Prediction of the species of the birds based on their calls

Name: Lakshit Gupta
Course: DATA 5322-01

Abstract

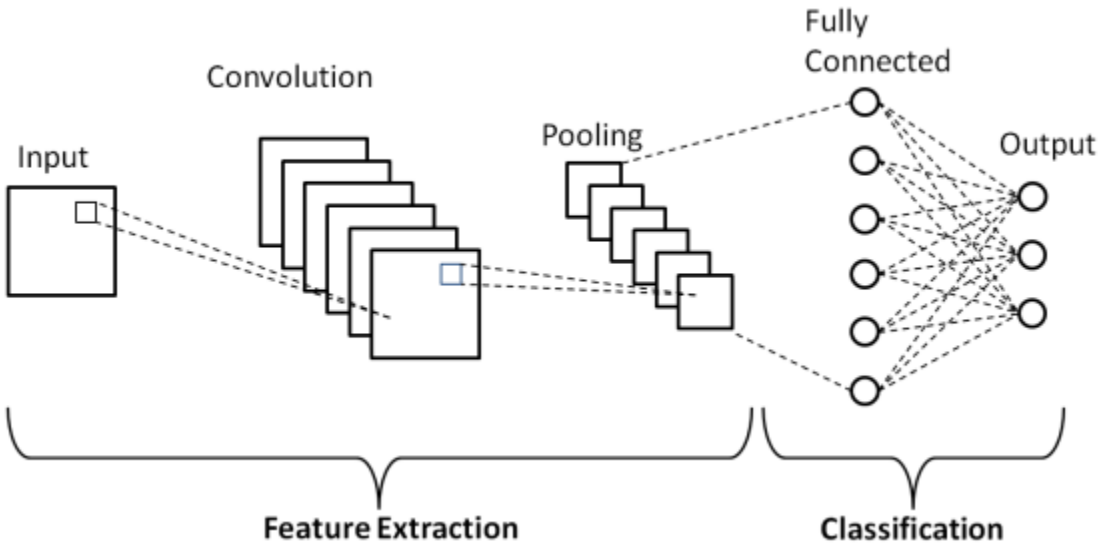
This report focuses on the use of neural networks to classify a number of bird species based on the calls of birds found in the Seattle area. The dataset provides us spectrograms of 10 different audio clips for each of the 12 species of the birds commonly found in the Seattle region American Crow, Barn 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 as well as 3 unlabeled test clips. Two neural network models were used, a binary classifier for two selected species that were blujay and norfli, and a multi-class classifier for all 12 species. Different neural network model architectures like CNN and LSTM along with different hyperparameters were used to get the best performing model with the best optimal parameters. The best multi class model was then used for the 3 test clips to predict which bird species were present in each of the test clips. Problems of overfitted data are also addressed in this report. Overall, this project highlighted that implementing the identification of bird species from audio recordings using deep learning techniques can be done and has potential uses in monitoring the environment and bird population research.

Introduction

Identifying bird species from their calls is an important task in studying the environment. Deep neural networks are used, which are advanced machine learning models, to classify 12 common bird species found in the Seattle area based on spectrograms of their call sounds. Neural networks are good at recognizing complex patterns in data like these spectrograms, making them suitable for this kind of task. But the performance can be affected by things like how different the bird calls sound from each other and if

multiple bird's voices are overlapping.
The main goals was to develop and test neural network models that can classify between two species for the binary classification and all 12 species for multi class classification using the provided data and to check how good these models perform on new, unlabeled test data that may have multiple birds calling, understanding the challenges and the limitations, and explore possible future research work that can be done. Able to accurately recognize bird calls automatically using deep learning could help in monitoring the bird populations and their habitats in nature.

Theoretical Background



[8] Basic structure of a Convolutional Neural Network

Neural Networks:

[1] Neural networks are a form of deep learning algorithms. There are multiple node layers that are there throughout the input, output hidden layers. Different weights and thresholds are attached to these nodes. Neural networks are effective for tasks like image recognition and natural language processing because they can recognize complex patterns from data. However, they can overfit, require large training datasets, and are computationally expensive. Neural networks can learn and multitask, continuously through complex algorithms to learn by themselves and identify solutions to unseen problems, compared to traditional computers that process data sequentially,

making it an important algorithm to classify and predict bird species based on their unique calls.

Convolutional Neural Networks:

[1] Convolutional neural networks (CNNs) represent a specific kind of neural network architecture that is widely used for processing data that has a structure similar to a grid, like images. Using concepts from linear algebra and matrix multiplication, they use convolutional layers resulting in map features that capture and analyze specific areas within visual inputs in order to identify and process patterns. CNNs use pooling layers to reduce the size of feature maps and increase efficiency.

Recurrent Neural Networks:

[1] RNN is one kind of neural network model that works well for processing sequential data. It can be used to process speech, text, or time series data. RNNs are able to retain an internal state or memory because of their unique architecture. By storing and reusing the output produced by its processing nodes, this neural network model allows RNNs to improve their prediction skills by learning from past inputs. Every node processes the data it receives, collecting and using all previously stored data. A feedback loop is a key component of RNNs that allows learning from earlier errors.

Long Short Term Memory:

[2] Long Short-Term Memory (LSTM) is an important type of recurrent neural networks. These were created to address the challenge of learning long term dependencies in sequential data such as speech, text, or time series. The input gate, output gates and the forget gate control a memory cell in LSTMs. By utilizing the information flow in and out of the cells, these gates let the LSTM to store or remove relevant data as it moves throughout the network. Because of these features LSTMs are suitable for tasks such as image and video analysis.

Epochs:

The number of epochs is the total number of times the network is shown the entire training set during training. It is a good practice to count the epochs until the validation accuracy begins to decline, even in cases where the training accuracy is increasing due to overfitting.

Batch Size:

The number of portions of samples that are sent to the network after a parameter changes is known as the batch size.

