BIRD SPECIES CLASSIFICATION USING AUDIO RECORDINGS

Submitted by

Amitha V

Group members :

Adithya Gagarin M P

Ankitha K

Krishnaveni T M

Sreelekshmi B

CONTENTS

1. Introduction

1.1. Project Overview

The primary objective of this project is to classify bird species based on their vocalizations using sound recognition techniques. By analysing bird calls and songs, we can automatically identify different species, which is valuable for ecological research, biodiversity monitoring, and conservation efforts.

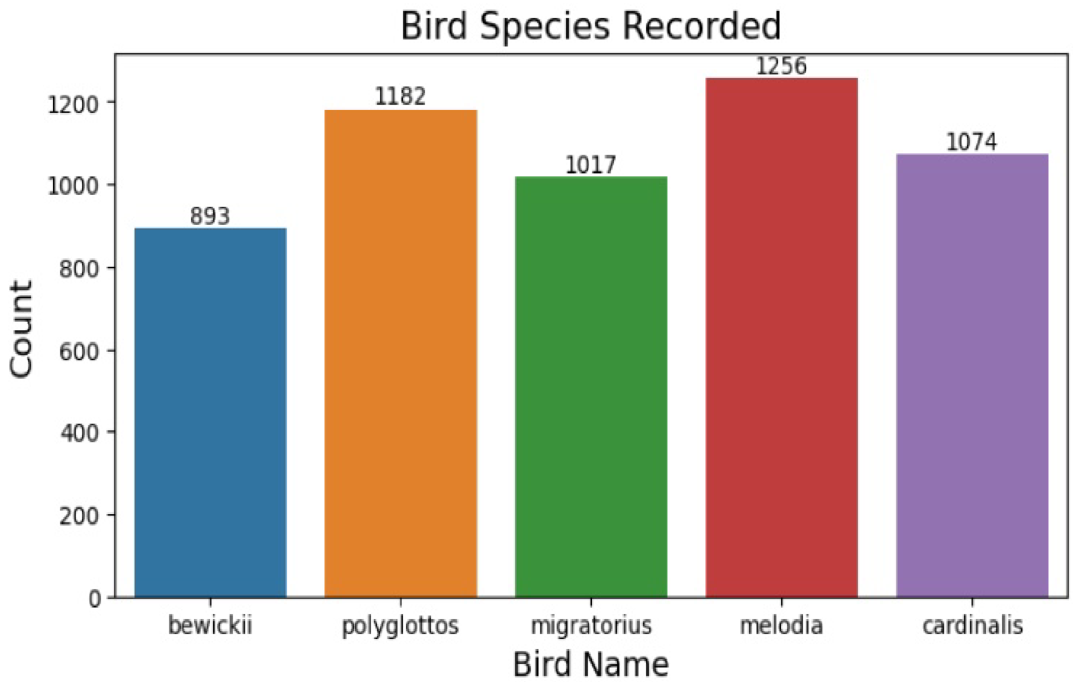
1.2. Objectives

- To develop an automated system that identifies bird species from audio recordings.
- To achieve high accuracy in bird species classification using machine learning techniques.
- To create a Python-based implementation that can be easily used and extended.

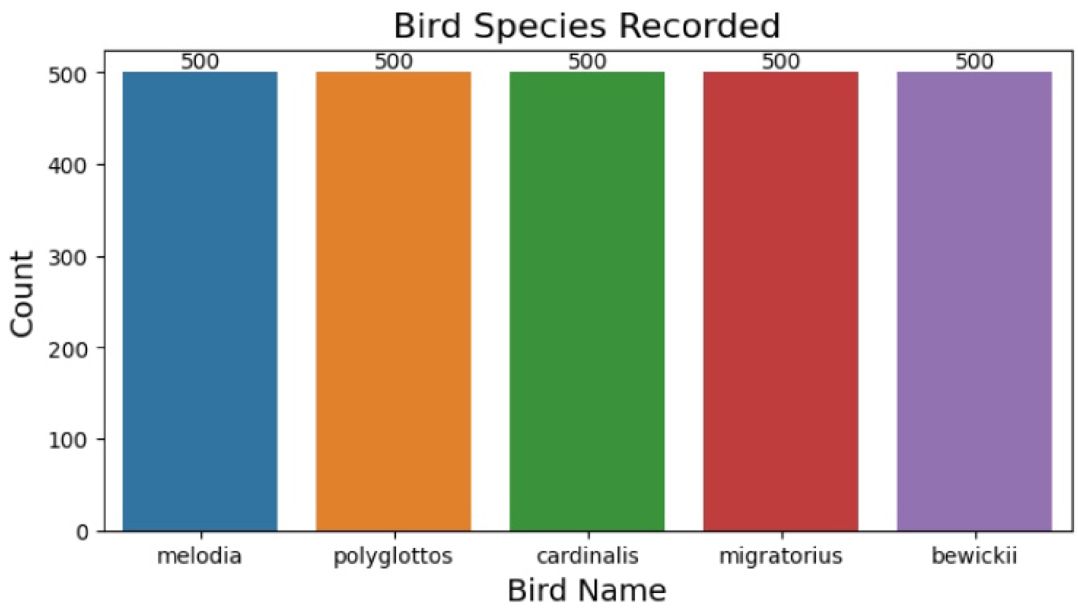
2. Methodology

2.1. Data Collection and Preprocessing

Audio data was sourced from the https://www.kaggle.com/datasets/jayaprakashpondy/birds-sound-dataset/data, which contains a total of 5422 recordings from 5 different bird species.



We selected a dataset comprising 2500 recordings with 500 audio clips per species.

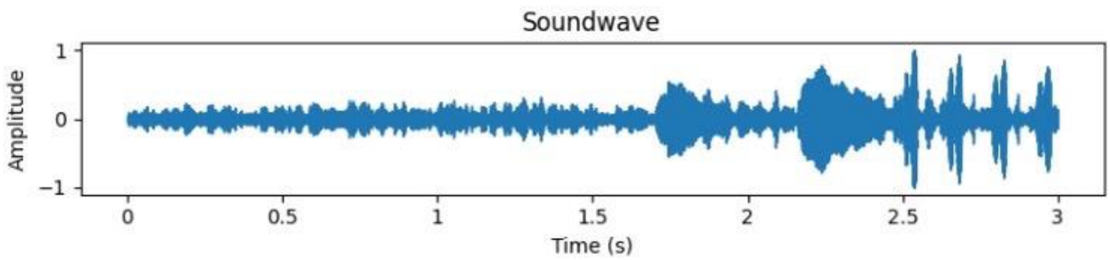


2.2. Feature Extraction

Using the Librosa library, we extracted Mel-Frequency Cepstral Coefficients (MFCCs), Mel spectrogram, Spectral bandwidth, Spectral centroid, RMS from each audio file.

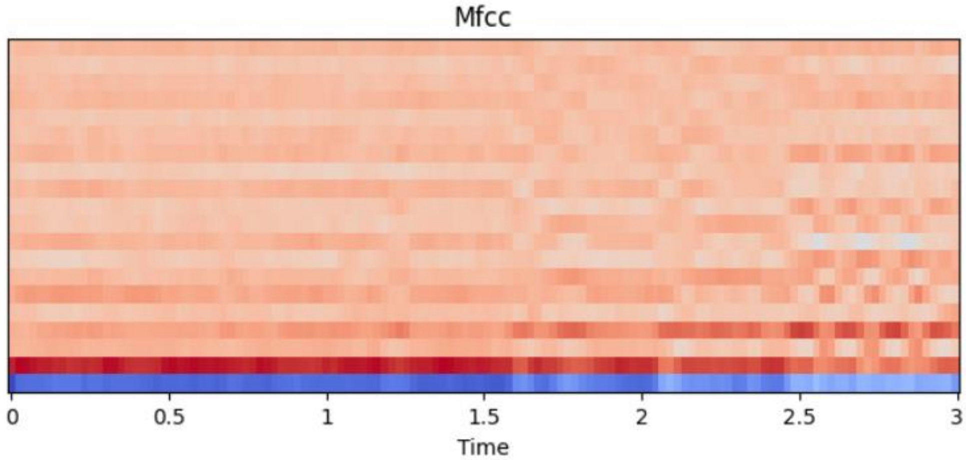
- **Soundwave**

  Sound waves are vibrations that travel through a medium, such as air, and can be characterized by properties like frequency, amplitude, and wavelength. These properties determine the pitch, loudness, and tone of the sound, respectively.



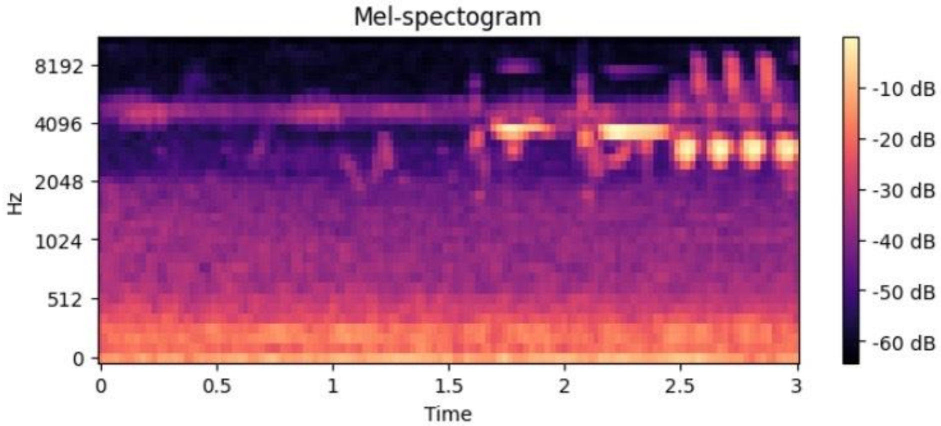
- **Mel-Frequency Cepstral Coefficients (MFCCs):**

  MFCCs are widely used in audio analysis as they effectively capture the power spectrum of a sound, making them useful for tasks such as speech and music recognition. It represent the short-term power spectrum of a sound, which is useful in distinguishing different audio signals.



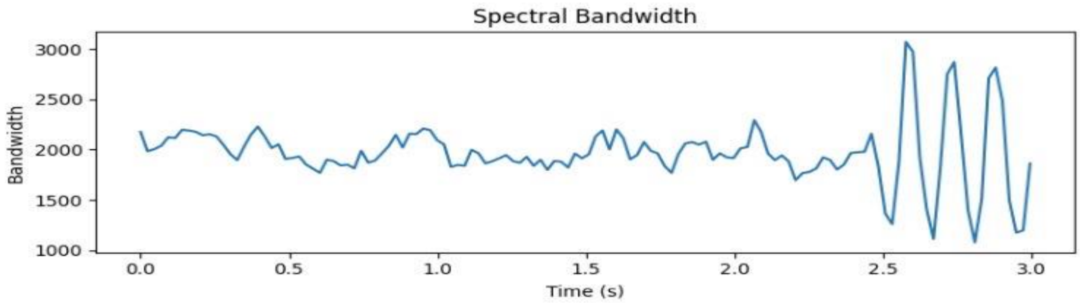
- **Mel spectrogram :**

  Visualization of the power distribution of audio frequencies, transformed into the mel scale to better represent human perception of sound.



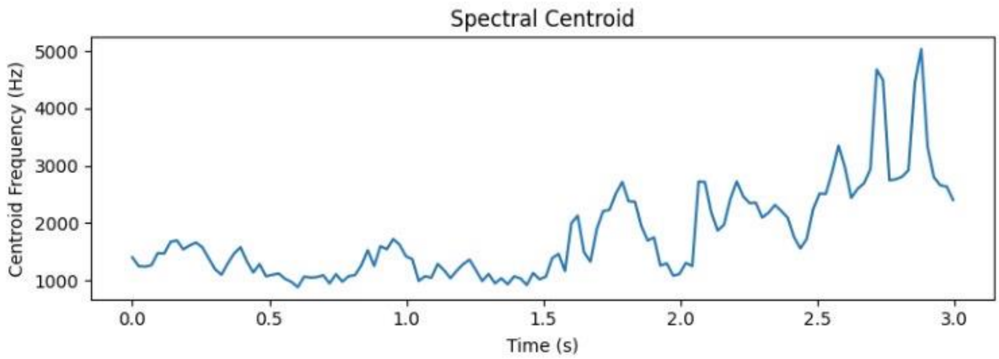
- **Spectral Bandwidth:**

  Spectral Bandwidth measures the width of the spectrum and can give insights into the timbre of the sound. It is calculated as the spread of the spectrum around its centroid.



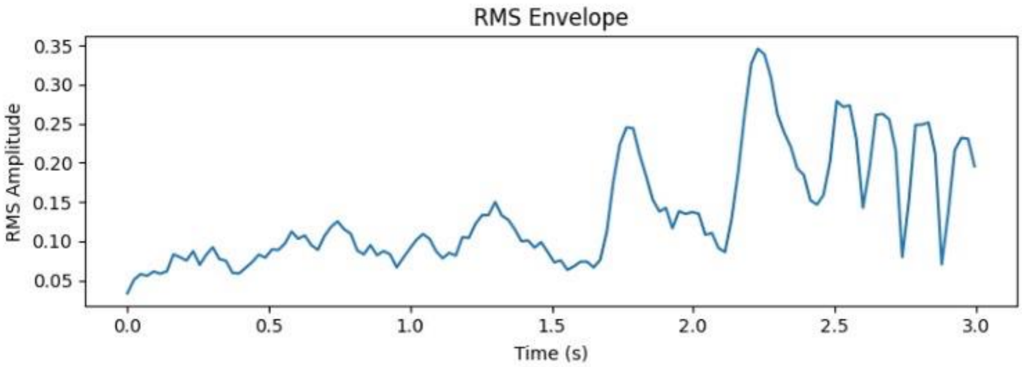
- **Spectral Centroid :**

  The spectral centroid is a measurement used in digital signal processing to characterize a sound's spectrum. It essentially indicates the "center of gravity" of the sound's frequency content.



- **RMS :**

  Root Mean Square is a measurement used in audio to quantify the average power or loudness of an audio signal over time.



3. Models Used

3.1. Support Vector Machines (SVMs) :

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression tasks. SVMs aim to find the hyperplane that best separates data points into different classes while maximizing the margin (distance) between the classes. This hyperplane is the decision boundary.

The margin is the distance between the hyperplane and the nearest data points from each class (support vectors). SVMs seek to maximize this margin because larger margins often lead to better generalization to unseen data. SVMs can efficiently perform non-linear classification using a technique called the kernel trick. This allows them to transform the input space into a higher-dimensional space where a linear separation may be possible.

Types of SVMs :

- Linear SVM : Used for linearly separable data.
- Non-linear SVM : Utilizes kernels (e.g., polynomial, radial basis function (RBF)) to handle non-linear decision boundaries.
- Support Vector Regression (SVR) : Extends SVM for regression tasks, aiming to fit as many instances as possible within a specified margin.

Advantages:

- Effective in high-dimensional spaces and when the number of dimensions exceeds the number of samples.
- Memory efficient due to their use of a subset of training points (support vectors) in the decision function.
- Versatile with different kernel functions for handling complex decision boundaries.

Disadvantages :

- Computational inefficiency for large datasets.
- Difficulty in choosing an appropriate kernel function and regularization parameters.
- Can be sensitive to noise in the data.

Applications : Text categorization, Image classification, Handwriting recognition, Bioinformatics, Finance (e.g., stock market forecasting)

3.2. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric, and instance-based learning algorithm used for classification and regression tasks. It works on the principle that data points close together are likely to have similar properties. To classify a new data point, KNN finds the **k** closest data points (neighbors) from the training data set.

For classification and regression:

The algorithm stores the entire training dataset. When given a new data point, it calculates the distances between the new point and all the points in the training data. It identifies the k nearest neighbors based on the chosen distance metric (like Euclidean distance).

- <u>For classification tasks:</u> The class label of the new data point is assigned based on the majority vote of its k nearest neighbors.
- <u>For regression tasks</u>, KNN predicts the value of a new data point by averaging the values of its k nearest neighbors.

Key Characteristics:

- <u>Simplicity</u>: Easy to implement and understand.
- <u>No Training Phase</u>: Unlike many other models, KNN does not have a training phase. All the training data is stored, and predictions are made in real-time.
- <u>Scalability Issues</u>: High memory consumption and slower prediction times with large datasets since all data must be stored and scanned for each prediction.

Applications: Pattern recognition, Handwriting detection, Image recognition, Recommendation systems

3.3. Random Forest

Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Random Forest combines the predictions of multiple decision trees to produce a single output. Each tree is trained on a random subset of the data (with replacement), promoting diversity among the trees. At each split in the tree, a random subset of features is considered, reducing correlation among trees.

For classification and regression:

Decision Trees are base learners that are grown using recursive binary splits based on feature values. For classification, the mode of the predictions from all trees is taken, while for regression, the mean of all predictions is used.

Key Characteristics:

- Robustness: Reduces overfitting by averaging multiple trees, which improves generalization.
- Interpretability: Feature importance can be evaluated, providing insights into which features are influential in the predictions.
- Scalability: Efficiently handles large datasets and high-dimensional spaces.

Applications: Fraud detection, Stock market analysis, Medical diagnosis, Customer segmentation, Bioinformatics

3.4. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a specialized type of deep learning model designed for processing and analysing structured grid-like data, such as images and videos. It can be also used for numerical data with inherent spatial relationships like time series (sequential data).

Architecture :

- Convolutional layers : These layers apply convolution operations to input data, extracting features through filters (kernels) that slide over the input spatial dimensions.
- Pooling layers : Pooling (e.g., max pooling, average pooling) reduces the spatial dimensions of the input, preserving important information while reducing computational complexity.
- Fully Connected layers: Traditionally placed at the end of CNNs, these layers perform classification based on the features extracted by earlier layers.

Feature Learning :

CNNs excel in learning hierarchical representations of features. Lower layers detect basic features like edges and textures, while higher layers combine these features to recognize complex patterns and objects. Common activation functions like ReLU (Rectified Linear Unit) are used in CNNs to introduce non-linearity, enabling the network to learn complex mappings between inputs and outputs.

