Bird Species Identification with Deep Learning

Tejaswi Neelapu

Abstract:

This report takes us through the application of deep learning techniques for identifying the bird species based on their calls, using data from Xeno-Canto [1], a crowd-sourced bird sounds archive. The data was further pre-processed to spectrograms of the original data. The dataset consists of spectrograms of 12 bird species calls which are found in the Seattle area. This approach involves building a neural network architecture for both binary and multi-class classification tasks and then using the trained model to predict bird species based on their calls which are unknown. This study aims to find the most accurate neural network structure while experimenting with different hyperparameters for identifying the bird species. Data preprocessing, model training, parameter tuning and plots are used to build a predictive model for identifying the bird species based on their calls. The models achieved maximum accuracies of 95.45% (binary), and 67.24% (multi-class). The multi-class model that had the highest accuracy was further taken to predict the 3 test clips of bird calls.

Introduction and Overview:

Identification of Bird species based on their calls is important and a great way to be aware of our surroundings and biodiversity. Usually, this job was done by experts in a different field, but with the help of deep learning, automated techniques are widely used for predicting the bird species. [2] The models that are being trained in this study are to find the application of neural networks in the classification of bird species using their calls, which are converted to the spectrogram data.

This study focuses on training neural network architectures for two basic tasks: binary classification where two species (Black-capped chickadee and Blue Jay) are selected and classified, multi-class classification where all the 12 species (American Crow, Barn Swallow, Black-capped chickadee, Blue Jay, Dark-eyed Junco, House Finch, Mallard, Northern Flicker, Red-winged Blackbird, Steller's Jay, Western Meadowlark and White-crowned Sparrow) are classified. The dataset is made of the spectrograms that have been pre-processed, which were produced by the sample of the audio clips and where the bird calls were detected. The preprocessing steps include data subsampling and spectrogram generation, that provide a working model for the data that can be read and viewed using the h5py package.

Hyperparameters like epochs and batch size are changed and the accuracies are compared to identify the best way for bird species classification. To sum up, the external test data is studied, which has the original sound clips that include 12 bird calls and that will be the basis to understand the difficulties of the bird species identification in different audio environments. With this study, one can develop the model for the bird species identification based on their calls using deep learning techniques.

Theoretical Background:

A neural network is an approach in artificial intelligence that helps the computer acquire capability to analyse data in the same way as the human brain does. It is a specific form of Machine Learning technique called Deep Learning, which is the network of interconnected nodes or neurons like neurons in the human brain, arranged in layers. [3]

Neurons and Layers: In a neural network, the graph has nodes and edges, commonly known as neurons, which take in the input layer, transform it into another form in the processing layer, and then forward the result to the other layer typically the output layer. Layers include:

> **Input Layer:** the access part when data is introduced into the network.
> **Hidden Layers:** the part where in data is transformed as per the learned weightage values that are present in between the layers.
> **Output Layer:** This part involves the last judgment or classification of an item.

Activation Functions: Functions through which non-linearity occurs in the network so that the network can learn from data and make complex decisions or computations. Key activation functions include:

> **ReLU (Rectified Linear Unit):** This activates nonlinear features through operation of negation of negative values.
> **Sigmoid:** input values are usually between 0 and 1, the sigmoid function is commonly used in the output layer for binary classification of neural networks.
> **Softmax:** A sigmoid function always produces an output value that lies between 0 and 1 where one of the classes is output and all other classes are put into competition to try to produce the output class. It transforms the logits or predicted which in fact, are raw scores into probabilities whereas the probability scores add up to 1.

Training: This involves differing the weights of connections and this done through the back propagation with the assistance of other algorithms such as stochastic gradient decent (SGD).

Loss Functions: Measures the variation of the output when the program or a model performs the calculation against the input, to facilitate the optimization efforts. Some of these are Mean Squared Error (MSE) which is used for regression problems and Cross-Entropy loss, which is commonly used in classification problems.

Regularization: This is achieved through methods like dropout that helps to reduce overfitting and batch normalization that helps the model learn features that could easily generalize to other data sets.

Network Architecture: The design of neural networks plays a significant role in defining a model's capabilities in learning and refining itself based on the data fed into it. Key architectural decisions include:

> **Depth:** This depends on the number of layers in the network, a deeper neural network is likely to involve more computations than a shallow one. But at the same time, more complex functions can be modelled with deeper networks, and training is much more difficult.

Width: The number of neurons in each layer. Wider networks can fit the data more comprehensively but might fit noise as well.

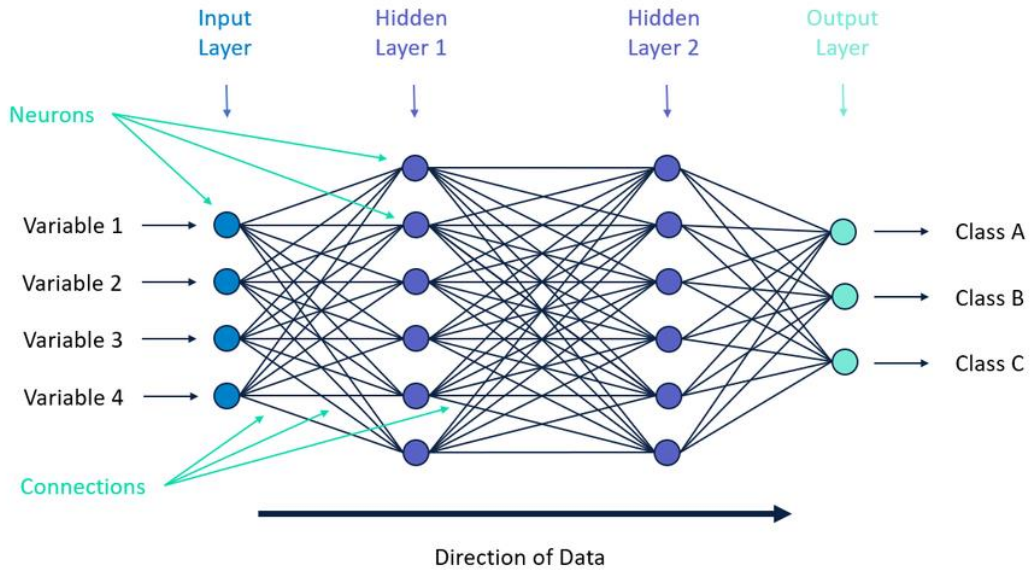
Hyperparameters:

Batch size: Specifies the number of training examples which is used in one iteration. A large batch size results in faster convergence but it also has issues with consuming more memory.

Epochs: Describes the full cycle through the given training data. There are several iterations of training a neural network where the model updates its parameters in order to minimize the loss function.

Convolutional Neural Networks (CNNs) are a type of deep learning models which are designed to analyse images or other types of visual data. They are composed of layers that enable the network to identify such features as edges, texture or shapes by applying or filtering across the image input. These layers help to gradually analyse higher levels of features within an image and is one of the major reasons why CNNs really excel in jobs such as image recognition and image classification. With such techniques like pooling to minimize the size of the data and fully connect layers to take final decision on the location of the object, CNNs are able to identify and classify objects in images.

Recurrent Neural Networks (RNNs) are other type of neural networks in the deep learning architecture, which are particularly designed to work on a sequence data. Contrary to other types of neural networks which do not possess connections that provide feed-backs, RNNs possess feed forward connections that facilitate the storing of past inputs so that contexts may be applied to the instances that are fed into the RNN. This means that RNNs are ideal for contexts where order and recurrence matter such as in language generation and recognition, speech and pattern prediction among others. RNN, in its working mechanism, allows processing of data step wise and contains a state that is a result of information from the previous step and that helps the RNN in learning the different patterns and dependency of the sequence data. [4]



[5]

Methodology:

The pre-processed data “spectrograms.h5” file is read with the help of h5py package. The original data can also be used, where this has the audio files and shall be further converted to spectrograms.

