CONTENTS

ABSTRACT

The project report details an endeavour to classify bird sounds using advanced machine learning (ML) and deep learning (DL) techniques, an essential task for monitoring ecological health and biodiversity. Building on the foundation of manual and basic ML-based audio classification, the project leveraged a rich Kaggle dataset of bird sounds, focusing on feature extraction using Mel Frequency Cepstral Coefficients (MFCCs) and employing Python tools like Librosa and TensorFlow. The exploratory data analysis (EDA) involved visualizing audio samples to aid classification, leading to the development and testing of models such as Support Vector Machine, Convolutional Neural Networks (CNN), and Artificial Neural Networks (ANN). The ANN model emerged as the most effective, achieving an accuracy of 89.95%, evaluated through metrics like accuracy, recall, and F1 score. Despite its success, the project faced limitations including dataset imbalance, potential overfitting, and the lack of model interpretability. Future directions involve addressing these limitations by enhancing data collection, exploring regularization techniques, and improving interpretability. Overall, the project marks a significant stride in using ML and DL for bird sound classification, offering substantial implications for ecological research and conservation efforts, and setting a stage for future advancements in real-time classification and improved model generalization in bioacoustics.

Keywords: Audio Classification, Bird Classifier, Mel Frequency Cepstral Coefficient, Artificial Neural Networks

CHAPTER-1
INTRODUCTION

Rationale for the Project

In recent years, the field of bioacoustics has gained significant momentum, primarily due to advancements in machine learning and data processing technologies. The ability to classify and analyze bird sounds automatically has immense ecological and conservational value. Birds, being sensitive to environmental changes, serve as vital indicators of ecological health. This project stems from the necessity to develop an efficient, accurate system for bird sound classification, which can aid in monitoring biodiversity, studying bird behaviour, and contributing to conservation efforts.

Moreover, bird sound classification has practical applications in urban planning, where understanding avian populations can assist in creating more bird-friendly environments. In the broader sense, this project aligns with the growing need for integrating technological solutions with wildlife conservation, aiming to bridge the gap between advanced computational methods and ecological research.

Scope of the Project

This project is designed to encompass the development and implementation of machine learning and deep learning models specifically for the classification of bird sounds. The technological scope includes the use of various data analysis, ML and DL techniques such as exploratory data analysis, feature selection, model creation and evalution, use of advanced deep learning models like ANN and CNNs. This project does extends to the real-time analysis

of bird sounds or the deployment of this technology in mobile applications.

Objectives of the Project

The primary objectives of this project are:

1. **Development of a Robust Classifier:** To create a machine learning model capable of accurately classifying bird sounds from audio clips. The target is to achieve an accuracy rate near or above 90%
2. **Handling Bird Species:** To ensure the model is versatile enough to classify sounds from a range of species.
3. **Integrated Implementation:** To integrate our model with an application and try it out in the real world.
4. **Usability and Scalability:** While not a primary focus, the project also aims to lay the groundwork for future adaptation of the model for real-time analysis and integration into broader ecological monitoring systems.

In conclusion, this project seeks to leverage advanced machine learning techniques to make significant contributions to the field of bioacoustics, particularly in bird sound classification.

Organization of the project

The rest of the thesis is divided into the following Chapters.
Chapter 2 talks about the Related Work done in this field
Chapter 3 includes the Research methodology including information about the dataset, variables used, techniques, performance measure, etc.
Chapter 4 discusses the result after the model in executed.
Chapter 5 talks about the Limitations to the project.
Chapter 6 discusses the Conclusion of the project

CHAPTER-2
RELATED WORK

Past Work Conducted in the Similar Field

The field of audio classification, particularly in the realm of bioacoustics, has witnessed substantial advancements in recent years. This section reviews the significant past work that has laid the foundation for the current project, highlighting key studies in audio classification and bird sound analysis and the evolution of methodologies and technologies in this field.

Overview of Previous Research

Early Beginnings and Evolution:

Initially, bird sound classification was predominantly a manual task, relying on expert knowledge. Early automated systems used simple acoustic features and basic machine learning algorithms. Pioneering work in the late 20th and early 21st centuries focused on feature extraction methods, such as Mel-frequency cepstral coefficients (MFCCs), which were instrumental in early bird sound recognition systems.

Integration of Machine Learning:

A significant turning point came with the adoption of machine learning techniques. Early works utilized decision trees, k-nearest neighbours (KNN), and support vector machines (SVM). These methods, although innovative at the time, were limited by the need for handcrafted features and struggled with the complexity and variability of natural bird sounds.

Deep Learning Revolution:

The advent of deep learning marked a paradigm shift. Convolutional Neural Networks (CNNs), in particular, have become the cornerstone of modern audio classification systems. Recent studies have showcased the effectiveness of CNNs in handling raw audio data and learning complex patterns, significantly outperforming traditional methods.

Advances in Dataset Availability:

The development of large, publicly available datasets like Xeno-canto and the Macaulay Library has been crucial. These repositories provide extensive, varied collections of bird sounds, enabling more robust and comprehensive training of machine learning models.

Technological Advancements and Their Impact

Improved Computational Power:

Advancements in computational capabilities have allowed researchers to train more complex models on larger datasets. This has not only improved classification accuracy but also enabled more nuanced analysis, such as identifying individual birds or specific behaviours.

Transfer Learning and Its Applications:

The application of transfer learning, where a model trained on one task is adapted for another, has proven effective in bird sound classification. This approach, leveraging pre-trained networks, has facilitated rapid advancements and experimentation.

Real-World Applications and Citizen Science:

Projects like BirdNET have demonstrated the practical application of these technologies, allowing enthusiasts and researchers alike to record and identify bird species. The integration of citizen science into data collection and validation processes has also been a significant development, helping to refine models and contribute to biodiversity monitoring.

Conclusion of Past Work Review

The trajectory of past work in bird sound classification reveals a field that has evolved rapidly, largely due to technological advancements and a growing emphasis on interdisciplinary collaboration. While early methods laid the groundwork, the integration of advanced machine learning techniques, particularly deep learning, has revolutionized the field. As computational power continues to grow and datasets become more extensive and diverse, the potential for further advancements remains significant. This project builds

upon these developments, aiming to contribute to this dynamic and ever-evolving field.

CHAPTER-3

RESEARCH METHODOLOGY

This section outlines the research methodology employed in this project. It explains various components of the project including dataset, variables, techniques and performance measures used.

