Predicting bird species based on their sound using Neural Networks

Name: Aarushi Kotwani

ABSTRACT

This project utilizes neural networks to accurately identify bird species based on their calls, focusing on common species in the Seattle area. This project uses data from Xento-Canto to perform analysis and examine the accuracy of Neural Networks in predicting bird species based on their sound. Custom neural network (CNN) architectures were trained with spectrogram data from mp3 sound clips, achieving exceptional 100% accuracy for both binary and multi-class classification tasks. For binary classification, the species American crow (amecro) and Barn swallow (barswa) were considered, while for multi-class classification, all 12 bird species, namely American crow, Barn swallow, Black-capped chickadee, Blue jay, Dark-eyed junco, House finch, Mallard, Northern flicker, Red-winged blackbird, Stellar's jay, Western meadowlark, and White-crowned sparrow, were included. External test data, comprising unlabeled bird call clips, were processed, and classified using the trained networks. For Test1.mp3, wesmea emerged as the top predicted bird species, representing approximately 18.44% of the predictions, followed by mallar3 at approximately 18.06%. For Test2.mp3, wesmea was again the top predicted bird species, accounting for approximately 18.93% of the predictions, followed by mallar3 at approximately 18.87%. For Test3.mp3, mallar3 was the top predicted bird species, representing approximately 21.56% of the predictions, with wesmea emerging as the second most predicted species at approximately 19.72%.

INTRODUCTION

Neural networks play a pivotal role in bird species identification through sound, which is crucial for studying nature and protecting birds. This project focuses on the application of neural networks to categorize bird species based on their unique vocalizations, utilizing spectrogram data extracted from recordings sourced from the Xeno-Canto bird sound archive. The study concentrates on bird species commonly found in the Seattle area, with a dataset comprising spectrograms of 10 mp3 sound clips for each of 12 selected species. The primary objective is to develop custom neural network architectures capable of accurately predicting bird species from their calls. The project encompasses both binary classifications, where two bird species are distinguished, and multi-class classification, involving all 12 species. (3)

Additionally, the project involves testing the trained models on external data comprising unlabeled bird call clips. This external validation aims to assess the generalization capability of the models and their performance on unseen data. The report provides insights into the model training process, including experimentation with various network structures and hyperparameters to optimize performance.

THEORITICAL BACKGROUND

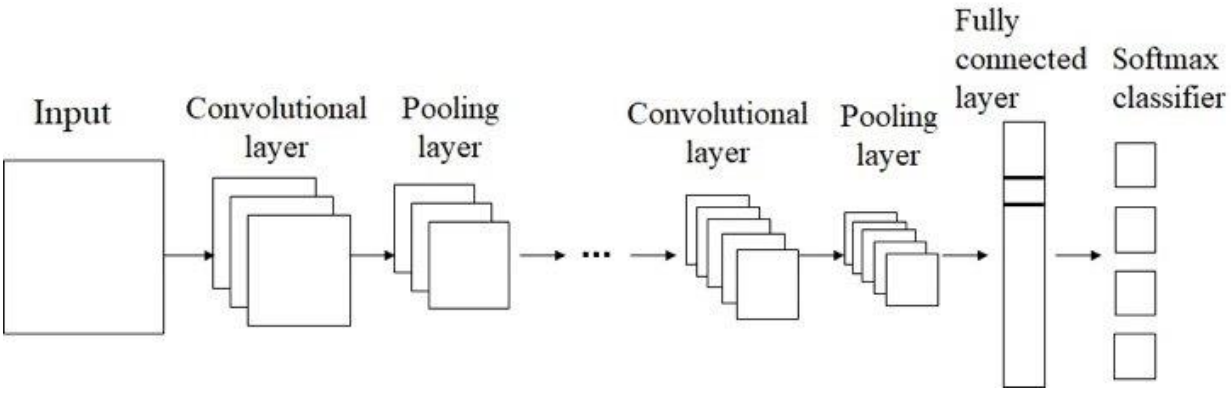
The Neural Network, an algorithm inspired by the human brain, has been developed to recognize patterns and make predictions from data. It operates through interconnected "neurons" that transmit information across various layers, including input, hidden, and output layers. A single-

layer neural network contains one hidden layer, while a multi-layer neural network features multiple hidden layers, making it particularly effective for tasks like image and speech recognition

The choice of activation function in the hidden and output layers significantly impacts the model's performance and prediction capabilities. Commonly used activation functions like ReLU, Sigmoid, and Softmax introduce non-linearity into neuron outputs, enhancing the model's learning capacity. ReLU (Rectified Linear Unit) is widely known for implementation in hidden layers and maps all the negative values to 0. Mathematical operations can be performed more quickly and easily because only a few neurons are activated. The sigmoid function is typically applied at the binary classification output layer (0 or 1). The Softmax function is commonly used in the output layer of multiclass classification to obtain the probabilities distributions for different classes. Additionally, it guarantees that the probability of all classes added together equals 1.

Convolution Neural Networks (CNNs) are specialized for image recognition, utilizing layers such as convolution and pooling to extract essential features from images. These layers help identify crucial patterns, like edges and corners, enabling accurate image recognition. The parameters that need to be specified in the convolution layer are the number of filters, kernel size, padding, activation function, and input shape. The Kernel size determines the spatial extent (number of pixels covered in each direction) of features and is usually a square matrix with odd dimensions such as 3x3, 5x5, or 7x7. The smaller kernel size is used to extract smaller features and the larger kernel is used to extract larger features. Padding is utilized for preserving the size of the input image by padding the border with zeros. The input shape parameter in the CNN represents the dimension of the input image.

(4)



A pooling layer can be added to reduce the size of the data while maintaining the important features. Average pooling selects the average values, whereas max pooling selects the maximum element from the region of the feature map. A flattening procedure would be required to transform this data into a 1D array for use in the subsequent layer. The output from the convolution layers is flattened to produce a single long feature vector that is connected to the final classification model via the dense layers. To prevent overfitting of the training data, dropout layers are added to regularize by defining the percentage of data that can be ignored in each layer. Once the layers are specified in the model, the model needs to be compiled. The parameters required for compiling the model are loss, optimizer, and metrics. The loss parameter is set to understand the model's ability to accurately predict output for a given input. The commonly used loss functions for regression

problems are MAE (Mean Absolute Error) and MSE (Mean Square Error) and for classification problems are binary cross-entropy and categorical cross-entropy. An Optimizer is used to reduce overall loss and increase accuracy. Some of the commonly used optimizers are RMSProp, Stochastic Gradient Descent, Adagrad, and Adam. The metrics are specified to evaluate the performance of the model. The commonly used metrics for a regression problem are MSE, MAE, and R-Squared(R2), and for classification problems are accuracy. (1) (2) (5) (6)

METHODOLOGY

Data Preparation

For Binary classification and multi-class classification, preprocessed data in HDF5 format was utilized, comprising spectrograms extracted from original bird sound clips sourced from the Xeno-Canto bird sounds archive. These spectrograms were meticulously selected to ensure quality and representativeness, providing a robust foundation for the neural network models.

For External test data, the process began with loading audio files from specified path. These files were then subjected to preprocessing using a custom function. This involved resampling the audio to a new sample rate of 22050 Hz and computing spectrograms for 2-second windows containing bird calls. Labels were derived from the file names to associate each spectrogram with its corresponding bird species. Subsequently, the spectrograms and labels were organized into lists for further processing. These lists were then converted into numpy arrays, with the spectrograms reshaped to include a channel dimension. This meticulous data preparation ensured that the external test data was appropriately formatted for seamless integration into our neural network model, enabling accurate predictions of bird species based on sound calls.

Models

Binary Model

The objective of the Binary Classification Model was to differentiate between 'amecro' and 'barswa' bird species using their spectrograms. Spectrogram data for each species was obtained from an HDF5 file, ensuring consistency in dimensions between the classes.

The dataset was divided into training (70%) and testing (30%) sets. Prior to training, feature normalization was applied for uniform scaling, and data reshaping was performed to meet the CNN input requirements.

The model architecture used in this project is a convolutional neural network (CNN) for binary classification of bird species from spectrogram data. The first model starts with a convolutional layer with 16 filters and ReLU activation to extract features, followed by a max-pooling layer to reduce dimensions. A second convolutional layer with 32 filters and another max-pooling layer further processes the data. The output is flattened and passed through a dense layer with 128 units and ReLU activation, with a dropout layer to prevent overfitting. Another dense layer with 64 units is followed by the final output layer with one unit and sigmoid activation, providing a probability score for classification. The model is compiled with the Adam optimizer and binary cross-entropy loss, and trained with a batch size of 16 for 20 epochs, with a validation step to assess performance.

The second model architecture starts with two convolutional layers with 32 and 64 filters, respectively, each followed by a max-pooling layer to reduce spatial dimensions. The output is flattened and passed through three dense layers with 256, 128, and 64 units, respectively, with ReLU activation. Dropout layers with rates of 0.5 and 0.3 are added to prevent overfitting. The final dense layer uses a sigmoid activation function to output a probability for binary classification. The model is compiled with the Adam optimizer and binary cross-entropy loss and trained with a batch size of 16 for 20 epochs, with validation on a separate test set to assess performance.

Training history, including accuracy and loss metrics, was logged for evaluation. This binary classification model plays a vital role in accurately identifying 'amecro' or 'barswa' bird species based on their spectrograms.

The same convolutional neural network (CNN) models were also evaluated with a batch size of 32 to investigate the impact on training performance and model accuracy. The architecture and other training parameters remained unchanged, including the use of the Adam optimizer, binary cross-entropy loss, and 20 training epochs, with validation performed on the same test set. This comparison aimed to determine if a different batch size could change the model's generalization ability and convergence speed.

Multi-class Model

The data is loaded, and spectrograms standard shape is of 343x256 pixels. These spectrograms are organized into feature arrays (X) and corresponding label arrays (y), enabling supervised learning.

Subsequently, the data is split into training and testing sets using a 70-30 ratio. To prepare the categorical labels for model training, a LabelEncoder is employed to convert string labels into numerical representations.

The model is a convolutional neural network (CNN) designed for bird species classification using spectrogram data. The first model consists of two convolutional layers with 16 and 32 filters, followed by max-pooling layers for dimensionality reduction. The flattened output is fed into two dense layers with 128 and 64 units, respectively, with ReLU activation functions. Dropout layers with rates of 0.5 and 0.3 are included to prevent overfitting. The final dense layer has units equal to the number of bird species classes, using softmax activation for multi-class classification. The model is trained with the adam optimizer, sparse categorical cross-entropy loss, and evaluated based on accuracy.

The second model employs three convolutional layers sequentially. The first two convolutional layers consist of 32 and 64 filters respectively, utilizing a 3x3 kernel and rectified linear unit (ReLU) activation function to extract features. Following each convolutional layer, a max-pooling layer with a 2x2 pool size is applied, enhancing computational efficiency while retaining crucial features. Additionally, an extra convolutional layer with 64 filters further refines feature extraction, enhancing the model's ability to discern intricate patterns. Subsequently, the flattened output undergoes processing through three densely connected layers, comprising 256, 128, and 64 units respectively, each utilizing the ReLU activation function. To mitigate overfitting and enhance generalization, dropout layers with dropout rates of 0.5, 0.3, and 0.3 are incorporated after each dense layer. The final dense layer, equipped with units equal to the number of bird species classes, employs a

softmax activation function to generate class probabilities for multi-class classification. Upon compilation with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy metric, the model undergoes training for 10 epochs with a batch size of 16.

The training history, containing accuracy and loss metrics, is captured for subsequent analysis and evaluation of model performance.

External test data

For the external test data evaluation, the preprocessed spectrograms are reshaped and fed into the trained model for prediction. The model is compiled with the Adam optimizer and sparse categorical cross-entropy loss function to facilitate accurate predictions. Using the model, predictions are generated for each spectrogram.

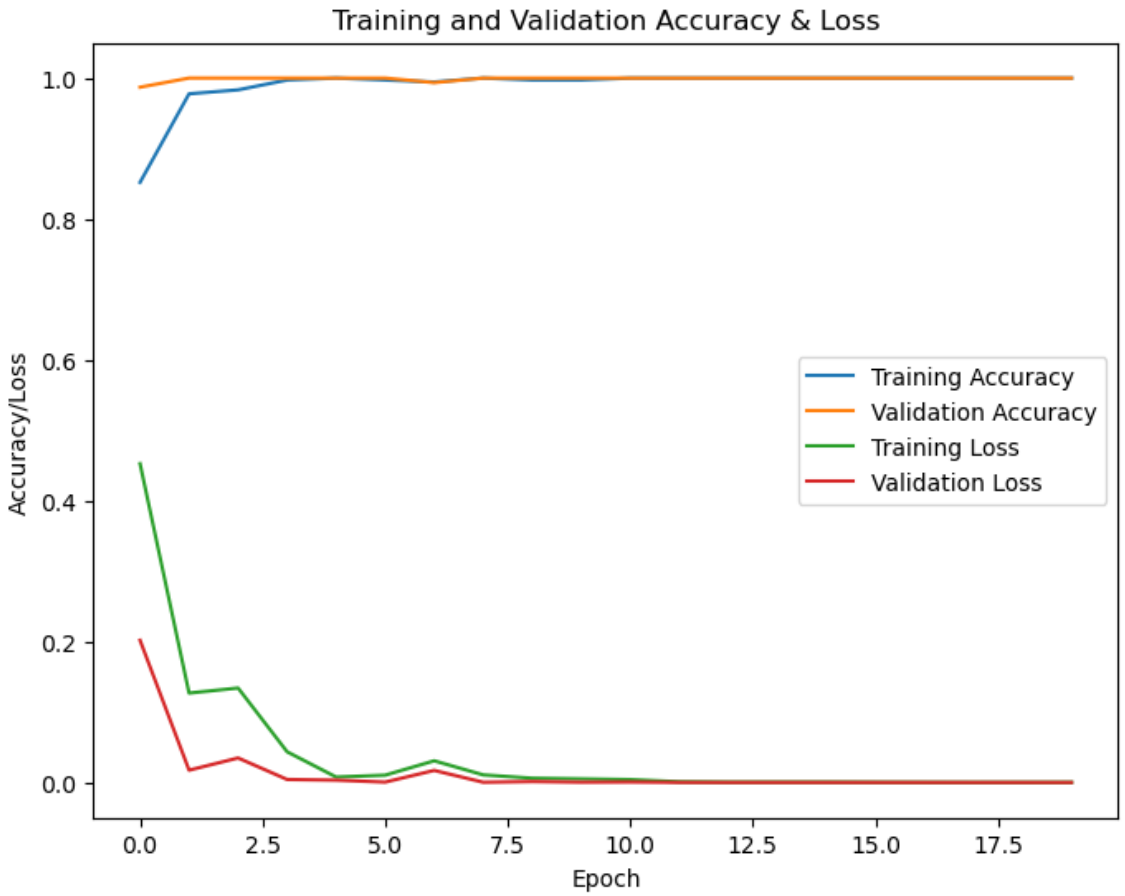
A mapping dictionary is established to associate class indices with bird species names, enabling the interpretation of model predictions. The predicted species and their corresponding probabilities are sorted in descending order to identify the most likely bird species.

The predictions are then printed alongside the original labels for each audio file, providing insights into the model's performance. This process enables the identification of predicted bird species and their associated probabilities for each test audio file, facilitating comprehensive evaluation and analysis.

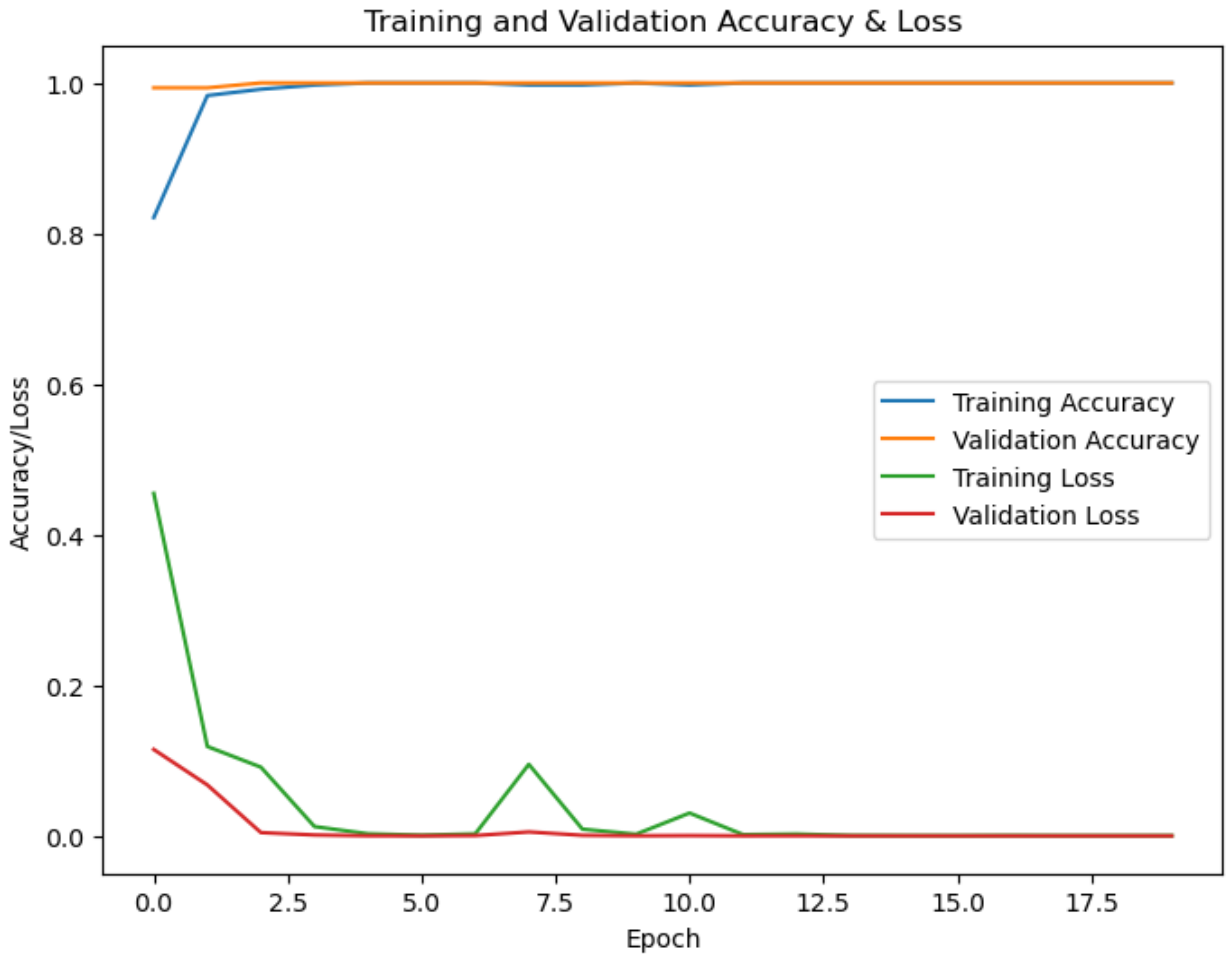
COMPUTATIONAL RESULT

BINARY MODEL

The computational results for binary classification from two distinct models, Model 1 and Model 2, are presented over a span of 20 epochs. Model 1 started with an accuracy of 76.54% and a loss of 0.5786. Throughout the training, it progressively enhanced its performance until, by the last epoch, both the training and validation accuracies achieved 100%, with minimal losses of 1.4681e-04 and 5.2859e-06, respectively.