For Binary classification, 2 bird species (Black-capped chickadee and Blue Jay) data is considered. The loaded data is rearranged by using transpose to make the shape (samples, time, frequency) from (time, frequency, samples). Then labels are assigned to each bird species 0 and 1. Which are further concatenated along the first axis which combines them into single labels array. The data and labels are joined together in the sample axis to make the training dataset. The data was split into training (80%) and testing (20%), random state is taken as 42 which makes sure that the split is reproducible. The model is built with Sequential neural network model from the keras package with different layers. A Flatten layer is used to convert the input data which has the shape (256, 343) into a 1D array. Two Dense (fully connected) layers with ReLU activation, followed by dropout layers are used to prevent overfitting. One more Dense layer with sigmoid activation function is used. The model is compiled using optimizer RMSprop and cross-entropy loss function binary cross-entropy because it is a binary model. Accuracy metric is used. The model is trained on the training data with 60 epochs and a batch size of 128. The epochs and batch size were differed to improve the accuracy. 20% of the training data is used as a validation set to control the model’s performance. Finally, the model is tested on the test data to find the accuracy.

For multi-class classification, all the 12 bird species (American Crow, Barn Swallow, Black-capped chickadee, Blue Jay, Dark-eyed Junco, House Finch, Mallard, Northern Flicker, Red-winged Blackbird, Steller’s Jay, Western Meadowlark and White-crowned Sparrow) data is considered. The loaded data is rearranged by using transpose to make the shape (samples, time, frequency) from (time, frequency, samples). Then labels are created for all the bird species. The label is an array filled with the index of the species. Which are further concatenated along the sample axis to create feature matrix and label vector. The integer labels are converted to one-hot encoded format. The data was split into training (80%) and testing (20%), random state is taken as 42 to make sure that the split is reproducible. The model is built with Sequential neural network model from the keras package with different layers. A Flatten layer is used to convert the 2D spectrogram input into a 1D array. Two Dense (fully connected) layers with ReLU activation, followed by dropout layers are used to prevent overfitting. One more Dense layer with softmax activation function is used. The model is compiled using optimizer RMSprop and cross-entropy loss function categorical cross-entropy because it is a multi-class model. Accuracy metric is used. The model is trained on the training data with 60 epochs and a batch size of 256. The epochs and batch size were differed to improve the accuracy. 20% of the training data is used as a validation set to control the model’s performance. Finally, the model is tested on the test data to find the accuracy.

For the external test data, the bird species shall be predicted based on the three external audio files. The multi-class classification model and the respective test data were loaded. The model architecture and weights, were loaded to another h5 file which will be used here. The external audio files which are the testing data here, are defined in a list. All the 12 bird

species (American Crow, Barn Swallow, Black-capped chickadee, Blue Jay, Dark-eyed Junco, House Finch, Mallard, Northern Flicker, Red-winged Blackbird, Steller's Jay, Western Meadowlark and White-crowned Sparrow) names are defined to match the corresponding class labels used for model training.
Each audio file was converted to a spectrogram first, then the spectrogram was pre-processed to make it suitable as an input to the model. The pre-processed spectrogram is further passed to the model for prediction. The model predicts the probabilities of each bird species present in all three test clips. [6]
For this prediction process, the trained multi-class classification model was used to identify the bird species from their calls that were present in the external test clips. The probabilities gave the likeliness of different bird calls being present in the test clips.

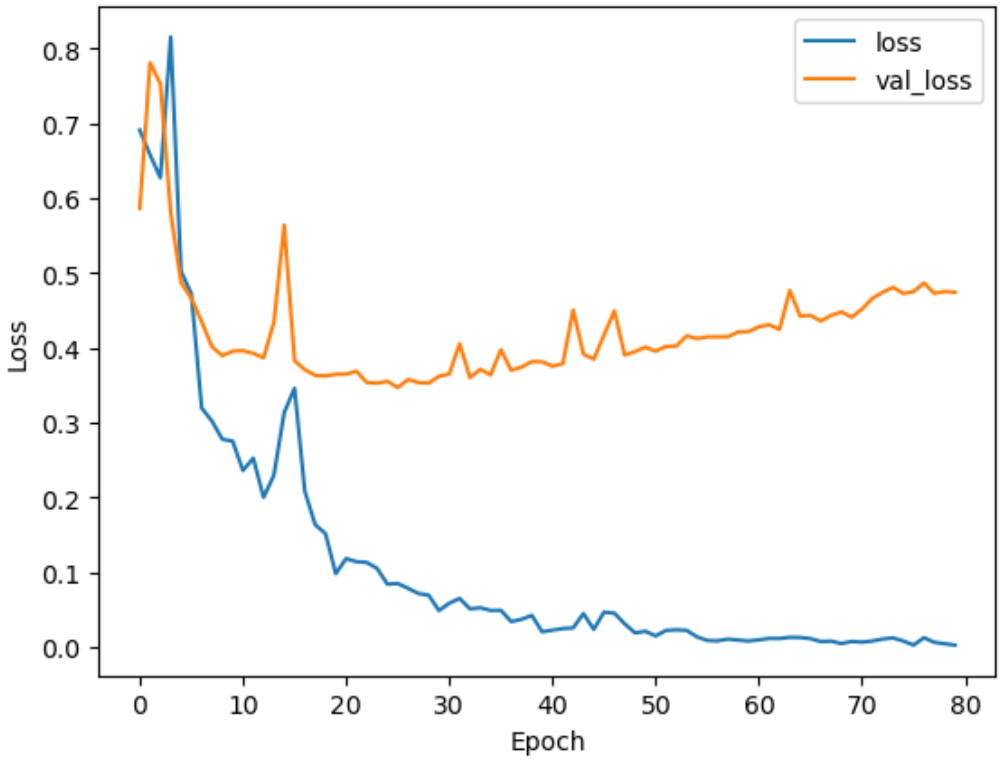
Computational Results:

Binary Model:

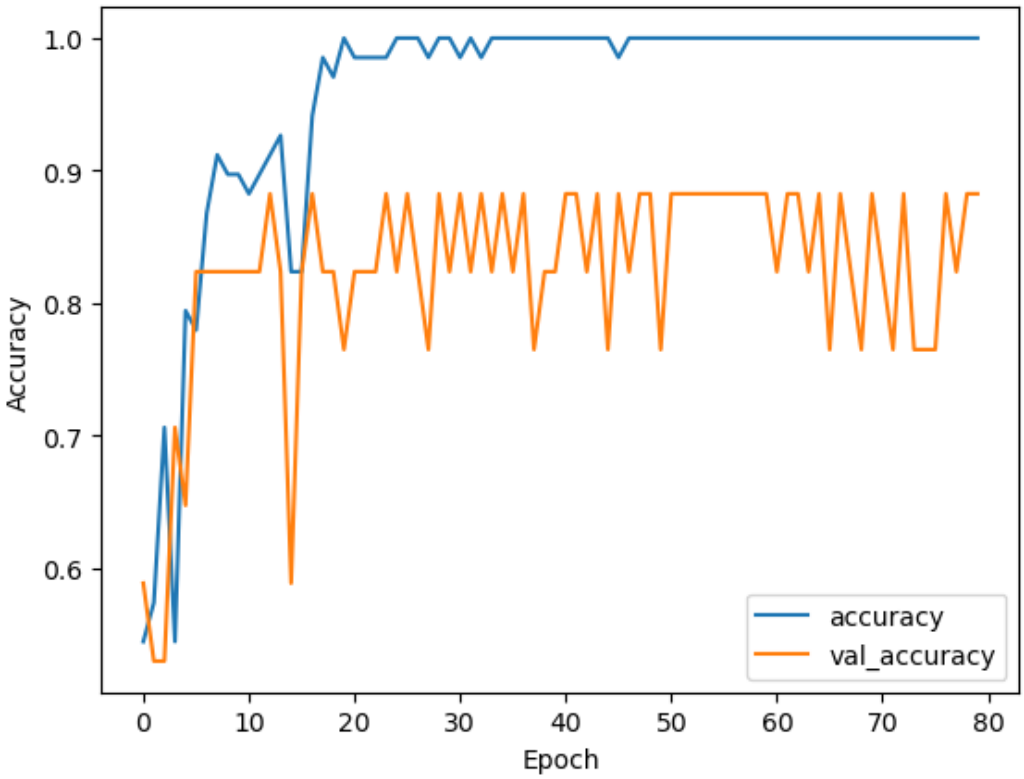
Out of 12 bird species, 2 were considered to build this binary model. Tried with a couple of bird pairs, but the accuracy was not improving even when the hyperparameters were changed. While choosing 'bkcchi' (Black-capped chickadee) and 'blujay' (Blue Jay), the accuracy improved while differing the hyperparameters. Hence, considered these two bird species.

Batch Size	Epochs	Test Accuracy
64	20	95.45%
128	20	95.45%
256	20	77.27%
128	10	90.91%
128	50	95.45%
128	60	95.45%

While having epochs (20, 50, 60) and batch size 128 the model gave the maximum accuracy which is 95.45%. This might be overfitting. In that case, other epochs with batch size 128 can be considered.



When epochs are increasing the loss is decreasing. The highest loss is seen when epochs are 5.



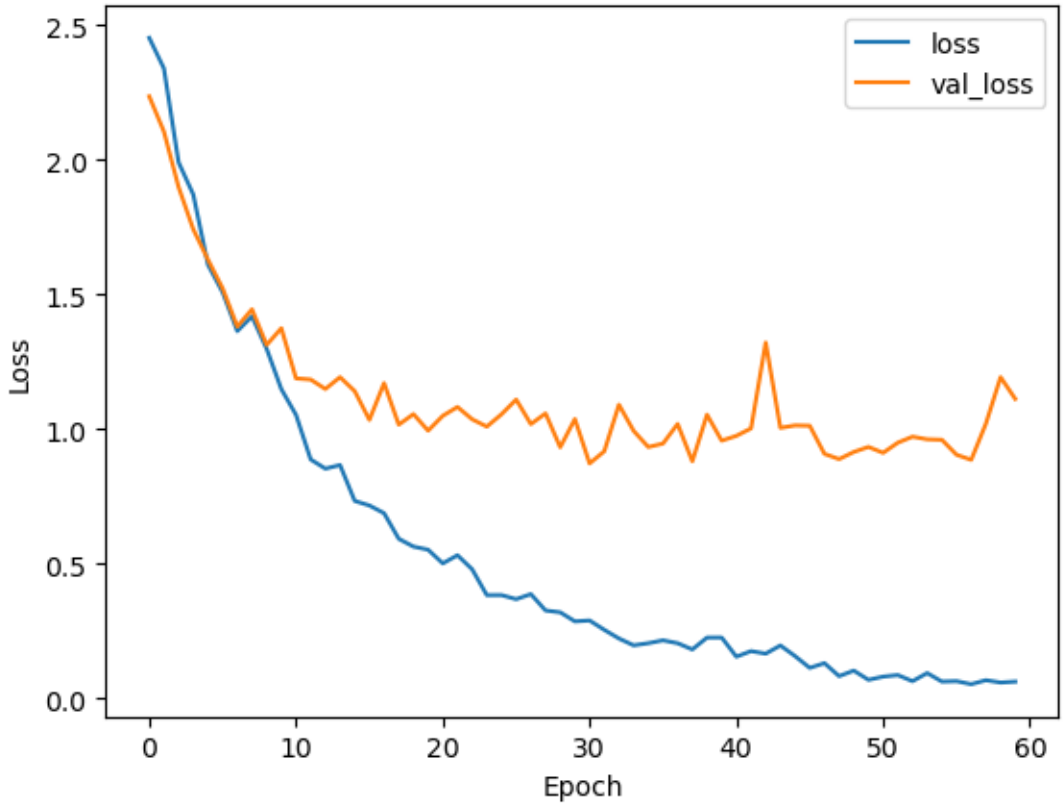
The accuracy is fluctuating a lot. When epochs are greater than 30, the accuracy is very high almost near to 100%.

Multi-class Model:

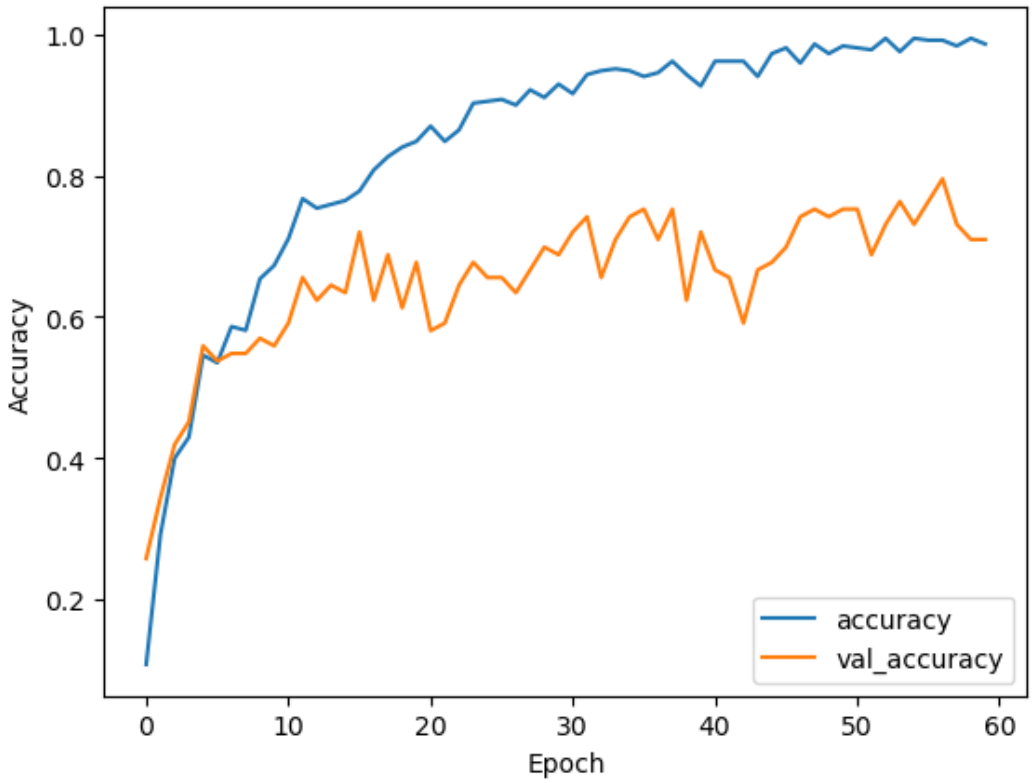
All 12 bird species (American Crow, Barn Swallow, Black-capped chickadee, Blue Jay, Dark-eyed Junco, House Finch, Mallard, Northern Flicker, Red-winged Blackbird, Steller's Jay, Western Meadowlark and White-crowned Sparrow) were considered to build this multi-class model. The accuracy was improving when the hyperparameters were changed.