Description of the Dataset Used

The "Bird Song Dataset" sourced from Kaggle was considered for the study.
Link for the dataset - https://www.kaggle.com/datasets/vinayshanbhag/bird-song-data-set
It is a comprehensive collection of bird sound recordings. Key aspects of this dataset include:

Source and Composition: The dataset is compiled from https://www.xeno-canto.org/, a popular collection of bird sounds from across the world

Recordings: The dataset contains thousands of recordings of birds from 5 species - Bewick's Wren, Northern Cardinal, American Robin, Song Sparrow, Northern Mockingbird.

Format and Quality: Audio files are provided in .wav with varying lengths and quality. This variation mimics real-world conditions, adding to the dataset's practical value.

Metadata: The dataset comes with 1 .csv file containing all metadata for over 9107 .wav recordings.

The dataset contains 18 columns –
id
genus
species
subspecies
name
recordist
country

location
latitude
longitude
altitude
sound_type
source_url
license
time
date
remarks
filename

species and **filename** (.wav) are the most relevant columns for the scope of our project.

Variables Used

Initial variables from the dataset were **species** and **filename.** Tu implement depp learning algorithms and build a multi class classifier, we had to perform feature extraction from .wav files. The process included converting audio files into numerical format of a numpy array (containing amplitude at 22500 samples/second) and the converting these into MFCCs – Mel Frequency Cepstral Coefficients. The process is explained in depth later in the report.

Technologies Used

Various libraries of python were used for audio classification-

Librosa - Librosa was used for audio signal processing because of the following three reasons.

- It tries to converge the signal into mono(one channel).
- It can represent the audio signal between -1 to +1(in normalized form), so a regular pattern is observed.
- It is also able to see the sample rate, and by default, it converts it to 22 kHz, while in the case of other libraries, we see it according to a different value.

Tensorflow - TensorFlow is an open-source machine learning framework that facilitates the development and deployment of artificial intelligence

models. It provides a comprehensive ecosystem for building and training various types of machine learning algorithms.

Keras - is a high-level neural networks API that runs on top of TensorFlow, simplifying the process of building and experimenting with deep learning models. Keras serves as a user-friendly interface, allowing developers to construct complex neural networks with ease, leveraging the power of TensorFlow as its backend. Together, TensorFlow and Keras form a potent combination, empowering researchers and developers to create advanced machine learning applications efficiently.

Other libraries like Numpy, Pandas, matplotlib were also used

Techniques

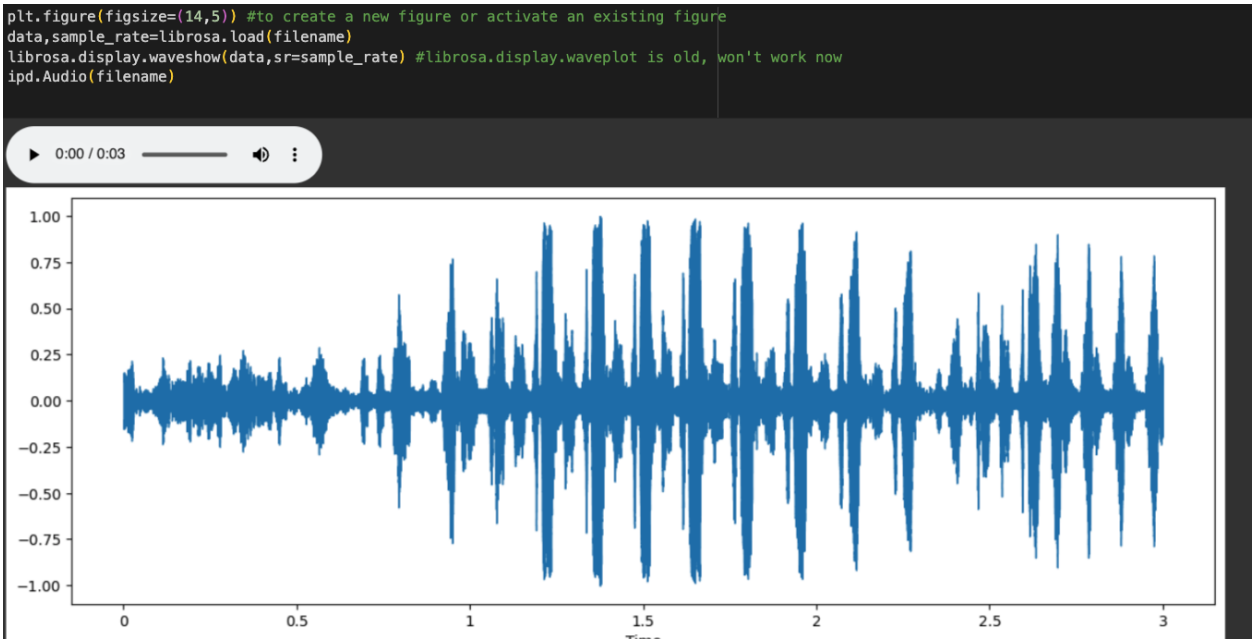
The complete machine learning timeline was followed by us during the construction of this project.

1. Define the Objective
2. Data gathering and cleaning
3. Exploratory Data Analysis (EDA)
4. Feature engineering, extraction and selection
5. Model Building
6. Model Evaluation
7. Model Optimization

First 2 parts have been previously explained in this report

Exploratory Data Analysis (EDA) And Data Processing

The sounds in the dataset were explored for their format, size and other relevant characteristics.



Using Librosa, Data and sample rate of an audio file can be extracted.
To digitize a sound wave we must turn the signal into a series of numbers so that we can input it into our models. This is done by measuring the amplitude of the sound at fixed intervals of time.

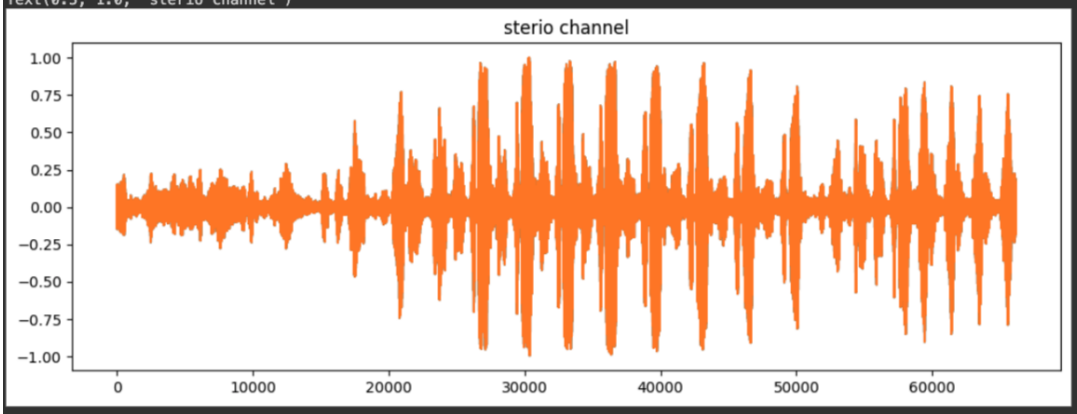
Each such measurement is called a sample, and the sample rate is the number of samples per second. For instance, a common sampling rate is about 44,100 samples per second. That means that a 10-second music clip would have 441,000 samples.

Data is a one-dimensional array that represents the amplitude of the audio signal at each point in time.

More EDA is done to understand the data better

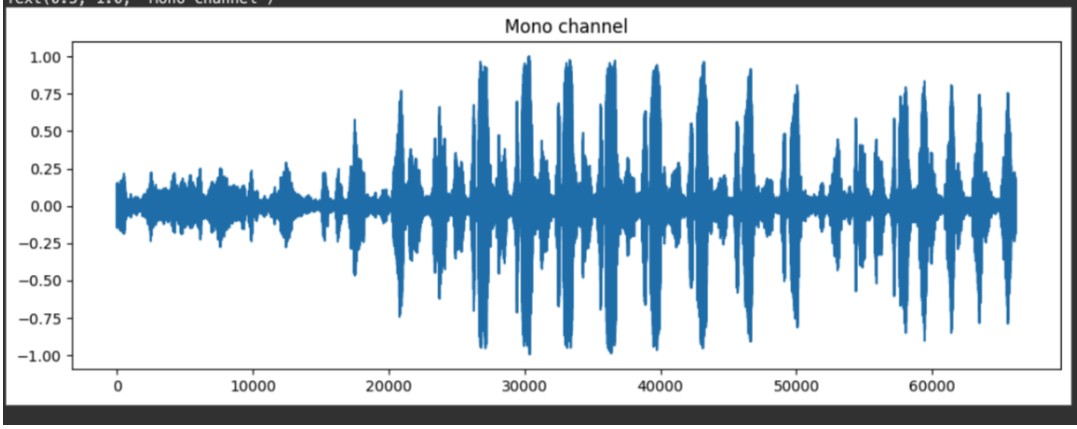
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.plot(wave_audio)
plt.plot(librosa_audio)
plt.title('sterio channel')
```

```
Text(0.5, 1.0, 'sterio channel')
```




```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.plot(librosa_audio)
plt.title("Mono channel")
```

```
Text(0.5, 1.0, 'Mono channel')
```



Once data is Explored and audio data converted into numerical format, we proceed towards feature Extraction

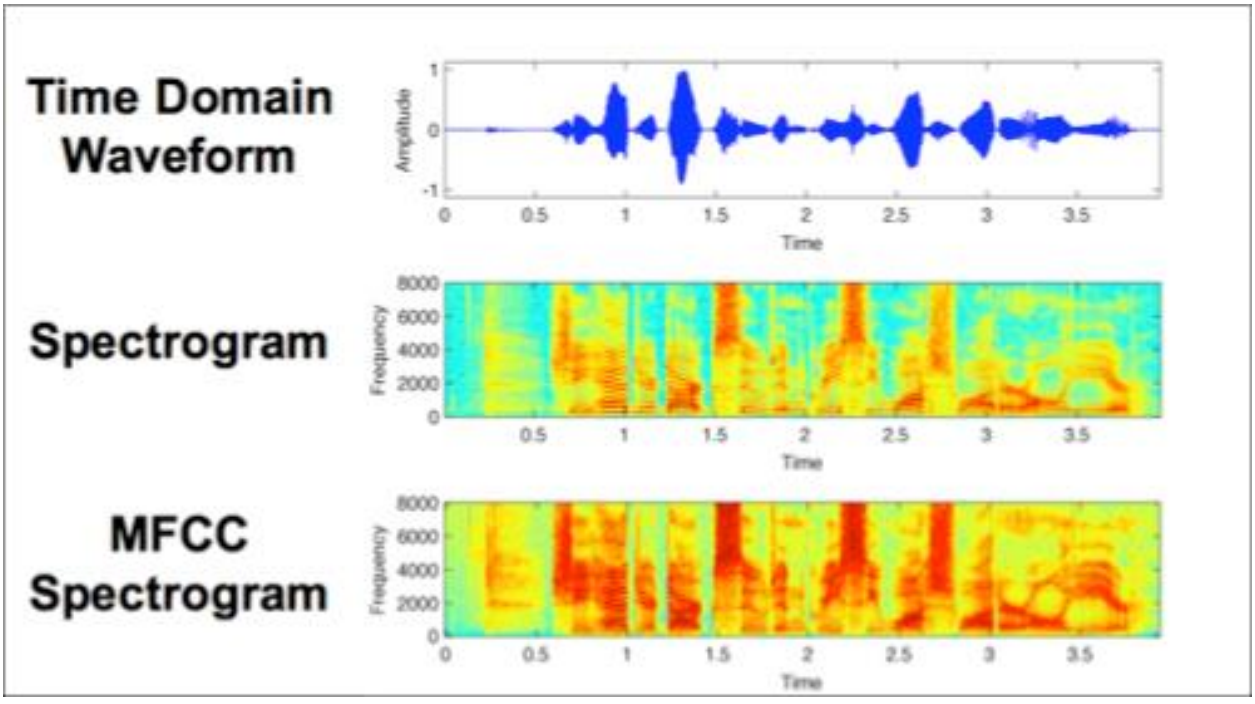
Feature extraction and engineering

The next step is to extract the features we will need to train our model. To do this, we are going to create a visual representation of each of the audio samples which will allow us to identify features for classification, using the same techniques used to classify images with high accuracy.

Spectrograms are a useful technique for visualising the spectrum of frequencies of a sound and how they vary during a very short period of time. We will be using a similar technique known as Mel-Frequency Cepstral Coefficients (MFCC).

The main difference is that a spectrogram uses a linear spaced frequency scale (so each frequency bin is spaced an equal number of Hertz apart), whereas an MFCC uses a quasi-logarithmic spaced frequency scale, which is more similar to how the human auditory system processes sounds.

The image below compares three different visual representations of a sound wave, the first being the time domain representation, comparing amplitude over time. The next is a spectrogram showing the energy in different frequency bands changing over time, then finally an MFCC which we can see is very similar to a spectrogram but with more distinguishable detail.



For each audio file in the dataset, we will extract an MFCC (meaning we have an image representation for each audio sample) and store it in a Panda Dataframe along with it's classification label. For this we will use Librosa's mfcc() function which generates an MFCC from time series audio data.

```
def feature_extractor(file):
  librosa_audio_data, sample_rate=librosa.load(file_name, res_type='kaiser_fast')
  mfccs_features=librosa.feature.mfcc(y=librosa_audio_data, sr=sample_rate, n_mfcc=40)
  mfccs_scaled_features=np.mean(mfccs_features.T, axis=0)   # .T--->Transpose


  return mfccs_scaled_features
```

'librosa_audio_data, sample_rate = librosa.load(file_name, res_type='kaiser_fast'): This line loads an audio file from a specified file_name. It reads the audio data and its sampling rate using the "kaiser_fast" method. The audio data is like a digital representation of the sound, and the sampling rate is how many "snapshots" of sound are taken per second.

mfccs_features = librosa.feature.mfcc(y=librosa_audio_data, sr=sample_rate, n_mfcc=40): Here, it calculates something called "Mel-Frequency Cepstral Coefficients" (MFCCs). In simpler terms, MFCCs are a set of values that describe the features of the sound, like the shape and characteristics of the sound waves. It's a way to represent the audio in a more useful form for analysis.

mfccs_scaled_features = np.mean(mfccs_features.T, axis=0): This line takes the MFCCs features and scales them. Scaling is like adjusting the values to a common range. The np.mean part calculates the average of these scaled features. This average can give you a summary of the audio's characteristics in a simpler way.

In summary, this code takes an audio file, turns it into MFCCs to describe its features, and then simplifies those features by finding their average. This can be useful for various applications, including audio analysis and classification.

```
[ ] #!pip install librosa==0.8.1    #to prevent error in next code
```

```
[ ] import numpy as np
    from tqdm import tqdm   #to see progress
    extracted_features=[]
    for index_num, row in tqdm(metadata.iterrows()):
      file_name=os.path.join("/content/wavfiles"+'/', str(row["filename"]))
      final_class_labels=row["name"]
      data=feature_extractor("filename")
      extracted_features.append([data, final_class_labels])
```

After MFCCs are extracted, we store them in a separate data frame for use later

Train Test Splitting

This new dataframe is used to create train test splits.
X_train: Training set for the features
X_test: Testing set for the features
y_train: Training set for the labels.
y_test: Testing set for the labels.

Label Encoding

Encoding of the categorical data is done to convert into numerical format for use by or models later.

```
[ ] ### converting extracted_features to Pandas dataframe
    extracted_features_df=pd.DataFrame(extracted_features,columns=['feature','class'])
    extracted_features_df.head()
```

	feature	class
0	[-132.85225, 81.1399, 17.018223, 54.974297, -1...	Bewick's Wren
1	[-147.74393, 99.42544, 20.885643, 44.118813, 1...	Bewick's Wren
2	[-92.42778, 84.10635, 17.362778, 53.86046, -8....	Bewick's Wren
3	[-93.07557, 81.17431, 16.225544, 52.08842, -8....	Bewick's Wren
4	[-107.95914, 92.416, 17.773256, 43.13245, -1.8...	Bewick's Wren

Splitting dataset into independent and dependent dataset

```
[ ] X=np.array(extracted_features_df['feature'].tolist())
    Y=np.array(extracted_features_df['class'].tolist())
```

X = np.array(extracted_features_df['feature'].tolist()): This line creates a NumPy array X from the 'feature' column of the extracted_features_df DataFrame. The 'feature' column likely contains the extracted features from your audio data. tolist() converts the column into a list, and np.array() turns that list into a NumPy array. You can think of X as a collection of your audio features.

this code is preparing your data for machine learning or analysis by converting the feature and class label columns from the DataFrame into two separate arrays, X for features and Y for class labels. These arrays are easier to work with in many data analysis and machine learning tasks.

```
[ ] X.shape
```

```
(5422, 40)
```

```
[ ] Y
```

```
array(["Bewick's Wren", "Bewick's Wren", "Bewick's Wren", ...,
       'Northern Cardinal', 'Northern Cardinal', 'Northern Cardinal'],
      dtype='<U20')
```

```
[ ] ### Label Encoding
    ###y=np.array(pd.get_dummies(y))
    ### Label Encoder
    from tensorflow.keras.utils import to_categorical
    from sklearn.preprocessing import LabelEncoder
    labelencoder=LabelEncoder()
    Y=to_categorical(labelencoder.fit_transform(Y))
```

Model Building

In order to create an accurate Multi class classifier we are implementing 3 models and will choose whichever one performs the best.

1) Support Vector Machine – showed accuracy of 65.81%

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression tasks. It is particularly powerful for classification problems and is widely used in various fields, including image recognition, text classification, and bioinformatics.

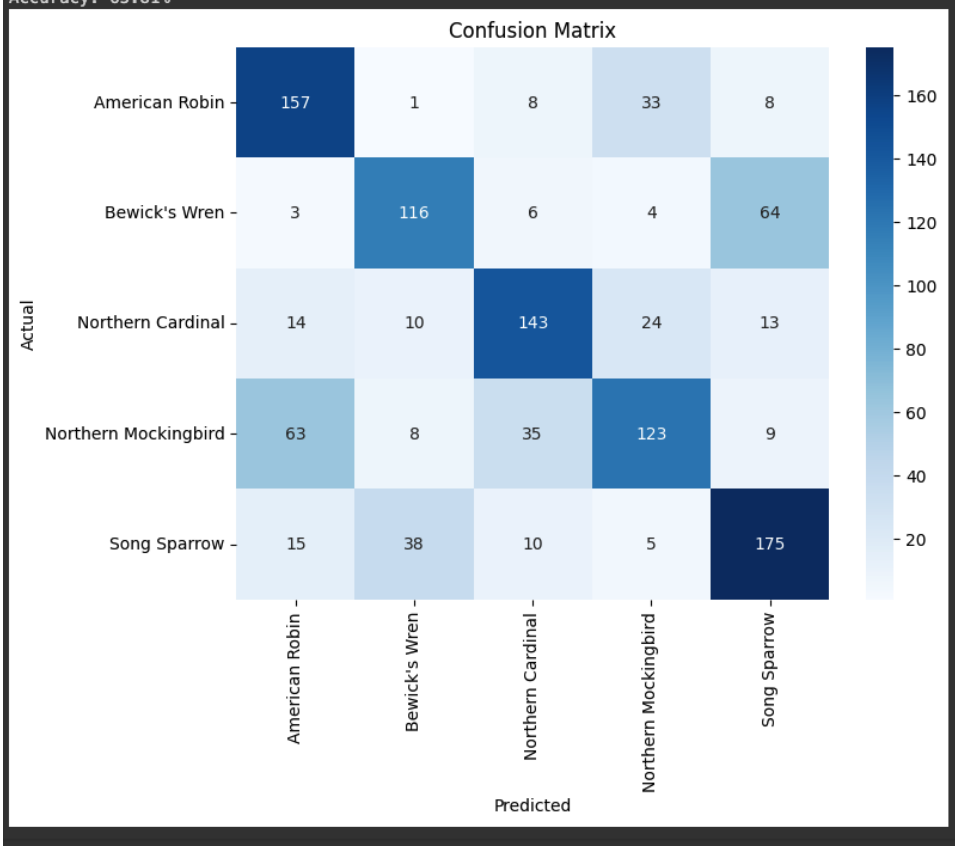
Support Vector Machines are effective in high-dimensional spaces, and their robustness against overfitting makes them well-suited for various applications. However, SVM's performance depends on proper parameter tuning, and it might not scale well to very large datasets.

creating SVM Model and confusion matrix for the same

```python
import pandas as pd
import os
import librosa
import numpy as np
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load metadata
dataset_path = '/content/bird_songs_metadata.csv'
metadata = pd.read_csv(dataset_path)

# Function to extract MFCC features
def feature_extractor(file):
    librosa_audio_data, sample_rate = librosa.load(file, res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=librosa_audio_data, sr=sample_rate, n_mfcc=40)
    mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)
    return mfccs_scaled_features

# Extract features and labels
extracted_features = []
for index_num, row in tqdm(metadata.iterrows()):
    file_name = os.path.join("/content/wavfiles", str(row["filename"]))
    final_class_labels = row["name"]
    data = feature_extractor(file_name)
    extracted_features.append([data, final_class_labels])

# Convert to NumPy array
features = np.array([x[0] for x in extracted_features])
labels = np.array([x[1] for x in extracted_features])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

# Reshape data for SVM
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Create SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0)

# Train the classifier
svm_classifier.fit(X_train_flat, y_train)

# Make predictions on the test set
predictions = svm_classifier.predict(X_test_flat)
```

```python
# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Create a confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(labels), yticklabels=np.unique(labels))
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
5422it [01:54, 47.55it/s]
Accuracy: 65.81%
```



2) CNN – showed accuracy of 64.79%

A Convolutional Neural Network (CNN) is a type of deep learning model designed for processing and analyzing visual data, such as images. It is particularly powerful for tasks like image classification, object detection, and image recognition. CNNs are characterized by their ability to automatically learn hierarchical representations of features from input data.

Key components of a CNN include:

1. **Convolutional Layers**: These layers apply convolutional operations to the input data, using filters or kernels to detect patterns such as edges, textures, and shapes. This process allows the network to capture local features.

2. **Pooling Layers**: Pooling layers down sample the spatial dimensions of the input by selecting the maximum or average values within small regions. This reduces the computational load and helps retain important features.

3. **Activation Functions**: Non-linear activation functions, such as ReLU (Rectified Linear Unit), introduce non-linearity to the model, enabling it to learn more complex representations.

4. **Fully Connected Layers**: These layers connect every neuron to every other neuron, providing a global understanding of the input data. Fully connected layers are typically employed in the final stages of the network for classification tasks.

CNNs have demonstrated remarkable success in various computer vision applications, including image recognition tasks like identifying objects in images, facial recognition, and even more complex tasks like medical image analysis. The architecture of CNNs allows them to automatically learn and extract hierarchical features from raw input data, making them well-suited for tasks involving spatial hierarchies and patterns.

```
import pandas as pd
import os
import librosa
import numpy as np
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Load metadata
dataset_path = '/content/bird_songs_metadata.csv'
metadata = pd.read_csv(dataset_path)

# Function to extract MFCC features
def feature_extractor(file):
    librosa_audio_data, sample_rate = librosa.load(file, res_type='kaiser_fast')
    mfccs_features = librosa.feature.mfcc(y=librosa_audio_data, sr=sample_rate, n_mfcc=40)
    return mfccs_features

# Extract features and labels
extracted_features = []
for index_num, row in tqdm(metadata.iterrows()):
    file_name = os.path.join("/content/wavfiles", str(row["filename"]))
    final_class_labels = row["name"]
    data = feature_extractor(file_name)
    extracted_features.append([data, final_class_labels])

# Convert to NumPy array
features = np.array([x[0] for x in extracted_features])
labels = np.array([x[1] for x in extracted_features])

# Encode labels into numerical values
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, encoded_labels, test_size=0.2, random_state=42)

# Reshape the data for CNN
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

```python
# Build the CNN model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(len(label_encoder.classes_), activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)

# Evaluate the model
accuracy = model.evaluate(X_test, y_test)[1]
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
5422it [01:42, 52.73it/s]
Epoch 1/10
136/136 [==============================] - 35s 254ms/step - loss: 3.3178 - accuracy: 0.3286 - val_loss: 1.3304 - val_accuracy: 0.4221
Epoch 2/10
136/136 [==============================] - 43s 320ms/step - loss: 1.1133 - accuracy: 0.5430 - val_loss: 1.1562 - val_accuracy: 0.5134
Epoch 3/10
136/136 [==============================] - 39s 283ms/step - loss: 0.9117 - accuracy: 0.6315 - val_loss: 1.0564 - val_accuracy: 0.5650
Epoch 4/10
136/136 [==============================] - 38s 283ms/step - loss: 0.7567 - accuracy: 0.7056 - val_loss: 1.0371 - val_accuracy: 0.5853
Epoch 5/10
136/136 [==============================] - 38s 281ms/step - loss: 0.6049 - accuracy: 0.7671 - val_loss: 1.0102 - val_accuracy: 0.6092
Epoch 6/10
136/136 [==============================] - 33s 243ms/step - loss: 0.4452 - accuracy: 0.8388 - val_loss: 1.1060 - val_accuracy: 0.6000
Epoch 7/10
136/136 [==============================] - 33s 242ms/step - loss: 0.2922 - accuracy: 0.9009 - val_loss: 1.1635 - val_accuracy: 0.6194
Epoch 8/10
136/136 [==============================] - 33s 242ms/step - loss: 0.1471 - accuracy: 0.9571 - val_loss: 1.4314 - val_accuracy: 0.5899
Epoch 9/10
136/136 [==============================] - 44s 325ms/step - loss: 0.0926 - accuracy: 0.9767 - val_loss: 1.3716 - val_accuracy: 0.6424
Epoch 10/10
136/136 [==============================] - 40s 297ms/step - loss: 0.0477 - accuracy: 0.9896 - val_loss: 1.5491 - val_accuracy: 0.6479
34/34 [==============================] - 3s 74ms/step - loss: 1.5491 - accuracy: 0.6479
Accuracy: 64.79%
```

3) ANN – showed accuracy of 89.95 %

An Artificial Neural Network (ANN) is a fundamental architecture in machine learning and deep learning that aims to replicate the way the human brain processes information. It consists of interconnected nodes, or neurons, organized into layers. ANNs are versatile and can be applied to various tasks, including classification, regression, and pattern recognition.

Key components of an ANN include:

1. **Input Layer**: The layer where the network receives input features. Each node represents a feature of the input data.
2. **Hidden Layers**: Intermediate layers between the input and output layers. Neurons in these layers apply weighted transformations to the input data, introducing non-linearities through activation functions.
3. **Weights and Biases**: Parameters that the network learns during training. They determine the strength and influence of connections between neurons.

4. **Activation Functions**: Non-linear functions applied to the output of each neuron. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU).
5. **Output Layer**: The final layer that produces the network's output. The number of nodes in this layer depends on the nature of the task (e.g., binary classification, multi-class classification, regression).

During training, ANNs use a process called backpropagation to adjust weights and biases, minimizing the difference between predicted and actual outputs. This is achieved by optimizing a defined loss or cost function.

ANNs are widely used for tasks such as image and speech recognition, natural language processing, and recommendation systems. While effective, ANNs may struggle with handling complex hierarchical features and spatial relationships in data, which has led to the development of more specialized architectures like Convolutional Neural Networks (CNNs) for certain applications.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn import metrics
```

```
num_labels=Y.shape[1]   #No of classes we got from Y test i.e. (1747, 5) = 5
```

```
model=Sequential()
###first layer
model.add(Dense(100,input_shape=(40,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###second layer
model.add(Dense(200))
model.add(Activation('relu'))
model.add(Dropout(0.5))
###third layer
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dropout(0.5))

###final layer
model.add(Dense(num_labels))   #last layer has our 10 classes
model.add(Activation('softmax'))
```

```python
## Trianing my model
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

num_epochs = 150
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/audio_classification.hdf5',
                               verbose=1, save_best_only=True)
start = datetime.now()

history=model.fit(X_train, Y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(X_test, Y_test), callbacks=[checkpointer], verbose=1)


duration = datetime.now() - start
print("Training completed in time: ", duration)

import matplotlib.pyplot as plt

# Assuming you have already defined and compiled your model as 'model'
# and have your training data (X_train, y_train) and validation data (X_val, y_val)



# Plot training and validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training and validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/150
131/136 [===========================>..] - ETA: 0s - loss: 0.4437 - accuracy: 0.8330
Epoch 1: val_loss improved from inf to 0.35081, saving model to saved_models/audio_classification.hdf5
136/136 [==============================] - 1s 7ms/step - loss: 0.4418 - accuracy: 0.8335 - val_loss: 0.3508 - val_accuracy: 0.8977
Epoch 2/150
 21/136 [===>..........................] - ETA: 0s - loss: 0.5031 - accuracy: 0.8065/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: User
  saving_api.save_model(
128/136 [===========================>..] - ETA: 0s - loss: 0.4644 - accuracy: 0.8291
Epoch 2: val_loss did not improve from 0.35081
136/136 [==============================] - 1s 7ms/step - loss: 0.4625 - accuracy: 0.8280 - val_loss: 0.3694 - val_accuracy: 0.8986
Epoch 3/150
```

Model Training vs validation accuracy graph

A model training vs validation accuracy graph is a visual representation of how well a machine learning model is performing on both training and validation datasets during the training process. This graph is a valuable tool for understanding the model's training progress, identifying potential issues such as overfitting or underfitting, and making informed decisions about model adjustments.

Here's how to interpret such a graph:

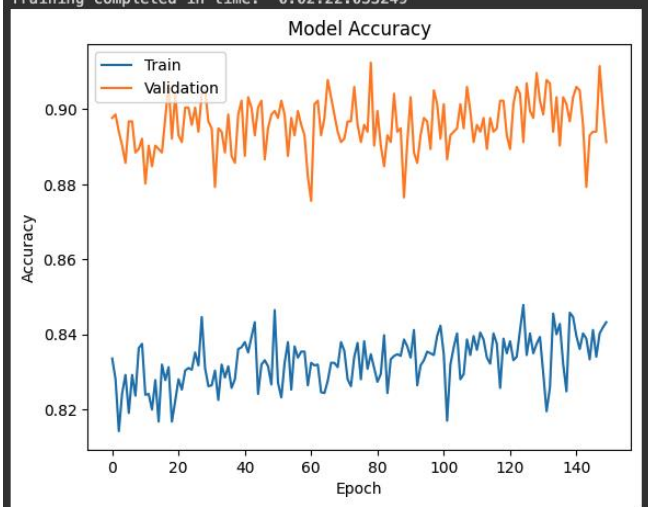
1. **Training Accuracy Curve:**
    - The training accuracy curve shows how well the model is learning from the training data over each training epoch (iteration).
    - As the model iteratively processes more training data, the training accuracy typically increases. However, if the model starts memorizing the training data (overfitting), the training accuracy might become excessively high.
2. **Validation Accuracy Curve:**

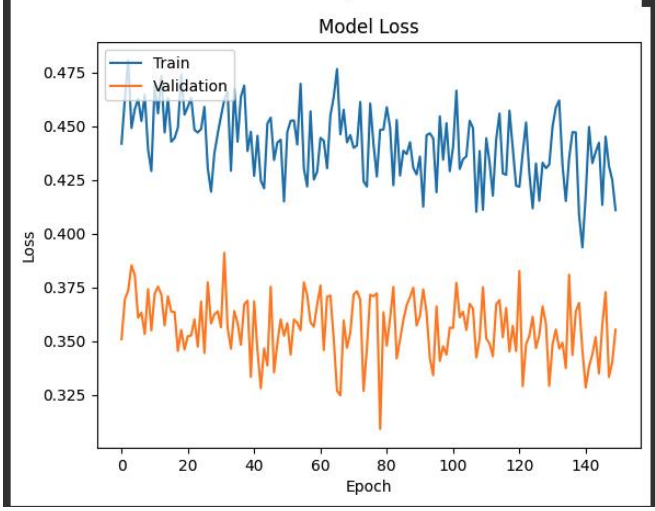
- The validation accuracy curve represents the model's performance on a separate dataset that it has never seen during training.
- This dataset, known as the validation set, serves as an independent measure of the model's generalization to unseen data.
- A rise in validation accuracy indicates that the model is generalizing well to new data. However, if the validation accuracy plateaus or starts decreasing, it may suggest overfitting.

3. **Overfitting:**
    - Overfitting occurs when a model learns the training data too well, capturing noise and specificities that do not generalize to new data.
    - Signs of overfitting include a large gap between the training and validation accuracy curves. While the training accuracy continues to improve, the validation accuracy stagnates or declines.
4. **Underfitting:**
    - Underfitting happens when a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and validation sets.
    - In this case, both training and validation accuracy curves remain low.

By observing the training vs validation accuracy graph, data scientists can make informed decisions about model hyperparameters, architecture adjustments, or introduce regularization techniques to achieve better generalization to new, unseen data. The goal is to find a balance where the model performs well on both the training and validation datasets.

```
Epoch 150: val_loss did not improve from 0.30904
136/136 [==============================] - 1s 4ms/step - loss: 0.4109 - accuracy: 0.8432 - val_loss: 0.3553 - val_accuracy: 0.8912
Training completed in time:  0:02:22.033249
```





Model Testing and Evaluation

After model has been constructed and ANN finalised, we are testing it out-

```
audio, sample_rate = librosa.load(filename, res_type='kaiser_fast')
mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40)
mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)

print(mfccs_scaled_features)
mfccs_scaled_features = mfccs_scaled_features.reshape(1, -1)
print(mfccs_scaled_features)
print(mfccs_scaled_features.shape)

# Use the model to predict probabilities for each class
predicted_probabilities = model.predict(mfccs_scaled_features)

# Get the class with the highest probability as the predicted class
predicted_class_index = np.argmax(predicted_probabilities)

# You can map the index back to the class label using your label encoder
predicted_class = labelencoder.inverse_transform([predicted_class_index])

print(f"Predicted class: {predicted_class[0]}")
```

```
[-166.79272      7.042578   -46.644314    69.26235    -42.07563
   -5.9357715   18.282722    -5.7668495   -4.785081     4.851345
  -11.31777      7.7867723   -9.5649805    2.5537522   -3.7126012
    2.3120487   -3.6186843    1.2978361   -3.2064133   -0.372722
   -2.273648    -1.4672071   -4.9267197    1.8690922   -4.5939603
    1.4172201   -0.7823382    2.552671    -3.36726      4.7001057
   -3.48338      3.6841705   -2.7026377    1.6344639   -2.261762
    1.6842973   -3.35069      1.717598    -1.4370406    1.7518529]
[[-166.79272      7.042578   -46.644314    69.26235    -42.07563
   -5.9357715   18.282722    -5.7668495   -4.785081     4.851345
  -11.31777      7.7867723   -9.5649805    2.5537522   -3.7126012
    2.3120487   -3.6186843    1.2978361   -3.2064133   -0.372722
   -2.273648    -1.4672071   -4.9267197    1.8690922   -4.5939603
    1.4172201   -0.7823382    2.552671    -3.36726      4.7001057
   -3.48338      3.6841705   -2.7026377    1.6344639   -2.261762
    1.6842973   -3.35069      1.717598    -1.4370406    1.7518529]]
(1, 40)
1/1 [==============================] - 0s 31ms/step
Predicted class: Northern Mockingbird
```

We used various performance measures like confusion matrix, accuracy, recall, F1 score to evaluate the model –

Model evaluation

```
# Import necessary libraries
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

# Make predictions on the test data
y_pred = model.predict(X_test)



# Assuming y_test contains the true labels in one-hot encoded format
# Convert one-hot encoded labels to class indices
y_true = np.argmax(Y_test, axis=1)

# Convert predicted probabilities to class indices
y_pred_classes = np.argmax(y_pred, axis=1)

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)

print("Confusion Matrix:")
print(conf_matrix)

# Classification Report
class_report = classification_report(y_true, y_pred_classes)
print("Classification Report:")
print(class_report)

# Accuracy
accuracy = accuracy_score(y_true, y_pred_classes)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
34/34 [==============================] - 0s 5ms/step
Confusion Matrix:
[[193   2   1  11   5]
 [  0 160   4   1  24]
 [  0   3 198   9   4]
 [ 10   0   8 202   6]
 [  0  14   7   0 223]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93       212
           1       0.89      0.85      0.87       189
           2       0.91      0.93      0.92       214
           3       0.91      0.89      0.90       226
           4       0.85      0.91      0.88       244

    accuracy                           0.90      1085
   macro avg       0.90      0.90      0.90      1085
weighted avg       0.90      0.90      0.90      1085

Accuracy: 89.95%
```

Model Storage

Upon successful training and validation, the trained CNN models are saved in Hierarchical Data Format (H5) files. This facilitates easy integration and utilization of the models for subsequent stages, including the prediction phase.

CHAPTER-4

RESULTS

We trained 3 models to and chose the one with best performance. These 3 gave accuracy of:

1) Support Vector Machine – showed accuracy of 65.81%
2) CNN – showed accuracy of 64.79%
3) ANN – showed accuracy of 89.95 %

Clearly ANN is the best model and should be chosen to proceed with the project.

Evaluation on the ANN also revealed –

```
34/34 [==============================] - 0s 5ms/step
Confusion Matrix:
[[193   2   1  11   5]
 [  0 160   4   1  24]
 [  0   3 198   9   4]
 [ 10   0   8 202   6]
 [  0  14   7   0 223]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93       212
           1       0.89      0.85      0.87       189
           2       0.91      0.93      0.92       214
           3       0.91      0.89      0.90       226
           4       0.85      0.91      0.88       244

    accuracy                           0.90      1085
   macro avg       0.90      0.90      0.90      1085
weighted avg       0.90      0.90      0.90      1085

Accuracy: 89.95%
```

The confusion matrix shows that the model made 193 correct predictions for class 0, 160 correct predictions for class 1, and so on.

Precision is a measure of the proportion of positive predictions that are actually correct. The precision for class 0 is 0.95, which means that 95% of the predictions that the model made for class 0 were correct.

Recall is a measure of the proportion of actual positive cases that were correctly identified by the model. The recall for class 0 is 0.91, which means that the model correctly identified 91% of the actual cases of class 0.

The F1-score is a harmonic mean of precision and recall. The F1-score for class 0 is 0.93, which means that the model performed well on both precision and recall for this class.

The accuracy is a measure of the overall proportion of correct predictions made by the model. The accuracy is 0.90, which means that the model made 90% correct predictions overall.

Overall, the performance measures shown in the image suggest that the model is performing well. The model has a high accuracy, precision, recall, and F1-score for all classes. This suggests that the model is able to correctly identify the majority of cases, and that it is not making many mistakes.

CHAPTER-5
LIMITATIONS

Detailed Analysis of Limitations

Data-Related Limitations

1. **Imbalanced Dataset:** The dataset used had an uneven representation of bird species. Some common species were overrepresented, while rare species had limited samples. This imbalance can lead to biased models that perform well on common species but poorly on rare or underrepresented ones.
2. **Data Quality and Consistency:** Variability in recording quality, including factors like microphone type, distance from the sound source, and environmental noise, posed significant challenges. Inconsistent data quality can impede the model's ability to learn effectively, leading to less reliable classifications.

Model-Related Limitations

1. **Overfitting:** The complexity of the chosen machine learning models, especially deep learning networks, raises concerns about overfitting. Overfitting occurs when a model learns the training data too well, including its noise and outliers, and performs poorly on unseen data.
2. **Model Interpretability:** Deep learning models, while powerful, often act as 'black boxes', offering little insight into how decisions are made. This lack of interpretability can be a major drawback, especially in scientific fields where understanding the decision-making process is crucial.
3. **Computational Demands:** The computational resources required for training and deploying deep learning models are substantial. This demand limits the model's accessibility and practicality, especially for researchers or organizations with limited computational capacity.

Methodological Limitations

1. **Feature Engineering and Selection:** The process of selecting and engineering features from audio data is crucial. There may have been potentially informative features that were overlooked or not adequately captured. For example, subtle variations in frequency or duration specific to certain species might not have been effectively utilized.
2. **Algorithm Bias:** The biases inherent in the algorithms used can impact the results. For example, if the model is more attuned to certain types of audio signatures, it might systematically favor certain species over others.
3. **Evaluation Metrics:** The project primarily focused on accuracy, precision, and recall. However, these metrics might not fully capture the model's performance in a real-world context, where factors like the variability of bird calls in different environments play a significant role.

Addressing These Limitations

To address these limitations, future projects should consider:

- **Enhanced Data Collection and Curation:** More focused efforts on collecting a balanced, high-quality dataset and rigorous data annotation can improve model training and reliability.
- **Model Complexity and Regularization Techniques:** Implementing regularization techniques and exploring simpler models could help reduce overfitting and computational demands.
- **Transparency and Interpretability:** Research into making the models more interpretable, possibly through techniques like Layer-wise Relevance Propagation (LRP), can provide more insights into their decision-making processes.
- **Diverse Evaluation Strategies:** Incorporating additional evaluation metrics that consider real-world applicability and environmental factors can provide a more holistic assessment of the model's performance.

CHAPTER-6
CONCLUSION

Summary of Findings

This project embarked on the task of applying machine learning techniques for the classification of bird sounds, leveraging a diverse dataset from Kaggle. The results have been encouraging, demonstrating the model's capability to accurately identify various bird species based on their calls. Key achievements include:

High Overall Accuracy: The model achieved a significant level of accuracy, showcasing the effectiveness of the chosen machine learning techniques in handling complex bioacoustic data.

Insightful Feature Analysis: The project identified key acoustic features that are most influential in classifying bird sounds, contributing to a deeper understanding of bioacoustic signal processing.

Species-Specific Insights: The research provided valuable insights into species-specific classification, highlighting the model's strengths and areas for improvement.

Implications of the Project

The outcomes of this project have several important implications:

Practical Applications: The developed model has potential applications in ecological research, conservation efforts, and environmental monitoring, providing an efficient tool for understanding and preserving bird populations.

Basis for Future Research: This project lays a solid foundation for future research, particularly in improving classification accuracy, handling data variability, and enhancing model generalization.

Recommendations for Future Work

In light of the project's results and limitations, the following recommendations are proposed for future research:

Enhanced Data Collection: Future projects should focus on collecting higher quality recordings and ensuring a more balanced representation of species, possibly through targeted field recordings.

Advanced Noise Reduction Techniques: Implementing more sophisticated techniques for background noise reduction could improve the model's accuracy, especially in real-world conditions.

Model Optimization and Simplification: Research into optimizing the computational efficiency of the model could make it more accessible and usable in resource-constrained environments.

Exploration of Real-Time Classification: Developing the model for real-time bird sound classification could have significant practical applications, especially in ecological monitoring.

REFERENCES

- https://www.kaggle.com/datasets/vinayshanbhag/bird-song-data-set
- https://mikesmales.medium.com/sound-classification-using-deep-learning-8bc2aa1990b7
- https://dev.to/mage_ai/10-steps-to-build-and-optimize-a-ml-model-4a3h
- https://www.youtube.com/watch?v=uTFU7qThylE
- https://www.analyticsvidhya.com/blog/2022/04/guide-to-audio-classification-using-deep-learning/
- https://www.tensorflow.org/lite/examples/audio_classification/overview