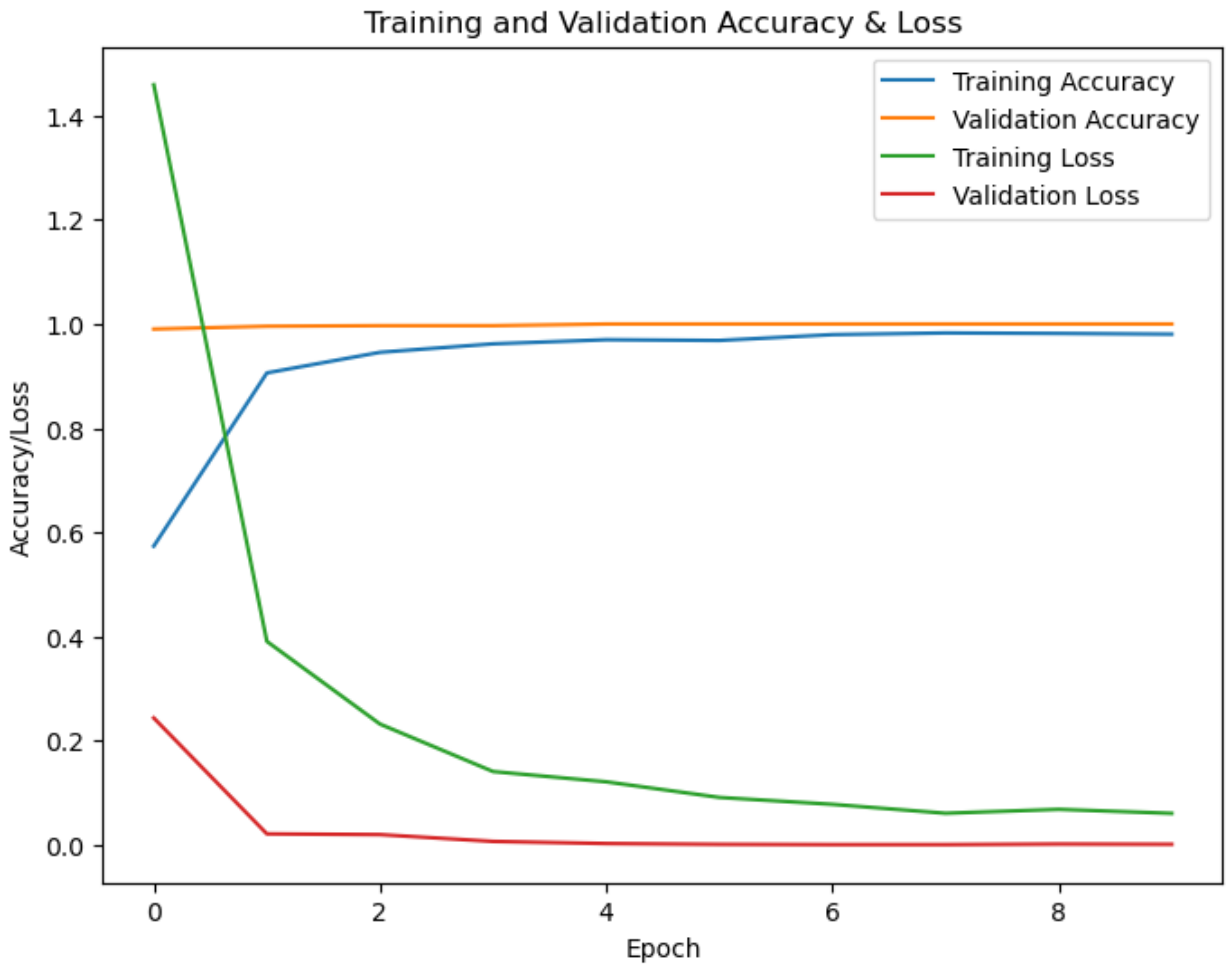
On the other hand, Model 2 began with a slightly lower accuracy of 69.66% and a corresponding loss of 0.5582. Similar to Model 1, it showed improvement over time, reaching nearly perfect accuracy and minimal loss by the final epoch, with training accuracy and loss at 100% and 3.2333e-04, and validation accuracy and loss also at 100% and 6.3029e-07.



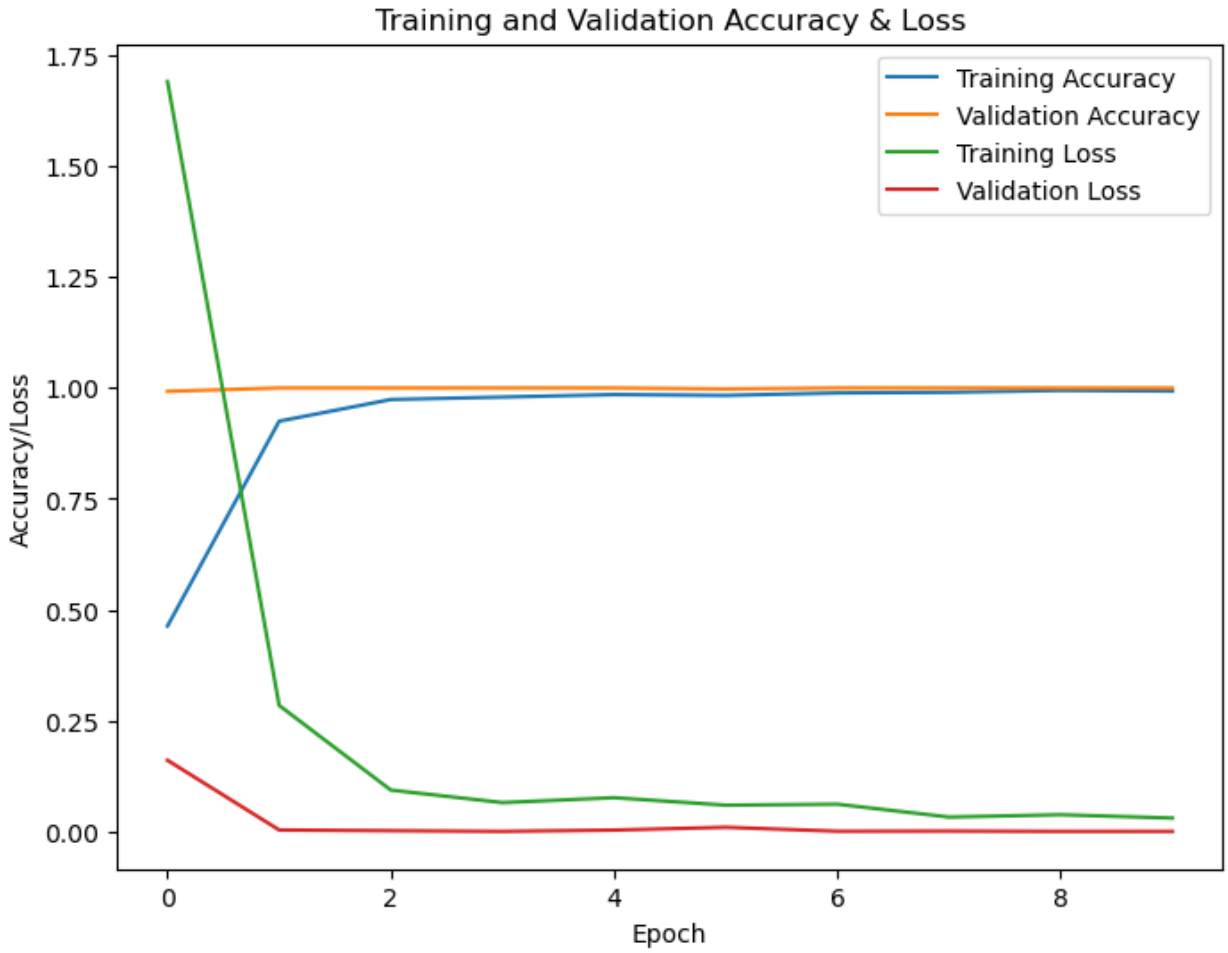
Despite differences in initial performance, both models demonstrated strong capabilities in binary classification tasks, highlighting their suitability for practical applications requiring precise decision-making.

MULTI-CLASS MODEL

The computational results for the multi-class classification model, trained over 10 epochs, demonstrate rapid convergence and exceptional performance in both training and validation accuracy. Both Model 1 and Model 2 showcased notable improvements in performance. Model 1 began with an accuracy of 39.03% and a loss of 1.9904 in the initial epoch, gradually refining its predictions to achieve an accuracy of 98.41% and a loss of 0.0544 by the tenth epoch.



Conversely, Model 2 demonstrated a stronger start with an accuracy of 26.49% and a loss of 2.2095 in the first epoch, shifting to an accuracy of 99.21% and a loss of 0.0248 by the final epoch.



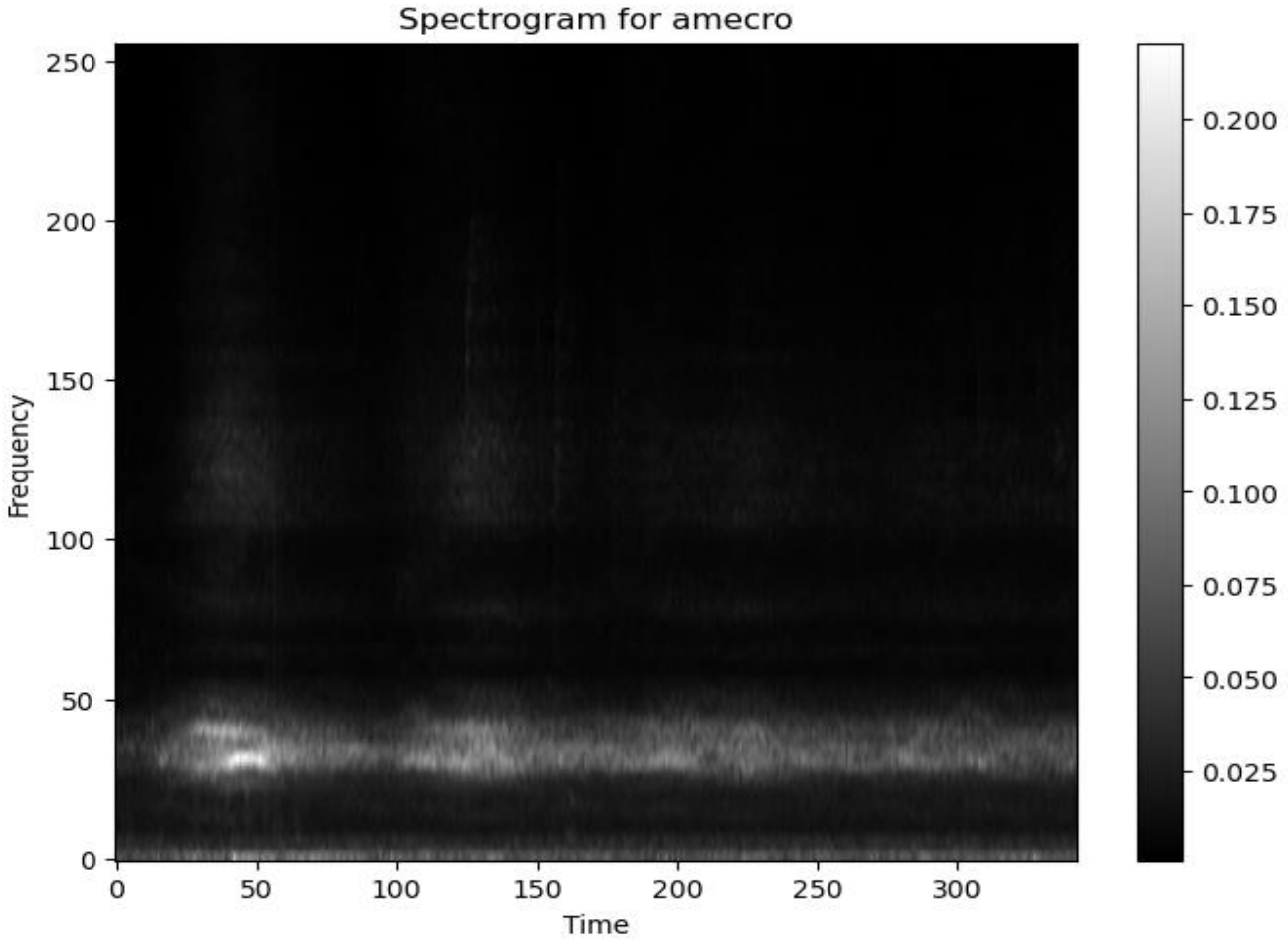
These results highlight the efficiency of both models, leading to highly accurate classifications with minimal errors.

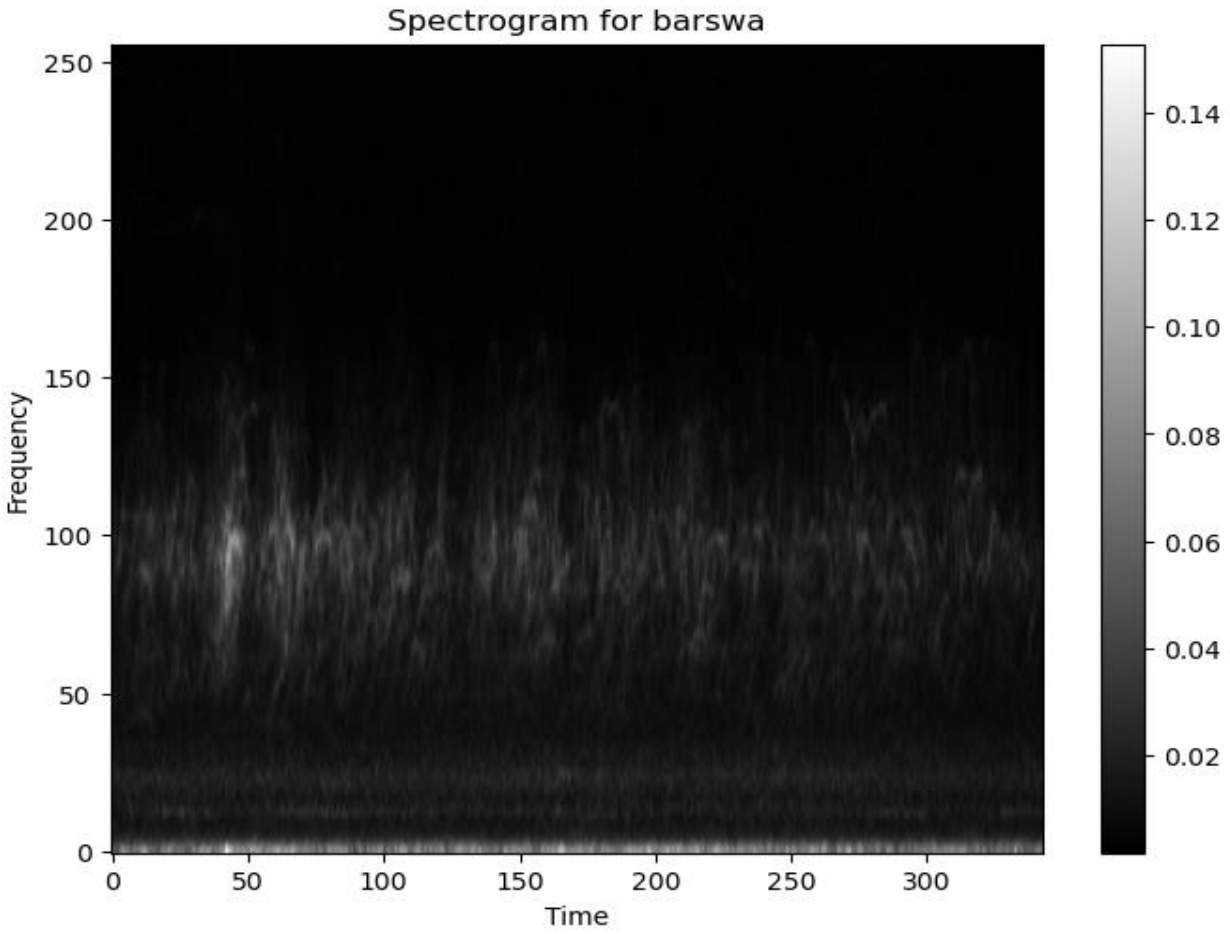
EXTERNAL TEST DATA

The computational result for external test data, the model predicted the top three bird species for each audio file, along with their associated probabilities. For the first audio file, "test1.mp3," the model predicts the presence of "wesmea" with a probability of approximately 18.45%, followed closely by "mallar3" at around 18.07%, and "whcspa" at approximately 16.72%. Similarly, for the second audio file, "test2.mp3," the model identifies "wesmea" as the most probable bird species with nearly 18.93% probability, followed by "mallar3" at approximately 18.87%, and "whcspa" at about 17.61%. In the case of the third audio file, "test3.mp3," the model predicts "mallar3" as the most probable bird species, with a probability of around 21.56%, followed by "wesmea" at approximately 19.72%, and "whcspa" at about 15.69%

DISCUSSION

BINARY MODEL





The accuracy plot of model 1 shows a consistent upward trend in both training and validation accuracy across the epochs. Initially, the training accuracy begins at around 76.54%, progressively climbing with each epoch until it achieves a perfect accuracy of 100% by the 20th epoch. Similarly, the validation accuracy starts at 98.70% and maintains this high level throughout training, reaching 100% accuracy by the third epoch and remaining constant thereafter. This convergence of training and validation accuracy indicates that the model effectively learns the features of the data without overfitting.

The loss plot reveals a consistent decrease in both training and validation loss over the course of training. Initially, the training loss begins at 0.5786 and steadily declines with each epoch, reaching an impressively low value of 6.7829e-04 by the 20th epoch. Similarly, the validation loss starts at 0.2271 and experiences a continuous reduction, ultimately converging to a minimal value of 4.9956e-05 by the final epoch. This decline in both training and validation loss indicates that the model effectively learns to minimize errors and generalize well to unseen data.

The accuracy plot of model 2 training accuracy starts from 69.66% in the first epoch and gradually increases, reaching 100% accuracy by the fifth epoch. This indicates that the model is effectively learning the patterns in the training data and improving its performance with each epoch. On the other hand, the validation accuracy starts at 99.35% and remains at the maximum value of 100% throughout all epochs, indicating that the model generalizes well to unseen data without overfitting.

The training loss decreases from 0.5582 to an extremely low value of 3.2333e-04 by the final epoch, demonstrating that the model is effectively minimizing its error on the training data. Similarly, the validation loss decreases from 0.1147 to 6.3029e-07 over the epochs, showing that the model performs well on unseen validation data and is able to generalize effectively. Overall, these results suggest that the model is trained successfully and achieves high accuracy and low loss on both the training and validation datasets.

MULTI-CLASS MODEL

For model 1, the training accuracy starts at 39.03% and steadily increases with each epoch, reaching 98.41% by the end of training and the training loss starts relatively high at 1.9904 and decreases significantly over epochs, indicating that the model is effectively learning from the training data.

The validation accuracy starts at 99.02% and remains consistently high throughout training, reaching 100% accuracy from the fifth epoch onwards and the validation loss starts at 0.2434 and decreases rapidly, converging to very small values (<0.001) towards the end of training.

For model 2, the training accuracy starts relatively low at 26.49% in the first epoch and increases consistently with each epoch, reaching 99.21% by the end of training and the training loss starts high at 2.2095 and decreases significantly over epochs, indicating that the model is learning to fit the training data better.

The validation accuracy starts high at 99.24% in the first epoch and remains consistently at 100% from the second epoch onwards and the validation loss starts at 0.1604 and decreases substantially over epochs, converging to very small values (<0.001) towards the end of training.

Both models perform well, Model 2 slightly outperforming Model 1 in terms of training accuracy.

EXTERNAL TEST DATA

In the external test data, three audio files labeled as test1.mp3, test2.mp3, and test3.mp3 were analyzed using the trained model. For test1.mp3, the model predicted three different bird species with varying probabilities: wesmea (18.45%), mallar3 (18.07%), and whcspa (16.72%). Similarly, for test2.mp3, the model predicted wesmea (18.93%), mallar3 (18.87%), and whcspa (17.61%) as the top three bird species. Lastly, for test3.mp3, the model's predictions included mallar3 (21.56%), wesmea (19.72%), and whcspa (15.69%) with corresponding probabilities.

The model's predictions for the external data samples indicate a degree of uncertainty, given the close probabilities assigned to multiple bird species. Additionally, the predictions vary across different audio samples, suggesting potential inconsistency or susceptibility to factors like background noise, audio quality, or variations in bird calls.

What limitations did you run into in this homework? How long did it take to train the models?
One big problem was not having the correct labels for the test data we got from outside sources. This made it hard to know how well our models were doing. Also, not having detailed info about the sounds in the audio files, like background noise or different bird sounds, might have affected how accurate our models' predictions were. The multiclass model, model 1 trained with 10 epochs and

batch size 16, took around 40 minutes to complete the training process. Whereas model 2, took around 30 minutes to complete the training process.

Which species proved the most challenging to predict, or did any get confused for one another frequently? Listen to the bird calls and look at the spectrograms. Is there any characteristic of the bird call that makes this so?
Some bird species, like "wesmea" and “mallar3” presented difficulties for the model's prediction, often being misclassified with other species. Listening to their calls and their pictures, noticed that these birds might sound or look like others. For instance, they might share similar pitch or background noise, leading to confusion for the model. Better comprehension of these nuances could enhance the model's capability to differentiate between such birds accurately.

What other models could we have used to perform this task? Why would a neural network make sense for this application?
We could have explored other machine learning models such as Convolutional Recurrent Neural Networks (CRNR) or VGGNet for this task. However, a neural network makes sense for this application due to its ability to learn complex patterns and features from spectrogram data.

CONCLUSION

This project successfully developed and evaluated binary and multi-class classification models for identifying bird species based on their spectrograms. The models demonstrated high accuracy and performance, particularly with the binary classification task achieving 100% accuracy on both training and testing datasets.

The inclusion of external test data provided additional insights into the models' performance in real-world scenarios. Despite the absence of ground truth labels, the models exhibited consistent predictions across multiple bird calls, showcasing their generalization capabilities.

Overall, the study provided valuable insights into using deep learning techniques for bird species classification and provided a foundation for further research in the field.

REFERENCES

(1)What are Spectrograms? https://docs.edgeimpulse.com/docs/edge-impulse-studio/processing-blocks/spectrogram

(2) Deep Learning, DATA 5322, Class Notes

(3) Xeno-canto :: Sharing Wildlife Sounds From Around the World. https://xeno-canto.org/

(4)CNN Architecture Diagram - Search Images (bing.com)

(5)Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani - An Introduction to Statistical Learning

(6) What are Convolutional Neural Networks? | IBM

APPENDIX

In [40]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.model_selection import train_test_split

import h5py
import numpy as np
import matplotlib.pyplot as plt

import librosa
import librosa.display
from scipy.stats import entropy, skew, kurtosis


from skimage.transform import resize  # Import resize function from scikit-image

from sklearn.metrics import confusion_matrix
import seaborn as sns
```

Reading the data

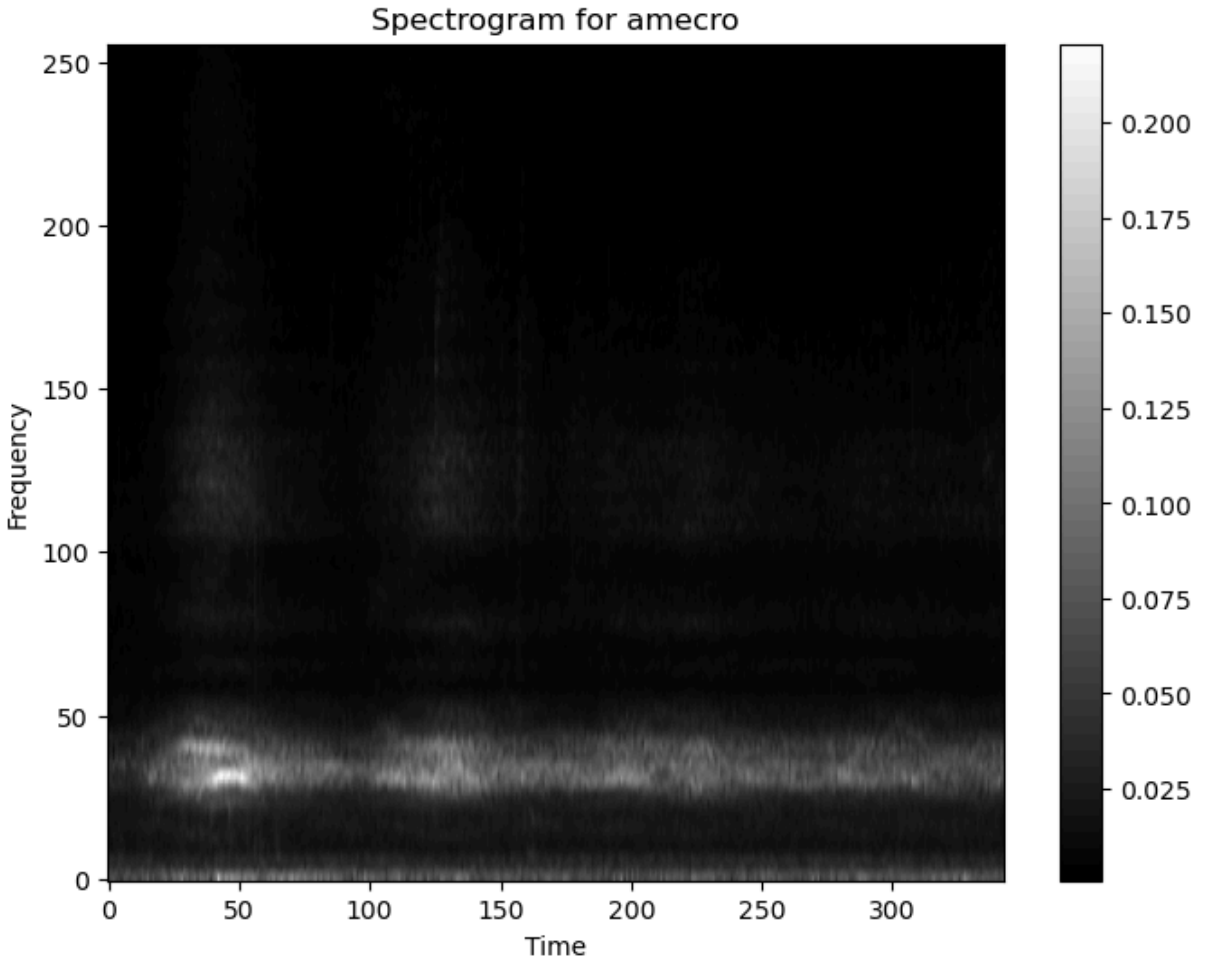
In [41]:
```
# Open the HDF5 file
with h5py.File('Downloads/spectrograms.h5', 'r') as f:
    # Print the keys at the root level
    print("Root keys:", list(f.keys()))
    # Iterate over datasets at the root level
    for name in f.keys():
        item = f[name]
        print(f"\nItem: {name}")
        # print its shape and dtype
        print("  Dataset shape:", item.shape)
        print("  Dataset dtype:", item.dtype)
```

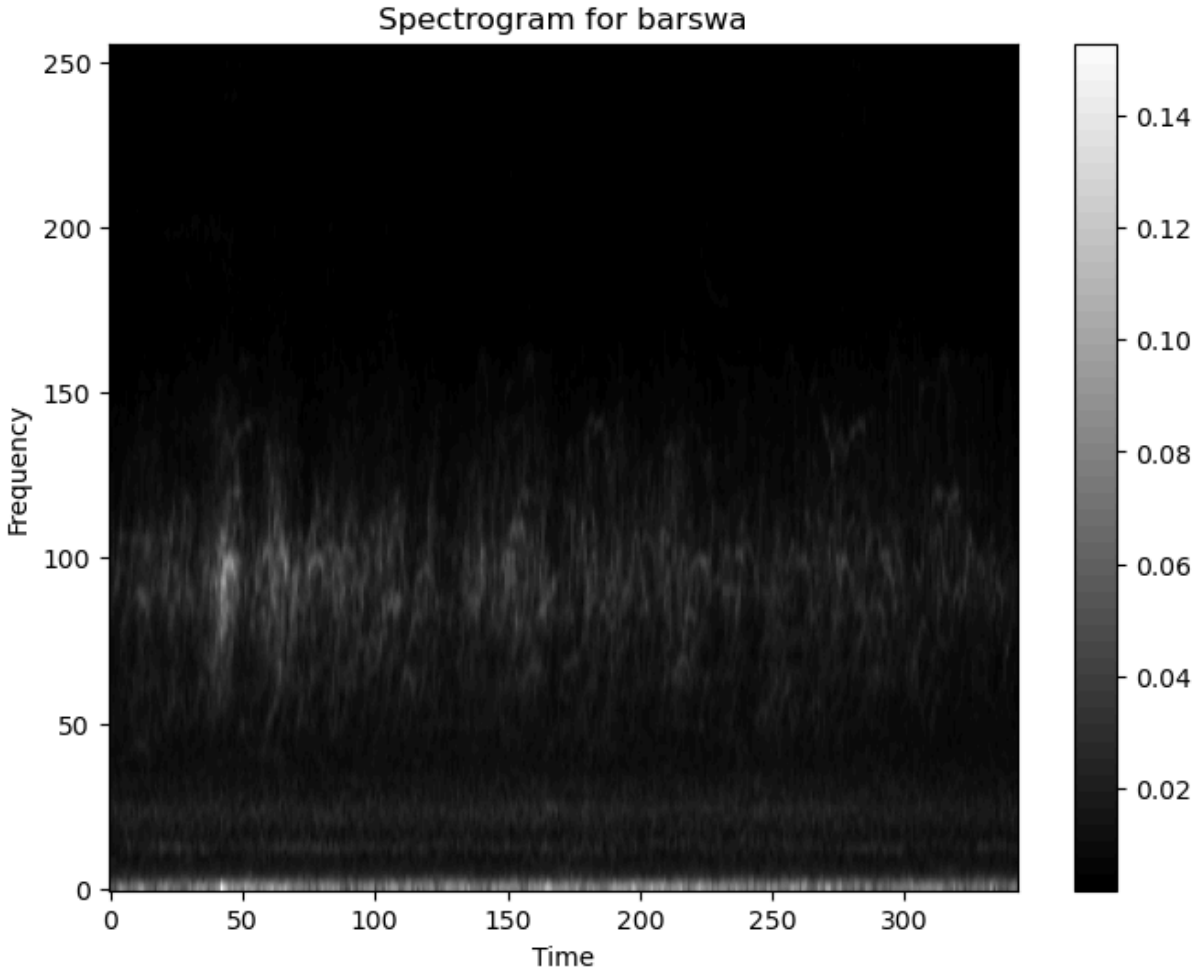
```
Root keys: ['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3',
'norfli', 'rewbla', 'stejay', 'wesmea', 'whcspa']

Item: amecro
  Dataset shape: (256, 343, 52)
  Dataset dtype: float64

Item: barswa
  Dataset shape: (256, 343, 55)
  Dataset dtype: float64

Item: bkcchi
  Dataset shape: (256, 343, 57)
  Dataset dtype: float64

Item: blujay
  Dataset shape: (256, 343, 50)
  Dataset dtype: float64

Item: daejun
  Dataset shape: (256, 343, 58)
  Dataset dtype: float64

Item: houfin
  Dataset shape: (256, 343, 44)
  Dataset dtype: float64

Item: mallar3
  Dataset shape: (256, 343, 36)
  Dataset dtype: float64

Item: norfli
  Dataset shape: (256, 343, 59)
  Dataset dtype: float64

Item: rewbla
  Dataset shape: (256, 343, 41)
  Dataset dtype: float64

Item: stejay
  Dataset shape: (256, 343, 40)
  Dataset dtype: float64

Item: wesmea
  Dataset shape: (256, 343, 36)
  Dataset dtype: float64

Item: whcspa
  Dataset shape: (256, 343, 51)
  Dataset dtype: float64
```

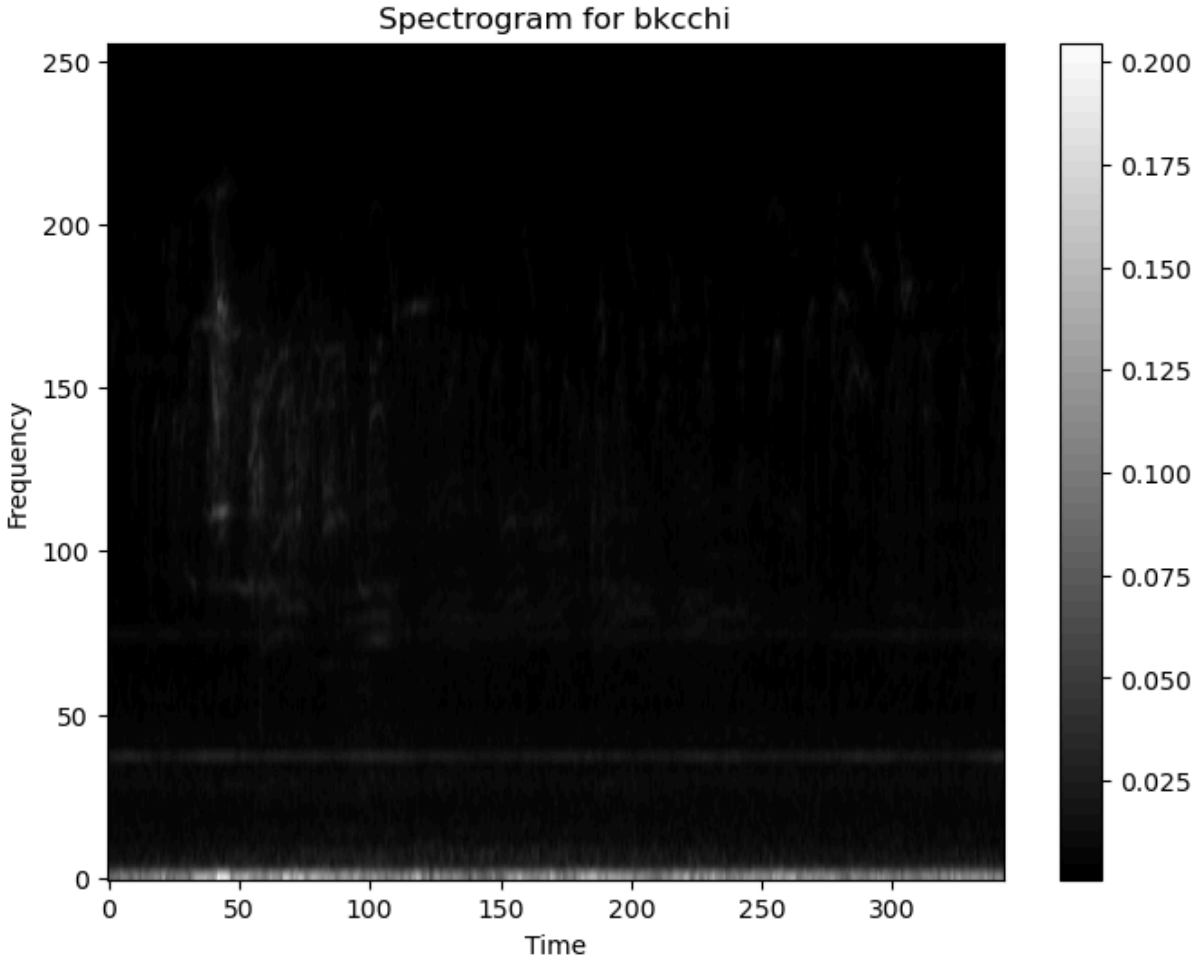
Exploration Data Analysis

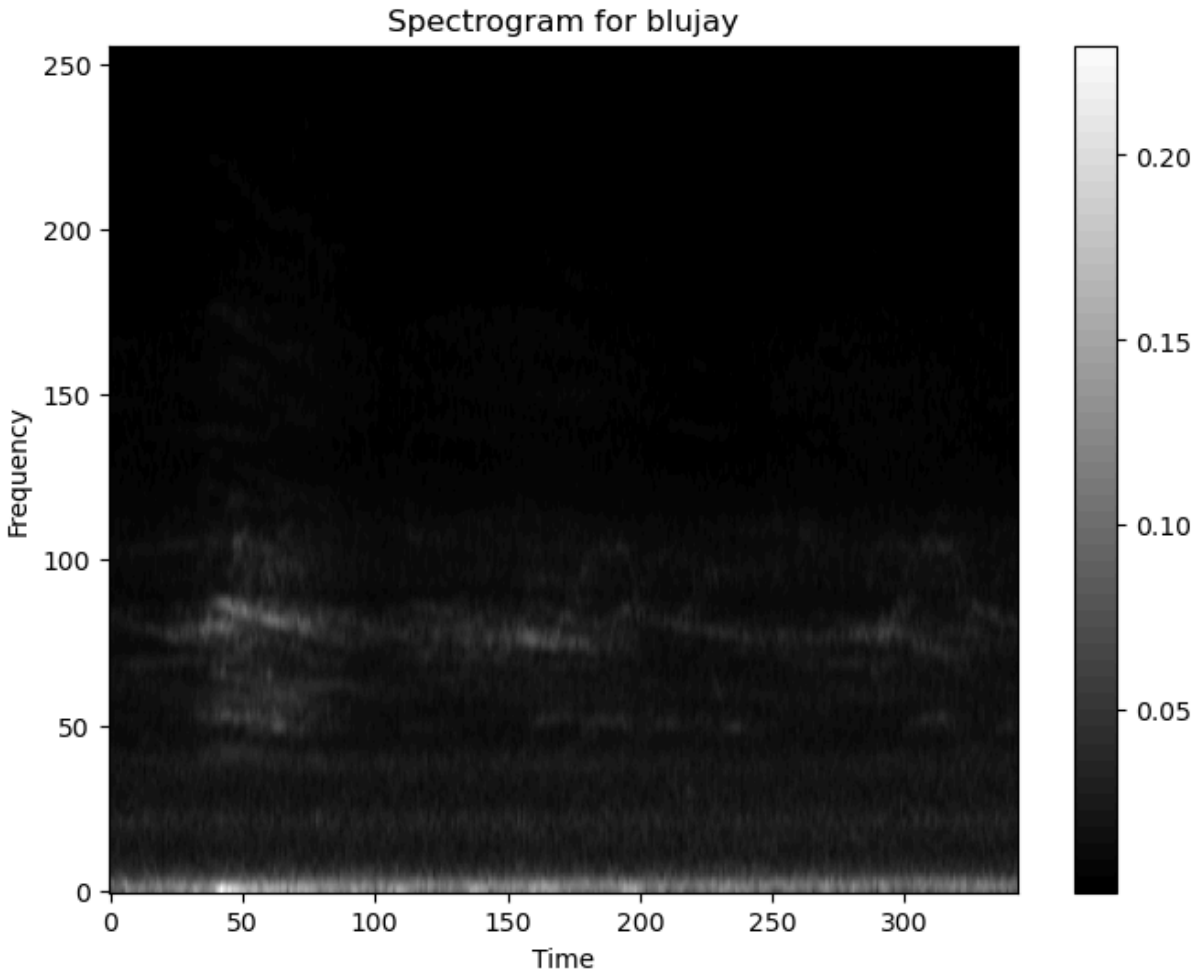
Plot the mean spectrogram for each bird species

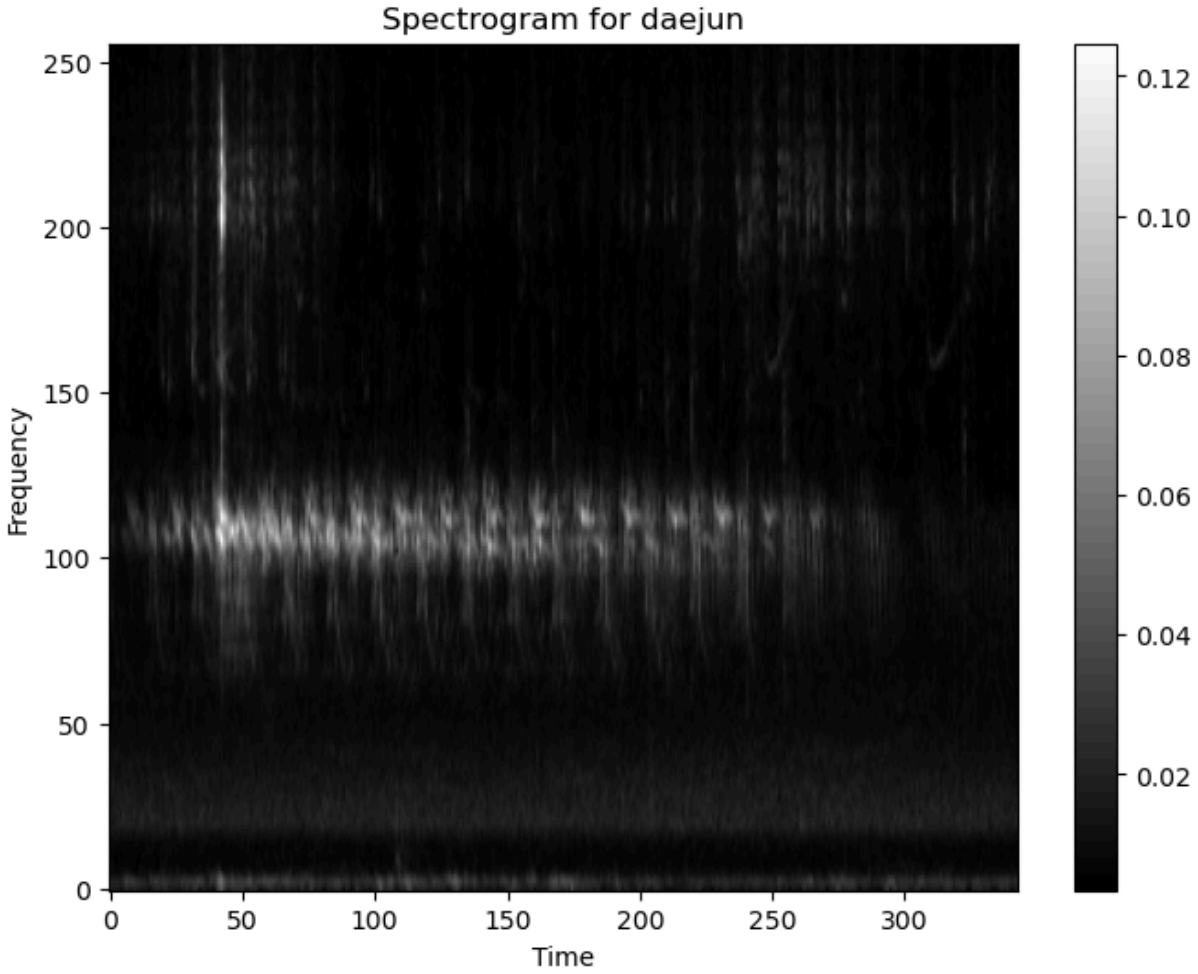
In [42]:
```
# Load the HDF5 file containing spectrogram data
file_path = 'Downloads/spectrograms.h5'
with h5py.File(file_path, 'r') as f:
    # Get the root keys (bird species)
    bird_species = list(f.keys())

