# BIRD SPECIES CLASSIFICATION

## ABSTRACT

This study was to predict the sounds of bird species by applying neural networks. The dataset is collected from Birdcall competition data which is originally Xeno-Canto a crowd-sourced bird sound archive. The dataset contains sounds of 12 bird species each containing 10 sound clips. This data has been preprocessed and converted into spectrograms. Spectrograms is a graphical representation of audio of frequencies that varies over time. In binary model any two bird species are selected from the data to build two models of Convolutional neural network (CNN). The best accuracy score for binary model is 100%. Similarly, in multi-class model all the 12 bird species are used for building a CNN models. Then best accuracy score for multi-class model is 68.39%. Three external raw sound clips which may contain sounds of multiple birds are predicted using the best accuracy of multi-class model. From the outcome it has been proved that there are multiple bird species call in each test clips.

## INTRODUCTION

Classification of bird species based on their sound is one of the most significant tasks. It would be helpful in identifying what kind of bird species are calling. In this study, neural networks will be used for classification of bird species based on their sound. Neural networks is a method in deep learning which is similar to the structure of human brain that teaches computer in processing data like text, audio, images etc. for classifying. The main objective of this study is to train the neural networks to predict the species of bird based on its call using the audio clips data. Binary model and multi-class model are used for building neural networks. External test data with three test clips were provided to predict the bird species based on their sound using multi-class model.

### Dataset Description

The original data contains audio clips of 12 bird species sound each containing 10 audio clips. The sounds of birds species in the data are American Crow (amecro), Barn Swallow (barswa), Black-capped Chickadee (bkcchi), Blue Jay(blujay), Dark-eyed Junco (daejun), House Finch (houfin), Mallard (mallar3), Northern Flicker (norfli), Red-winged Blackbird (rewbla), Stellar's Jay (stejay), Western Meadowlark (wesmea) and White-crowned Sparrow (whcspa). [2]

## THEORETICAL BACKGROUND

Deep Learning is considered as the subset of Machine Learning that uses neural networks for solving and building models. The neural network model is similar to the structure and functioning human brain which contains multiple layers of neural network interconnected. Neural networks works by taking input and passing it to multiple layers of networks, weighting them up and producing predicted output with the help of activation function.

An activation function is a mathematical function that get applied to the output of a neuron in the neural network. These functions introduce nonlinearity, and they help the network discover complex dependencies, and maps from the input to the output non-linearly. If activation functions are not used in a neural network, the network turns to be a simple linear function provoking the inability to learn complex patterns in data. [4]

Some Activation Functions are:

**Sigmoid**: The sigmoid activation function scales the values from 0 to 1. This function is mainly used in the output layer of many binary classification problems since it results to outputs that are in the probability form.

**Softmax**: Softmax activation function is applied at the output layer of the neural network for multi-class learning task. It generates its outputs as a probability distribution on a set of multiple classes.

**Rectified Linear Unit (ReLu)**: Rectified Linear Unit is still one of the most used activation functions. It makes them positive by setting negative value to zero which makes it efficient for deep networks computation. [4]

**Convolutional Neural Networks (CNN)**

Convolution neural network is one of the type of neural network which is used in this study for processing image or spectrogram data for classification. CNN architecture contains Convolution layer, pooling layer, and fully connected layer.

**Convolution layer**: A convolution Layer accepts a set of weights and then computes the dot product between two matrices where in this case, one matrix is called a kernel while the other matrix is a small portion of the input field. Every kernel applies the weights to different values of the inputs. The total inputs are summed, and this gives a value for the position of each kernel. The convolution maps are passed through a nonlinear activation layer such as Rectified Linear Unit, which removes negative numbers of the filtered image.

**Pooling Layer**: Pooling is useful in reducing the size of the representation in terms of space and hence reducing the computation and weights needed. The pooling operation is performed independently on each slice of the representation. Pooling layers help control overfitting by reducing the number of calculations and parameters in the network. The most popular process in pooling is max pooling, which gives the maximum output from the neighborhood.

**Fully Connected Layer :** After several iterations of convolution and pooling layers at the end of the network there is a fully connected layer. In fully connected layers an input vector is given in the form of flattened pixels of the given image which is already filtered and downsized by the convolution and pooling layers used. This Fully Connected layer focuses on the representation of the output from the inputs. The softmax or sigmoid activation function is applied at the last layer to the fully connected layers outputs to obtain the probability of a class of the image.[3]

Regularization technique such as Dropout is used in neural network model between the layers to prevent overfitting in model and to give the better performance.

The neural networks are compiled using compilation techniques optimisers, loss and metrics. 'rmsprop' is used as optimizer, binary crossentropy for binary models and categorical crossentropy for multi-class model is used as loss function which is used to calculate error rate of the predicted output. Accuracy is used as metrics to calculate the training and test accuracy score to check how well the model performs.

Hyperparameters like epochs, batch size values are given while fitting the model. Epoch is the number of times the entire training dataset is passed through the neural network and Batch size is the number of training samples that are fed to neural networks at once. These hyperparameters when used can significantly affect the performance and speed of model during training process.[5]

## METHODOLOGY

### Data Exploration and Preprocessing

The original data contains audio clips of 12 bird species sound each containing 10 audio clips. These sound clips are then preprocessed which involves where the sound clip is subsampled to half its sample rate 22050 Hz. The loud parts of the sound clip which are greater than 0.5 seconds are identified, and 2-second windows of sound are selected where a bird call is detected. A spectrogram is produced for each 2-second window. All bird calls for all clips in each species are saved individually and this results in an uneven number of samples in each species. The image size for all the spectrograms is recorded as 343 (time), 256(frequency). This preprocessed data is then saved in a hdf5 format file and this file is then loaded to view the data and used for model building.

### Model Building

Firstly, starting with binary classification any of two bird species from the data are selected which are American crow (amecro) and Bluejay (blujay). These two bird species data are then added into a empty list and then labels were also created and added into empty list. In labelling, 'amecro' is labelled as 0 and 'blujay' is labelled as 1. The two bird species data and labels are splitted into training and testing with the ratio of 70:30. For binary model, two CNN model architectures are created with input shape of (343,256,1) where 343,356 is the size of spectrogram and 1 is channel dimension. The first model consisting of a CNN architecture contains three convolutional layers of kernels 32, 64, 128 with matrix size of 3x3 along with max pooling layers of matrix size 2x2, a flatten layer and two fully connected layers one for sigmoid activation for generating probabilities of each class and another for classifying data. Then, the second model is an extension of Model 1, incorporating a dropout layer of value 0.5 between the flatten and fully connected layers. Both models were compiled using the rmsprop optimizer, binary cross-entropy loss, and accuracy metric. The first model is fitted with batch size of 32 and 10 epochs and the second model is fitted with batch size of 32 and 20 epochs.

In multi-class model the neural network is build for all the 12 bird species. Similar, to binary class all the 12 bird species data is added into list and then labels for all these species are also created and then added into the list. In labelling 'amecro' is labelled as 0, 'barswa' : 1, 'bkcchi' : 2, 'blujay' : 3, 'daejun' : 4, 'houfin' : 5, 'mallar3' : 6, 'norfli' : 7, 'rewbla' : 8, 'stejay' : 9, 'wesmea' : 10, 'whcspa' : 11. The birds data list and labels data are splitted into training and testing with the ratio of 70:30. After splitting the data into training and testing, this data is normalized using Standard scaler. For multi-class model, two CNN model architectures are created with input shape of (343,256,1) where 343,356 is the size of spectrogram and 1 is channel dimension. The first model consisting of a CNN architecture contains three convolutional layers of kernels 32, 64, 128 with matrix size of 3x3 along with max pooling layers of matrix size 2x2, a flatten layer and two fully connected layers one for sigmoid activation for generating probabilities of each class and another for classifying data. Then, the second model is an extension of Model 1, incorporating a dropout layer of value 0.5 between the flatten and fully connected layers. These two models were then compiled using the rmsprop optimizer, categorical cross-entropy loss, and accuracy metric. The first model is fitted with batch size of 32 and 10 epochs and the second model is fitted with batch size of 32 and 20 epochs.

## Data Testing

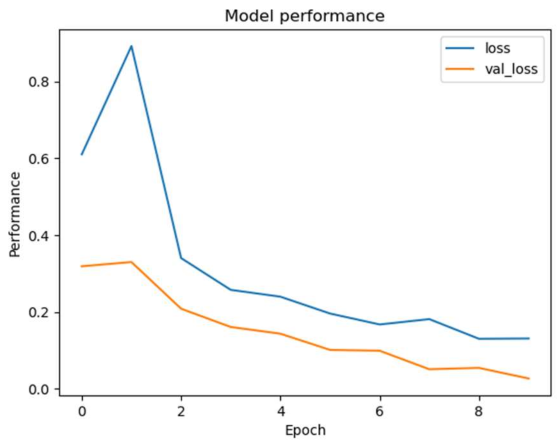
To predict the species of bird based on it call, external test data which contains three test clips of raw sound. These test clips are then converted into spectrograms using the same methodology that is done for original data while preprocessing. The final shape of spectrogram is resulted in the size of 343 (time), 256 (frequency). All the list of spectrograms are then converted into a list of array. Then, multi-class model of 12 species network used for predicting which birds are calling. By applying the multi-class model, it produced the probability distribution for 12 possible species of each spectrogram. Thus, by identifying the most probable indices, these indices were converted to species names with the help of dictionary. The possible species were printed for each clip, providing a clear view of the list of potential species in the clips.

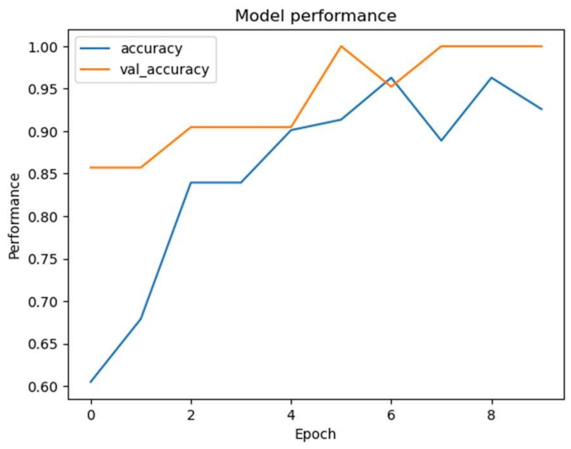
## COMPUTATIONAL RESULTS

### Binary Model

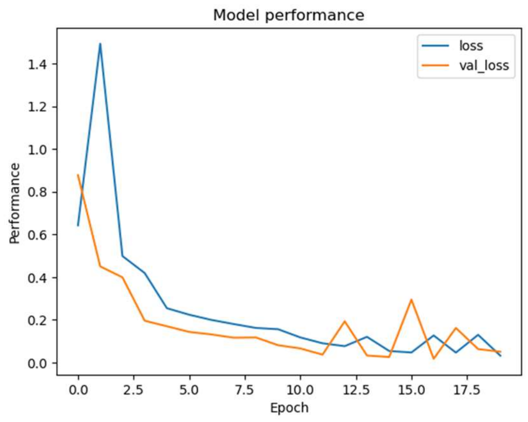
| | Training Accuracy | Test Accuracy | Train Loss | Test Loss |
|---|---|---|---|---|
| **Model 1** | 95.06% | 100% | 10.55% | 2.63% |
| **Model 2** | 100% | 95.24% | 2.30% | 5.10% |

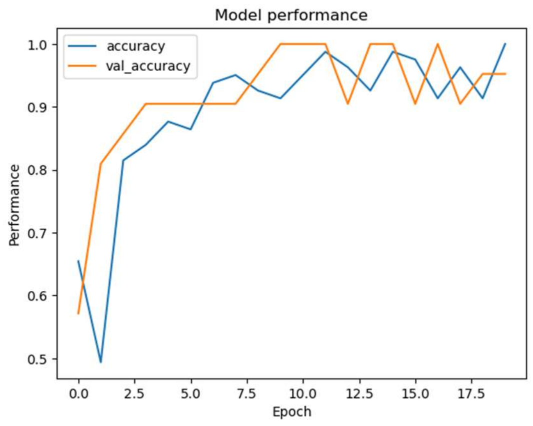
### Model 1





### Model 2 (with dropout)

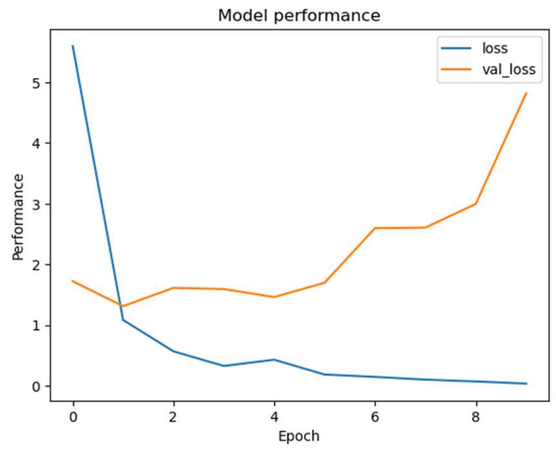


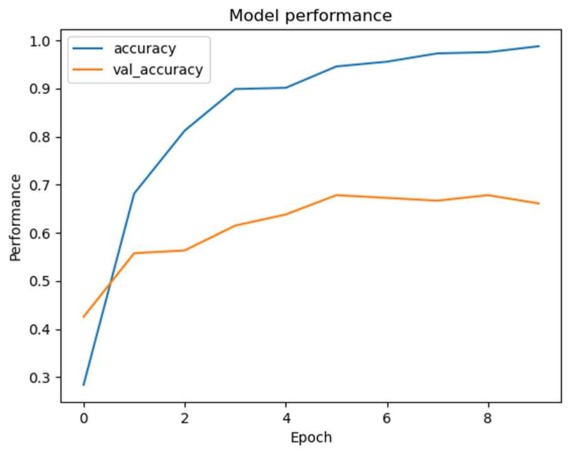


## Multi-class Model

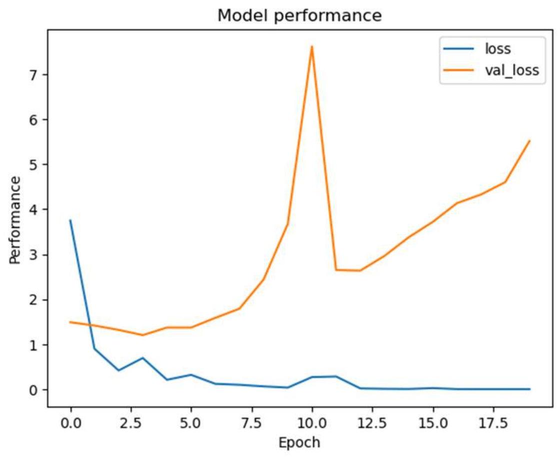
| | Training Accuracy | Test Accuracy | Training Loss | Test Loss |
|---|---|---|---|---|
| **Model 1** | 99.01% | 66.09% | 0.03% | 4.81% |
| **Model 2** | 100% | 68.39% | 0.00% | 5.51% |

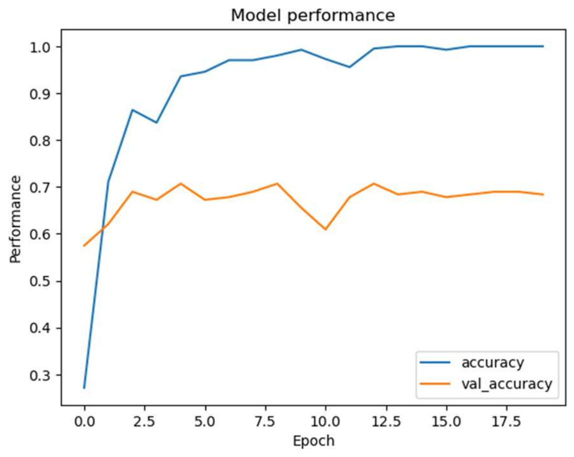
## Model 1





## Model 2 (using dropout)





**External Test Data**

**Test 1**

| **Species** | **Probability** |
|---|---|
| American Crow (amecro) | 52.60% |
| Northern Flicker (norfli) | 45.98% |

**Test 2**

| **Species** | **Probability** |
|---|---|
| American Crow (amecro) | 51.60% |
| Northern Flicker (norfli) | 47.85% |

**Test 3**

| **Species** | **Probability** |
|---|---|
| Northern Flicker (norfli) | 57.29% |
| American Crow (amecro) | 42.28% |

The three test clips are predicted with multi-class model of 12 bird species. From these test results it has been proved that there are multiple species in each test clip based on probability value. The species predicted in Test 1, Test 2 and Test 3 clips are American Crow and Northern Flicker.

## DISCUSSION

In Binary model, both the models have been evaluated based on their training and testing accuracy. The first model has a training accuracy of 95.06%, test accuracy of 100% and a loss of 2.63%. The use of dropout regularization in the second model helps in preventing overfitting and improves the performance of the model. Now, the second model has a training accuracy of 100%, test accuracy of 95.24% and the loss of 5.10%. Compared with these two models, both the models have the same test accuracy scores and have same performance.

In multi-class model the two models, have been evaluated based on accuracy and loss of training and test data. The first model has a training accuracy of 99.01%, test accuracy of 66.09% and loss of 4.81%. The use of dropout regularization in the second model helps in preventing if the model is overfitting and improves the performance of the model. Now, the second model has a training accuracy of 100%, test accuracy of 68.39% and loss of 5.51%. Compared with these two models, both the models have almost similar test accuracy scores, and both these models have same performance.

**What limitations did you run into in this homework? How long did it take to train the models?**
Preprocessing the audio clip data into spectrograms was the most confusing and challenging part. The main limitations which we ran into was the computational time which had taken more while running the models. So, to get the good computational time we require more RAM storage in the systems.

**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?**

The characteristics of bird call depend on its pitch, frequency, and time. From spectrograms it can be observed that Black-capped Chickadee (bkchhi) and Dark-eyed Junco (daejun) have almost same frequency which makes it difficult to predict.

**What other models could we have used to perform this task? Why would a neural network make sense for this application?**

Models like Support Vector Machine (SVM), random forest can be used for classification of audio species. Neural networks models makes sense for this application because neural network can easily extract features from high-dimensional data such as images, spectrograms, audio and other similar data automatically.

## CONCLUSION

The study is mainly focused on training neural networks and predicting the species of bird based on it call from the audio clips. The audio clips are converted into spectrogram data which resulted in size of 343 (Time) x 256 (frequency) and uneven number of samples for each species. Binary and Multi-class model are used for building neural networks. For binary class any two species from the dataset such as ‘amerco’ and ‘blujay’ are selected to build neural network models and perform classification. The best accuracy score for binary model is resulted as 100%. In multi-class, all the 12 bird species from the dataset are used for the classification. The best accuracy score for multi-class model is resulted as 68.39%. Three sound clips with external raw sound data is preprocessed and converted to spectrograms and with the best multi-class model is used for predicting the species of bird call from three test clips. The top 2 predictors for test 1, test 2 and test 3 are American Crow and Northern Flicker. Based on the probability values it is proved that there are multiple species in the audio clip. In conclusion, this study provides useful knowledge on the use of neural networks about environmental and species surveillance.

## BIBLIOGRAPHY

[1] Audio Classification using Deep Learning and TensorFlow https://medium.com/@oluyaled/audio-classification-using-deep-learning-and-tensorflow-a-step-by-step-guide-5327467ee9ab

[2] Xeno-Canto : sharing wildlife sounds from around the world https://xeno-canto.org/

[3] Convolutional Neural Networks https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

[4] Everything you need to know about “Activation Functions” in Deep learning models Everything you need to know about “Activation Functions” in Deep learning models | by Vandit Jain | Towards Data Science

[5] How do you select the appropriate batch size and number of epochs for neural network training? https://www.linkedin.com/advice/3/how-do-you-select-appropriate-batch-size-number

## APPENDIX

# APPENDIX

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from matplotlib import image

import os

from skimage.transform import resize

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelBinarizer, Label
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score

import tensorflow as tf
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.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, Embedding, LST
from keras.applications import imagenet_utils, ResNet50
from keras.applications.imagenet_utils import preprocess_input
from keras.datasets import mnist, cifar10
from keras.utils import to_categorical
from keras.datasets import imdb

from scipy.sparse import csr_matrix
```

In [2]:
```
import h5py
f = h5py.File('spectrograms.h5', 'r')
keys = list(f.keys())
keys
```

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

In [3]:
```
for species_key in keys:
    spectrograms = f[species_key]
    print(f'Shape of {species_key}', spectrograms.shape)
```

```
Shape of amecro (256, 343, 52)
Shape of barswa (256, 343, 55)
Shape of bkcchi (256, 343, 57)
Shape of blujay (256, 343, 50)
Shape of daejun (256, 343, 58)
Shape of houfin (256, 343, 44)
Shape of mallar3 (256, 343, 36)
Shape of norfli (256, 343, 59)
Shape of rewbla (256, 343, 41)
Shape of stejay (256, 343, 40)
Shape of wesmea (256, 343, 36)
Shape of whcspa (256, 343, 51)
```

In [ ]:

## BINARY CLASSIFICATION

In [142…

```
spectrogram_species= []
labels = []

# Loop through each bird species in the HDF5 file
for species_key in ['amecro', 'blujay']:
    spectrograms = f[species_key][:].transpose((2, 1, 0))
    for spectrogram in spectrograms:

        spectrogram_species.append(spectrogram)
        labels.append(species_key)
spectrogram_species = np.array(spectrogram_species)

encoder = LabelEncoder()
labels = encoder.fit_transform(labels)
```

In [144…

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(spectrogram_species, labels, test_
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(71, 343, 256) (31, 343, 256) (71,) (31,)
```

In [145…

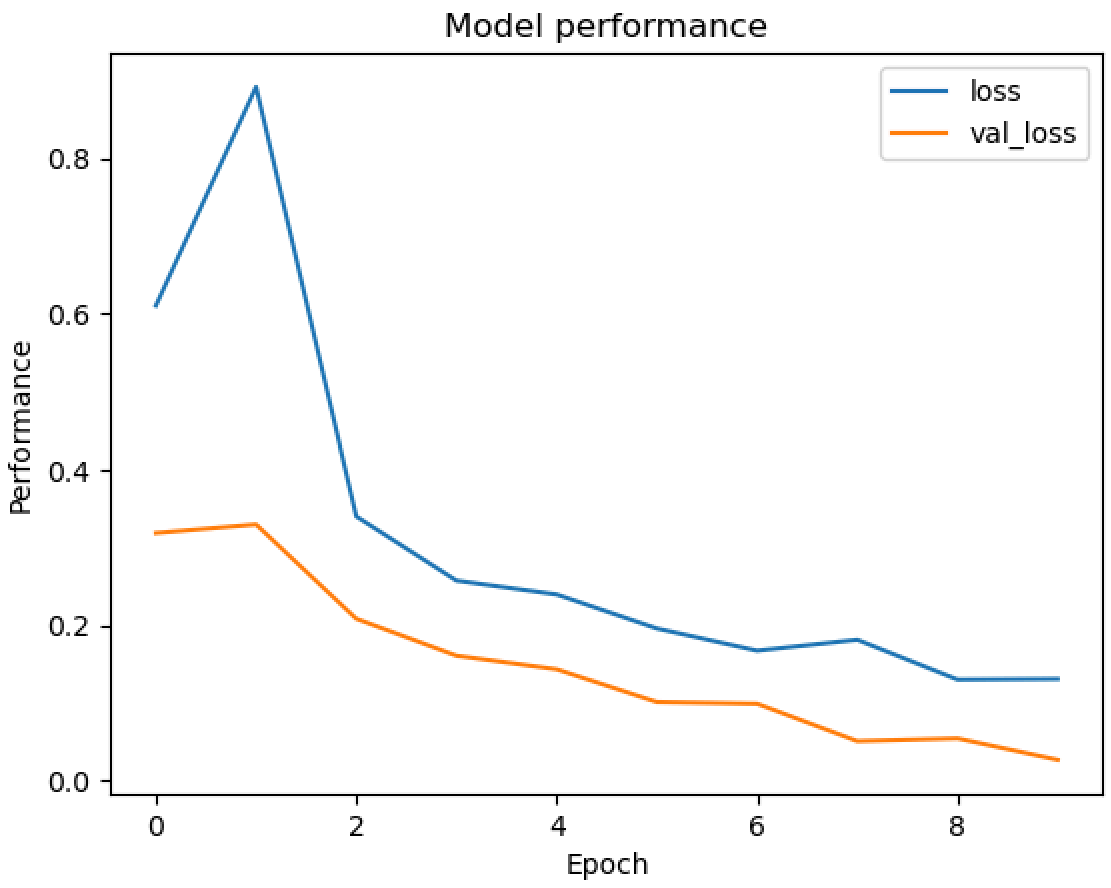
```
binary_model = Sequential([
    Input(shape=(343,256,1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

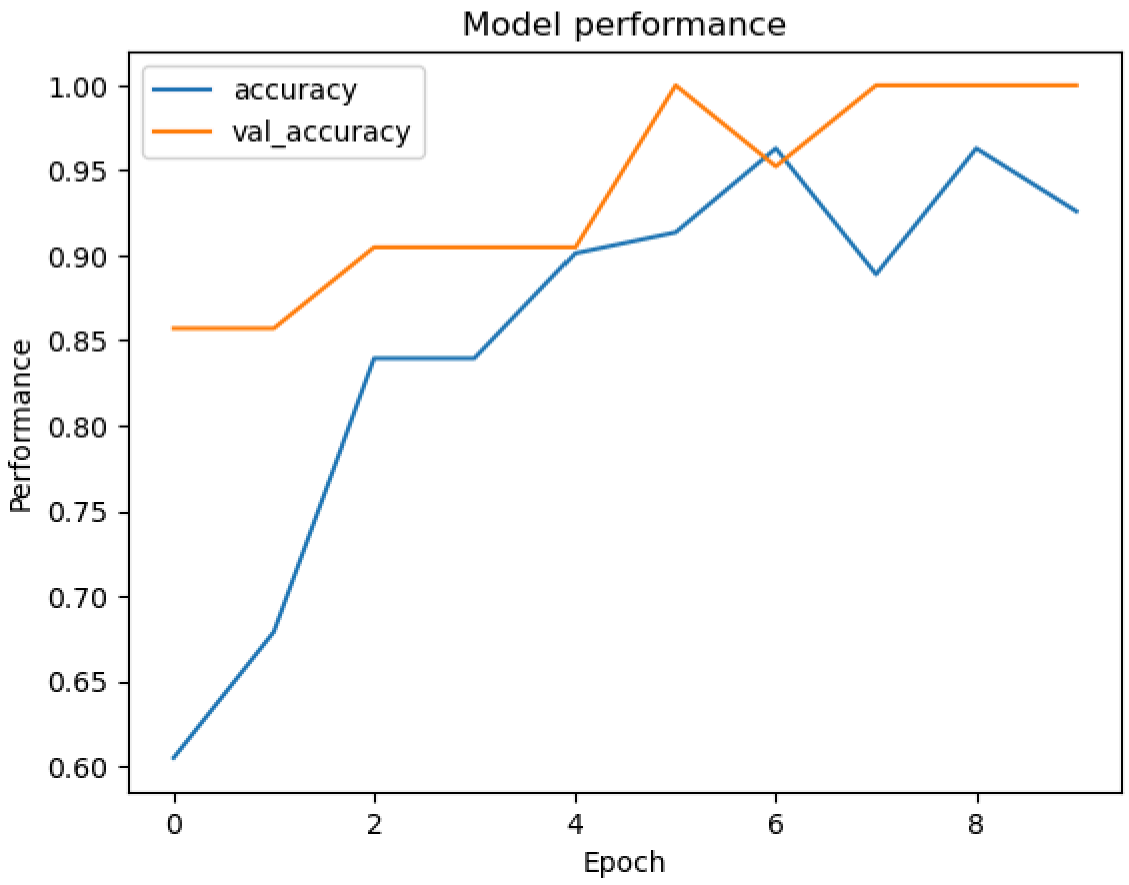
In [146…

```
# Compile the model
binary_model.compile(optimizer='rmsprop', loss='binary_crossentropy',  metrics=['accur

# Train the model
history = binary_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=
Epoch 1/10
3/3 [==============================] - 2s 632ms/step - loss: 0.6374 - accuracy: 0.662
0 - val_loss: 1.4291 - val_accuracy: 0.5806
Epoch 2/10
3/3 [==============================] - 2s 560ms/step - loss: 0.9583 - accuracy: 0.633
8 - val_loss: 0.3607 - val_accuracy: 0.8387
Epoch 3/10
3/3 [==============================] - 2s 571ms/step - loss: 0.3560 - accuracy: 0.845
1 - val_loss: 0.5323 - val_accuracy: 0.7097
Epoch 4/10
3/3 [==============================] - 2s 564ms/step - loss: 0.2738 - accuracy: 0.887
3 - val_loss: 0.2267 - val_accuracy: 0.8387
Epoch 5/10
3/3 [==============================] - 2s 555ms/step - loss: 0.2096 - accuracy: 0.901
4 - val_loss: 0.2061 - val_accuracy: 0.8387
Epoch 6/10
3/3 [==============================] - 2s 562ms/step - loss: 0.1944 - accuracy: 0.887
3 - val_loss: 0.1965 - val_accuracy: 0.8710
Epoch 7/10
3/3 [==============================] - 2s 563ms/step - loss: 0.1750 - accuracy: 0.901
4 - val_loss: 0.1842 - val_accuracy: 0.8710
Epoch 8/10
3/3 [==============================] - 2s 562ms/step - loss: 0.1567 - accuracy: 0.929
6 - val_loss: 0.1858 - val_accuracy: 0.8710
Epoch 9/10
3/3 [==============================] - 2s 541ms/step - loss: 0.1368 - accuracy: 0.915
5 - val_loss: 0.1663 - val_accuracy: 0.8710
Epoch 10/10
3/3 [==============================] - 2s 560ms/step - loss: 0.1232 - accuracy: 0.943
7 - val_loss: 0.2343 - val_accuracy: 0.8710
```

In [9]:

```python
# Plot the accuracy and loss
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()



plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()
```





In [10]:
```
train_val = binary_model.evaluate(X_train, y_train, verbose=0)
print("Train loss: {:.2f}%".format(train_val[0]*100))
```

```
test_val = binary_model.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.2f}%".format(test_val[0]*100))
```

```
Train loss: 10.55%
Test loss: 2.63%
```

In [11]:

```
train_score = binary_model.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy: {:.2f}%".format(train_score[1]*100))

test_score = binary_model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(test_score[1]*100))
```

```
Train Accuracy: 95.06%
Test Accuracy: 100.00%
```

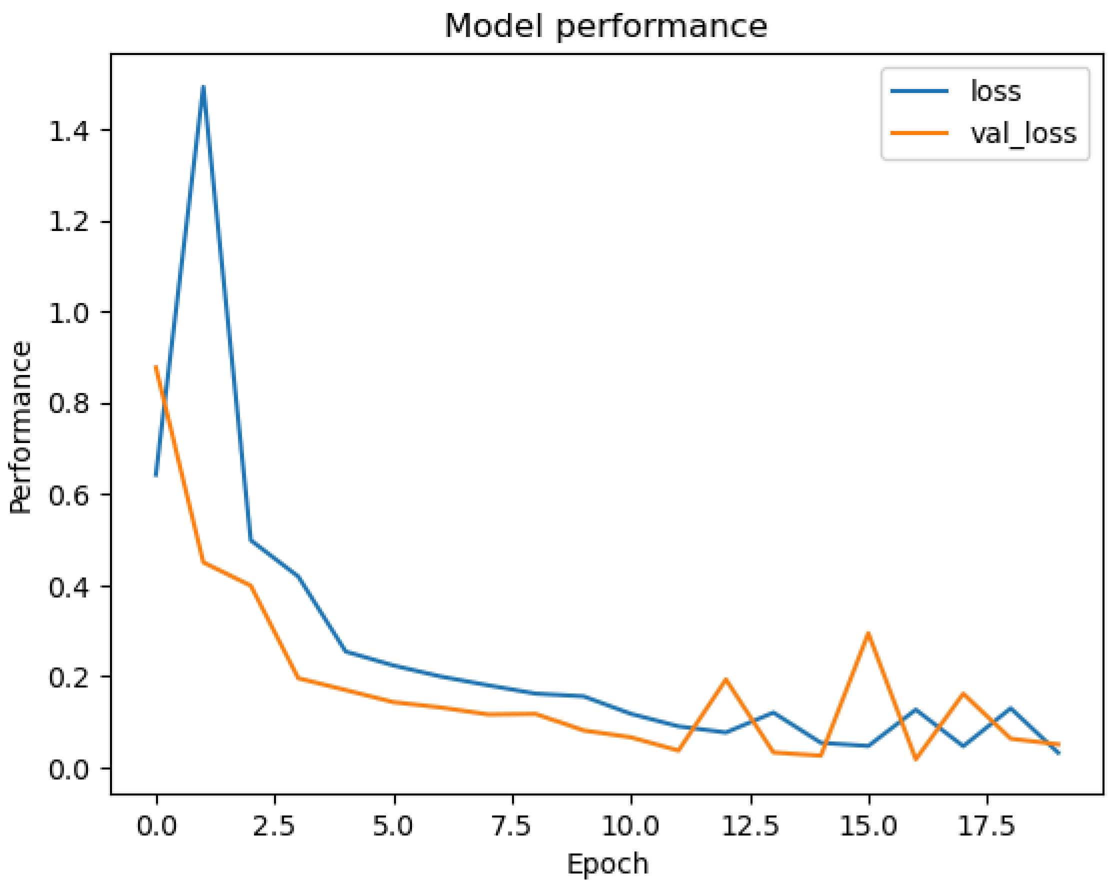
In [12]:

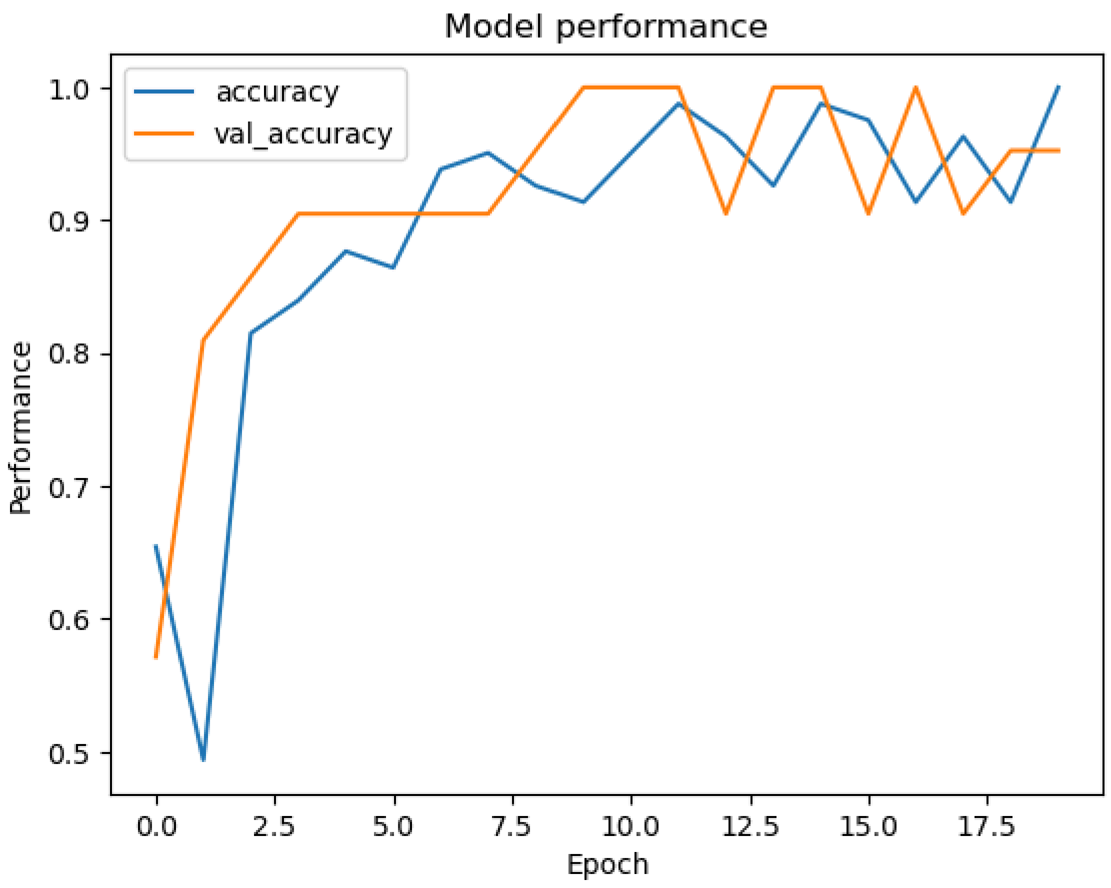
```
binary_model_2 = Sequential([
    Input(shape=(343,256,1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

In [13]:

```
# Compile the model
binary_model_2.compile(optimizer='rmsprop', loss='binary_crossentropy',  metrics=['acc

# Train the model
history = binary_model_2.fit(X_train, y_train, validation_data=(X_test, y_test), epoch
Epoch 1/20
3/3 [==============================] - 2s 626ms/step - loss: 0.6426 - accuracy: 0.654
3 - val_loss: 0.8767 - val_accuracy: 0.5714
Epoch 2/20
3/3 [==============================] - 2s 604ms/step - loss: 1.4924 - accuracy: 0.493
8 - val_loss: 0.4503 - val_accuracy: 0.8095
Epoch 3/20
3/3 [==============================] - 2s 589ms/step - loss: 0.4987 - accuracy: 0.814
8 - val_loss: 0.3992 - val_accuracy: 0.8571
Epoch 4/20
3/3 [==============================] - 2s 579ms/step - loss: 0.4193 - accuracy: 0.839
5 - val_loss: 0.1961 - val_accuracy: 0.9048
Epoch 5/20
3/3 [==============================] - 2s 577ms/step - loss: 0.2545 - accuracy: 0.876
5 - val_loss: 0.1701 - val_accuracy: 0.9048
Epoch 6/20
3/3 [==============================] - 2s 583ms/step - loss: 0.2242 - accuracy: 0.864
2 - val_loss: 0.1437 - val_accuracy: 0.9048
Epoch 7/20
3/3 [==============================] - 2s 588ms/step - loss: 0.1999 - accuracy: 0.938
3 - val_loss: 0.1319 - val_accuracy: 0.9048
Epoch 8/20
3/3 [==============================] - 2s 592ms/step - loss: 0.1806 - accuracy: 0.950
6 - val_loss: 0.1166 - val_accuracy: 0.9048
Epoch 9/20
3/3 [==============================] - 2s 583ms/step - loss: 0.1623 - accuracy: 0.925
9 - val_loss: 0.1178 - val_accuracy: 0.9524
Epoch 10/20
3/3 [==============================] - 2s 586ms/step - loss: 0.1563 - accuracy: 0.913
6 - val_loss: 0.0816 - val_accuracy: 1.0000
Epoch 11/20
3/3 [==============================] - 2s 571ms/step - loss: 0.1179 - accuracy: 0.950
6 - val_loss: 0.0664 - val_accuracy: 1.0000
Epoch 12/20
3/3 [==============================] - 2s 590ms/step - loss: 0.0905 - accuracy: 0.987
7 - val_loss: 0.0373 - val_accuracy: 1.0000
Epoch 13/20
3/3 [==============================] - 2s 607ms/step - loss: 0.0771 - accuracy: 0.963
0 - val_loss: 0.1933 - val_accuracy: 0.9048
Epoch 14/20
3/3 [==============================] - 2s 588ms/step - loss: 0.1206 - accuracy: 0.925
9 - val_loss: 0.0332 - val_accuracy: 1.0000
Epoch 15/20
3/3 [==============================] - 2s 584ms/step - loss: 0.0544 - accuracy: 0.987
7 - val_loss: 0.0262 - val_accuracy: 1.0000
Epoch 16/20
3/3 [==============================] - 2s 593ms/step - loss: 0.0474 - accuracy: 0.975
3 - val_loss: 0.2949 - val_accuracy: 0.9048
Epoch 17/20
3/3 [==============================] - 2s 589ms/step - loss: 0.1271 - accuracy: 0.913
6 - val_loss: 0.0181 - val_accuracy: 1.0000
Epoch 18/20
3/3 [==============================] - 2s 589ms/step - loss: 0.0470 - accuracy: 0.963
0 - val_loss: 0.1622 - val_accuracy: 0.9048
Epoch 19/20
3/3 [==============================] - 2s 595ms/step - loss: 0.1301 - accuracy: 0.913
6 - val_loss: 0.0634 - val_accuracy: 0.9524
Epoch 20/20
3/3 [==============================] - 2s 586ms/step - loss: 0.0326 - accuracy: 1.000
0 - val_loss: 0.0510 - val_accuracy: 0.9524
```

In [14]:
```
# Plot the accuracy and loss
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()



plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()
```





In [15]:
```
train_score = binary_model_2.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy: {:.2f}%".format(train_score[1]*100))

test_score = binary_model_2.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(test_score[1]*100))
```

```
Train Accuracy: 100.00%
Test Accuracy: 95.24%
```

In [16]:
```
train_val = binary_model_2.evaluate(X_train, y_train, verbose=0)
print("Train loss: {:.2f}%".format(train_val[0]*100))

test_val = binary_model_2.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.2f}%".format(test_val[0]*100))
```

```
Train loss: 2.30%
Test loss: 5.10%
```

## MULTI-CLASS MODEL

In [110…
```
spectrogram_species= []
labels = []

for species_key in keys:
    spectrograms = f[species_key][:].transpose((2, 1, 0))
    for spectrogram in spectrograms:
        spectrogram_species.append(spectrogram)
        labels.append(species_key)

spectrogram_species = np.array(spectrogram_species)
```

```
encoder = LabelEncoder()
labels = encoder.fit_transform(labels)
labels = to_categorical(labels, num_classes=12)
```

In [114…

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(spectrogram_species, labels, test_
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(405, 343, 256) (174, 343, 256) (405, 12) (174, 12)
```

In [115…

```
scaler = StandardScaler()
X_train_2D = X_train.reshape(-1, X_train.shape[-1])
X_train_scaled = scaler.fit_transform(X_train_2D)
X_train = X_train_scaled.reshape(X_train.shape)


X_test_2D = X_test.reshape(-1, X_test.shape[-1])
X_test_scaled = scaler.transform(X_test_2D)
X_test = X_test_scaled.reshape(X_test.shape)
```

In [116…

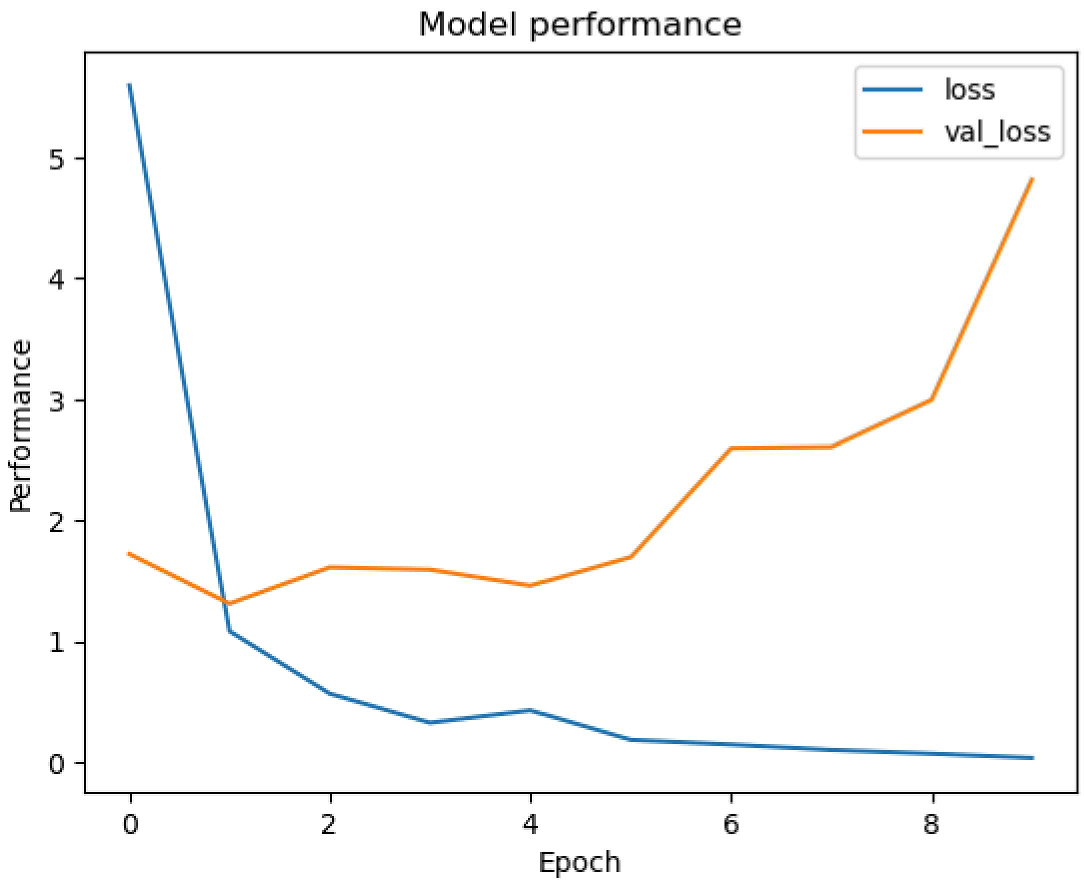
```
multi_model = Sequential([
    Input(shape=(343,256,1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(12, activation='softmax')
])
```

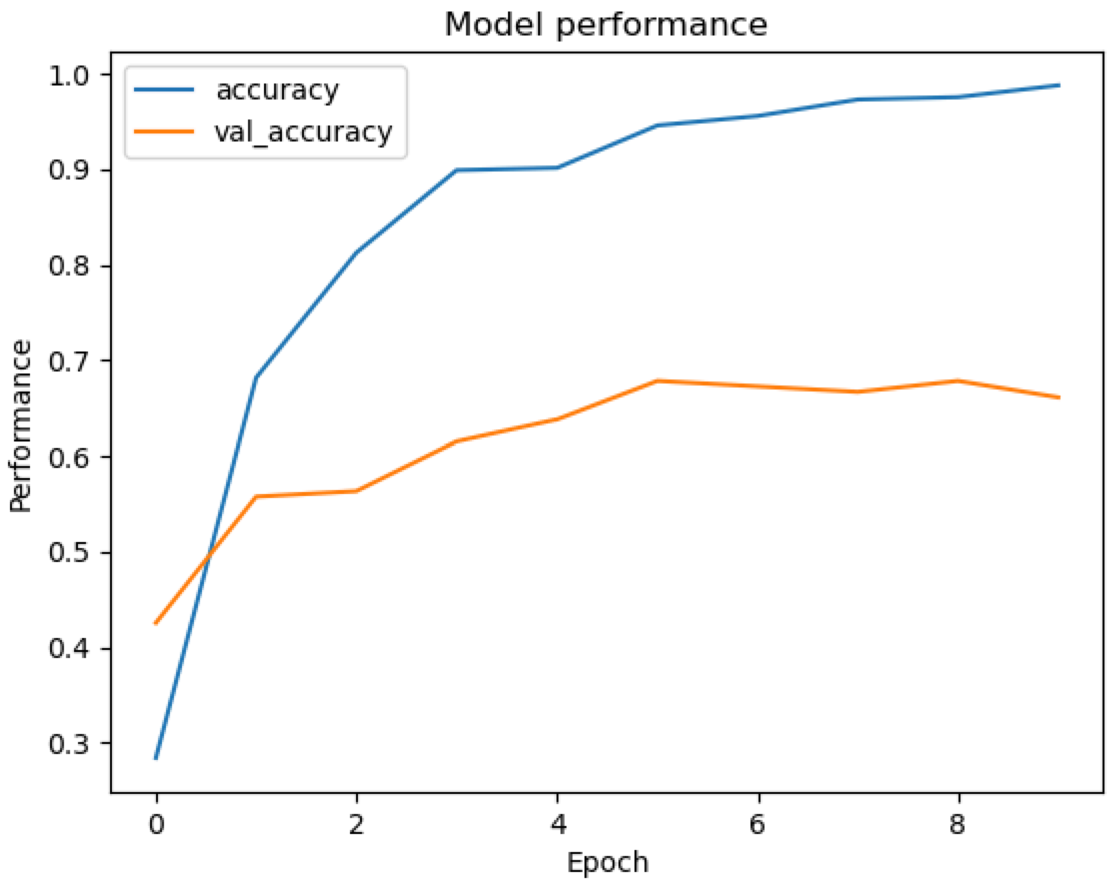
In [117…

```
# Compile the model
multi_model.compile(optimizer='rmsprop', loss='categorical_crossentropy',  metrics=['a

# Train the model
history = multi_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=1
Epoch 1/10
13/13 [==============================] - 10s 718ms/step - loss: 5.5968 - accuracy: 0.
2840 - val_loss: 1.7226 - val_accuracy: 0.4253
Epoch 2/10
13/13 [==============================] - 9s 693ms/step - loss: 1.0841 - accuracy: 0.6
815 - val_loss: 1.3121 - val_accuracy: 0.5575
Epoch 3/10
13/13 [==============================] - 9s 695ms/step - loss: 0.5672 - accuracy: 0.8
123 - val_loss: 1.6125 - val_accuracy: 0.5632
Epoch 4/10
13/13 [==============================] - 9s 700ms/step - loss: 0.3266 - accuracy: 0.8
988 - val_loss: 1.5936 - val_accuracy: 0.6149
Epoch 5/10
13/13 [==============================] - 9s 697ms/step - loss: 0.4292 - accuracy: 0.9
012 - val_loss: 1.4613 - val_accuracy: 0.6379
Epoch 6/10
13/13 [==============================] - 9s 697ms/step - loss: 0.1850 - accuracy: 0.9
457 - val_loss: 1.6976 - val_accuracy: 0.6782
Epoch 7/10
13/13 [==============================] - 9s 706ms/step - loss: 0.1463 - accuracy: 0.9
556 - val_loss: 2.5967 - val_accuracy: 0.6724
Epoch 8/10
13/13 [==============================] - 9s 699ms/step - loss: 0.1014 - accuracy: 0.9
728 - val_loss: 2.6066 - val_accuracy: 0.6667
Epoch 9/10
13/13 [==============================] - 9s 699ms/step - loss: 0.0711 - accuracy: 0.9
753 - val_loss: 2.9950 - val_accuracy: 0.6782
Epoch 10/10
13/13 [==============================] - 9s 701ms/step - loss: 0.0361 - accuracy: 0.9
877 - val_loss: 4.8174 - val_accuracy: 0.6609
```

In [118…

```python
# Plot the accuracy and loss
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()

# Plot the accuracy and loss
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()
```





In [119…

```
train_score = multi_model.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy: {:.2f}%".format(train_score[1]*100))
```

```
test_score = multi_model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(test_score[1]*100))
```

```
Train Accuracy: 99.01%
Test Accuracy: 66.09%
```

In [120…

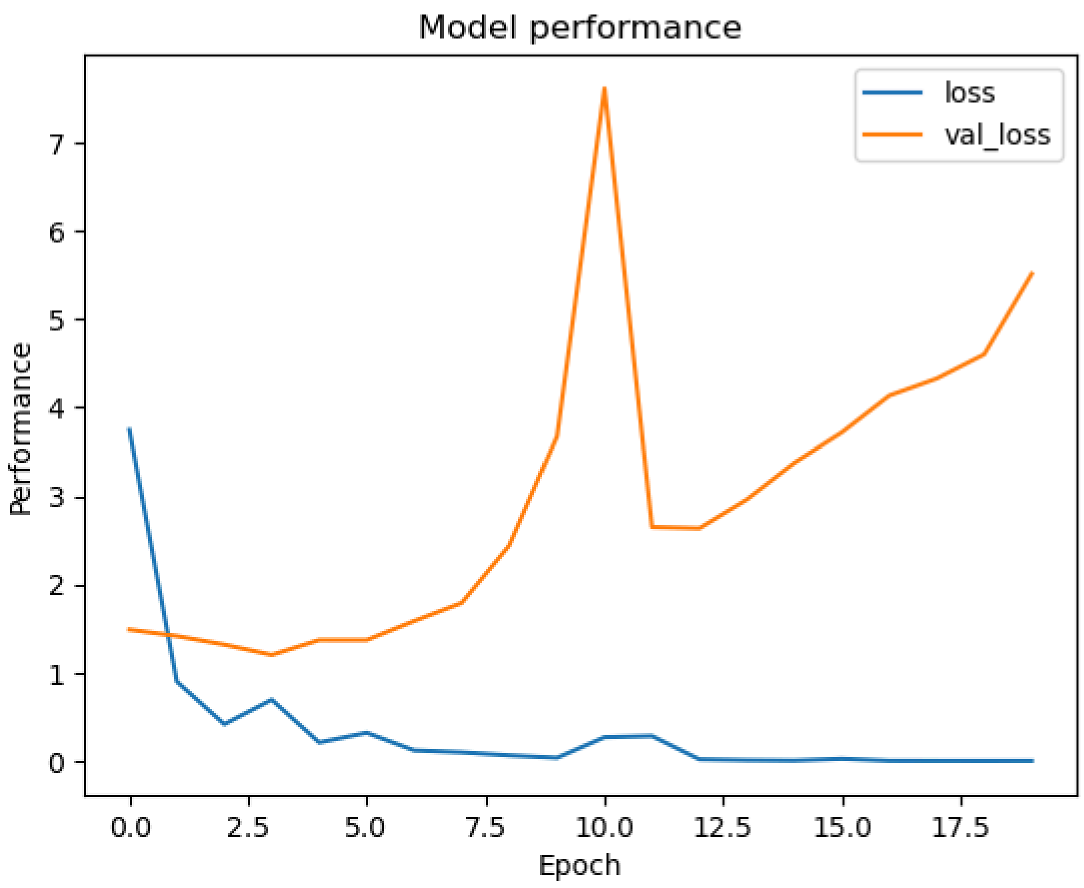
```
multi_model_2 = Sequential([
    Input(shape=(343,256,1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dense(12, activation='softmax')
])
```

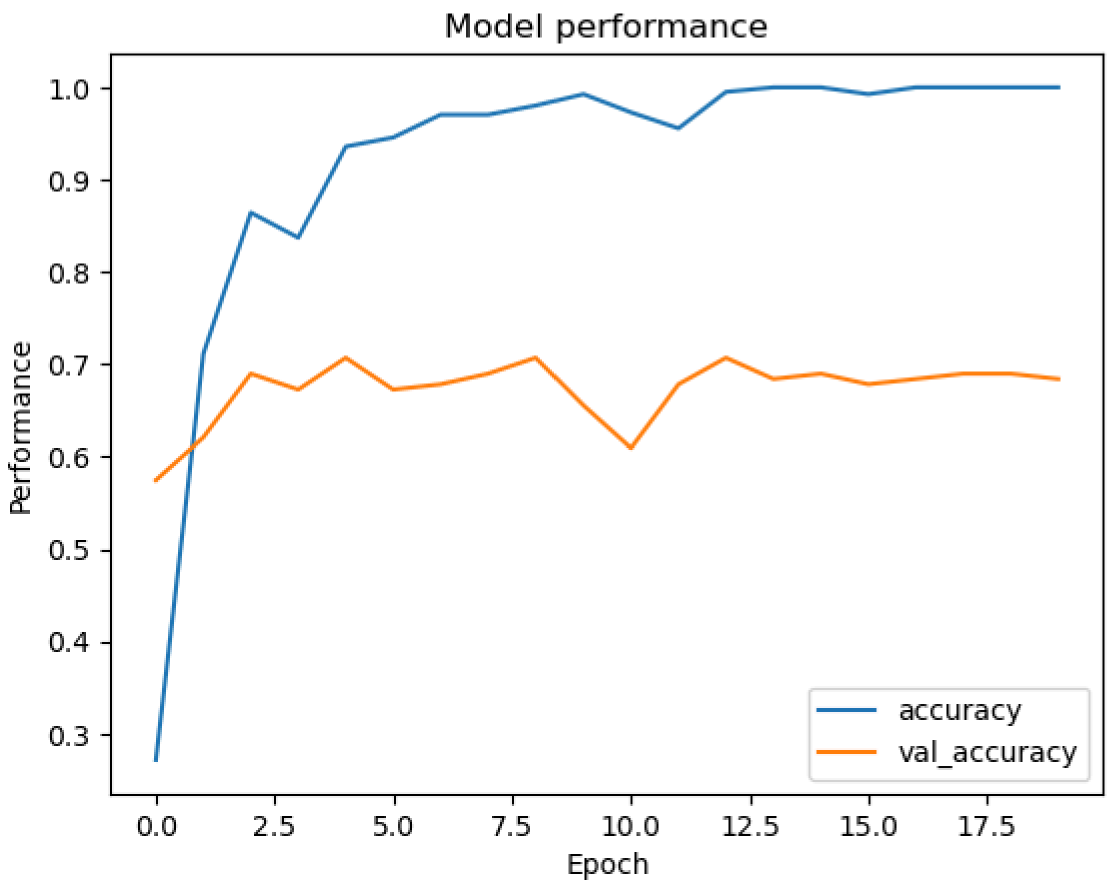
In [121…

```
# Compile the model
multi_model_2.compile(optimizer='rmsprop', loss='categorical_crossentropy',  metrics=[

# Train the model
history = multi_model_2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs
Epoch 1/20
13/13 [==============================] - 10s 743ms/step - loss: 3.7463 - accuracy: 0.
2716 - val_loss: 1.4868 - val_accuracy: 0.5747
Epoch 2/20
13/13 [==============================] - 11s 855ms/step - loss: 0.8983 - accuracy: 0.
7111 - val_loss: 1.4138 - val_accuracy: 0.6207
Epoch 3/20
13/13 [==============================] - 11s 865ms/step - loss: 0.4152 - accuracy: 0.
8642 - val_loss: 1.3169 - val_accuracy: 0.6897
Epoch 4/20
13/13 [==============================] - 11s 869ms/step - loss: 0.6925 - accuracy: 0.
8370 - val_loss: 1.2005 - val_accuracy: 0.6724
Epoch 5/20
13/13 [==============================] - 11s 862ms/step - loss: 0.2084 - accuracy: 0.
9358 - val_loss: 1.3691 - val_accuracy: 0.7069
Epoch 6/20
13/13 [==============================] - 11s 855ms/step - loss: 0.3184 - accuracy: 0.
9457 - val_loss: 1.3687 - val_accuracy: 0.6724
Epoch 7/20
13/13 [==============================] - 11s 863ms/step - loss: 0.1188 - accuracy: 0.
9704 - val_loss: 1.5842 - val_accuracy: 0.6782
Epoch 8/20
13/13 [==============================] - 11s 861ms/step - loss: 0.0970 - accuracy: 0.
9704 - val_loss: 1.7888 - val_accuracy: 0.6897
Epoch 9/20
13/13 [==============================] - 11s 862ms/step - loss: 0.0621 - accuracy: 0.
9802 - val_loss: 2.4376 - val_accuracy: 0.7069
Epoch 10/20
13/13 [==============================] - 11s 863ms/step - loss: 0.0346 - accuracy: 0.
9926 - val_loss: 3.6728 - val_accuracy: 0.6552
Epoch 11/20
13/13 [==============================] - 11s 870ms/step - loss: 0.2682 - accuracy: 0.
9728 - val_loss: 7.6124 - val_accuracy: 0.6092
Epoch 12/20
13/13 [==============================] - 11s 868ms/step - loss: 0.2818 - accuracy: 0.
9556 - val_loss: 2.6475 - val_accuracy: 0.6782
Epoch 13/20
13/13 [==============================] - 11s 880ms/step - loss: 0.0171 - accuracy: 0.
9951 - val_loss: 2.6339 - val_accuracy: 0.7069
Epoch 14/20
13/13 [==============================] - 11s 876ms/step - loss: 0.0079 - accuracy: 1.
0000 - val_loss: 2.9603 - val_accuracy: 0.6839
Epoch 15/20
13/13 [==============================] - 11s 880ms/step - loss: 0.0041 - accuracy: 1.
0000 - val_loss: 3.3738 - val_accuracy: 0.6897
Epoch 16/20
13/13 [==============================] - 11s 872ms/step - loss: 0.0237 - accuracy: 0.
9926 - val_loss: 3.7169 - val_accuracy: 0.6782
Epoch 17/20
13/13 [==============================] - 11s 868ms/step - loss: 0.0013 - accuracy: 1.
0000 - val_loss: 4.1329 - val_accuracy: 0.6839
Epoch 18/20
13/13 [==============================] - 11s 876ms/step - loss: 5.6255e-04 - accurac
y: 1.0000 - val_loss: 4.3258 - val_accuracy: 0.6897
Epoch 19/20
13/13 [==============================] - 11s 871ms/step - loss: 2.4125e-04 - accurac
y: 1.0000 - val_loss: 4.6002 - val_accuracy: 0.6897
Epoch 20/20
13/13 [==============================] - 11s 870ms/step - loss: 5.4370e-04 - accurac
y: 1.0000 - val_loss: 5.5117 - val_accuracy: 0.6839
```

In [122…

```
# Plot the accuracy and loss
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()

# Plot the accuracy and loss
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()
```





In [123…

```
train_score = multi_model_2.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy: {:.2f}%".format(train_score[1]*100))

test_score = multi_model_2.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(test_score[1]*100))
```

```
Train Accuracy: 100.00%
Test Accuracy: 68.39%
```

## EXTERNAL TEST DATA

In [124…

```
import librosa
import librosa.display
from IPython.display import Audio
```

In [125…

```
audio_recording="test_birds/test_birds/test1.mp3"
data,rate=librosa.load(audio_recording)
print(type(data),type(rate))
```

```
<class 'numpy.ndarray'> <class 'int'>
```

In [126…

```
print(f"Sample Rate: {rate} Hz ")
print(f"Number of Samples : {data.shape[0]}")
length = data.shape[0] / rate
print("Length of the audio file is {:.2f} seconds".format(length))
```

```
Sample Rate: 22050 Hz
Number of Samples : 513216
Length of the audio file is 23.28 seconds
```

In [127…
```
audio_recording="test_birds/test_birds/test2.mp3"
data,rate=librosa.load(audio_recording)
print(type(data),type(rate))


print(f"Sample Rate: {rate} Hz ")
print(f"Number of Samples : {data.shape[0]}")
length = data.shape[0] / rate
print("Length of the audio file is {:.2f} seconds".format(length))
```

```
<class 'numpy.ndarray'> <class 'int'>
Sample Rate: 22050 Hz
Number of Samples : 115776
Length of the audio file is 5.25 seconds
```

In [128…
```
audio_recording="test_birds/test_birds/test3.mp3"
data,rate=librosa.load(audio_recording)
print(type(data),type(rate))


print(f"Sample Rate: {rate} Hz ")
print(f"Number of Samples : {data.shape[0]}")
length = data.shape[0] / rate
print("Length of the audio file is {:.2f} seconds".format(length))
```

```
<class 'numpy.ndarray'> <class 'int'>
Sample Rate: 22050 Hz
Number of Samples : 350208
Length of the audio file is 15.88 seconds
```

In [129…
```
#PREPROCESSING EXTERNAL TEST CLIPS

# Path to the folder containing test clips
test_clips_folder = 'test_birds/test_birds'

# Get paths to all files in the folder
sound_clip_paths = [os.path.join(test_clips_folder, file) for file in os.listdir(test_

# Initialize a list to store spectrograms
spectrograms = []

for path in sound_clip_paths:
    # Load the audio file
    audio, sr = librosa.load(path, sr=22050)

    # Identify "loud" parts greater than 0.5 seconds
    loud_parts = librosa.effects.split(audio, top_db=20)

    # Extract 2-second windows containing bird calls
    for interval in loud_parts:
        start, end = interval
        duration = end - start
        if duration >= 0.5:
            for i in range(start, end - 2 * sr + 1, sr):
                window = audio[i:i + 2 * sr]

                # Compute spectrogram
                spectrogram = librosa.feature.melspectrogram(y=window, sr=sr, n_fft=20
                spectrogram = librosa.power_to_db(spectrogram, ref=np.max)
                spectrogram = resize(spectrogram, (343,256))
                spectrograms.append(spectrogram)

# Convert spectrograms list to numpy array
spectrograms = np.array(spectrograms)
```

In [141…

```
predictions = multi_model.predict(spectrograms)

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

for i, (pred, path) in enumerate(zip(predictions, sound_clip_paths)):
    print(f"Predictions for {path}:")

    predicted_species = np.argmax(pred)
    predicted_species_name = species_mapping[predicted_species]
    predicted_probability = pred[predicted_species] * 100

    for species_idx, probability in enumerate(pred):
        species_name = species_mapping[species_idx]
        if species_idx == predicted_species:
            print(f"Species: {species_name}, Probability: {probability * 100:.2f}%")
        else:
            print(f"Species: {species_name}, Probability: {probability * 100:.2f}%")
```

```
2/2 [==============================] - 0s 40ms/step
Predictions for test_birds/test_birds\test1.mp3:
Species: amecro, Probability: 52.60%
Species: barswa, Probability: 0.00%
Species: bkcchi, Probability: 0.00%
Species: blujay, Probability: 0.00%
Species: daejun, Probability: 0.00%
Species: houfin, Probability: 0.00%
Species: mallar3, Probability: 0.00%
Species: norfli, Probability: 45.98%
Species: rewbla, Probability: 0.00%
Species: stejay, Probability: 0.00%
Species: wesmea, Probability: 1.42%
Species: whcspa, Probability: 0.00%

Predictions for test_birds/test_birds\test2.mp3:
Species: amecro, Probability: 51.60%
Species: barswa, Probability: 0.00%
Species: bkcchi, Probability: 0.00%
Species: blujay, Probability: 0.00%
Species: daejun, Probability: 0.00%
Species: houfin, Probability: 0.00%
Species: mallar3, Probability: 0.00%
Species: norfli, Probability: 47.85%
Species: rewbla, Probability: 0.00%
Species: stejay, Probability: 0.00%
Species: wesmea, Probability: 0.55%
Species: whcspa, Probability: 0.00%

Predictions for test_birds/test_birds\test3.mp3:
Species: amecro, Probability: 42.28%
Species: barswa, Probability: 0.00%
Species: bkcchi, Probability: 0.00%
Species: blujay, Probability: 0.00%
Species: daejun, Probability: 0.00%
Species: houfin, Probability: 0.00%
Species: mallar3, Probability: 0.00%
Species: norfli, Probability: 57.29%
Species: rewbla, Probability: 0.00%
Species: stejay, Probability: 0.00%
Species: wesmea, Probability: 0.42%
Species: whcspa, Probability: 0.00%
```

```
In [ ]:
```