Activation Functions:

[3] Activation functions are essential in neural networks because they convert the input signals that a neuron receives into an output signal that is transferred to the following layer. In order for neural networks to reproduce the complex, non-linear relationships found in real-world data, these introduce non-linearity. A neural network would only function as a linear regression model in the absence of activation functions, being unable to efficiently solve nonlinear problems. Model's ability to learn complex patterns and make accurate predictions are strongly influenced by the activation function selection. Sigmoid, Softmax, and ReLU functions are examples of common non-linear options. Neural networks can approximate extremely complex functions and change their behavior in response to changing data by using multiple hidden layers with non-linear activations.

Spectrograms:

[4] A spectrogram is a graphic representation that helps to understand and analyze the various pitches or frequencies that make up a sound, as well as how those frequencies change over a certain period of time. The intensity of a given frequency at a given point in time is shown by the different colors in the spectrogram image. This process makes it easy to understand patterns and features in various sounds such as bird calls, which would be challenging to understand just by way of hearing alone.

Methodology

For the preprocessing of the data in case of binary and multi class classification the data provided was already preprocessed in form of HDF5 files. Started by loading the data from those files, then for the case of binary classification two of the bird species were selected ‘blujay’ and ‘norfli’ and were assigned labels as 0 and 1 respectively. Following that, for ensuring the consistency between the dimensions of spectrogram resizing operations were performed to keep the spectrograms of fixed dimension of 343 by 256. After the dimensions were same for both the species, the data was then divided into train and test for training the model and evaluating it, keeping 70% as the training

data and the remaining 30% as the testing data. In case of multi class classifications all the bird species were taken 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 and were assigned labels from 0 to 11 respectively. Same steps were further performed for the multi class as it was done for the binary classification, resizing operations were performed to keep the spectrograms of fixed dimension of 343 by 256. In the case where the external data was tested the 3 mp3 files were stored in a folder for further analysis. Started by resampling the files to a speed rate of 22050 Hz to maintain consistency. Then using the [5]Librosa library's function root mean square was calculated for each of the audio files to identify the loud amplitude levels and mark those as loud. Based on that audio signal within the bird call window is extracted and stored. Maintaining the consistency of the dimensions for the spectrogram, padding and trimming are performed to keep the image of the spectrogram in the size of 343 (time) x 256 (frequency). Finally the spectrogram and the labels assigned to it are appended into a list.

Binary Classification:

Exploration of the spectrograms of the two bird species Blue Jay and Northern Flicker was done. Multiple neural network models are performed for the classification of these two bird species. Firstly a CNN model with 3 max pooling, 1 flatten layer and 3 convolutional layers with 'relu' as the activation function for the input layers and 'sigmoid' as the activation function for the dense output layer is performed for the binary classification. Then 3 more models were implemented with varying epochs and batch sizes with the same activation functions to check how the performance of the model changes with these tunings. After that the model which performed the best, having those configurations we added a new dropout layer which further improved the performance of our model.

Multi-Class Classification:

After the preprocessing was done on the multi-class classification data, the bird species were labeled from 0-11 in sequence. Multiple models were evaluated for this problem. Model 1 is a Convolutional Neural Network (CNN) comprising convolutional and max-pooling layers followed by dense layers. It's trained using categorical cross-entropy loss. Model 2 is similar but trained for a longer duration with more epochs and a larger batch size. Model 3 is the same as Model 2 but with an even larger batch size for potentially faster training. Model 4 is also a CNN but includes Dropout layers after

dense layers to prevent overfitting. Lastly, Model 5 is a Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) cells, suitable for sequential data like spectrograms. It's trained with dropout regularization as well. Each model aims to learn patterns in audio data and classify them into different labeled categories of bird species, based on the features extracted from spectrograms.

External Test Data:

The prediction was done on the external test data using the best model from the multi class classification. Firstly, the test data was taken and prepared to fit into the trained model. Then, using the best-trained model, which was saved earlier, predictions on the test data were done, predicting which bird species each sound belongs to. After predicting, the guesses were sorted to see which bird species the model thinks are most likely. Apart from that probabilities were also calculated by what percent our model is sure that the mp3 file has those bird species in it.

Computational Results

- **Binary Classification:**

  In the binary classification, all models achieved perfect accuracy of 100% on both the training and test data, indicating that they were able to accurately classify bird species into their respective labels. Model 1, trained for 10 epochs with a batch size of 15, had the highest training loss of 3.14% and a test loss of 1.37%. Model 2, trained for 20 epochs with a batch size of 30, had a slightly lower training loss of 2.22% and an even lower test loss of 0.96%. Model 3, trained for 20 epochs with a batch size of 50, had a similar performance to Model 2 with a training loss of 2.22% and a test loss of 0.995%. Model 4, which included dropout layers, achieved the lowest training loss of 0.91% and the lowest test loss of 0.88%. The RNN model, also with dropout layers, had a slightly higher training loss of 1.07% and a test loss of 1.63%. Overall, all models performed well, showing which model performed the best in classifying different species correctly.

Models (Epochs, batch size)	Train Acc.	Train Loss	Test Acc.	Test Loss
CNN Model 1 (10,15)	100%	3.14%	100%	1.37%
CNN Model 2 (20,30)	100%	2.22%	100%	0.96%
CNN Model 3 (20,50)	100%	2.22%	100%	0.995%
CNN Model 4 (20,50) + dropout layer	100%	0.91%	100%	0.88%
RNN Model (20,50) + dropout layer	100%	1.07%	100%	1.63%

- **Multi Class Classification:**

  In the multi-class classification, Model 1, trained for 10 epochs with a batch size of 15, achieved a training accuracy of 92.48% and a test accuracy of 99.89%, with corresponding losses of 53.32% and 2.3%, respectively. Model 2, trained for 20 epochs with a batch size of 30, had a higher training accuracy of 94.21% and a slightly lower test accuracy of 99.13%, with losses of 31.32% and 4.26%. Model 3, trained with the increased batch size of 50, showed similar performance with a training accuracy of 94.38% and a test accuracy of 99.35%, along with losses of 21.32% and 2.29%. Model 4, which was with dropout layers, had improved performance with a training accuracy of 96.38% and a test accuracy of 99.89%, with losses of 11.38% and 5.97%. The LSTM model, also with dropout layers, exhibited lower training accuracy of 82.22% and a test accuracy of 93.82%, with higher losses of 56.95% and 29.80%, respectively. Overall, Model 4 had the best performance among the models, achieving high accuracy and minimum loss on both training and test data.