Batch Size	Epochs	Test Accuracy
128	50	63.79%
256	50	65.52%
256	20	56.90%
256	40	64.66%
256	60	67.24%
256	80	64.66%

While having epochs 60 and batch size 256 the model is giving 67.24% accuracy. This was the highest accuracy achieved in this model.



When epochs are increasing the loss is decreasing. The highest loss is seen when epochs are very less that is below 10.



The accuracy is fluctuating. When epochs are greater than 10, the accuracy is increasing and when epochs are greater than 50 the accuracy is almost 100%.

External Test Data:
Used the multi-class model to train the data. This was further saved as a spectrogram which were reshaped and used to predict the birds based on the audio files.

```
| Audio File     | Predicted Bird Probabilities |
|----------------|------------------------------|
1/1 [==============================] - 0s 57ms/step
| test_birds/test1.mp3 |
|                | amecro: 0.0048
|                | barswa: 0.0000
|                | bkcchi: 0.0000
|                | blujay: 0.0000
|                | daejun: 0.0000
|                | houfin: 0.0000
|                | mallar3: 0.0000
|                | norfli: 0.0000
|                | rewbla: 0.0000
|                | stejay: 0.0000
|                | wesmea: 0.0000
|                | whcspa: 0.9952
```

```
1/1 [==============================] - 0s 24ms/step
| test_birds/test2.mp3 |
|                | amecro: 0.0000
|                | barswa: 0.0000
|                | bkcchi: 0.0000
|                | blujay: 0.9567
|                | daejun: 0.0000
|                | houfin: 0.0000
|                | mallar3: 0.0433
|                | norfli: 0.0000
|                | rewbla: 0.0000
|                | stejay: 0.0000
|                | wesmea: 0.0000
|                | whcspa: 0.0000
1/1 [==============================] - 0s 25ms/step
| test_birds/test3.mp3 |
|                | amecro: 0.0000
|                | barswa: 0.0000
|                | bkcchi: 0.0000
|                | blujay: 0.0000
|                | daejun: 0.0000
|                | houfin: 0.0000
|                | mallar3: 0.0000
|                | norfli: 0.0000
|                | rewbla: 0.0000
|                | stejay: 0.0000
|                | wesmea: 0.0000
|                | whcspa: 1.0000
```

Bird species predicted with high probability:

Audio File	Bird Species with high probability
test1	whcspa (White-crowned Sparrow)
test2	blujay (Blue Jay)
test3	whcspa (White-crowned Sparrow)

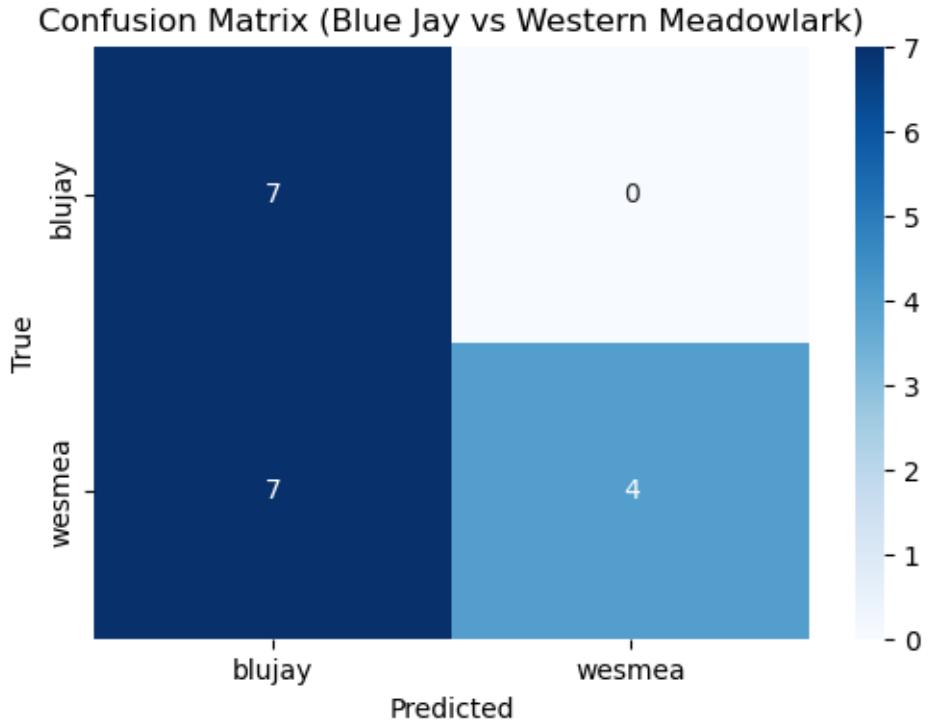
The call identification of each of the three test clips is raw sound data in mp3 format, and some test clips had multiple bird calls. The audio files were transformed into spectrograms, and then pre-processed and used to feed the 12-species network in order to determine which birds were singing. As for the prediction of each of the three clips in the experiment, results are shown in the above table.

Discussion:

There were several limitations while working on this homework. One difficulty was in preparing the audio files, converting them into a format that would match the requirements of the algorithm. Changes in the time, frequency and the transformation of features also had a few intermediary steps like resizing and normalization. Any of these steps could potentially differ the representation and further affect the final output prediction offered by the model. [7]

Training the models did not take much time. The training time was also affected by choice of the number of epochs used as well as the batch size. Hyperparameters were further adjusted to bring additional improvements to model performance which increased the training duration. Binary model took around 6 seconds to get trained. Multi-class model took around 22 seconds to train. To predict the external test clip, each step took 54 milli seconds.

Blue Jay and Western Meadowlark are being confused to one another, that is because both sound the same the only difference is the pitch. Blue Jay is louder but they sound the same. The background noise is also contributing to misclassification up to an extent.



Here from the confusion matrix, it can be observed that the model identified all the Blue Jays (7) correctly. But struggled with Western Meadowlarks, misclassifying 7 of them as Blue Jays. This proves that the selected model has a high accuracy of identifying Blue Jays, but is less accurate when differentiating between Western Meadowlarks and Blue Jays, based on the fact that both the birds are almost similar in sound, and the background noise as put it.

SVMs can be used which are usually used for classification. But using Neural Network would be a good fit here because they can pick up from spectrogram images and doesn't require feature extraction to be done manually. And they can adapt to varying complexity of bird calls and background noise. Studies show that techniques like CNNs and RNNs have achieved great performance in audio classification tasks.

Conclusions:

In this study, the main focus was the training of neural networks to classify the bird species by their calls using spectrograms. Testing with different hyperparameters, binary and multi-

class models were made. The spectrograms were used as an input to the neural network being trained then used to learn the patters and features that were the indicators of bird species. The binary model achieved high accuracy with certain species pairs (Black-capped chickadee and Blue Jay), with the best model achieving 95.45% accuracy. The multi-class model, which trained to classify all 12 species, achieved its best accuracy of 67.24%. Even though some species had some unique features in their calls which makes them easy to classify, there were other species which were almost the same. Although, the predictions are not always correct, the classification of bird species was giving good accuracy in some cases.

Although there were challenges like data imbalance and confusion of some bird calls, through the use of deep learning methods and through the careful selection of model architectures and hyperparameters, there is space to work on the improvement of the bird species classification systems which will therefore be more accurate contributing to the progress in the field of biodiversity.

Bibliography/References:

[1] Vellinga W. *Xeno-canto - Bird sounds from around the world*. Xeno-canto Foundation for Nature Sounds. November, 2018. https://www.gbif.org/dataset/b1047888-ae52-4179-9dd5-5448ea342a24

[2] Krisztian Balog, Linda Cappellato, Nicola Ferro and Craig Macdonald. *Audio Based Bird Species Identification using Deep Learning Techniques*. CEUR Workshop Proceedings: Volume-1609. Evora, Portugal: September, 2016. https://ceur-ws.org/Vol-1609/16090547.pdf

[3] Amazon Web Services (AWS), What is a Neural Network. https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain

[4] Gaurav Gupta, Meghana Kshirsagar, Ming Zhong, Shahrzad Gholami and Juan Lavista Ferres. *Comparing recurrent convolutional neural networks for large scale bird species classification*. Article number: 17085. August, 2021. https://www.nature.com/articles/s41598-021-96446-w

[5] Alteryx Community, It's a No Brainer: An Introduction to Neural Networks https://community.alteryx.com/t5/Data-Science/It-s-a-No-Brainer-An-Introduction-to-Neural-Networks/ba-p/300479

[6] Ketan Doshi. *Audio Deep Learning Made Simple: Sound Classification, Step-by-Step*. Towards Data Science. March, 2021. https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5

[7] Tuomas Oikarinen, Karthik Srinivasan, Olivia Meisner, Julia B. Hyman, Shivangi Parmar, Adrian Fanucci-Kiss, Robert Desimone, Rogier Landman and Guoping Feng. *Deep convolutional network for animal sound classification and source attribution using dual audio recordings*. JASA: PMC6786887. February, 2019. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6786887/

Appendix:

In [1]:
```
import h5py
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import image
import os
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LassoCV, LogisticRegression
from sklearn.preprocessing import LabelBinarizer, StandardScaler, OneHotEncoder, La
from sklearn.metrics import mean_absolute_error, accuracy_score, mean_squared_error
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, Embedding,
from keras.applications import imagenet_utils, ResNet50
from keras.applications.imagenet_utils import preprocess_input
from keras.datasets import mnist, cifar10, imdb
from scipy.sparse import csr_matrix
from scipy import interpolate
from tensorflow.keras.utils import to_categorical
from scipy.io import wavfile
from scipy.signal import spectrogram
from keras.models import load_model
from pydub import AudioSegment
import librosa
```

```
C:\Users\tneelapu\AppData\Roaming\Python\Python39\site-packages\pydub\utils.py:17
0: RuntimeWarning: Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may
not work
  warn("Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may not work",
RuntimeWarning)
```

In [26]:
```
f = h5py.File('spectrograms.h5', 'r')
list(f.keys())
```

Out[26]:
```
['amecro',
 'barswa',
 'bkcchi',
 'blujay',
 'daejun',
 'houfin',
 'mallar3',
 'norfli',
 'rewbla',
 'stejay',
 'wesmea',
 'whcspa']
```

In [27]:
```
for key in f.keys():
    dataset = f[key]
    print(f"Shape of {key}: {dataset.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)
```

Binary Model (selected 3 pairs and choose the one that was giving a better accuracy)

mallar3,blujay

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

mallar3_data = f['mallar3'][:].transpose((2, 0, 1))
blujay_data = f['blujay'][:].transpose((2, 0, 1))

labels_mallar3 = np.zeros(mallar3_data.shape[0])
labels_blujay = np.ones(blujay_data.shape[0])

data = np.concatenate((mallar3_data, blujay_data), axis=0)
labels = np.concatenate((labels_mallar3, labels_blujay), axis=0)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dropout(0.4),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='rmsprop'
              , loss='binary_crossentropy'
              , metrics=['accuracy'])

history = model.fit(X_train, y_train
                    , epochs=20, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
C:\Users\neela\anaconda3\lib\site-packages\keras\src\layers\reshaping\flatten.py:3
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first layer
in the model instead.
  super().__init__(**kwargs)
Test Accuracy: 94.44%
```

In [4]:
```
history = model.fit(X_train, y_train
                    , epochs=20, batch_size=32
                    , validation_split=0.2, verbose=0)
```

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

```
Test Accuracy: 94.44%
```

In [5]:
```
history = model.fit(X_train, y_train
                    , epochs=10, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 94.44%
```

In [7]:
```
history = model.fit(X_train, y_train
                    , epochs=60, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 94.44%
```

All the test accuracies are the same even after changing the batch size and epochs.

mallar3,wesmea

In [9]:
```
f = h5py.File('spectrograms.h5', 'r')

mallar3_data = f['mallar3'][:].transpose((2, 0, 1))
wesmea_data = f['wesmea'][:].transpose((2, 0, 1))

labels_mallar3 = np.zeros(mallar3_data.shape[0])
labels_wesmea = np.ones(wesmea_data.shape[0])

data = np.concatenate((mallar3_data, wesmea_data), axis=0)
labels = np.concatenate((labels_mallar3, labels_wesmea), axis=0)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dropout(0.4),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='rmsprop'
              , loss='binary_crossentropy'
              , metrics=['accuracy'])

history = model.fit(X_train, y_train
                    , epochs=10, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 93.33%
```

In [10]:
```
history = model.fit(X_train, y_train
                    , epochs=10, batch_size=8
```

```
                      , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 93.33%
```

In [12]:
```
history = model.fit(X_train, y_train
                      , epochs=30, batch_size=250
                      , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 93.33%
```

All the test accuracies are the same even after changing the batch size and epochs.

bkcchi,blujay

In [148…
```
f = h5py.File('spectrograms.h5', 'r')

bkcchi_data = f['bkcchi'][:].transpose((2, 0, 1))
blujay_data = f['blujay'][:].transpose((2, 0, 1))

labels_bkcchi = np.zeros(bkcchi_data.shape[0])
labels_blujay = np.ones(blujay_data.shape[0])

data = np.concatenate((bkcchi_data, blujay_data), axis=0)
labels = np.concatenate((labels_bkcchi, labels_blujay), axis=0)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dropout(0.4),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(X_train, y_train, epochs=20, batch_size=64
                      , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 95.45%
```

In [147…
```
history = model.fit(X_train, y_train, epochs=20, batch_size=256
                      , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 77.27%
```

In [149…
```
history = model.fit(X_train, y_train, epochs=20, batch_size=512
                      , validation_split=0.2, verbose=0)
```

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

```
Test Accuracy: 95.45%
```

In [17]:
```
history = model.fit(X_train, y_train, epochs=20, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 95.45%
```

In [139…
```
history = model.fit(X_train, y_train, epochs=10, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 90.91%
```

In [140…
```
history = model.fit(X_train, y_train, epochs=50, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 95.45%
```

In [141…
```
history = model.fit(X_train, y_train, epochs=60, batch_size=128
                    , validation_split=0.2, verbose=0)

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

```
Test Accuracy: 95.45%
```

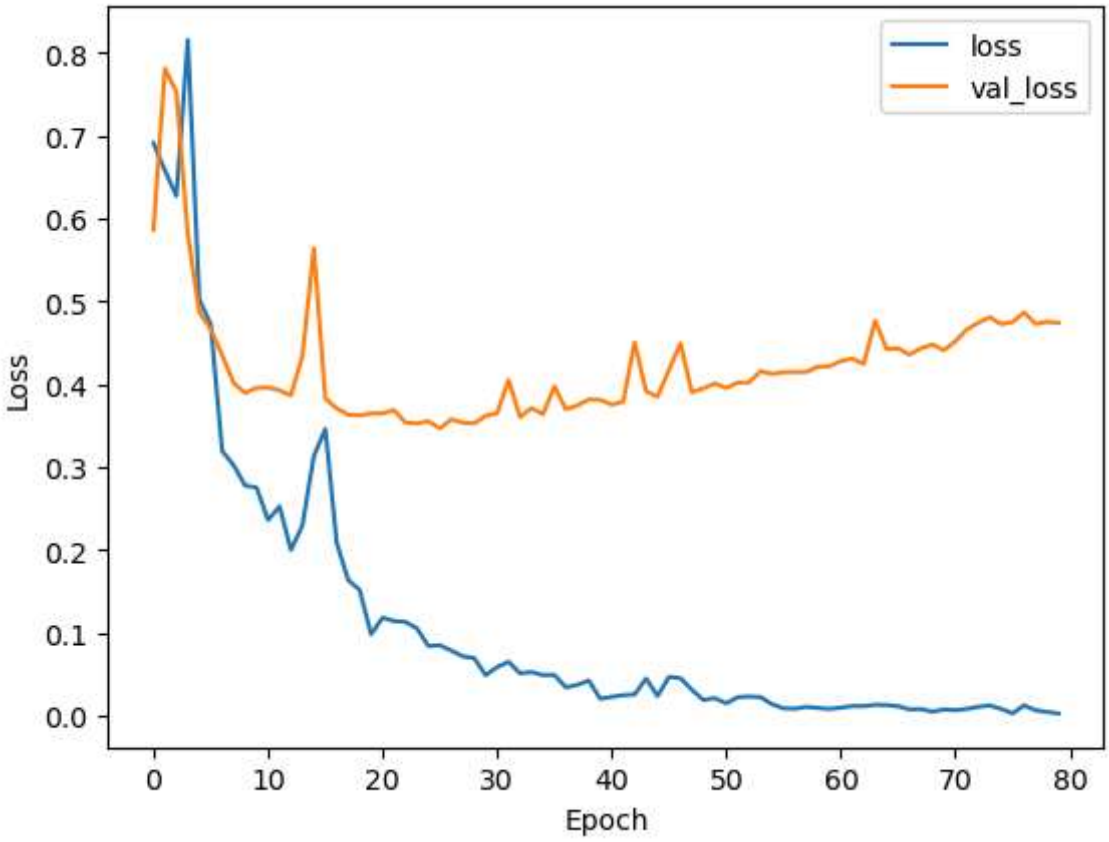
In [46]:
```
%%time

f = h5py.File('spectrograms.h5', 'r')

bkcchi_data = f['bkcchi'][:].transpose((2, 0, 1))
blujay_data = f['blujay'][:].transpose((2, 0, 1))

labels_bkcchi = np.zeros(bkcchi_data.shape[0])
labels_blujay = np.ones(blujay_data.shape[0])

data = np.concatenate((bkcchi_data, blujay_data), axis=0)
labels = np.concatenate((labels_bkcchi, labels_blujay), axis=0)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dropout(0.4),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy']

history = model.fit(X_train, y_train, epochs=60, batch_size=128
                    , validation_split=0.2, verbose=0)
```

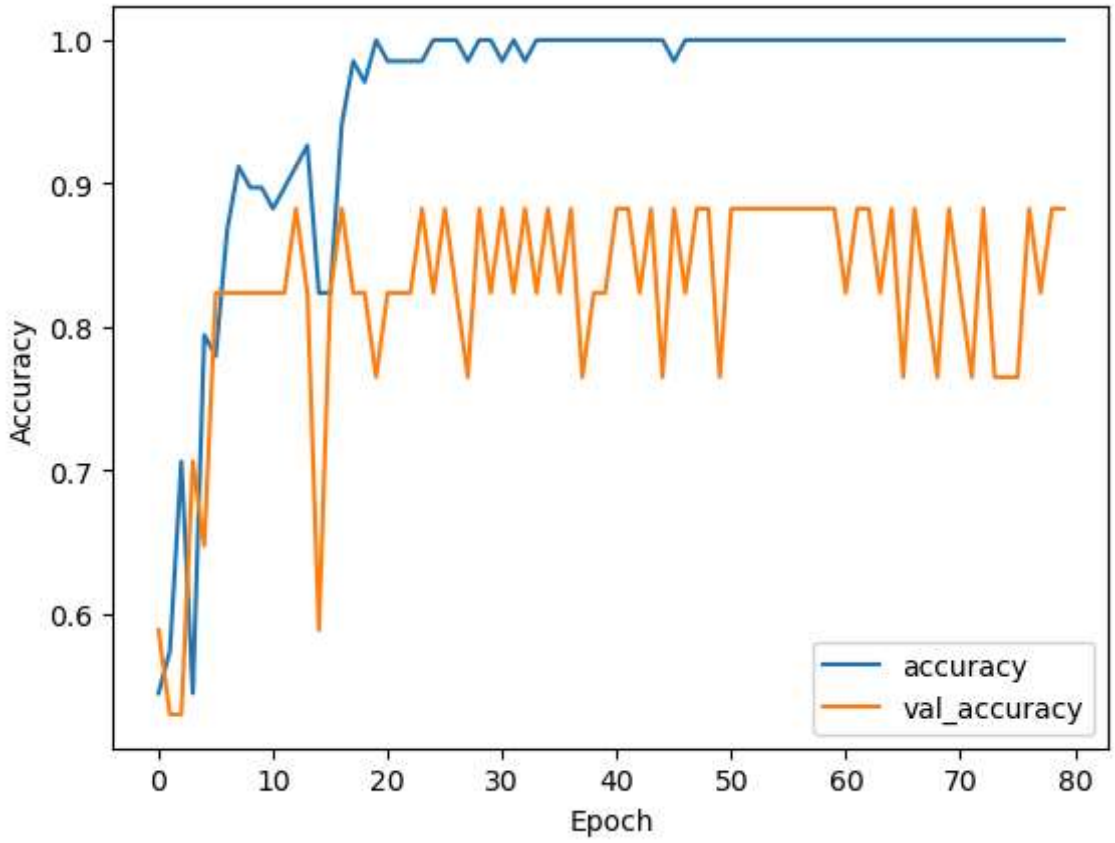
```
score = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(score[1]*100))
```

```
Test Accuracy: 95.45%
Wall time: 5.97 s
```

In [ ]:
```
model.save('spectrograms_model.h5')
```

In [16]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['loss', 'val_loss'], loc='upper right')
plt.show();

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['accuracy', 'val_accuracy'], loc='lower right')
plt.show();
```





While having epochs=60, batch_size=128 the model achieved the maximum (95.45%) accuracy.

Multi-class Model

In [6]:
```
f = h5py.File('spectrograms.h5', 'r')

data_list = []
labels_list = []

species_list = list(f.keys())

for species_key in f.keys():
    species_data = f[species_key][:].transpose((2, 0, 1))
    data_list.append(species_data)
    species_label = np.full((species_data.shape[0],), fill_value=species_list.index
    labels_list.append(species_label)

X = np.concatenate(data_list, axis=0)
y = np.concatenate(labels_list, axis=0)

y_one_hot = to_categorical(y, num_classes=len(species_list))

X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(species_list), activation='softmax')
])
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=50, batch_size=256, validation_split=0

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

```
Test Accuracy: 65.52%
```

In [28]:

```
model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(species_list), activation='softmax')
])

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=50, batch_size=128, validation_split=0

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

```
Test Accuracy: 63.79%
```

In [31]:

```
model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(species_list), activation='softmax')
])

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=20, batch_size=256, validation_split=0

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

```
Test Accuracy: 56.90%
```

In [32]:

```
model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(species_list), activation='softmax')
])

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=40, batch_size=256, validation_split=0

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

```
Test Accuracy: 64.66%
```

In [33]:

```
model = Sequential([
    Flatten(input_shape=(256, 343)),
```

```
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(species_list), activation='softmax')
])

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=60, batch_size=256, validation_split=0

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

```
Test Accuracy: 67.24%
```

In [47]:

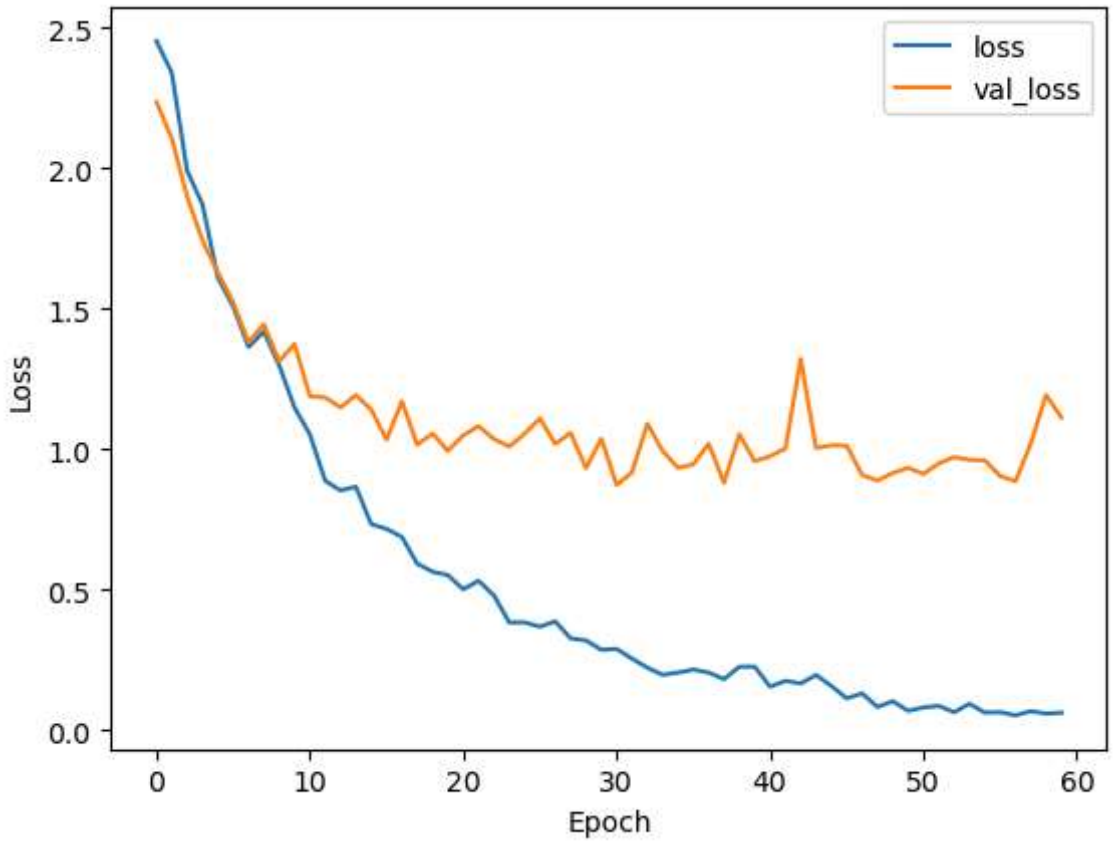
```
%%time

f = h5py.File('spectrograms.h5', 'r')

data_list = []
labels_list = []

species_list = list(f.keys())

for species_key in f.keys():
    species_data = f[species_key][:].transpose((2, 0, 1))
    data_list.append(species_data)
    species_label = np.full((species_data.shape[0],), fill_value=species_list.index
    labels_list.append(species_label)

X = np.concatenate(data_list, axis=0)
y = np.concatenate(labels_list, axis=0)

y_one_hot = to_categorical(y, num_classes=len(species_list))

X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(species_list), activation='softmax')
])

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=60, batch_size=256, validation_split=0

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

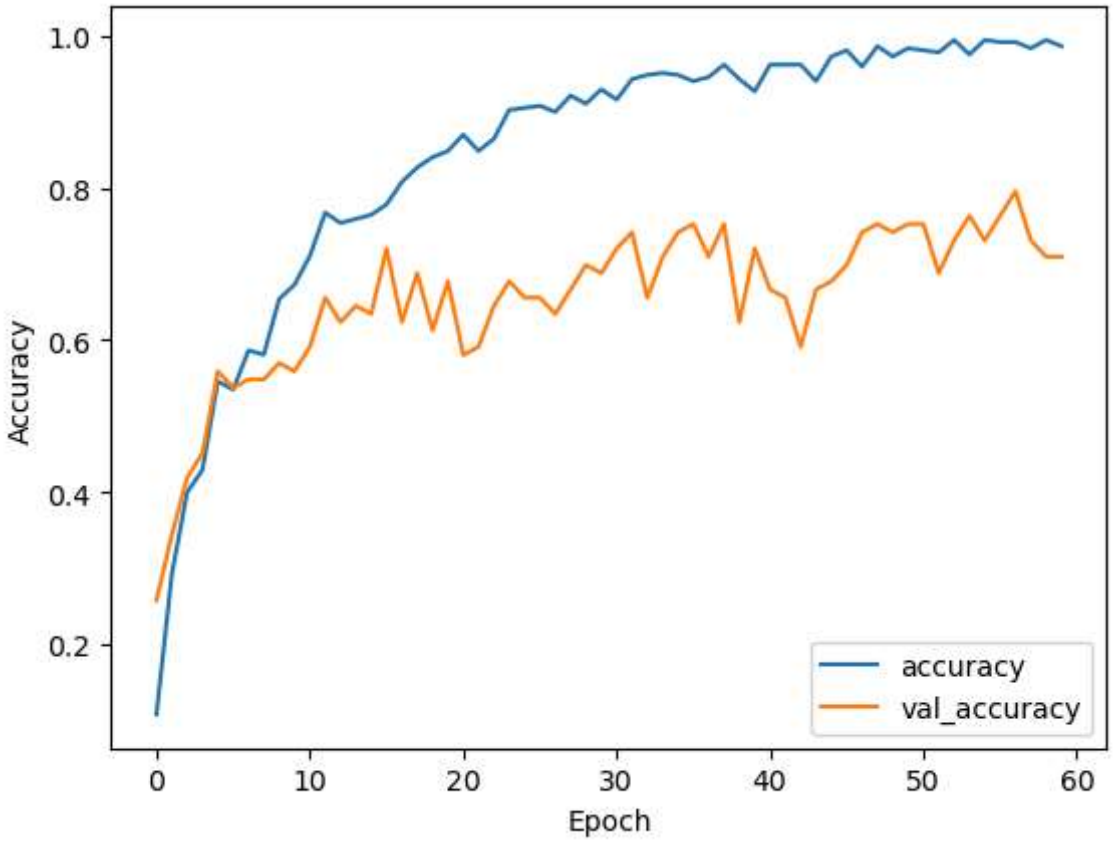
```
Test Accuracy: 64.66%
Wall time: 22.7 s
```

In [37]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['loss', 'val_loss'], loc='upper right')
plt.show();

plt.plot(history.history['accuracy'])
```

```
plt.plot(history.history['val_accuracy'])
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['accuracy', 'val_accuracy'], loc='lower right')
plt.show();
```





The best test accuracy was achieved when epochs=60, batch_size=256 which was 67.24%