Applications : CNNs are widely used in computer vision tasks, including: Image classification, Object detection and localization, Image segmentation, Facial recognition, Medical image analysis, Autonomous driving

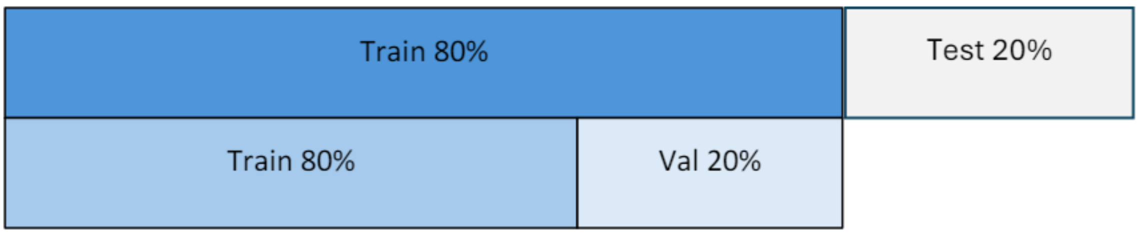
4. Model Training

4.1. Train and Split

In the process of developing machine learning models for audio classification, various audio features such as MFCC, RMS, etc. were extracted and utilized. These features, including MFCC (Mel-Frequency Cepstral Coefficients), bandwidth, spectral centroid, and RMS, were applied in the implementation of four models, namely Support Vector Classifier (SVC), Random Forest (RF), K-Nearest Neighbors (KNN), and 1D Convoluted Neural Network (CNN).

The dataset was split into training and test data for the first three models - SVC, RF, and KNN.

However, for the CNN model, the data was divided into train (X_train, Y_train) and test data. The train data was further split into train (x_train, y_train) and validation data. The CNN model was trained using the train (x_train, y_train) and validation data, and then tested with the test data.



4.2. Cross-Validation

Cross-validation was carried out to assess the generalization ability of the models. The cross_val_score function was used for performing Stratifiedk-fold cross-validation. 5-fold crossvalidation was performed on SVC, RF, and KNN to ensure a more robust estimate of the model's performance on unseen data.

StratifiedKFold is a variation of k-fold which returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set. This technique involves splitting the training data into folds, training the model on a subset of folds (training set) and evaluating its performance on the remaining folds (validation set). This process is repeated for all folds.

4.3. Scaling

To improve the performance of the classification model, the training and test data were scaled using StandardScaler. StandardScaler is a commonly used technique that normalizes features by subtracting the mean and dividing by the standard deviation. It transforms each feature in your dataset to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the model's learning process.

Additionally, scalar statistics (mean and standard deviation) are calculated. This is then used to scale the data for prediction.

4.4. Model Parameters

4.4.1. Support Vector Machines:

The SVC model was trained with hyperparameters:

- C: Regularization parameter set to 1.
- gamma: Kernel coefficient set to 0.1.

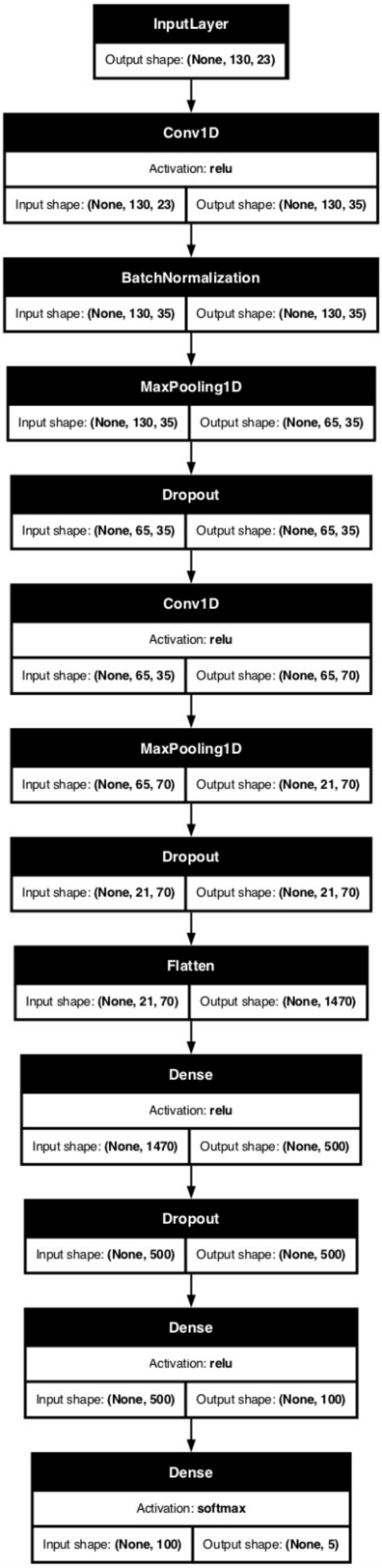
4.4.2. K-Nearest Neighbors:

To determine the optimal value for n_neighbors, hyperparameter tuning is performed by running the KNN_classifier function with different values of n (ranging from 1 to 9) and evaluating the model's accuracy on the testing data.

Seaborn (sns.pointplot) is utilized to visualize the accuracy scores for different n values . This plot helps us identify the optimal number of neighbors that yields the best performance for our bird sound classification task. Here the optimal value of **k** is found to be 3 which is used for further actions.

4.4.3. Random Forest:

n_estimators : Controls the number of trees. Higher number of trees leads to better performance but increases the training time. Here n_estimators is set to 300, which provides good balance between accuracy and computational efficiency.



4.4.4. Convolutional Neural Network:

The implemented model has the following components:

- **Input Layer:** The model takes audio features as input, represented by a 3D tensor where the first dimension represents the number of samples (audio snippets) and the second and third dimensions represent the feature vector size for each sample.
- **Convolutional Blocks:** The model utilizes two convolutional blocks: Each block consists of a Conv1D layer with 35 and 70 filters in the first and second blocks, respectively. A batch normalization layer is incorporated in the first block for improved stability.
- **MaxPooling1D** layer down samples the data in each block, reducing the dimensionality.
- **Dropout layers** (0.2 and 0.3 dropout rate) for regularization to prevent overfitting.
- **Dense Layers:** After flattening the output from the convolutional blocks, two fully connected (dense) layers with 500 and 100 units are added with ReLU activation for non-linearity with the first layer followed by a dropout layer (0.5 rate).
- **Output Layer:** The final layer has number of units (5 in this case) and uses softmax activation for multi-class classification.

Model Compilation and Training:

The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss (suitable for multi-class classification with integer labels), and accuracy metric.
The fit function trains the model using the training data (x_train, y_train) for 30 epochs and validation data (x_val, y_val) to monitor performance during training and prevent overfitting.

4.5. Model Fitting and Prediction

The model is trained on the entire training set using the fit method. The trained model is then used to predict the class labels for the unseen test data using the predict method.

4.6. Performance Evaluation

Various metrics were employed to evaluate the performance of the model on the test set.

- **Accuracy**: It is the most basic metric, representing the proportion of correctly classified bird sounds. The accuracy_score function calculates the overall accuracy.
- **F1-score**: This metric combines precision and recall, providing a more balanced view of model performance. A macro average is used here, which calculates the F1 score for each class and then averages them.
- **Precision**: It measures the proportion of predicted positive labels that are truly positive. The weighted average precision across all classes is calculated.
- **Recall**: It measures the proportion of actual positive labels that are correctly identified by the model. Like precision, weighted average recall is calculated.
- **Classification report**: Generated using the classification_ report function from sklearn metrics. This report provides insights into the model's performance for each bird species class.

4.7. Advanced Analysis

- **Confusion Matrix**: Visualizes the distribution of actual vs. predicted labels. The diagonal entries indicate the number of correctly predicted instances whereas off-diagonal entries show misclassified instances. This is a helpful tool for identifying classes where the model struggles and potential areas for improvement.
- **Precision-Recall Curve**: Precision-recall curve for each bird species is generated. This curve allows you to visualize the trade-off between precision and recall at different classification thresholds.
- **Loss Plots (CNN)**: The loss plot shows how the model's loss (error) on the training and validation data changes with each training epoch (iteration). Ideally, the training loss should steadily decrease as the model learns. A large gap between training and validation loss indicates overfitting.
- **Accuracy Plots (CNN)**: The accuracy plot shows how the model's performance (percentage of correct predictions) on the training and validation data changes with each epoch. A significant difference between training and validation accuracy suggests overfitting.

5. Web deployment with Flask and HTML

A web application for bird sound classification is developed with flask and HTML. Users can upload audio recordings of bird sounds, and the application utilizes a pre-trained machine learning model to identify the bird species.

5.1. Technologies Used:

- **Backend:** Flask (Python web framework)
- **Frontend:** HTML
- **Audio Processing:** Librosa (Python library)
- **Machine Learning Model:** Pre-trained 1D Convolutional Neural Network (CNN) for bird sound classification
- **Deployment:** Deployed locally on port 5000

7.2. Implementation:

This handles user interactions, audio file processing, model prediction, and result generation. The Flask application utilizes several key functionalities:

- **File Upload and Audio Preprocessing:** Users can upload audio files for file extension for supported audio formats (e.g., WAV, MP3) through an HTML form. Librosa is used to load the audio, extract features (MFCCs, spectral centroid, bandwidth, RMS), and perform scaling using pre-loaded normalization parameters.
- **Model Loading and Prediction:** A pre-trained CNN model is loaded using Pickle. Pre-calculated mean and standard deviation for model normalization are also loaded. The pre-processed audio features are fed to the loaded model for prediction. The predicted class is converted to the corresponding bird species label.
- **Result Display:** Based on the predicted class, the application renders a specific HTML page displaying the identified bird species.

5.3 Backend

Flask is a popular web framework written in Python. It’s known for its simplicity and flexibility, making it a great choice for building web applications of all sorts.

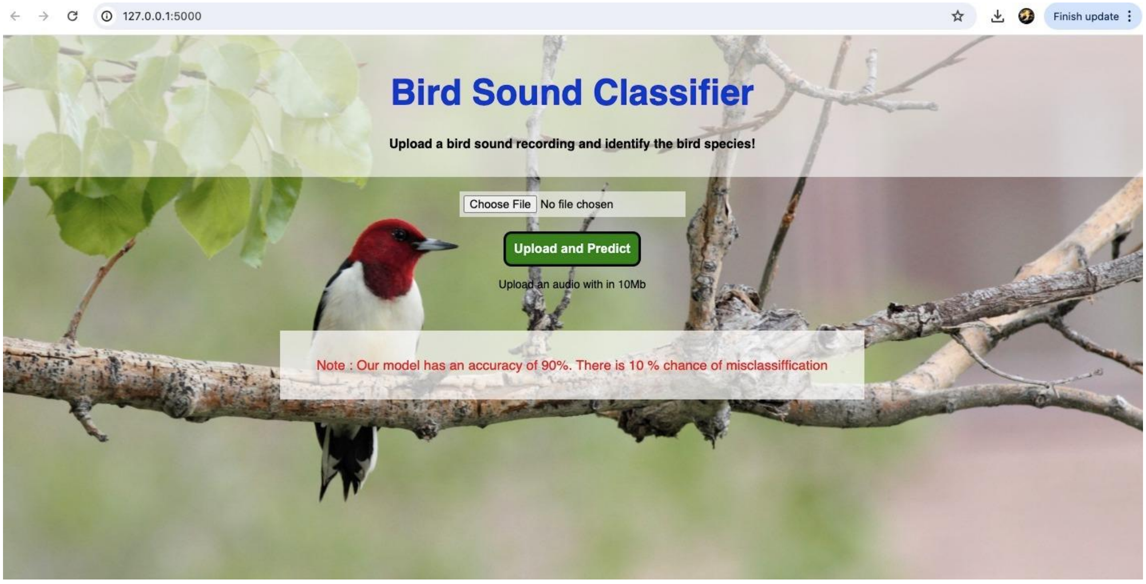
Key features of Flask:

- <u>Simple and Minimalist</u>: Flask’s core is small and easy to understand, making it a good choice for beginners.
- <u>Flexible</u>: You can extend Flask with third-party libraries to add features like database access, form validation, and user authentication.
- <u>Versatile</u>: Flask can be used to build a variety of web applications, from simple static websites to complex APIs and interactive web apps.

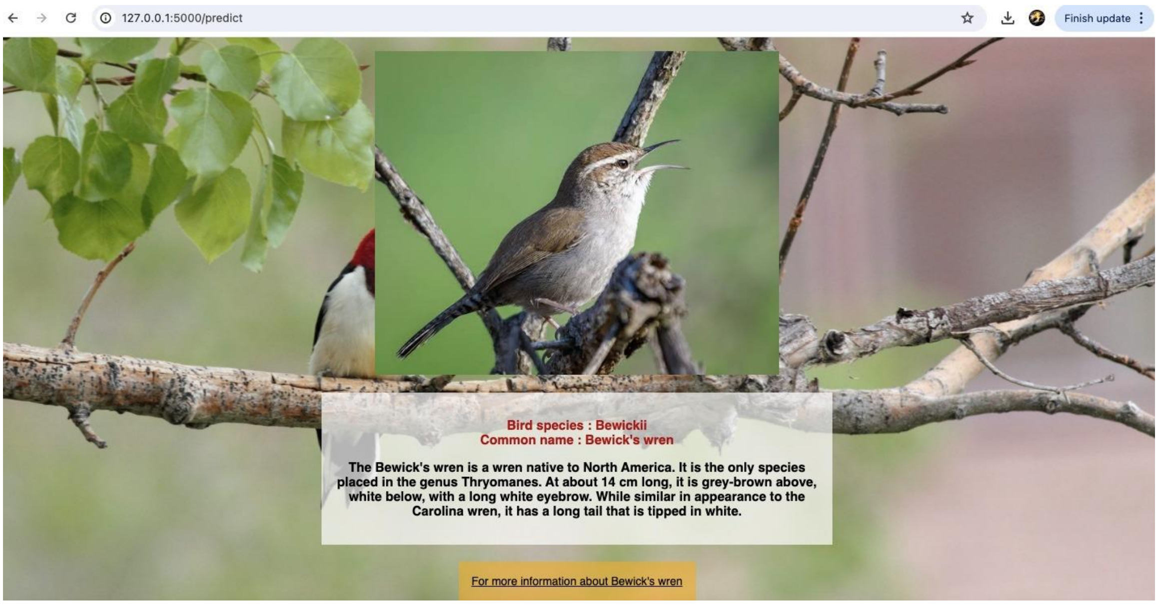
5.4. Frontend Design

The HTML code provides a user-friendly interface with the following elements:

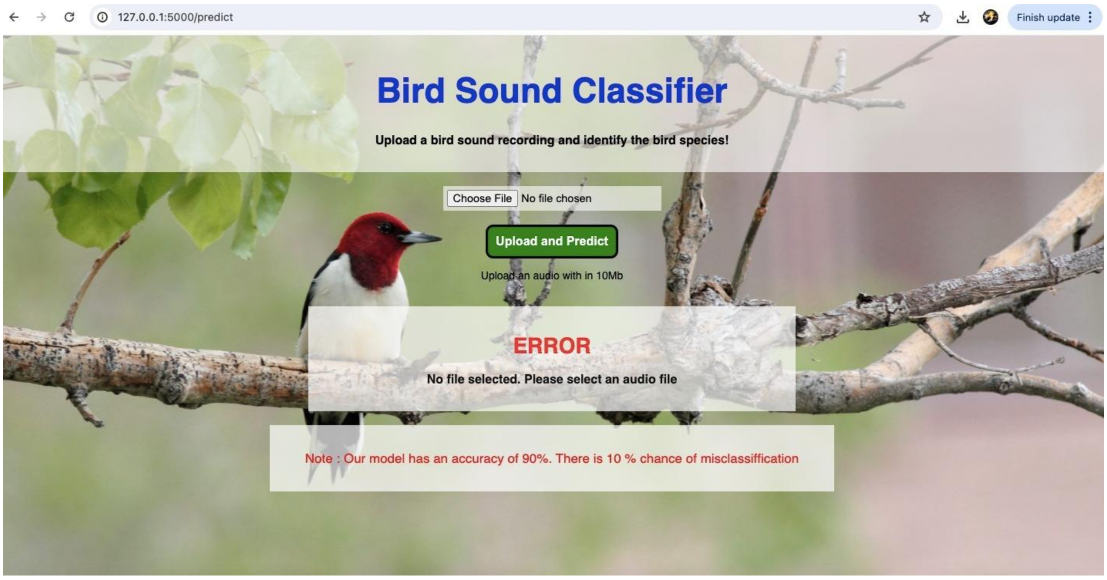
- **Homepage:** Introduces the application with a title, description, and upload instructions. Provides users with the option to select an audio file for classification.



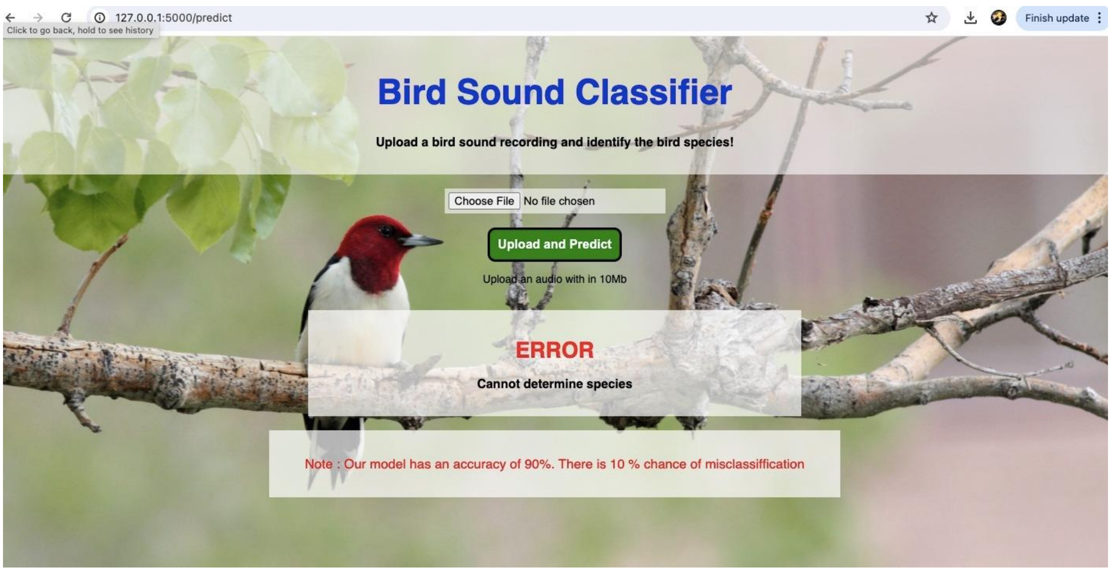
- **Result Pages:** Separate HTML pages exist for each bird species, displaying the identified bird along with relevant information and images.



- **Error Handling:** The application displays informative error messages if no file is selected, unsupported format is used.



If the probability of the prediction is less than 70%, this error is shown.



6. Results and Analysis

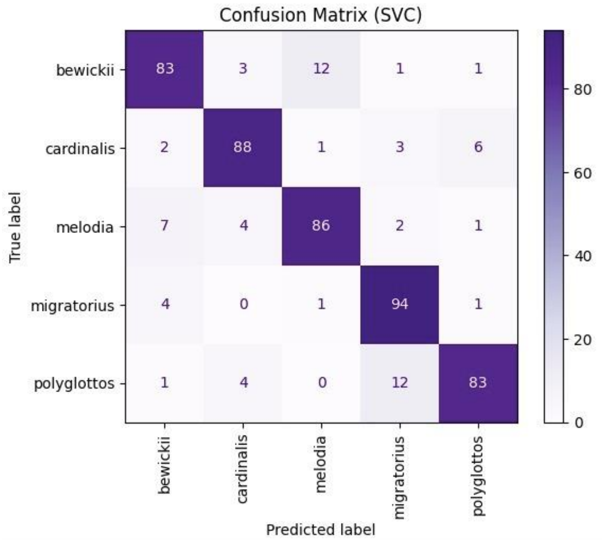
6.1. Support Vector Classifier

Test Accuracy of Support Vector Algorithm: 0.868
F1-score : 0.8677
Precision : 0.8692Recall : 0.868

Classification report:

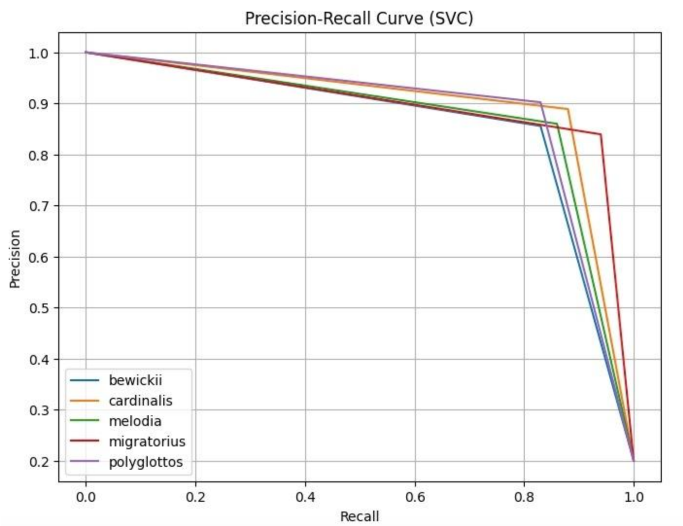
Here the f1 score is roughly equal which indicates the model isn't favouring one class over the other in predictions

	precision	recall	f1-score	support
bewickii	0.86	0.83	0.84	100
cardinalis	0.89	0.88	0.88	100
melodia	0.86	0.86	0.86	100
migratorius	0.84	0.94	0.89	100
polyglottos	0.90	0.83	0.86	100
accuracy			0.87	500
macro avg	0.87	0.87	0.87	500
weighted avg	0.87	0.87	0.87	500



Confusion matrix:

The number of correct predictions is slightly higher for migratorius. Higher misclassification for bewicki and polyglottus.



Precision-recall curve:

The curve indicates the performance is good for all the classes.

6.2. K-Nearest Neighbors

Accuracy : 0.84
F1-score : 0.84
Precision : 0.86
Recall : 0.84

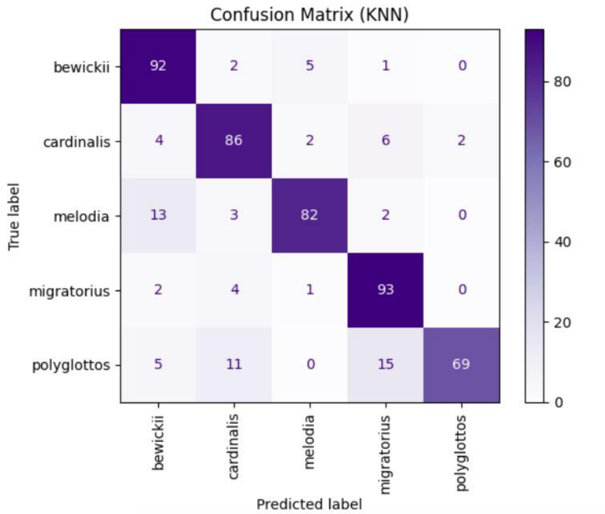
	precision	recall	f1-score	support
bewickii	0.79	0.92	0.85	100
cardinalis	0.81	0.86	0.83	100
melodia	0.91	0.82	0.86	100
migratorius	0.79	0.93	0.86	100
polyglottos	0.97	0.69	0.81	100
accuracy			0.84	500
macro avg	0.86	0.84	0.84	500
weighted avg	0.86	0.84	0.84	500

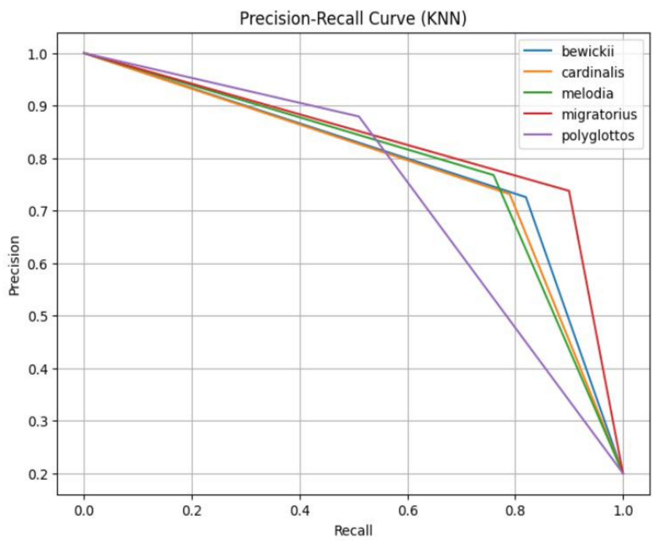
Classification report:

Here the high precision and low recall (e.g. polyglottos) indicates a lot of false positives. Similarly, low precision and high recall (e.g. bewikki) suggests some actual positives are missing.

Confusion matrix:

The number of correct predictions is significantly lower for polyglottus in the KNN model.





Precision-recall curve:
Among the species polyglottus has worst performance.

6.3. Random Forest Classifier

Accuracy: 0.85
F1-score : 0.8496
Precision : 0.8518
Recall : 0.85

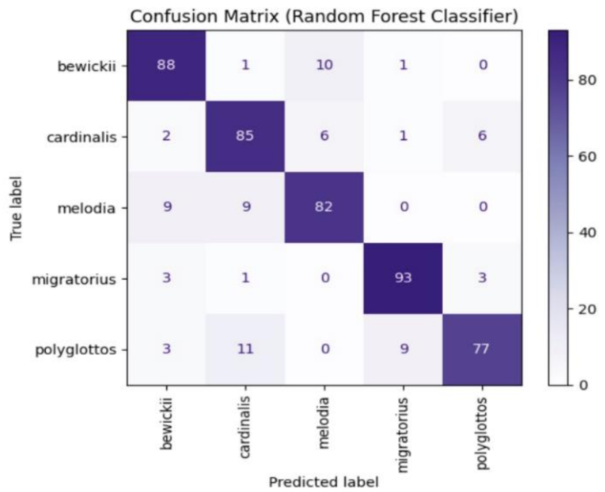
	precision	recall	f1-score	support
bewickii	0.84	0.88	0.86	100
cardinalis	0.79	0.85	0.82	100
melodia	0.84	0.82	0.83	100
migratorius	0.89	0.93	0.91	100
polyglottos	0.90	0.77	0.83	100
accuracy			0.85	500
macro avg	0.85	0.85	0.85	500
weighted avg	0.85	0.85	0.85	500

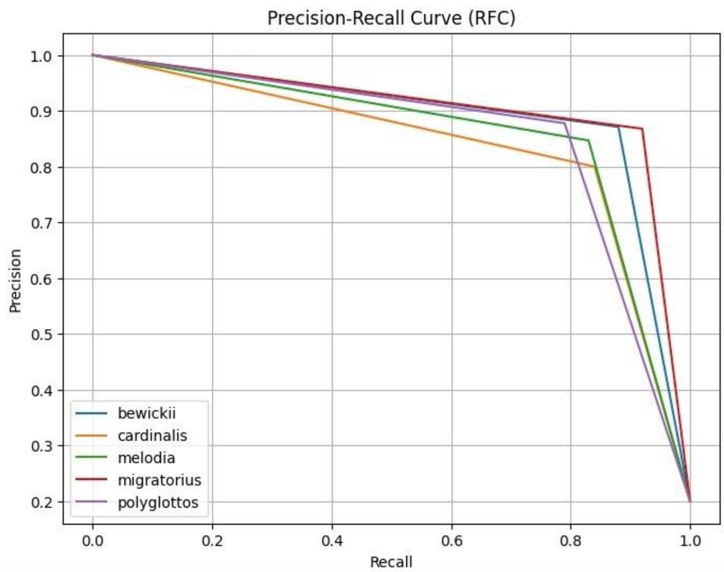
Classification report:

Polyglottus has high precision and low recall which indicates some actual positives are missing.

Confusion matrix:

The number of correctly predicted instances is low for polyglottos and high for migratorius.





Precision-recall curve:

Since all the curves are roughly near it indicates good performance with migratorius performing the best.

6.4. CNN

- Test Accuracy: 0.90

	precision	recall	f1-score	support
bewickii	0.89	0.93	0.91	100
cardinalis	0.93	0.90	0.91	100
melodia	0.91	0.89	0.90	100
migratorius	0.90	0.87	0.88	100
polyglottos	0.88	0.91	0.89	100
accuracy			0.90	500
macro avg	0.90	0.90	0.90	500
weighted avg	0.90	0.90	0.90	500

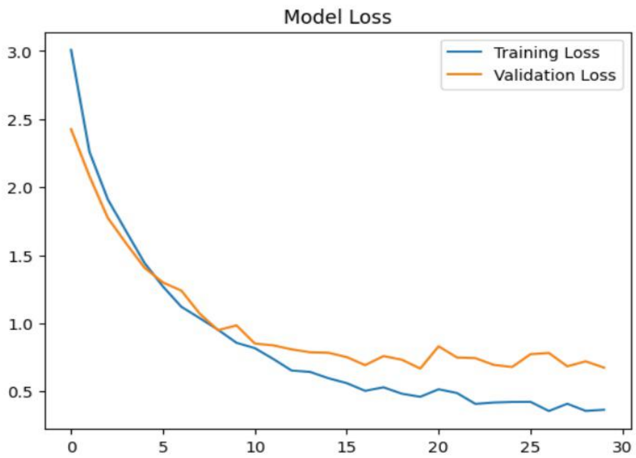
Classification report:

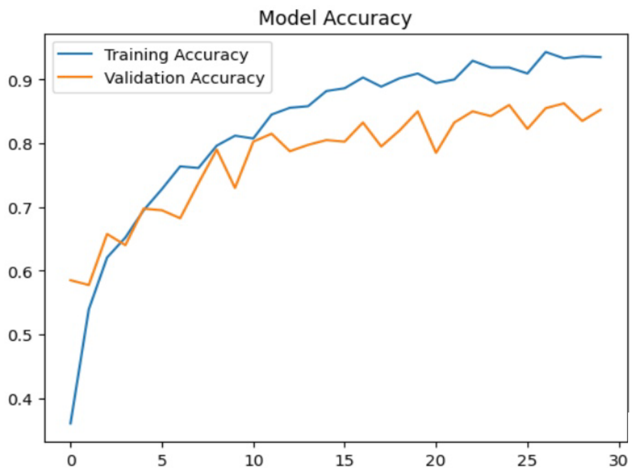
Here F1 score is close which suggests the model performs well in identifying true positives and avoid false positives.

Loss Plot:

The training loss exhibits a steady decrease, indicating successful learning.

Validation loss follows a similar trend with slight fluctuations, suggesting the model generalizes well on unseen data.





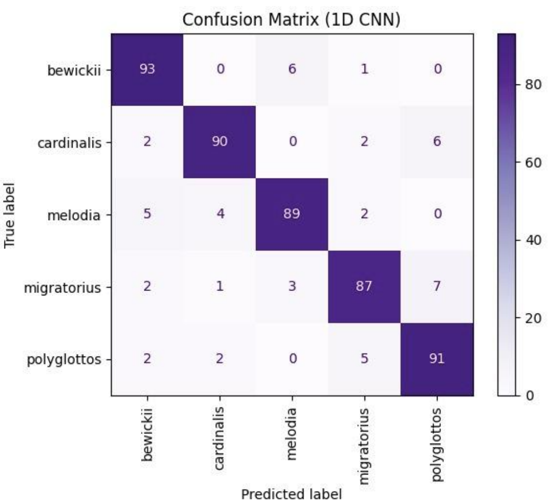
Accuracy Plot:

The training accuracy in this plot increases steadily, reflecting the model's ability to learn from the training data.

Validation accuracy also shows a positive trend, confirming the model's effectiveness on unseen data.

Confusion matrix:

The number of correctly predicted instances are nearly same for all species and misclassification is slightly more for migratorius.



7. Conclusion

7.2. Summary

This project demonstrated the feasibility of classifying bird species based on their vocalizations using a different model. The system achieved good accuracy and provided a foundation for further research in automated bird sound recognition.

From the various models used CNN has the best potential to predict bird species on unseen data. For CNN the accuracy is 90% and the F1 score is high which indicates good overall performance.

7.2. Future Work

- **Improving Dataset:** Collecting more diverse and extensive audio samples.
- **Advanced Models:** Exploring RNNs and hybrid models for better performance.
- **Real-time Classification:** Developing a real-time classification system for field use.

8. References

- Kaggle: A community-driven database of shared bird sound recordings.
- Research papers on audio feature extraction and deep learning in bioacoustics.
- Tensorflow.org
- Scikit-learn.org
- W3schools.com

Appendix - I

Exploratory Data Analysis (EDA)

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import soundfile
import librosa
import csv
import IPython.display as ipd
```

1 Load csv file

[2]:
```
metadata = pd.read_csv('/Users/user/Jupyter/DATASET/bird_songs_metadata.csv') ␣
↪# Read csv file
metadata.head(3)
```

[2]:
```
        id       genus   species subspecies            name  \
0  557838  Thryomanes  bewickii        NaN  Bewick's Wren
1  557838  Thryomanes  bewickii        NaN  Bewick's Wren
2  557838  Thryomanes  bewickii        NaN  Bewick's Wren

                 recordist        country  \
0  Whitney Neufeld-Kaiser  United States
1  Whitney Neufeld-Kaiser  United States
2  Whitney Neufeld-Kaiser  United States

                              location  latitude  longitude altitude  \
0  Arlington, Snohomish County, Washington   48.0708  -122.1006      100
1  Arlington, Snohomish County, Washington   48.0708  -122.1006      100
2  Arlington, Snohomish County, Washington   48.0708  -122.1006      100

                 sound_type                  source_url  \
0  adult, sex uncertain, song  //www.xeno-canto.org/557838
1  adult, sex uncertain, song  //www.xeno-canto.org/557838
2  adult, sex uncertain, song  //www.xeno-canto.org/557838
```

```
                                         license   time        date  \
0  //creativecommons.org/licenses/by-nc-sa/4.0/  11:51  2020-03-14
1  //creativecommons.org/licenses/by-nc-sa/4.0/  11:51  2020-03-14
2  //creativecommons.org/licenses/by-nc-sa/4.0/  11:51  2020-03-14

                                          remarks      filename
0  Recorded with Voice Record Pro on iPhone7, nor…  557838-0.wav
1  Recorded with Voice Record Pro on iPhone7, nor…  557838-1.wav
2  Recorded with Voice Record Pro on iPhone7, nor…  557838-4.wav
```

[3]:
```
metadata.shape
```

[3]:
```
(5422, 18)
```

[4]:
```
metadata.isnull().sum()
```

[4]:
```
id               0
genus            0
species          0
subspecies    3876
name             0
recordist        0
country          0
location         0
latitude        90
longitude       90
altitude        42
sound_type       0
source_url       0
license          0
time             0
date             0
remarks       1859
filename         0
dtype: int64
```

[5]:
```
# Drop columns source_url, remarks, license, time, date, recordist and␣
 ↪subspecies
metadata.drop(columns=['source_url','remarks','license','time','date',␣
 ↪'recordist', 'subspecies'], inplace=True)

# Rename column names
metadata = metadata.rename(columns={
                                    'id':'File_Id', 'genus':'Genus', 'species':
 ↪'Species', 'name':'English_Name',
                                    'country':'Country', 'location':'Location',␣
 ↪'latitude':'Latitude',
```

```
                                    'longitude':'Longitude', 'altitude':
↪'Altitude', 'location':'Location',
                                    'sound_type':'Type', 'filename':'Filename'
                                   })
metadata.head(3)
```

[5]:

```
   File_Id       Genus   Species   English_Name        Country  \
0   557838  Thryomanes  bewickii  Bewick's Wren  United States
1   557838  Thryomanes  bewickii  Bewick's Wren  United States
2   557838  Thryomanes  bewickii  Bewick's Wren  United States

                                Location  Latitude  Longitude Altitude  \
0  Arlington, Snohomish County, Washington   48.0708  -122.1006      100
1  Arlington, Snohomish County, Washington   48.0708  -122.1006      100
2  Arlington, Snohomish County, Washington   48.0708  -122.1006      100

                        Type      Filename
0  adult, sex uncertain, song  557838-0.wav
1  adult, sex uncertain, song  557838-1.wav
2  adult, sex uncertain, song  557838-4.wav
```

[6]:

```
counts = metadata['Species'].value_counts()
counts
```

[6]:

```
Species
melodia        1256
polyglottos    1182
cardinalis     1074
migratorius    1017
bewickii        893
Name: count, dtype: int64
```

[7]:

```
count = np.max(counts)
count
```

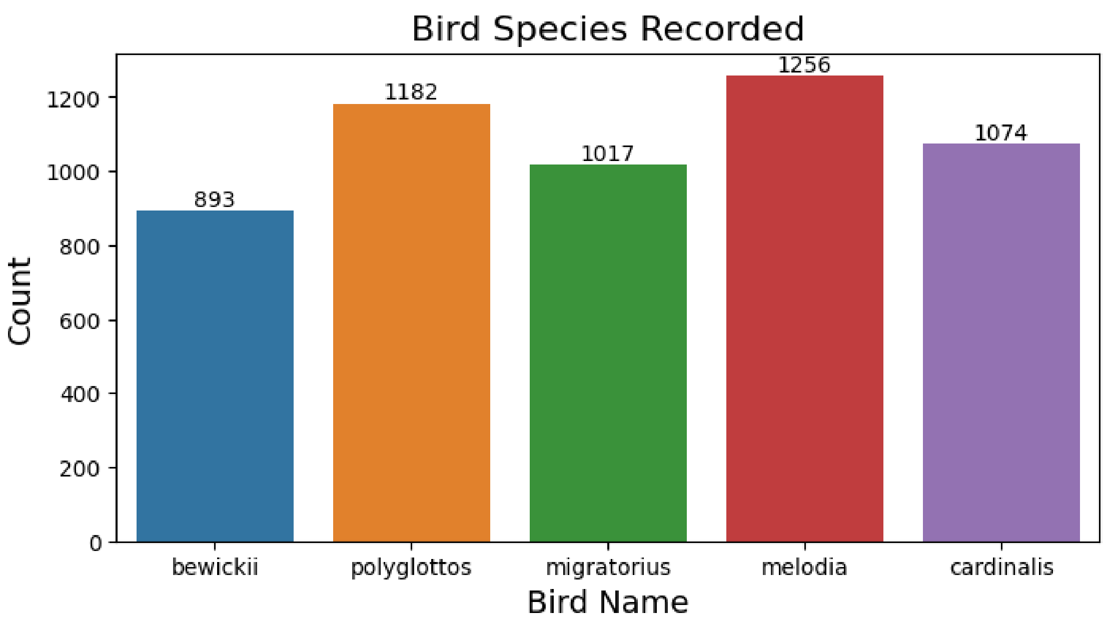
[7]: 1256

[8]:

```
def species_countplot(plot_data):
    # Create the countplot
    plt.figure(figsize=(8, 4))
    sns.countplot(x='Species', data=plot_data)

    # Rotate x-axis labels for better readability
    plt.xticks(rotation=0)

    # Get bar containers (rects) from the current axes
    bars = plt.gca().containers[0]
```

```
    # Get bar labels (counts)
    bar_labels = [x.get_height() for x in bars]
    int_bar_labels = [int(x) for x in bar_labels]

    # Set bar label positions
    plt.bar_label(bars, int_bar_labels)

    # Add title and show the plot
    plt.title('Bird Species Recorded', fontsize=16)
    plt.xlabel('Bird Name', fontsize=14)
    plt.ylabel('Count', fontsize=14)
    plt.show()
```

[9]:
```
# Plot species count
species_countplot(metadata)
```



1.1 Balancing Dataset

[10]:
```
# To get the species
species_to_keep = counts.index.tolist()
species_to_keep
```

[10]: ['melodia', 'polyglottos', 'cardinalis', 'migratorius', 'bewickii']

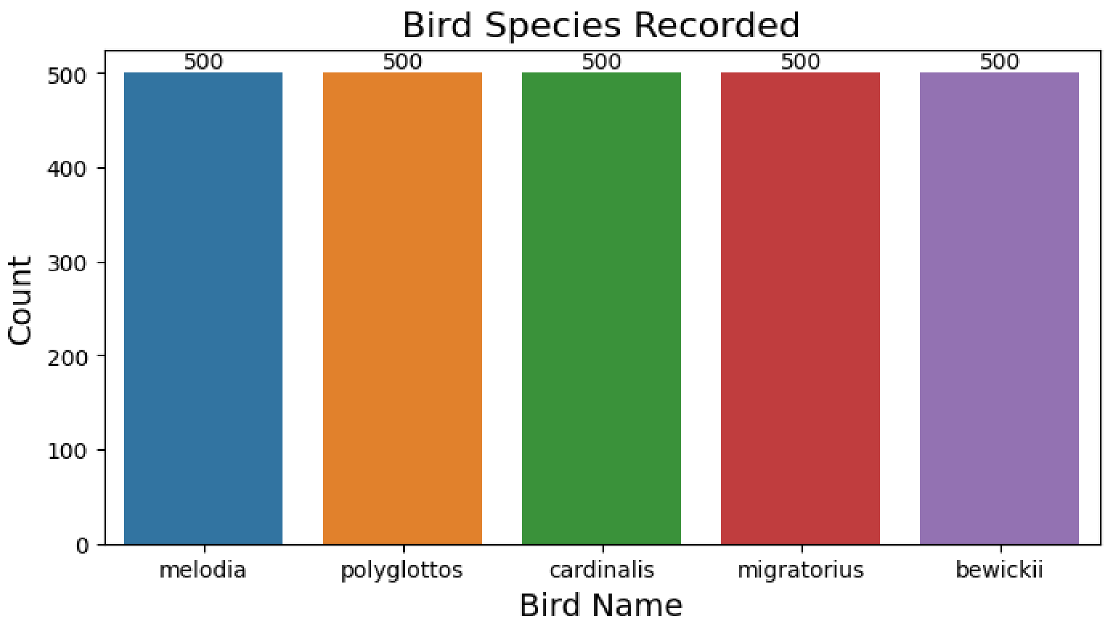
[11]:
```
# Create a new empty DataFrame
df = pd.DataFrame()

for species in species_to_keep:
  # Sample 500 recordings from the original DataFrame for each species
  sampled_df = metadata[metadata['Species'] == species].sample(500,␣
 ↪random_state=42, replace=False)

  # Add the sampled data to the new DataFrame
  df = pd.concat([df, sampled_df], ignore_index=True)

df.head(5)
```

[11]:
```
   File_Id      Genus  Species  English_Name        Country  \
0   363142  Melospiza  melodia  Song Sparrow  United States
1   490351  Melospiza  melodia  Song Sparrow  United States
2   551290  Melospiza  melodia  Song Sparrow  United States
3   549591  Melospiza  melodia  Song Sparrow  United States
4   105818  Melospiza  melodia  Song Sparrow  United States

                                            Location  Latitude  Longitude  \
0       Yampa River Botanic Park, Routt Co, Colorado   40.4725  -106.8311
1  Hawk Rise Sanctuary (near  Linden), Union Coun…   40.6052   -74.2495
2                    Rahway, Union County, New Jersey   40.6061   -74.2772
3  Thornton Creek Ravine, Seattle, King County, W…   47.7022  -122.3088
4  Battelle Darby Metro Park--Darby Dan Training …   39.9410   -83.2250

  Altitude                       Type       Filename
0     2100                       song   363142-0.wav
1        0                       song  490351-10.wav
2       10                       song  551290-10.wav
3       60  adult, sex uncertain, song   549591-9.wav
4      267                       Song  105818-11.wav
```

[12]:
```
df.shape
```

[12]:
```
(2500, 11)
```

[13]:
```
# Plot species count for modified dataframe
species_countplot(df)
```



Save the filtered dataset

[14]:
```
#df.to_csv('/Users/user/Jupyter/project/filtered_dataset.csv', index=False)  #␣
↪Don't save the index as a column
```

2 Audio features visualization

[15]:
```
data = pd.read_csv('/Users/user/Jupyter/project/filtered_dataset.csv')  # Read␣
↪csv file
data.head(3)
```

[15]:
```
   File_Id      Genus  Species  English_Name        Country  \
0   363142  Melospiza  melodia  Song Sparrow  United States
1   490351  Melospiza  melodia  Song Sparrow  United States
2   551290  Melospiza  melodia  Song Sparrow  United States

                                            Location  Latitude  Longitude  \
0        Yampa River Botanic Park, Routt Co, Colorado   40.4725  -106.8311
1  Hawk Rise Sanctuary (near  Linden), Union Coun…   40.6052   -74.2495
2                      Rahway, Union County, New Jersey   40.6061   -74.2772

  Altitude  Type       Filename
0     2100  song   363142-0.wav
1        0  song  490351-10.wav
2       10  song  551290-10.wav
```

2.0.1 Audio features visualization for a single audio

[16]:
```
example = df['Filename'].iloc[200]
example
```

[16]: '205806-15.wav'

[17]:
```
# Define the path
audio_path = f'/Users/user/Jupyter/DATASET/wavfiles/{example}'
audio_path
```

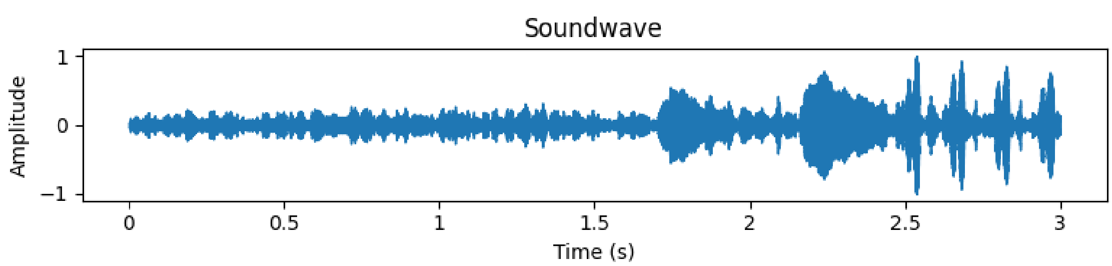
[17]: '/Users/user/Jupyter/DATASET/wavfiles/205806-15.wav'

[18]:
```
# Audio display
ipd.display(ipd.Audio(audio_path))
```

<IPython.lib.display.Audio object>

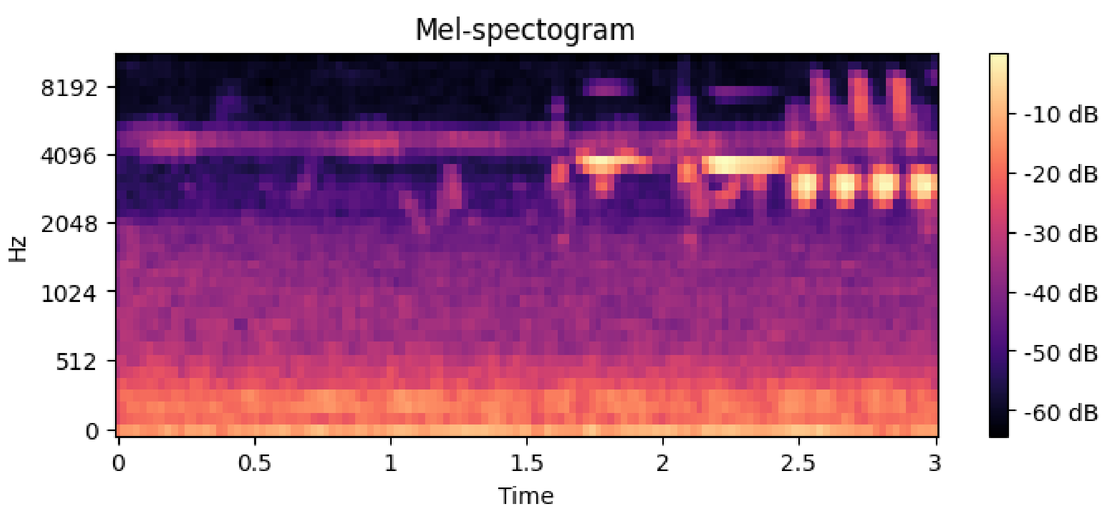
2.0.2 Soundwave plot

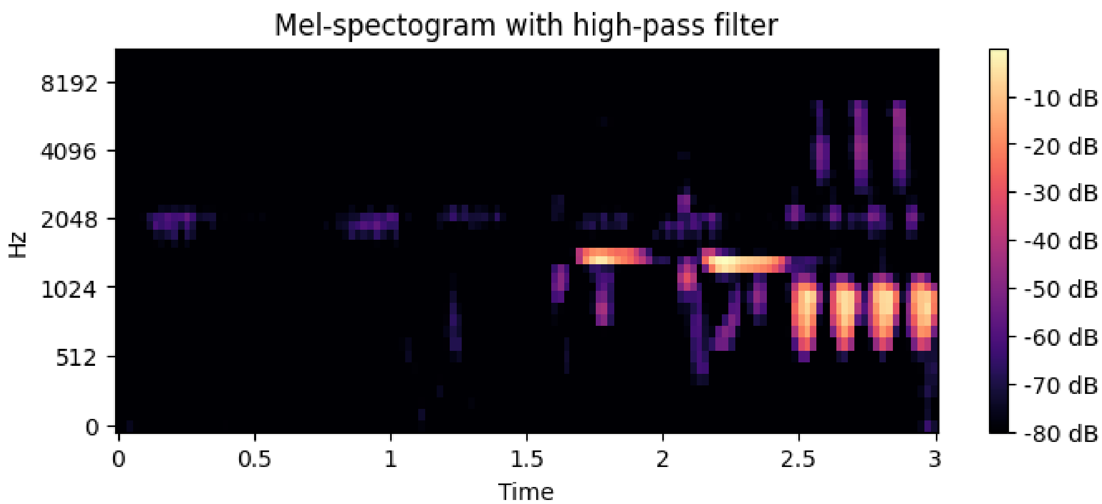
[19]:
```
# Load Audio
signal, sr = librosa.load(audio_path)

# Plot sound wave
plt.figure(figsize=(8, 2))
librosa.display.waveshow(signal, sr=sr)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Soundwave')
plt.tight_layout()
plt.show()
```



2.0.3 Mel-spectrogram visualization

[20]:
```
# Plot mel-spectrogram
s = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=40)

plt.figure(figsize=(8,3))
librosa.display.specshow(librosa.power_to_db(s,ref=np.max), x_axis='time',␣
 ↪y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title('Mel-spectogram')
plt.show()


# Plot mel-spectogram with high-pass filter
s = librosa.feature.melspectrogram(y=signal, sr=sr, n_mels=40, fmin=1800)

plt.figure(figsize=(8,3))
librosa.display.specshow(librosa.power_to_db(s**2,ref=np.max), x_axis='time',␣
 ↪y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title('Mel-spectogram with high-pass filter')
plt.show()
```

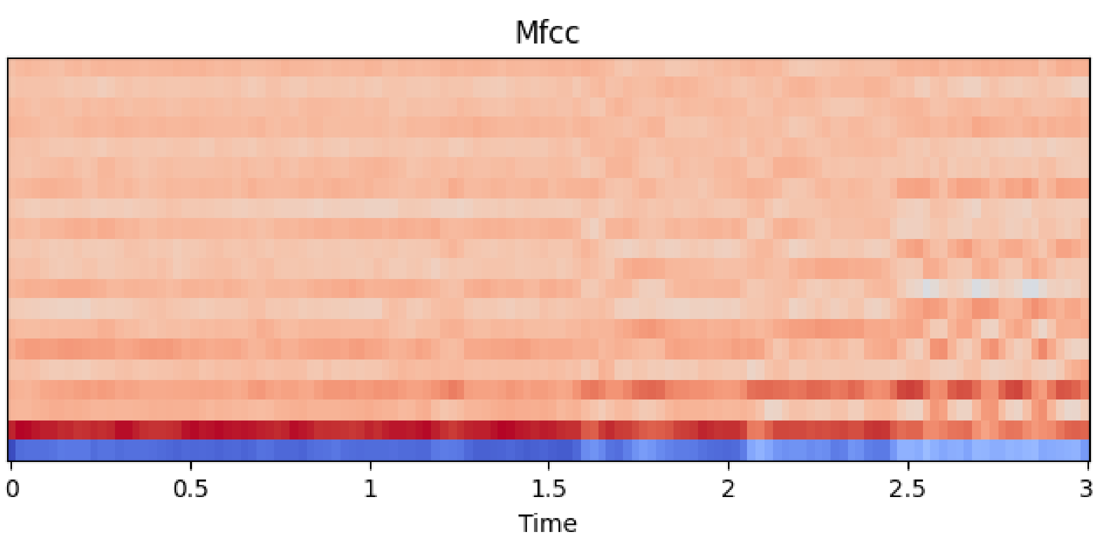




2.0.4 Mel-frequency cepstral coefficient (mfcc) visualization

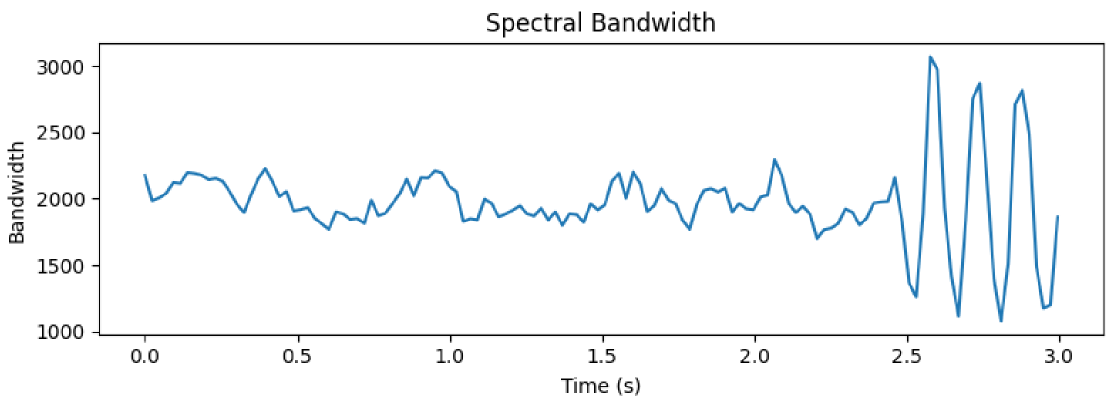
[21]:
```
# plot mfcc
plt.figure(figsize=(8,3))
mfcc = librosa.feature.mfcc(y=signal, sr=sr)
librosa.display.specshow(mfcc, sr=sr, x_axis='time')
plt.title('Mfcc')
plt.show
```

[21]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



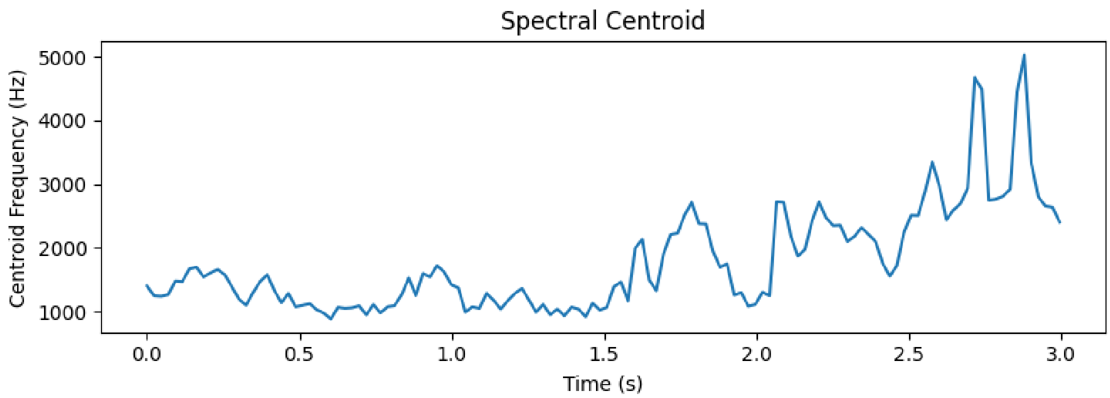
2.0.5 Spectral bandwidth visualization

[22]:
```
# Spectral bandwidth plot
plt.figure(figsize=(8,3))
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=signal, sr=sr)
time = librosa.frames_to_time(np.arange(len(spectral_bandwidth.T)), sr=sr)
plt.plot(time, spectral_bandwidth.T)
plt.title('Spectral Bandwidth')
plt.xlabel('Time (s)')
plt.ylabel('Bandwidth')
plt.tight_layout()
plt.show()
```



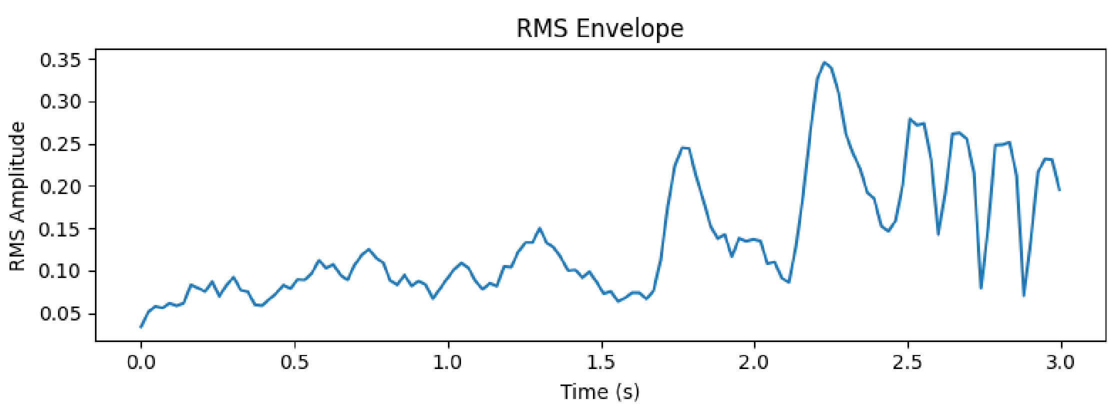
2.0.6 Spectral centroid visualization

[23]:
```
# Spectral centroid plot
plt.figure(figsize=(8,3))
spectral_centroid = librosa.feature.spectral_centroid(y=signal, sr=sr)
plt.plot(time, spectral_centroid[0])
plt.title('Spectral Centroid')
plt.xlabel('Time (s)')
plt.ylabel('Centroid Frequency (Hz)')
plt.tight_layout()
plt.show()
```



2.0.7 Root mean square (RMS) visualization

[24]:
```python
# RMS plot
plt.figure(figsize=(8,3))
rms = librosa.feature.rms(y=signal)
plt.plot(time, rms[0])
plt.title('RMS Envelope')
plt.xlabel('Time (s)')
plt.ylabel('RMS Amplitude')
plt.tight_layout()
plt.show()
```



Appendix - II

Main Code

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import librosa
import csv

from sklearn.metrics import accuracy_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import precision_score, recall_score,␣
 ↪precision_recall_curve
```

1 Load csv file

[2]:
```
metadata = pd.read_csv('/Users/user/Jupyter/my project/filtered_dataset.csv') ␣
 ↪# Read csv file
metadata.head(3)
```

[2]:
```
   File_Id      Genus  Species  English_Name        Country  \
0   363142  Melospiza  melodia  Song Sparrow  United States
1   490351  Melospiza  melodia  Song Sparrow  United States
2   551290  Melospiza  melodia  Song Sparrow  United States

                                             Location  Latitude  Longitude  \
0       Yampa River Botanic Park, Routt Co, Colorado   40.4725  -106.8311
1  Hawk Rise Sanctuary (near  Linden), Union Coun…   40.6052   -74.2495
2                    Rahway, Union County, New Jersey   40.6061   -74.2772

  Altitude  Type        Filename
0     2100  song   363142-0.wav
1        0  song  490351-10.wav
2       10  song  551290-10.wav
```

[3]:
```
metadata.shape
```

[3]: (2500, 11)

[4]:
```
metadata['Species'].value_counts()
```

[4]:
```
Species
melodia        500
polyglottos    500
cardinalis     500
migratorius    500
bewickii       500
Name: count, dtype: int64
```

[5]:
```
has_duplicates = metadata['Filename'].duplicated().any()
has_duplicates
```

[5]: False

[6]:
```
metadata.isnull().sum()
```

[6]:
```
File_Id         0
Genus           0
Species         0
English_Name    0
Country         0
Location        0
Latitude       38
Longitude      38
Altitude       14
Type            0
Filename        0
dtype: int64
```

2 Audio Feature Extraction

[7]:
```
# Create the dictionary
name_dict = dict(zip(metadata['Filename'], metadata['Species']))
name_dict
```

[7]:
```
{'363142-0.wav': 'melodia',
 '490351-10.wav': 'melodia',
 '551290-10.wav': 'melodia',
 '549591-9.wav': 'melodia',
 '105818-11.wav': 'melodia',
 '366598-2.wav': 'melodia',
 '111653-9.wav': 'melodia',
 '288000-8.wav': 'melodia',
```

```
'54018-6.wav': 'polyglottos',
'170052-9.wav': 'polyglottos',
'541426-1.wav': 'polyglottos',
'541496-5.wav': 'polyglottos',
'321789-1.wav': 'polyglottos',
…}
```

[8]:
```
# Define the directory containing audio files
audio_dir = '/Users/user/Jupyter/my project/wavfiles'

# os.listdir() to get filenames
filenames = os.listdir(audio_dir)

# Create audio paths by combining directory and filenames
audio_paths = [os.path.join(audio_dir, filename) for filename in filenames]
```

2.0.1 Feature Extraction

[9]:
```
mel, mfcc = [], []
spectral_centroid, rms, bandwidth, chromagram = [], [], [], []
filename, labels = [], []

for path in audio_paths:
    file = os.path.basename(path)  # Extract filename from UR

    if file in name_dict:
        # Load audio file
        y, sr = librosa.load(path)
        filename.append(file)
        labels.append(name_dict[file])

        # Extract MFCCs
        mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mels=20, n_mfcc=130)
        mfcc.append(mfccs.T)

        # Extract mel-spectrogram
        melspectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=20,
 ↪htk=True, fmin=1400)
        S = librosa.power_to_db(melspectrogram)
        mel.append(S.T)

        # spectral centroid
        spectral_centroid.append((librosa.feature.spectral_centroid(y=y,
 ↪sr=sr)).T)

        # bandwidth
        bandwidth.append((librosa.feature.spectral_bandwidth(y=y, sr=sr)).T)
```

```
        #Extract root mean square(rms)for analyzing the energy content of audio␣
↪signals
        rms.append((librosa.feature.rms(y=y)).T)
```

[10]:
```
# Convert audio_labels to dataframe
audio_labels = pd.DataFrame(labels, columns=['Species'])
audio_labels['Filename'] = filename
audio_labels.head(5)
```

[10]:

	Species	Filename
0	cardinalis	109035-6.wav
1	melodia	366597-4.wav
2	cardinalis	317266-5.wav
3	bewickii	351076-1.wav
4	migratorius	537326-4.wav

[11]:
```
print('Shape of mfcc :', np.array(mfcc).shape)
print('Shape of mel-spectrogram :', np.array(mel).shape)
print('Shape of spectral_centroid :', np.array(spectral_centroid).shape)
print('Shape of rms :', np.array(rms).shape)
print('Shape of bandwidth :', np.array(bandwidth).shape)
print('Shape of chromagram :', np.array(chromagram).shape)
```

```
Shape of mfcc : (2500, 130, 20)
Shape of mel-spectrogram : (2500, 130, 20)
Shape of spectral_centroid : (2500, 130, 1)
Shape of rms : (2500, 130, 1)
Shape of bandwidth : (2500, 130, 1)
Shape of chromagram : (0,)
```

2.1 Scale Data

[12]:
```
from sklearn.preprocessing import StandardScaler
```

[13]:
```
# Create a StandardScaler object
scaler = StandardScaler()

def scale(data):

    if len(data.shape) == 2:

        scaler.fit(data)  # Fit the scaler
        data = scaler.transform(data)  # Scale data using the fitted scaler
        return data, scaler.mean_, scaler.scale_

    elif len(data.shape) == 3:
```

```
        batch, n_row, n_col = data.shape
        data_reshape = data.reshape(-1, n_row * n_col)  # flatten the np arrays␣
 ↪to 1D

        scaler.fit(data_reshape)      # Fit the scaler
        data_scaled = scaler.transform(data_reshape)  # Scale data using the␣
 ↪fitted scaler

        data = data_scaled.reshape(-1, n_row, n_col)  # reshape the data to the␣
 ↪original shape
        return data, scaler.mean_, scaler.scale_

    else:
        raise ValueError("Input array must be 2D or 3D.")



def processed(data, mean=False):

    data = np.array(data)

    if mean:
        data = np.mean(data, axis=1)
        return scale(data)
    else:
        return scale(data)
```

2.2 Encode Data

[14]:
```
from sklearn.preprocessing import LabelEncoder
```

[15]:
```
#Label Encoder
le = LabelEncoder()

audio_labels['Encoded'] = le.fit_transform(labels)
audio_labels.head(5)
```

[15]:

	Species	Filename	Encoded
0	cardinalis	109035-6.wav	1
1	melodia	366597-4.wav	2
2	cardinalis	317266-5.wav	1
3	bewickii	351076-1.wav	0
4	migratorius	537326-4.wav	3

[16]:
```
# Create a dictionary with encoded labels as key and species as values
class_label = dict(zip(audio_labels['Encoded'],audio_labels['Species']))
class_label
```

[16]:
```
{1: 'cardinalis',
 2: 'melodia',
 0: 'bewickii',
 3: 'migratorius',
 4: 'polyglottos'}
```

[17]:
```
# Sort the dictionary by key in ascending order
sorted_class_labels = dict(sorted(class_label.items()))

# Extract a list of values (bird species)
class_labels = list(sorted_class_labels.values())

class_labels
```

[17]:
```
['bewickii', 'cardinalis', 'melodia', 'migratorius', 'polyglottos']
```

2.3 Split Data

mfcc, bandwidth, spectral centroid, rms are calculated

[18]:
```
from sklearn.model_selection import train_test_split
```

[19]:
```
# Combine all features
all_features = np.concatenate((np.array(mfcc), np.array(bandwidth), np.
 ↪array(spectral_centroid), np.array(rms)),axis=2)
```

[20]:
```
# calculate scalar mean and stantard deviation
data , mean, std = processed(all_features, mean=True)
```

[21]:
```
# Split the dataset ito features and target variable
X = data
Y = audio_labels['Encoded']

# Split the dataset ito train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=42, stratify=Y)
```

2.4 Cross validation

[22]:
```
from sklearn.model_selection import StratifiedKFold, cross_val_score
```

[23]:
```
# Stratified KFold ensures classes are proportionally distributed in each fold
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

3 KNN classifier

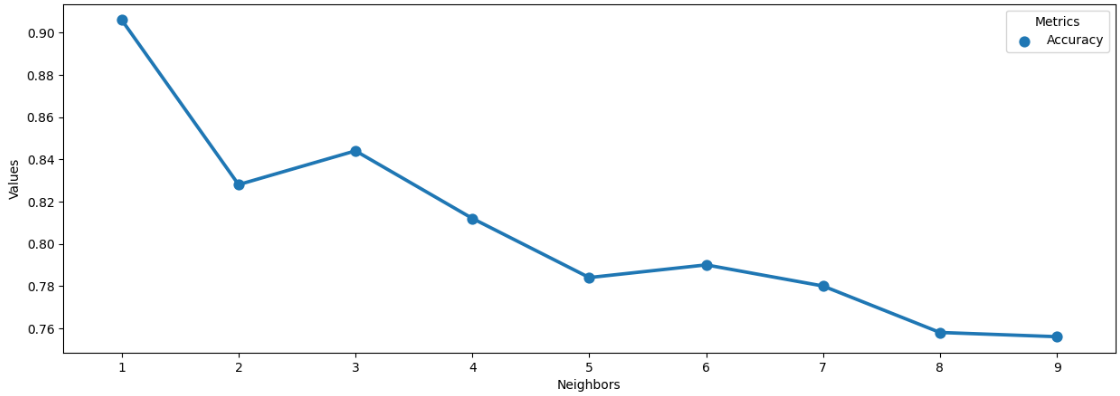
[24]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[25]:
```
def KNN_classifier(n):

    # Create the KNN model with n_neighbors (number of neighbors) set to 3
    knn = KNeighborsClassifier(n_neighbors=n)

    # Train the model on the training data
    knn.fit(X_train, Y_train)

    # Make predictions on the testing data
    pred_knn = knn.predict(X_test)

    return pred_knn
```

[26]:
```
val_accuracy = []
neighbors = []

for neighbor in range(1,10):

    pred_knn = KNN_classifier(neighbor)  # Make predictions

    val_acc = accuracy_score(Y_test,pred_knn)  # Evaluate model accuracy
    val_accuracy.append(val_acc)
    neighbors.append(neighbor)
```

[27]:
```
# Tuning n_neighbors for the best accuracy
Tuning_neighbors = {'Accuracy':val_accuracy, 'Neighbors':neighbors}
Tuning_neighbors_df = pd.DataFrame.from_dict(Tuning_neighbors)

plot_df = Tuning_neighbors_df.melt('Neighbors', var_name='Metrics',␣
 ↪value_name='Values')
fig, ax = plt.subplots(figsize=(15,5))
sns.pointplot(x='Neighbors', y='Values', hue='Metrics', data=plot_df, ax=ax)
```

[27]:
```
<Axes: xlabel='Neighbors', ylabel='Values'>
```



3.0.1 Cross validation

[28]:
```python
knn = KNeighborsClassifier(n_neighbors=3)
```

[29]:
```python
knn_cv_scores = cross_val_score(knn, X_train, Y_train, cv=cv)
print('cv scores :', knn_cv_scores)
print('Average accuracy :', np.mean(knn_cv_scores))
```

```
cv scores : [0.835  0.835  0.815  0.74   0.8225]
Average accuracy : 0.8094999999999999
```

3.0.2 Evaluation

[30]:
```python
knn_pred = KNN_classifier(3)

# Evaluate model accuracy
knn_accuracy = accuracy_score(Y_test, knn_pred)
print(f'Accuracy : {knn_accuracy:.4f}')

# Calculate f1 score precision and recall
knn_f1 = f1_score(Y_test, knn_pred, average='macro')
knn_precision = precision_score(Y_test, knn_pred, average='weighted')
knn_recall = recall_score(Y_test, knn_pred, average='weighted')

# Print the results
print(f'F1-score : {knn_f1:.4f}')
print(f'Precision : {knn_precision:.4f}')
print(f'Recall : {knn_recall}')
```

```
Accuracy : 0.8440
F1-score : 0.8428
Precision : 0.8564
Recall : 0.844
```

3.0.3 Classification Report

[31]:
```
print(classification_report(Y_test, knn_pred, target_names=class_labels))
```

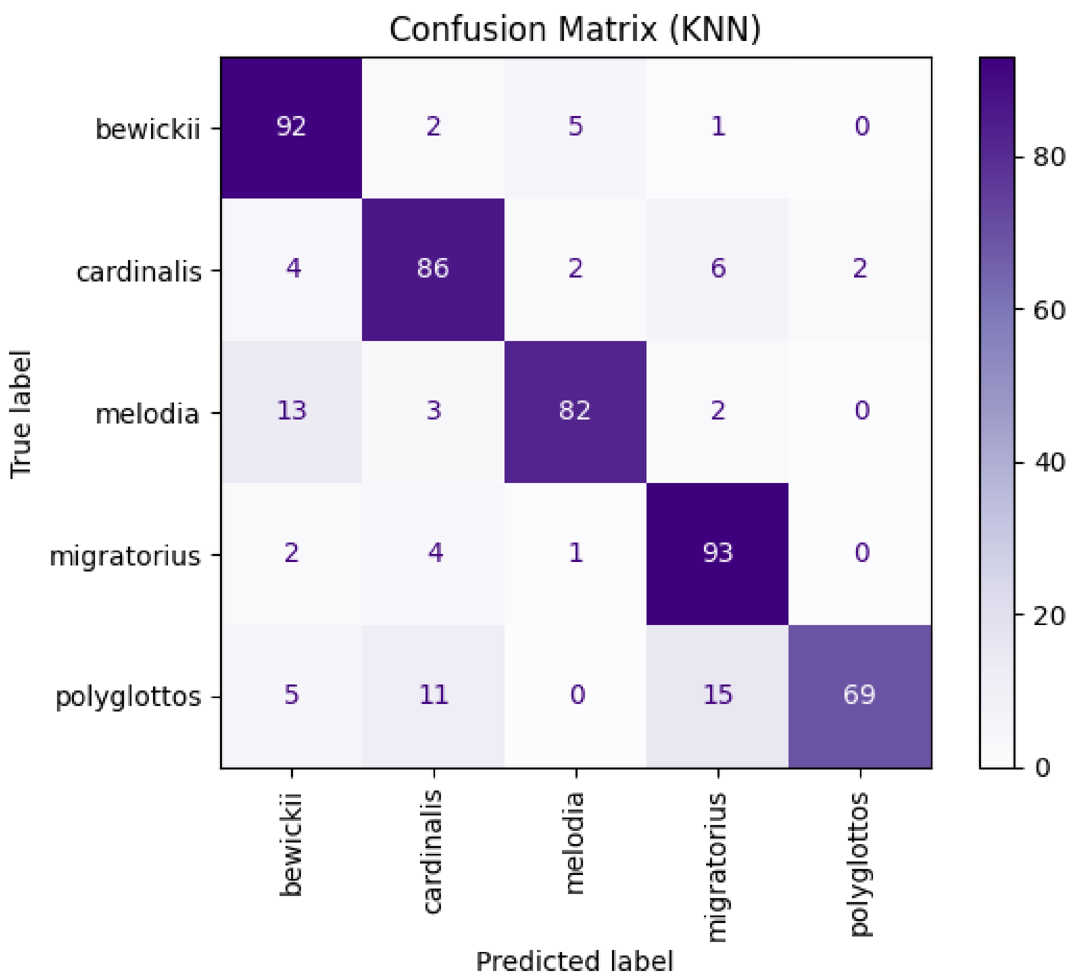
```
              precision    recall  f1-score   support

    bewickii       0.79      0.92      0.85       100
  cardinalis       0.81      0.86      0.83       100
     melodia       0.91      0.82      0.86       100
 migratorius       0.79      0.93      0.86       100
 polyglottos       0.97      0.69      0.81       100

    accuracy                           0.84       500
   macro avg       0.86      0.84      0.84       500
weighted avg       0.86      0.84      0.84       500
```

3.0.4 confusion matrix

[32]:
```
def confusion_matrix_plot(cm, model):
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,␣
 ↪display_labels=class_labels)
    disp.plot(cmap=plt.cm.Purples)
    plt.xticks(rotation=90)
    plt.title(f'Confusion Matrix ({model})')
    plt.show()
```

[33]:
```
# Generate confusion matrix
knn_confusion_matrix = confusion_matrix(Y_test, knn_pred)
```

[34]:
```
confusion_matrix_plot(knn_confusion_matrix, 'KNN')
```

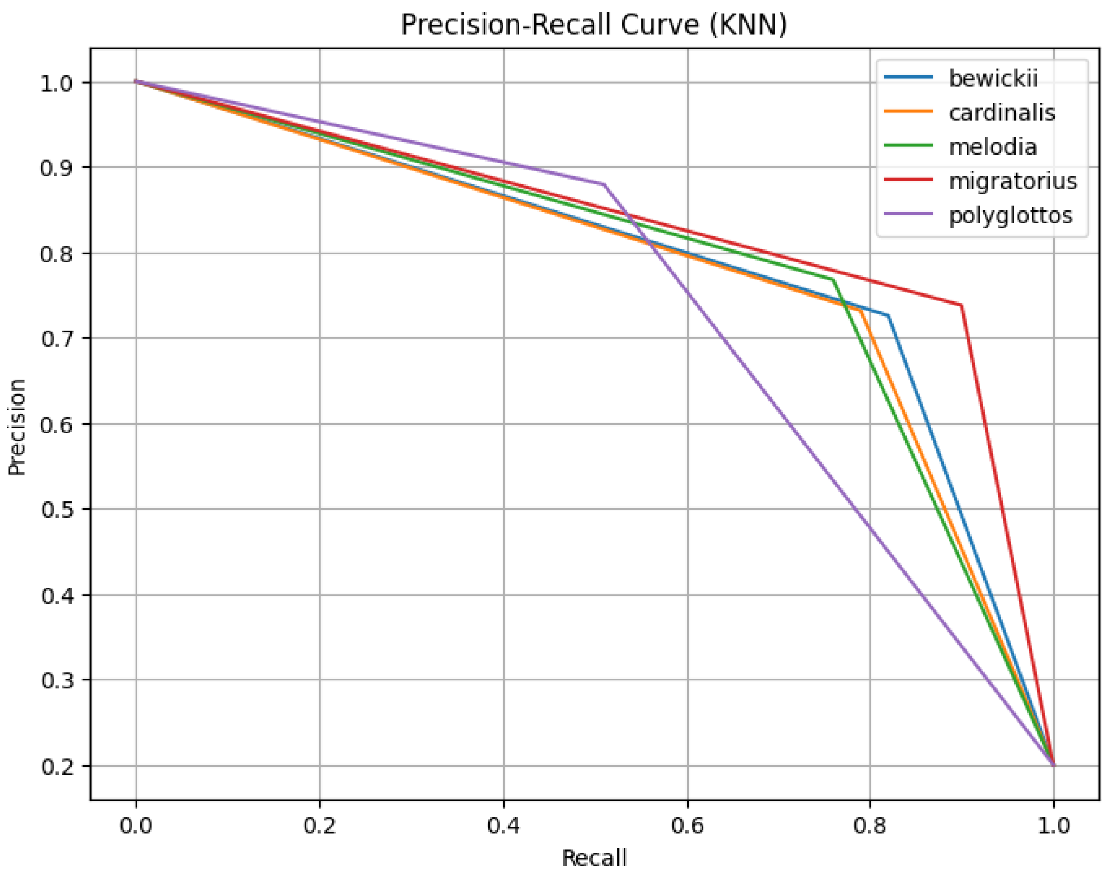


3.0.5 Precision Recall Curve

[35]:
```
def Precision_Recall_Curve(y_pred, model):

    prc_data = {}

    for i, class_label in enumerate(class_labels):
        precision, recall, thresholds = precision_recall_curve(Y_test == i,
 ↪y_pred == i)
        prc_data[class_label] = (precision, recall)

    # Plot the precision-recall curves
    plt.figure(figsize=(8, 6))
    for class_label, (precision, recall) in prc_data.items():
        plt.plot(recall, precision, label=class_label)  # Use class_label
 ↪directly
```

```
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve ({model})')
    plt.legend()
    plt.grid(True)
    plt.show()
```

[36]:
```
Precision_Recall_Curve(pred_knn, 'KNN')
```



4 Support Vector Algorithm

[37]:
```
from sklearn.svm import SVC
```

[38]:
```
svc = SVC(C=1, gamma=0.1)
```

4.0.1 Cross validation

[39]:
```
svc_cv_scores = cross_val_score(svc, X_train, Y_train, cv=cv)
print('cv scores :', svc_cv_scores)
print('Average accuracy :', np.mean(svc_cv_scores))
```

```
cv scores : [0.89   0.8725 0.87   0.8175 0.83   ]
Average accuracy : 0.8560000000000001
```

4.0.2 Evaluation

[40]:
```
svc.fit(X_train,Y_train)

svc_pred = svc.predict(X_test)

svc_accuracy = accuracy_score(Y_test,svc_pred)
print('Test Accuracy of Support Vector Algorithm: ',svc_accuracy)

# Calculate f1 score precision and recall
svc_f1 = f1_score(Y_test, svc_pred, average='macro')
svc_precision = precision_score(Y_test, svc_pred, average='weighted')
svc_recall = recall_score(Y_test, svc_pred, average='weighted')

# Print the results
print(f'F1-score : {svc_f1:.4f}')
print(f'Precision : {svc_precision:.4f}')
print(f'Recall : {svc_recall}')
```

```
Test Accuracy of Support Vector Algorithm:  0.868
F1-score : 0.8677
Precision : 0.8692
Recall : 0.868
```

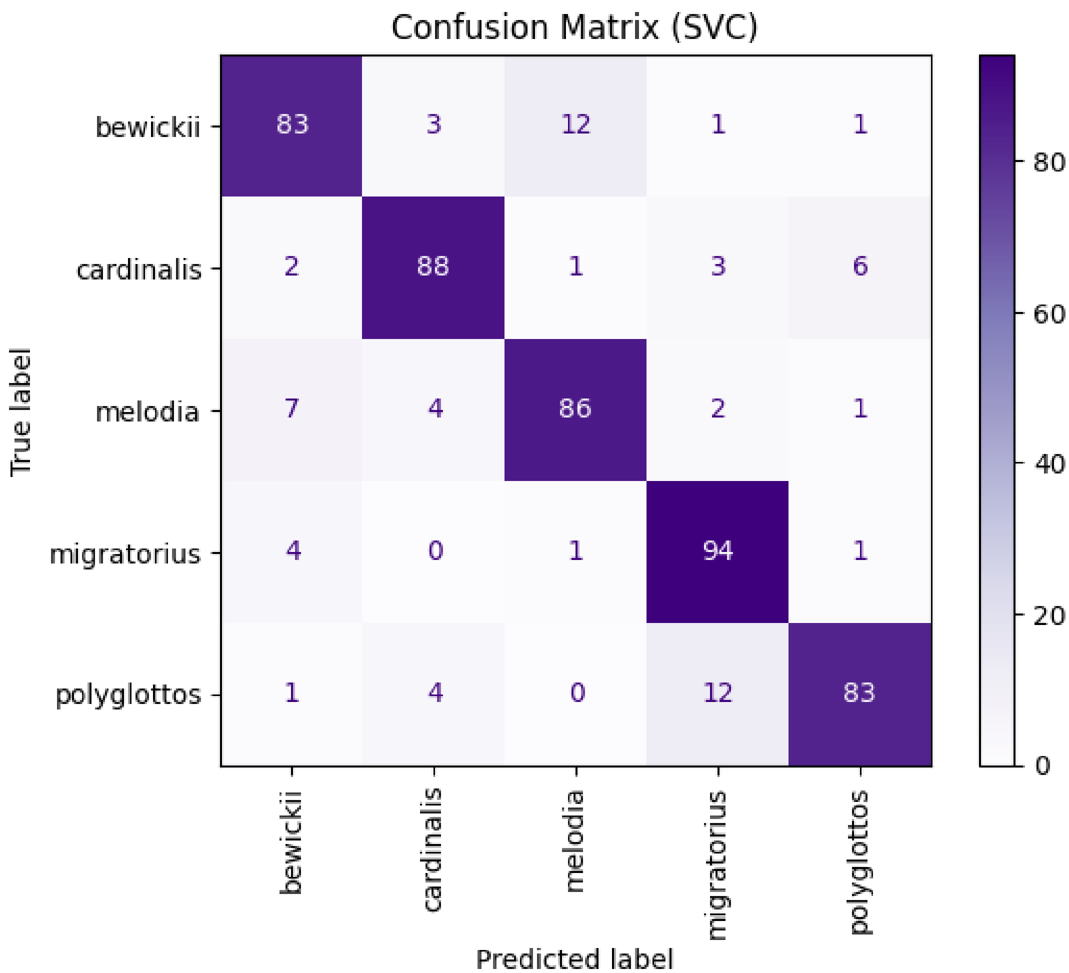
4.0.3 Classification Report

[41]:
```
print(classification_report(Y_test, svc_pred, target_names=class_labels))
```

```
              precision    recall  f1-score   support

    bewickii       0.86      0.83      0.84       100
  cardinalis       0.89      0.88      0.88       100
     melodia       0.86      0.86      0.86       100
 migratorius       0.84      0.94      0.89       100
 polyglottos       0.90      0.83      0.86       100

    accuracy                           0.87       500
   macro avg       0.87      0.87      0.87       500
weighted avg       0.87      0.87      0.87       500
```

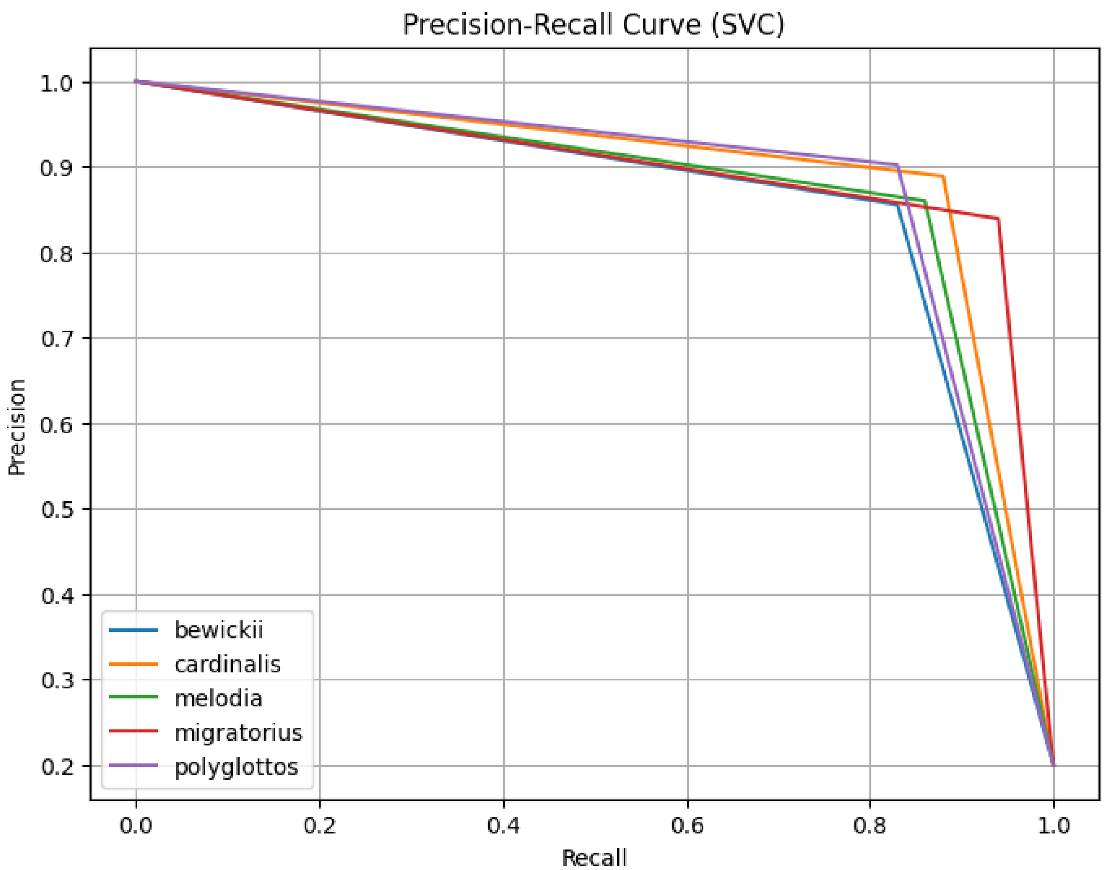
4.0.4 confusion matrix

[42]:
```
# Generate confusion matrix
svc_confusion_matrix = confusion_matrix(Y_test, svc_pred)
confusion_matrix_plot(svc_confusion_matrix, 'SVC')
```



4.0.5 Precision Recall Curve

[43]:
```
# Precision-recall curve for each bird species
Precision_Recall_Curve(svc_pred, 'SVC')
```



5 Random forest classifier

[44]:
```python
from sklearn.ensemble import RandomForestClassifier
```

[45]:
```python
# Create the Random Forest Classifier model
rfc_model = RandomForestClassifier(n_estimators=300)  # n_estimators (number of␣
↪trees)
```

5.0.1 Cross validation

[46]:
```python
rfc_cv_scores = cross_val_score(rfc_model, X_train, Y_train, cv=cv)
print('cv scores :', rfc_cv_scores)
print('Average accuracy :', np.mean(rfc_cv_scores))
```

```
cv scores : [0.8425 0.87   0.85   0.795  0.7975]
Average accuracy : 0.8310000000000001
```

5.0.2 Evaluation

[47]:
```
# Train the model
rfc_model.fit(X_train, Y_train)

# Make predictions on the testing set
rfc_pred = rfc_model.predict(X_test)

# Evaluate the model's accuracy
rfc_accuracy = accuracy_score(Y_test, rfc_pred)
print("Accuracy:", rfc_accuracy)

# Calculate f1 score precision and recall
rfc_f1 = f1_score(Y_test, rfc_pred, average='macro')
rfc_precision = precision_score(Y_test, rfc_pred, average='weighted')
rfc_recall = recall_score(Y_test, rfc_pred, average='weighted')

# Print the results
print(f'F1-score : {rfc_f1:.4f}')
print(f'Precision : {rfc_precision:.4f}')
print(f'Recall : {rfc_recall}')
```

```
Accuracy: 0.852
F1-score : 0.8517
Precision : 0.8528
Recall : 0.852
```

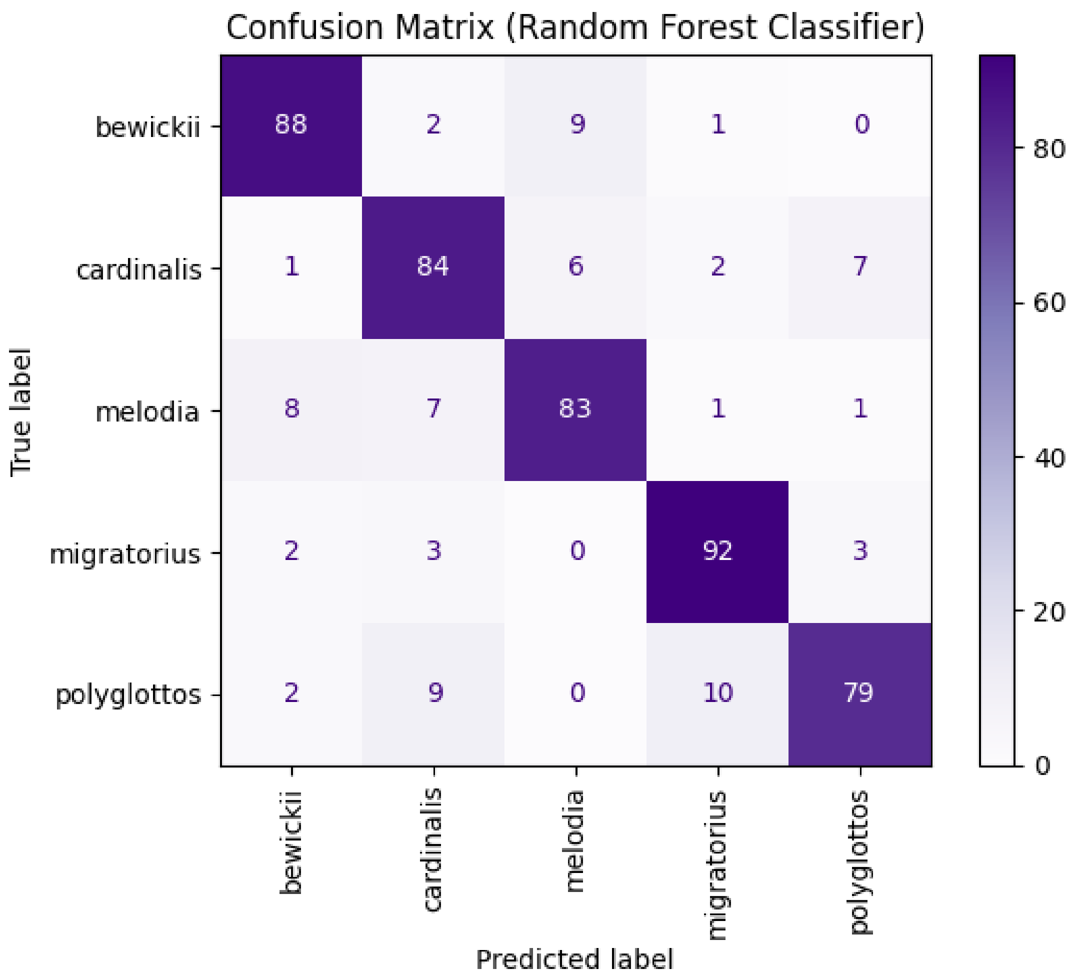
5.0.3 Classification Report

[48]:
```
print(classification_report(Y_test, rfc_pred, target_names=class_labels))
```

```
              precision    recall  f1-score   support

    bewickii       0.87      0.88      0.88       100
  cardinalis       0.80      0.84      0.82       100
     melodia       0.85      0.83      0.84       100
 migratorius       0.87      0.92      0.89       100
 polyglottos       0.88      0.79      0.83       100

    accuracy                           0.85       500
   macro avg       0.85      0.85      0.85       500
weighted avg       0.85      0.85      0.85       500
```

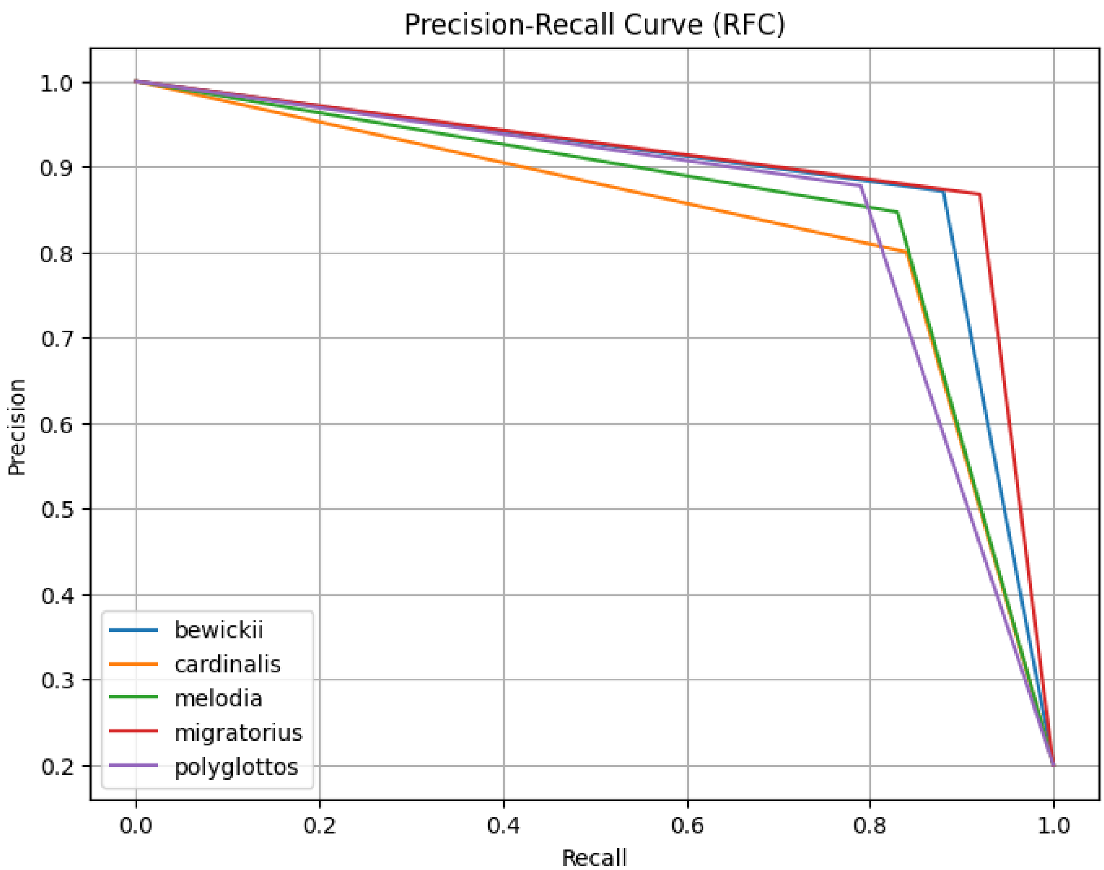
5.0.4 confusion matrix

[49]:
```
# Generate confusion matrix
rfc_confusion_matrix = confusion_matrix(Y_test, rfc_pred)
confusion_matrix_plot(rfc_confusion_matrix, 'Random Forest Classifier')
```



5.0.5 Precision Recall Curve

[50]:
```
# Precision-recall curve for each bird species
Precision_Recall_Curve(rfc_pred, 'RFC')
```



6 1D CNN

[51]:
```python
import tensorflow as tf
```

```
2024-06-19 22:53:57.042164: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

6.0.1 Build 1D CNN Model

[52]:
```python
def build_1d_model(audio_features):

    num_classes = 5

    tf.keras.backend.clear_session()

    # Audio features input layer
```

```python
    inputs = tf.keras.layers.Input(shape=(audio_features.
↪shape[1],audio_features.shape[2]), name='Audio_Features')

    # First convolutional block
    x = tf.keras.layers.Conv1D(filters=35, kernel_size=5, name='conv_1',␣
↪activation='relu', padding='same',
                                kernel_regularizer=tf.keras.regularizers.l2(0.
↪02))(inputs)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.MaxPool1D(pool_size=2, name='pool_1')(x)
    x = tf.keras.layers.Dropout(rate=0.2)(x)

    # Second convolutional block
    x = tf.keras.layers.Conv1D(filters=70, kernel_size=5, name='conv_2',␣
↪activation='relu', padding='same',
                                    kernel_regularizer=tf.keras.regularizers.
↪l2(0.02))(x)
    x = tf.keras.layers.MaxPool1D(pool_size=3, name='pool_2')(x)
    x = tf.keras.layers.Dropout(rate=0.3)(x)

    # Flatten the output for feeding into the dense layers
    x = tf.keras.layers.Flatten()(x)

    # Dense layers with a dropout layer
    x = tf.keras.layers.Dense(units=500, name='dense_1', activation='relu')(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Dense(units=100, name='dense_2', activation='relu')(x)

    # Last dense layer
    outputs = tf.keras.layers.Dense(units=num_classes, name='dense_3',␣
↪activation='softmax')(x)

    # Build model and print summary
    model = tf.keras.Model(inputs=[inputs],
                           outputs=outputs,
                           name='Birds')

    print(model.summary())

    # Compile model
    model.compile(optimizer='Adam', loss=tf.keras.losses.
↪SparseCategoricalCrossentropy(), metrics=['accuracy'],
              )

    return model
```

6.0.2 Scale & Split data

with mfcc, bandwidth, spectral_centroid and rms

[53]:
```
# Combine all features
features = np.concatenate((mfcc, bandwidth, spectral_centroid, rms), axis=2)
```

[54]:
```
# calculate scalar mean and stantard deviation for cnn
data, mean_cnn, std_cnn = processed(features, mean=False)
```

[55]:
```
# Split the dataset ito features and target variable
X = data
Y = audio_labels['Encoded']
```

[56]:
```
# Split the dataset to train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=42, stratify=Y)

# Split the train dataset to train and validation(val)
x_train, x_val, y_train, y_val = train_test_split(X_train, Y_train, test_size=0.
↪2, random_state=42, stratify=Y_train)
```

[57]:
```
print("Training set size:", len(x_train))
print("Test set size:", len(X_test))
print("val set size:", len(x_val))

print("Training set class distribution:", np.unique(y_train,␣
↪return_counts=True))
print("val set class distribution:", np.unique(y_val, return_counts=True))
print("Test set class distribution:", np.unique(Y_test, return_counts=True))
```

```
Training set size: 1600
Test set size: 500
val set size: 400
Training set class distribution: (array([0, 1, 2, 3, 4]), array([320, 320, 320,
320, 320]))
val set class distribution: (array([0, 1, 2, 3, 4]), array([80, 80, 80, 80,
80]))
Test set class distribution: (array([0, 1, 2, 3, 4]), array([100, 100, 100, 100,
100]))
```

6.0.3 Train the model

[58]:
```
cnn_model = build_1d_model(x_train)
```

```
Model: "Birds"
```

Layer (type)	Output Shape	Param #
Audio_Features (InputLayer)	(None, 130, 23)	0
conv_1 (Conv1D)	(None, 130, 35)	4,060
batch_normalization (BatchNormalization)	(None, 130, 35)	140
pool_1 (MaxPooling1D)	(None, 65, 35)	0
dropout (Dropout)	(None, 65, 35)	0
conv_2 (Conv1D)	(None, 65, 70)	12,320
pool_2 (MaxPooling1D)	(None, 21, 70)	0
dropout_1 (Dropout)	(None, 21, 70)	0
flatten (Flatten)	(None, 1470)	0
dense_1 (Dense)	(None, 500)	735,500
dropout_2 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 100)	50,100
dense_3 (Dense)	(None, 5)	505

Total params: 802,625 (3.06 MB)

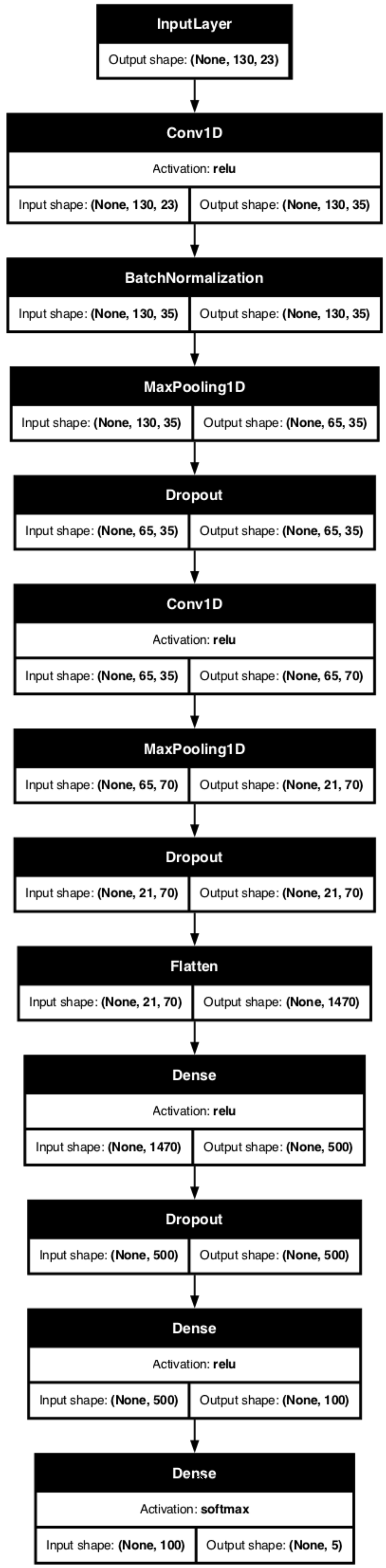
Trainable params: 802,555 (3.06 MB)

Non-trainable params: 70 (280.00 B)

None

[59]:
```
tf.keras.utils.plot_model(cnn_model, dpi=70, show_layer_activations= True,
                          show_shapes=True)
```

[59]:


InputLayer
Output shape: (None, 130, 23)
Conv1D
Activation: relu
Input shape: (None, 130, 23)
Output shape: (None, 130, 35)
BatchNormalization
Input shape: (None, 130, 35)
Output shape: (None, 130, 35)
MaxPooling1D
Input shape: (None, 130, 35)
Output shape: (None, 65, 35)
Dropout
Input shape: (None, 65, 35)
Output shape: (None, 65, 35)
Conv1D
Activation: relu
Input shape: (None, 65, 35)
Output shape: (None, 65, 70)
MaxPooling1D
Input shape: (None, 65, 70)
Output shape: (None, 21, 70)
Dropout
Input shape: (None, 21, 70)
Output shape: (None, 21, 70)
Flatten
Input shape: (None, 21, 70)
Output shape: (None, 1470)
Dense
Activation: relu
Input shape: (None, 1470)
Output shape: (None, 500)
Dropout
Input shape: (None, 500)
Output shape: (None, 500)
Dense
Activation: relu
Input shape: (None, 500)
Output shape: (None, 100)
Dense
Activation: softmax
Input shape: (None, 100)
Output shape: (None, 5)

[60]:
```python
history = cnn_model.fit(x_train, y_train, epochs=30, validation_data=(x_val,
    y_val))
```

```
Epoch 1/30
50/50               5s 39ms/step -
accuracy: 0.2773 - loss: 3.5234 - val_accuracy: 0.5050 - val_loss: 2.4936
Epoch 2/30
50/50               2s 35ms/step -
accuracy: 0.5183 - loss: 2.3138 - val_accuracy: 0.6200 - val_loss: 2.0482
Epoch 3/30
50/50               2s 32ms/step -
accuracy: 0.5768 - loss: 1.9777 - val_accuracy: 0.6075 - val_loss: 1.7696
Epoch 4/30
50/50               2s 41ms/step -
accuracy: 0.6281 - loss: 1.7030 - val_accuracy: 0.6700 - val_loss: 1.5896
Epoch 5/30
50/50               2s 39ms/step -
accuracy: 0.6790 - loss: 1.4822 - val_accuracy: 0.7325 - val_loss: 1.3844
Epoch 6/30
50/50               2s 33ms/step -
accuracy: 0.7147 - loss: 1.3529 - val_accuracy: 0.6875 - val_loss: 1.2646
Epoch 7/30
50/50               2s 34ms/step -
accuracy: 0.7600 - loss: 1.1482 - val_accuracy: 0.7750 - val_loss: 1.1340
Epoch 8/30
50/50               2s 34ms/step -
accuracy: 0.7717 - loss: 1.0520 - val_accuracy: 0.7525 - val_loss: 1.0508
Epoch 9/30
50/50               2s 34ms/step -
accuracy: 0.7788 - loss: 0.9437 - val_accuracy: 0.7900 - val_loss: 0.9657
Epoch 10/30
50/50               3s 36ms/step -
accuracy: 0.7979 - loss: 0.9079 - val_accuracy: 0.7625 - val_loss: 0.9306
Epoch 11/30
50/50               2s 33ms/step -
accuracy: 0.8045 - loss: 0.8167 - val_accuracy: 0.8125 - val_loss: 0.8200
Epoch 12/30
50/50               2s 36ms/step -
accuracy: 0.8495 - loss: 0.7173 - val_accuracy: 0.7875 - val_loss: 0.8471
Epoch 13/30
50/50               2s 37ms/step -
accuracy: 0.8504 - loss: 0.6942 - val_accuracy: 0.7925 - val_loss: 0.7871
Epoch 14/30
50/50               2s 36ms/step -
accuracy: 0.8633 - loss: 0.6348 - val_accuracy: 0.8200 - val_loss: 0.7591
```

```
Epoch 15/30
50/50            2s 35ms/step -
accuracy: 0.8468 - loss: 0.6720 - val_accuracy: 0.8400 - val_loss: 0.7071
Epoch 16/30
50/50            2s 37ms/step -
accuracy: 0.8805 - loss: 0.5706 - val_accuracy: 0.8150 - val_loss: 0.7276
Epoch 17/30
50/50            2s 34ms/step -
accuracy: 0.8755 - loss: 0.5812 - val_accuracy: 0.8225 - val_loss: 0.7413
Epoch 18/30
50/50            3s 34ms/step -
accuracy: 0.8899 - loss: 0.5245 - val_accuracy: 0.8500 - val_loss: 0.6552
Epoch 19/30
50/50            2s 36ms/step -
accuracy: 0.8914 - loss: 0.5156 - val_accuracy: 0.8300 - val_loss: 0.7179
Epoch 20/30
50/50            2s 39ms/step -
accuracy: 0.9056 - loss: 0.4584 - val_accuracy: 0.8450 - val_loss: 0.6640
Epoch 21/30
50/50            3s 42ms/step -
accuracy: 0.9201 - loss: 0.4448 - val_accuracy: 0.8300 - val_loss: 0.6812
Epoch 22/30
50/50            2s 35ms/step -
accuracy: 0.8985 - loss: 0.4923 - val_accuracy: 0.8500 - val_loss: 0.5971
Epoch 23/30
50/50            2s 37ms/step -
accuracy: 0.9364 - loss: 0.4087 - val_accuracy: 0.8300 - val_loss: 0.7061
Epoch 24/30
50/50            2s 35ms/step -
accuracy: 0.9156 - loss: 0.4234 - val_accuracy: 0.8550 - val_loss: 0.6537
Epoch 25/30
50/50            2s 38ms/step -
accuracy: 0.9364 - loss: 0.3810 - val_accuracy: 0.8375 - val_loss: 0.6573
Epoch 26/30
50/50            2s 35ms/step -
accuracy: 0.9176 - loss: 0.4114 - val_accuracy: 0.8625 - val_loss: 0.6172
Epoch 27/30
50/50            2s 36ms/step -
accuracy: 0.9417 - loss: 0.3554 - val_accuracy: 0.8675 - val_loss: 0.5932
Epoch 28/30
50/50            2s 35ms/step -
accuracy: 0.9290 - loss: 0.3665 - val_accuracy: 0.8350 - val_loss: 0.6155
Epoch 29/30
50/50            2s 39ms/step -
accuracy: 0.9450 - loss: 0.3578 - val_accuracy: 0.8600 - val_loss: 0.6614
Epoch 30/30
50/50            2s 39ms/step -
accuracy: 0.9376 - loss: 0.3696 - val_accuracy: 0.8600 - val_loss: 0.5913
```

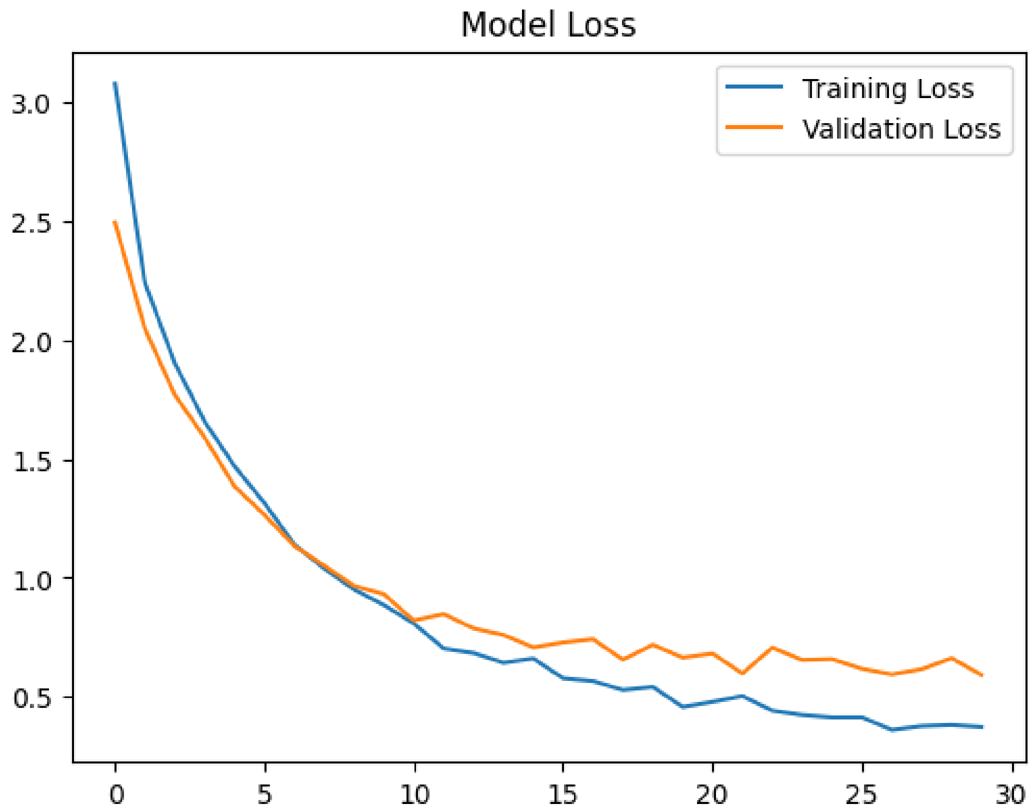
6.0.4 Evaluation

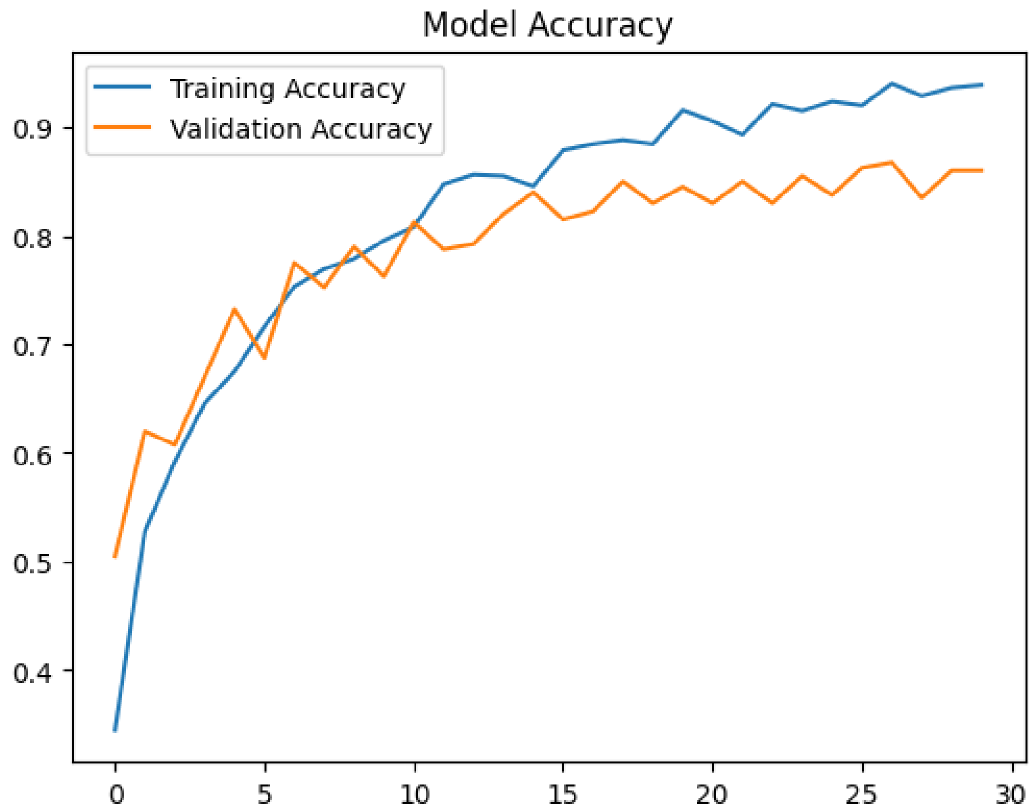
[61]:
```python
# Evaluate the model on test data
test_loss, test_acc = cnn_model.evaluate(X_test, Y_test)
print(f'Test Accuracy: {test_acc:.4f}')
```

```
16/16              0s 8ms/step -
accuracy: 0.9176 - loss: 0.4403
Test Accuracy: 0.9000
```

6.0.5 Loss and Accuracy Plot

[62]:
```python
# Plot the training and validation loss/accuracy
import matplotlib.pyplot as plt

plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("Model Loss")
plt.legend()
plt.show()

plt.plot(history.history["accuracy"], label="Training Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("Model Accuracy")
plt.legend()
plt.show()
```


Model Loss
Training Loss
Validation Loss
3.0
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
30



6.0.6 Classification report

[63]:
```
# Generate classification report
cnn_pred = cnn_model.predict(X_test)
cnn_pred_classes = tf.math.argmax(cnn_pred, axis=1)  # Get predicted class␣
↪labels
```

```
16/16          0s 17ms/step
```

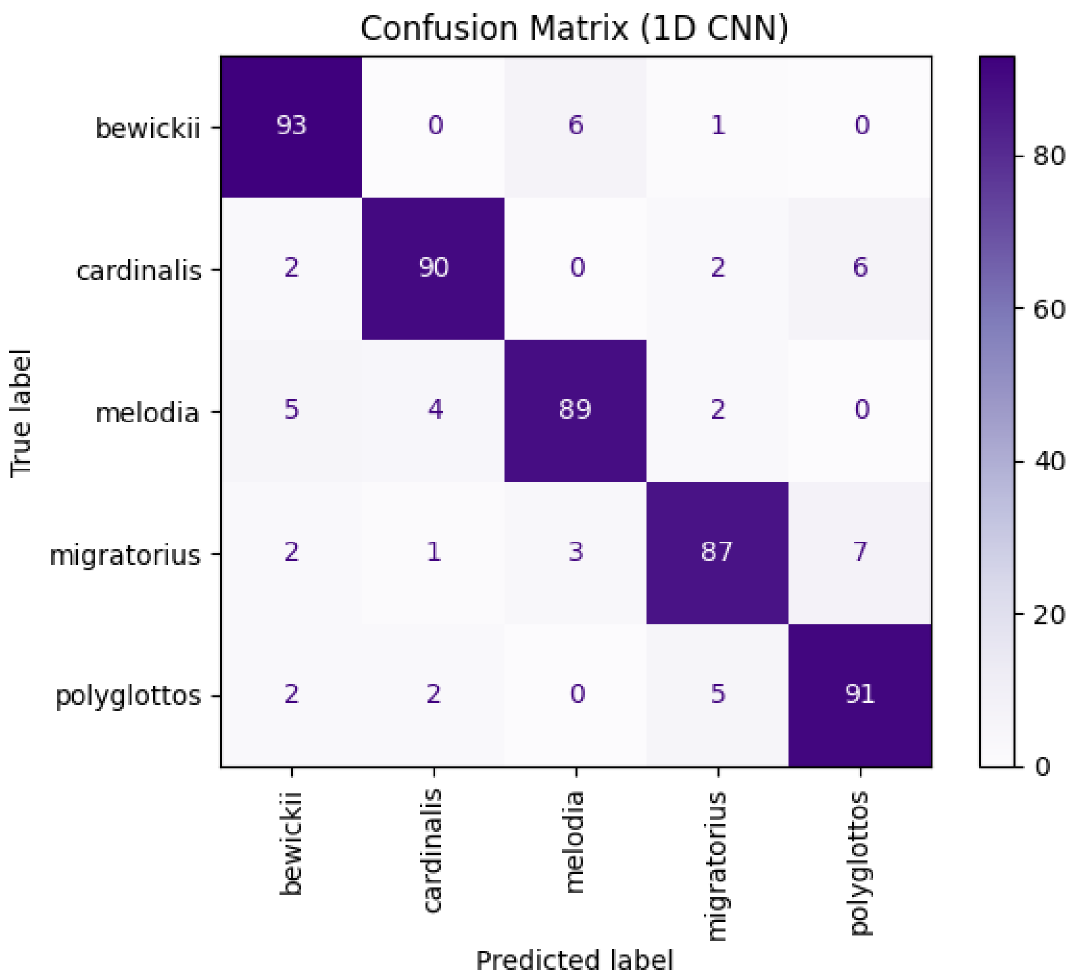
[64]:
```
print(classification_report(Y_test, cnn_pred_classes,␣
↪target_names=class_labels))
```

```
              precision    recall  f1-score   support

    bewickii       0.89      0.93      0.91       100
  cardinalis       0.93      0.90      0.91       100
     melodia       0.91      0.89      0.90       100
 migratorius       0.90      0.87      0.88       100
 polyglottos       0.88      0.91      0.89       100

    accuracy                           0.90       500
```

```
   macro avg       0.90      0.90      0.90       500
weighted avg       0.90      0.90      0.90       500
```

6.0.7 Confusion Matrix

[65]:
```
# Generate confusion matrix
cnn_confusion_matrix = confusion_matrix(Y_test, cnn_pred_classes)
confusion_matrix_plot(cnn_confusion_matrix, '1D CNN')
```



7 Save model

[66]:
```
import pickle
```

[67]:
```
# Models to save
models = ['KNN.pkl', 'SVC.pkl', 'Random Forest.pkl','1D_CNN.pkl']
```

```
# Location for saving the model (including model name and extension)
save_path = [f'/Users/user/Jupyter/my project/{model}' for model in models]

for model, path in zip([knn, svc, rfc_model, cnn_model], save_path):
    with open(path, "wb") as f:# Open the file in binary write mode ('wb') for
 ↪writing the pickled model
      pickle.dump(model, f)

print('Models saved successfully')
```

```
Models saved successfully
```

7.0.1 Save scalar statistics

[68]:
```
# save the precalculated scaling statistics mean and standard deviation for cnn
 ↪model
np.save('mean_cnn.npy', mean_cnn)
np.save('std_cnn.npy', std_cnn)

# save the precalculated scaling statistics mean and standard deviation for
 ↪machine-learning model
np.save('mean.npy', mean)
np.save('std.npy', std)
```

APPENDIX – III
FLASK

```
import os
from flask import Flask, render_template, request, app
import pickle
import librosa
import numpy as np
import tensorflow as tf
from werkzeug.utils import secure_filename


def extract(signal):
  sr = 2205

  # Extract MFCCs
  mfccs = librosa.feature.mfcc(y=signal, sr=sr, n_mels=20, n_mfcc=130)
  mfcc = mfccs.T

  spectral_centroid = (librosa.feature.spectral_centroid(y=signal, sr=sr)).T
  bandwidth = (librosa.feature.spectral_bandwidth(y=signal, sr=sr)).T
  rms = (librosa.feature.rms(y=signal)).T

  return np.concatenate((mfcc , bandwidth, spectral_centroid, rms), axis=1)


def scale(data):
  data_reshaped = np.expand_dims(data, axis=0)
  reshaped_data = data_reshaped.reshape(data_reshaped.shape[0], -1)
  scaled_data = (reshaped_data - mean_cnn) / std_cnn
  new_data = scaled_data.reshape(data_reshaped.shape)

  return new_data


# Load model, precalculated mean and stantard deviation of cnn model
model = pickle.load(open('1D_CNN.pkl', 'rb'))
mean_cnn = np.load('mean_cnn.npy')
```

```
std_cnn = np.load('std_cnn.npy')

app = Flask(__name__, static_folder='static')

# Define allowed audio extensions
ALLOWED_EXTENSIONS = {'mp3', 'wav', 'ogg', 'flac'}
# Set the maximum content length (in bytes)
app.config['MAX_CONTENT_LENGTH'] = 10 * 1024 * 1024  # 10 Megabytes

def allowed_file(filename):
  return '.' in filename and \
      filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/')
def home():
  return render_template('home.html')


@app.route('/predict', methods=['POST'])
def upload_audio():
  if 'audio_file' not in request.files:
    return render_template('error.html', message='No file selected. Please select an
                                                   audio file')

  file = request.files['audio_file']
  if file.filename == '':
    return render_template('error.html', message='No file selected. Please select an
                                                   audio file')

  if file and allowed_file(file.filename):
    filename = secure_filename(file.filename)

    file.save(f'uploads/{filename}')  # Save the file to a designated location
    filepath = os.path.join(os.getcwd(),f'uploads/{filename}')  # Get the filepath

    # Preprocess audio data
    y, sr = librosa.load(filepath, duration=3)

    # Remove filepath
     os.remove(filepath)
```

```
    if len(y) == 0:
        return render_template('error.html', message='Cannot predict bird species
                                      with this audio. Please select another audio')
    else:
        features = extract(y)
        scaled_data = scale(features)

        class_labels = {0: 'bewickii', 1: 'cardinalis', 2: 'melodia',
                        3: 'migratorius', 4: 'polyglottos'}

        # Use bird classification model
        prediction = model.predict(scaled_data)
        # Find the class with the highest probability
        predicted_class = tf.math.argmax(prediction, axis=1)
        # Find the highest probability
        predicted_prob = prediction[0][int(predicted_class)]

        if predicted_prob < 0.7:
            return render_template('error.html', message='Cannot determine species')
        else:
            return render_template(f'result_{int(predicted_class)}.html')

  else:
      return render_template('error.html', message='Unsupported file format. '
                  'Please select an audio file with extension .wav, .mp3, .ogg or .flac')


if __name__ == '__main__':
  app.run(debug=True)
```

APPENDIX - IV

HTML FILES

Home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Bird Sound Classifier</title>
      <style type="text/css">
          body {
                    margin: 0;
                    padding: 0;
                    background-image: url("{{ url_for('static', filename='bird5.jpeg') }}");
                    background-size: cover;
                    background-position: center;
                    background-attachment: fixed;
                    font-family: sans-serif;
                    }
          .container {
                    background-color: rgba(255, 255, 255, 0.6);
                    top:0;
                    padding: 16px;
                    border: none;
                    }
           .input {
                    color: black;
                    padding: 16px;
                    font-size: 16px;
                    border: none;
                     }
          .btn {
                    color: black;
                    background: rgba(255, 255, 255, 0.5);
                    padding: 5px;
                    border : black;
                    font-size: 16px;
                    }
```

```css
.btn-btn {
        color: white;
        background: green;
        padding: 10px;
        border-radius: 10px;
        font-weight: 600;
        border : solid black;
        font-size: 16px;
        }
.text {
        background: rgba(255, 255, 255, 0.6);
        width: 600px;
        color: black;
        padding: 16px;
        font-size: 16px;
        border: none;
        }
.link {
        background-color: rgba(240, 173, 38, 0.6);
        color: black;
        padding: 16px;
        font-size: 14px;
        border: none;
         }

.error {
        animation: blink 2s ease-in-out infinite;
         }

@keyframes blink {
  0% { opacity: 1; }
  50% { opacity: 0; }
  100% { opacity: 1; }
  }

. footer {
        bottom:0;
        padding: 16px;
        width:700px;
        background: rgba(255, 255, 255, 0.7);
        }
```

```html
    </style>
</head>
 <body>
    <center>
    <div class="container">
        <h1 style="font-size: 280%; color: #103bc9; font-weight: 900; ">
                                    Bird Sound Classifier </h1
         <p style="font-weight:600;">
             Upload a bird sound recording and identify the bird species! </p>
     </div>
     < br>
     <div>
         <form action="/predict" method="post" enctype="multipart/form-data">
        <input class="btn" type="file" id="file" name="audio_file"
                                  accept=".mp3,.wav,.ogg,.flac" style="font-size:90%;" >
     <br>
     <br>
        <button class="btn-btn" type="submit">Upload and Predict</button>
         <p  style="color:black; font-weight:400; font-size: 14px;">
                          Upload an audio with in 10Mb </p>
          </form>
      </div>
      <br>
        {% block content %}


        {% endblock %}
      <br>
      <div class="footer">
         <p style="color:red; font-weight:500; font-size: 17px;">
                 Note : Our model has an accuracy of 91%.
                 There is 10 % chance of misclassiffication </p>
      </div>
 </body>
</html>
```

Result_0.html

```
{% extends 'home.html' %}

{% block content %}

  <img src="/static/bewikki.jpeg" alt="Bewick's wren" width="500" height="400">
  <br>
  <br>
  <div class="text">
       <p style="color:#ad1e09; font-weight: 900;">
             Bird species : Bewickii<br>
             Common name : Bewick's wren</p>

       <p style="font-weight: 600;">
          The Bewick's wren is a wren native to North America.
          It is the only species placed in the genus Thryomanes.
          At about 14 cm long, it is grey-brown above, white below,
          with a long white eyebrow. While similar in appearance to the
          Carolina wren, it has a long tail that is tipped in white. </p>
  </div>
  <br>
  <br>
  <div style="margin-bottom:40px">
       <a class="link" href="https://en.wikipedia.org/wiki/Bewick's_wren">
             For more information about Bewick's wren</a>
  </div>

{% endblock %}
```

Result_1.html

```
{% extends 'home.html' %}

{% block content %}

  <img src="/static/cardinalis.jpeg" alt="Northern Cardinal" width="500" height="400">
  <br>
```

```
  <br>
  <div class="text">
        <p style="color:#ad1e09; font-weight: 900;">
              Bird species : Cardinalis<br>
              Common name : Northern Cardinal</p>

        <p style="font-weight: 600;">
          The male Northern cardinal is a perfect combination of familiarity,
          conspicuousness, and style: a shade of red you can't take your eyes off.
          Even the brown females sport a sharp crest and warm red accents.
          Cardinals don't migrate and they don't molt into a dull plumage,
          so they're still breathtaking in winter's snowy backyards.
          In summer, their sweet whistles are one of the first sounds of the
          morning.</p>
  </div>
  <br>
  <br>
  <div style="margin-bottom:40px">
        <a class="link" href="https://en.wikipedia.org/wiki/Northern_cardinal">
              For more information about Northern Cardinal</a>
  </div>

{% endblock %}
```

Result_2.html

```
{% extends 'home.html' %}

{% block content %}

  <img src="/static/melodia.jpeg" alt="Song sparrow" width="500" height="400">
  <br>
  <br>
  <div class="text">
        <p style="color:#ad1e09; font-weight: 900;">
              Bird species : Melodia<br>
              Common name : Song Sparrow </p>
```

```
        <p style="font-weight: 600;">
                The song sparrow is a medium-sized New World sparrow.
                Among the native sparrows in North America, it is easily one
                of the most abundant, variable and adaptable species.</p>
  </div>
  <br>
  <br>
  <div style="margin-bottom:40px">
        <a class="link" href="https://en.wikipedia.org/wiki/Song_sparrow">
                For more information about Song sparrow</a>
  </div>

{% endblock %}
```

Result_3.html

```
{% extends 'home.html' %}

{% block content %}

  <img src="/static/migratorius.jpeg" alt="American robin" width="500"
                                                            height="400">
  <br>
  <br>
  <div class="text">
        <p style="color:#ad1e09; font-weight: 900;">
                Bird species : Migratorius<br>
                Common name : American robin</p>

        <p style="font-weight: 600;">
            The quintessential early bird, American Robins are common sights on
            lawns across North America, where you often see them
            tugging earthworms out of the ground. Robins are popular birds
            for their warm orange breast, cheery song, and early appearance
            at the end of winter. Though they're familiar town and city birds,
            American Robins are at home in wilder areas, too, including mountain
            forests and Alaskan wilderness.</p>
```

```
   </div>
   <br>
   <br>
   <div style="margin-bottom:40px">
        <a class="link" href="https://en.wikipedia.org/wiki/American_robin">
              For more information about American robin</a>
   </div>

{% endblock %}
```

Result_4.html

```
{% extends 'home.html' %}

{% block content %}

   <img src="/static/polyglottos.jpeg" alt="Northern mockingbird" width="500"
                                                       height="400">
   <br>
   <br>
   <div class="text" >
        <p style="color:#ad1e09; font-weight: 900;">
              Bird species : Polyglottos <br>
              Common name : Northern mockingbird</p>

        <p style="font-weight: 600;">
          The Northern Mockingbird is a mockingbird commonly found in
          North America, of the family Mimidae. These slender-bodied gray
          birds apparently pour all their color into their personalities.
          They sing almost endlessly, even sometimes at night,
          and they flagrantly harass birds that intrude on their territories,
          flying slowly around them or prancing toward them, legs extended,
          flaunting their bright white wing patches.</p>
   </div>
   <br>
   <br>
   <div style="margin-bottom:40px">
        <a href="https://en.wikipedia.org/wiki/Northern_mockingbird">
```

```
                    For more information about Northern mockingbird</a>
   </div>

{% endblock %}
```

Error.html

```
{% extends 'home.html' %}

{% block content %}

    <div class="text">
        <h1 class="error" style="font-size:28px; color: red;
                                        font-weight:700;">ERROR</h1>
            <p class="error" style="font-size:16px; color: black; font-weight:600;">
                    {{message}}</p>
    </div>

{% endblock %}
```