CAPUCHINBIRD CALL DETECTION USING MACHINE LEARNING

A project report submitted in partial
fulfillment of the requirements for the degree of
Master of Science in
Computer Science

By

PRASAD ARUN BELSARE
M.S., University of Colorado, Denver, 2026

May 2026
University of Colorado Denver

This project report for the Master of Science degree by

Prasad Arun Belsare

to be approved for the

Computer Science Program

by

Gita Alaghband, Chair

Ashis Biswas, Advisor

Date 16 May, 2026

Belsare, Prasad Arun (Master of Science, Computer Science)

CAPUCHINBIRD CALL DETECTION USING MACHINE LEARNING

M.S. Course Project directed by Assistant Professor Ashis Biswas

ABSTRACT

Wildlife researchers often record large amounts of forest audio, but identifying the specific bird calls they need is time consuming and difficult. The Capuchinbird offers valuable information through its vocal behavior, yet its call is easily lost within other sounds of the forest. Existing tools struggle with noisy conditions and limited data, leaving a gap for a system that is both accurate and practical for real field use.

I built a machine learning pipeline that transforms raw audio into detailed spectrograms and trains multiple deep learning models to separate true bird calls from complex acoustic clutter. The system was then applied to full forest recordings to estimate call counts with improved speed and consistency. The results show that automatic detection is not only possible but powerful. This work has the potential to support large scale conservation efforts and bring faster insight to wildlife monitoring teams.

This project report is approved for recommendation to the Graduate committee.

Project Advisor:

____

Assistant Professor Ashis Biswas

ACKNOWLEDGEMENTS

I want to express my gratitude to Professor Ashis Biswas for helping me with this research. His excellent expertise of machine learning was important to my job. My attitude to technical challenges has radically changed since he taught me to concentrate on how the algorithms work rather than just using them.

When I got stuck on implementation or felt unsure about my methodology, his feedback helped me get back on track. I especially appreciated his advice on model optimization. He pushes his students to be better while still being incredibly supportive, asking us to think about both the code and its real-world use.

Thanks also to the Computer Science department for providing the computational resources I required to run these models. Professor Biswas was always open to discuss ideas, and his guidance helped me become a better researcher.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 The Problem

In many wildlife studies, researchers depend on long audio recordings taken from different parts of the forest to understand what is happening in those areas. These recordings can hold important clues about animals, but they also come with a lot of background noise. Sounds from insects, wind, and other birds all mix together, and it becomes difficult to focus on one specific species. For the Capuchinbird, this is a real issue because its call does not always stand out clearly and can easily get lost in all the other sounds. As a result, researchers have to sit and listen for long periods just to find a few meaningful moments, which takes a lot of effort and can get very repetitive.

Even though there are some tools for detecting bird sounds, they are often built for much larger datasets or for many species at once. They usually do not work well when the recordings are messy or when the goal is to track only one specific bird. Because of these limitations, the process continues to be slow and manual. This makes it hard for researchers to study the Capuchinbird consistently, and it also limits how much data they can analyze. A better solution would help them find the bird's calls faster and make the whole process more practical for real field studies.

1.2 Motivation and Challenges

The main motivation for this project comes from the amount of time and effort researchers currently spend trying to separate useful bird calls from long and noisy forest recordings. There are a few tools that can help with general bird sound classification, but many of them are made for large collections of species or depend on big, well curated datasets. They often work well in clean audio conditions, but their accuracy drops when the environment is messy, which is usually the case in real forest recordings. For the Capuchinbird, this becomes a challenge because its call does not always appear clearly and can be hidden under many other sounds.

One major drawback of the current situation is that researchers still end up doing a lot of the work manually, which limits how much data they can process. At the same time, completely ignoring the noise or using simple filtering techniques is not enough, because important details in the call may also be removed. This gap shows the need for a system that can handle real recordings, learn the unique structure of the Capuchinbird call, and work even when the audio is not perfect. Developing a model that can detect these calls faster and more reliably would help researchers focus on analysis instead of sorting through hours of audio, and it would also make long term monitoring more practical for conservation efforts.

1.3 Problem Statement

Listening to long hours of forest recordings to detect bird calls is slow and tiring, and it makes large-scale monitoring difficult. For the Capuchinbird, this task is even harder because of background forest noise and the variety in its calls, which usually requires trained experts.

To address this, the project aims to develop a system that uses machine learning to automatically recognize and count Capuchinbird calls. This will help make acoustic monitoring more efficient and accessible for conservation efforts.

1.4 Research Aims (or Objectives)

1. This project aims to improve the process of identifying Capuchinbird calls in long forest recordings through the following goals:
2. Build a machine learning model that can recognize Capuchinbird calls even when they are mixed with background noise from the forest.
3. Prepare and use audio data by converting recordings into spectrograms, which allows the model to learn the unique patterns that distinguish the Capuchinbird call.
4. To determine which model provides the most dependable outcomes for this particular task, compare various deep learning techniques.
5. Develop a detection method that can scan full recordings, count the number of calls, and make the overall analysis faster and easier for researchers.

1.5 Outline of the report

- **Chapter 1: Introduction** Presents the problem, project objectives, final goals, and the structure of the report.
- **Chapter 2: Literature Review** Discusses earlier research on audio analysis and bird call detection, and summarizes prior work in the field.
- **Chapter 3: Methodology** Describes the steps taken in the project, including data preparation, model design, and the experimental setup.

- **Chapter 4: Results and Analysis** Presents the main findings, performance results, and interpretation of the outcomes.
- **Chapter 5: Conclusion and Future Work** Summarizes the contributions of the project and suggests possible future improvements or extension

CHAPTER 2
LITERATURE REVIEW

2.1 Salamon and Bello: Environmental Sound Classification with CNNs

Salamon and Bello studied how convolutional neural networks can be used to classify environmental sounds by treating them as images in the form of spectrograms. Their work showed that CNNs are good at capturing both local and global patterns in the time–frequency representation of audio, which helps when the recordings include sudden changes or overlapping sounds. They emphasized that real world audio is rarely clean, so the model needs to handle a lot of unpredictability. One of their most important contributions was their focus on data augmentation. They attempted many augmentation methods such as time stretching, pitch shifting, dynamic range compression, and blending in background noise. These methods improved the model's ability to generalize, particularly when the dataset was small.

Another point they discussed is how hand crafted features, which were used in older sound classification systems, often fail when the audio conditions change. CNNs, on the other hand, are more adaptable since they learn features directly from the data. Their experiments demonstrated that a well designed CNN can outperform many traditional machine learning approaches in noisy outdoor environments. This paper is useful for my project because the Capuchinbird recordings I worked with also have noise and variations, and their findings support the idea that deep learning can handle these challenges better than simpler methods.

2.2 BirdNET: Deep Learning for Avian Diversity Monitoring

The BirdNET paper presents a large scale deep learning system designed to identify thousands of bird species from real recordings collected in the field. The authors used spectrogram based analysis and trained the model on a huge dataset that includes recordings from different regions, seasons, and background conditions. As a result, the model can identify a wide variety of species even in the presence of wind, insects, or other birds. BirdNET's ability to be used practically in conservation and ecological research in addition to classification is one of its advantages. Researchers can more easily track bird populations over time because to the system's speedy processing of lengthy recordings.

Despite its advantages, the paper recognizes a number of difficulties. Some species have relatively little labeled audio data, which makes it challenging for the model to learn their calls accurately. Additionally, a lot of birds make sounds that vary based on the circumstances, which can lower the accuracy or confidence of the model. The authors note that although a broad model is effective for many species, there are instances in which concentrating on a single species or a small group can result in superior performance. Given that I am dealing with a single species and a small dataset, this immediately relates to my topic. The BirdNET study shows both the potential and the limitations of large models and suggests that targeted models can sometimes offer more reliable detection for species with subtle or variable calls.

CHAPTER 3
METHODS

This chapter shows in detail how I addressed the problem and how the system was constructed from start to finish. The main goal was to design a workflow that could take raw audio from the forest, transform it into a form that a model could understand, train classifiers on short samples, and then use those classifiers to scan long recordings and count Capuchinbird calls. While the processes look easy, a lot of little decisions like audio processing, window widths, model structure, and assessment had a huge impact on how well the system worked. The following sections detail each stage of the technique and how it ties to the aims of the project.

3.1 Data Collection

For this project I worked with two kinds of audio recordings. The first set contained Capuchinbird calls from Xeno Canto. Each file usually has several calls inside it, so I wrote a small script that downloads the recordings by using their clip IDs. After downloading them, I used a table of time stamps that marks the beginning and end of each call. The code loads the full recording, slices out only the portion of the audio that contains the call, and saves each call as its own short clip. This gave me a cleaner set of examples that focus on the Capuchinbird vocalizations rather than the entire recording.

The second set of data came from general forest and wildlife recordings that did not contain Capuchinbird calls. I used another script that reads a list of clip IDs and URLs and downloads the audio from an online sound library. Many of these files were quite long, so after downloading them I cut each one into short segments. These short pieces act as negative examples for training because they contain all sorts of background sounds like insects, wind, and other birds. Using both positive and negative examples helped create a balanced dataset for the models.

3.2 Preprocessing

The recordings differed in quality, length, and format since the audio was collected from numerous locations. I started by resampling each audio file to 16 kHz and converting them to a single channel in order to clean things up. This step guarantees that all clips are treated the same manner by the model.

Next, I adjusted every clip to a fixed length of three seconds. If a clip was shorter than three seconds, I padded the end with zeros. If it was longer, I kept only the first three seconds. This created a consistent input size that later made it easier to build and train the models.

After setting the length, each clip was transformed into a spectrogram. A spectrogram is a time frequency map that shows how the sound changes from moment to moment. I used the short time

Fourier transform with a window size of three hundred and twenty samples and a step size of thirty two samples. This produced a clear visual representation of the audio. The values were then converted into magnitude form and reshaped so the models could treat them like images.

These preprocessing steps created a stable and organized dataset. Both the Capuchinbird calls and the background recordings went through the same process, so the models learned from inputs that were consistent and fairly close to what they would see in real forest recordings.

3.3 Model Architectures

The project uses three different neural network models to test how well each one can identify bird calls. Each of the models start with the same fundamental structure because they all use spectrograms as input, but each handle the data differently in the temporal direction. The purpose of evaluating several models was to evaluate whether the standard convolution method would be enough, or if adding a sequence learning component would help the system understand the call more precisely.

3.3.1 Convolutional Neural Network (CNN)

The first model is a standard CNN architecture. It consists of three convolutional blocks, each containing a Conv2D layer, batch normalization, max pooling, and dropout regularization. The convolutional layers extract spatial features from the mel spectrogram by learning hierarchical patterns—starting with low level features such as edges and frequency bands in early layers, progressing to high level patterns characteristic of Capuchinbird vocalizations in deeper layers. After feature extraction, the output is flattened and sent through fully connected dense layers with dropout for final binary classification.

Using spectrogram representations as the baseline, this model shows that CNNs may efficiently learn discriminative features without explicitly modeling temporal relationships.

3.3.2 CNN with LSTM(Long Short-Term Memory) Layer

The second model extends the CNN by including bidirectional LSTM layers. The CNN component continues to utilize the same way to extract spectral features. But after that, instead of going straight to a decision, the features are rearranged into a sequence. Each point in the sequence represents a small time slice of the audio. This sequence goes through two stacked bidirectional LSTM layers. The first layer has 64 units and the second has 32 units. Both layers are bidirectional. This lets the model notice patterns that happen over time, like when a call rises in pitch or repeats itself.