Models (Epochs, batch size)	Train Acc.	Train Loss	Test Acc.	Test Loss
CNN Model 1 (10,15)	92.48%	53.32%	2.3%	1.37%
CNN Model 2 (20,30)	94.21%	31.32%	99.13%	4.26%
CNN Model 3 (20,50)	94.38%	21.32%	99.35%	2.29%

CNN Model 4 (20,50) + dropout layer	96.38%	11.38%	99.89%	5.97%
RNN Model (20,50) + dropout layer	82.22%	56.95%	93.82%	29.80%

- **External Test Data:**

  For external test data, predictions were made for each test sample. For "test1", the model predicted the most likely bird species to be Northern Flicker with a probability of 0.966, followed by Mallard and American Crow with probabilities of 0.018 and 0.012, respectively. Similarly, for "test2", the model predicted Northern Flicker with a probability of 0.986 as the most expected species, followed by Mallard and American Crow. And same is for "test3", Northern Flicker was again predicted as the top species with a probability of 0.981, followed by Mallard and American Crow. The predictions show Northern Flicker as the most probable species across all test samples, followed by other species such as Mallard and American Crow.

Test Sample	Species	Probability
test1.mp3	Northern Flicker	96.62%
	Mallard	1.79%
	American Crow	1.17%
	Western Meadowlark	0.38%
	Dark-eyed Junco	0.02%
test2.mp3	Northern Flicker	98.58%
	Mallard	0.93%
	American Crow	0.40%
	Western Meadowlark	0.08%,
	Red-winged Blackbird	0.004%

test3.mp3	Northern Flicker	98.13%
	Mallard	1.48%
	American Crow	0.23%
	Western Meadowlark	0.15%,
	Dark-eyed Junco	0.005%

Discussion

- **Binary Classification:**

  In Binary classification Blue Jay and Northern Flicker bird species are being classified.

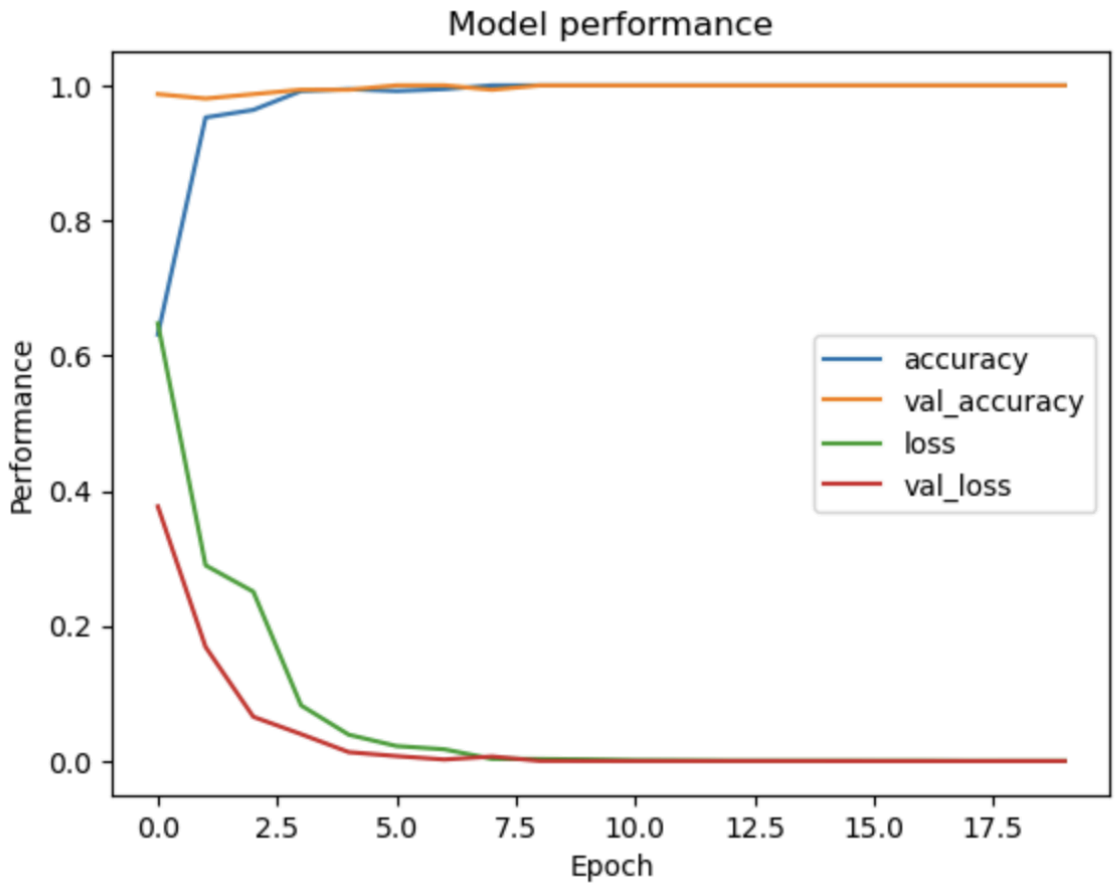


Figure 1. CNN model with dropout (Epoch: 20 and Batchsize: 50)

The epoch number is displayed on the x-axis, and different performance metrics between 0 and 1 are displayed on the y-axis. The model's accuracy on the training set is shown by the blue line, which starts lower and increases quickly to indicate that learning was effective. The validation accuracy on unseen data is represented by the orange line, which shows the same upward trend but at a slightly reduced level. The training loss is represented by the green line and decreases quickly as the model gets better. The validation loss is represented by the red line, which decreases correspondingly but remains higher than the training loss. So, we can say that the model appears to be learning effectively from the training data based on increasing accuracy and decreasing loss.