    # Plot the mean spectrogram for each bird species
    for species in bird_species:
        spectrogram = np.mean(f[species][:], axis=-1)  # Collapse the channels by t
        plt.figure(figsize=(8, 6))
        plt.imshow(spectrogram, aspect='auto', origin='lower', cmap='gray')  # Whit
        plt.colorbar()
        plt.title(f'Spectrogram for {species}')
        plt.xlabel('Time')
        plt.ylabel('Frequency')
        plt.gca().set_facecolor('white')  # Set background to white
        plt.show()
```

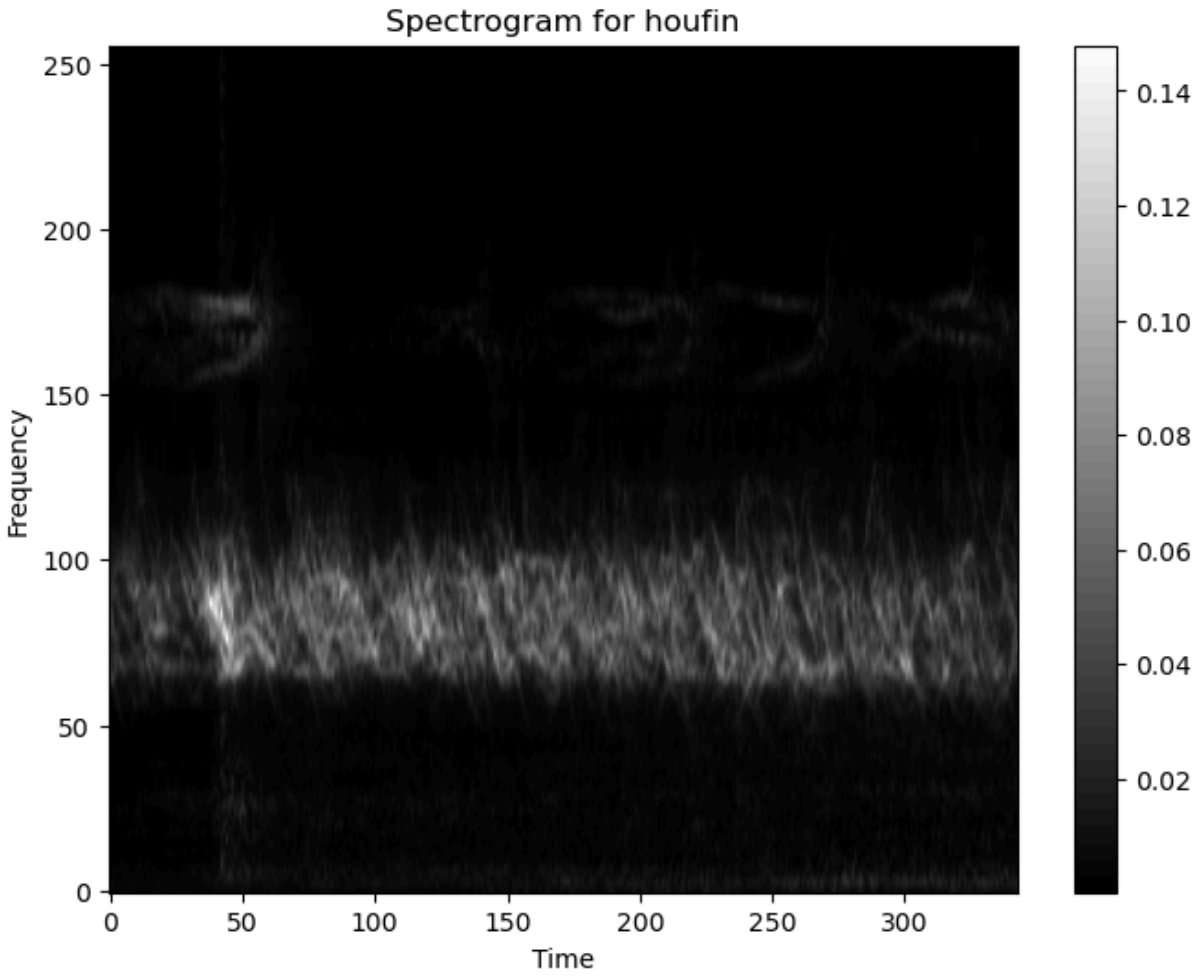


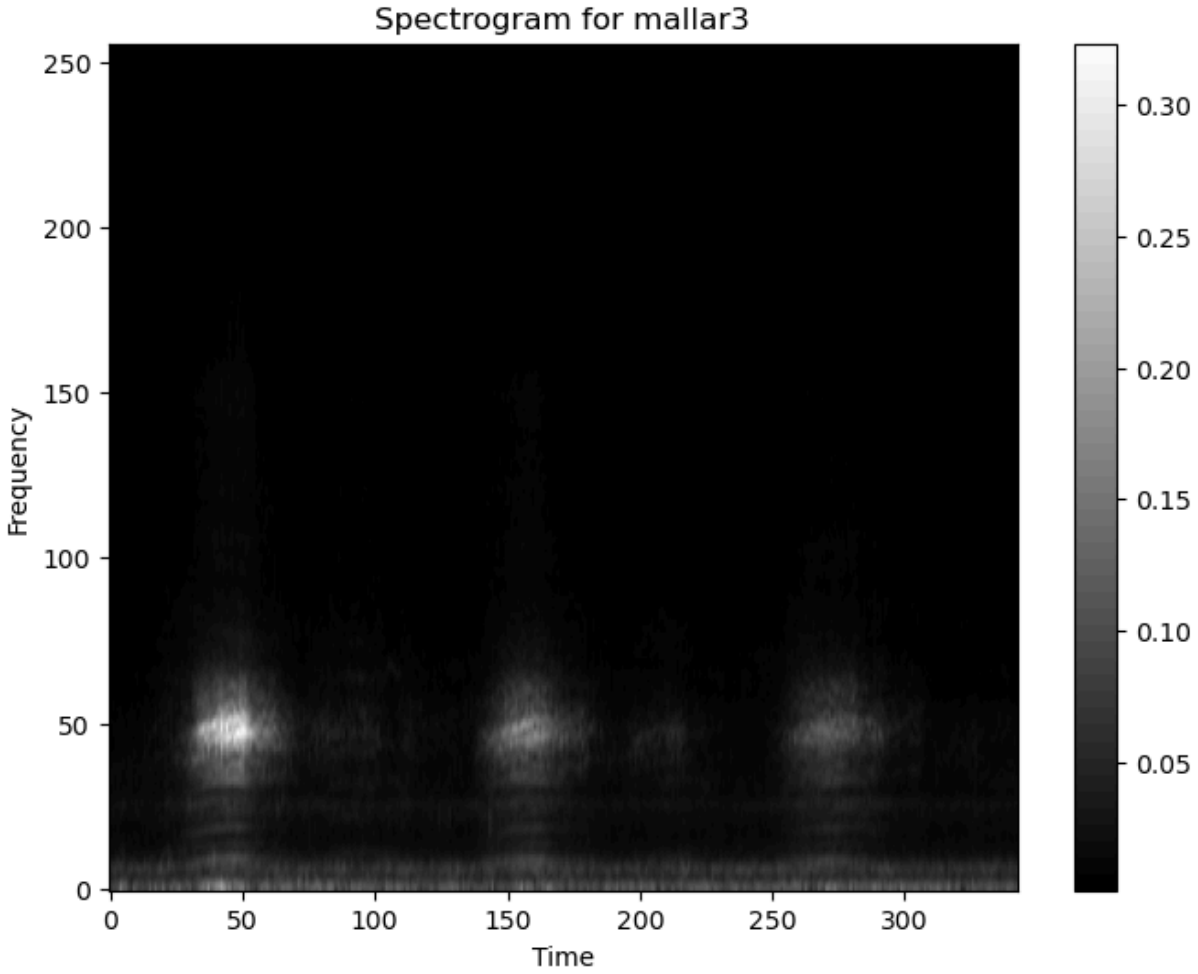

Spectrogram for barswa
Frequency
Time
0
50
100
150
200
250
300
0.14
0.12
0.10
0.08
0.06
0.04
0.02

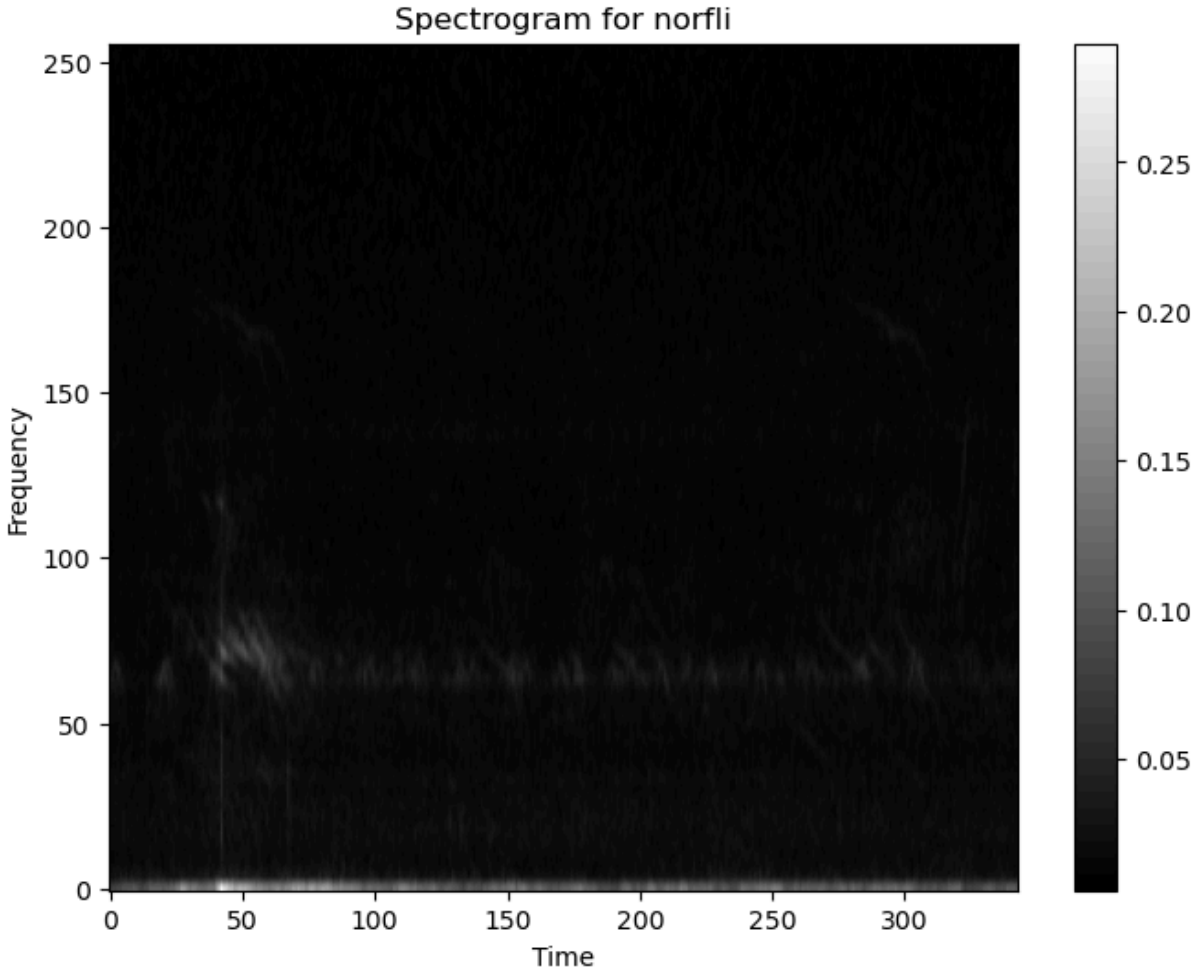

Spectrogram for bkcchi
Frequency
Time
0
50
100
150
200
250
300
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025

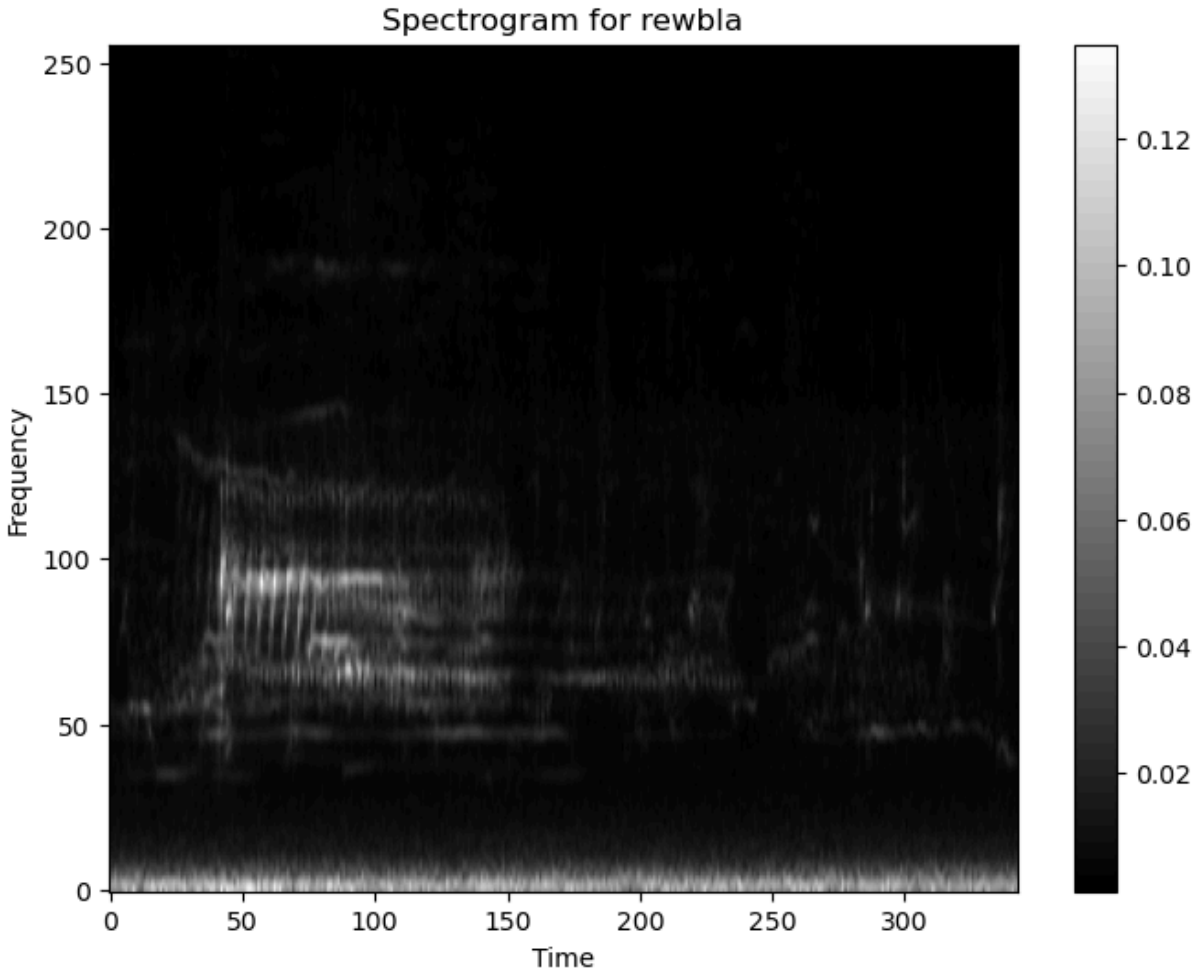

Spectrogram for blujay
Frequency
Time
0
50
100
150
200
250
300
0.20
0.15
0.10
0.05

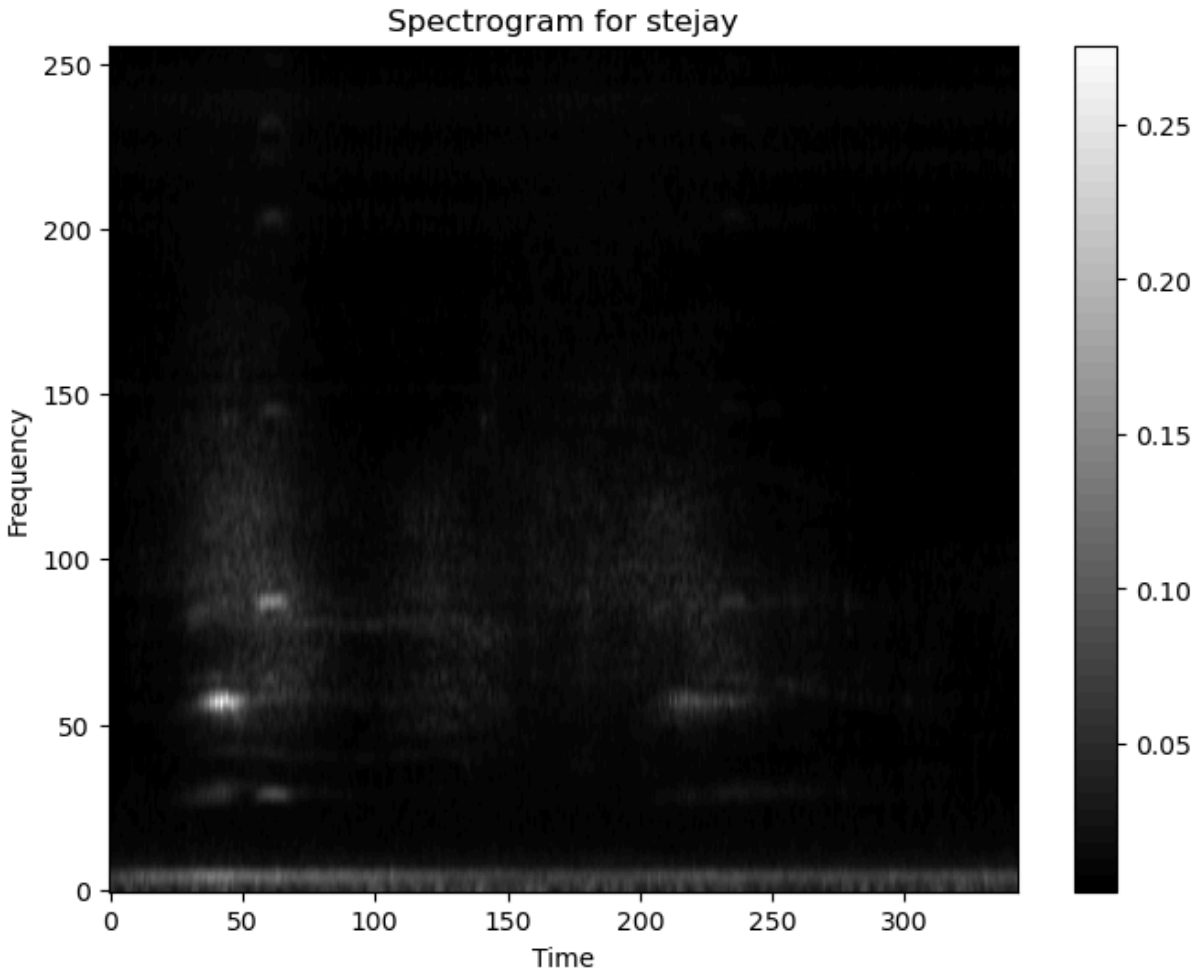

Spectrogram for daejun
Frequency
Time
0
50
100
150
200
250
300
0.12
0.10
0.08
0.06
0.04
0.02

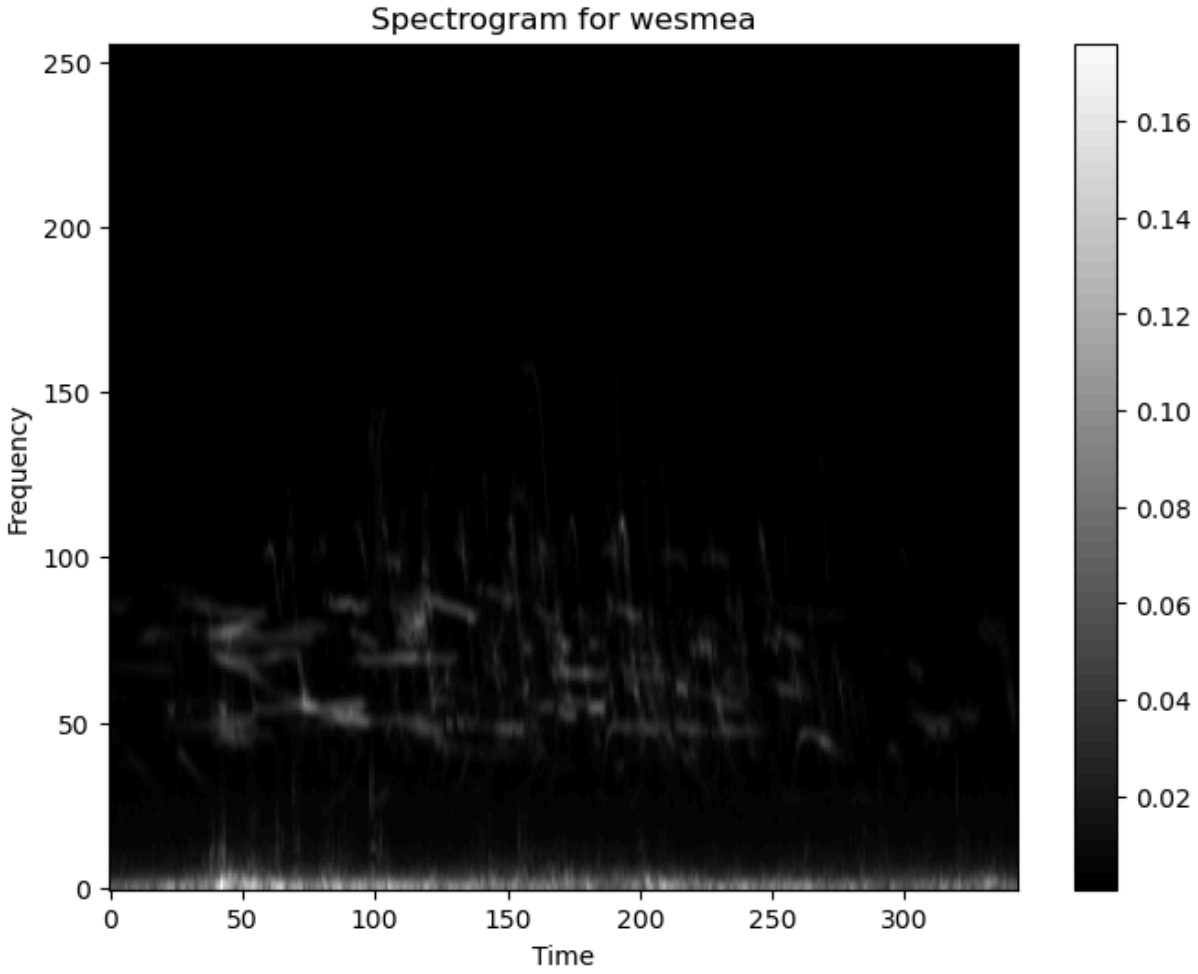

Spectrogram for houfin
Frequency
Time
0
50
100
150
200
250
300
0.14
0.12
0.10
0.08
0.06
0.04
0.02

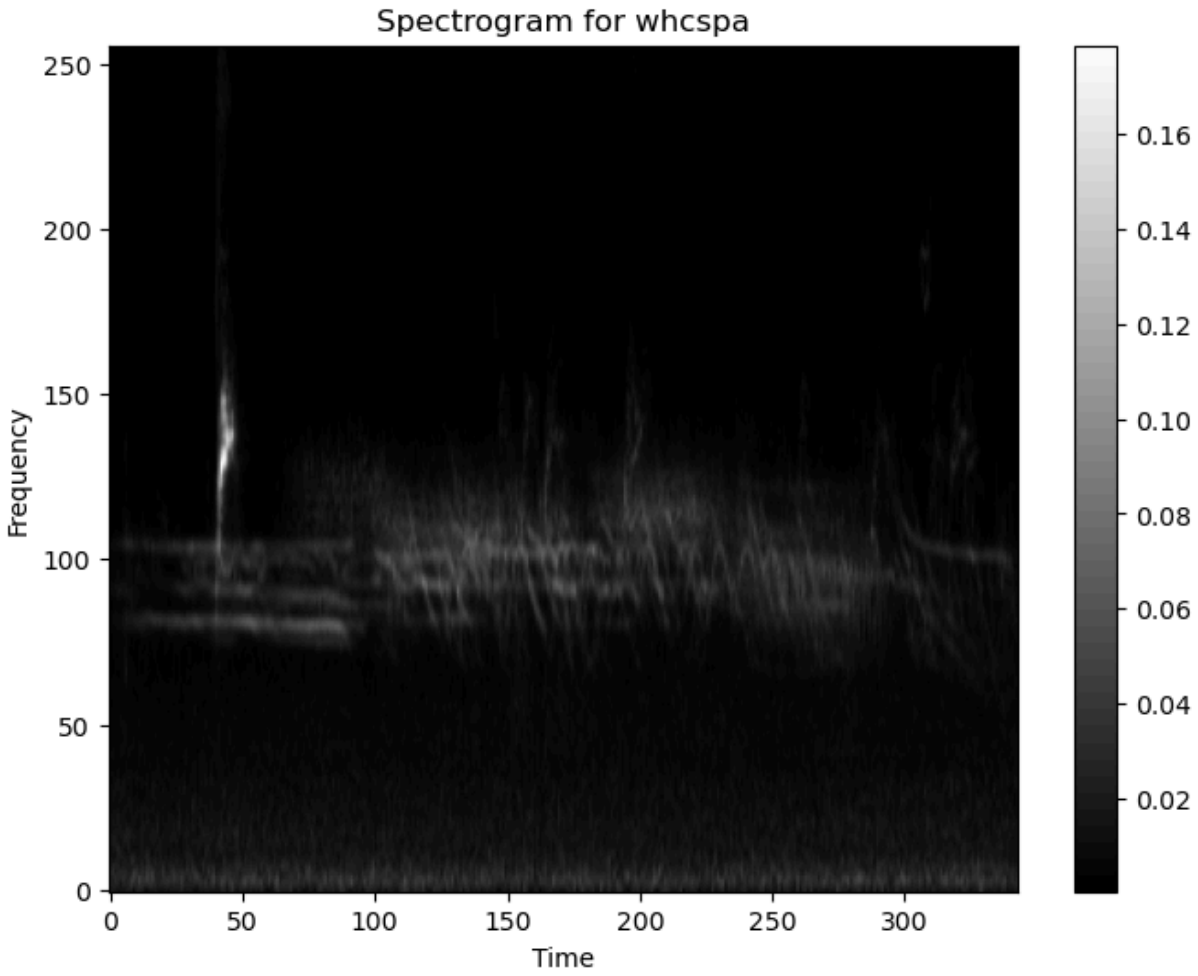

Spectrogram for mallar3
Frequency
Time
0
50
100
150
200
250
300
0.30
0.25
0.20
0.15
0.10
0.05


Spectrogram for norfli
Frequency
Time
0
50
100
150
200
250
300
0.25
0.20
0.15
0.10
0.05


Spectrogram for rewbla
Frequency
Time
0
50
100
150
200
250
300
0.12
0.10
0.08
0.06
0.04
0.02


Spectrogram for stejay
Frequency
Time
0
50
100
150
200
250
300
0.25
0.20
0.15
0.10
0.05


Spectrogram for wesmea
Frequency
Time
0
50
100
150
200
250
300
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02



Function to extract spectral features from a spectrogram, spectral entropy

In [43]:

```python
# Function to compute spectral entropy
def compute_spectral_entropy(spectrogram):
    # Compute power spectrum
    power_spectrum = np.abs(spectrogram)
    power_spectrum /= np.sum(power_spectrum, axis=0)  # Normalize
    # Compute spectral entropy
    spectral_entropy = entropy(power_spectrum, axis=0)

    return spectral_entropy

# Function to extract spectral features from a spectrogram
def extract_spectral_features(spectrogram):
    # Compute spectral features
    spectral_centroid = librosa.feature.spectral_centroid(S=spectrogram)
    spectral_bandwidth = librosa.feature.spectral_bandwidth(S=spectrogram)
    spectral_contrast = librosa.feature.spectral_contrast(S=spectrogram)
    spectral_rolloff = librosa.feature.spectral_rolloff(S=spectrogram)
    spectral_flatness = librosa.feature.spectral_flatness(S=spectrogram)
    spectral_flux = librosa.onset.onset_strength(S=spectrogram)
    spectral_entropy = compute_spectral_entropy(spectrogram)
```

```
    return spectral_centroid, spectral_bandwidth, spectral_contrast, spectral_rollo

# Load the HDF5 file containing spectrogram data
file_path = 'Downloads/spectrograms.h5'
with h5py.File(file_path, 'r') as f:
    # the root keys (bird species)
    bird_species = list(f.keys())

    # Iterate through each bird species
    for species in bird_species:
        print(f"Processing {species}...")

        # Initialize lists to store spectral features
        all_spectral_features = []

        # Get the group corresponding to the current bird species
        group = f[species]

        # Convert the dataset to a NumPy array
        spectrogram = np.array(group)

        # Extract spectral features
        spectral_features = extract_spectral_features(spectrogram)
        all_spectral_features.append(spectral_features)

        # Print or further process the extracted spectral features
        if all_spectral_features:
            print(f"Spectral features for {species}:")
            for idx, features in enumerate(all_spectral_features, start=1):

                for feature_name, feature_data in zip(['Spectral centroid', 'Spectr
                                                       'Spectral rolloff', 'Spectra
                                                       'Spectral entropy'], feature
                    print(f"{feature_name}: {feature_data.shape}")
```

```
Processing amecro...
Spectral features for amecro:
Spectral centroid: (256, 1, 52)
Spectral bandwidth: (256, 1, 52)
Spectral contrast: (256, 7, 52)
Spectral rolloff: (256, 1, 52)
Spectral flatness: (256, 1, 52)
Spectral flux: (256, 52)
Spectral entropy: (343, 52)
Processing barswa...
Spectral features for barswa:
Spectral centroid: (256, 1, 55)
Spectral bandwidth: (256, 1, 55)
Spectral contrast: (256, 7, 55)
Spectral rolloff: (256, 1, 55)
Spectral flatness: (256, 1, 55)
Spectral flux: (256, 55)
Spectral entropy: (343, 55)
Processing bkcchi...
Spectral features for bkcchi:
Spectral centroid: (256, 1, 57)
Spectral bandwidth: (256, 1, 57)
Spectral contrast: (256, 7, 57)
Spectral rolloff: (256, 1, 57)
Spectral flatness: (256, 1, 57)
Spectral flux: (256, 57)
Spectral entropy: (343, 57)
Processing blujay...
Spectral features for blujay:
Spectral centroid: (256, 1, 50)
Spectral bandwidth: (256, 1, 50)
Spectral contrast: (256, 7, 50)
Spectral rolloff: (256, 1, 50)
Spectral flatness: (256, 1, 50)
Spectral flux: (256, 50)
Spectral entropy: (343, 50)
Processing daejun...
Spectral features for daejun:
Spectral centroid: (256, 1, 58)
Spectral bandwidth: (256, 1, 58)
Spectral contrast: (256, 7, 58)
Spectral rolloff: (256, 1, 58)
Spectral flatness: (256, 1, 58)
Spectral flux: (256, 58)
Spectral entropy: (343, 58)
Processing houfin...
Spectral features for houfin:
Spectral centroid: (256, 1, 44)
Spectral bandwidth: (256, 1, 44)
Spectral contrast: (256, 7, 44)
Spectral rolloff: (256, 1, 44)
Spectral flatness: (256, 1, 44)
Spectral flux: (256, 44)
Spectral entropy: (343, 44)
Processing mallar3...
Spectral features for mallar3:
```

```
Spectral centroid: (256, 1, 36)
Spectral bandwidth: (256, 1, 36)
Spectral contrast: (256, 7, 36)
Spectral rolloff: (256, 1, 36)
Spectral flatness: (256, 1, 36)
Spectral flux: (256, 36)
Spectral entropy: (343, 36)
Processing norfli...
Spectral features for norfli:
Spectral centroid: (256, 1, 59)
Spectral bandwidth: (256, 1, 59)
Spectral contrast: (256, 7, 59)
Spectral rolloff: (256, 1, 59)
Spectral flatness: (256, 1, 59)
Spectral flux: (256, 59)
Spectral entropy: (343, 59)
Processing rewbla...
Spectral features for rewbla:
Spectral centroid: (256, 1, 41)
Spectral bandwidth: (256, 1, 41)
Spectral contrast: (256, 7, 41)
Spectral rolloff: (256, 1, 41)
Spectral flatness: (256, 1, 41)
Spectral flux: (256, 41)
Spectral entropy: (343, 41)
Processing stejay...
Spectral features for stejay:
Spectral centroid: (256, 1, 40)
Spectral bandwidth: (256, 1, 40)
Spectral contrast: (256, 7, 40)
Spectral rolloff: (256, 1, 40)
Spectral flatness: (256, 1, 40)
Spectral flux: (256, 40)
Spectral entropy: (343, 40)
Processing wesmea...
Spectral features for wesmea:
Spectral centroid: (256, 1, 36)
Spectral bandwidth: (256, 1, 36)
Spectral contrast: (256, 7, 36)
Spectral rolloff: (256, 1, 36)
Spectral flatness: (256, 1, 36)
Spectral flux: (256, 36)
Spectral entropy: (343, 36)
Processing whcspa...
Spectral features for whcspa:
Spectral centroid: (256, 1, 51)
Spectral bandwidth: (256, 1, 51)
Spectral contrast: (256, 7, 51)
Spectral rolloff: (256, 1, 51)
Spectral flatness: (256, 1, 51)
Spectral flux: (256, 51)
Spectral entropy: (343, 51)
```

BINARY CLASSIFICATION

amecro and barswa

Information: barswa: Spectral features for barswa: Spectral centroid: (256, 1, 55) Spectral bandwidth: (256, 1, 55) Spectral contrast: (256, 7, 55) Spectral rolloff: (256, 1, 55) Spectral flatness: (256, 1, 55) Spectral flux: (256, 55) Spectral entropy: (343, 55) amecro: Spectral features for amecro: Spectral centroid: (256, 1, 52) Spectral bandwidth: (256, 1, 52) Spectral contrast: (256, 7, 52) Spectral rolloff: (256, 1, 52) Spectral flatness: (256, 1, 52) Spectral flux: (256, 52) Spectral entropy: (343, 52)

CNN

In [44]:
```
# Load Data from HDF5 File
file_path = 'Downloads/spectrograms.h5'
with h5py.File(file_path, 'r') as f:
    amecro_data = np.array(f['amecro'])
    barswa_data = np.array(f['barswa'])

# Ensure consistency in dimensions
size_diff = barswa_data.shape[2] - amecro_data.shape[2]
if size_diff > 0:
    amecro_data = np.pad(amecro_data, ((0, 0), (0, 0), (0, size_diff)), mode='const
elif size_diff < 0:
    barswa_data = barswa_data[:, :, :amecro_data.shape[2]]

# Concatenate data for both classes
X = np.concatenate((amecro_data, barswa_data), axis=0)
y = np.concatenate((np.zeros(amecro_data.shape[0]), np.ones(barswa_data.shape[0])),

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Normalize the feature data
X_train_normalized = X_train / np.max(X_train)
X_test_normalized = X_test / np.max(X_train)

# Reshape the input data to have the correct shape
X_train_reshaped = X_train_normalized.reshape(-1, X_train_normalized.shape[1], X_tr
X_test_reshaped = X_test_normalized.reshape(-1, X_test_normalized.shape[1], X_test_

# Reduced Complexity Model Building
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=X_train_reshaped.shape[1:]),
    MaxPooling2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Model Compilation with the same optimizer and loss function
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Model Training with the same settings
history = model.fit(X_train_reshaped, y_train, batch_size=16, epochs=20, validation
```

```
Epoch 1/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 12s 139ms/step - accuracy: 0.7654 - loss: 0.5786 - val_ac
curacy: 0.9870 - val_loss: 0.2018
Epoch 2/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 114ms/step - accuracy: 0.9546 - loss: 0.1745 - val_acc
uracy: 1.0000 - val_loss: 0.0174
Epoch 3/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 109ms/step - accuracy: 0.9897 - loss: 0.0929 - val_acc
uracy: 1.0000 - val_loss: 0.0347
Epoch 4/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 114ms/step - accuracy: 0.9978 - loss: 0.0449 - val_acc
uracy: 1.0000 - val_loss: 0.0042
Epoch 5/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 109ms/step - accuracy: 1.0000 - loss: 0.0083 - val_acc
uracy: 1.0000 - val_loss: 0.0032
Epoch 6/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - accuracy: 0.9991 - loss: 0.0056 - val_acc
uracy: 1.0000 - val_loss: 4.5733e-04
Epoch 7/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 114ms/step - accuracy: 0.9906 - loss: 0.0412 - val_acc
uracy: 0.9935 - val_loss: 0.0170
Epoch 8/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 114ms/step - accuracy: 1.0000 - loss: 0.0162 - val_acc
uracy: 1.0000 - val_loss: 2.3275e-04
Epoch 9/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 114ms/step - accuracy: 0.9941 - loss: 0.0081 - val_acc
uracy: 1.0000 - val_loss: 0.0012
Epoch 10/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 115ms/step - accuracy: 0.9995 - loss: 0.0037 - val_acc
uracy: 1.0000 - val_loss: 3.5917e-04
Epoch 11/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 110ms/step - accuracy: 1.0000 - loss: 0.0060 - val_acc
uracy: 1.0000 - val_loss: 5.6458e-04
Epoch 12/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - accuracy: 1.0000 - loss: 8.2566e-04 - val
_accuracy: 1.0000 - val_loss: 6.2765e-05
Epoch 13/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 117ms/step - accuracy: 1.0000 - loss: 3.8125e-04 - val
_accuracy: 1.0000 - val_loss: 2.0715e-05
Epoch 14/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 111ms/step - accuracy: 1.0000 - loss: 4.2219e-04 - val
_accuracy: 1.0000 - val_loss: 2.3878e-05
Epoch 15/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 111ms/step - accuracy: 1.0000 - loss: 5.1368e-04 - val
_accuracy: 1.0000 - val_loss: 2.2937e-05
Epoch 16/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - accuracy: 1.0000 - loss: 2.2414e-04 - val
_accuracy: 1.0000 - val_loss: 1.3396e-05
Epoch 17/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 115ms/step - accuracy: 1.0000 - loss: 1.4922e-04 - val
_accuracy: 1.0000 - val_loss: 9.2205e-06
Epoch 18/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 112ms/step - accuracy: 1.0000 - loss: 4.7289e-05 - val
_accuracy: 1.0000 - val_loss: 7.6369e-06
Epoch 19/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 115ms/step - accuracy: 1.0000 - loss: 1.2306e-04 - val
```

```
_accuracy: 1.0000 - val_loss: 6.4831e-06
Epoch 20/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 116ms/step - accuracy: 1.0000 - loss: 1.4681e-04 - val
_accuracy: 1.0000 - val_loss: 5.2859e-06
```

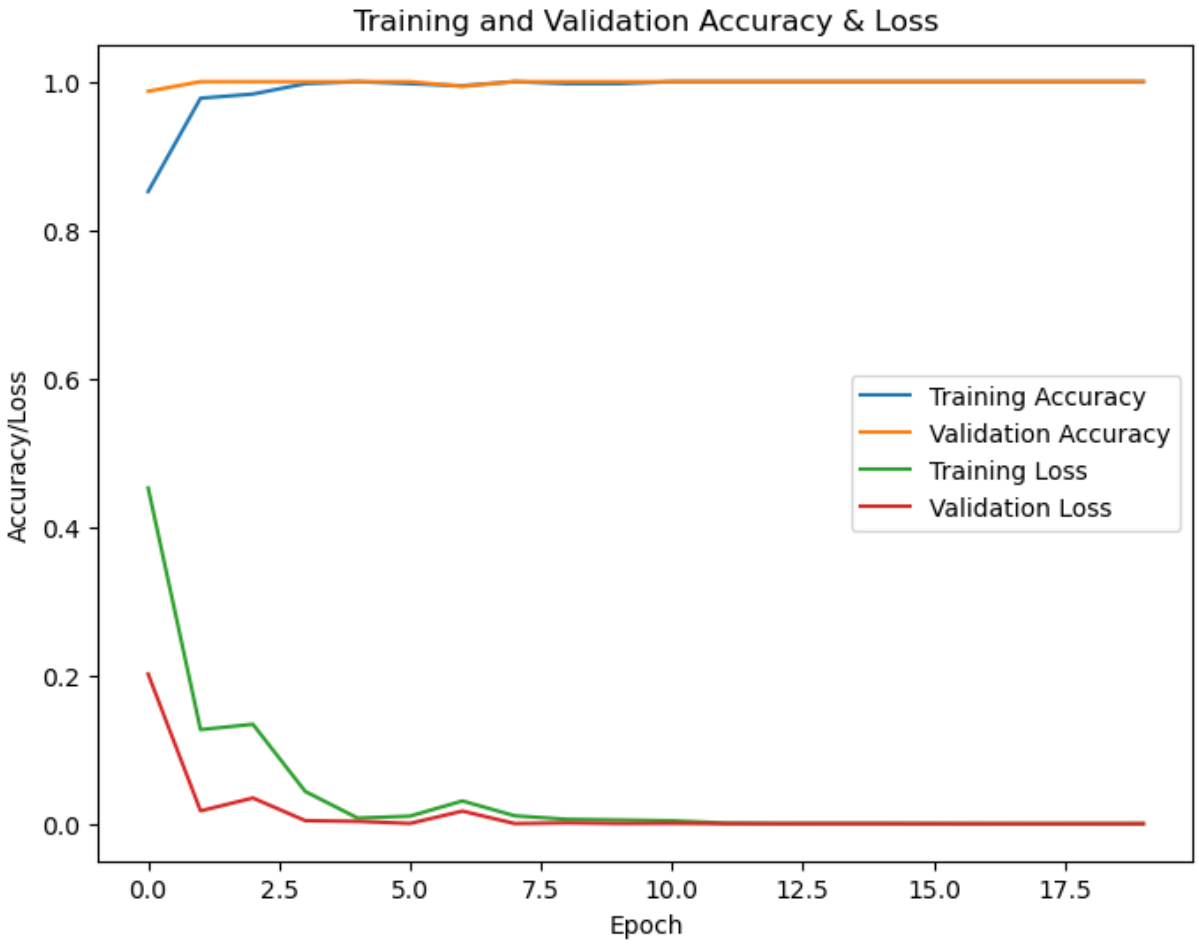
In [45]:
```python
# Get training accuracy
train_loss, train_accuracy = model.evaluate(X_train_reshaped, y_train)
print(f'Train Loss: {train_loss}, Train accuracy: {train_accuracy}')
print("Training Accuracy:", train_accuracy)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 1.0000 - loss: 3.8769e-06
Train Loss: 4.321396772866137e-06, Train accuracy: 1.0
Training Accuracy: 1.0
```

In [46]:
```python
# Model Evaluation
test_loss, test_accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_accuracy}')
print("Testing Accuracy:", test_accuracy)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 1.0000 - loss: 4.4444e-06
Test loss: 5.2858649723930284e-06, Test accuracy: 1.0
Testing Accuracy: 1.0
```

In [47]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy/Loss')
plt.title('Training and Validation Accuracy & Loss')
plt.legend()
plt.show()
```



Model 2 : With additional layers and additional dropout but with same batch size =16

In [48]:

```
# Load Data from HDF5 File
file_path = 'Downloads/spectrograms.h5'
with h5py.File(file_path, 'r') as f:
    amecro_data = np.array(f['amecro'])
    barswa_data = np.array(f['barswa'])

# Ensure consistency in dimensions
size_diff = barswa_data.shape[2] - amecro_data.shape[2]
if size_diff > 0:
    amecro_data = np.pad(amecro_data, ((0, 0), (0, 0), (0, size_diff)), mode='const
elif size_diff < 0:
    barswa_data = barswa_data[:, :, :amecro_data.shape[2]]

# Concatenate data for both classes
X = np.concatenate((amecro_data, barswa_data), axis=0)
y = np.concatenate((np.zeros(amecro_data.shape[0]), np.ones(barswa_data.shape[0])),

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Normalize the feature data
X_train_normalized = X_train / np.max(X_train)
X_test_normalized = X_test / np.max(X_train)
```

```
# Reshape the input data to have the correct shape
X_train_reshaped = X_train_normalized.reshape(-1, X_train_normalized.shape[1], X_tr
X_test_reshaped = X_test_normalized.reshape(-1, X_test_normalized.shape[1], X_test_

# Model Building
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=X_train_reshaped.shape[1:]),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),  # Adding Dropout
    Dense(128, activation='relu'),
    Dropout(0.3),  # Adding Dropout
    Dense(64, activation='relu'),  # Additional hidden layer
    Dense(1, activation='sigmoid')
])

# Model Compilation
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Model Training
history = model.fit(X_train_reshaped, y_train, batch_size= 16, epochs=20, validatio
```

```
Epoch 1/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 15s 367ms/step - accuracy: 0.6966 - loss: 0.5582 - val_ac
curacy: 0.9935 - val_loss: 0.1147
Epoch 2/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 328ms/step - accuracy: 0.9786 - loss: 0.1396 - val_acc
uracy: 0.9935 - val_loss: 0.0676
Epoch 3/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 334ms/step - accuracy: 0.9954 - loss: 0.0590 - val_acc
uracy: 1.0000 - val_loss: 0.0043
Epoch 4/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 7s 325ms/step - accuracy: 0.9972 - loss: 0.0117 - val_acc
uracy: 1.0000 - val_loss: 0.0011
Epoch 5/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 335ms/step - accuracy: 1.0000 - loss: 0.0034 - val_acc
uracy: 1.0000 - val_loss: 1.0009e-04
Epoch 6/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 328ms/step - accuracy: 1.0000 - loss: 8.7362e-04 - val
_accuracy: 1.0000 - val_loss: 4.7554e-05
Epoch 7/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 328ms/step - accuracy: 1.0000 - loss: 0.0024 - val_acc
uracy: 1.0000 - val_loss: 3.7197e-04
Epoch 8/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 326ms/step - accuracy: 0.9984 - loss: 0.0534 - val_acc
uracy: 1.0000 - val_loss: 0.0052
Epoch 9/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 326ms/step - accuracy: 0.9956 - loss: 0.0130 - val_acc
uracy: 1.0000 - val_loss: 6.9600e-04
Epoch 10/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 326ms/step - accuracy: 1.0000 - loss: 0.0029 - val_acc
uracy: 1.0000 - val_loss: 9.4027e-05
Epoch 11/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 329ms/step - accuracy: 0.9972 - loss: 0.0286 - val_acc
uracy: 1.0000 - val_loss: 3.4514e-04
Epoch 12/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 329ms/step - accuracy: 1.0000 - loss: 0.0023 - val_acc
uracy: 1.0000 - val_loss: 3.8178e-05
Epoch 13/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 329ms/step - accuracy: 1.0000 - loss: 0.0030 - val_acc
uracy: 1.0000 - val_loss: 8.6929e-05
Epoch 14/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 326ms/step - accuracy: 1.0000 - loss: 5.3592e-04 - val
_accuracy: 1.0000 - val_loss: 5.5907e-06
Epoch 15/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 328ms/step - accuracy: 1.0000 - loss: 2.5447e-04 - val
_accuracy: 1.0000 - val_loss: 2.9747e-06
Epoch 16/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 329ms/step - accuracy: 1.0000 - loss: 1.2889e-04 - val
_accuracy: 1.0000 - val_loss: 3.3748e-06
Epoch 17/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 330ms/step - accuracy: 1.0000 - loss: 5.1224e-04 - val
_accuracy: 1.0000 - val_loss: 2.3630e-06
Epoch 18/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 335ms/step - accuracy: 1.0000 - loss: 1.4248e-04 - val
_accuracy: 1.0000 - val_loss: 1.1648e-06
Epoch 19/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 328ms/step - accuracy: 1.0000 - loss: 1.5524e-04 - val
```

```
_accuracy: 1.0000 - val_loss: 9.2762e-07
Epoch 20/20
23/23 ━━━━━━━━━━━━━━━━━━━━ 8s 325ms/step - accuracy: 1.0000 - loss: 3.2333e-04 - val
_accuracy: 1.0000 - val_loss: 6.3029e-07
```

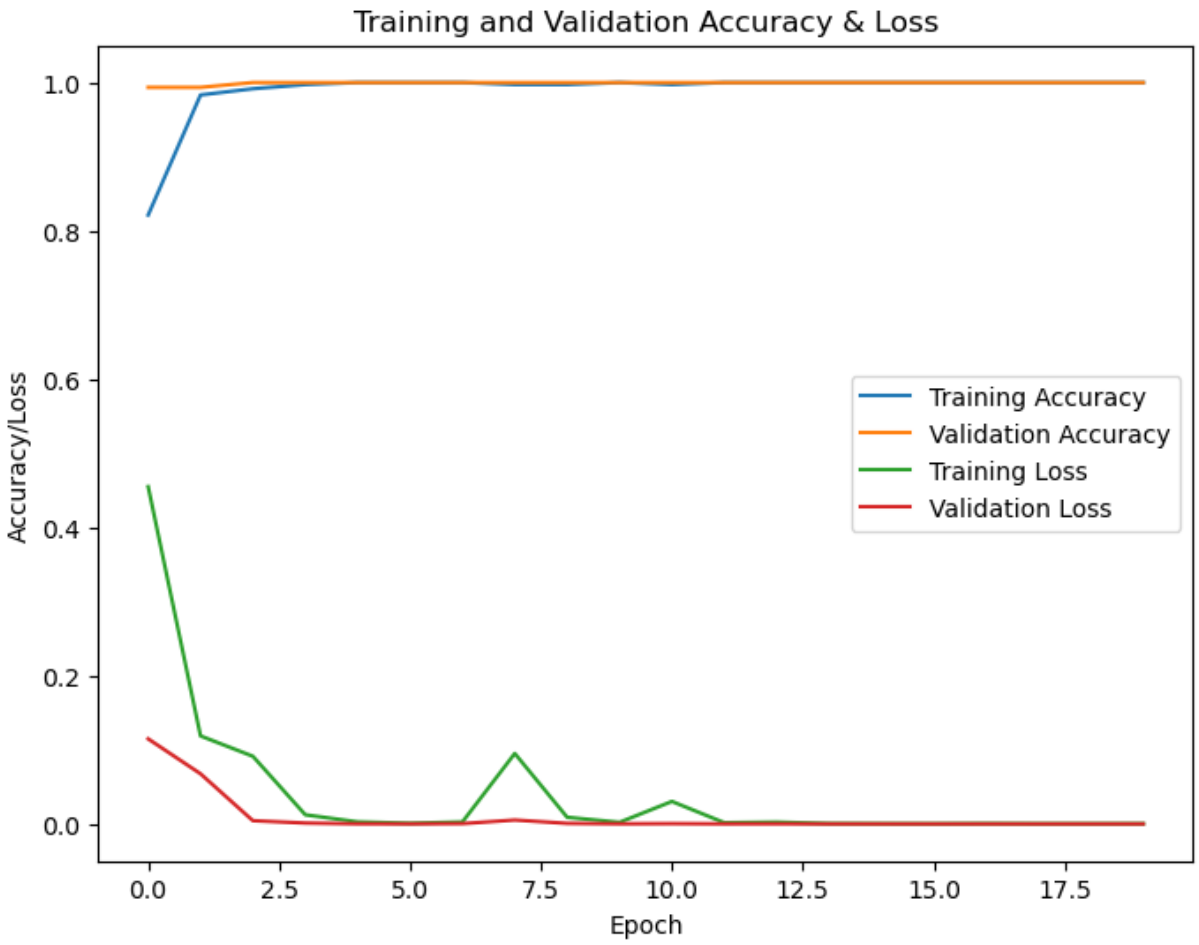
In [49]:
```python
# Get training accuracy
train_loss, train_accuracy = model.evaluate(X_train_reshaped, y_train)
print(f'Train Loss: {train_loss}, Train accuracy: {train_accuracy}')
print("Training Accuracy:", train_accuracy)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 1.0000 - loss: 3.6615e-07
Train Loss: 4.265288566784875e-07, Train accuracy: 1.0
Training Accuracy: 1.0
```

In [50]:
```python
# Model Evaluation
test_loss, test_accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_accuracy}')
print("Testing Accuracy:", test_accuracy)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 99ms/step - accuracy: 1.0000 - loss: 5.2612e-07
Test loss: 6.302915380729246e-07, Test accuracy: 1.0
Testing Accuracy: 1.0
```

In [51]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy/Loss')
plt.title('Training and Validation Accuracy & Loss')
plt.legend()
plt.show()
```



In [ ]:
```
#### Trying with Batch size = 32
```

In [54]:
```
# Load Data from HDF5 File
file_path = 'Downloads/spectrograms.h5'
with h5py.File(file_path, 'r') as f:
    amecro_data = np.array(f['amecro'])
    barswa_data = np.array(f['barswa'])

# Ensure consistency in dimensions
size_diff = barswa_data.shape[2] - amecro_data.shape[2]
if size_diff > 0:
    amecro_data = np.pad(amecro_data, ((0, 0), (0, 0), (0, size_diff)), mode='const
elif size_diff < 0:
    barswa_data = barswa_data[:, :, :amecro_data.shape[2]]

# Concatenate data for both classes
X = np.concatenate((amecro_data, barswa_data), axis=0)
y = np.concatenate((np.zeros(amecro_data.shape[0]), np.ones(barswa_data.shape[0])),

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Normalize the feature data
X_train_normalized = X_train / np.max(X_train)
X_test_normalized = X_test / np.max(X_train)
```

```
# Reshape the input data to have the correct shape
X_train_reshaped = X_train_normalized.reshape(-1, X_train_normalized.shape[1], X_tr
X_test_reshaped = X_test_normalized.reshape(-1, X_test_normalized.shape[1], X_test_

# Reduced Complexity Model Building
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=X_train_reshaped.shape[1:]),
    MaxPooling2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Model Compilation with the same optimizer and loss function
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Model Training with the same settings
history = model.fit(X_train_reshaped, y_train, batch_size=32, epochs=20, validation
```

