Seattle Bird Call Classification Using Neural Networks

Abstract

The aim of this study is to classify various kinds of bird species based on their calls or chirping sounds in Seattle. We developed two models using the spectrograms of 10 MP3 sound clips of various lengths for each of 12 bird species. The first model is a binary classifier which distinguishes between two bird species based on different characteristics. The second model is a multi-class classifier which identifies different chirping sounds of birds. We used different neural network models and hyperparameters to build binary and multi-class classifiers. The accuracy which we obtained on Binary classifier comes out to be around 96.67% and for multi-class classifier, the best accuracy is around 71.26%. With the help of this study, we will be able to gain a better understanding of how neural networks can be applied for classification of audio data in our study of bird species classification based on their calls.

Introduction

It is possible to gain valuable insights into ecological health and biodiversity through acoustic monitoring of bird species. We developed and tested different neural network models to classify different bird species, especially those in Seattle's avian community. We used the spectrogram data from the Birdcall competition data, originally from Xeno-Canto, a crowd-sourced bird sounds archive. The dataset contains additional recordings of the 264 species in the Birdcall competition along with its corresponding metadata. We pre-processed the original data to focus of the 12 different bird species found in Seattle. By using pre-processed data, we developed binary and multi-class classification models. The binary model differentiates between two selected species, while the multi-class model classifies all 12 bird species based on their calls. This study will provide insight into how neural networks may be applied to bioacoustics in a broader context. It will also provide insight into the strengths and limitations of deep learning in environmental monitoring.

Background

Neural Networks

A neural network is a computational model consisting of interconnected nodes or neurons that are organized into layers. Layers can have as many as a dozen number of nodes or millions, depending on how complex neural networks will be required to uncover hidden patterns. Information is then fed through these nodes and according to this information, neural network adjusts the strength of the connection called weights during the training phase to learn from the data. This allows the network to recognize patterns, make predictions, and complete various tasks related to problems.

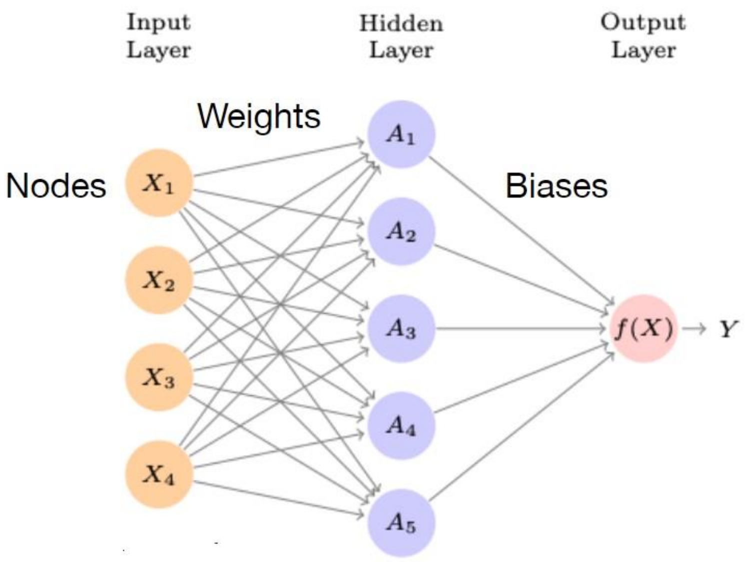


Fig 1:- Neural Network Architecture.

The architecture of a neural network consists of three layers such as the input layer, the hidden layer, and the output layer.

The input layer of a neural network is the first layer. It receives input information from external sources. The input data is available in the form of text, numbers, images, or audio files. Input is also a linear combination of data observations.[6]

Hidden layers are middle layers in neural networks. There can be one or more hidden layers in a neural network depending on how deep the training data is used to uncover patterns. This layer extracts relevant patterns from input data and transfers them to the next layer for analysis. It also improves the network's efficiency by recognizing the most important patterns from the input nodes. Hidden layers are ideal for performing mathematical computations on input data.[6]

The output layer consists of rigorous mathematical computations carried out by the hidden layer to obtain results based on these calculations. Additionally, it serves as a final output for bringing the information learned by the neural network. It is also a linear combination of the activation functions. [6]

The activation function is used to perform complex computations to uncover the patterns in the input data and it also introduces the non-linearity into the output of the network. We know that the neural network has neurons that work in correspondence with weight, bias, and their respective activation function. In a neural network, we would update the weights and biases of the neurons based on the error at the output layer. This process is known as back-propagation. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. If there is no activation function in the neural network, it is essentially just a linear regression model. By non-linearly transforming the input, the activation function is capable of learning and performing more complex tasks. [2]

The most popular activation functions are sigmoid and RELU functions.

The sigmoid activation function has values between 0 and 1, which can be interpreted as a probability that the input belongs to a specific class. The sigmoid function produces very small gradients, which can result in neural networks stagnating. [3] Furthermore, gradients will disappear beyond 1 and 0 as well. The equation of sigmoid function can be represented as

$$A = 1 / (1 + e^{-x})$$

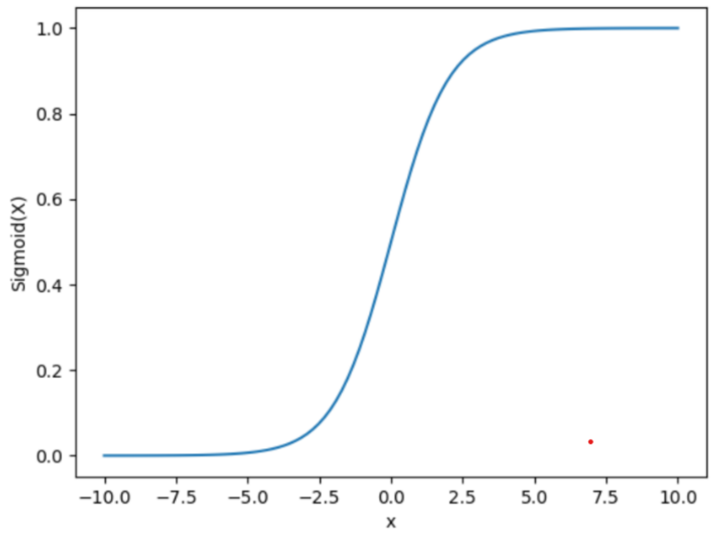


Fig 2:- Sigmoid function.[2]

ReLU (*Rectified linear unit*) is a common choice for hidden layer activation. This method uses only a simple thresholding operation, which makes it computationally efficient. Since the gradients are 1 if $x > 0$, it is less susceptible to vanishing gradients. When a value is negative, the gradient is zero, resulting in a neuron that will never be updated. The equation of ReLU function can be represented as

$$A(x) = \max(0,x)$$

It gives an output x if x is positive and 0 otherwise. [3]

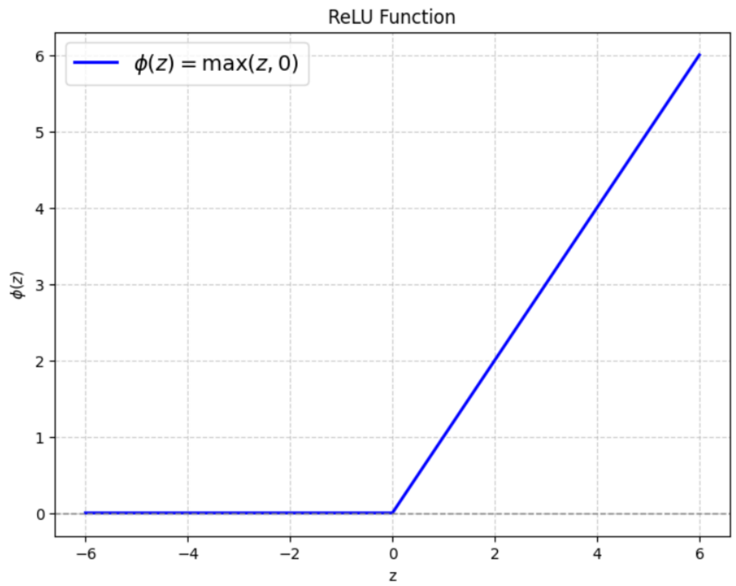


Fig 3:- ReLU Function

Convolutional Neural Network

The Convolutional Neural Network (CNN) is a class of neural networks used to process data with a grid-like topology, such as images. Digital images represent visual data in binary form. The image contains a series of pixels arranged in a grid-like arrangement and containing colour and brightness values.

CNNs typically have three layers: a convolutional layer, a pooling layer, and a fully connected layer.

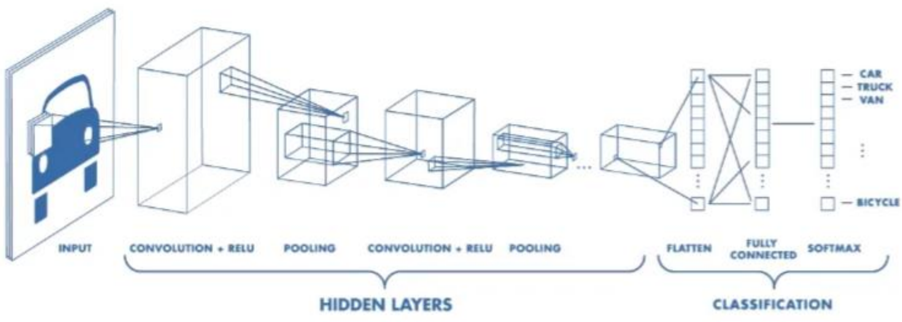


Fig 4:- Architecture of a CNN (Source)

Convolution is the core building block of CNN. It carries most of the network's computation. In this layer, the dot product of two matrices is calculated, where the first matrix is the set of learnable parameters known as a kernel, and the second matrix is the restricted receptive field. Kernels are smaller in size but more detailed than images. In other words, if the image has RGB channels, the kernel height and width will be relatively small, while the depth is large.

The kernel slides over the height and width of the image, producing an image representation. This creates a two-dimensional representation of the image known as an activation map that shows the kernel's response at each spatial position.

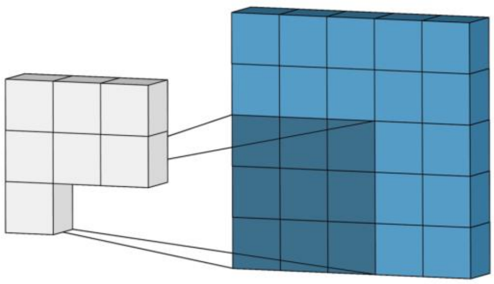


Fig 5:- Convolution Operation (source)

In the pooling layer, nearby outputs are used to replace the network output at certain locations. It decreases computation requirements and weights by reducing the spatial size of the representation. During pooling, every slice is processed separately.

Several pooling functions are available, including the average of the rectangular neighbourhood, L2 norm of the rectangular neighbourhood, and a weighted average. The most popular process is max pooling, which reports the maximum output.

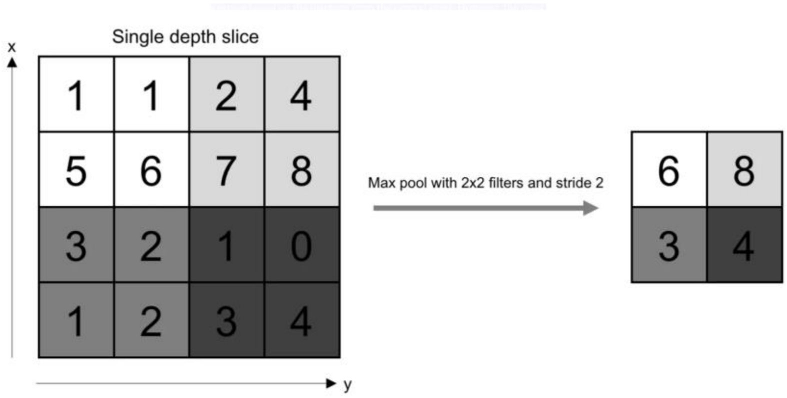


Figure 6: Pooling Operation (Source: O'Reilly Media)

In a Fully Connected Neural Network, neurons are fully connected to those in the preceding and succeeding layers. As a result, it can be computed using matrix multiplication and bias effects. Fully Connected layer maps inputs and outputs based on representation. [5]

Methodology

Data Preprocessing

We have been given spectrograms of audio files for 12 bird species taken from the Birdcall competition, originally from Xeno-Canto, a crowd-sourced bird sound archive. The sound clips were subsampled to 22050 Hz. We then took clips where the frequency of bird calls is loud enough to be audible to human ears to distinguish the sound of birds. The loud parts were sampled in such a way that frequency is greater than 0.5 seconds to identify the bird call. In the next step, a spectrogram is generated for each of the two 2-second windows, resulting in a "picture" of the bird call that is 343 (time) x 256 (frequency). All clips were saved individually so that the number of samples was uneven. [8]

Binary Classification model

We choose Northern Flicker and Steller's Jay as our two bird species for binary classification. Following that, we sampled our spectrogram to filter the two species by their calls, 'norfli' for northern flicker and 'stejay' for steller jay. We then labelled them in binary format. We labelled the Northern flicker 0 and the Steller Jay 1. After that, we combined the data and labels and normalized the data. After normalizing the data, we transposed it to match the input shape for our Convolutional Neural Network Model which is in the format (frequency, timesteps, no. of samples). Then we categorized the labels.

Following preprocessing the data, we split the training and validation sets in a 70-30 ratio, i.e., 70% training and 30% validation. We then defined our Convolutional model. There are 32 filters with 3x3 kernels in our CNN model to avoid overfitting and to keep computational cost low. We used ReLU as activation function that is suitable for extracting low-level features from the input shape of 256x343 pixels with a single channel. Furthermore, we used a MaxPooling2D layer to reduce the spatial dimensions by half, making the model invariant to small input translations. A second Conv2D and MaxPooling2D layer is added to further extract features, and a Dropout layer prevents overfitting. The model ends with a Flatten layer for converting 2D features to 1D vectors, a Dense fully-connected layer for classifying data, and a Dense layer with sigmoid activation for generating class probabilities. Next, we compiled our model with Adam as the optimizer and loss function as binary-cross entropy. Following that, we fitted the model with 20 epochs on our training and validation sets. After that, we calculated the validation loss and accuracy of the model, along with its confusion matrix.

Multi-Classification model

We followed almost the same steps as performed in the binary classification model. The only change we did was sampled the entire data and label all the 12 species and then appended the data and labels in the empty list and then combined them. After this we performed normalization of the data like the one performed in binary classification and convert the labels to categorical data. After that we split the data into 70 to 30 ratio. Then we defined two CNN models for multi-class classification.

We started by specifying an input layer (256 x 343 x 1). Two Conv2D layers with 32 and 64 filters are employed to extract features, each using a 3x3 kernel size and ReLU activation. The second model has additional layers with 128 and 256 filters to improve the performance of the model. In each convolutional layer, a MaxPooling2D layer with a 2x2 pool size simplifies the spatial dimensions, helping us to focus on important features while minimizing computation. The Flatten layer creates a 1D feature vector from the 2D feature maps. The dropout layers prevent overfitting by randomly setting input units to 0 during training at a rate of 50% to handle higher-level reasoning. Lastly, a second Dense layer classifies the inputs into 12 categories using a SoftMax activation function. Next, we compiled our model with Adam as the optimizer and loss function as categorical-cross entropy. Following that, we fitted the model with 20 epochs and batch size of 128 on our training and validation sets for the 1st model. For the second model, we run 30 epochs and reduced the batch size to 64. After that, we calculated the validation loss and accuracy of the model, along with its confusion matrix for both the models.

Predictions on Pretrained multi-class model

We have been provided with three test clips that we have to fit into our multiclass classification and predict which species of bird were present in these test clips. To make predictions of these audio clips, we first converted them into spectrograms by following the same preprocessing steps as when creating bird species spectrogram. Then we predicted bird species based on the multi-class classification model on the test data spectrogram. Then we check if there are more than one bird species present in the clips. After that we print the results.

Results

Binary Classification Model

After building the neural network for Binary Classification of the Northern Flicker and Steller's Jay, the accuracy of the model comes out to be around 96.67% with 5.9% loss. The F1-score of the Northern Flicker is slightly better (around 97%) as compared to Steller's Jay. The Following is our Model Performance plots for Binary Model.

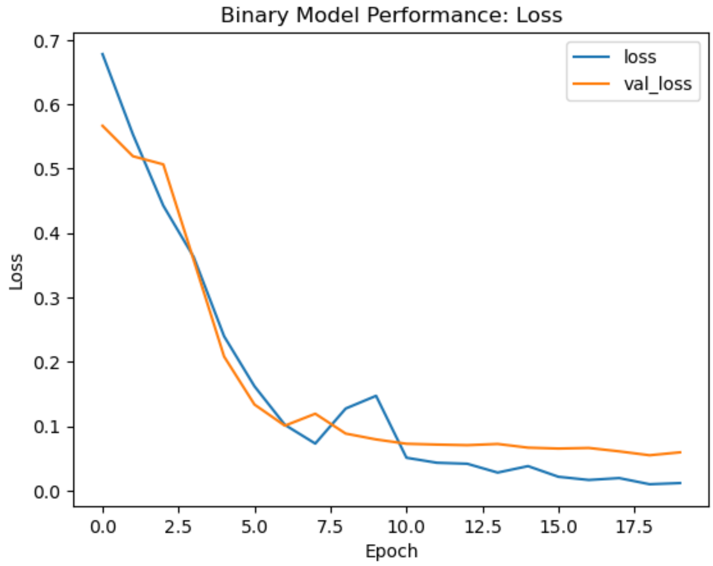


Fig-7:- Loss Performance of Binary Model

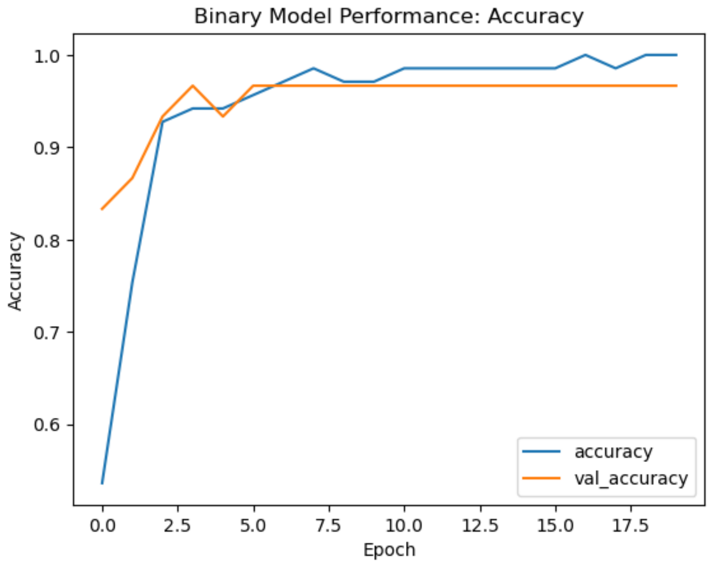


Fig-8:- Accuracy Performance of Binary Model

The following is the confusion matrix for the binary model to classify the two selected bird species.

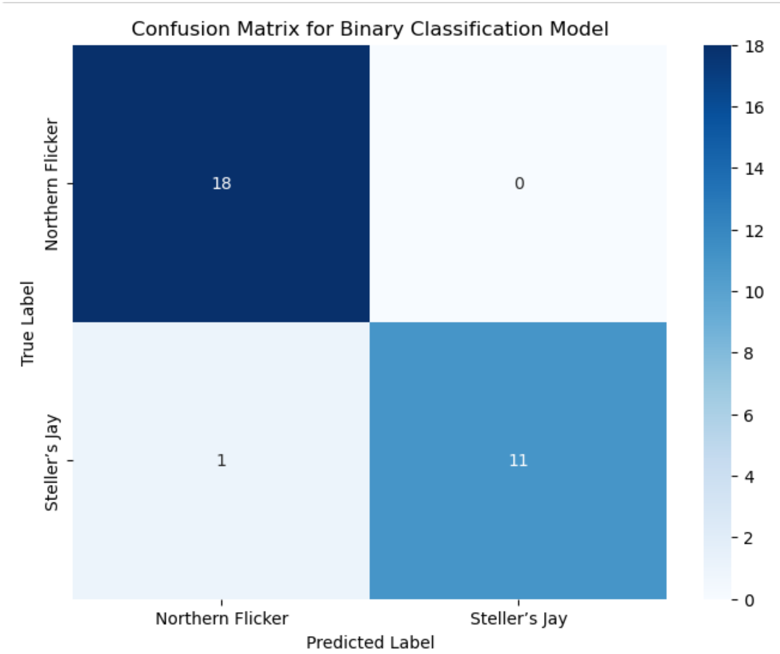


Fig-9:- Confusion Matrix for Binary Model

Multi-class Classification Model

The following is the table for accuracy and loss performance of both the CNN models

Models	Accuracy	Loss
CNN with 2 layers	67.24%	12.7 %
CNN with 4 layers	71.26%	17.8%

The following are the performance graphs for both the CNN models

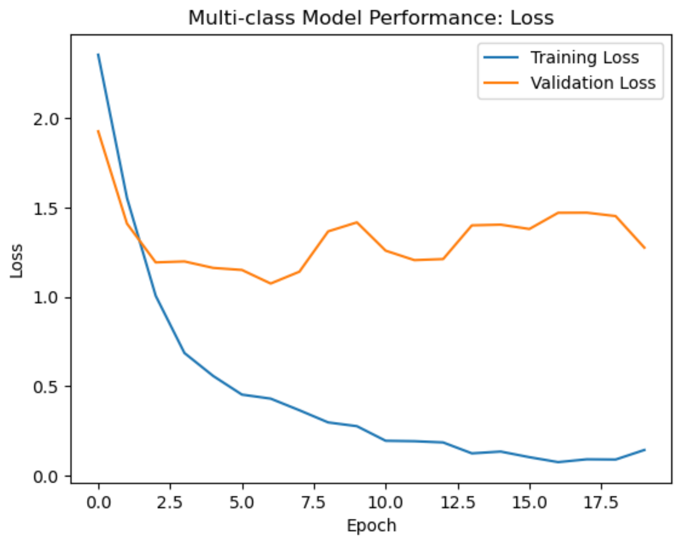


Fig-10: Loss Performance for 1^{st} Multi-class Model

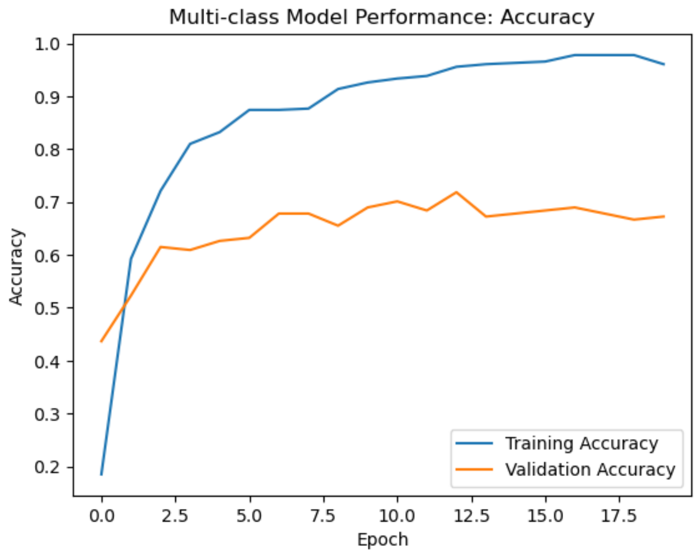


Fig 11:- Accuracy Performance for 1st Multi-class Model

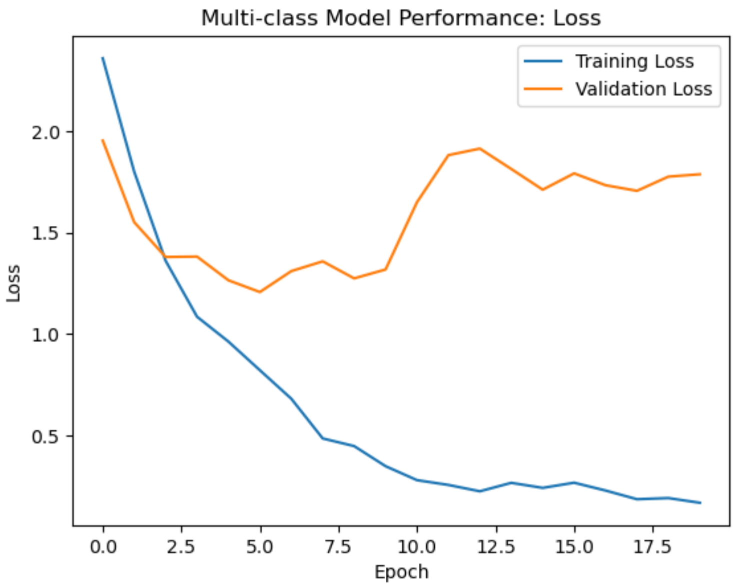


Fig 12:- Loss Performance for 2nd Multi-class Model

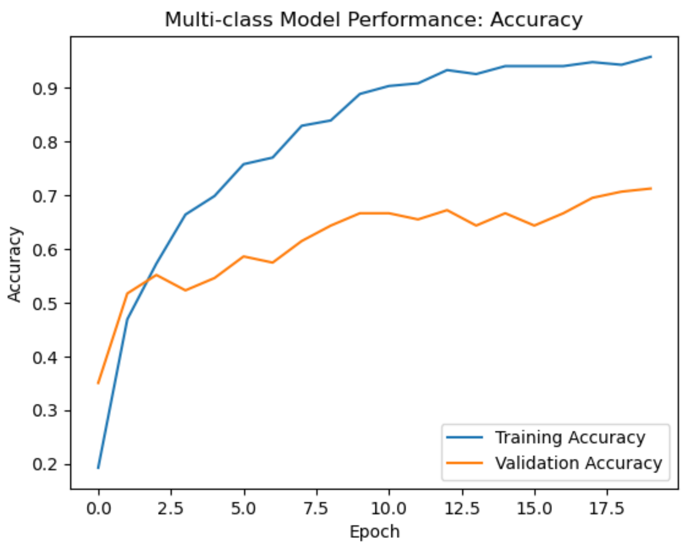


Fig 13:- Accuracy Performance for 2nd Multi-class Model

The following is the confusion matrix for the best performing multi-class model to classify all 12 bird species.

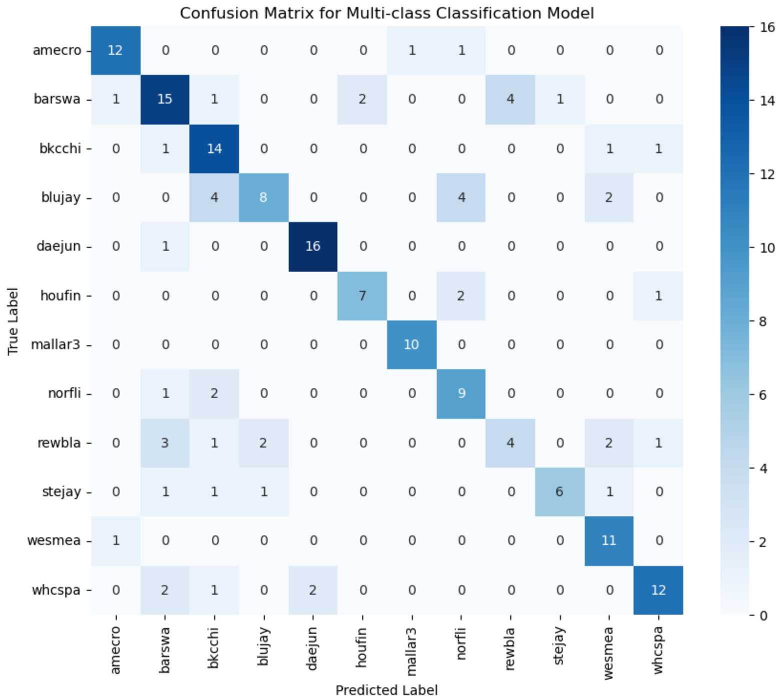


Fig 14:- Confusion Matrix for Best Performing Multi-class Model

After we build the multi-class model, we then predicted the bird species form our test spectrogram by fitting the test files into the model. Also, we did check for the presence of multiple birds. The results came out to be that all the test clips have one bird in come and that is dark-eyed junco. The following is the table for the results. Also, there were no sightings of multiple birds in the clips.

	File	Predictions
0	test_birds/test1.mp3	[[(daejun, 1.0), (amecro, 0.0), (barswa, 0.0)]...
1	test_birds/test2.mp3	[[(daejun, 1.0), (amecro, 0.0), (barswa, 0.0)]]
2	test_birds/test3.mp3	[[(daejun, 1.0), (amecro, 0.0), (barswa, 0.0)]...

Fig 15:- Prediction table for Test Clips

Discussion

The neural network models which we build and train to predict the bird species around Seattle did perform well. Our binary classifier gave us some good predictions with minimum classification error and did classify the Northern Flicker and Steller's Jay bird species according to their calls. Our multi-class classification model did perform decently and was able to class most of the bird species correctly based on their calls. When we added additional layers and reduced the batch size, it did improve the accuracy of the model, but it also increased the loss.

The limitation which we ran into was the computational cost it takes to run the neural network models on our personnel machines. It took most of the computation power and sometimes the models were crashing as well.

The time it takes to run the multi-class models were almost 15 to 20 minutes for our machines which is more compared to the time it takes to run binary models which is around 45 seconds to a minute.

When we refer to the confusion matrix for our multi-class classification, the most challenging bird species that were difficult to predict were the barn swallow and red-winged blackbird. These two birds were mis-classified more number of times as compared to other species. The barn swallow was mis-judged with house finch for most of the time. There are different characteristics that make any confuse with the bird calls and one common characteristic is the frequency of the sound. Most of the birds have almost the same frequency and the pitch of the sound and this makes it difficult to distinguish between the birds.

We could use other models like a Support Vector Machine which is good in clear separation of margins (in our case the audio spectrograms) and handling noisy data and Random Forest where we can build multiple decision trees to make suitable audio classification for bird calls for different species.

Despite using alternative models such as Random Forests and SVMs, neural networks are ideal for this application as it can automatically learn complex features from high-dimensional data, handle non-linear relationships, and scale with larger datasets.

Conclusion

So, we build and train neural network models to classify the bird species based on their calls. Our models gave us decent results, particularly the binary classifier giving the accuracy of 96.67%. Despite computational challenges, neural networks handled complex audio data effectively, although multi-class models took longer to train. Also, red-winged blackbirds and barn swallows presented significant classification challenges. Even though there are alternative models like Support Vector Machines and Random Forests, neural networks remain advantageous because they can extract features automatically from high-dimensional data, making them suitable for a variety of applications. Overall, this study offers insight into how neural networks can aid environmental monitoring and biodiversity conservation efforts.

References

1. The Bird Recordings dataset from Kaggle. https://www.kaggle.com/datasets/rohanrao/xeno-canto-bird-recordings-extended-a-m, https://www.kaggle.com/datasets/rohanrao/xeno-canto-bird-recordings-extended-n-z

2. Artificial Neural Networks, by Geeks-for-Geeks, uploaded on June 02, 2023. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/

3. Activation Functions in Neural Networks, by Geeks-for-Geeks, uploaded on May 03, 2024. https://www.geeksforgeeks.org/activation-functions-neural-networks/

4. Complete Guide to Neural Networks, by Frederik vom Lehn, published on August 18, 2023. https://medium.com/advanced-deep-learning/complete-guide-to-neural-networks-7eccbc3bbd80

5. Introduction to Convolution Neural Network, by Geeks-for-Geeks, uploaded on March 14, 2024. https://www.geeksforgeeks.org/introduction-convolution-neural-network/

6. Convolutional Neural Networks, published by Mayank Mishra on August 26, 2020. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

7. Artificial Neural Networks, by Premanand S, published on September 22, 2022. https://www.analyticsvidhya.com/blog/2021/06/artificial-neural-networks-better-understanding/

8. An Introduction to Artificial Neural Networks (ANNs), by James Howell, published on February 07, 2024. https://101blockchains.com/artificial-neural-networks-anns/

9. Lecture Materials for Deep Learning Classes.

Appendix

Seattle Bird Call Classification Using Neural Networks

Libraries

In [1]:
```
import numpy as np
import pandas as pd
import os
import h5py
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import librosa
import librosa.display
import IPython.display as ipd
from matplotlib import image
from skimage.transform import resize
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LassoCV
from sklearn.preprocessing import LabelBinarizer, StandardScaler
from sklearn.metrics import mean_absolute_error, classification_report, confusion_matrix
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
from keras.datasets import mnist, cifar10
from keras.utils import to_categorical
from keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import decode_predictions, preprocess_input
from keras.applications import imagenet_utils
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Lambda
from tensorflow.keras.applications.mobilenet import preprocess_input
```

Loading the dataset

In [2]:
```
# h5 file of Bird species data
species_data = h5py.File('spectrograms.h5', 'r')

# Keys for different species
species_keys = list(species_data.keys())

species_keys
```

Out[2]:
```
['amecro',
 'barswa',
 'bkcchi',
 'blujay',
 'daejun',
 'houfin',
 'mallar3',
 'norfli',
 'rewbla',
 'stejay',
 'wesmea',
 'whcspa']
```

In [3]:
```
# Shape of the spectogram data
# Here, we selected one species out of 12 to see the shape
dset = species_data['stejay']

dset.shape
```
Out[3]:
```
(256, 343, 40)
```

Binary Classification Model

We will use Northern Flicker and Steller's Jay bird species for this model

In [4]:
```
# Load data for the selected species
# norfli sound for Northern Flicker data and stejay for Steller's Jay data
norfli_data = species_data['norfli'][:]
stejay_data = species_data['stejay'][:]
```
In [5]:
```
norfli_data.shape
```
Out[5]:
```
(256, 343, 59)
```
In [6]:
```
# Now Let's label the selected species
# Using 0 for Northern Flicker birds and 1 for Steller's Jay birds
norfli_data_labels = np.zeros(norfli_data.shape[2])
stejay_data_labels = np.ones(stejay_data.shape[2])
```
In [7]:
```
# After the above step, combining the data and labels
X_bin = np.concatenate((norfli_data, stejay_data), axis=2)
y_bin = np.concatenate((norfli_data_labels, stejay_data_labels), axis=0)
```
In [8]:
```
# Normalizing the data
X_bin = X_bin / np.max(X_bin)

# Reshaping X to match the expected input shape for CNNs (frequency, timesteps, no. of s
X_bin = np.transpose(X_bin, (2, 0, 1))
X_bin = X_bin[..., np.newaxis]  # This will add a channel dimension
```
In [9]:
```
# Now, Converting the labels to categorical data
y_bin = to_categorical(y_bin, 2)
```
In [10]:
```
# Splitting the data into training and validation sets (70% training set and 30% validat
X_train, X_val, y_train, y_val = train_test_split(X_bin, y_bin, test_size=0.3, random_st
```
In [11]:
```
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)
```
```
(69, 256, 343, 1) (30, 256, 343, 1) (69, 2) (30, 2)
```
In [12]:
```
# Defining the CNN model for our Binary classification
Bin_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 343, 1)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
```

```
    Dense(2, activation='sigmoid')
])
```

```
C:\Users\hirshikesh\anaconda3\lib\site-packages\keras\src\layers\convolutional\base_con
v.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first layer in t
he model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [13]:
```
# Model summary
Bin_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 341, 32)	320
max_pooling2d (MaxPooling2D)	(None, 127, 170, 32)	0
dropout (Dropout)	(None, 127, 170, 32)	0
conv2d_1 (Conv2D)	(None, 125, 168, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 62, 84, 64)	0
dropout_1 (Dropout)	(None, 62, 84, 64)	0
flatten (Flatten)	(None, 333312)	0
dense (Dense)	(None, 128)	42,664,064
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258

Total params: 42,683,138 (162.82 MB)

Trainable params: 42,683,138 (162.82 MB)

Non-trainable params: 0 (0.00 B)

In [14]:
```
# Compiling the model
Bin_model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
```

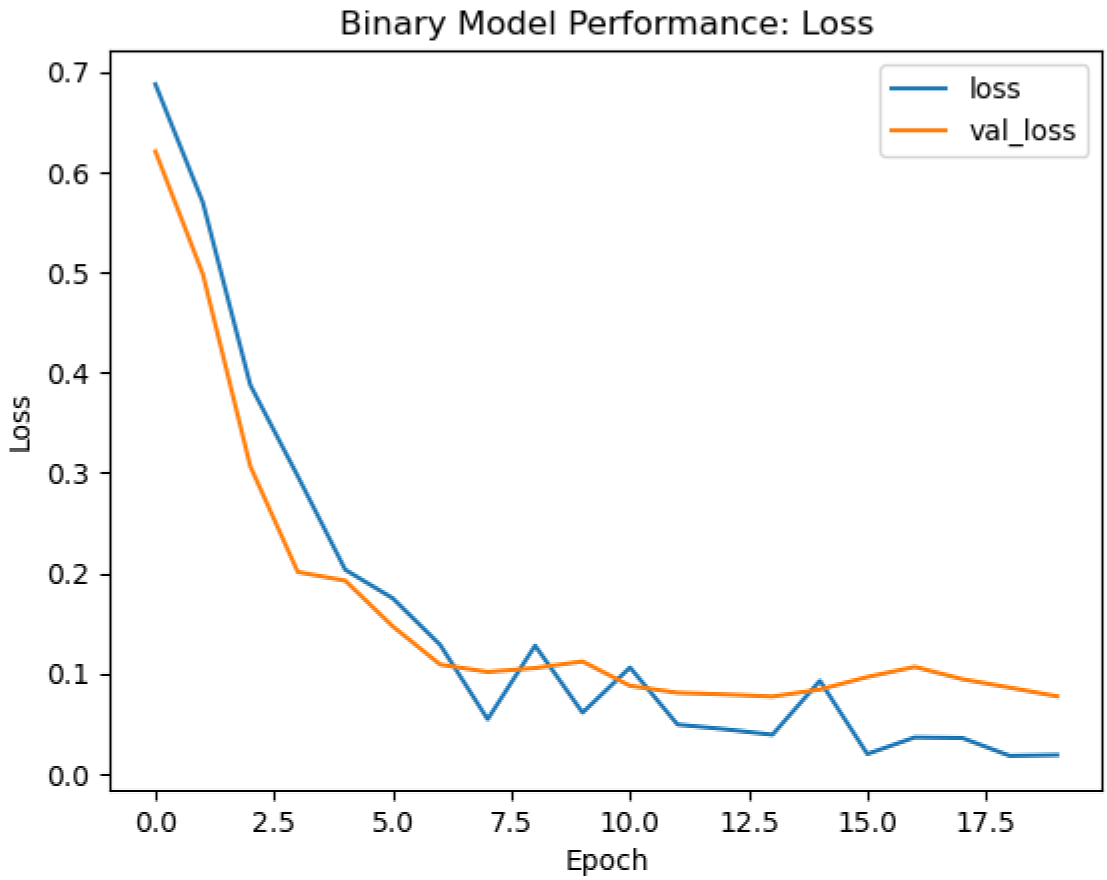
In [15]:
```
# Training the model
history = Bin_model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
```

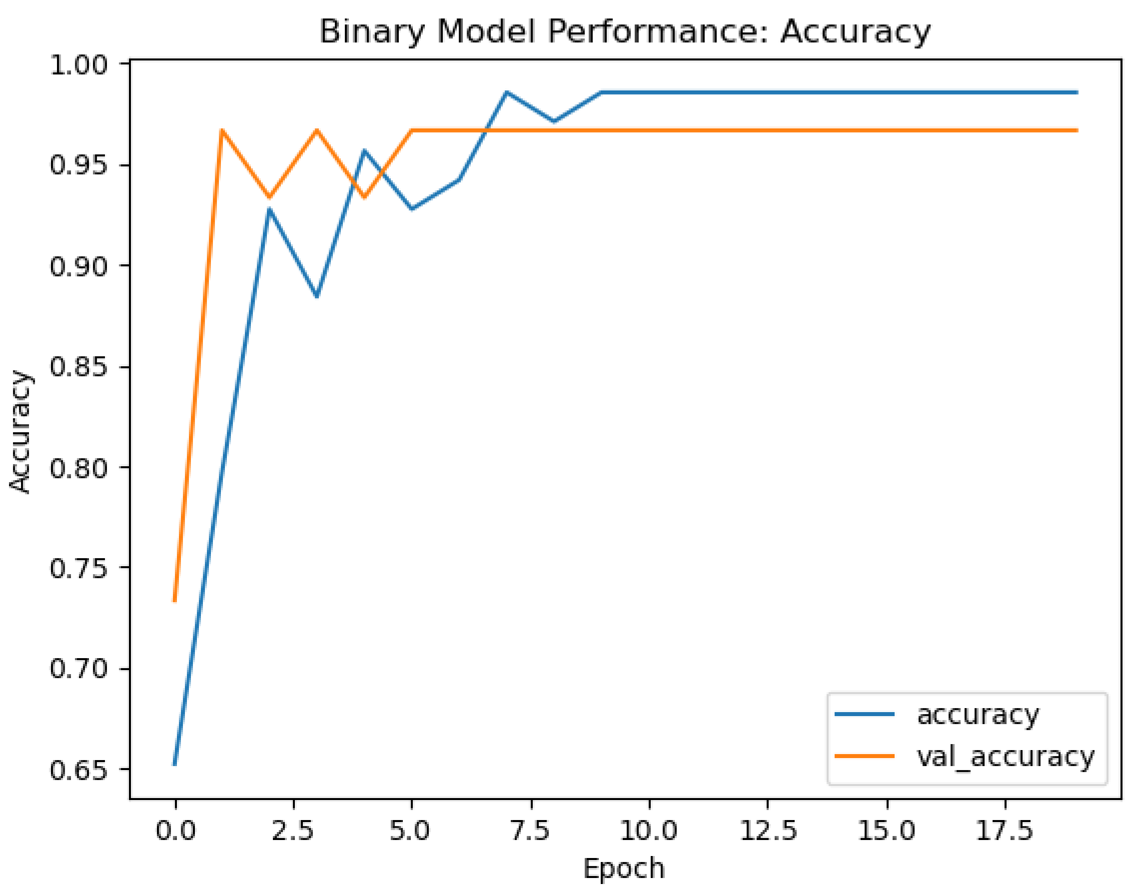
```
Epoch 1/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 630ms/step - accuracy: 0.6542 - loss: 0.6896 -
 val_accuracy: 0.7333 - val_loss: 0.6206
Epoch 2/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 591ms/step - accuracy: 0.7423 - loss: 0.5978 -
 val_accuracy: 0.9667 - val_loss: 0.4982
Epoch 3/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 599ms/step - accuracy: 0.9130 - loss: 0.3986 -
 val_accuracy: 0.9333 - val_loss: 0.3064
Epoch 4/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 578ms/step - accuracy: 0.9186 - loss: 0.2556 -
 val_accuracy: 0.9667 - val_loss: 0.2011
Epoch 5/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 564ms/step - accuracy: 0.9431 - loss: 0.2363 -
 val_accuracy: 0.9333 - val_loss: 0.1925
Epoch 6/20
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 578ms/step - accuracy: 0.9364 - loss: 0.1697 -
 val_accuracy: 0.9667 - val_loss: 0.1470
Epoch 7/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 589ms/step - accuracy: 0.9398 - loss: 0.1364 -
 val_accuracy: 0.9667 - val_loss: 0.1089
Epoch 8/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 591ms/step - accuracy: 0.9810 - loss: 0.0622 -
 val_accuracy: 0.9667 - val_loss: 0.1015
Epoch 9/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 577ms/step - accuracy: 0.9699 - loss: 0.1113 -
 val_accuracy: 0.9667 - val_loss: 0.1054
Epoch 10/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 585ms/step - accuracy: 0.9888 - loss: 0.0523 -
 val_accuracy: 0.9667 - val_loss: 0.1119
Epoch 11/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 583ms/step - accuracy: 0.9888 - loss: 0.0861 -
 val_accuracy: 0.9667 - val_loss: 0.0877
Epoch 12/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 592ms/step - accuracy: 0.9810 - loss: 0.0596 -
 val_accuracy: 0.9667 - val_loss: 0.0807
Epoch 13/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 589ms/step - accuracy: 0.9810 - loss: 0.0550 -
 val_accuracy: 0.9667 - val_loss: 0.0791
Epoch 14/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 569ms/step - accuracy: 0.9810 - loss: 0.0452 -
 val_accuracy: 0.9667 - val_loss: 0.0770
Epoch 15/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 588ms/step - accuracy: 0.9810 - loss: 0.1142 -
 val_accuracy: 0.9667 - val_loss: 0.0840
Epoch 16/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 587ms/step - accuracy: 0.9810 - loss: 0.0241 -
 val_accuracy: 0.9667 - val_loss: 0.0965
Epoch 17/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 585ms/step - accuracy: 0.9888 - loss: 0.0295 -
 val_accuracy: 0.9667 - val_loss: 0.1065
Epoch 18/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 589ms/step - accuracy: 0.9810 - loss: 0.0454 -
 val_accuracy: 0.9667 - val_loss: 0.0943
Epoch 19/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 587ms/step - accuracy: 0.9888 - loss: 0.0151 -
 val_accuracy: 0.9667 - val_loss: 0.0859
Epoch 20/20
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 579ms/step - accuracy: 0.9888 - loss: 0.0154 -
 val_accuracy: 0.9667 - val_loss: 0.0772
```

In [16]:

```python
# Model Performance Plot
# Loss performance
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Binary Model Performance: Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['loss', 'val_loss'], loc='upper right')
plt.show();

# Accuracy performance
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Binary Model Performance: Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['accuracy', 'val_accuracy'], loc='lower right')
plt.show();
```





In [17]:

```
# Evaluating the model on the validation set
val_loss, val_accuracy = Bin_model.evaluate(X_val, y_val)

# Result
print(f'Validation Loss for Binary Model: {val_loss}')
print(f'Validation Accuracy for Binary Model: {val_accuracy}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 130ms/step - accuracy: 0.9667 - loss: 0.0772
```

```
Validation Loss for Binary Model: 0.07719671726226807
Validation Accuracy for Binary Model: 0.9666666388511658
```

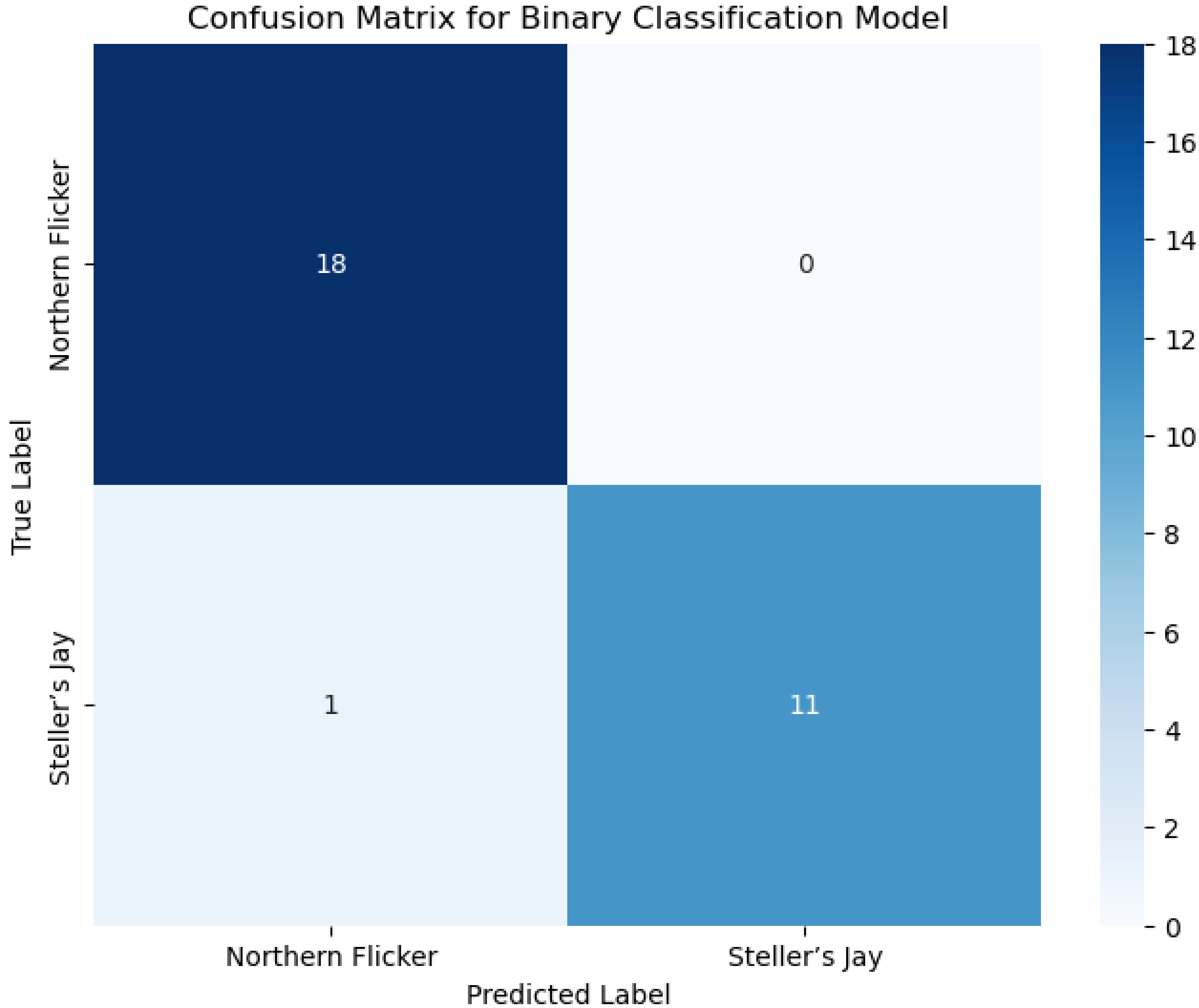
In [18]:
```python
# Predicting new data
predictions = Bin_model.predict(X_val)

# Converting probabilities to class labels
predicted_classes = (predictions > 0.5).astype(int)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
```

In [19]:
```python
# Now, converting labels to integer labels
true_classes = np.argmax(y_val, axis=1)
predicted_classes = (predictions > 0.5).astype(int)
predicted_classes = np.argmax(predicted_classes, axis=1)

# Classification report
report = classification_report(true_classes, predicted_classes, target_names=['Northern
print(report)
```

```
                precision    recall  f1-score   support

Northern Flicker       0.95      1.00      0.97        18
   Steller's Jay       1.00      0.92      0.96        12

        accuracy                           0.97        30
       macro avg       0.97      0.96      0.96        30
    weighted avg       0.97      0.97      0.97        30
```

In [20]:
```python
# Confusion Matrix
matrix = confusion_matrix(true_classes, predicted_classes)

# Heatmap For confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Northern Flicker', 'Steller's Jay'],
            yticklabels=['Northern Flicker', 'Steller's Jay'])
plt.title('Confusion Matrix for Binary Classification Model')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```



Multi-Class Classification Model for all 12 Bird Species

In [21]:
```
# Load data for all species and their labels
# We created a empty lists for data and their labels
# X_multi : data samples for all bird species
# y_multi : labels for all birds species
X_multi = []
y_multi = []

# Now, looping over each species key to load the for all species data
for index, key in enumerate(species_keys):
    data = species_data[key][:]
    # We created a label array to fill the species index
    labels = np.full(data.shape[2], index)
    X_multi.append(data)
    y_multi.append(labels)
```

In [22]:
```
# Now, combining the data and labels from all species
X_multi = np.concatenate(X_multi, axis=2)
y_multi = np.concatenate(y_multi, axis=0)
```

In [23]:
```
# Normalizing the data
X_multi = X_multi / np.max(X_multi)
```

In [24]:
```
# Reshaping X to match the expected input shape for CNNs (frequency, timesteps, samples)
X_multi = np.transpose(X_multi, (2, 0, 1))
X_multi = X_multi[..., np.newaxis]  # this will adds a channel dimension
```

In [25]:
```
# Converting the labels to categorical data
y_multi = to_categorical(y_multi, len(species_keys))
```

In [26]:
```
# Splitting the data into training and validation sets (70% training set and 30% validat
X_train, X_val, y_train, y_val = train_test_split(X_multi, y_multi, test_size=0.3, rando
```

In [27]:
```
# Defining the CNN model for Multi-class classification
Multi_class_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(256, 343, 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(12, activation='softmax')
])
```

In [28]:
```
# Model summary
Multi_class_model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 254, 341, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 127, 170, 32)	0
conv2d_3 (Conv2D)	(None, 125, 168, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 62, 84, 64)	0
flatten_1 (Flatten)	(None, 333312)	0
dense_2 (Dense)	(None, 256)	85,328,128
dropout_3 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 12)	3,084

Total params: 85,350,028 (325.58 MB)

Trainable params: 85,350,028 (325.58 MB)

Non-trainable params: 0 (0.00 B)

In [29]:
```
# Compiling the model
Multi_class_model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['a
```

In [30]:
```
# Training the model
history = Multi_class_model.fit(X_train, y_train, epochs=20, batch_size=128, validation_
```

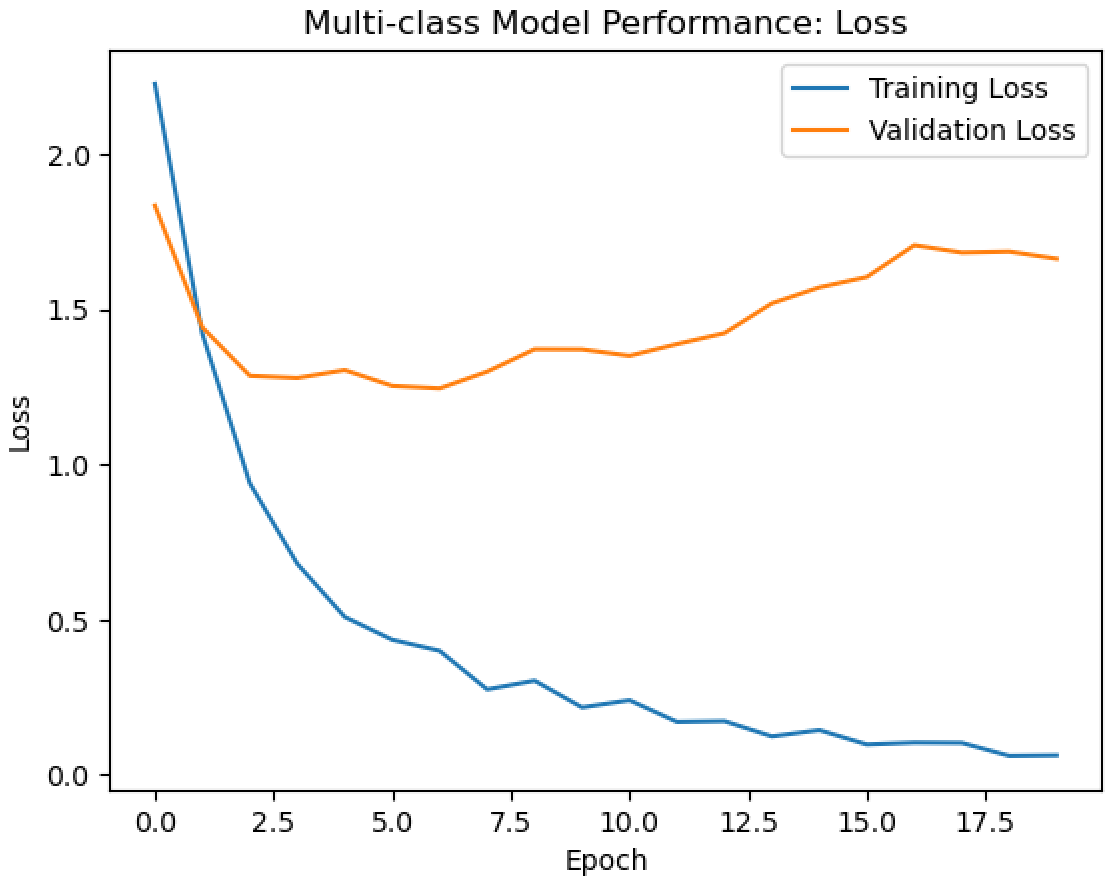
```
Epoch 1/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 3s/step - accuracy: 0.2160 - loss: 2.3056 - v
al_accuracy: 0.3908 - val_loss: 1.8335
Epoch 2/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.5690 - loss: 1.4658 - va
l_accuracy: 0.5632 - val_loss: 1.4413
Epoch 3/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.7315 - loss: 0.9518 - va
```

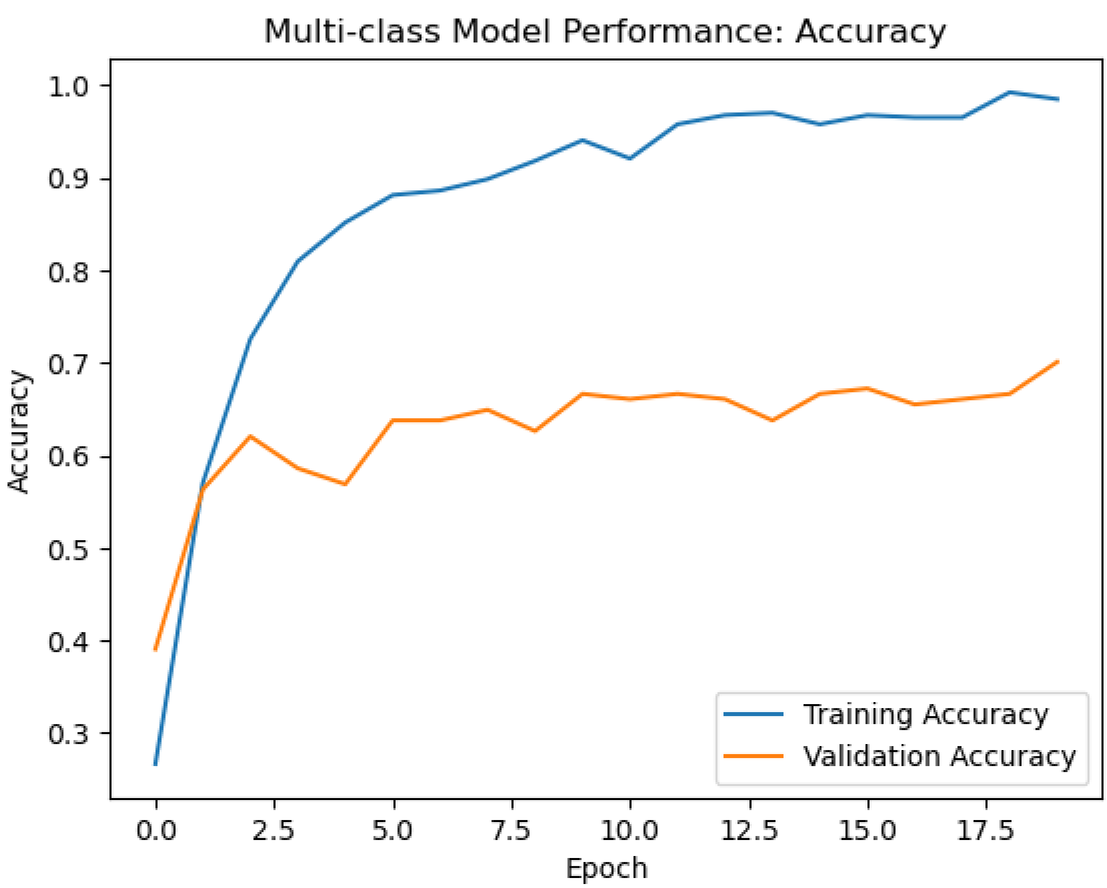
```
l_accuracy: 0.6207 - val_loss: 1.2854
Epoch 4/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8107 - loss: 0.7061 - va
l_accuracy: 0.5862 - val_loss: 1.2784
Epoch 5/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8374 - loss: 0.5553 - va
l_accuracy: 0.5690 - val_loss: 1.3036
Epoch 6/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8760 - loss: 0.4586 - va
l_accuracy: 0.6379 - val_loss: 1.2529
Epoch 7/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8879 - loss: 0.3946 - va
l_accuracy: 0.6379 - val_loss: 1.2448
Epoch 8/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9017 - loss: 0.2709 - va
l_accuracy: 0.6494 - val_loss: 1.2986
Epoch 9/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9174 - loss: 0.3122 - va
l_accuracy: 0.6264 - val_loss: 1.3708
Epoch 10/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9378 - loss: 0.2284 - va
l_accuracy: 0.6667 - val_loss: 1.3702
Epoch 11/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9267 - loss: 0.2211 - va
l_accuracy: 0.6609 - val_loss: 1.3496
Epoch 12/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9553 - loss: 0.1605 - va
l_accuracy: 0.6667 - val_loss: 1.3874
Epoch 13/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9700 - loss: 0.1815 - va
l_accuracy: 0.6609 - val_loss: 1.4226
Epoch 14/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9676 - loss: 0.1310 - va
l_accuracy: 0.6379 - val_loss: 1.5187
Epoch 15/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9587 - loss: 0.1405 - va
l_accuracy: 0.6667 - val_loss: 1.5700
Epoch 16/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9684 - loss: 0.0982 - va
l_accuracy: 0.6724 - val_loss: 1.6036
Epoch 17/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9612 - loss: 0.1144 - va
l_accuracy: 0.6552 - val_loss: 1.7058
Epoch 18/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9659 - loss: 0.0952 - va
l_accuracy: 0.6609 - val_loss: 1.6820
Epoch 19/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9900 - loss: 0.0659 - va
l_accuracy: 0.6667 - val_loss: 1.6854
Epoch 20/20
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9860 - loss: 0.0558 - va
l_accuracy: 0.7011 - val_loss: 1.6623
```

In [31]:

```python
# Model Performance Plot for Loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Multi-class Model Performance: Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()

# Model Performance Plot for Accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```python
plt.title('Multi-class Model Performance: Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
plt.show()
```





In [32]:
```python
# Evaluating the model on the validation set
val_loss, val_accuracy = Multi_class_model.evaluate(X_val, y_val)
```

```
# Result
print(f'Validation Loss for Multi-class Model: {val_loss}')
print(f'Validation Accuracy for Multi-class Model: {val_accuracy}')
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 127ms/step - accuracy: 0.6834 - loss: 1.6372
Validation Loss for Multi-class Model: 1.6622616052627563
Validation Accuracy for Multi-class Model: 0.7011494040489197
```

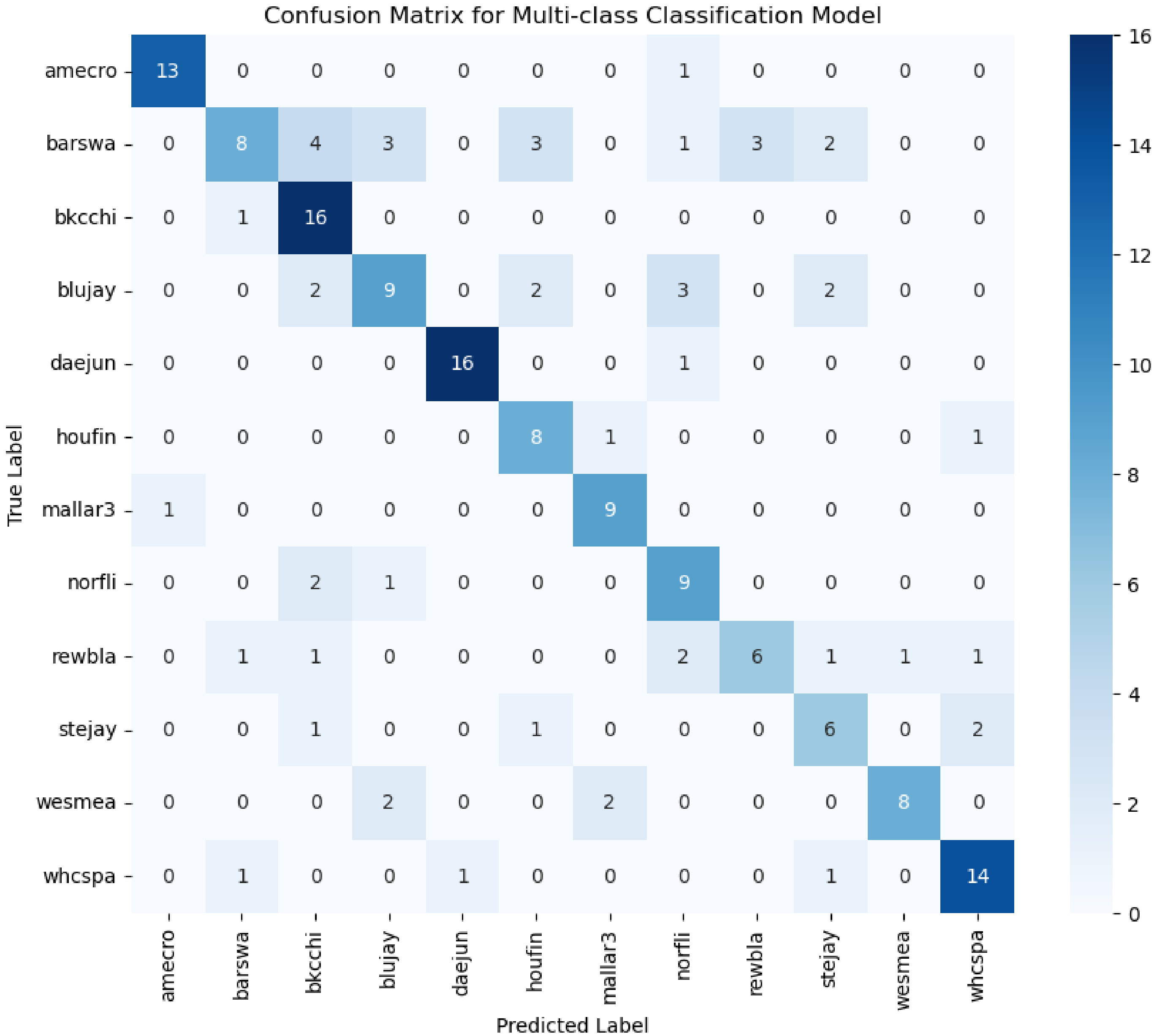
In [33]:
```
# Predicting new data
predictions = Multi_class_model.predict(X_val)

# Converting probabilities to class labels
predicted_classes = np.argmax(predictions, axis=1)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 129ms/step
```

In [34]:
```
# Now, converting labels to integer labels
true_classes = np.argmax(y_val, axis=1)

# Classification report for multi-class model
report = classification_report(true_classes, predicted_classes, target_names = species_k
print(report)
```

```
              precision    recall  f1-score   support

      amecro       0.93      0.93      0.93        14
      barswa       0.73      0.33      0.46        24
      bkcchi       0.62      0.94      0.74        17
      blujay       0.60      0.50      0.55        18
      daejun       0.94      0.94      0.94        17
      houfin       0.57      0.80      0.67        10
     mallar3       0.75      0.90      0.82        10
      norfli       0.53      0.75      0.62        12
      rewbla       0.67      0.46      0.55        13
      stejay       0.50      0.60      0.55        10
      wesmea       0.89      0.67      0.76        12
      whcspa       0.78      0.82      0.80        17

    accuracy                           0.70       174
   macro avg       0.71      0.72      0.70       174
weighted avg       0.72      0.70      0.69       174
```

In [35]:
```
# Confusion Matrix for multi-class model
matrix = confusion_matrix(true_classes, predicted_classes)

# Heatmap For confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=species_keys,
            yticklabels=species_keys)
plt.title('Confusion Matrix for Multi-class Classification Model')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```



In [36]:
```python
# Defining the 2nd CNN model for Multi-class classification
Multi_class_model2 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(256, 343, 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(128, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(256, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(12, activation='softmax')
])
```

In [37]:
```python
# Model summary
Multi_class_model2.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #

conv2d_4 (Conv2D)	(None, 254, 341, 32)	320
max_pooling2d_4 (MaxPooling2D)	(None, 127, 170, 32)	0
conv2d_5 (Conv2D)	(None, 125, 168, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 62, 84, 64)	0
conv2d_6 (Conv2D)	(None, 60, 82, 128)	73,856
max_pooling2d_6 (MaxPooling2D)	(None, 30, 41, 128)	0
conv2d_7 (Conv2D)	(None, 28, 39, 256)	295,168
max_pooling2d_7 (MaxPooling2D)	(None, 14, 19, 256)	0
flatten_2 (Flatten)	(None, 68096)	0
dense_4 (Dense)	(None, 256)	17,432,832
dropout_4 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 12)	3,084

Total params: 17,823,756 (67.99 MB)

Trainable params: 17,823,756 (67.99 MB)

Non-trainable params: 0 (0.00 B)

In [38]:
```
# Compiling the model
Multi_class_model2.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['
```

In [39]:
```
# Training the model
history2 = Multi_class_model2.fit(X_train, y_train, epochs=30, batch_size=64, validation
```

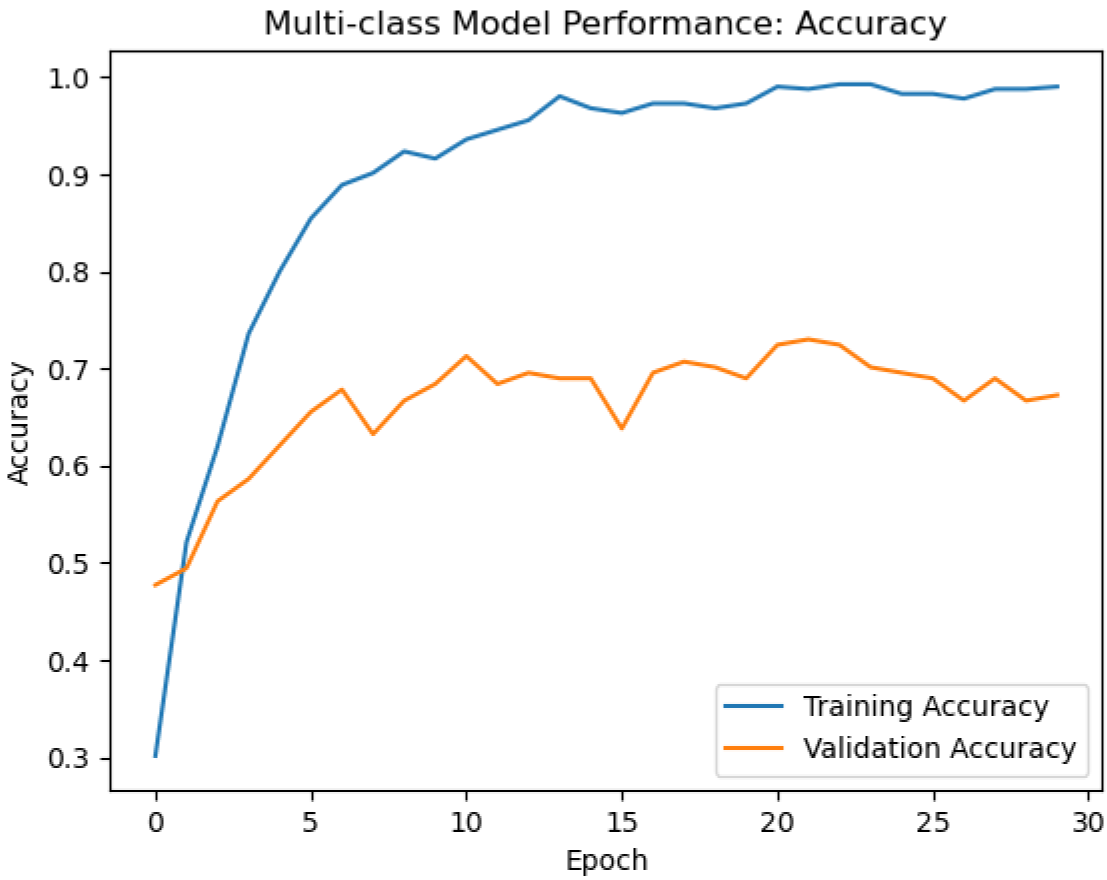
```
Epoch 1/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.2316 - loss: 2.2958 - v
al_accuracy: 0.4770 - val_loss: 1.7246
Epoch 2/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.5040 - loss: 1.6057 - v
al_accuracy: 0.4943 - val_loss: 1.4462
Epoch 3/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.5891 - loss: 1.1873 - v
al_accuracy: 0.5632 - val_loss: 1.3274
Epoch 4/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.7353 - loss: 0.8904 - v
al_accuracy: 0.5862 - val_loss: 1.2149
Epoch 5/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.7833 - loss: 0.6365 - v
al_accuracy: 0.6207 - val_loss: 1.2232
Epoch 6/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.8751 - loss: 0.4586 - v
al_accuracy: 0.6552 - val_loss: 1.2980
Epoch 7/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.8914 - loss: 0.3801 - v
al_accuracy: 0.6782 - val_loss: 1.3552
Epoch 8/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.8949 - loss: 0.3602 - v
al_accuracy: 0.6322 - val_loss: 1.5987
Epoch 9/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9228 - loss: 0.2966 - v
al_accuracy: 0.6667 - val_loss: 1.6180
Epoch 10/30
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9102 - loss: 0.2689 - v
al_accuracy: 0.6839 - val_loss: 1.4741
Epoch 11/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.9461 - loss: 0.2141 - v
al_accuracy: 0.7126 - val_loss: 1.6364
Epoch 12/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9449 - loss: 0.1678 - v
al_accuracy: 0.6839 - val_loss: 1.9117
Epoch 13/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.9541 - loss: 0.1289 - v
al_accuracy: 0.6954 - val_loss: 2.1707
Epoch 14/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9870 - loss: 0.0925 - v
al_accuracy: 0.6897 - val_loss: 2.2550
Epoch 15/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.9681 - loss: 0.1350 - v
al_accuracy: 0.6897 - val_loss: 2.2391
Epoch 16/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 1s/step - accuracy: 0.9653 - loss: 0.0990 - v
al_accuracy: 0.6379 - val_loss: 2.4522
Epoch 17/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9645 - loss: 0.1239 - v
al_accuracy: 0.6954 - val_loss: 2.0301
Epoch 18/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9744 - loss: 0.0984 - v
al_accuracy: 0.7069 - val_loss: 1.9972
Epoch 19/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9656 - loss: 0.1064 - v
al_accuracy: 0.7011 - val_loss: 2.0504
Epoch 20/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.9694 - loss: 0.0698 - v
al_accuracy: 0.6897 - val_loss: 2.1285
Epoch 21/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9849 - loss: 0.0465 - v
al_accuracy: 0.7241 - val_loss: 2.1781
Epoch 22/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9880 - loss: 0.0479 - v
al_accuracy: 0.7299 - val_loss: 2.3860
Epoch 23/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9912 - loss: 0.0298 - v
al_accuracy: 0.7241 - val_loss: 2.6870
Epoch 24/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9903 - loss: 0.0595 - v
al_accuracy: 0.7011 - val_loss: 2.7997
Epoch 25/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9898 - loss: 0.0383 - v
al_accuracy: 0.6954 - val_loss: 2.9740
Epoch 26/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.9782 - loss: 0.1390 - v
al_accuracy: 0.6897 - val_loss: 2.5107
Epoch 27/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - accuracy: 0.9773 - loss: 0.0799 - v
al_accuracy: 0.6667 - val_loss: 2.6212
Epoch 28/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.9854 - loss: 0.0429 - v
al_accuracy: 0.6897 - val_loss: 2.7523
Epoch 29/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.9896 - loss: 0.0493 - v
al_accuracy: 0.6667 - val_loss: 2.8336
Epoch 30/30
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9869 - loss: 0.0573 - v
al_accuracy: 0.6724 - val_loss: 2.9570
```

In [40]:

```
# Model Performance Plot for Loss
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
```

```python
plt.title('Multi-class Model Performance: Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training Loss', 'Validation Loss'], loc='upper right')
plt.show()

# Model Performance Plot for Accuracy
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Multi-class Model Performance: Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
plt.show()
```



In [41]:
```
# Evaluating the model on the validation set
val_loss2, val_accuracy2 = Multi_class_model2.evaluate(X_val, y_val)

# Result
print(f'Validation Loss for Multi-class Model2: {val_loss2}')
print(f'Validation Accuracy for Multi-class Model2: {val_accuracy2}')
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 168ms/step - accuracy: 0.6759 - loss: 3.1402
Validation Loss for Multi-class Model2: 2.9570038318634033
Validation Accuracy for Multi-class Model2: 0.6724137663841248
```

In [42]:
```
# Predicting new data
predictions2 = Multi_class_model2.predict(X_val)

# Converting probabilities to class labels
predicted_classes2 = np.argmax(predictions2, axis=1)
```

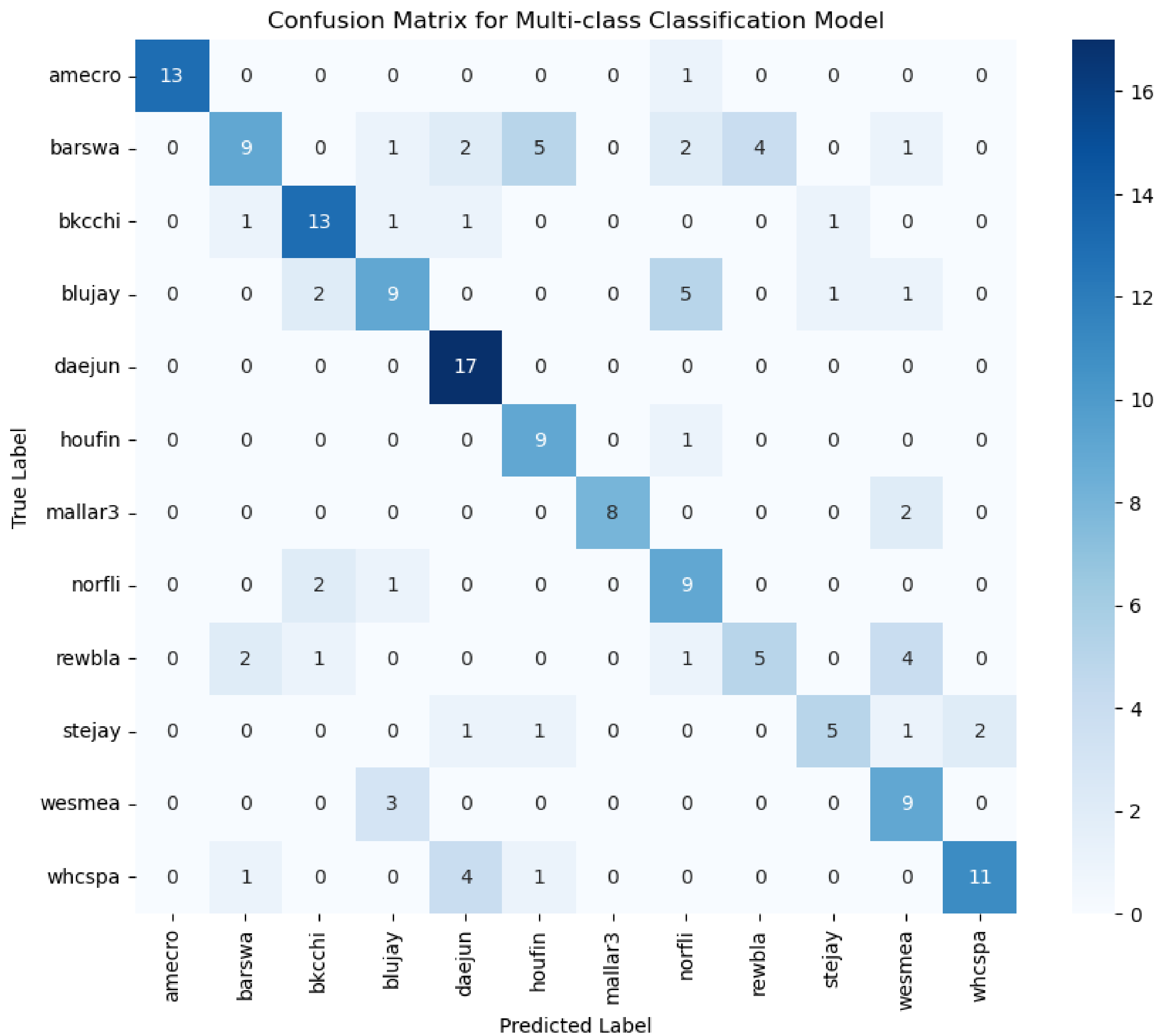
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 186ms/step
```

In [43]:
```
# Now, converting labels to integer labels
true_classes2 = np.argmax(y_val, axis=1)

# Classification report for multi-class model
report2 = classification_report(true_classes2, predicted_classes2, target_names = specie
print(report)
```

```
              precision    recall  f1-score   support

     amecro       0.93      0.93      0.93        14
     barswa       0.73      0.33      0.46        24
     bkcchi       0.62      0.94      0.74        17
     blujay       0.60      0.50      0.55        18
     daejun       0.94      0.94      0.94        17
     houfin       0.57      0.80      0.67        10
    mallar3       0.75      0.90      0.82        10
     norfli       0.53      0.75      0.62        12
     rewbla       0.67      0.46      0.55        13
```

```
      stejay       0.50      0.60      0.55        10
      wesmea       0.89      0.67      0.76        12
      whcspa       0.78      0.82      0.80        17

    accuracy                           0.70       174
   macro avg       0.71      0.72      0.70       174
weighted avg       0.72      0.70      0.69       174
```

In [44]:
```
# Confusion Matrix for multi-class model
matrix2 = confusion_matrix(true_classes2, predicted_classes2)

# Heatmap For confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(matrix2, annot=True, fmt="d", cmap="Blues",
            xticklabels=species_keys,
            yticklabels=species_keys)
plt.title('Confusion Matrix for Multi-class Classification Model')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```



Multi-Class Model on External Test Data

In [52]:
```
# Function to preprocess the test audio files and generate spectrograms for the same
def preprocess_test_clips(clip_path):
    y, sr = librosa.load(clip_path, sr=None)
    y = librosa.resample(y, orig_sr=sr, target_sr=22050)
    intervals = librosa.effects.split(y, top_db=20)
    clips = [y[start:end] for start, end in intervals if (end - start) > 0.5 * sr]

    spectrograms = []
    for clip in clips:
        if len(clip) > 2 * sr:
            for i in range(0, len(clip) - 2 * sr, 2 * sr):
                window = clip[i:i + 2 * sr]
                S = librosa.feature.melspectrogram(y=window, sr=22050, n_mels=256, fmax=
                S_dB = librosa.power_to_db(S, ref=np.max)
                S_dB_resized = resize(S_dB, (256, 343))  # Resize to match input shape o
                spectrograms.append(S_dB_resized)
        else:
            window = librosa.util.fix_length(clip, 2 * sr)
            S = librosa.feature.melspectrogram(y=window, sr=22050, n_mels=256, fmax=8000
            S_dB = librosa.power_to_db(S, ref=np.max)
            S_dB_resized = resize(S_dB, (256, 343))  # Resize to match input shape of th
            spectrograms.append(S_dB_resized)

    return spectrograms
```

In [53]:
```
# Function to predict bird species from the test spectrograms
def predict(spectrograms, model, species_keys):
    predictions = []
    for spectrogram in spectrograms:
        spectrogram = np.expand_dims(spectrogram, axis=0)  # Adding the  batch dimension
        spectrogram = np.expand_dims(spectrogram, axis=-1)  # This is to add channel dim
        prediction = model.predict(spectrogram)
        predicted_class = np.argmax(prediction, axis=1)[0]
        predictions.append((species_keys[predicted_class], prediction[0]))
    return predictions
```

In [54]:
```
# Function to analyze predictions and format output
def analyze_preds(predictions, species_keys):
    detailed_predictions = []
    for species, probs in predictions:
        sorted_probs = sorted(zip(species_keys, probs), key=lambda x: x[1], reverse=True
        top_preds = sorted_probs[:3]
        detailed_predictions.append(top_preds)
    return detailed_predictions
```

In [55]:
```
# Load the trained model
model = Multi_class_model2

# Directory containing the test clips
test_clips = 'test_birds/'

# Processing each test file and storing the results
test_files = [os.path.join(test_clips, f) for f in os.listdir(test_clips) if f.endswith(
results = []

for test_file in test_files:
    spectrograms = preprocess_test_clips(test_file)
    predictions = predict(spectrograms, model, species_keys)
    detailed_predictions = analyze_preds(predictions, species_keys)

    results.append({
        'File': test_file,
        'Predictions': detailed_predictions
    })
```

```
# Displaying results in a table
results_df = pd.DataFrame(results)
print(results_df)
print("\n")

# Checking the presence of multiple bird calls in each clip
for index, row in results_df.iterrows():
    multiple_bird_call = False
    for prediction in row['Predictions']:
        if len(prediction) > 1 and prediction[1][1] > 0.5:
            multiple_bird_call = True
            break
    if multiple_bird_call:
        print(f"{row['File']} may contain more than one bird call.")
    else:
        print(f"{row['File']} is likely to contain single bird call.")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
                     File                                        Predictions
0  test_birds/test1.mp3  [[(daejun, 1.0), (amecro, 0.0), (barswa, 0.0)]...
1  test_birds/test2.mp3    [[(daejun, 1.0), (amecro, 0.0), (barswa, 0.0)]]
2  test_birds/test3.mp3  [[(daejun, 1.0), (amecro, 0.0), (barswa, 0.0)]...


test_birds/test1.mp3 is likely to contain single bird call.
test_birds/test2.mp3 is likely to contain single bird call.
test_birds/test3.mp3 is likely to contain single bird call.
```

References

- 1) Lecture Notes
- 2) Ch10-1 and Ch10-2 Kereas files from the lecture materials
- 3) Keras and Tensorflow documentation: https://www.tensorflow.org/guide
- 4) Audio Classification model of CNN for constructing spectograms: https://github.com/jeffprosise/Deep-Learning/blob/master/Audio%20Classification%20(CNN).ipynb
- 5) Librosa documentation: https://librosa.org/doc/latest/index.html
- 6) Librosa API reference: https://librosa.org/doc/latest/api.html