Bird calls often have structure in time. Some calls might start low and rise high, or they might repeat certain sounds. A regular CNN treats the whole spectrogram as one static image, so it can miss these patterns. The LSTM gives the model a way to notice when one part of the call leads to another part. This should help the model better understand calls that have these time based features.

3.3.3 CNN with GRU(Gated Recurrent Unit) Layer

Similar to the second model, the third model substitutes two stacked bidirectional GRU layers for LSTM layers. There are 64 units in the first GRU layer and 32 in the second. GRUs and LSTMs both read sequences and remember patterns over time. The main difference is that GRUs use a simpler design.

The LSTM has three internal gates that control what it remembers and forgets. A GRU only has two gates. This makes GRUs lighter and faster to train. They need fewer parameters to do almost the same task. In fact, GRUs often operate just as well as LSTMs, especially for audio jobs.

Because GRUs are simpler, they run faster and use less memory. If we wish to apply the model in a real-world application where speed is important, this may come in handy later. The CNN GRU model still receives the benefit of understanding how the call changes over time, but it does it more effectively than the LSTM version.

3.3.4 Purpose of Using Three Models

I designed three different models to explore how much information the system needs in order to recognize a Capuchinbird call. The primary goal was to determine whether the spectrogram's visual pattern was sufficient on its own or if the sound's timing improved the model's comprehension of the call. CNN simply takes into account the appearance of the spectrogram as a single image. In contrast, the CNN with an LSTM layer and the CNN with a GRU layer approach the spectrogram as a sequence, which allows them to read the sound from left to right and see how it varies across the clip. Using these three designs allows the project to look at the problem from different angles instead of relying on only one wcay of interpreting the audio. The goal is not to assume which structure is better but to explore how each architecture processes the same input. This approach gives a more complete view of how the model learns the characteristics of the Capuchinbird call.

3.4 Training Setup

The models were trained with the Adam optimizer at a learning rate of 0.001 and the binary cross entropy loss. I chose a batch size of 16 to keep the training consistent while also fitting well into memory. Each model could run for up to fifty epochs, but training ended after fifteen epochs when the validation loss no longer improved. I also saved the model state at the point where the validation AUC reached its best value so that the strongest version of each model was used in the later stages.

A learning rate reduction was added as well. If the validation score stayed the same for several epochs, the learning rate was dropped by half, with a lower limit of 1e-7 so it did not shrink endlessly. The main CNN architecture was built with three groups of layers that used sixteen, thirty two, and sixty four filters. Each group completed a three-by-three convolution, batch normalization, dropout, and ReLU activation. Dropout was set at 0.3 in the convolution blocks and 0.5 in the thick layer. The dense layer contained one hundred and twenty eight units before the final output.

Data augmentation played an important role during training. About half of the training clips received some form of change. These included pitch shifts of up to two semitones, speed changes of roughly twenty percent, small amounts of random noise with a standard deviation of 0.005, and slight gain adjustments in the range of 0.8 to 1.2. The validation set was kept clean so that the models were tested on inputs that were not altered. The data was divided using an eighty to twenty split, keeping the label balance similar in both sets.

3.5 Sliding Window Detection Pipeline

After the models were trained on three second clips, the next step was to use them for detecting calls inside full length forest recordings. I separated each of these recordings into overlapping windows because they can last for several minutes. Each window was three seconds long, and the window went forward by one second at a time. Every window went through the identical processing processes used in training, which included resampling and turning the audio into a spectrogram.

The model produced a probability for each window, showing how likely it was that the window contained a Capuchinbird call. These probabilities were turned into binary predictions based on a chosen threshold. A single call might appear in several windows in a row, so I grouped consecutive positive predictions and counted them as one event. This prevented the system from counting the same call multiple times and made the final call estimate more realistic.

3.6 Evaluation

To evaluate the models, I used two kinds of checks. The first check focused on the short clips used during training and validation. I computed common classification measures for these clips, including accuracy, precision, recall, F1 score, and AUC. These values helped show how well the models learned to separate Capuchinbird calls from other sounds.

The second check involved running the models on long forest recordings using the sliding window approach. The system produced a set of predictions across the entire recording, and the grouped detections gave an estimate of how many calls were present. This part of the evaluation helped show how the models behave in recordings that contain a variety of natural sounds.

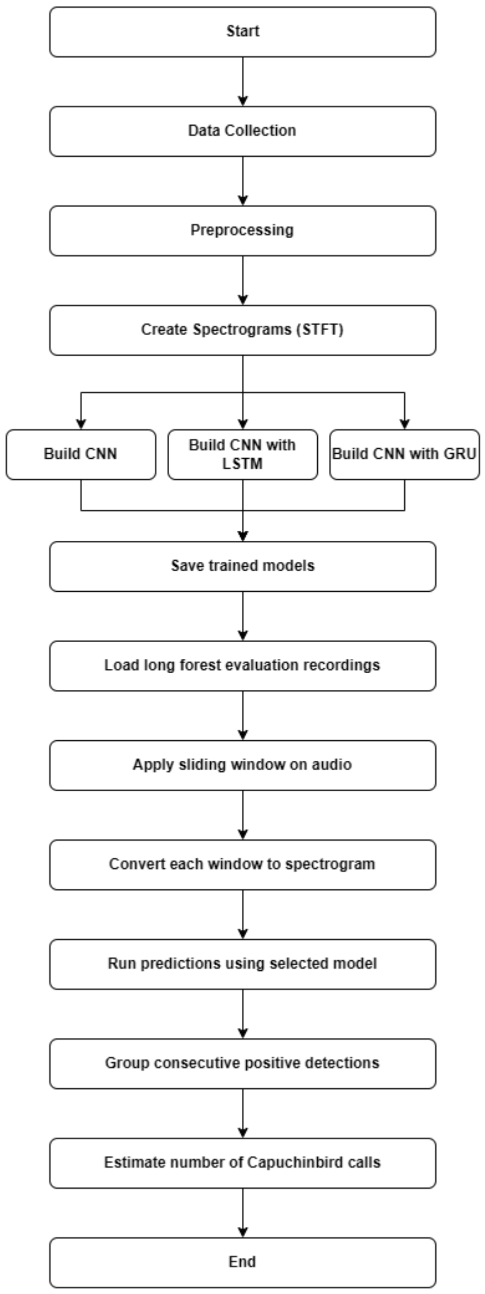
Start
Data Collection
Preprocessing
Create Spectrograms (STFT)
Build CNN
Build CNN with LSTM
Build CNN with GRU
Save trained models
Load long forest evaluation recordings
Apply sliding window on audio
Convert each window to spectrogram
Run predictions using selected model
Group consecutive positive detections
Estimate number of Capuchinbird calls
End

Figure 3.1: Methodology flow

CHAPTER 4
RESULTS AND DISCUSSIONS

4.1 Performance

To compare the three architectures fairly, I measured their accuracy, precision, recall, F1 score, and AUC on the validation clips. The values are listed in the table below and give a quick overview of how well each model identified Capuchinbird calls.

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**AUC**
CNN	0.9938	0.9773	1.0000	0.9885	1.0000
CNN LSTM	0.8951	0.7955	0.8140	0.8046	0.9730
CNN GRU	0.9691	1.0000	0.8837	0.9383	0.9893

Table 4.1: Comparison of model performance metrics.

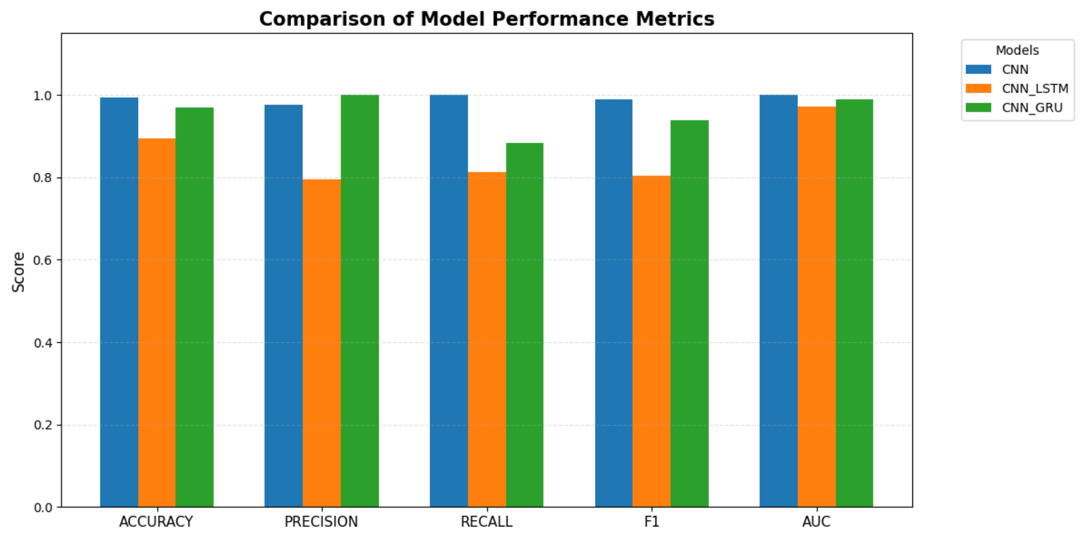

Figure 4.1: Model performance comparison

From these results, the CNN model clearly performed the best across almost all measures. The GRU model also did well, especially in terms of precision, but it missed more calls than the CNN. The LSTM model had the lowest overall performance. These numbers match what is seen in the comparison graph and help guide the choice of the final model.

4.2 Detection Process on Forest Recordings

After evaluating the models on the short clips, the next step was to apply the detection process to the long forest recordings. There were one hundred audio clips in this set, and each recording had different background conditions such as insects, wind, and distant animal sounds. These recordings were several minutes long, so they could not be sent into the model all at once.

To analyse them, each recording was divided into three second windows with a one second step. Every window was processed in the same way as the training clips, including resampling and conversion to a spectrogram. The model produced one prediction for each window, showing how likely it was that a Capuchinbird call was present.

Since one call can appear in more than one window, the positive predictions often came in groups. These groups were combined so that they counted as a single call event. This approach made it possible to turn the raw window predictions into a clear estimate of how many calls were present in each forest recording.

4.3 Final Model Selection

After comparing the three models, the CNN model was selected as the final one for the full detection stage. In the early tests on longer audio, it demonstrated the most consistent behavior and performed best on the short samples. The precision score indicated that it made very few false alarms, while the high recall score indicated that it was less likely to miss actual calls. These qualities made it a suitable choice for analysing the one hundred forest recordings.

Once the CNN model was selected, it was used to process all of the forest clips using the sliding window method described in the previous section. The model gave a prediction for each window, and these predictions were grouped to form single call events. This produced a clear count of Capuchinbird calls for every forest recording. The grouped detections were more consistent with the way human listeners identify calls, and they provided a simple and practical way to estimate call activity across all recordings.

4.4 Results

The bar chart displays the number of detected calls for each forest recording. Most files had only a few calls, while some contained much higher activity. The pie chart summarizes this pattern by showing how many recordings had no calls, low call activity, or high call activity. These two figures give a clear view of how the detections were spread across the one hundred recordings.

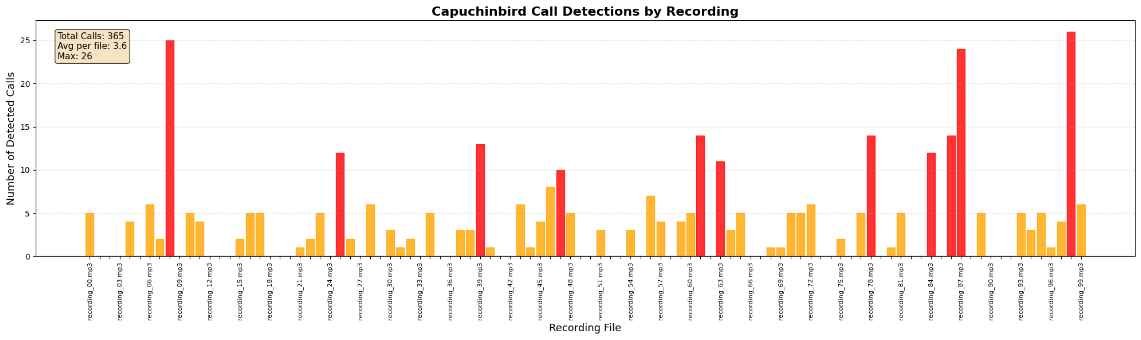

Figure 4.2: Call Detections by Recording

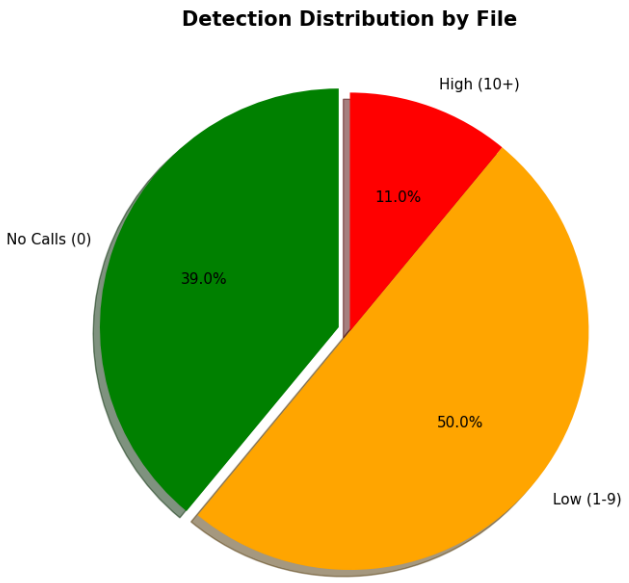

Figure 4.3: Detection Distribution by File

CHAPTER 5
CONCLUSIONS

5.1 Summary

This project explored automatic detection of Capuchinbird calls using spectrogram based models. The CNN model gave the strongest performance and was able to detect calls in the forest recordings with good consistency. The sliding window method worked well for converting raw predictions into clear call events. The final results showed that call activity varies widely across the one hundred recordings, and the system was able to capture these patterns in a reliable way.

5.2 Contributions

This work shows that a simple CNN can perform well for bird call detection when the preprocessing is consistent and the audio is transformed into a clear visual form. The project also provides a full pipeline from data preparation to model training and detection on long recordings. A limitation is that the model was trained on a limited number of labelled clips, and background noise in some recordings still affects sensitivity. Even with these challenges, the system gives a practical way to estimate call activity across many files.

5.3 Future Research

Future work can focus on expanding the training set with more diverse examples and exploring ways to separate overlapping sounds. Another direction is to test models that work directly on raw audio without converting it into spectrograms. It may also be useful to study how the system behaves with recordings from different locations or seasons. Since the LSTM and GRU models did not offer clear advantages here, they seem less promising for this task unless more temporal structure is added to the data.

REFERENCES

[1] J. Salamon and J. P. Bello, “Deep convolutional neural networks and data augmentation for environmental sound classification,” *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7829341

[2] S. Kahl, C. M. Wood, M. Eibl, and H. Klinck, “Birdnet: A deep learning solution for avian diversity monitoring,” *Ecological Informatics*, vol. 61, p. 101236, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574954121000273