The difference between the training and test shows the model may be overfitting as it performs extremely well on training data but does not apply as well to new, unseen test data.

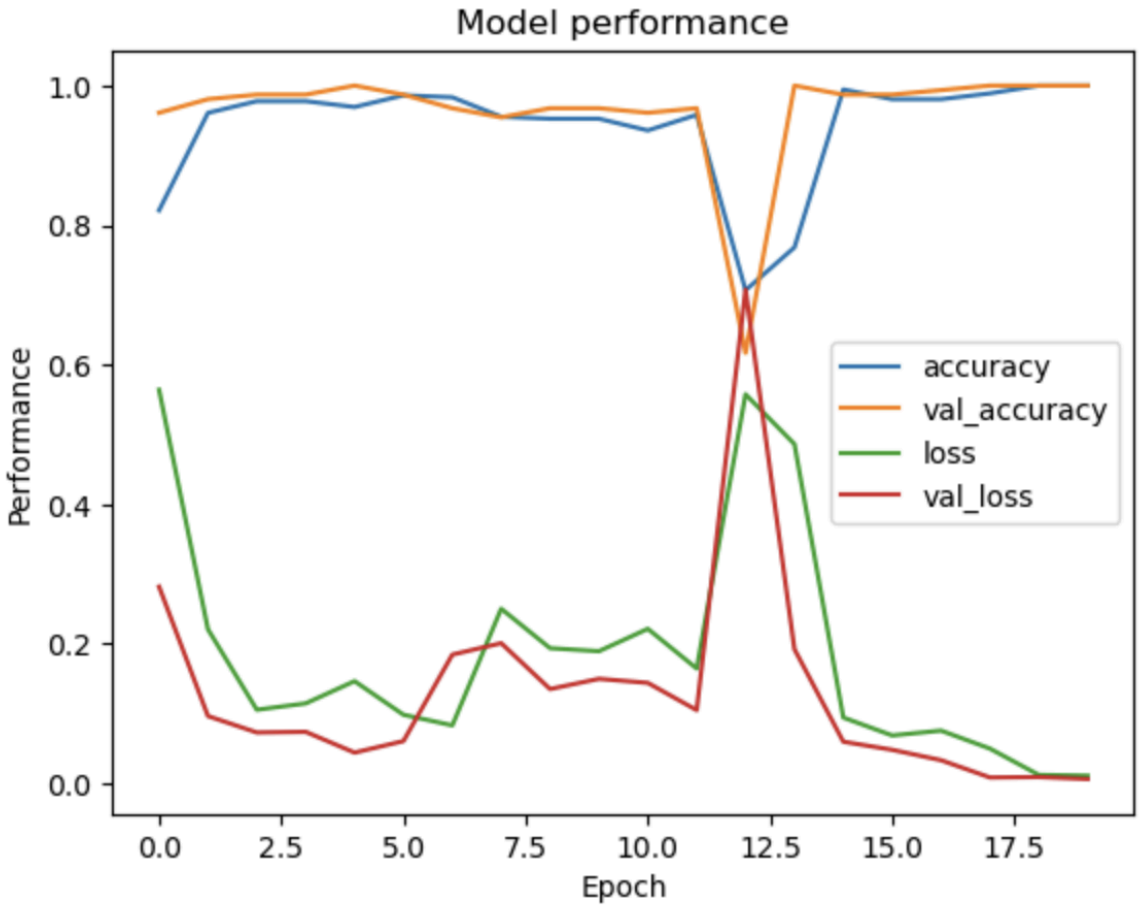


Figure 2. RNN model with dropout (Epoch: 20 and Batchsize: 50)

From the above plot we can say that overfitting to this extent is not good for our model and also cannot expect it to make accurate predictions on the unseen data.

- **Multi Class Classification:**

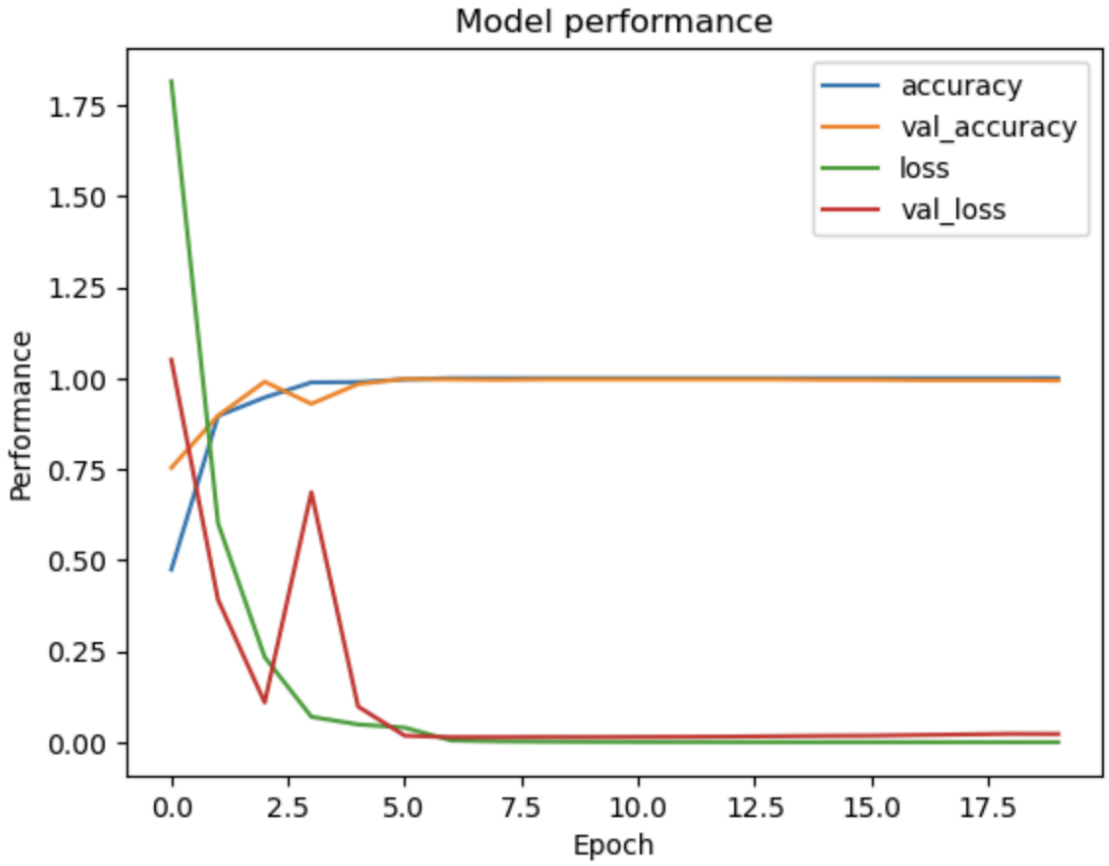


Figure 3. CNN model without dropout (Epoch: 20 and Batchsize: 50)

This shows that some of the patterns from the training data have been learned by the model, allowing high accuracy. But the difference between training and validation data shows that it has not generalized enough and may have some issues with completely new unseen data that wasn't part of its training data.

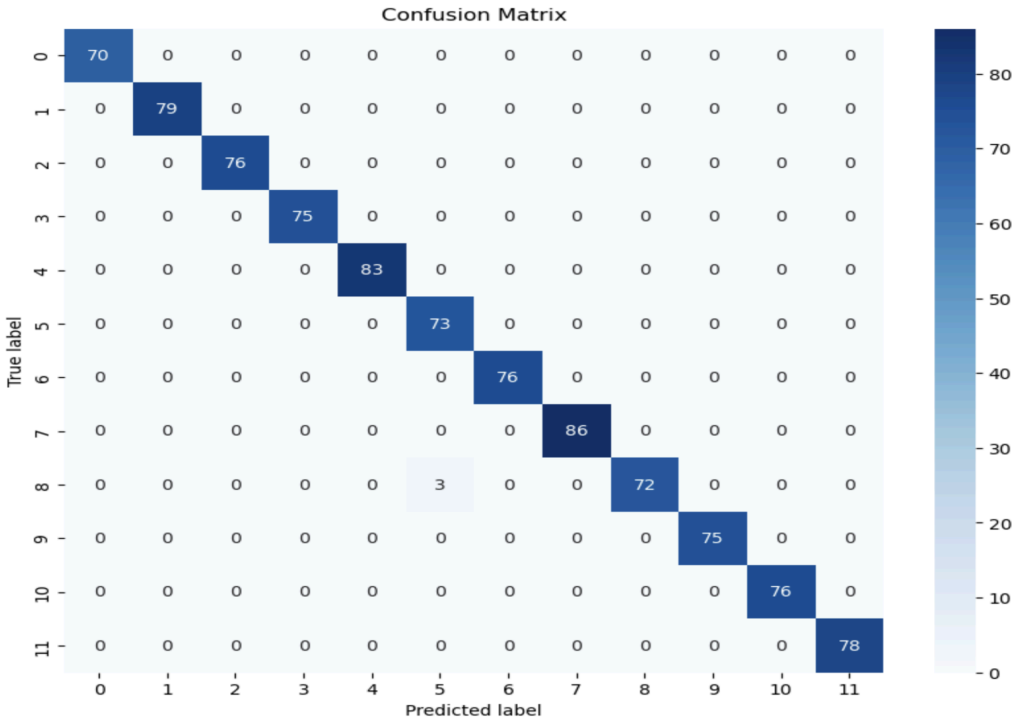
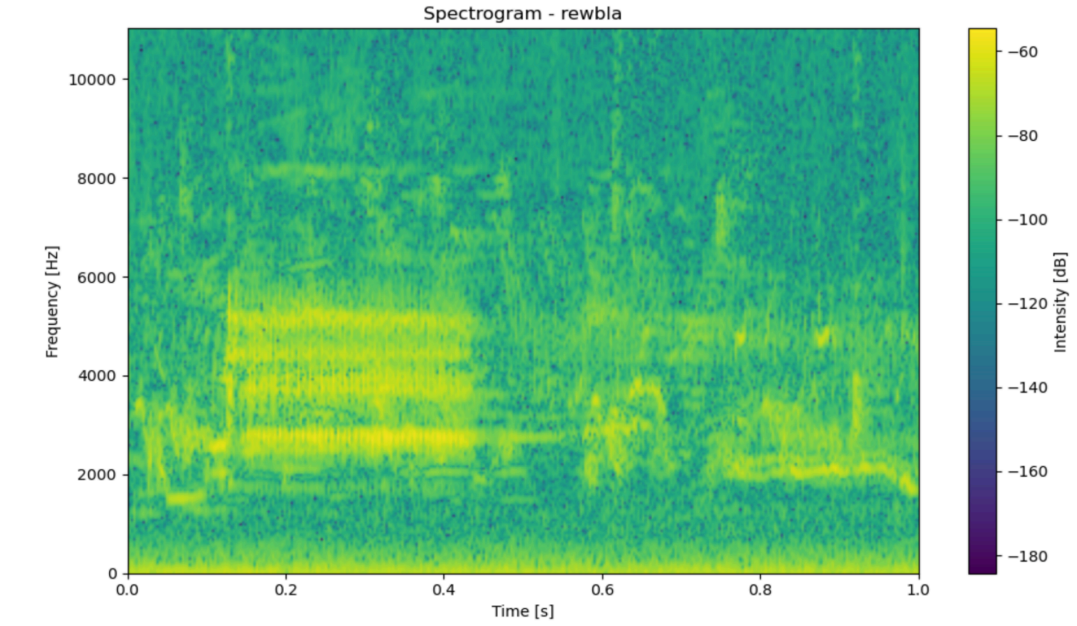
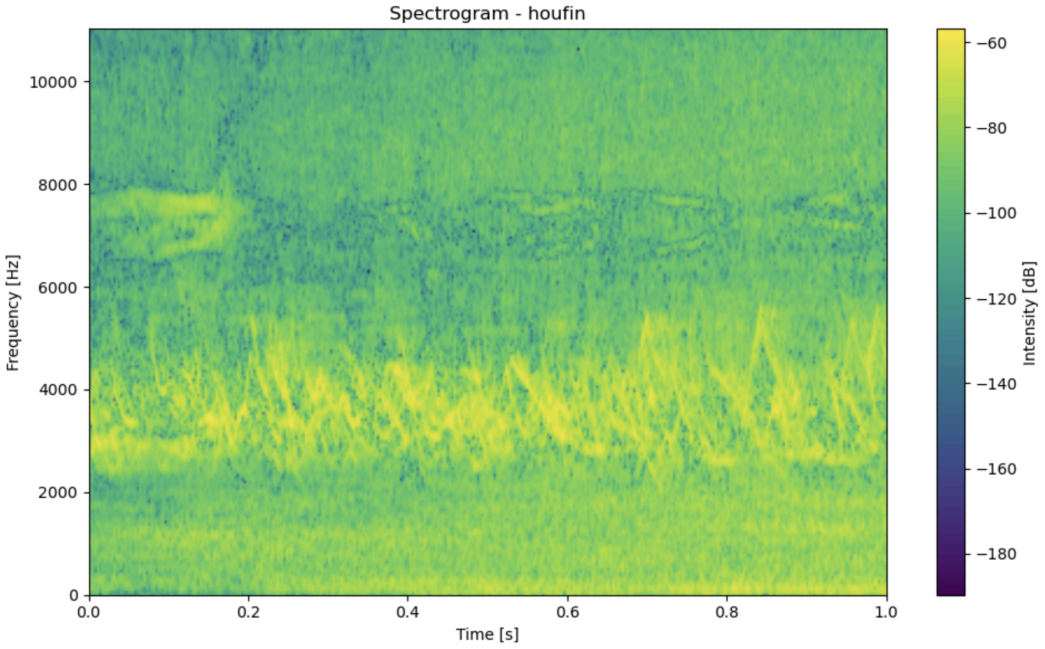


Figure 4. Confusion matrix for CNN model without dropout (Epoch: 20 and Batchsize: 50)

From the above confusion matrix we can see that Red-winged Blackbird that is label 8 is being misclassified as House Finch which is label 5.





From the above spectrograms of the two bird species we can say that there is a possibility of ‘rewbla’ being misclassified as ‘houfin’ as at higher amplitudes both the sounds look kind of the same.

- **External Test Data:**

  The model always predicts the bird species Northern Flicker with the highest probability for the majority of the audio files when the testing was done. This shows that the model predicts Northern Flicker more often than it does other bird species. There might be multiple reasons for this. Firstly, the model could start to favor Northern Flicker if the dataset has a greater number of that species in the audio files than other species. Also, the data for Northern Flicker may contain unique features, for example high pitch sound and frequency which help with the model's predictions.

  In the audio file test1.mp3 there seems to be the most number of bird species in that clip as the model is predicting the top 3 birds with more probability in this clip than the others.

What limitations did you run into in this homework? How long did it take to train the models?

Some of the models were not running on the personal system as it required high computational power so used the Data Science computers for this homework. All the models for the multi-class took around 10-15 min to run. Varying parameters were used for each model so every model took a different amount of run time.

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?

For the multi class classification ‘rewbla’ was the most challenging to predict as it was continuously getting confused with ‘houfin’ due to similar spectrogram windows. Some of the birds can be confused for others when they have similar kind of bird calls or when there is a considerable amount of background noise in the mp3 files.

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

A number of models, such as VGGNet, CRNN (Convolutional Recurrent Neural Networks) and YAMNet could have been used. A neural network can identify greater detailed patterns in the data so we can say that it is a good choice for this task. Neural networks perform better at understanding the complex sounds and patterns of bird calls.

It can look at all of these details and develop a model that distinguishes better between various bird species.

Conclusion

In this project, different models were used to classify bird species based on their calls. First, models were trained to distinguish between two bird species, Blue Jay and Northern Flicker, getting perfect accuracy with various neural network models. Then, starting with multi class classification, this was to classify twelve bird species, achieving high accuracy with different models, including CNNs and RNNs. The models performed well, but some showed signs of overfitting, where they learned too much from the training data and did not do well with the new unseen test data. When testing with 3 different external mp3 audio files, the model consistently predicted one species, Northern Flicker, more often than others, possibly because of its unique features or maybe because it is present more in the training data. Overall, the models performed well in classifying bird species based on their calls, but further improvements could be made to handle the complexity of a bigger dataset better.

References

1. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/
2. https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/
3. https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf
4. https://www.izotope.com/en/learn/understanding-spectrograms.html#:~:text=Waveform,with%20variable%20brightness%20or%20color.
5. https://www.linkedin.com/pulse/audio-classification-tensorflow-keras-librosa-srikanth-chellappa-t1rvc/
6. https://www.ibm.com/topics/neural-networks (Neural Networks and CNN)
7. https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a
8. https://www.upgrad.com/blog/cnn-vs-rnn/

Appendix

```
# Import libraries
import h5py
import numpy as np
import random
import os
import librosa
import librosa.display
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, LSTM,SimpleRNN
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
from keras.models import Sequential
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix
import seaborn as sns
from scipy.signal import spectrogram
from scipy import signal
from tensorflow.keras.models import load_model
import pandas as pd
import warnings
warnings.filterwarnings("ignore")


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


list(f.items())


for species, dataset in f.items():
    spectrogram_data = np.array(dataset)
    frequencies, times, spectrogram = signal.spectrogram(spectrogram_data, fs=22050)
    spectrogram = np.squeeze(spectrogram)
    print("Times shape:", times.shape)
    print("Frequencies shape:", frequencies.shape)
    print("Spectrogram shape:", spectrogram.shape)

    if times.shape[0] != spectrogram.shape[1]:
        times = np.linspace(0, 1, spectrogram.shape[1])
```

```
    if frequencies.shape[0] != spectrogram.shape[0]:
        frequencies = np.linspace(0, 22050 / 2, spectrogram.shape[0])

    # Plot the spectrogram
    plt.figure(figsize=(10, 6))
    plt.pcolormesh(times, frequencies, 10 * np.log10(spectrogram[:, :, 0]), shading='gouraud',
cmap='viridis')
    plt.title(f"Spectrogram - {species}")
    plt.xlabel("Time [s]")
    plt.ylabel("Frequency [Hz]")
    plt.colorbar(label='Intensity [dB]')
    plt.tight_layout()
    plt.show()


blujay_features = []
norfli_features = []
labels = []

for key in f.keys():
    if key == 'blujay':
        blujay_features.append(np.array(f[key]))
        labels.extend([0] * f[key].shape[0])
    elif key == 'norfli':
        norfli_data = np.array(f[key])[:, :, :50]
        norfli_features.append(norfli_data)
        labels.extend([1] * norfli_data.shape[0])


blujay_features = np.concatenate(blujay_features, axis=0)
norfli_features = np.concatenate(norfli_features, axis=0)


def plot_waveform(features, title, color):
    plt.figure(figsize=(12, 6))
    for i in range(features.shape[0]):
        plt.plot(features[i], color=color, alpha=0.7, linewidth=0.7)
    plt.title(title)
    plt.xlabel('Sample')
    plt.ylabel('Amplitude')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

# Plot waveforms for blujay
plot_waveform(blujay_features, 'Waveform - Blujay', 'blue')
```

```
# Plot waveforms for norfli
plot_waveform(norfli_features, 'Waveform - Norfli', 'red')

features = np.concatenate((blujay_features, norfli_features), axis=0)
labels = np.array(labels)


print("Features shape:", features.shape)
print("Labels shape:", labels.shape)




np.random.seed(123)
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.3, random_state=123)


print("Size of X_train:", X_train.shape)
print("Size of X_test:", X_test.shape)
print("Size of y_train:", y_train.shape)
print("Size of y_test:", y_test.shape)
```

Binary Model

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 50, 1)),
    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')
])


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

history = model.fit(X_train, y_train, epochs=10, batch_size=15, validation_data=(X_test, y_test))


plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
```

```
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()


test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

With increased epoch to 20 and batch size to 30

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 50, 1)),
    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')
])


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

history = model.fit(X_train, y_train, epochs=20, batch_size=30, validation_data=(X_test, y_test))


plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
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()


test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

With increased epoch to 20 and batch size to 50

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 50, 1)),
    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')
])


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

history = model.fit(X_train, y_train, epochs=20, batch_size=50, validation_data=(X_test, y_test))


plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
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()


test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

With Dropout

```
modeldrop = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 50, 1)),
    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(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])


modeldrop.compile(optimizer=Adam()
                  , loss='binary_crossentropy'
                  , metrics=['accuracy'])

historydrop = modeldrop.fit(X_train, y_train, epochs=20, batch_size=50, validation_data=(X_test,
y_test))


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


test_loss, test_accuracy = modeldrop.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

RNN model

```
X_train_rnn = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2])
X_test_rnn = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2])

rnn_binmodel = Sequential([
    SimpleRNN(64, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    Dense(128, activation='relu'),
    Dropout(0.5),
```

```
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

rnn_binmodel.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

history_binrnn = rnn_binmodel.fit(X_train_rnn, y_train, epochs=20, batch_size=50,
validation_data=(X_test_rnn, y_test))


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


test_loss, test_accuracy = rnn_binmodel.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

Multi Class Model

```
import numpy as np

# Minimum number of slices
min_slices = 36

features_list = []
labels_list = []
species_names = []

label_mapping = {species: i for i, (species, _) in enumerate(f.items(), start=0)}

for species, data in f.items():
    feature_data = np.array(data)
    num_slices = feature_data.shape[2]
    if num_slices < min_slices:
```

```
        padded_data = np.pad(feature_data, ((0, 0), (0, 0), (0, min_slices - num_slices)),
mode='constant')
        features_list.append(padded_data)
    elif num_slices > min_slices:
        cropped_data = feature_data[:, :, :min_slices]
        features_list.append(cropped_data)
    else:
        features_list.append(feature_data)

    labels_list.extend([label_mapping[species]] * feature_data.shape[0])
    species_names.extend([species] * feature_data.shape[0])

features = np.concatenate(features_list, axis=0)
labels = np.array(labels_list)

labels_one_hot = to_categorical(labels - 1, num_classes=len(np.unique(labels)))

for species, label in zip(species_names, labels):
    print(f"Bird species: {species}, Class Label: {label}")

# Print shapes to verify
print("Features shape:", features.shape)
print("Labels shape:", labels.shape)


np.random.seed(123)
X_train, X_test, y_train, y_test = train_test_split(features, labels_one_hot, test_size=0.3,
random_state=123)


print("Size of X_train:", X_train.shape)
print("Size of X_test:", X_test.shape)
print("Size of y_train:", y_train.shape)
print("Size of y_test:", y_test.shape)




multimodel = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 36, 1)),
    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(len(np.unique(labels)), activation='softmax')
])

multimodel.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])


multimodel.summary()


historymulti = multimodel.fit(X_train, y_train, epochs=10, batch_size=15, validation_data=(X_test,
y_test))


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


test_loss, test_accuracy = multimodel.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

With increased epoch to 20 and batch size to 30

```
multimodel = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 36, 1)),
    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(len(np.unique(labels)), activation='softmax')
])

multimodel.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])


historymulti = multimodel.fit(X_train, y_train, epochs=20, batch_size=30, validation_data=(X_test,
y_test))
```

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


test_loss, test_accuracy = multimodel.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

With increased epoch to 20 and batch size to 50

```
multimodel = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 36, 1)),
    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(len(np.unique(labels)), activation='softmax')
])

multimodel.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])


historymulti = multimodel.fit(X_train, y_train, epochs=20, batch_size=50, validation_data=(X_test,
y_test))


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

```
test_loss, test_accuracy = multimodel.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

With dropout