```
Epoch 1/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 8s 238ms/step - accuracy: 0.7216 - loss: 0.5740 - val_acc
uracy: 0.9870 - val_loss: 0.2271
Epoch 2/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 163ms/step - accuracy: 0.9652 - loss: 0.1513 - val_acc
uracy: 1.0000 - val_loss: 0.0308
Epoch 3/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 159ms/step - accuracy: 0.9933 - loss: 0.0569 - val_acc
uracy: 1.0000 - val_loss: 0.0052
Epoch 4/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 153ms/step - accuracy: 0.9877 - loss: 0.0374 - val_acc
uracy: 1.0000 - val_loss: 0.0030
Epoch 5/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 153ms/step - accuracy: 0.9861 - loss: 0.0364 - val_acc
uracy: 0.9935 - val_loss: 0.0525
Epoch 6/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 160ms/step - accuracy: 0.9934 - loss: 0.0324 - val_acc
uracy: 0.9935 - val_loss: 0.0274
Epoch 7/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 151ms/step - accuracy: 0.9793 - loss: 0.1355 - val_acc
uracy: 1.0000 - val_loss: 0.0046
Epoch 8/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 156ms/step - accuracy: 0.9973 - loss: 0.0221 - val_acc
uracy: 1.0000 - val_loss: 0.0014
Epoch 9/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 156ms/step - accuracy: 1.0000 - loss: 0.0028 - val_acc
uracy: 1.0000 - val_loss: 6.5933e-04
Epoch 10/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 151ms/step - accuracy: 0.9988 - loss: 0.0049 - val_acc
uracy: 1.0000 - val_loss: 8.8077e-04
Epoch 11/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 153ms/step - accuracy: 0.9870 - loss: 0.0341 - val_acc
uracy: 1.0000 - val_loss: 0.0032
Epoch 12/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 151ms/step - accuracy: 1.0000 - loss: 0.0074 - val_acc
uracy: 1.0000 - val_loss: 0.0015
Epoch 13/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 164ms/step - accuracy: 0.9991 - loss: 0.0050 - val_acc
uracy: 1.0000 - val_loss: 4.3141e-04
Epoch 14/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 155ms/step - accuracy: 1.0000 - loss: 0.0026 - val_acc
uracy: 1.0000 - val_loss: 4.0524e-04
Epoch 15/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 155ms/step - accuracy: 0.9948 - loss: 0.0863 - val_acc
uracy: 1.0000 - val_loss: 5.9771e-04
Epoch 16/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 159ms/step - accuracy: 0.9985 - loss: 0.0093 - val_acc
uracy: 1.0000 - val_loss: 0.0016
Epoch 17/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 156ms/step - accuracy: 1.0000 - loss: 0.0041 - val_acc
uracy: 1.0000 - val_loss: 4.3741e-04
Epoch 18/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 152ms/step - accuracy: 1.0000 - loss: 0.0019 - val_acc
uracy: 1.0000 - val_loss: 1.3978e-04
Epoch 19/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 159ms/step - accuracy: 1.0000 - loss: 9.0734e-04 - val
```

```
_accuracy: 1.0000 - val_loss: 7.0028e-05
Epoch 20/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 173ms/step - accuracy: 1.0000 - loss: 6.7829e-04 - val
_accuracy: 1.0000 - val_loss: 4.9956e-05
```

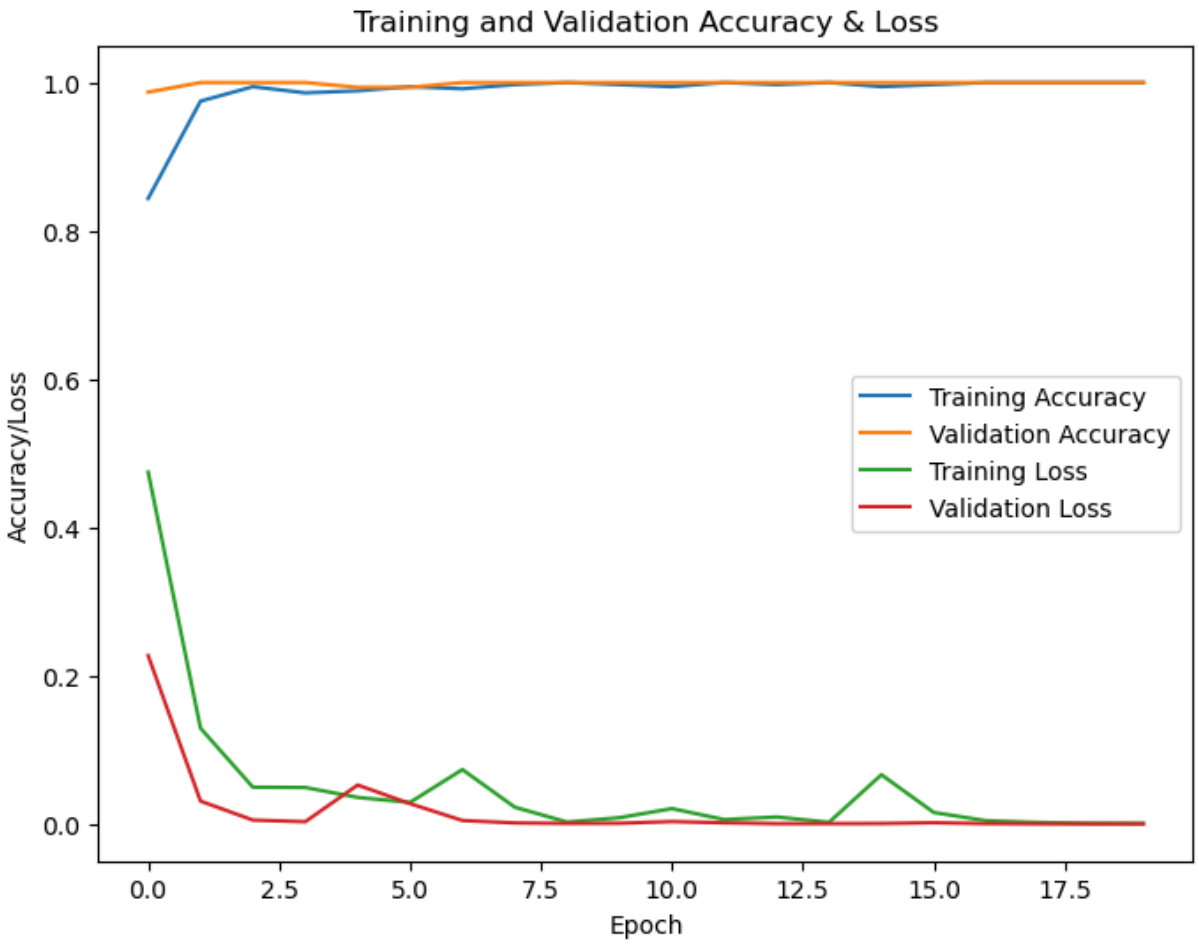
In [55]:
```python
# Get training accuracy
train_loss, train_accuracy = model.evaluate(X_train_reshaped, y_train)
print(f'Train Loss: {train_loss}, Train accuracy: {train_accuracy}')
print("Training Accuracy:", train_accuracy)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 54ms/step - accuracy: 1.0000 - loss: 3.0705e-05
Train Loss: 3.44696018146351e-05, Train accuracy: 1.0
Training Accuracy: 1.0
```

In [56]:
```python
# Model Evaluation
test_loss, test_accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_accuracy}')
print("Testing Accuracy:", test_accuracy)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 1.0000 - loss: 4.2999e-05
Test loss: 4.9955822760239244e-05, Test accuracy: 1.0
Testing Accuracy: 1.0
```

In [57]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy/Loss')
plt.title('Training and Validation Accuracy & Loss')
plt.legend()
plt.show()
```



Adding layers and dropout with a batch size of 32

In [58]:

```
# Load Data from HDF5 File
file_path = 'Downloads/spectrograms.h5'
with h5py.File(file_path, 'r') as f:
    amecro_data = np.array(f['amecro'])
    barswa_data = np.array(f['barswa'])

# Ensure consistency in dimensions
size_diff = barswa_data.shape[2] - amecro_data.shape[2]
if size_diff > 0:
    amecro_data = np.pad(amecro_data, ((0, 0), (0, 0), (0, size_diff)), mode='const
elif size_diff < 0:
    barswa_data = barswa_data[:, :, :amecro_data.shape[2]]

# Concatenate data for both classes
X = np.concatenate((amecro_data, barswa_data), axis=0)
y = np.concatenate((np.zeros(amecro_data.shape[0]), np.ones(barswa_data.shape[0])),

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Normalize the feature data
X_train_normalized = X_train / np.max(X_train)
X_test_normalized = X_test / np.max(X_train)
```

```
# Reshape the input data to have the correct shape
X_train_reshaped = X_train_normalized.reshape(-1, X_train_normalized.shape[1], X_tr
X_test_reshaped = X_test_normalized.reshape(-1, X_test_normalized.shape[1], X_test_

# Model Building
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=X_train_reshaped.shape[1:]),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),  # Adding Dropout
    Dense(128, activation='relu'),
    Dropout(0.3),  # Adding Dropout
    Dense(64, activation='relu'),  # Additional hidden layer
    Dense(1, activation='sigmoid')
])

# Model Compilation
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Model Training
history = model.fit(X_train_reshaped, y_train, batch_size= 32, epochs=20, validatio
```

```
Epoch 1/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 14s 662ms/step - accuracy: 0.6452 - loss: 0.6173 - val_ac
curacy: 0.9870 - val_loss: 0.2086
Epoch 2/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 7s 563ms/step - accuracy: 0.9484 - loss: 0.2179 - val_acc
uracy: 0.9870 - val_loss: 0.2667
Epoch 3/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 7s 558ms/step - accuracy: 0.9910 - loss: 0.1201 - val_acc
uracy: 0.9870 - val_loss: 0.1290
Epoch 4/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 450ms/step - accuracy: 0.9858 - loss: 0.2374 - val_acc
uracy: 1.0000 - val_loss: 0.0185
Epoch 5/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 442ms/step - accuracy: 1.0000 - loss: 0.0188 - val_acc
uracy: 1.0000 - val_loss: 0.0037
Epoch 6/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 445ms/step - accuracy: 1.0000 - loss: 0.0071 - val_acc
uracy: 1.0000 - val_loss: 5.3243e-04
Epoch 7/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 443ms/step - accuracy: 1.0000 - loss: 0.0045 - val_acc
uracy: 1.0000 - val_loss: 3.4434e-04
Epoch 8/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 457ms/step - accuracy: 1.0000 - loss: 9.5026e-04 - val
_accuracy: 1.0000 - val_loss: 2.2441e-04
Epoch 9/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 450ms/step - accuracy: 1.0000 - loss: 9.9923e-04 - val
_accuracy: 1.0000 - val_loss: 1.3287e-04
Epoch 10/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 445ms/step - accuracy: 1.0000 - loss: 9.2808e-04 - val
_accuracy: 1.0000 - val_loss: 6.5126e-05
Epoch 11/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 439ms/step - accuracy: 1.0000 - loss: 7.5582e-04 - val
_accuracy: 1.0000 - val_loss: 4.0700e-05
Epoch 12/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 453ms/step - accuracy: 1.0000 - loss: 3.0971e-04 - val
_accuracy: 1.0000 - val_loss: 3.9383e-05
Epoch 13/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 444ms/step - accuracy: 1.0000 - loss: 9.3662e-04 - val
_accuracy: 1.0000 - val_loss: 2.9377e-05
Epoch 14/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 450ms/step - accuracy: 1.0000 - loss: 4.0459e-04 - val
_accuracy: 1.0000 - val_loss: 1.9217e-05
Epoch 15/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 449ms/step - accuracy: 1.0000 - loss: 8.9670e-04 - val
_accuracy: 1.0000 - val_loss: 1.4647e-05
Epoch 16/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 441ms/step - accuracy: 1.0000 - loss: 3.2640e-04 - val
_accuracy: 1.0000 - val_loss: 1.0095e-05
Epoch 17/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 6s 459ms/step - accuracy: 1.0000 - loss: 1.1716e-04 - val
_accuracy: 1.0000 - val_loss: 8.6318e-06
Epoch 18/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 439ms/step - accuracy: 1.0000 - loss: 2.2646e-04 - val
_accuracy: 1.0000 - val_loss: 9.4119e-06
Epoch 19/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 445ms/step - accuracy: 1.0000 - loss: 1.7807e-04 - val
```

```
_accuracy: 1.0000 - val_loss: 8.3598e-06
Epoch 20/20
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 437ms/step - accuracy: 1.0000 - loss: 1.1367e-04 - val
_accuracy: 1.0000 - val_loss: 9.1082e-06
```

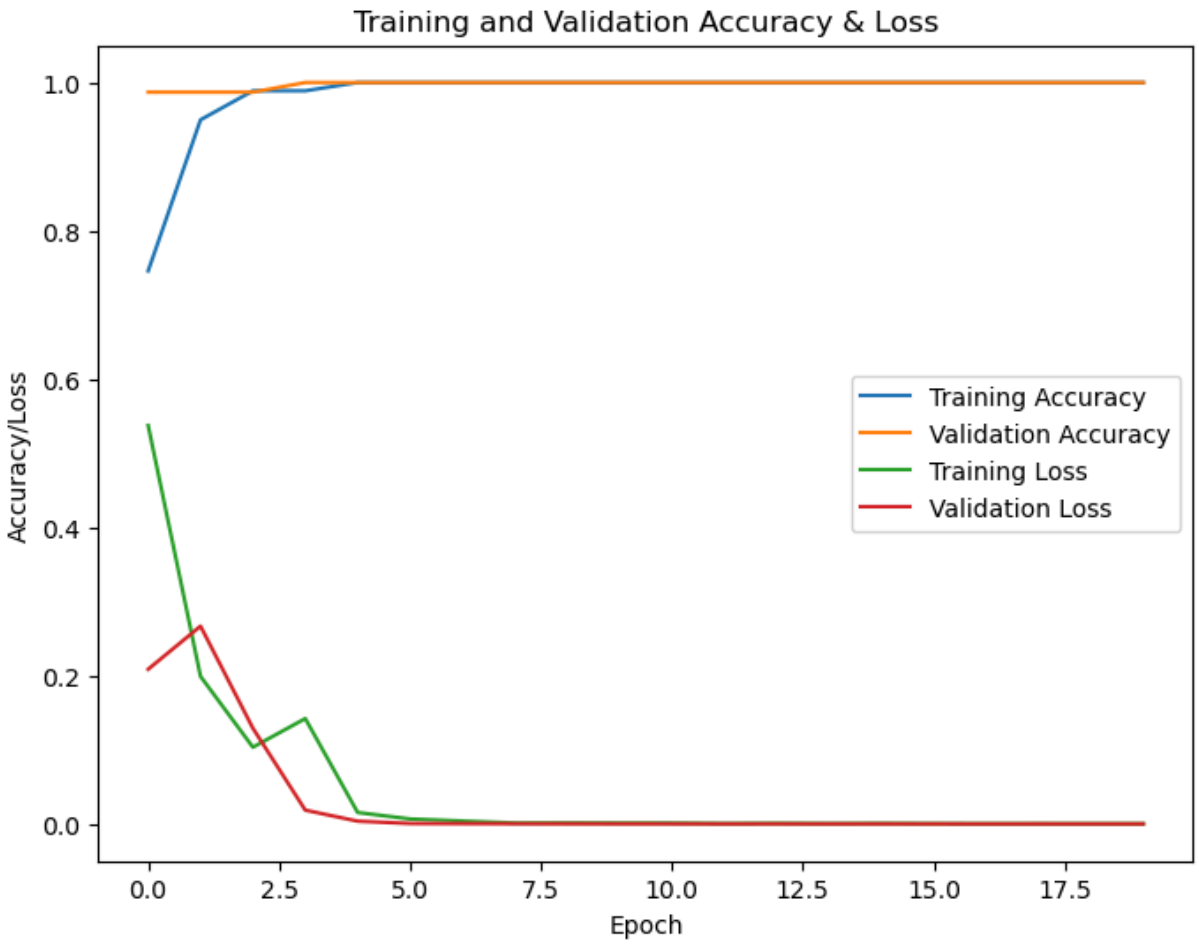
In [59]:
```python
# Get training accuracy
train_loss, train_accuracy = model.evaluate(X_train_reshaped, y_train)
print(f'Train Loss: {train_loss}, Train accuracy: {train_accuracy}')
print("Training Accuracy:", train_accuracy)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 75ms/step - accuracy: 1.0000 - loss: 2.5802e-06
Train Loss: 3.470942601779825e-06, Train accuracy: 1.0
Training Accuracy: 1.0
```

In [60]:
```python
# Model Evaluation
test_loss, test_accuracy = model.evaluate(X_test_reshaped, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_accuracy}')
print("Testing Accuracy:", test_accuracy)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 83ms/step - accuracy: 1.0000 - loss: 5.9966e-06
Test loss: 9.108201993512921e-06, Test accuracy: 1.0
Testing Accuracy: 1.0
```

In [61]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy/Loss')
plt.title('Training and Validation Accuracy & Loss')
plt.legend()
plt.show()
```



Multi-class Model

In [62]:
```
# Load data and check keys
with h5py.File('Downloads/spectrograms.h5', 'r') as f:
    print(list(f.keys()))
```

```
['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3', 'norfli', 'r
ewbla', 'stejay', 'wesmea', 'whcspa']
```

In [63]:
```
import h5py
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from skimage.transform import resize

# Open the HDF5 file and load the data
with h5py.File('Downloads/spectrograms.h5', 'r') as f:
    keys = ['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3',
    X = []
    y = []
    for key in keys:
        # Resize each spectrogram to a fixed shape (e.g., 343x256)
```

```
        resized_spectrogram = resize(f[key][:], (f[key].shape[0], 343, 256))
        X.append(resized_spectrogram)  # Spectrograms
        y.extend([key] * len(resized_spectrogram))  # Labels

# Convert lists to numpy arrays
X = np.concatenate(X, axis=0)
y = np.array(y)

# Reshape X to add a channel dimension
X = X.reshape(-1, 343, 256, 1)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Convert string labels to numerical labels using LabelEncoder
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Define the CNN Model
model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(343, 256, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(len(keys), activation='softmax')
])

# Compile the Model with Adam Optimizer
model.compile(optimizer=Adam(),
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

# Train the Model
history = model.fit(X_train, y_train_encoded, epochs=10, batch_size=16, validation_
```

```
C:\Users\akotwani\AppData\Local\anaconda3\Lib\site-packages\keras\src\layers\convolu
tional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argum
ent to a layer. When using Sequential models, prefer using an `Input(shape)` object
as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 78s 530ms/step - accuracy: 0.3903 - loss: 1.9904 - val_
accuracy: 0.9902 - val_loss: 0.2434
Epoch 2/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 67s 495ms/step - accuracy: 0.8798 - loss: 0.4870 - val_
accuracy: 0.9957 - val_loss: 0.0209
Epoch 3/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 64s 475ms/step - accuracy: 0.9397 - loss: 0.2637 - val_
accuracy: 0.9967 - val_loss: 0.0195
Epoch 4/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 64s 474ms/step - accuracy: 0.9544 - loss: 0.1631 - val_
accuracy: 0.9967 - val_loss: 0.0064
Epoch 5/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 64s 473ms/step - accuracy: 0.9734 - loss: 0.1143 - val_
accuracy: 1.0000 - val_loss: 0.0023
Epoch 6/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 64s 470ms/step - accuracy: 0.9661 - loss: 0.0949 - val_
accuracy: 1.0000 - val_loss: 8.4169e-04
Epoch 7/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 64s 473ms/step - accuracy: 0.9841 - loss: 0.0548 - val_
accuracy: 1.0000 - val_loss: 2.5465e-04
Epoch 8/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 5051s 38s/step - accuracy: 0.9825 - loss: 0.0634 - val_
accuracy: 1.0000 - val_loss: 2.5003e-04
Epoch 9/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 61s 452ms/step - accuracy: 0.9860 - loss: 0.0605 - val_
accuracy: 1.0000 - val_loss: 0.0014
Epoch 10/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 86s 481ms/step - accuracy: 0.9841 - loss: 0.0544 - val_
accuracy: 1.0000 - val_loss: 9.2527e-04
```

In [65]:
```python
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test_encoded)
print('Test accuracy:', test_acc)
```

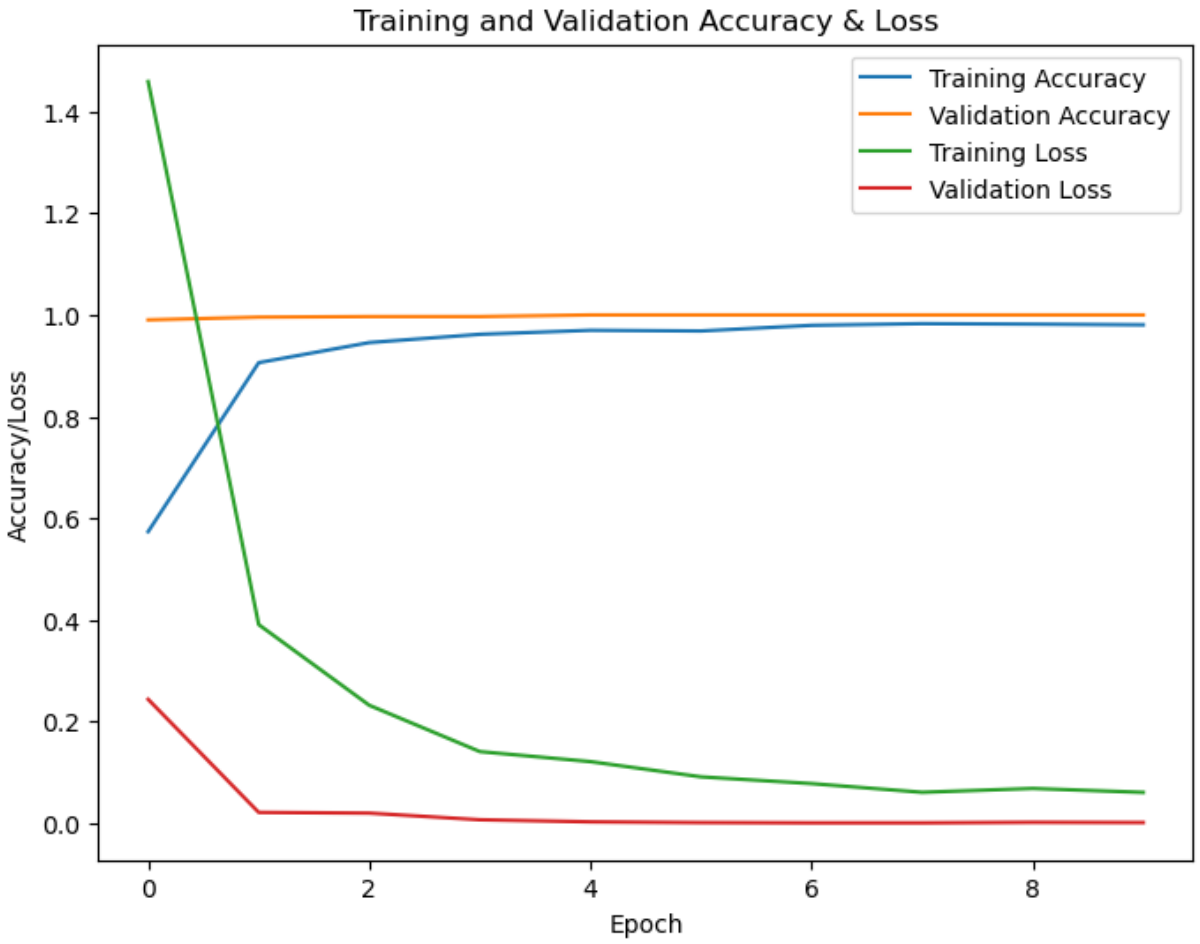
```
29/29 ━━━━━━━━━━━━━━━━━━━━ 4s 119ms/step - accuracy: 1.0000 - loss: 0.0014
Test accuracy: 1.0
```

In [66]:
```python
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test_encoded)
print('Test accuracy:', test_acc)
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 4s 122ms/step - accuracy: 1.0000 - loss: 0.0014
Test accuracy: 1.0
```

In [67]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy/Loss')
plt.title('Training and Validation Accuracy & Loss')
```

```
plt.legend()
plt.show()
```



Model 2 : Adding layers and drop out

In [68]:
```
#Open the HDF5 file and load the data
with h5py.File('Downloads/spectrograms.h5', 'r') as f:
    keys = ['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3',
    X = []
    y = []
    for key in keys:
        # Resize each spectrogram to a fixed shape (e.g., 343x256)
        resized_spectrogram = resize(f[key][:], (f[key].shape[0], 343, 256))
        X.append(resized_spectrogram)  # Spectrograms
        y.extend([key] * len(resized_spectrogram))  # Labels

# Convert lists to numpy arrays
X = np.concatenate(X, axis=0)
y = np.array(y)

# Reshape X to add a channel dimension
X = X.reshape(-1, 343, 256, 1)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

```
# Convert string labels to numerical labels using LabelEncoder
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 256, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(len(keys), activation='softmax')
])

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

# Train the model
history = model.fit(X_train, y_train_encoded, epochs=10, batch_size=16, validation_
```

```
C:\Users\akotwani\AppData\Local\anaconda3\Lib\site-packages\keras\src\layers\convolu
tional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argum
ent to a layer. When using Sequential models, prefer using an `Input(shape)` object
as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.2649 - loss: 2.2095 - val_ac
curacy: 0.9924 - val_loss: 0.1604
Epoch 2/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.8977 - loss: 0.3503 - val_ac
curacy: 1.0000 - val_loss: 0.0031
Epoch 3/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.9749 - loss: 0.0956 - val_ac
curacy: 1.0000 - val_loss: 0.0015
Epoch 4/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.9732 - loss: 0.0949 - val_ac
curacy: 1.0000 - val_loss: 3.7980e-05
Epoch 5/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 146s 1s/step - accuracy: 0.9829 - loss: 0.0509 - val_ac
curacy: 1.0000 - val_loss: 0.0029
Epoch 6/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 149s 1s/step - accuracy: 0.9852 - loss: 0.0454 - val_ac
curacy: 0.9978 - val_loss: 0.0092
Epoch 7/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 151s 1s/step - accuracy: 0.9888 - loss: 0.0564 - val_ac
curacy: 1.0000 - val_loss: 3.5402e-04
Epoch 8/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 160s 1s/step - accuracy: 0.9923 - loss: 0.0259 - val_ac
curacy: 1.0000 - val_loss: 7.4458e-04
Epoch 9/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 164s 1s/step - accuracy: 0.9929 - loss: 0.0493 - val_ac
curacy: 1.0000 - val_loss: 1.7937e-06
Epoch 10/10
135/135 ━━━━━━━━━━━━━━━━━━━━ 150s 1s/step - accuracy: 0.9921 - loss: 0.0248 - val_ac
curacy: 1.0000 - val_loss: 9.3515e-06
```

In [69]:
```python
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test_encoded)
print('Test accuracy:', test_acc)
```

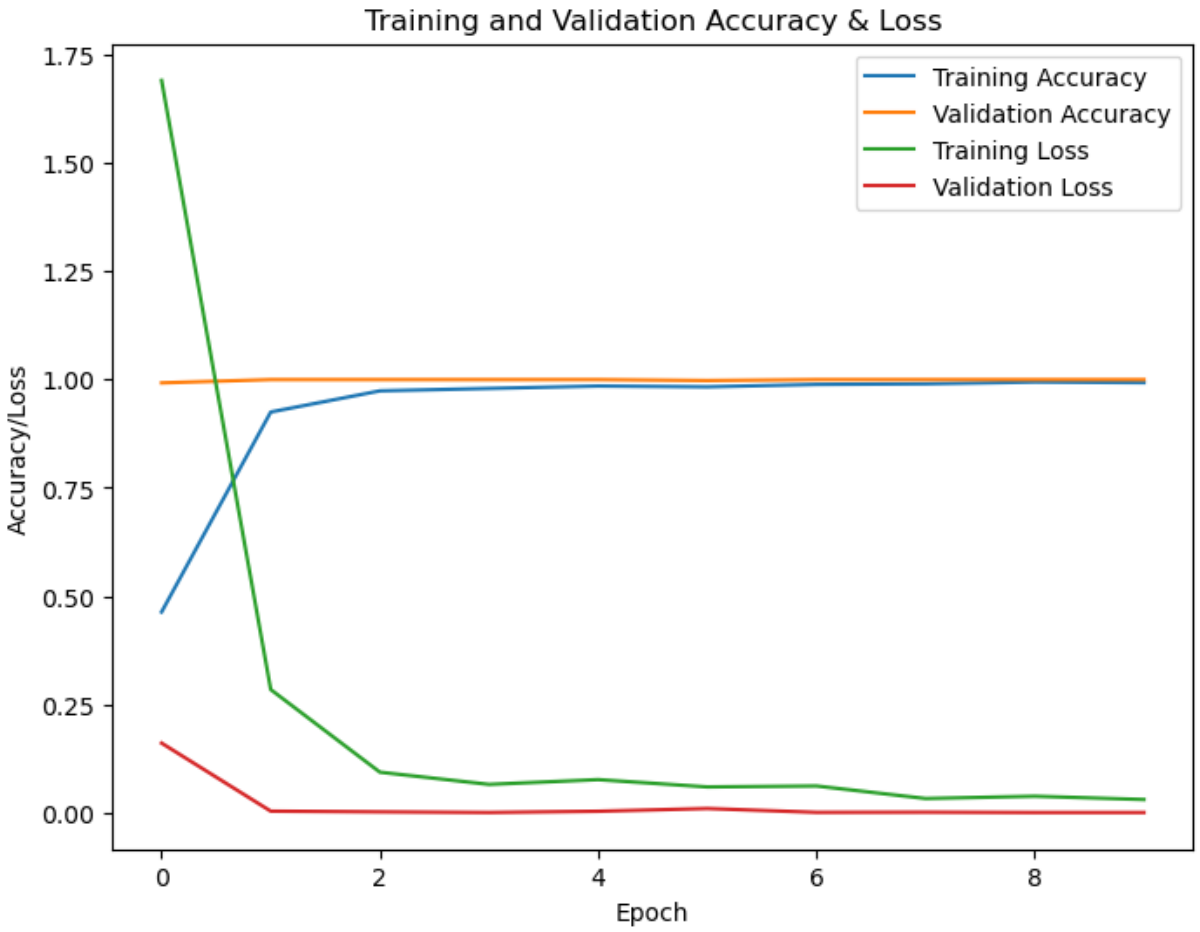
```
29/29 ━━━━━━━━━━━━━━━━━━━━ 10s 331ms/step - accuracy: 1.0000 - loss: 1.4925e-05
Test accuracy: 1.0
```

In [70]:
```python
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test_encoded)
print('Test accuracy:', test_acc)
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 10s 333ms/step - accuracy: 1.0000 - loss: 1.4925e-05
Test accuracy: 1.0
```

In [71]:
```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Accuracy/Loss')
plt.title('Training and Validation Accuracy & Loss')
```

```
plt.legend()
plt.show()
```



External Test Data

In [72]:
```
import IPython.display as ipd
```

In [73]:
```
import os
import numpy as np
import librosa
import matplotlib.pyplot as plt


def compute_energy(audio_signal, n_fft, hop_length):
    """Compute the energy of an audio signal."""
    return librosa.feature.rms(y=audio_signal, frame_length=n_fft, hop_length=hop_l


def extract_loud_parts(energy, threshold):
    """Extract the indexes of loud parts of the audio signal."""
    return np.where(energy > threshold)[1]


def extract_bird_call_windows(audio_signal, loud_indexes, sample_rate, window_size,
    """Extract 2-second windows of sound where a bird call is detected."""
```

```
    windows = []
    for loud_idx in loud_indexes:
        start_time = loud_idx * hop_length / sample_rate
        end_time = start_time + window_size
        window = audio_signal[int(start_time * sample_rate):int(end_time * sample_r
        windows.append(window)
    return windows



def compute_spectrogram(audio_signal, sample_rate, n_fft, hop_length):
    """Compute the spectrogram of an audio signal."""
    return librosa.feature.melspectrogram(y=audio_signal, sr=sample_rate, n_fft=n_f
                                          n_mels=256, fmax=8000)


def preprocess_audio(audio_path, new_sample_rate=22050, min_loud_part_length=0.5, b
                     n_fft=2048, hop_length=512):
    """Preprocess audio data."""
    # Load the audio file and resample to the new sample rate
    audio_signal, sample_rate = librosa.load(audio_path, sr=new_sample_rate)
    # Compute the energy of the audio signal
    energy = compute_energy(audio_signal, n_fft, hop_length)
    # Extract the indexes of loud parts of the audio signal
    loud_indexes = extract_loud_parts(energy, np.max(energy) * 0.5)
    # Extract 2-second windows of sound where a bird call is detected
    bird_call_windows = extract_bird_call_windows(audio_signal, loud_indexes, sampl

    # Compute spectrograms for each bird call window
    spectrograms = []
    for window in bird_call_windows:
        spectrogram = compute_spectrogram(window, sample_rate, n_fft, hop_length)
        # Ensure the spectrogram has the desired shape (343x256)
        if spectrogram.shape[1] < 343:
            padding = np.zeros((256, 343 - spectrogram.shape[1]))
            spectrogram = np.hstack((spectrogram, padding))
        elif spectrogram.shape[1] > 343:
            spectrogram = spectrogram[:, :343]
        spectrograms.append(spectrogram)
    # Extract labels from file name
    labels = [os.path.splitext(os.path.basename(audio_path))[0] for _ in range(len(
    return spectrograms, labels


# Initialize lists for storing all spectrograms and labels
all_specs = []
all_labels = []

file_names = ['Downloads/test_birds/test1.mp3',
              'Downloads/test_birds/test2.mp3',
              'Downloads/test_birds/test3.mp3']

for audio in file_names:
    specs, labels = preprocess_audio(audio)
    all_specs.extend(specs)
    all_labels.extend(labels)
```

```
# Convert lists to numpy arrays
all_specs = np.array(all_specs)
all_labels = np.array(all_labels)
```

In [74]:
```
all_specs
```

Out[74]:
```
array([[[8.61781165e-02, 8.13594908e-02, 3.41617875e-02, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [7.99247921e-02, 3.55773687e-01, 4.98237044e-01, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [5.00426650e-01, 1.46379006e+00, 1.80137181e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [2.88795472e-05, 1.24284270e-05, 4.87457328e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [3.47215682e-05, 1.32188161e-05, 3.85080966e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [5.58032407e-05, 2.01904113e-05, 7.57667021e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00]],

       [[3.33165348e-01, 2.02639475e-01, 7.52855884e-03, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [5.51799059e-01, 6.44358099e-01, 2.12053135e-01, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [5.01183629e-01, 1.39171016e+00, 2.27287745e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [1.25304914e-05, 6.41128690e-06, 5.15808370e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [1.42480239e-05, 6.76330592e-06, 7.75733952e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [1.69670875e-05, 1.10299243e-05, 1.01988771e-05, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00]],

       [[2.90582189e-03, 2.60312692e-03, 1.64706483e-02, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [8.31177607e-02, 1.55584410e-01, 3.14225823e-01, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [7.06731975e-01, 2.18413687e+00, 2.74331284e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [4.16178955e-05, 1.48574190e-05, 5.38115955e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [3.92065922e-05, 1.72255059e-05, 4.45566002e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [3.38074606e-05, 1.67326634e-05, 9.21666287e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00]],

       ...,

       [[1.14072813e-02, 1.15348827e-02, 1.55756879e-03, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [2.73291580e-02, 1.71520449e-02, 1.49772316e-02, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [5.72420992e-02, 1.19678583e-02, 5.37931286e-02, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [2.96329472e-06, 7.93511754e-06, 5.02154444e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [2.72364332e-06, 3.58543866e-06, 2.60847628e-06, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [2.31851482e-06, 4.25154758e-06, 5.90496211e-06, ...,
```

```
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]],

       [[2.32991815e-01, 7.98816383e-02, 2.03093351e-03, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [2.31925339e-01, 6.47531524e-02, 1.03269583e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [3.81339341e-01, 1.24398991e-01, 4.18204218e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [1.51582026e-05, 6.90289107e-06, 4.20583547e-06, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [2.14826759e-05, 7.72506428e-06, 5.05813205e-06, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [1.98114249e-05, 1.02303602e-05, 8.98836242e-06, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]],

       [[2.60544538e-01, 6.52236566e-02, 3.58193251e-03, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [3.29218179e-01, 1.15414701e-01, 1.68262366e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [4.49774772e-01, 1.65626913e-01, 2.75780987e-02, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [1.39866916e-06, 4.02117621e-06, 6.52407107e-06, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [1.50988467e-06, 4.95696258e-06, 4.08317737e-06, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [3.82623193e-06, 8.83428402e-06, 4.89983768e-06, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]]])
```

In [75]:
```
all_labels
```

Out[75]:
```
array(['test1', 'test1', 'test1', 'test1', 'test1', 'test1', 'test1',
       'test1', 'test1', 'test1', 'test1', 'test1', 'test1', 'test1',
       'test1', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test2', 'test2', 'test2', 'test2', 'test2',
       'test2', 'test2', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3', 'test3', 'test3',
       'test3', 'test3', 'test3', 'test3', 'test3'], dtype='<U5')
```

In [76]:
```python
print("Shape of spectrograms array:", all_specs.shape)
print("Shape of labels array:", all_labels.shape)
```

```
Shape of spectrograms array: (229, 256, 343)
Shape of labels array: (229,)
```

In [77]:
```python
# Check input shape
print('Input shape expected by the model:', model.input_shape)
print('Shape of spectrograms (all_specs):', all_specs.shape)
```

```
Input shape expected by the model: (None, 343, 256, 1)
Shape of spectrograms (all_specs): (229, 256, 343)
```

In [78]:
```python
# Reshape spectrograms and add channel dimension
all_specs_reshaped = np.expand_dims(all_specs, axis=-1)
# Check the shape and data type after reshaping and conversion
print('Shape of reshaped spectrograms:', all_specs_reshaped.shape)
```

```
Shape of reshaped spectrograms: (229, 256, 343, 1)
```

In [79]:
```python
test_spectrograms_reshaped = all_specs_reshaped
# Compile the model
model.compile(optimizer='adam',
```

```
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

predictions = model.predict(test_spectrograms_reshaped)

SPECIES_CLASS = {
    0: 'amecro', 1: 'barswa', 2: 'bkcchi', 3: 'blujay', 4: 'daejun', 5: 'houfin',
    6: 'mallar3', 7: 'norfli', 8: 'rewbla', 9: 'stejay', 10: 'wesmea', 11: 'whcspa'
}

# List of filenames for the three audio files
file_names = ['Downloads/test_birds/test1.mp3',
              'Downloads/test_birds/test2.mp3',
              'Downloads/test_birds/test3.mp3']

predicted_species = []

for i, file_name in enumerate(file_names):
    bird_prob = predictions[i]
    # Converting probabilities to speciess names
    bird_pred = [(SPECIES_CLASS[j], prob) for j, prob in enumerate(bird_prob)]
    bird_pred.sort(key=lambda x: x[1], reverse=True)  # Sort
    # Select top 3 probabilities
    top3_species = bird_pred[:3]
    predicted_species.append((file_name, top3_species))

# Print the predicted bird speciess along with their original labels
for file_name, bird_prob in predicted_species:
    print("Original Label:", file_name)
    for speciess, prob in bird_prob:
        print("Predicted Bird :", speciess, "- Probability:", prob)
```

```
WARNING:tensorflow:5 out of the last 38 calls to <function TensorFlowTrainer.make_pr
edict_function.<locals>.one_step_on_data_distributed at 0x0000020430B82160> triggere
d tf.function retracing. Tracing is expensive and the excessive number of tracings c
ould be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors w
ith different shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function has reduce_retra
cing=True option that can avoid unnecessary retracing. For (3), please refer to http
s://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorfl
ow.org/api_docs/python/tf/function for  more details.
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 324ms/step
Original Label: Downloads/test_birds/test1.mp3
Predicted Bird : wesmea - Probability: 0.18448363
Predicted Bird : mallar3 - Probability: 0.18068194
Predicted Bird : whcspa - Probability: 0.1671998
Original Label: Downloads/test_birds/test2.mp3
Predicted Bird : wesmea - Probability: 0.18933687
Predicted Bird : mallar3 - Probability: 0.18871363
Predicted Bird : whcspa - Probability: 0.17609403
Original Label: Downloads/test_birds/test3.mp3
Predicted Bird : mallar3 - Probability: 0.21559522
Predicted Bird : wesmea - Probability: 0.19716549
Predicted Bird : whcspa - Probability: 0.15690945
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: