

ENVIRONMENTAL SOUND CLASSIFICATION

Deep Audio Analysis of Capuchin Bird Calls

A Project Submitted in
Partial Fulfilment of the Requirements

for the Degree of
Bachelor of Technology
in
Computer Science Engineering

As part of the core course “**SEM3501 – Seminar/Case Study**”

By

Harshita Nauhwar, 200C2030195, 200413

harshita.nauhwar.20cse@bmu.edu.in

UNDER THE SUPERVISION OF

Dr. Sachin Chaudhary

SCHOOL OF ENGINEERING AND TECHNOLOGY



**BML MUNJAL
UNIVERSITY™**

A **HERO GROUP** INITIATIVE

BML MUNJAL UNIVERSITY, GURUGRAM
November, 2022

CANDIDATE'S DECLARATION

I hereby certify that the work on the project entitled, “**Environmental Sound Classification**”, in partial fulfillment of requirements for the award of Degree of Bachelor of Technology in School of Engineering and Technology at BML Munjal University, having University Roll No.: 200C2030195 is an authentic record of our own work carried out during a period from July 2022 to December 2022 under the supervision of Dr. Sachin Choudhary.

(Harshita Nauhwar, 200C2030195)

SUPERVISOR'S DECLARATION

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dr. Sachin Chaudhary
Faculty Supervisor Name

Signature

CONTENT

Abstract	5
Acknowledgment	6
Introduction	7
Introduction to Project	8-11
Overview	8
Existing System	8-9
User Requirement Analysis	10
Feasibility Study	11
Literature Review	12-13
Comparison.....	12-13
Problem Statement & Objective	14
Exploratory Data Analysis	15-19
Dataset	15
Exploratory Data Analysis and Visualizations	16-19
Methodology	20-28
Overview	20
ML Algorithm Discussion	21
Implementation of Algorithm with Screen Shots/ Figures	22-28
Results	29
Conclusion & Future Scope	30
References	31
Plagiarism Report	32

ABSTRACT

The project's main goal is to create a machine learning model that can categorize various environmental sound classes. Sound extracts from various datasets, a convolutional neural network model, and audio data augmentation techniques will all be used to identify the environmental sounds.

Although music and speech signals have been the focus of audio recognition research for long, environmental sound recognition (ESR) has recently drawn increased attention. Over the past ten years, there has been a marked surge in ESR research.

Specifically, Bird Call Classification was performed to count the number of a certain bird call in a recording and train a TensorFlow model to apply it to the Forest Sound Recordings and identify the same.

The project includes model implementations of Conventional Neural Network using TensorFlow. Matplotlib is used for Visualization Purposes. Converting Audio Classification to Image Processing helped conclude the number of birdcalls.

In conclusion, the project aims to explore TensorFlow & Keras and its functionality while exploring Neural Networks and to classify Environmental Sounds.

ACKNOWLEDGMENT

It is with deep sense of gratitude and reverence that we express our sincere thanks to our supervisor **Dr. Sachin Chaudhary** for his guidance, encouragement, help and useful suggestions throughout. His untiring and painstaking efforts, methodical approach and individual help made it possible for our group to complete this work in time.

His guidance and scientific approach served a veritable incentive for completion of this work. We wouldn't have completed this project without the cooperation, kindness and general help extended to us by our supervisor **Dr. Sachin Chaudhary** during the completion of this work.

This acknowledgement will remain incomplete if we fail to express our deep sense of obligation to our parents and God for their consistent blessings and encouragement.

INTRODUCTION

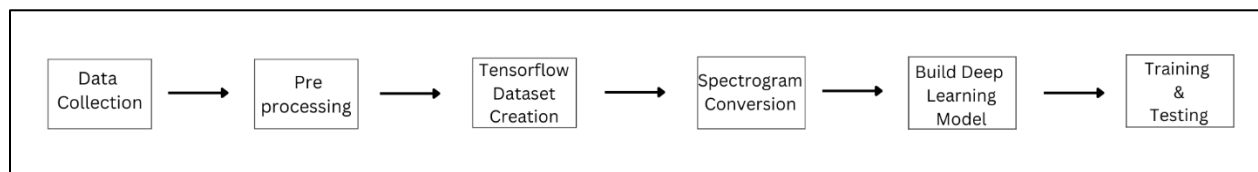
The research community has recently begun to pay more attention to the issue of automatic ambient sound classification. Over the past ten years, a lot of research has been done on modelling and detecting environmental noises. Environmental noises include a variety of common, man-made and natural sounds (i.e., sounds one encounters in daily life other than speech and music). Recent efforts to improve machine audition heavily rely on environmental sound recognition (ESR).

Speech and music are two genres of auditory signals that have undergone substantial research. Early speech and music recognition algorithms only reflected existing speech and music recognition paradigms. These algorithms didn't work well for large databases, though, because environmental sounds have a lot of non-stationary characteristics.

Data collection, pre-processing, feature extraction, feature selection, and data categorization are all steps in ESC. Performance can be greatly enhanced by altering any of the ESC phases and adding fresh techniques. The researchers have mostly employed a small number of standard datasets in their research.

Background noise must be eliminated during pre-processing in order to prepare the data for feature extraction.

Different types of features, including temporal, spectral, and dynamic time wrapping features, have been used by researchers. To decrease the number of features, only choose the most crucial ones. Researchers employ a range of feature selection strategies.



INTRODUCTION TO PROJECT

Overview

Environmental Sound Classification is implemented in many ways including counting animals or identifying species of animals and birds, for safety purpose in forest and urban areas.

Therefore, we have chosen an important part of the environment i.e., BIRDS. They are an important group of the environment. Even their bird calls are a crucial indicator of the health of the environment

We've recognized the challenges associated with bird call recognition. Implementation of a Conventional Neural Network was done using TensorFlow to count how many Capuchinbird calls are recorded in a particular clip.

Pandas, Keras & Matplotlib were used in addition as well for Visualization as well as Conclusive purposes.

Existing System

- Birdcall Recognition using Deep Convolutional Neural Network

Bird Call Recognition using Deep Convolutional Neural Network

Technology advancements have made it simple to use autonomous recording devices to monitor the environment (ARU).

As ARUs can be deployed in the field for weeks or months at a time, they make it simple for ecologists to monitor the environment without wasting time on frequent trips to field sites. ARU also make it possible for long-term, non-invasive passive environmental monitoring (PAM) because they may be deployed in large-scale deployments both spatially and temporally with no maintenance required. Ecologists frequently listen to acoustic recordings of animal calls to identify and count the number of distinct animal calls in order to track and monitor environmental changes.

Despite the widespread application of ARU in environmental monitoring, inadequate tools exist to analyse acoustic recordings for automated bird cry detection. Numerous ecologists still rely on manual identification, which slows down the processing of acoustic recordings.

Some tools, like SoundID (Boucher, 2014), employ a semi-automated process, but their calibration times are lengthy, their use necessitates a thorough understanding of signal processing, and their recognizers are designed specifically for a given call and do not generalise to other calls.

The following difficulties affect the work of automatic bird call recognition in acoustic recordings further (Priyadarshani, Marsland, and Castro 2018):

- Significant variation in bird calls between and within species.
- the mingling of bird sounds with background noise.
- Bird sounds that are repeated again and over, particularly at dawn and dusk chorus. Birds may produce incomplete, quick, or long calls depending on the situation;
- for example, during breeding season when they are busy with incubation and/or chick rearing, they may produce quick calls.
- Bird vocalisations may also vary in intensity depending on the distance and angle at which they are picked up by the ALU's microphones.

In benchmarked automatic bird call recogniser tasks, deep learning convolutional neural network (CNN) based architectures have currently shown the greatest success.

- The LifeCELF Bird Competition (BirdCELF), an annual benchmarking challenge to assess the state-of-the-art of audio based identification systems at scale, is one such benchmarking challenge.
- Three out of the five research groups that participated in BirdCELF 2016 and submitted working notes did so using CNN, more especially shallow-CNN (18 layers) .
- Deep-CNN (> 50 layers) architectures were created to handle more difficult image recognition problems and to improve recognition accuracy.
- The BirdCELF 2017 winner achieved a 0.714 Mean Average Precision (MAP) on 1500 bird species using a pre-trained deep learning network called Inception-v4.

USER REQUIREMENT ANALYSIS

The ordinary person can distinguish bird calls in their backyard, and experts can identify hundreds of bird species only by listening to their song. Humans are reasonably accurate auditory identifiers of birds.

As a result, it is not unexpected that bird call surveys are a popular technique for estimating bird population sizes and that conservation managers have used some of these techniques to track species for conservation goals.

Human-led surveys have been shown to have problems with misclassification of species due to observers' varying hearing abilities, as well as problems with species detection and identification. They can sometimes be expensive and logistically difficult. Additionally, the majority of bird population measurement techniques are either unsuitable or pricey for species with low populations.

With the development of technology, the use of autonomous recording units (ARUs) for bird population monitoring has increased. It has been acknowledged that this technology has the ability to solve some human problems and offers some additional benefits. The main concern is whether or not ARU recordings are equivalent to human hearing given that it is anticipated that they will replace or at least enhance human listening in the future.

This is particularly significant since developing protocols—which calls for knowledge of the ARUs' advantages and disadvantages for recording sounds under various circumstances—is one of the first steps in making this technology usable for conservation and/or study.

Based on sound theory, we hypothesized that:

- (a) calls broadcast from speakers situated significantly lower than listening stations would be recorded by recorders and humans but those broadcast from higher sites would not, since sound would go over the recorders/people.
- (b) speakers located in line of sight of autonomous recorders/human observers would be heard better and there would be less obstruction of the sound waves; and
- (c) low-frequency sounds would be recorded more effectively
- (d) reduced distances between the speaker and the autonomous recorder/human observer would result in better recordings.

FEASIBILITY STUDY

There are no other survey articles on ESC that we are aware of. The techniques used in the various phases of ESC have not been differentiated in the review articles in the field of ESC. Every stage of the ESC process is thoroughly explained in the study in this paper. The datasets used, techniques used for pre-processing, different feature extraction techniques and audio features, and classification techniques used by researchers in the past are illustrated in different sections. *The most recent study on ESR is using spectrograms along with data augmentation.*

The most recent methods and studies conducted by experts in the fields of ESR and ESC are covered in this review.

They originally sent out human observers to their first listening post. Then, every trial had the same structure: Six radio stations played a series of bird cries in response to a sound signal (a shotgun blast). Another shotgun blast signalled the conclusion of the trial to human watchers at the end of the broadcast. The next trial started after the observers had had 10 minutes to move to their new listening station.

At the conclusion of the experiment, a double shot was fired to signal that it was time to head back to base. Human observers were not aware of the six broadcast locations, but they visited the seven listening stations during the daytime before the experiment to become familiar with their positions along the route. In order to prevent observers from learning the locations of the broadcasts, experimenters with their broadcast equipment were deployed to their sites before the human observers began the experiment. After the start of each trial, experimenters would turn on speakers at predetermined intervals.

High-quality recordings of the species' calls were used to choose each individual bird call. Wavelets were used to denoise the files. The songs were concatenated and a tone marker was inserted at the start of the sequence. In this manner, when compared to one another, all of the songs on the tape were at the estimated ideal volume.

They anticipated that observers would typically be able to hear sound from several of the speakers even though broadcasts from different speakers weren't supposed to overlap. In real life, some experimenters started the speakers a little early or a little late, which led to some song overlap. Three nocturnal bird calls that the observers were familiar with were broadcast by each speaker. Based on the volume of the starting tone, each speaker's broadcasting volume was modified until it was equal across the board. One speaker was used to play the music, and a sound meter was used to calibrate every other speaker.

LITERATURE REVIEW

Comparison

A. Convolutional Neural Network (CNN)

Deep learning neural network, or CNN. The classification accuracy of the various suggested CNNs has surpassed that of ESR.

In the first use of CNN, PiczackCNN showed that it performs better than a machine learning model based on MFCCs.

- To discover the ideal setting and achieve high accuracy, many hyperparameters like padding, the size of the max-pooling layer, and stride length are modified.

Using CNN, 92.90% accuracy is attained on UrbanSound8k. Parallel CNN has the most accuracy when employed in research, with sequential, parallel, and end-to-end CNN all being used.

ESR is suggested using a two-stream CNN with a decision-level fusion model.

- Two CNNs are fed with the two feature sets, and the output of their SoftMax layers are combined using the Dempster Shafer Evidence theory to achieve an accuracy of 97.20%.
- When end-to-end learning CNN and LMS-based CNN are merged, the accuracy has also been enhanced using DS Evidence Theory.
- On the ESC-10 and ESC-50 datasets, two-layer CNN provides accuracy of 77.00% and 49.00%, respectively.

B. Tensor Deep Stacking Network (TDSN)

While TDSN and deep stacking networks (DSN) are similar, DSN uses sequential hidden layers in each module while TDSN uses parallel hidden layers. On the ESC-10 dataset, TDSN achieves an accuracy of 56.0%.

C. **Convolutional Recurrent Neural Network (CRNN)**

Recurrent neural networks (RNN) and CNN are coupled for ESC. The features are extracted with CNN, and the extracted features are temporally aggregated with RNN. For ESC, CRNN has shown to be successful.

In the study, deep convolutional generative adversarial networks are used to extract features, and CRNN is then used to classify the data.

D. **Image Recognition Networks**

ESC can employ a very sophisticated CNN that was initially created for image classification.

In a study, significant accuracy is reached using AlexNet and GoogLeNet on the image-based features for ESC. Use of a deep neural network in the VGG fashion.

E. **Deep Belief Neural Networks (DBNN)**

Traditional DNN faced issues with sluggish learning and a need for a lot of training data, which led to the rise in popularity of DBNN.

- For ESC, DBNN outperforms GMM-based HMM and NN with two or five layers.

Researchers are looking into other classifiers for ESR, including self-organizing maps, self-supervised learning-based deep classifiers, and Bayesian belief networks.

PROBLEM STATEMENT & OBJECTIVE

Problem Statement:

Effective bird monitoring techniques are required to determine species existence and abundance, to assess the effects of existing species management policies for conservation, and to determine the overall balance in a particular biome.

Because it can be used even when an individual is hidden from view, bird song is frequently used to identify, track, and measure species.

The ordinary person can distinguish birdcalls in their backyard, and experts can identify hundreds of bird species only by listening to their song. Humans are reasonably accurate auditory identifiers of birds.

As a result, it is not unexpected that birdcall surveys are a popular technique for estimating bird population sizes and that conservation managers have used some of these techniques to track species for conservation goals.

There is not yet an adequate method for automated bird call recognition in acoustic recordings due to high variations in bird calls and the challenges associated with bird call recognition.

We recognized the problem of Birdcall Recognition and identified Capuchinbird for the same purpose.

Therefore, the task is to develop source code and a machine learning model to count how many Capuchinbird calls are recorded in a particular clip.

Objective:

By the end of this project, we aim to obtain a Deep Learning Neural Network Model which will successfully analyze and count the number of birdcalls in a specific given clip.

The same will be implemented using Conventional Neural Network using TensorFlow, Keras, Matplotlib and Pandas library.

EXPLORATORY DATA ANALYSIS

Datasets:

Data Acquisition was done using Public Datasets obtained from Kaggle.
Capuchinbird Calls was the Data chosen.

About the Dataset:

The data is divided into Training and Testing Data.

We have supplied enough audio clips in the Training Data to produce a passable model, but you can also locate, parse, modify, and use more audio samples to enhance the performance of your model.

– Testing Set

[1] Forest Recordings - These are the Recordings to use to count the number of Calls within. Each clip is ~3 min and includes a mix of Capuchinbirds and other sounds.

- Contains 100 files

– Training Set

[2] Parsed Capuchinbird Clips - These are parsed 3 sec clips that include specific bird calls from Capuchinbirds only

- Contains 217 files

[3] Parsed-Not-Capuchinbird Clips - These files are parsed sounds from other animals/birds that are useful in training what is not a Capuchinbird.

- Contains 593 files

These sounds from the downloaded sound repository are filtered and labeled manually to generate the training and testing dataset for the Conventional Neural Network to be developed.

Exploratory Data Analysis & Visualizations:

Exploratory data analysis is a necessary and important stage in refining the given data set(s) in a different type of analysis to comprehend the insights of the main characteristics of the various entities of the data set, such as column(s), row(s), and statistical methods.

Features:

- Cleaning up the given dataset is made easier with an understanding of it.
- You can clearly see the features and how they relate to one another.
- Establishing rules for vital variables and eliminating/leaving out non-essential ones.
- Handling human error or missing values.
- Recognizing outliers.
- Maximizing dataset's insights.

1. **Loading the Dataset** such that it is easily accessible and one doesn't need to upload it every time to run it.

This is done using several methods, but the one we used is the **Google Drive Mounting method** (shown below). Using this a storage device's files and directories are made accessible to users through the computer's file system.

▼ 2. Build Data Loading Function

▼ 2.1 Define Paths to Files

```
[ ] from google.colab import drive
    drive.mount('/content/drive',force_remount=True)
    base_path = "/content/drive/MyDrive/birdcall"
    CAPUCHIN_FILE = os.path.join(base_path, 'Parsed_Capuchinbird_Clips', 'XC3776-3.wav')
    NOT_CAPUCHIN_FILE = os.path.join(base_path, 'Parsed_Not_Capuchinbird_Clips', 'afternoon-birds-song-in-forest-0.1
```

Mounted at /content/drive

```
[ ] CAPUCHIN_FILE
```

```
'/content/drive/MyDrive/birdcall/Parsed_Capuchinbird_Clips/XC3776-3.wav'
```


2. Building the Data Loading Function

Created to convert the audio files into a 16khz mono audio

The function used for the same is pre-defined by TensorFlow libraries and is shown below:

2.2 Build Dataloading Function

```
[ ] def load_wav_16k_mono(filename):
    # Load encoded wav file
    file_contents = tf.io.read_file(filename)
    # Decode wav (tensors by channels)
    wav, sample_rate = tf.audio.decode_wav(file_contents, desired_channels=1)
    # Removes trailing axis
    wav = tf.squeeze(wav, axis=-1)
    sample_rate = tf.cast(sample_rate, dtype=tf.int64)
    # Goes from 44100Hz to 16000hz - amplitude of the audio signal
    wav = tfio.audio.resample(wav, rate_in=sample_rate, rate_out=16000)
    return wav #16 Hz & single channel
```

```
file_contents = tf.io.read_file(CAPUCHIN_FILE)
file_contents
```

```
<tf.Tensor: shape=(), dtype=string, numpy=b'RIFF\xbc\t\x04\x00WAVEfmt
\x10\x00\x00\x00\x01\x00\x01\x000\xac\x00\x00\x088\x01\x00\x00\x01\x00data\x98\t\x04\x00\x0c5\xf1(\xfbf\x7f\x07F\x13|
\xb3\xaa\xaf\xad\xad\x00\xafdf\x22\x0b5\xeb\x06\x0f5\x0b9m\x00\x01\x01\x0c9\x0c6\x0d0-
\x06\x0ba\x0db0\xe4[\xefg\xf9\x02\x0ff\x0c3\x04\x87\x0b\x0ec\x14m\x1e\x0c6%\xc4+\xb22\xa3:\xd7@~\x17C\x868B0c[D\x0c5B\xda>\x9e
3"\x15"v!\xb7!("/\xc4
\x1d\x0d\x18\x01\x14\x0d\x01\x04\x0b\x0b\x04\x0b\x0f\x0a3\x0b\x0f\x0d\x01\x075\x0f3V\x0a0\x0f\x0a\x04\x04\x0e3\x01\x0a\x0f\x0d\x0a\x01
```

3. Plotting Wave

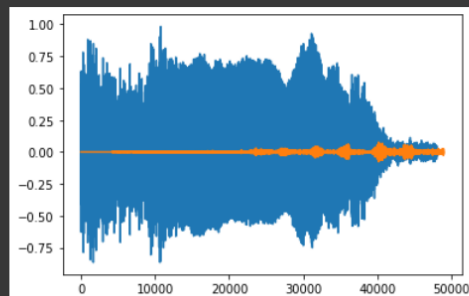
The Data files are loaded here using the `load_wav_16k_mono` function and are visualized using Matplotlib.

We can observe the difference between the audio files that have the Capuchin bird calls(**blue**) and the files which have random Environment Sounds(**orange**)

▼ 2.3 Plot Wave

```
# Loading wave files using the function above
wave = load_wav_16k_mono(CAPUCHIN_FILE)
nwave = load_wav_16k_mono(NOT_CAPUCHIN_FILE)
```

```
[ ] plt.plot(wave)      #blue
    plt.plot(nwave)    #orange
    plt.show()
```



4. Building the Pre-processing Function to convert to Spectrogram

Created to convert the audio file (that was gone through the data loading function) to Spectrogram for Image Processing aspect

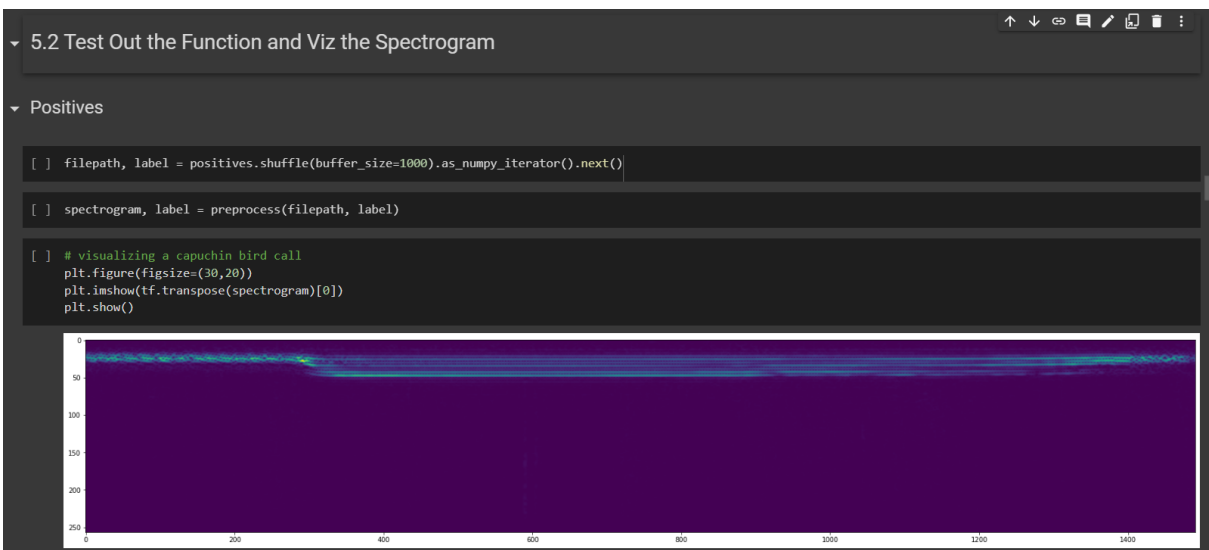
▼ 5.1 Build Preprocessing Function

```
[ ] def preprocess(file_path, label):  
    wav = load_wav_16k_mono(file_path)  
    wav = wav[:48000]  
    zero_padding = tf.zeros([48000] - tf.shape(wav), dtype=tf.float32)  
    wav = tf.concat([zero_padding, wav],0)  
    spectrogram = tf.signal.stft(wav, frame_length=320, frame_step=32)  
    spectrogram = tf.abs(spectrogram)  
    spectrogram = tf.expand_dims(spectrogram, axis=2)  
    return spectrogram, label
```

5. Testing the function defined to convert to Spectrogram

Visualizations are done to test the audio files that are converted to Spectrogram using the Pre-processing function.

- Positives, i.e., the files which include only Capuchinbirds calls shows a defines Spectrogram
- Negatives, i.e., the files which include only Environment sounds shows a random Spectrogram

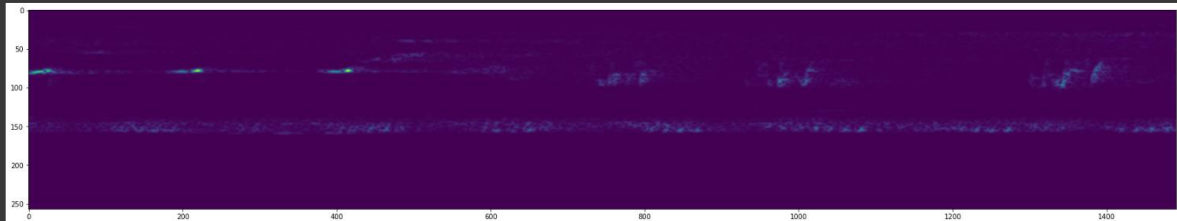


▼ Negatives

```
[ ] filepath, label = negatives.shuffle(buffer_size=1000).as_numpy_iterator().next()

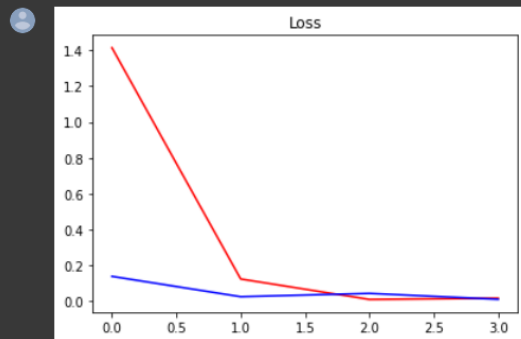
[ ] spectrogram, label = preprocess(filepath, label)

[ ] # visualizing non capuchin bird call
plt.figure(figsize=(30,20))
plt.imshow(tf.transpose(spectrogram)[0])
plt.show()
```

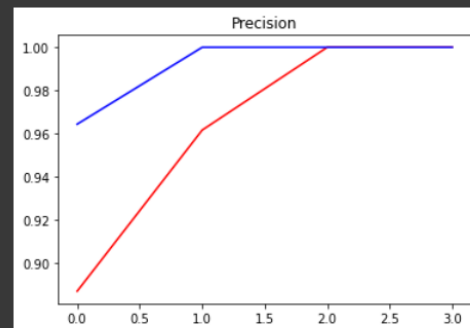


6. Fit Model, View Loss and KPI Plots

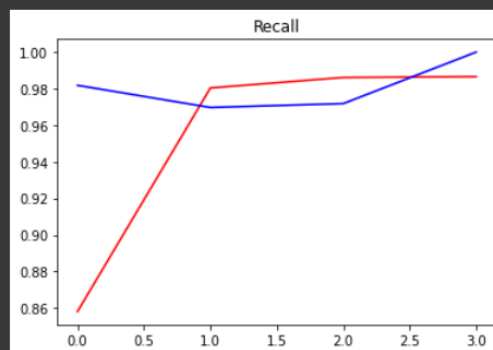
```
plt.title('Loss')
plt.plot(hist.history['loss'], 'r')
plt.plot(hist.history['val_loss'], 'b')
plt.show()
```



```
plt.title('Precision')
plt.plot(hist.history['precision'], 'r')
plt.plot(hist.history['val_precision'], 'b')
plt.show()
```



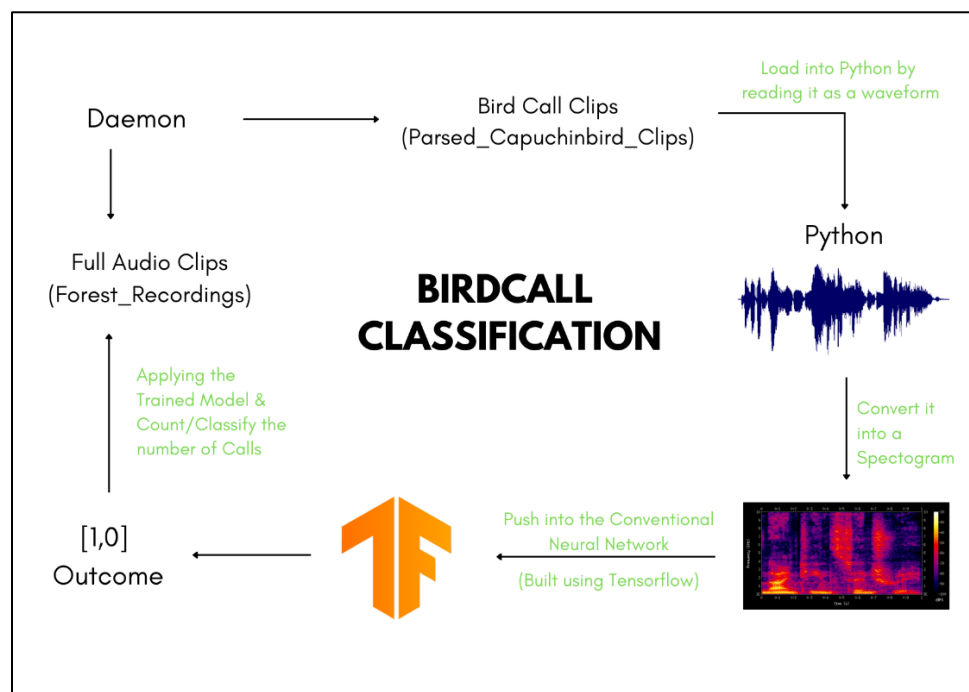
```
[ ] plt.title('Recall')
plt.plot(hist.history['recall'], 'r')
#plt.plot(hist.history['val_recall'], 'b')
plt.show()
```



METHODOLOGY

Overview:

- Though our train files (Parsed_Capuchinbird_clips/ Parsed_Not_Capuchinbird_clips .wav files) are only a few seconds long, the audio files for testing (Forest Recordings) are around 3 minutes long.
- To get the final count, we divided the audio stream into short pieces (3 second segments), and use our model to forecast each segment, and then add each prediction.
- .wav files are loaded to determine their wavelengths.
- The wavelengths are then turned into spectrograms (Image processing) now for Visualization purposes
- A Conventional Neural Network model is now developed. After that, the spectrograms are fed into the model to start producing predictions.
- Next, we will turn our attention to the Forest recording files, we'll segment them as we previously indicated, and carry out all of the aforementioned actions on those files.
- This will then successfully complete our Model That Can Identify the Density Of The Capuchin Bird Calls.



ML Algorithm Discussion:

Convolutional neural networks (CNNs) have been one of the most effective deep neural architectures.

The agility of CNNs in minimizing variances and extracting spatial correlations for large-scale picture identification is the major factor driving their spread across many domains.

This research explores CNNs with audio classification in light of the amazing accomplishments of CNNs.

About Convolutional neural networks (CNNs):

Convolutional neural networks outperform other neural networks when given inputs such as images, voice, or audio, for example.

There are three basic categories of layers in them:

- Constellation layer
- gathering layer
- FC (fully-connected) layer

A convolutional network's first layer is the convolutional layer. The fully-connected layer is the last layer, even though convolutional layers, further convolutional layers, or pooling layers, can come after it.

The CNN becomes more complicated with each layer, detecting larger areas of the audio.

[4] Early layers emphasise basic elements.

[5] The larger features or shapes of the object are first recognized when the audio data moves through the CNN layers, and eventually the intended object is recognized.

A CNN performs a Rectified Linear Unit (ReLU) adjustment on the feature map following each convolution operation, adding nonlinearity to the model.

As was previously mentioned, the first convolution layer may be followed by another convolution layer. When this occurs, CNN's organizational structure may become hierarchical.

IMPLEMENTATION OF ALGORITHM WITH SCREEN SHOTS

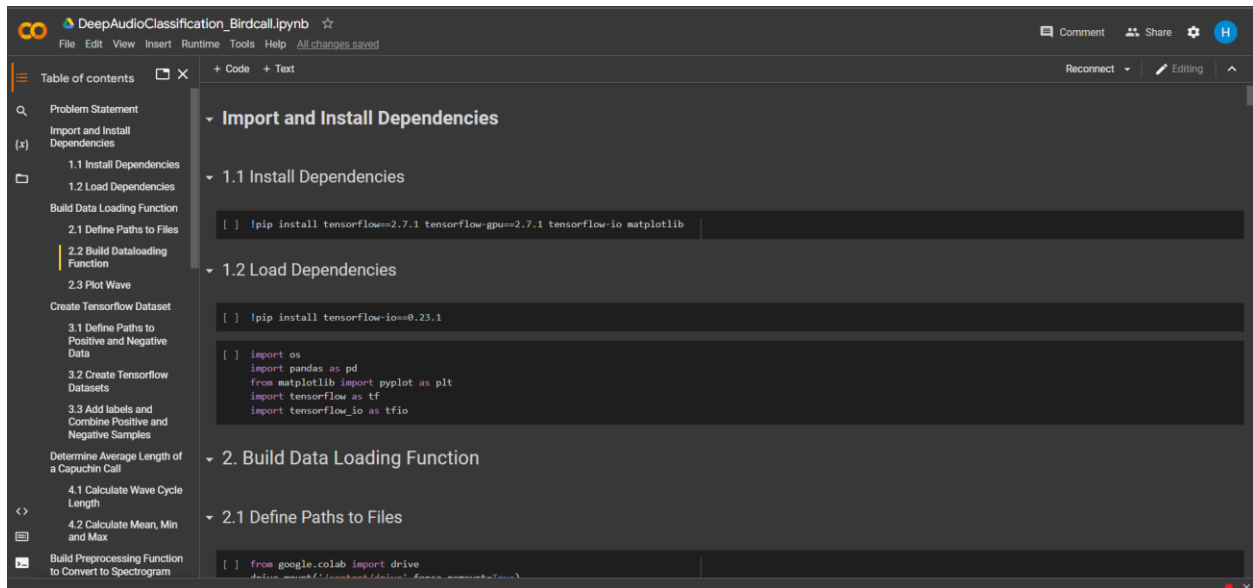
Implementation Steps:

1. Choosing the Dataset & Defining Paths to Files
2. Importing and Installing Dependencies
3. Building Data Loading Function
4. Define Paths to Positive and Negative Data
5. Create TensorFlow Datasets
6. Add labels and Combine Positive and Negative Samples
7. Determine Average Length of a Capuchin Call
 - Calculate Wave Cycle Length
8. Build Pre-processing Function to Convert to Spectrogram
9. Create & Split into Training and Testing Partitions
10. Testing One batch
11. Build Deep Learning Model
 - Load TensorFlow Dependencies
 - Build Sequential Model, Compile and View Summary
12. Make a Prediction on a Single Clip
13. Build Forest Recording Parsing Functions
 - Build Function to Convert Clips into Windowed Spectrograms
 - Convert Longer Clips and Make Predictions
 - Group Consecutive Detections
14. Make Predictions
 - Loop over all recordings and make predictions
 - Convert Predictions into Classes
 - Group Consecutive Detections
15. Export the Results

Screen Shots:

The screenshots provided below shows the implementation of the Convolutional neural networks Model on Google Colab.

The link for the same will be provided in the references.



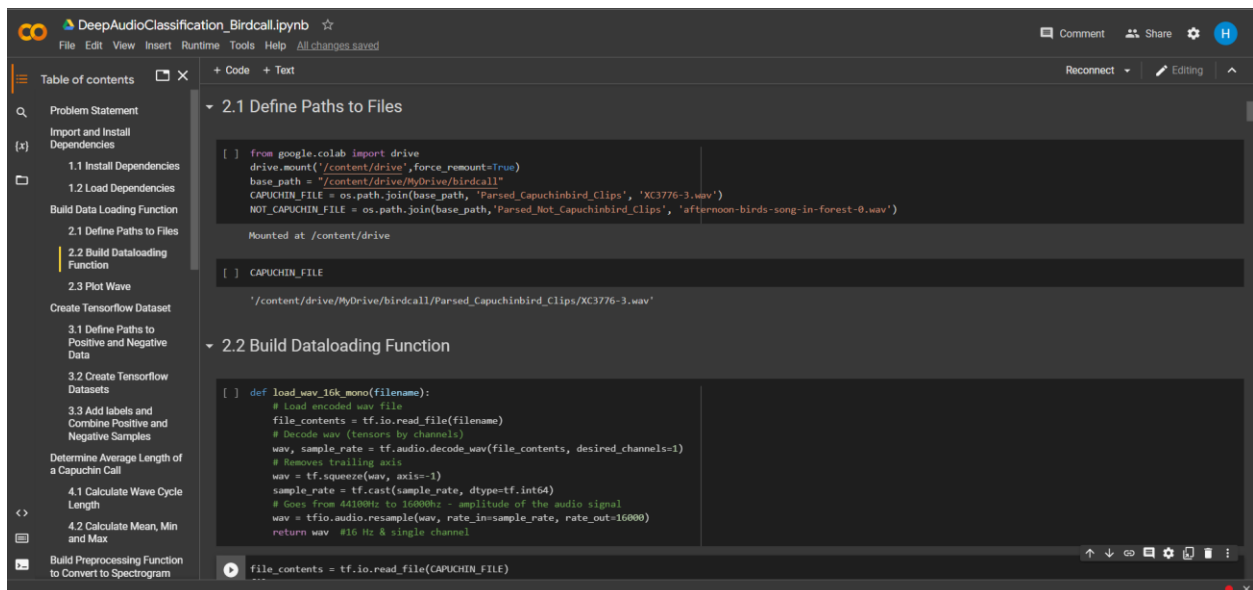
This screenshot shows the Google Colab interface for the notebook 'DeepAudioClassification_Birdcall.ipynb'. The left sidebar contains a table of contents with sections like 'Problem Statement', 'Import and Install Dependencies', 'Build Data Loading Function', 'Create Tensorflow Dataset', and 'Build Preprocessing Function'. The main editor area is divided into sections: 'Import and Install Dependencies', '1.1 Install Dependencies' (with code to install tensorflow, tensorflow-gpu, tensorflow-io, and matplotlib), '1.2 Load Dependencies' (with code to import os, pandas, matplotlib, pyplot, tensorflow, and tensorflow-io), '2. Build Data Loading Function', and '2.1 Define Paths to Files' (with code to import drive from google.colab).

```
!pip install tensorflow==2.7.1 tensorflow-gpu==2.7.1 tensorflow-io matplotlib
```

```
!pip install tensorflow-io==0.23.1
```

```
import os
import pandas as pd
from matplotlib import pyplot as plt
import tensorflow as tf
import tensorflow_io as tfio
```

```
from google.colab import drive
drive.mount('/content/drive')
```



This screenshot shows the Google Colab interface for the notebook 'DeepAudioClassification_Birdcall.ipynb', focusing on the '2.1 Define Paths to Files' and '2.2 Build Dataloading Function' sections. The code in '2.1' mounts the drive and defines paths for 'CAPUCHIN_FILE' and 'NOT_CAPUCHIN_FILE'. The code in '2.2' defines a function 'load_wav_16k_mono' that reads a wav file, decodes it, removes the trailing axis, and resamples it to 16000 Hz.

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
base_path = "/content/drive/MyDrive/birdcall"
CAPUCHIN_FILE = os.path.join(base_path, 'Parsed_Capuchinbird_Clips', 'XC3776-3.wav')
NOT_CAPUCHIN_FILE = os.path.join(base_path, 'Parsed_Not_Capuchinbird_Clips', 'afternoon-birds-song-in-forest-0.wav')

Mounted at /content/drive
```

```
CAPUCHIN_FILE
'/content/drive/MyDrive/birdcall/Parsed_Capuchinbird_Clips/XC3776-3.wav'
```

```
def load_wav_16k_mono(filename):
    # Load encoded wav file
    file_contents = tf.io.read_file(filename)
    # Decode wav (tensors by channels)
    wav, sample_rate = tf.audio.decode_wav(file_contents, desired_channels=1)
    # Removes trailing axis
    wav = tf.squeeze(wav, axis=-1)
    sample_rate = tf.cast(sample_rate, dtype=tf.int64)
    # Goes from 44100Hz to 16000Hz - amplitude of the audio signal
    wav = tfio.audio.resample(wav, rate_in=sample_rate, rate_out=16000)
    return wav #16 Hz & single channel
```

```
file_contents = tf.io.read_file(CAPUCHIN_FILE)
```


Table of contents

2.1 Define Paths to Files

2.2 Build Dataloading Function

2.3 Plot Wave

Create Tensorflow Dataset

3.1 Define Paths to Positive and Negative Data

3.2 Create Tensorflow Datasets

3.3 Add labels and Combine Positive and Negative Samples

Determine Average Length of a Capuchin Call

4.1 Calculate Wave Cycle Length

4.2 Calculate Mean, Min and Max

Build Preprocessing Function to Convert to Spectrogram

5.1 Build Preprocessing Function

5.2 Test Out the Function and Viz the Spectrogram

Positives

Negatives

5. Build Preprocessing Function to Convert to Spectrogram

5.1 Build Preprocessing Function

```
def preprocess(file_path, label):
    wav = load_wav_16k_mono(file_path)
    wav = wav[:48000]
    zero_padding = tf.zeros([48000] - tf.shape(wav), dtype=tf.float32)
    wav = tf.concat([zero_padding, wav], 0)
    spectrogram = tf.signal.stft(wav, frame_length=320, frame_step=32)
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.expand_dims(spectrogram, axis=2)
    return spectrogram, label
```

5.2 Test Out the Function and Viz the Spectrogram

Positives

```
filepath, label = positives.shuffle(buffer_size=1000).as_numpy_iterator().next()

spectrogram, label = preprocess(filepath, label)

# visualizing a capuchin bird call
plt.figure(figsize=(30,20))
plt.imshow(tf.transpose(spectrogram)[0])
```

Table of contents

2.1 Define Paths to Files

2.2 Build Dataloading Function

2.3 Plot Wave

Create Tensorflow Dataset

3.1 Define Paths to Positive and Negative Data

3.2 Create Tensorflow Datasets

3.3 Add labels and Combine Positive and Negative Samples

Determine Average Length of a Capuchin Call

4.1 Calculate Wave Cycle Length

4.2 Calculate Mean, Min and Max

Build Preprocessing Function to Convert to Spectrogram

5.1 Build Preprocessing Function

5.2 Test Out the Function and Viz the Spectrogram

Positives

Negatives

6. Create Training and Testing Partitions

6.1 Create a Tensorflow Data Pipeline

Preprocessing, caching and mixing up the data samples to avoid over-fitting Then training on 16 sets at a time

```
data = data.map(preprocess)
data = data.cache()
data = data.shuffle(buffer_size=1000)
data = data.batch(16)
data = data.prefetch(8)
```

WARNING:tensorflow:Using a while_loop for converting ID3AudioResample

6.2 Split into Training and Testing Partitions

```
train = data.take(36)
test = data.skip(36).take(15)
```

6.3 Test One Batch

```
samples, labels = train.as_numpy_iterator().next()

samples.shape
# 16 different examples of the spectrogram with the shape of 1491/257/1
```

Table of contents

2.1 Define Paths to Files

2.2 Build Dataloading Function

2.3 Plot Wave

Create Tensorflow Dataset

3.1 Define Paths to Positive and Negative Data

3.2 Create Tensorflow Datasets

3.3 Add labels and Combine Positive and Negative Samples

Determine Average Length of a Capuchin Call

4.1 Calculate Wave Cycle Length

4.2 Calculate Mean, Min and Max

Build Preprocessing Function to Convert to Spectrogram

5.1 Build Preprocessing Function

5.2 Test Out the Function and Viz the Spectrogram

Positives

Negatives

7. Build Deep Learning Model

7.1 Load Tensorflow Dependencies

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten
```

7.2 Build Sequential Model, Compile and View Summary

```
model = Sequential()

#adding convolutional layer, 16 different kernels of shape 3/2
model.add(Conv2D(16, (3,3), activation='relu', input_shape=(1491, 257,1))) # relu- rectified linear unit
model.add(Conv2D(16, (3,3), activation='relu'))

#flattening from 3d to single dimension
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile('Adam', loss='BinaryCrossentropy', metrics=[tf.keras.metrics.Recall(), tf.keras.metrics.Precision()]) #using adam optimizer

model.summary()
```

Layer (type)	Output Shape	Param #
Model: "sequential_1"		


```
DeepAudioClassification_Birdcall.ipynb
File Edit View Insert Runtime Tools Help
Table of contents
6.3 Test One Batch
Build Deep Learning Model
7.1 Load Tensorflow Dependencies
7.2 Build Sequential Model, Compile and View Summary
7.3 Fit Model, View Loss and KPI Plots
Make a Prediction on a Single Clip
8.1 Get One Batch and Make a Prediction
8.2 Convert Logits to Classes
Build Forest Parsing Functions
9.1 Load up MP3s
9.2 Build Function to Convert Clips into Windowed Spectrograms
9.3 Convert Longer Clips and Make Predictions
9.4 Group Consecutive Detections
10.1 Loop over all recordings and make predictions
+ Code + Text
9. Build Forest Parsing Functions
9.1 Load up MP3s
[ ] # for forest recordings
def load_mp3_16k_mono(filename):
    """ Load a wav file, convert it to a float tensor, resample to 16 kHz single-channel audio. """
    res = tf.io.audio.AudioTensor(filename)
    # Convert to tensor and combine channels
    tensor = res.to_tensor()
    tensor = tf.math.reduce_sum(tensor, axis=1) / 2
    # Extract sample rate and cast
    sample_rate = res.rate
    sample_rate = tf.cast(sample_rate, dtype=tf.int64)
    # Resample to 16 kHz
    wav = tf.io.audio.resample(tensor, rate_in=sample_rate, rate_out=16000)
    return wav

[ ] mp3 = os.path.join(base_path, 'Forest Recordings', 'recording_00.mp3')

[ ] wav = load_mp3_16k_mono(mp3)

[ ] # slicing into same size audio for multiple predictions
audio_slices = tf.keras.utils.timeseries_dataset_from_array(wav, wav, sequence_length=48000, sequence_stride=48000, batch_size=1)

[ ] len(audio_slices)
60
```

```
DeepAudioClassification_Birdcall.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Table of contents
6.3 Test One Batch
Build Deep Learning Model
7.1 Load Tensorflow Dependencies
7.2 Build Sequential Model, Compile and View Summary
7.3 Fit Model, View Loss and KPI Plots
Make a Prediction on a Single Clip
8.1 Get One Batch and Make a Prediction
8.2 Convert Logits to Classes
Build Forest Parsing Functions
9.1 Load up MP3s
9.2 Build Function to Convert Clips into Windowed Spectrograms
9.3 Convert Longer Clips and Make Predictions
9.4 Group Consecutive Detections
10.1 Loop over all recordings and make predictions
+ Code + Text
9.2 Build Function to Convert Clips into Windowed Spectrograms
[ ] def preprocess_mp3(sample, index):
    sample = sample[0]
    zero_padding = tf.zeros([48000] - tf.shape(sample), dtype=tf.float32)
    wav = tf.concat([zero_padding, sample], 0)
    spectrogram = tf.signal.stft(wav, frame_length=320, frame_step=32)
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.expand_dims(spectrogram, axis=2)
    return spectrogram
9.3 Convert Longer Clips and Make Predictions
[ ] audio_slices = tf.keras.utils.timeseries_dataset_from_array(wav, wav, sequence_length=16000, sequence_stride=16000, batch_size=1)
audio_slices = audio_slices.map(preprocess_mp3)
audio_slices = audio_slices.batch(64)

[ ] yhat = model.predict(audio_slices)
yhat = [1 if prediction > 0.5 else 0 for prediction in yhat]

[ ] yhat
[0,
0,
0,
0,
0,
0,
1,
0,
0,
0]
```

```
DeepAudioClassification_Birdcall.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Table of contents
6.3 Test One Batch
Build Deep Learning Model
7.1 Load Tensorflow Dependencies
7.2 Build Sequential Model, Compile and View Summary
7.3 Fit Model, View Loss and KPI Plots
Make a Prediction on a Single Clip
8.1 Get One Batch and Make a Prediction
8.2 Convert Logits to Classes
Build Forest Parsing Functions
9.1 Load up MP3s
9.2 Build Function to Convert Clips into Windowed Spectrograms
9.3 Convert Longer Clips and Make Predictions
9.4 Group Consecutive Detections
10.1 Loop over all recordings and make predictions
+ Code + Text
9.4 Group Consecutive Detections
from itertools import groupby

[ ] yhat = [key for key, group in groupby(yhat)]
calls = tf.math.reduce_sum(yhat).numpy()

[ ] calls
7
10. Make Predictions
10.1 Loop over all recordings and make predictions
[ ] results = {}
for file in os.listdir(os.path.join(base_path, 'Forest Recordings')):
    FILEPATH = os.path.join(base_path, 'Forest Recordings', file)

    wav = load_mp3_16k_mono(FILEPATH)
    audio_slices = tf.keras.utils.timeseries_dataset_from_array(wav, wav, sequence_length=48000, sequence_stride=48000, batch_size=1)
    audio_slices = audio_slices.map(preprocess_mp3)
    audio_slices = audio_slices.batch(64)

    yhat = model.predict(audio_slices)

    results[file] = yhat
```


RESULTS

The results obtained from the Deep Learning Model showcases the number of Capuchinbirds calls in each Forest Recording Audio file

The model ends in exporting the results in a .csv format which is shown below

	A	B	C	D
1	audio file (Forest Recording)	capuchin_calls		
2	recording_00.mp3	5		
3	recording_01.mp3	0		
4	recording_02.mp3	0		
5	recording_03.mp3	0		
6	recording_04.mp3	4		
7	recording_05.mp3	0		
8	recording_06.mp3	5		
9	recording_07.mp3	2		
10	recording_08.mp3	23		
11	recording_09.mp3	0		
12	recording_10.mp3	5		
13	recording_11.mp3	10		
14	recording_12.mp3	0		
15	recording_13.mp3	0		
16	recording_14.mp3	0		
17	recording_15.mp3	1		
18	recording_16.mp3	10		
19	recording_17.mp3	3		
20	recording_18.mp3	0		
21	recording_19.mp3	0		
22	recording_20.mp3	0		
23	recording_21.mp3	0		
24	recording_22.mp3	2		

◀ ▶

results

⊕

CONCLUSION & FUTURE WORK

Conclusion:

This document presents a comprehensive overview of the ESC-related research. The researchers who operate in this field can benefit from a thorough examination of the datasets, features, and classifiers. The research is hampered by the absence of a shared dataset for ESC.

Many researchers have combined characteristics since a single trait cannot give precision. Recent research demonstrates the success of deep neural networks. Up until this point, a convolutional neural network has been used to achieve the highest accuracy.

Future Work:

We'd like to mention a few potential future research and development directions in closing. Future research into environmental sound classification can examine large datasets, more durable characteristics, unique feature combinations, and novel neural network architectures.

1. **Database expansion:** The only publicly accessible benchmark dataset that is larger than UrbanSound8K is UrbanSound8K. The current dataset can be increased by adding more audio recordings and audio files of various types.
2. **Robust features and feature combinations:** It can be inferred from a careful analysis of the literature that feature combinations perform better than individual features. Therefore, different alternative feature combinations can be investigated in the future. Investigating additional, more reliable features is possible. New features most likely produce better outcomes.
3. **New neural network architectures:** It is possible to look at new neural network architectures that haven't been used before. The neural networks for ESC must be combined with a lot of work because doing so has improved results in related fields.
4. **Meta analysis of prior literature:** Only thorough review is performed in this paper. Future research publications may be subjected to meta-analysis and simulation.

REFERENCES

- [1] Castro, Isabel, et al. "Experimental Test of Birdcall Detection by Autonomous Recorder Units and by Human Observers Using Broadcast - [Scite Report]." scite.ai, scite.ai/reports/experimental-test-of-birdcall-detection-zRKzxK8, 10 February 2019
- [2] Sankupellay, Mangalam & Konovalov, Dmitry, "Bird Call Recognition using Deep Convolutional Neural Network", ResNet-50. 10.13140/RG.2.2.31865.31847, 2018
- [3] Anam Bansal, Naresh Kumar Garg, et al. "Environmental Sound Classification: A descriptive review of the literature." sciencedirect.com, <https://www.sciencedirect.com/science/article/pii/S2667305322000539>, Volume 16, November 2022, 200115
- [4] Project Link :
<https://colab.research.google.com/drive/1zmQwSPGgBX750UXpNJBnI1210qpI16Fh?usp=sharing>

PLAGIARISM REPORT

Environmental Sound Classification

ORIGINALITY REPORT

5%

SIMILARITY INDEX

0%

INTERNET SOURCES

10%

PUBLICATIONS

0%

STUDENT PAPERS

PRIMARY SOURCES

1

Anam Bansal, Naresh Kumar Garg.
"Environmental Sound Classification: A
descriptive review of the literature", Intelligent
Systems with Applications, 2022

Publication

5%

Exclude quotes Off

Exclude bibliography Off