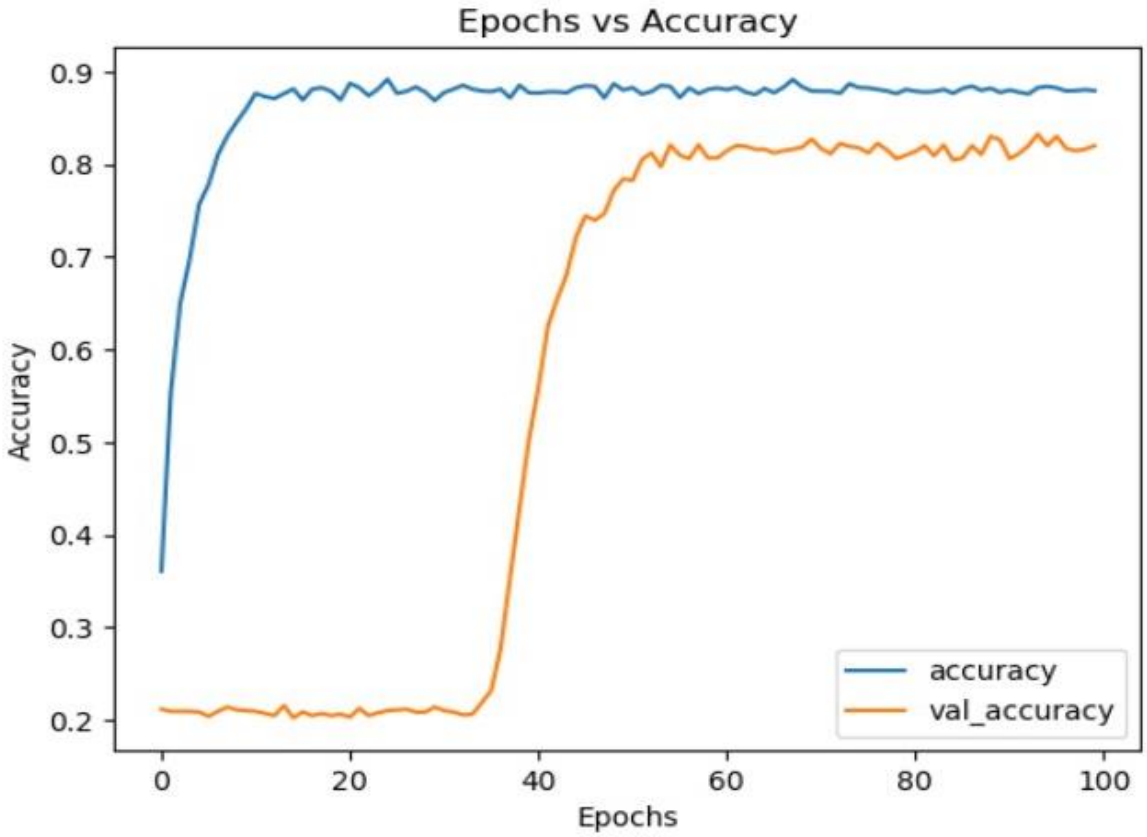


Fig 5.1: Accuracy graph

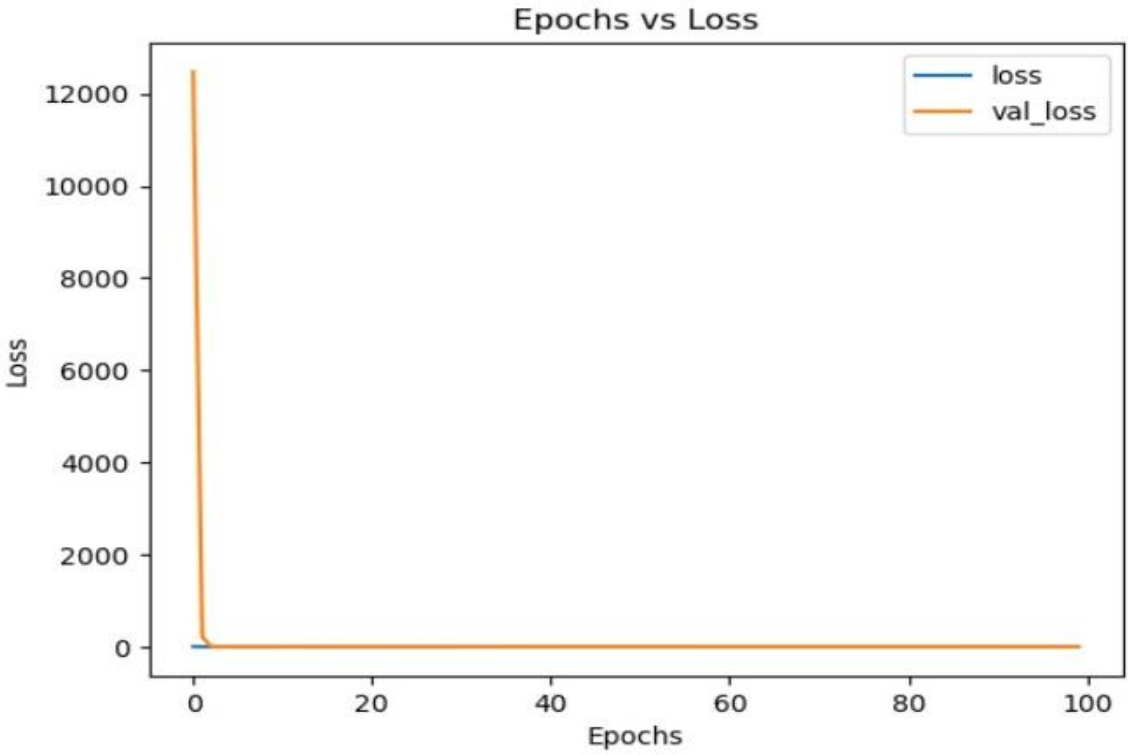


Fig 5.2: Loss graph

	precision	recall	f1-score
0	0.22	0.22	0.22
1	0.19	0.19	0.19
2	0.19	0.16	0.18
3	0.17	0.20	0.19
4	0.23	0.23	0.23

Fig 5.3: Classification report

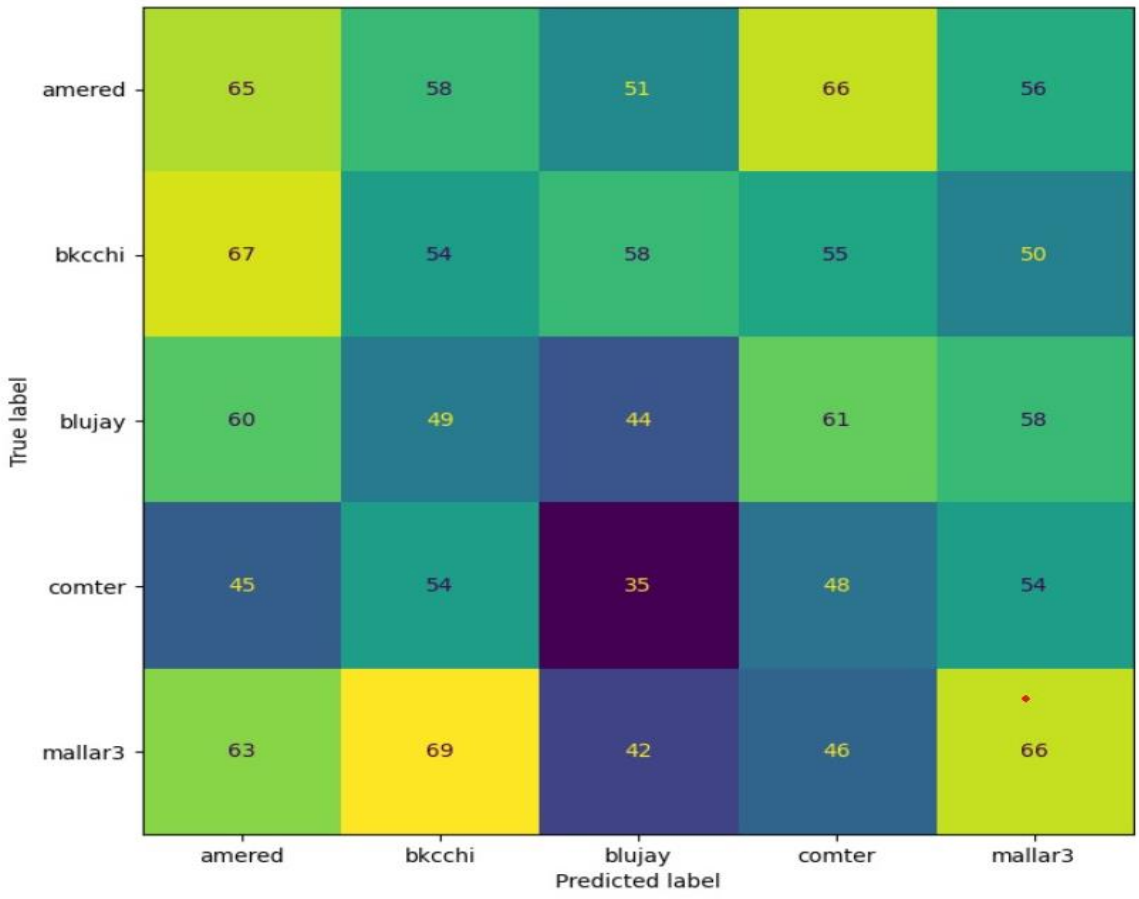


Fig 5.4: confusion matrix

From all these above performance evaluations we can conclude that training accuracy is greater than validation accuracy and training and validation loss is having a small difference. Also, we can see that the loss was high at the initial epoch. Precision, recall and f1-score of each class is shown. From the confusion matrix we can confirm that the fifth class(mallard) is having more correct prediction (true positive) than other classes. And the class having least correct prediction is blue jay.

6. MODEL DEPLOYMENT

Model deployment in deep learning refers to the process of making a trained deep learning model available for use in a production environment. The goal of model deployment is to provide a way for end-users to interact with the model and make predictions based on new input data. It mainly focusses on the user interface of the system. The user interface created here is using HTML, CSS and bootstrap. These make the web application more interactive for users. The figures below show the user interface of the proposed system. The user interface is very simple and easy to understand in this system. There are only some elements displayed on the pages. There is a file upload option provided. The user has to choose any mp3 audio file from the local storage through that. Then there is a submit button. On click of the button, the uploaded file will be passed to a function where audio is trimmed and converted to mel-spectrogram. And also, they are pre-processed before passing it into the model. All these pre-processed data and inputted data is displayed in result page along with the result. The output may be one of the 5 labels Blue Jay, Black-capped Chickadee, Mallard, Common Tern or American Redstart.

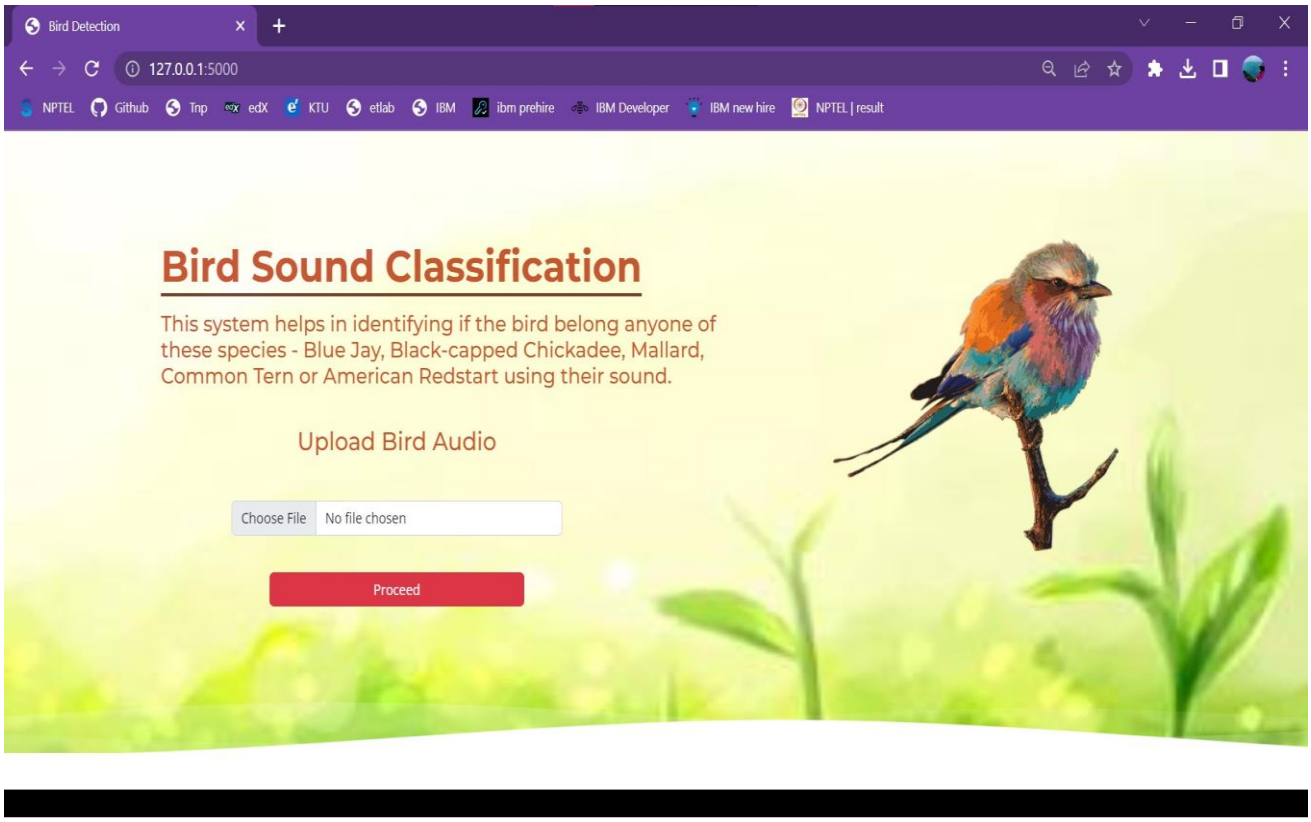


Fig 6.1: Home page

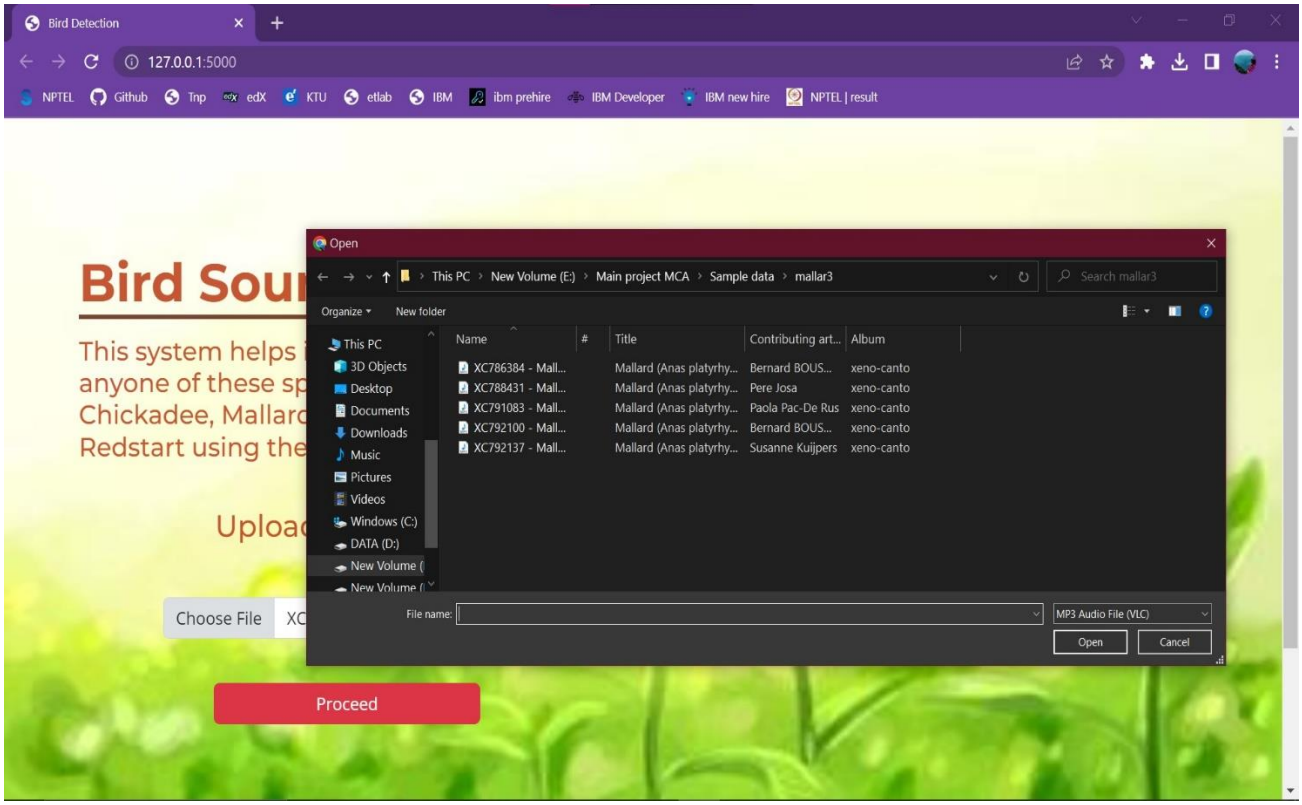


Fig 6.2: Home page when choosing file

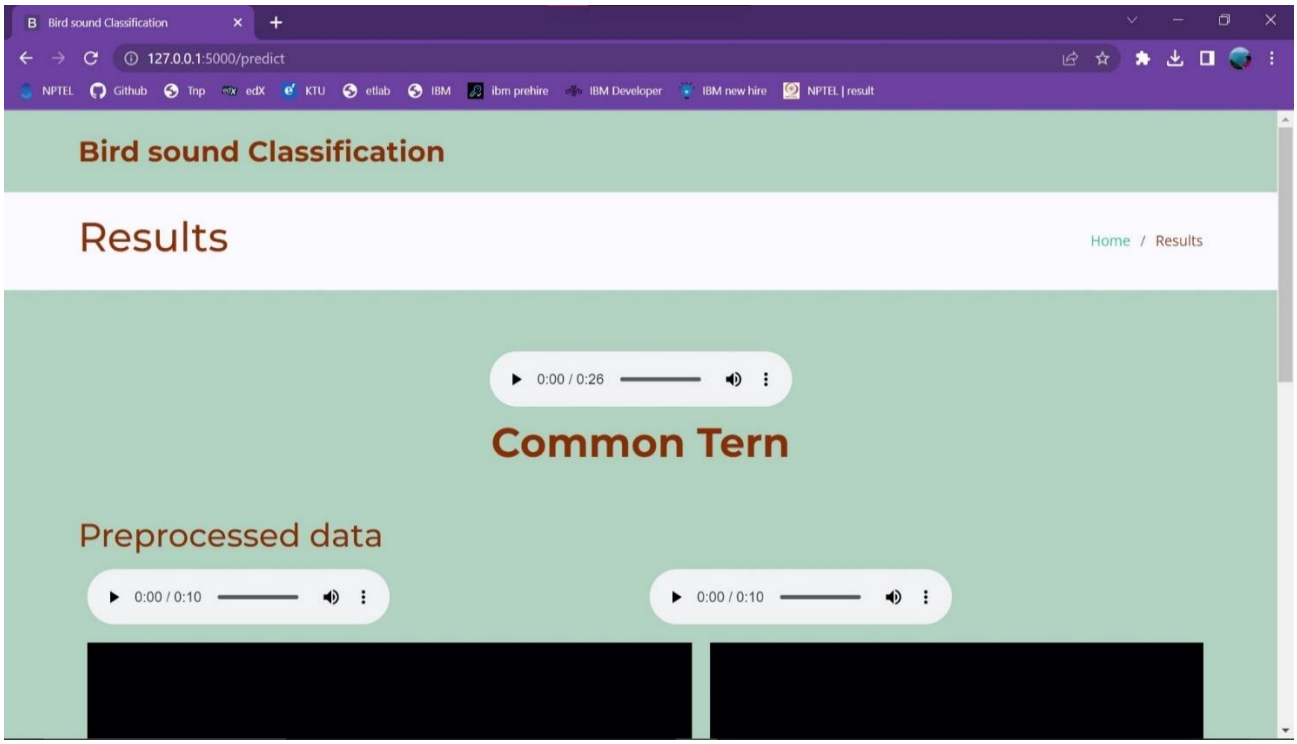


Fig 6.3: Result page input and output section

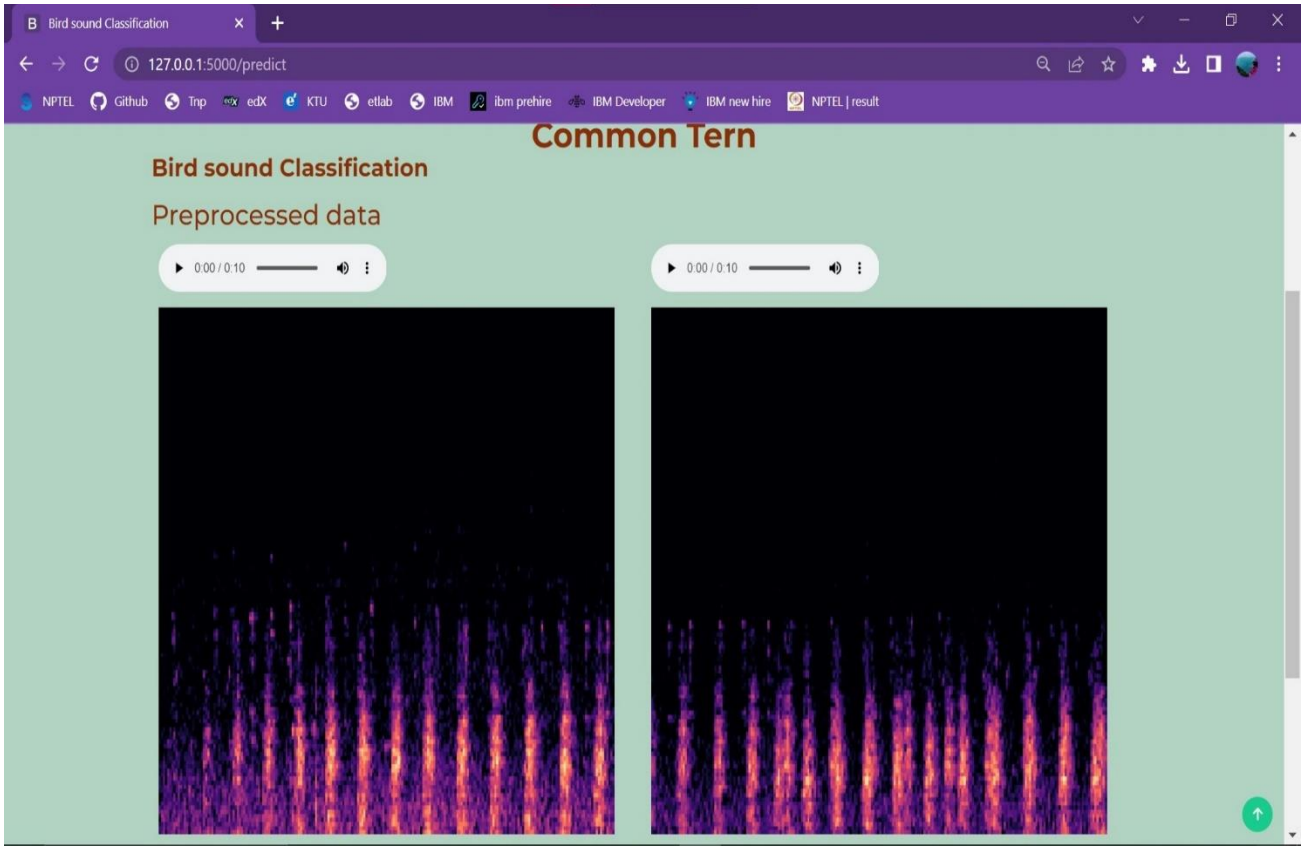


Fig 6.4: Result page pre-processed data section

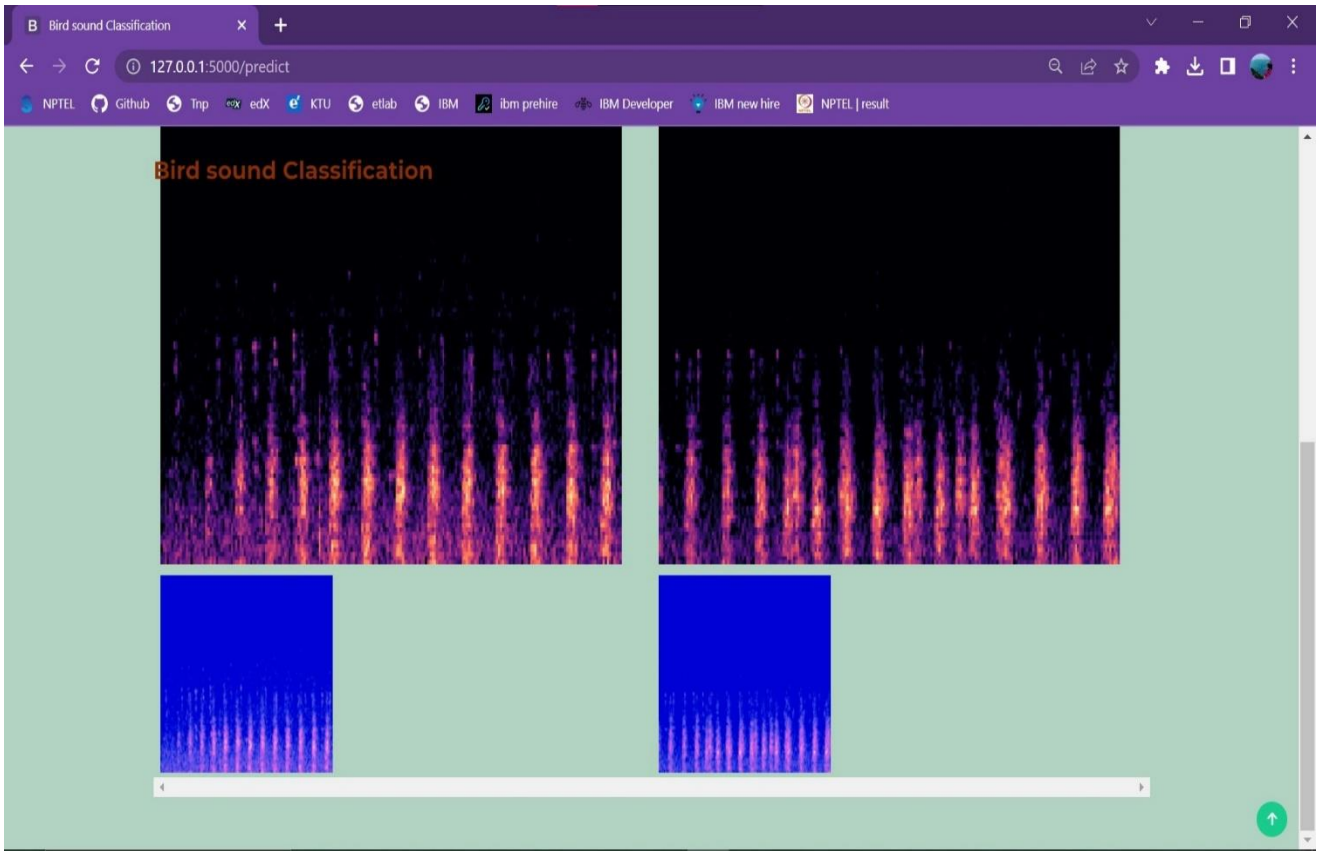


Fig 6.5: Result page model input section

7. GIT HISTORY

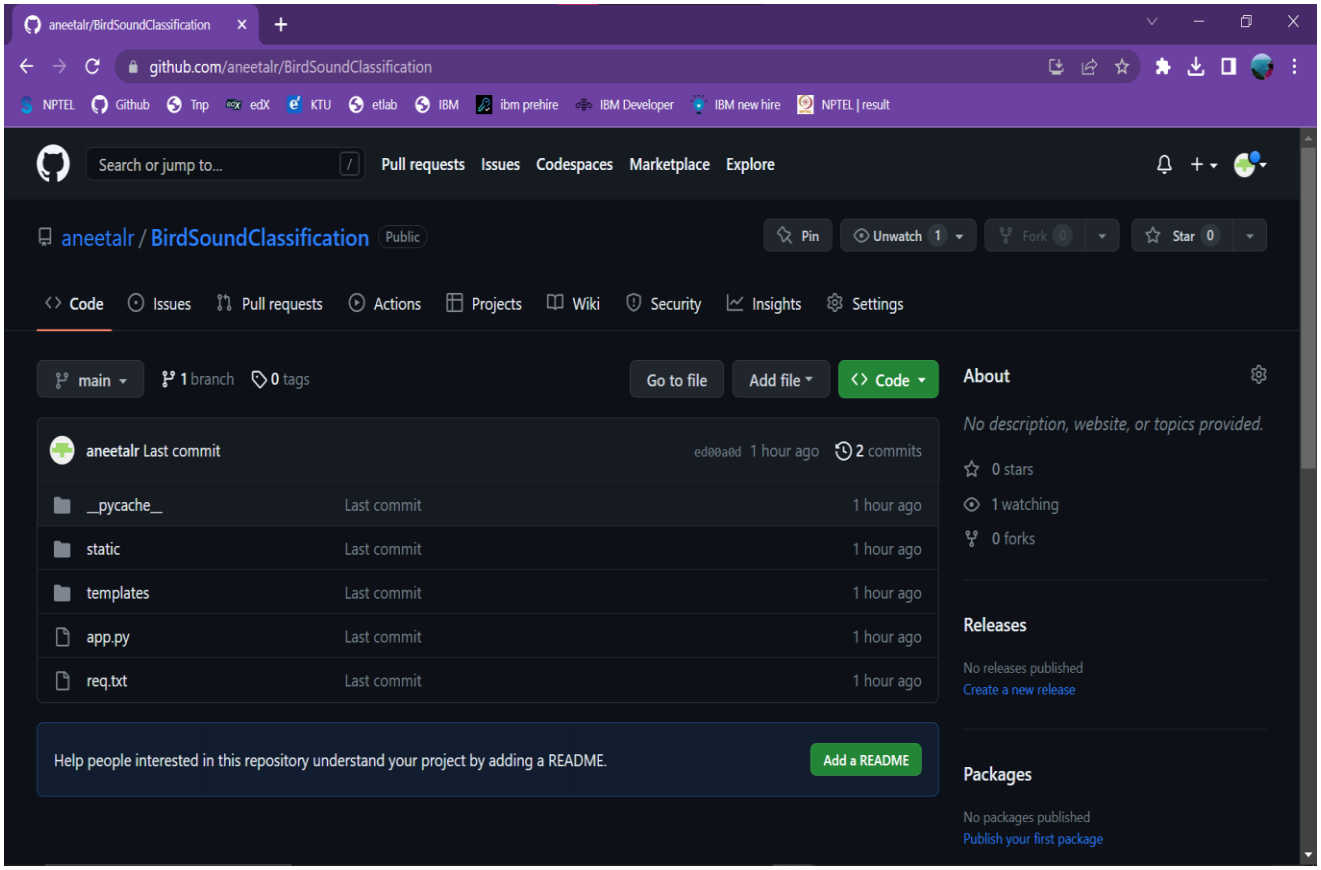


Fig 7.1: Git repository

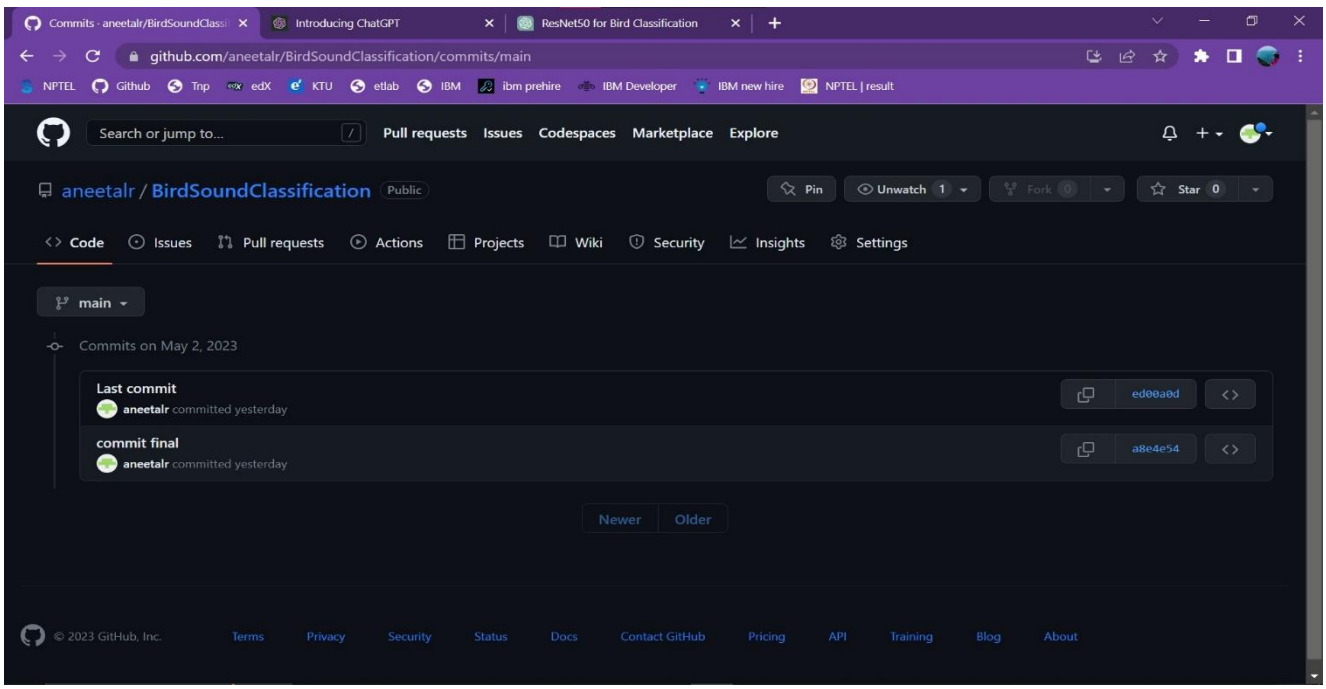


Fig 7.2: Git history

8. CONCLUSION

The main aim of this project was to create a web application called Bird sound classification, to identify the bird by their sound. This system handles with pre-recorded audios of bird. In this project, we focused on classifying five bird species: Blue Jay, Black-capped Chickadee, Mallard, Common Tern, and American Redstart. First step is converting the bird sound recordings to mel-spectrograms, we can extract useful features that can be used as input to the ResNet50 model by this conversion. The project would involve pre-processing the dataset to extract mel-spectrograms, training the ResNet50 model on the mel-spectrograms, and evaluating the performance of the model on a test dataset. The model has 88.07% of training accuracy, 81.64% of validation accuracy and 81.73% of evaluation accuracy. I used Google Colab and Jupyter Notebook for developing this application. It made the work so easy and efficient. The use of Graphics Processing Unit (GPU) can accelerate the training process and yield faster results. The models were trained six to ten times for different number of epochs and finally an optimum number 20 was chosen as the number of epochs for training in the final stage. Saving the model after each epoch can help us to choose the best model based on the validation loss and validation accuracy. By this I have achieved the above-mentioned accuracy. Implementing, the bird sound classification system using ResNet50 and mel-spectrograms could accurately classify bird species based on their vocalizations. This system could be useful for researchers studying bird behaviour, conservationists monitoring bird populations, or bird enthusiasts interested in identifying the birds they hear in the wild.

9. FUTURE WORK

Currently, our proposed system focusses on a relatively small set of bird species. However, it's possible to expand the scope of these systems to include a wider variety of birds. This could involve training the system on more bird species or developing new algorithms that can recognize a wider range of bird calls. This system is designed to analyze pre-recorded audio. However, there is potential to develop real-time systems that can identify birds in the wild as they call. This could be useful for monitoring bird populations, studying bird behavior, or detecting the presence of rare or endangered species. Also, Bird sound classification systems could be integrated with other technologies to create more powerful tools for bird researchers and conservationists. We can also upgrade the system in a way that sound from a video is extracted and use that audio to identify.

10. APPENDIX

10.1 Minimum Software Requirements

Operating System : Windows, Linux, Mac

Software : Kaggle, Google Colab

10.2 Minimum Hardware Requirements

Hardware capacity : 256 GB (minimum)

RAM : 8 GB

Processor : Intel Core i5 preferred

GPU : Nvidia K80 (Kaggle)

Display : 1366 * 768

11. REFERENCES

1. https://birdnet.cornell.edu/
2. https://www.kaggle.com/c/birdsong-recognition/data
3. Ishan Taneja, Vipul Arora, and Rajiv Ratn Shah, "A Comparative Study of Bird Sound Classification Using Mel-Frequency Spectrograms and Convolutional Neural Networks," in IEEE Access, vol. 8, pp. 132693-132704, 2020, doi: 10.1109/ACCESS.2020.3019977.
4. M.A.Umar, M.Jibrin and A.Bashir, "Bird Sound Classification Using Convolutional Neural Networks and Mel-Spectrogram," in IEEE Access, vol. 8, pp. 134468-134478, 2020, doi: 10.1109/ACCESS.2020.3016258.
5. https://xeno-canto.org/
6. https://datagen.tech/guides/computer-vision/resnet-50/
7. https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/
8. J. Chen, C. Du and X. Liu, "An Efficient Bird Sound Classification System using Mel-spectrogram and ResNet50," in IEEE Access, vol. 8, pp. 215661-215670, 2020, doi: 10.1109/ACCESS.2020.3041762.
9. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
10. https://medium.com/@kenneth.ca95/a-guide-to-transfer-learning-with-keras-using-resnet50-a81a4a28084b