Mini Project Report

on

Avian Soundscape Analysis: Machine Learning for Forest Bird Monitoring

Submitted by

B.M. Dhiraj 21BDS009
M. Virinchi 21BDS039
M. Sai Tarun 21BDS040
Prerana Bhat 21BDS048

Under the guidance of

Dr. Sunil Saumya
Head of Department, Data Science and Artificial Intelligence

DEPARTMENT OF DATA SCIENCE AND ARTIFICIAL INTELLIGENCE
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DHARWAD
26/04/2024

Certificate

This is to certify that the project, entitled '*Avian Soundscape Analysis: Machine Learning for Forest Bird Monitoring*', is a bonafide record of the Mini Project coursework presented by the students whose names are given below during 2023-24 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Data Science and Artificial Intelligence.

Roll No	Names of Students
21BDS009	B.M. Dhiraj
21BDS039	M. Virinchi
21BDS040	M. Sai Tarun
21BDS048	Prerana Bhat

Dr. Sunil Saumya
(Project Supervisor)

Contents

List of Figures

List of Tables

1 Introduction

Birds are important indicators of biodiversity health due to their ecological significance, sensitivity to environmental changes and wide distribution across various habitats. They provide valuable insights into ecosystem dynamics and serve as early warning systems for identifying disturbances or changes in ecological conditions.
According to the 2020 State of the World's Birds report by BirdLife International, one in eight bird species worldwide is threatened with extinction. It also highlights that over 40% of the bird species worldwide are experiencing population declines.
The conservation of avian biodiversity, especially in ecologically sensitive regions like Amazon Rainforest, Western Ghats of India, Madagascar etc. , is of paramount importance in the face of rapid anthropogenic pressures such as habitat modification and climate change. These regions harbour diverse ecosystems supporting large levels of biodiversity, including numerous endemic and endangered bird species. However, these ecosystems are increasingly threatened by habitat loss and climate changes, necessitating the use of advanced conservation tools and technologies for rapid monitoring of bird diversity.

Conducting traditional observer-based bird biodiversity surveys over large areas is expensive and challenging. In this context, the utilisation of machine learning (ML) for the automated detection and classification of bird species from audio recordings holds immense potential.
PAM (Passive Acoustic Monitoring) is a technique used to detect, record and analyze sounds in the environment without actively disturbing the area of organisms producing the sound. It involves deploying recording devices such as microphones, in natural habitats to capture audio over extended periods of time.
By leveraging PAM combined with ML-based analytical tools, conservationists can study more areas over shorter time periods, making it easier to understand bird population densities.

Our mini-project focuses on the Capuchin bird (Perissocephalus tricolour), a species native to humid forests in South America. Despite not yet being recognised as endangered, the Capuchin bird population has been experiencing a steady decline primarily due to habitat loss caused by deforestation.

In our work, we aim to utilise machine learning techniques to address two critical objectives :

1. Check whether capuchin bird sounds are present in that specific forest's audio recording.
2. Check how many capuchin bird sounds are present in that specific forest's audio and consequently help predict the presence of Capuchin bird population in forests and their population density in that specific forest.

2 Related Work

Avian monitoring, powered by machine learning (ML), represents a unique approach in wildlife conservation, detecting and classifying bird species through their vocals. Previous studies, such as the one by Stowell and Towsey (2014), have experimented with various preprocessing and feature extraction like MFCCs, Spectrograms and Mel Spectra and even used ensemble methods of learning like random forests for classifying avian audio samples. Stowell further experimented with bioacoustics coupled with Machine Learning in 2018 and Deep Learning in 2022.

In September 2020, Ming Zhong and team used machine learning and deep learning methods for bioacoustics analysis of around 24 species (including birds and amphibians).

In Cihun-Siyong Alex Gong, Chih-Hui Simon Su and their team tried using Deep Neural Networks and LSTM models for bioacoustic classification

In 2024, Cornell Lab of Ornithology floated a contest on Kaggle aiming to study about understudied Indian bird species, collecting a dataset of sounds from various bird species populating the Western Ghats region, which is a biodiversity hotspot.

3 Data and Methods

The Data:

There are 810 .wav files of which 217 are ‘Parsed_Capuchinbird_Clips’ and 593 are ‘Parsed_Not_Capuchinbird_Clips’. Each of these audio clips are one minute long and are totally binary in nature as their names suggest. It is clearly visible that there is a clear imbalance in the data and it posed challenges going forward into the model implementations.

Apart from this, there is a collection of forest audio clippings each 3 minute long, these clips contain both capuchinbird sounds and other forest noises. These clippings are the ones on which the models shall be tested on.

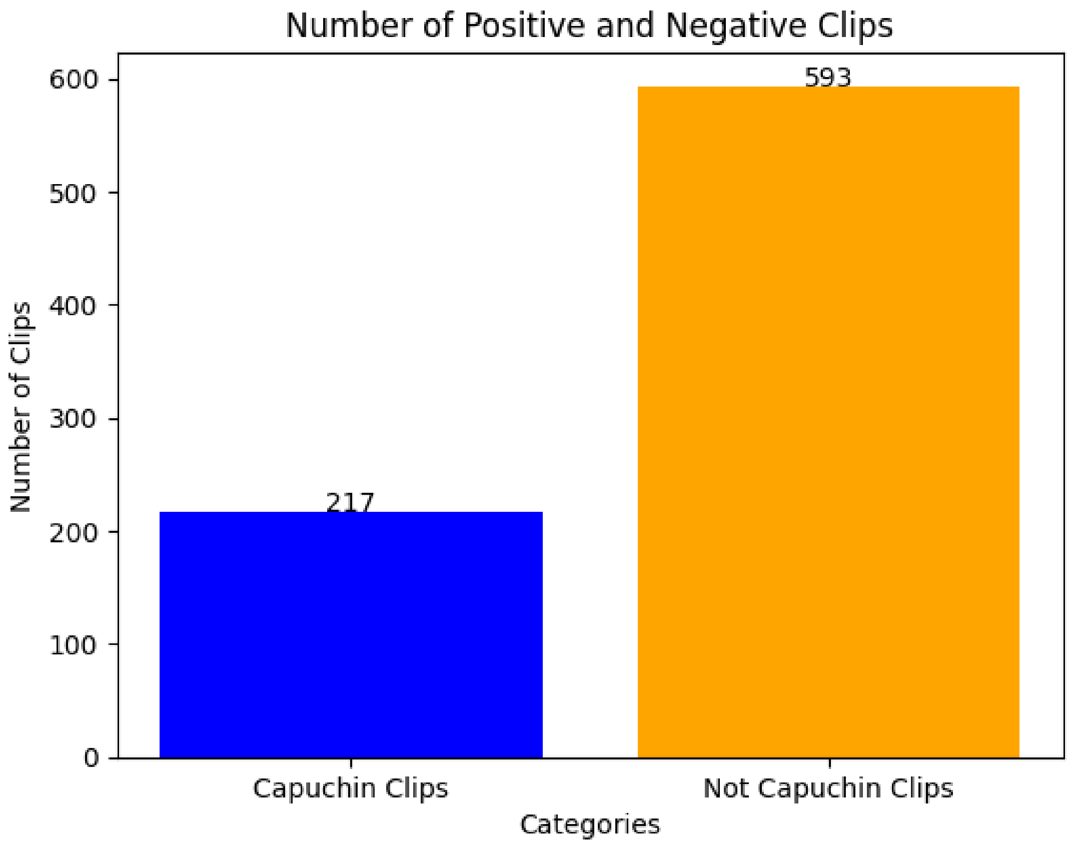


Figure 1 : The Data

Solution Architecture :

The following is a workflow diagram on the working of our solution. It gives an idea about the models used and flow of the approach.

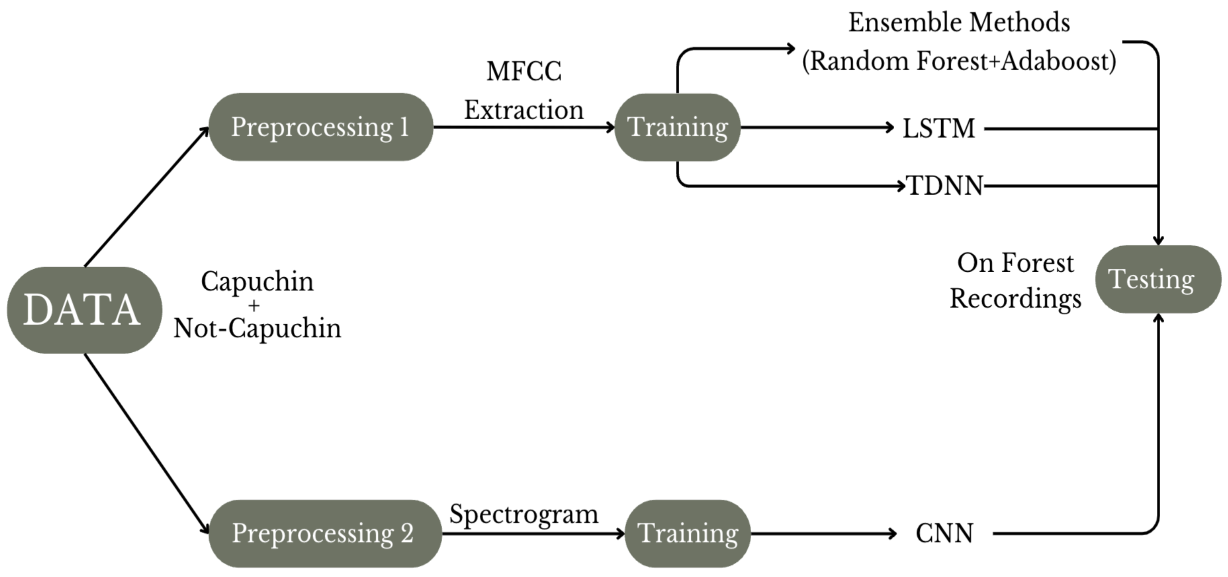


Figure 2 : Solution Architecture

Method 1: Ensemble Learning

Assigning Labels to the Data:

Before we do anything with the audio, we need to assign labels so the model knows what's what. The Capuchinbird clips get a label of "1," while the other clips get a label of "0."

Extracting Features from the Audio:

As we are done with the data analysis ,the next step is to extract the features and make our data fit into our model . For that we need to break the audio into manageable chunks and extract some features from them. For pre-processing we have used Mel-Frequency Cepstral Coefficients (MFCCs). These are popular in audio analysis because they capture the unique characteristics of sound. Here's what was done

1. **Load the Audio:** We used librosa to read the .wav files, keeping the original sample rate.
2. **Segment the Audio:** We divided each clip into smaller units of three seconds. This makes it easier to process and analyze.
3. **Extract MFCC Features:** For each segment, we computed the MFCCs and took the mean of those values. This gives us a pretty good idea of the segment's overall sound characteristics.

The parsed_capuchinbird clips and parsed_noncapuchin clips have an average length of 3sec . The idea is to divide the test audio (3 mins) into 3sec segments and classify them into capuchin bird and non capuchin bird voices based on the training, dividing the data into optimal chunks can help us get meaningful MFCC features which in return helps to identify the patterns within the segments.

Possible Solution to address the imbalance in the data with random forest :

- When training the model like random forest we adjusted the class weights to give more importance to the minority class. This encourages the model to pay more attention to the Capuchin class during training.

Building a Classifier:

With the features in hand, we need to build a machine-learning model that can tell the difference between Capuchin and non-Capuchin segments. For this, I went with a Random Forest classifier. It's robust and works well with audio data. The steps here are straightforward:

1. **Split the Data:** The features and labels are divided into a training set and a testing set, with 80% for training and 20% for testing.
2. **Train the Model:** A Random Forest model is set up with 100 trees and trained with the training data. The more trees, the better it generally performs.

Random Forest :

Random Forest is a machine learning algorithm based on the ensemble learning approach, where multiple models are combined to achieve better performance

Bootstrap Sampling:

To create diversity among the trees, Random Forest uses a technique called "bagging" (bootstrap aggregating). In bagging, each tree is trained on a random subset of the original data, with replacement. This means that the same data point can be selected more than once for a tree's training set, resulting in unique subsets for each tree.

Random Feature Selection:

In addition to bootstrapping the data, Random Forest introduces randomness in the feature selection process. When a decision tree splits at a node, it considers a random subset of features to determine the best split, rather than evaluating all features. This additional randomness makes the trees less correlated with each other, leading to a more robust ensemble.

Each decision tree is grown by splitting the training data at each node based on a specific feature and threshold that best separates the classes (in a classification task) or predicts a continuous value (in a regression task).

Hyperparameters:

Random Forest has several parameters that can be tuned to adjust its behaviour, including:

- Number of Trees: The total number of decision trees in the forest. More trees generally improve stability and accuracy.
  Here in our model we have used 100 decision trees .
- Max Features: The number of features considered at each node split, which influences the diversity among trees.
  Max_features is set to 'sqrt'.
- Max Depth: The maximum depth of each tree. Limiting depth can prevent overfitting.
  Max depth when set to 5, accuracy is 97.023 , for 2 it is 90.127
- Min Samples Split/Leaf: Minimum number of samples required to split a node or be in a leaf, impacting the size of the trees.

During training, each tree in the forest learns to classify the segments based on the provided labels .

Since the trees are trained on different subsets of the data and with different feature sets, they collectively capture a wide range of patterns and relationships within the audio data.

When making predictions on new data , each tree in the forest predicts whether the given segment is a Capuchinbird or not based on its learned splits.

Adaboost :

AdaBoost (Adaptive Boosting) is an ensemble learning technique designed to create a strong classifier by combining several weak learners, typically shallow decision trees.

Architecture

- **Ensemble Approach**: AdaBoost combines multiple weak learners to form a stronger ensemble classifier.

- **Weak Learners**: In AdaBoost, the base estimator is often a shallow decision tree (stump), typically with a maximum depth of 1. The weak learners used in our model are decision tree stumps

- **Boosting Process**: In each iteration, AdaBoost assigns weights to training samples. Initially, all samples have equal weights, but after each iteration, the weights of misclassified samples increase. This allows subsequent trees to focus more on challenging cases.

Hyperparameters

- **Base Estimator**: Defines the type of weak learner used (commonly a decision tree with limited depth).
- **Number of Estimators**: Indicates the number of weak learners in the ensemble. A larger number of estimators can lead to a more accurate model. In our model the estimators are set to 50
- **Learning Rate**: Controls the magnitude of weight adjustments across iterations. A lower learning rate means more gradual changes, while a higher rate leads to more dramatic adjustments. Learning rate is set to 1 in our model

Although experiments were done by taking other values , the above mentioned ones were giving the best results.

Prediction Mechanism

- **Weighted Voting**: Predictions are made by aggregating the outputs of all weak learners, using their weights as a basis for the voting mechanism. The final prediction is determined by the weighted majority vote.
- **Bias and Variance**: AdaBoost aims to reduce both bias and variance by iteratively refining the model's focus on challenging samples.

Method 2: LSTM

Long Short-Term Memory (LSTM) networks, a variant of RNNs, have emerged as a powerful tool for modelling sequential data due to their ability to capture long-term dependencies while mitigating the vanishing gradient problem. In the context of audio classification, where the temporal dynamics of sound play a crucial role, LSTM networks are well-suited for capturing the temporal patterns present in audio signals. By leveraging the memory cells and gating mechanisms inherent in LSTM units, the model can effectively learn and recognize the complex temporal structures underlying Capuchinbird calls.

Data Preprocessing:

The audio data consists of recordings from two categories: Capuchinbird calls and non-Capuchinbird sounds. The preprocessing steps include:

Loading audio files using the librosa library.

Extracting Mel-frequency cepstral coefficients (MFCCs) as features, which capture the frequency characteristics of the audio.

Flattening the MFCC features and computing their mean to obtain a feature vector for each audio sample.

Assigning labels based on the folder structure: 1 for Capuchinbird calls and 0 for non-Capuchinbird sounds.

Data Splitting:

The dataset is split into training, validation, and test sets, in a peculiar fashion so as to deal with the data imbalance mentioned previously. The split is as follows:

- Training set: 150 Capuchinbird samples and 400 non-Capuchinbird samples.
- Validation set: 33 Capuchinbird, 100 non-Capuchinbird samples.
- Test set: 37 Capuchinbird, 93 non-Capuchinbird.
- Labels are one-hot encoded for compatibility with the model.

Model Architecture:

The classification model architecture comprises layers of Bidirectional Long Short-Term Memory (LSTM) units, followed by fully connected layers with dropout regularization. The model summary is as follows:

- Bidirectional LSTM layer (128 units) with dropout (0.05).
- Two dense layers with ReLU activation (128 units each) and dropout (0.3).
- Two additional dense layers with ReLU activation (64 units each).
- Final dense layer with softmax activation for classification into two classes.

Model Training:

The model is trained using the Adam optimizer with a learning rate of 1e-4 and categorical cross-entropy loss. Key training parameters include:

- Batch size: 3
- Epochs: 30
- Class weights are adjusted to handle class imbalance.
- Early stopping is implemented to prevent overfitting, monitoring validation loss with patience set to 3.

The trained model is evaluated on the test set formed from the split and also on the forest recordings.

Method 3 : CNN

In this approach spectrograms are used for pre-processing contrary to the usage of MFCC's for the same in previous methods. Most of the given data is in .wav form clips that are 3 seconds long, the first thing we do after importing necessary libraries, is load and preprocess the audio data. We ensure that the audio is in a mono-channel format and resample the audio from 44,100 Hz (original sample rate) to 16,000 Hz to ensure uniformity and improve computational efficiency.

For better visualization, we plot the waveform for one random sample from Capuchin and Not Capuchin folders each. This plot shows the variation of amplitudes (y-axis) with time (x-axis) for a Capuchin call audio (Blue) and Not Capuchin audio (Orange)

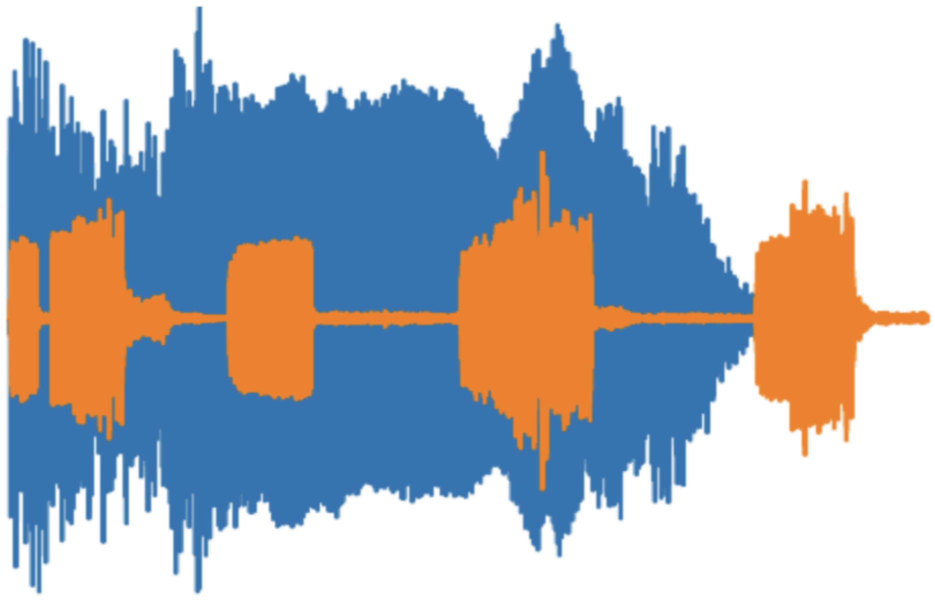


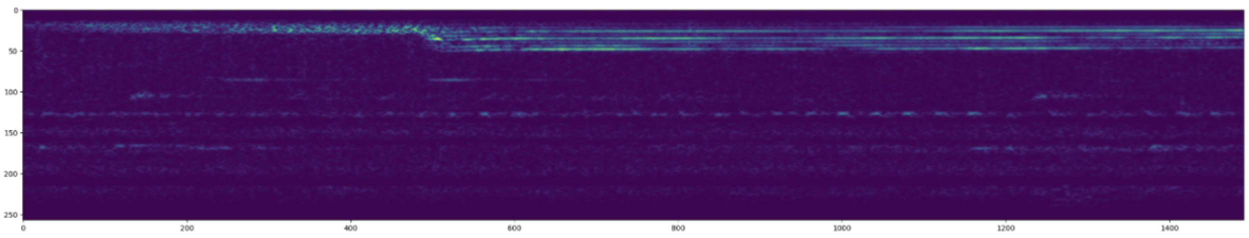
Figure 3 : Audio waveforms for capuchin and not-capuchin

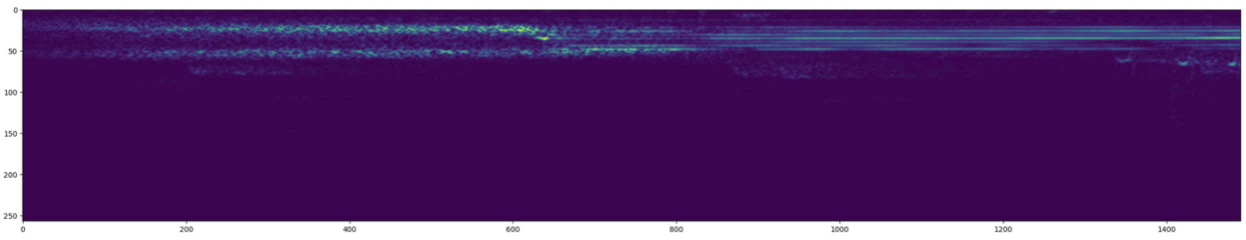
Next, we prepare the dataset for training and testing by defining paths for positive and negative clips and concatenating them. We link the files in these directories to the .wav format and add their respective labels, which are in terms of binary classification i.e. 0 for negative and 1 for positive.

We complete the pre-processing steps by converting the acquired waveforms into a spectrogram. These visualized audio signals in the form of spectrograms (X axis - time, Y axis - frequency) will be used by the deep learning model to analyze and interpret the results accordingly.

We observe that the spectrograms for Capuchin audio are significantly different from spectrograms for not Capuchin audio. Here are few samples from both the classes to prove the same

Capuchinbird :





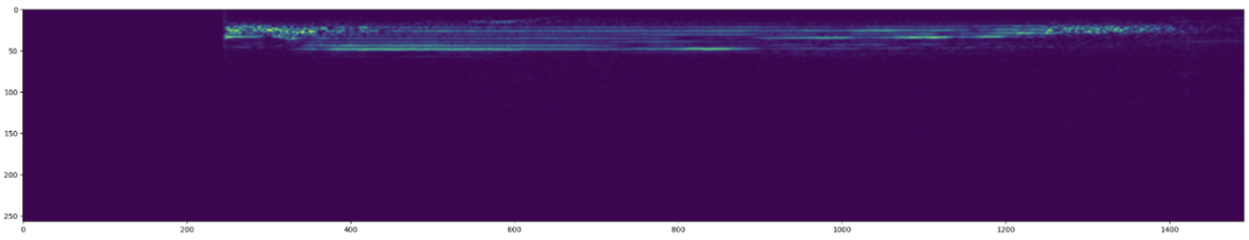
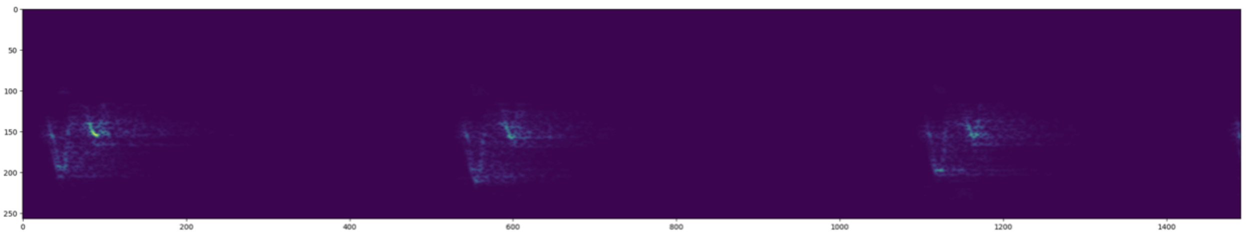


Figure 4 : Capuchin spectrograms

Not Capuchinbird :



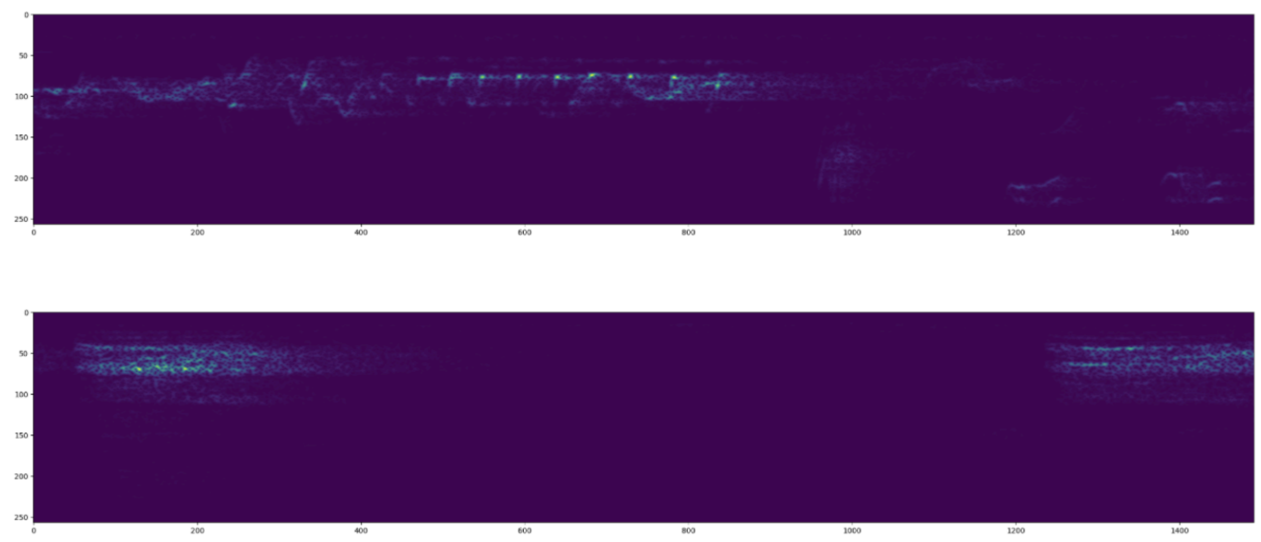


Figure 5 : Not capuchin Spectrograms

Next, we continue with building the Deep Learning model. For partitioning the training and testing samples, we divide the data into batches :

Training samples = 567
Test samples = 243

We build a Convolutional Neural Network (CNN) model designed for a binary classification task. The input to the model is a 3D tensor with dimensions (1491, 257, 1).

Dimension 1 (1491) represents the number of time frames in the spectrogram for each sample. Dimension 2 (257) represents the frequency bins / frequency points in the spectrogram. Dimension 3 (1) represents the number of channels in the spectrogram.

The model starts with two convolutional layers, each followed by a max-pooling layer. The convolutional layers extract relevant features from the input data, while the max-pooling layers reduce the spatial dimensions of the feature maps, effectively downsampling the input. The use of convolutional layers is well-suited for processing spatial or temporal data, such as images or audio signals, as they can capture local dependencies and patterns.

After the convolutional and max-pooling layers, the model flattens the feature maps and passes them through a fully connected layer with 128 units and a ReLU activation function. This dense layer aims to learn higher-level representations from the extracted features. Finally, the model ends with a single output unit with a sigmoid activation, which can be interpreted as the probability of the input belonging to the target class (e.g., the capuchin bird). We give a threshold to the output probability and classify it according to the resultant probability.

We include L2 regularization on the convolutional and dense layers to help prevent overfitting by adding a penalty term to the loss function, encouraging the model to learn more generalizable features.

Method 4: TDNN

Time-delay neural networks (TDNNs) are a sort of neural network design that are often used for sequence modelling applications, especially in speech recognition and natural language processing. TDNNs are particularly well-suited for speech recognition tasks.

TDNN has certain key features which make it stand out from the rest, which are:

1. **Temporal Modelling:** Speech signals are sequences of data, and the temporal relationship among successive frames is critical for interpretation. TDNN is designed to capture and understand the temporal dependencies of input audio signals by applying convolutional filters within a context window. This allows the neural network to understand temporal patterns and correlations while also learning relevant properties across multiple time periods.
2. **Shift Invariance:** Shift invariance in TDNN refers to the network's capability to recognize patterns in input data regardless of the positions they appear in the sequence. In speech recognition, this property is essential because the timing of phonemes and words may vary with the accent, pitch, and other external factors. The TDNN can effectively learn features that are sensitive to temporal shifts, resulting in better generalization performance.
3. **Hierarchical Feature Learning:** TDNN has a hierarchical architecture that allows the network to incrementally learn the abstract representations of the input data. Lower layers of the network capture low-level Temporal Relationships such as spectral characteristics... While the Higher layers have the ability to learn more complex and wider temporal relationships. Like every successive layer acts as the enriched form of Input for the Next layer.

TDNN Architecture:

The key element of a TDNN is the convolutional layer with time delays. This enables the network to efficiently collect local context in the speech signals while maintaining some degree of shift-invariance.

1. **Input Layer:**
   This initial layer receives the preprocessed Audio data, represented as a sequence of feature vectors (like MFCCs, iVectors, and so on) as a frame. Whereas each vector represents a frame of data at a certain time step, these features capture aspects of the audio signals at each time step, such as spectral content and energy.

2. **Convolutional Layer:**
   - The Convolutional Layer with Time Delays is the core of the TDNN architecture. It uses a convolutional filter that slides along the input stream. However, unlike a typical DNN, this filter can access the input data at various time delays.
   - Dilation is used in convolution to establish time delays and also capture temporal relationships over longer periods while not dramatically increasing the number of parameters or computing complexity.
   - Using variable dilation rates within the Stacked convolutional layer, the TDNN will capture the local temporal context at different scales. Smaller dilation rates concentrate on very close time steps, whereas higher rates consider information further down the sequence. This helps the network to learn a more detailed representation of the speech signal.

   From the stack of convolutional layers, The first layers usually focus on capturing fine-grained local features, then the successive layers might learn more complex patterns that emerge from the combination of these local features.
   This layer contains a non-linear activation layer. This function introduces non-linearity into the network, allowing it to model complex relationships between the features.

3. **Output Layer:**
   The Final Layer provides the network's output, with the softmax activation function which provides the likelihood probabilities for different classes in a classification task.

Working Mechanism of TDNN :

1. **Feed Forward Pass:**
   Similar to Classic Feed Forward Neural Networks, input data is forwarded through the network layer by layer, while additionally capturing frames based on the context window and transforming them into an enriched input representation for the next layers. The input Data is passed through the Convolutional Layers where, each convolutional layer with time delays applies its filters to the sequence, considering the delayed inputs based on the dilation rates.which captures the local temporal patterns based on the fixed-size window then followed through the activation function which introduces non-linearity and makes the model capable of learning and understanding complex patterns.

2. **Compute Loss:**
   The model's prediction is compared to the actual capuchin bird label (ground truth) from each recording. the cost binary cross entropy function is used for classification to calculate the error between the predicted and actual probabilities.

3. **Backpropagation:**
   - Backpropagation involves calculating the gradients of the loss function concerning the model parameters, which include weights and biases. Backpropagation in TDNNs begins at the output layer and proceeds backward through the network. Gradients are computed layer by layer, propagating errors from the output layer back to the preceding layers.
   - The gradients in each layer are used to update the parameters via optimization methods like stochastic gradient descent (SGD), Adam.
   - The backpropagation process helps the TDNN refine its ability to capture context within the bird vocalizations and differentiate between the calls of Capuchin Bird and Other noises.

4. **Gradient Descent:**

The gradients computed by backpropagation are then used by an optimization algorithm, such as stochastic gradient descent (SGD), to update the model parameters in the direction that minimizes the loss function. The update rule for the parameters involves subtracting a fraction of the gradient, controlled by the learning rate hyperparameter with make the model better to classify the bird calls and other noises with every epoch.

5. **Repeat:**
    - During the feedforward pass, the input data is transformed through the convolutional layers with time delays, capturing local temporal patterns. The backpropagation process then refines the network's ability to differentiate between classes by propagating gradients back through the layers.
    - By iterating through the forward pass, error calculation, and backpropagation with weight updates, the TDNN steadily improves its ability to map input features (bird vocalizations)

The model has been designed with the stacked convolutional layers at the beginning then followed by the flattening layer

This architecture design is for a classification task. I.e classifying Capuchin Bird calls from other noises. The convolutional layers extract relevant features from the input data, Flatten layer simplifies the processed data structure by converting it into a 1D vector, while Dense layers process this flattened data to make decisions Dense layers are crucial for learning complex functions and improving classification accuracy and perform final classification.

4 Results and Discussions

Results from Ensemble Learning (Random Forest Classifier + Adaboost):

Random Forest Results:

	Precision	Recall	f1-score	support
Capuchinbird	0.98	1	0.99	123
Not Capuchinbird	1	0.96	0.98	51
accuracy			0.99	174
Macro avg	0.99	0.98	0.99	174
Weighted avg	0.99	0.99	0.99	174

Table 1 : Random forest results

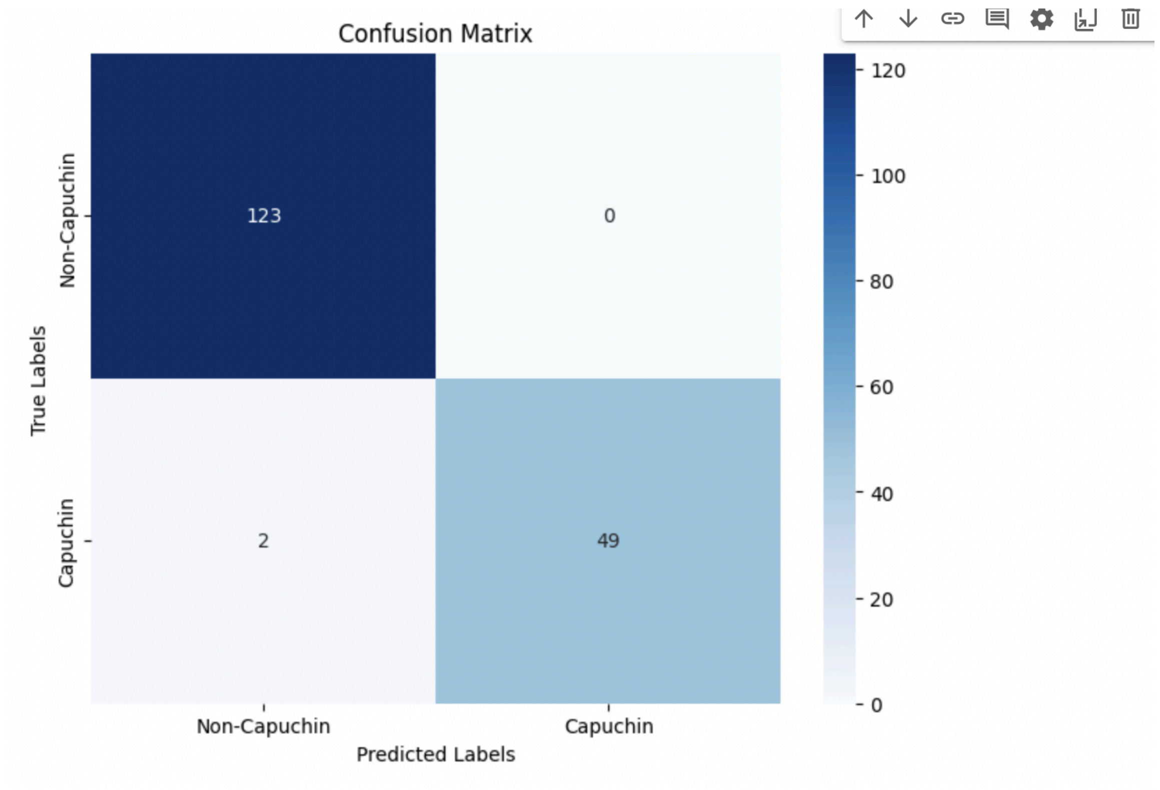


Figure 6 : Confusion matrix - Random Forest

Adaboost results

	Precision	Recall	f1-score	support
Capuchinbird	0.99	1	1	123
Not Capuchinbird	1	0.98	0.99	51
accuracy			0.99	174
Macro avg	0.99	0.98	0.99	174
Weighted avg	0.99	0.99	0.99	174

Table 2 : Adaboost results

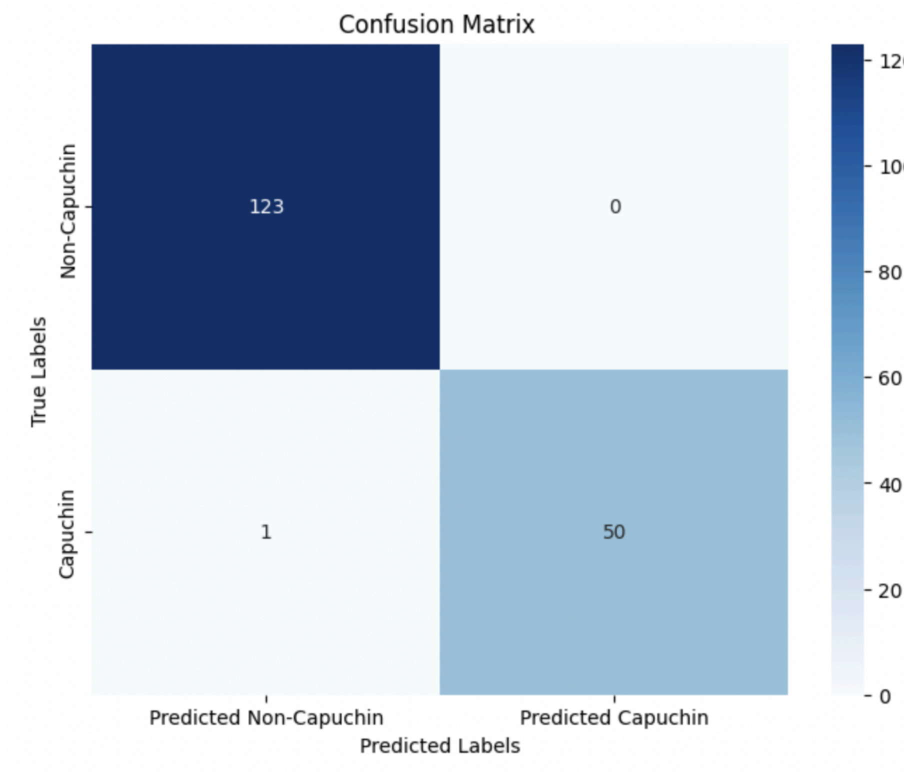


Figure 7 : Confusion matrix - Adaboost

Results on the forest recordings :

During evaluation of both the models, we noticed that while some clips were classified correctly, others were frequently misclassified.

Potential Causes of Inconsistency :

- **Quality of Clips**: Noisy or distorted clips may lead to incorrect feature extraction.
- **Data Imbalance**: An imbalance in the number of samples across different classes could bias the model.

Results from Long Short Term Memory (LSTM)

Upon the completion of training and validation we obtained following plots for loss function and accuracy:

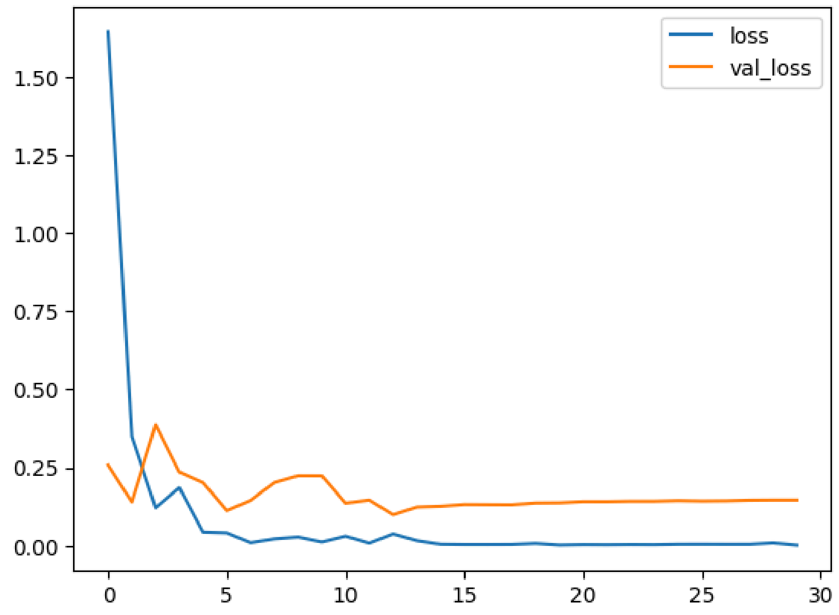


Figure 8 : LSTM Train and Val loss

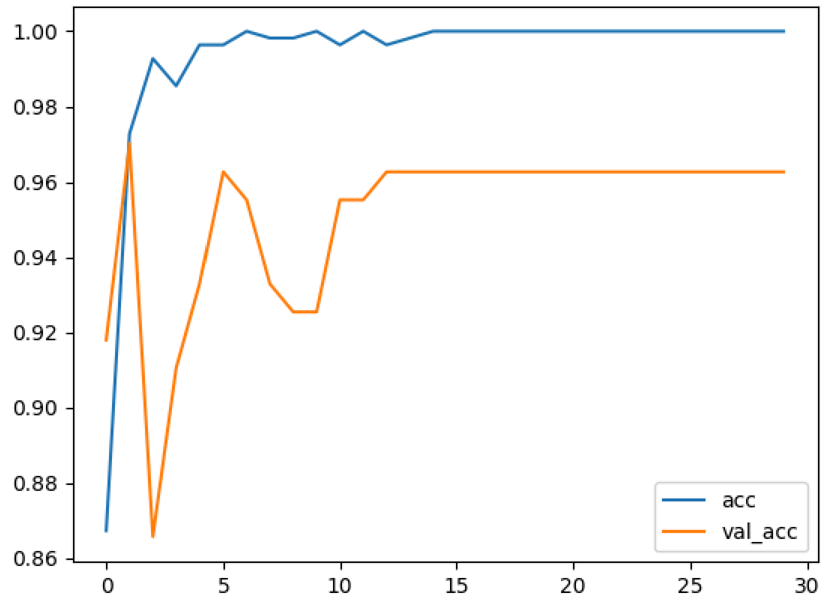


Figure 9 : LSTM Train and Val accuracy

Furthermore LSTM method yielded a test accuracy of 82%, the classification report and confusion matrix of the same are given below:

	Precision	Recall	f1-score	support
Capuchinbird	0.8889	0.8602	0.8743	93
Not Capuchinbird	0.6389	0.6970	0.6667	33
accuracy			0.8175	126
Macro avg	0.7639	0.7786	0.7705	126
Weighted avg	0.8234	0.8175	0.8199	126

Table 3 : LSTM results

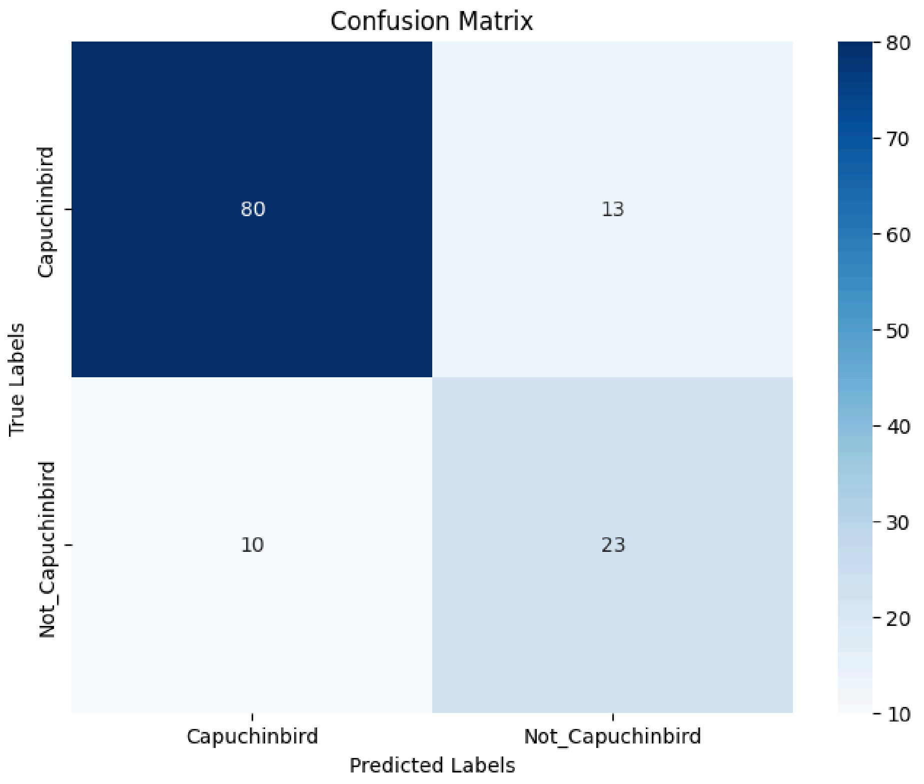


Figure 10 : Confusion matrix - LSTM

Results on the forest clippings:

The model was run on forest clippings:

- Detecting presence of capuchin bird
- Finding the number of capuchinbird sounds if present
- Giving time stamps of the capuchinbird calls

While there are some misclassifications which can be owed to aforementioned data imbalance and the 82% testing accuracy, we get mostly accurate predictions, consistent with the results of other models as well.

Results from CNN:

On training the model for 10 epochs and validating it, we got the following results after 10th epoch:

	Loss	Precision	Recall
Training	0.1464	0.9944	0.9942
Validation	0.1123	1.0000	1.0000

Table 4 :CNN results

The validation accuracy being 100% can be attributed to the imbalanced nature of the dataset which we have mentioned previously.

Plot for loss : Train (red) and validation (blue)

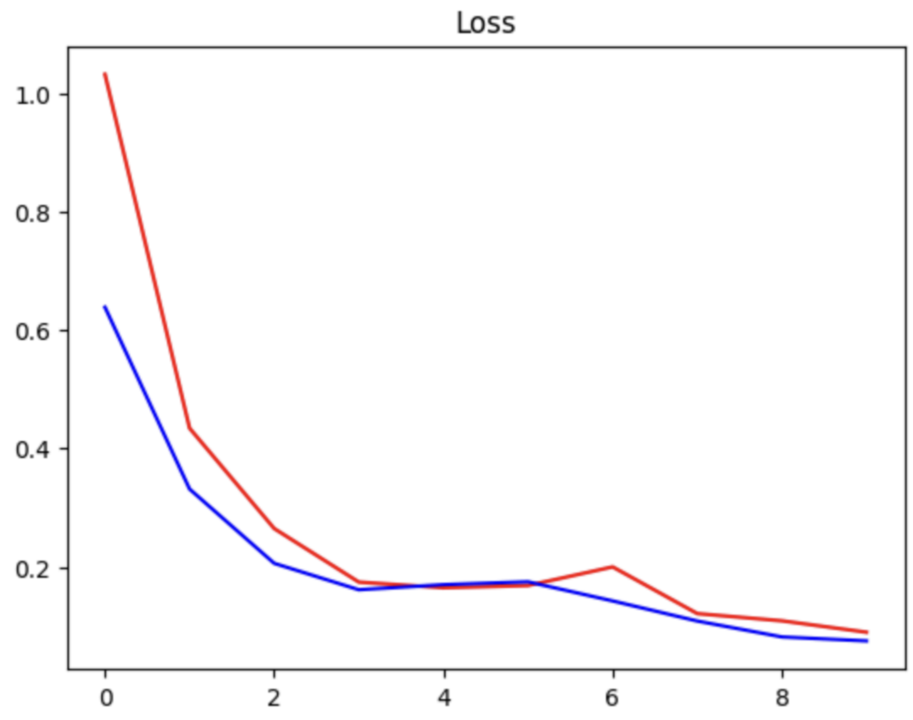


Figure 11 : CNN Train and Val loss

Plot for precision : Train (red) and validation (blue)

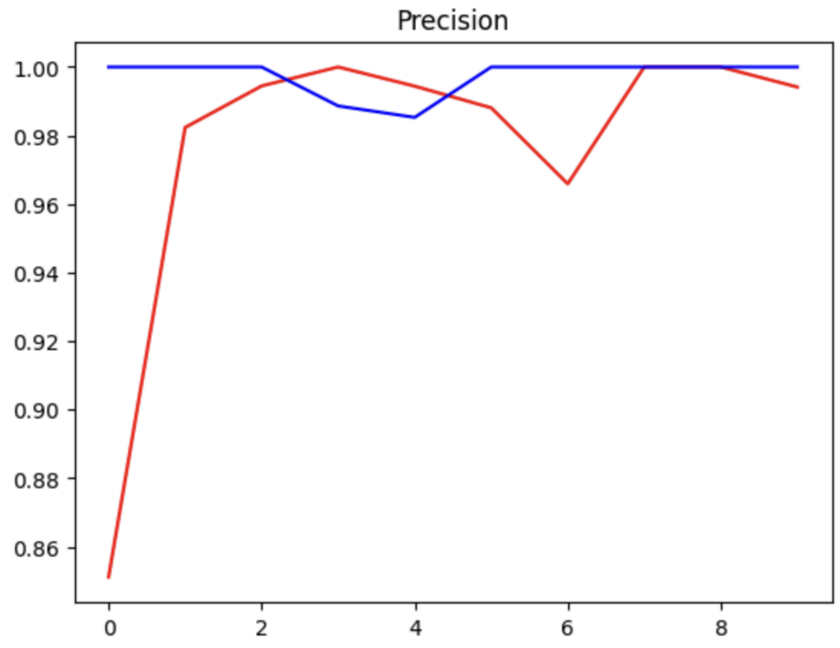


Figure 12 : CNN Train and Val precision

Plot for recall : Train (red) and validation (blue)

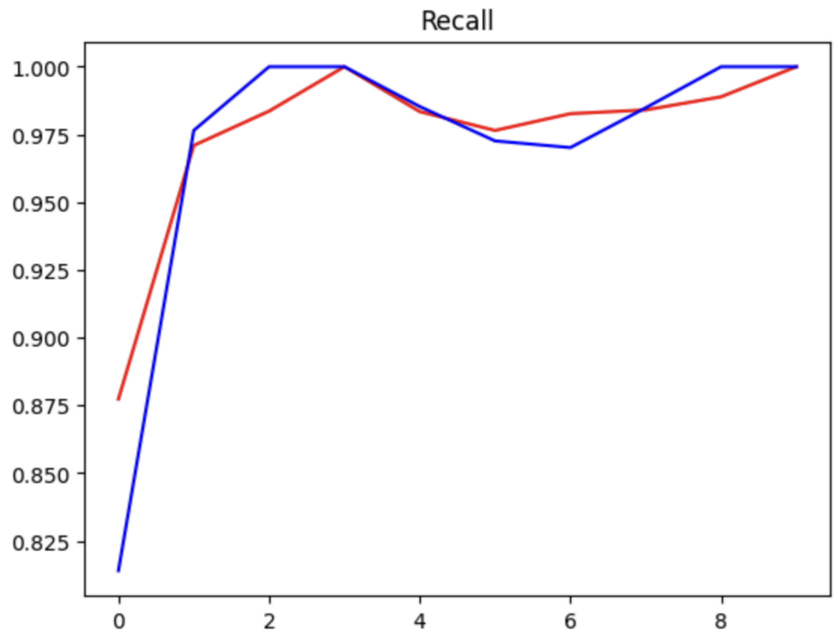


Figure 13 : CNN Train and Val recall

We test the model on a set of 16 randomly generated samples and assign the class as 1 idf prediction is > 0.5 and 0 otherwise.

The next part is making predictions on the files in the forest recordings directory.

As mentioned previously, each clip in this directory is about three minutes long in the form of mp3 files, and we intend to capture the call density (number of calls) in each clip. We carry out this process as follows:

- First we convert the mp3 format input into tensors and resample it to 16 kHz single-channel audio.
- We segment the audio into time-series slices with a sequence length of 48000 samples and a stride of 48000 samples, forming a dataset with a batch size of 64.
- These audio slices are preprocessed and the trained model is used to predict on these preprocessed audio slices.
- We run the following process for all the files in the forest recordings and obtain the computed results. The results contain clips of zeros and ones where the total ones are outputted to compute the overall score of the clips.

We finish by storing the post-processed results to a CSV file. Each row of the CSV represents a file recording, with columns for the recording filename and the count of capuchin calls. Like other models there are few misclassifications, other than those we get mostly accurate predictions, consistent with the results of our other methods.

Results from Method TDNN:

	Precision	Recall	f1-score	support
Capuchinbird	0.89	0.36	0.52	44
Not Capuchinbird	0.81	0.98	0.89	118
accuracy			0.81	162
Macro avg	0.85	0.67	0.70	162
Weighted avg	0.83	0.81	0.79	162

Table 5 : TDNN results

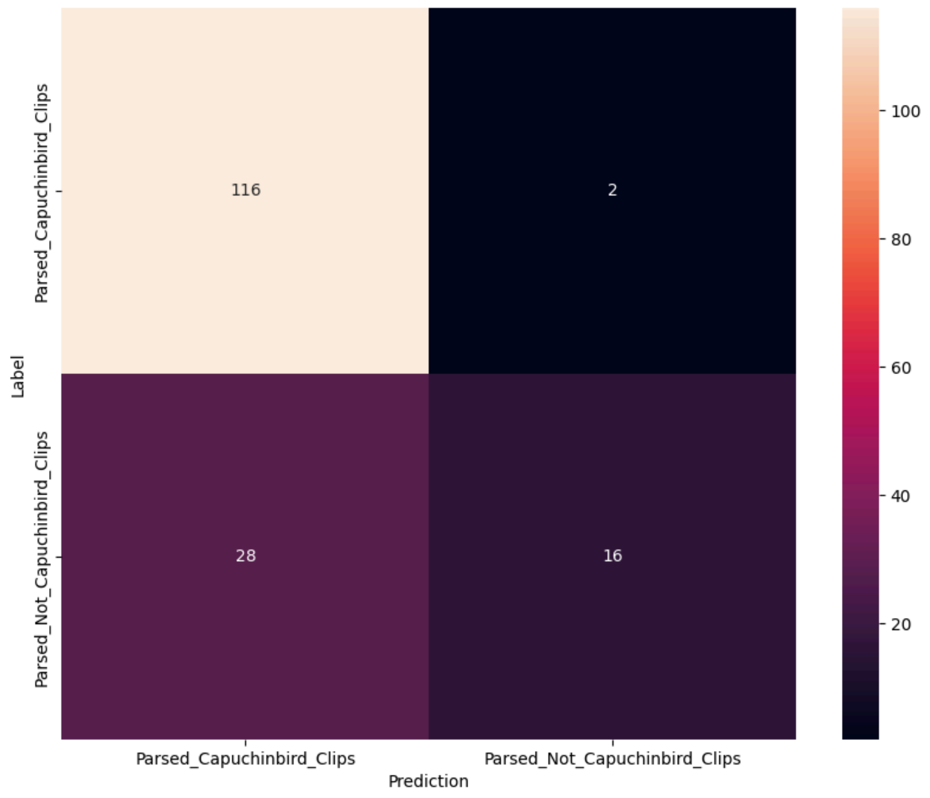


Figure 14 : Confusion matrix - TDNN

When the model is tested on the 3 minute forest recordings clips :

- Each recording was segmented into 72 segments, each 2.5 seconds long.
- Out of the 72 segments, 16 were classified as "BIRD CALL" segments, and 56 were classified as "NOISE" segments.
- The model was able to successfully identify 16 segments containing Capuchin Bird calls with the exact time of occurrences within the 3-minute forest recording.
- The remaining 56 segments were classified as "NOISE", which could include other environmental sounds, such as birds chirping, crickets sounds, crow, dove sounds.. etc.
- The distribution of the detected "BIRD CALL" segments suggests that the model is able to capture the temporal patterns and context within the bird vocalizations, as the calls are detected at various time points throughout the recording.

We get largely accurate predictions, consistent with the results of other models, but we can see some misclassifications, like sometimes the call starts at one segment and ends in another due to which sometimes the calls number can be greater than actual count.

5 Conclusion

Our project demonstrates the feasibility of using machine learning to detect bird species presence and estimate their population densities in forests, focusing on the Capuchin bird in South America. While we encountered challenges like data imbalance and audio variability, our models showed promise in identifying Capuchin bird calls in forest recordings.

Despite this progress, there's room for improvement namely in, consistency across different models and the need for more labelled data highlight. Expanding datasets from diverse ecosystems and integrating real-time monitoring could enhance our models' capabilities.

Moreover, our solutions aren't limited to Capuchin birds; they can extend to other species, aiding broader biodiversity conservation efforts. By leveraging machine learning and acoustic monitoring, we can better understand and protect forest ecosystems, preserving biodiversity for generations to come.

References

https://www.hp.com/in-en/workstations/industries/data-science/unlocked-challenge.html

https://www.kaggle.com/competitions/birdclef-2024

https://www.researchgate.net/publication/259235118_Random_Forests_and_Decision_Trees

https://www.researchgate.net/figure/Module-architecture-of-AdaBoost-classifier_fig6_224216262

https://www.isca-archive.org/interspeech_2015/peddinti15b_interspeech.pdf

https://kaleidoescape.github.io/tdnn/

https://www.researchgate.net/publication/328836649_Bird_Sound_Recognition_Using_a_Convolutional_Neural_Network

https://medium.com/@premtibadiya/music-genre-classification-using-rnn-lstm-1c212ba21e06

https://www.kaggle.com/code/abdallahaboelkhair/heartbeat-sound-lstm-classification

https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/tje2.12082

Stowell D, Plumbley MD. 2014. Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning. *PeerJ* 2:e488 https://doi.org/10.7717/peerj.488

https://www.sciencedirect.com/science/article/abs/pii/S0003682X20304795

https://pubmed.ncbi.nlm.nih.gov/34941869/