```
multimodeldrop = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(343, 36, 1)),
    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'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(labels)), activation='softmax')
])

multimodeldrop.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])


multimodeldrop.summary()


historymultidrop = multimodeldrop.fit(X_train, y_train, epochs=20, batch_size=50,
validation_data=(X_test, y_test))


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


test_loss, test_accuracy = multimodeldrop.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
y_pred = multimodeldrop.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_test, axis=1)

conf_matrix = confusion_matrix(y_true_labels, y_pred_labels)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()


model_save_path =
'/Users/lakshitgupta/Library/CloudStorage/OneDrive-SeattleUniversity/Quater3/Machine
Learning-2/Written Homeworks/multi_model.h5'
multimodeldrop.save(model_save_path)
print("Model saved successfully.")
```

LSTM model

```
X_train_rnn = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2])
X_test_rnn = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2])

rnn_model = Sequential([
    LSTM(128, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(labels)), activation='softmax')
])

rnn_model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

history_rnn = rnn_model.fit(X_train_rnn, y_train, epochs=20, batch_size=50,
validation_data=(X_test_rnn, y_test))


plt.plot(history_rnn.history['accuracy'], label='accuracy')
plt.plot(history_rnn.history['val_accuracy'], label='val_accuracy')
plt.plot(history_rnn.history['loss'], label='loss')
plt.plot(history_rnn.history['val_loss'], label='val_loss')
plt.title('Model performance')
```

```
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()


test_loss, test_accuracy = rnn_model.evaluate(X_test_rnn, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

External test data

```
def preprocess_mp3(audio_file_path, new_sample_rate=22050, min_loud_part_length=0.5,
bird_call_window_size=2,
                   n_fft=2048, hop_length=512):
    y, sr = librosa.load(audio_file_path, sr=new_sample_rate)
    energy = librosa.feature.rms(y=y, frame_length=n_fft, hop_length=hop_length)
    loud_indexes = np.where(energy > np.max(energy) * 0.5)[1]
    spectrograms = []
    labels = []

    for loud_idx in loud_indexes:
        start_time = loud_idx * hop_length / sr
        end_time = start_time + bird_call_window_size

        bird_call_audio = y[int(start_time * sr):int(end_time * sr)]

        spectrogram = librosa.feature.melspectrogram(y=bird_call_audio, sr=sr, n_fft=n_fft,
hop_length=hop_length,
                                                     n_mels=36, fmax=8000)

        if spectrogram.shape[1] < 343:
            padding = np.zeros((36, 343 - spectrogram.shape[1]))
            spectrogram = np.hstack((spectrogram, padding))
        elif spectrogram.shape[1] > 343:

            spectrogram = spectrogram[:, :343]

        spectrograms.append(spectrogram)
        labels.append(os.path.splitext(os.path.basename(audio_file_path))[0])

    return spectrograms, labels

folder_path = 'test_birds'
```

```
all_spectrograms = []
all_labels = []

for audio_file in os.listdir(folder_path):
    if audio_file.endswith('.mp3'):
        audio_file_path = os.path.join(folder_path, audio_file)
        spectrograms, labels = preprocess_mp3(audio_file_path)
        all_spectrograms.extend(spectrograms)
        all_labels.extend(labels)

all_spectrograms = np.array(all_spectrograms)
all_labels = np.array(all_labels)


audio="/Users/lakshitgupta/Downloads/test_birds/test1.mp3"
waves,freq=librosa.load(audio)
audio1="/Users/lakshitgupta/Downloads/test_birds/test2.mp3"
waves1,freq1=librosa.load(audio1)
audio2="/Users/lakshitgupta/Downloads/test_birds/test3.mp3"
waves2,freq2=librosa.load(audio2)
plt.specgram(waves, Fs=freq)
plt.xlabel("Time [s]")
plt.ylabel("Frequency");
plt.title('Audio 1')
plt.show()


plt.specgram(waves1, Fs=freq1)
plt.xlabel("Time [s]")
plt.ylabel("Frequency");
plt.title('Audio 2')
plt.show()


plt.specgram(waves2, Fs=freq2)
plt.xlabel("Time [s]")
plt.ylabel("Frequency");
plt.title('Audio 3')
plt.show()


print(all_labels.shape)
print(all_spectrograms.shape)


test_spectrograms_reshaped = all_spectrograms.reshape(-1, 343, 36, 1)
```

```
model_save_path =
'/Users/lakshitgupta/Library/CloudStorage/OneDrive-SeattleUniversity/Quater3/Machine
Learning-2/Written Homeworks/multi_model.h5'
multimodel = load_model(model_save_path)
multimodel.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
predictions = multimodel.predict(test_spectrograms_reshaped)

# Mapping of class indices to bird species names
class_to_species = {
    0: 'amecro', 1: 'barswa', 2: 'bkcchi', 3: 'blujay', 4: 'daejun', 5: 'houfin',
    6: 'mallar3', 7: 'norfli', 8: 'rewbla', 9: 'stejay', 10: 'wesmea', 11: 'whcspa'
}


file_names = [file.split('.')[0] for file in os.listdir(folder_path) if file.endswith('.mp3')]

predicted_species_with_labels = []

for i, file_name in enumerate(file_names):
    species_probs = predictions[i]
    species_predictions = [(class_to_species[j], prob) for j, prob in enumerate(species_probs)]
    species_predictions.sort(key=lambda x: x[1], reverse=True)
    predicted_species_with_labels.append((file_name, species_predictions))

# Print the predicted bird species along with their original labels
#for file_name, species_probs in predicted_species_with_labels:
#    print("Original Label:", file_name)
#    for species, prob in species_probs:
#        print("Predicted Bird Species:", species, "- Probability:", prob)

for file_name, species_probs in predicted_species_with_labels:
    print("Original Label:", file_name)
    for i, (species, prob) in enumerate(species_probs[:5]):
        species_key = [key for key, value in class_to_species.items() if value == species][0]
        print(f"Top {i+1} Predicted Bird Species:", species, "- Class:", species_key, "- Probability:",
 prob)
```