External Test data

In [33]:
```
%%time

f = h5py.File('spectrograms.h5', 'r')

data_list = []
labels_list = []

species_list = list(f.keys())

for species_key in f.keys():
    species_data = f[species_key][:].transpose((2, 0, 1))
    data_list.append(species_data)
    species_label = np.full((species_data.shape[0],), fill_value=species_list.index
    labels_list.append(species_label)

X = np.concatenate(data_list, axis=0)
y = np.concatenate(labels_list, axis=0)

y_one_hot = to_categorical(y, num_classes=len(species_list))

X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(species_list), activation='softmax')
])

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accur

history = model.fit(X_train, y_train, epochs=60, batch_size=256, validation_split=0

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

model.save('spectrograms_model.h5')
```

```
Test Accuracy: 66.38%
Wall time: 22.8 s
```

```
C:\ProgramData\Anaconda3\lib\site-packages\keras\src\engine\training.py:3000: User
Warning: You are saving your model as an HDF5 file via `model.save()`. This file f
ormat is considered legacy. We recommend using instead the native Keras format, e.
g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

In [44]:
```
def load_and_convert_to_spectrogram(audio_file):
    y, sr = librosa.load(audio_file, sr=None)
    spectrogram = librosa.feature.melspectrogram(y=y, sr=sr)
    spectrogram_db = librosa.power_to_db(spectrogram, ref=np.max)
    return spectrogram_db

def preprocess_spectrogram(spectrogram):
    resized_spectrogram = np.resize(spectrogram, (256, 343))
    mean = np.mean(resized_spectrogram)
    std = np.std(resized_spectrogram)
    normalized_spectrogram = (resized_spectrogram - mean) / std
    preprocessed_spectrogram = np.expand_dims(normalized_spectrogram, axis=-1)
    return preprocessed_spectrogram

model = load_model('spectrograms_model.h5')
```

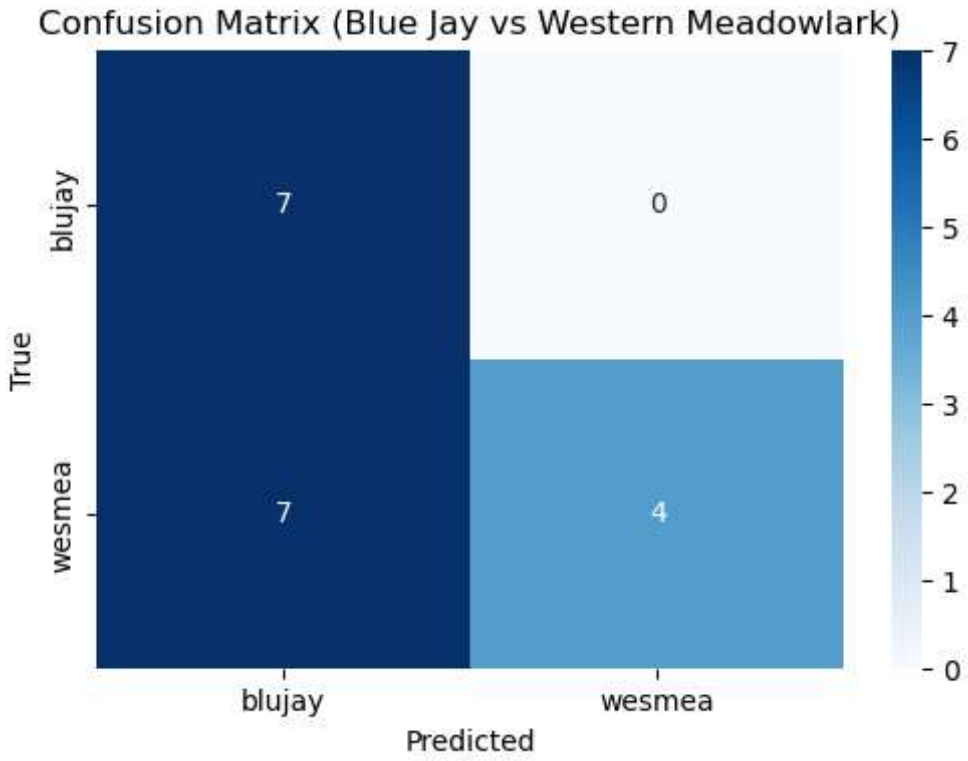
```
audio_files = ["test_birds/test1.mp3", "test_birds/test2.mp3", "test_birds/test3.mp

bird_species = ['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar

print("Results:")
print("=========")
print("| Audio File    | Predicted Bird Probabilities |")
print("|---------------|------------------------------|")
for audio_file in audio_files:
    spectrogram = load_and_convert_to_spectrogram(audio_file)
    preprocessed_spectrogram = preprocess_spectrogram(spectrogram)
    prediction = model.predict(np.array([preprocessed_spectrogram]))[0]
    print(f"| {audio_file} |")
    for species, prob in zip(bird_species, prediction):
        print(f"|               | {species}: {prob:.4f}")
```

```
Results:
=========
| Audio File    | Predicted Bird Probabilities |
|---------------|------------------------------|
1/1 [==============================] - 0s 57ms/step
| test_birds/test1.mp3 |
|               | amecro: 0.0048
|               | barswa: 0.0000
|               | bkcchi: 0.0000
|               | blujay: 0.0000
|               | daejun: 0.0000
|               | houfin: 0.0000
|               | mallar3: 0.0000
|               | norfli: 0.0000
|               | rewbla: 0.0000
|               | stejay: 0.0000
|               | wesmea: 0.0000
|               | whcspa: 0.9952
1/1 [==============================] - 0s 24ms/step
| test_birds/test2.mp3 |
|               | amecro: 0.0000
|               | barswa: 0.0000
|               | bkcchi: 0.0000
|               | blujay: 0.9567
|               | daejun: 0.0000
|               | houfin: 0.0000
|               | mallar3: 0.0433
|               | norfli: 0.0000
|               | rewbla: 0.0000
|               | stejay: 0.0000
|               | wesmea: 0.0000
|               | whcspa: 0.0000
1/1 [==============================] - 0s 25ms/step
| test_birds/test3.mp3 |
|               | amecro: 0.0000
|               | barswa: 0.0000
|               | bkcchi: 0.0000
|               | blujay: 0.0000
|               | daejun: 0.0000
|               | houfin: 0.0000
|               | mallar3: 0.0000
|               | norfli: 0.0000
|               | rewbla: 0.0000
|               | stejay: 0.0000
|               | wesmea: 0.0000
|               | whcspa: 1.0000
```

Confusion Matrix for discussion

In [31]:
```
f = h5py.File('spectrograms.h5', 'r')
blujay_data = f['blujay'][:].transpose((2, 0, 1))
wesmea_data = f['wesmea'][:].transpose((2, 0, 1))

labels_blujay = np.zeros(blujay_data.shape[0])
labels_wesmea = np.ones(wesmea_data.shape[0])

data = np.concatenate((blujay_data, wesmea_data), axis=0)
labels = np.concatenate((labels_blujay, labels_wesmea), axis=0)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, ra

model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dropout(0.4),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=RMSprop(), loss=binary_crossentropy, metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=20, batch_size=64, validation_split=0.

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

y_pred_prob = model.predict(X_test).ravel()
y_pred = (y_pred_prob > 0.5).astype(int)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['blujay', 'wesmea']
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (Blue Jay vs Western Meadowlark)')
plt.show()
```

```
Test Accuracy: 61.11%
1/1 [==============================] - 0s 54ms/step
```


Confusion Matrix (Blue Jay vs Western Meadowlark)
blujay
wesmea
True
7
0
7
4
blujay
wesmea
Predicted
7
6
5
4
3
2
1
0

In [ ]: