Statistical Machine Learning 2

Deep Learning

Nidhi Trivedi

Birds Classification on the basis of Sound

Abstract

This report is about the investigation of the bird species classification through the sound analysis using the neural networks. The main goal is to find the bird species by the sound. The dataset used for this research is firstly from Xeno-Canto[1] which is a crowd-sourced bird sounds archive. The pre-processed data has been utilized in this study which already had 10 audio clips for 12 bird species converted to spectrograms. The spectogram captures the time-frequency representations of the bird calls. The study implements two distinct models: the binary classification model to classify two selected bird species, 'Northern Flicker' (norfli) and 'Blue Jay' (blujay) and the multiclass model which will classify all 12 species. The binary classification has achieved accuracy of 86.36% for batch size 128 while the multiclass has achieved the accuracy of 66.38% for batch size 128. The project includes an analysis of three audio clips where the trained multi-class model is used to predict the bird species present in each clip. For all the three audio American Crow has been predicted. The report has the methodology, computational results and discussion that are related to the performance, limitations and improvements of neural network models. The test data of external sound clips are used to check the multi class models predictions. Overall, this project demonstrates the application of nueral networks and spectogram analysis for bird species identification.

Introduction

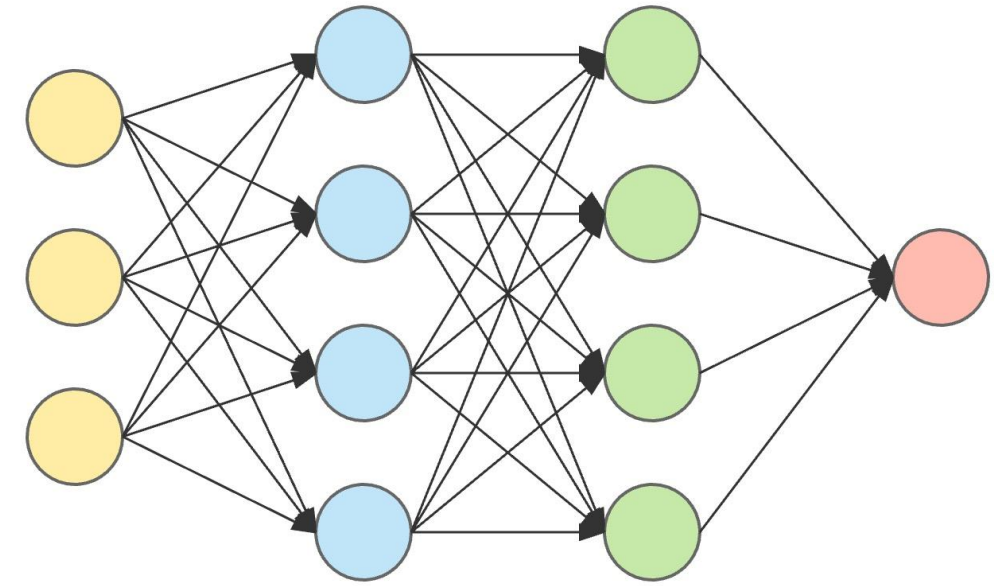
Different birds make different sounds and it is important to identify which bird is making which sound. Identifying the birds correctly from their sound can help the environmental experts to better understand birds behaviour and help them protect. Traditional methods of birds identification rely on human experts visually identifying birds birds in the field and this can be time consuming and prone to errors. The advance machine learning method like Neural Network automates this process. This study focuses on neural network models to classify bird species from their sounds. The dataset utilized is from the Xeno-Canto crowd-sourced bird sounds archive. In this study pre-processed data is used with 12 bird species named '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', 'White-crowned Sparrow '. Two important neural models are utilised for the classification. A binary model is used to differentiate between two bird species, 'Northern Flicker' (norfli) and 'Blue Jay' (blujay) and a multiclass model is used to classify all 12 birds present in the dataset. The analysis also includes external test data in the form of raw sound clips, which is pre-processed and evaluated using the multiclass model. Overall, this report is designed to contribute to the field of automated bird species classification by doing analysis of the neural network-based approaches.

Theoretical Background

Deep learning is a subfield of machine learning which involves training neural networks with multiple layers to learn hierarchical representations of data. It is capable of learning complex patterns and relationships within data. It is different from machine learning in by the type of data it works with and the methods in which it learns. It eliminates some of the data pre-

processing and process unstructured data like text and images. It has become very successful in different areas, one is recognizing objects in pictures and videos. It can identify things like animals, cars and other objects by just looking at the images. It also helps computers to understand human language, allow them to understand what people are sating or typing like Alexa and Siri.

A neural network is a powerful machine learning model and a way of teaching computers to process data in a way human brain does. They are very useful in finding patterns in data like images, sounds, or text. It has simple units called as neurons connected together in layers. The neurons receives the data from multiple sources , performs computation on it and pass it on to the next layer. A basic neural network has three layers : Input layer is the first layer in a neural network model. It receives input data that needs to be processed. The number of nodes in the input layer corresponds to the number of features in the data, hidden layer is the intermediate layer between the input and the output layers. This layer performs the actual computation on the data and output layer is the final layer of model. The number of output nodes depends on the problem, for eg. For binary problem one output node represents the probability of belonging of the species. For multi class classification problem, there would be one node for each class.



[6]

There are different types of nueral networks that are used for different types of data. Convolutional nueral network (CNNs) are great for handling data that looks like grid, such as images or sounds where Recurrent neural networks (RNNs) is designed to handle the data that comes in sequences, like text or speech. They have memory that helps them understand pattern across different steps in the sequence.

Neural networks are complex models that rely on various parameters and hyperparameters to function optimally. Some of the important parameters that require careful selection are layers, batch size, number of epochs, loss fucntion, activation function. The layers are building blocks of nueral network, **Flatten** layer converts multi-dimensional input into single dimensional array, in **Dense** layer every nueron is connected to every nueron in the previous

layer. **Batch size** is the number of training samples processed befor updating the model weights. **Epocs** is the number of times the entire training dataset is passed through the model Activation function define the output of a nueron given an input or set of input, **Relu** output the input directly if it is positive otherwise zero, **sigmoid** function outputs a value between 0 and 1 and is suitable for binary classificaiton problems, **softmax** function outputs a disctribution over multiple classes, and is useful for multiclass classification tasks. Loss function measures how well model predictions match the actual data. **Binary cross entropy** is used for binary classification tasks, it measures the performance of a classification model whose output is a probability value between 0 and 1. **Categorial cross entropy** is used for multiclass classification tasks, measure the performance when output is distributed over multiple classes.

Spectograms is a visual representstion of the spectrum of frequncies in a sound signal as they vary with time, It is a common technique used to analyze audio signals in the time-frequency domain.

Methodology

The analysis is done by using the strong machine learning model , neural networks. For this classification, binary and multiclass models are being utilised. The data used for this analysis is already preprocessed and is converted into spectrograms. The process starts by loading the preprocessed spectrogram data for two bird species - Northern Flicker and Blue Jay - from an HDF5 file into separate datasets. The code extracts the shape information to check the dimensions of the spectogram images for species.

Binary model

Data Preparation:

For binary classification model two bird species are being considered : “Northern Flicker ” and “Blue Jay”. The data is first transposed and rearranged. Labels are created for classes i.e 0 is for “Northern Flicker” and 1 is for “Blue Jay”. The data and labels are then combined. This data is now split into training and test sets using “train_test_split” function. The data is divided into 80-20% ratio with a random state value of 42.

Model Fitting :

A neural network model is created. The model consist of “Flatten” parameter to flatten the input data, two “Dense” layer with 128 and 64 neurons and it is using “Relu” activation function, one “Dense” layer with 1 neurons and “sigmoid” activation function. The model is compiled on “rmsprop” optimizer, binary cross-entropy loss function and accuracy is the considered metrics. The model is trained for different set of epocs and batch size to observe the change in accuracy and check which one provides the best. The code also plots the training and validation loss as well as accuracy over the epocs.

Multiclass model

Data Preparation :

For multiclass classification model all bird species are being considered named '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', 'White-crowned Sparrow '. The data is first transposed and rearranged. Initializing empty list for data and labels. For each species the code is extracting spectrogram data, transposing it and appending it to the data and labels. The data and labels are converted to numpy arrays. This data is now split into training and test sets using "train_test_split" function. The data is divided into 80-20% ratio with a random state value of 42. One hot encoding is performed as it is necessary for multi-class classification.

Model Fitting :

A neural network model is created. The model consist of "Flatten" parameter to flatten the input data, two "Dense" layer with 128 and 64 neurons and it is using "Relu" activation function, one "Dense" layer with number of nuerons equal to number of species and "softmax" activation function. The model is compiled on "rmsprop" optimizer, categorial cross-entropy loss function and accuracy is the considered metrics. The model is trained for different set of epocs and batch size to observe the change in accuracy and check which one provides the best. The code also plots the training and validation loss as well as accuracy over the epocs.

External Test Data

Data Preparation :

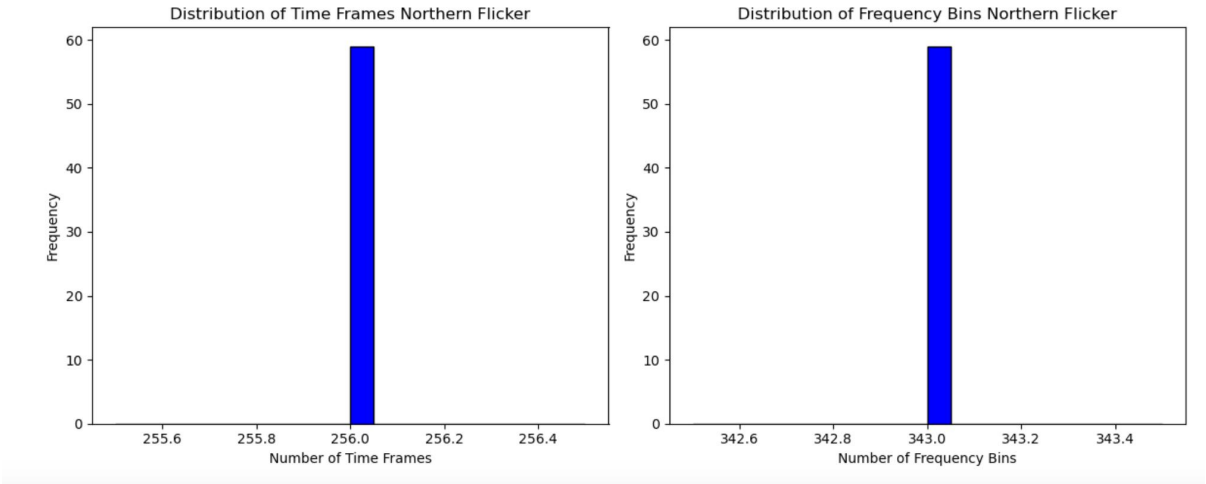
First data is prepared by creating spectograms from the audio files. A spectrogram is like a picture that shows how the sounds in an audio file vary over time. Once spectrograms are ready, they are flatten which helps them turn into a format that can be easily used by neural network model. Next, these flattened spectrograms are reshaped back into their original size but with an additional dimension, which is needed for the neural network. This reshaped data now looks like 256 by 343 pixel images.

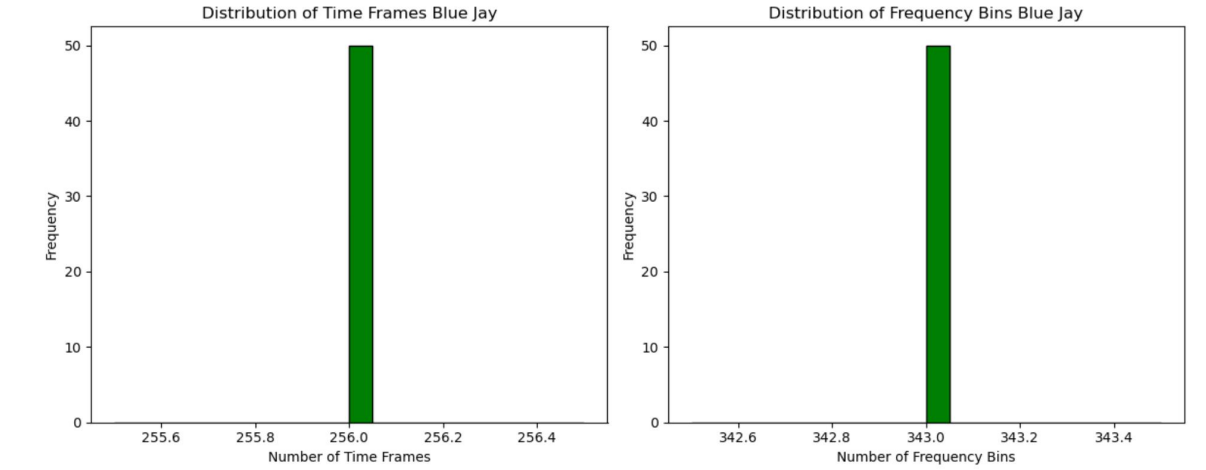
Model Fitting :

In this stage, a pre-trained nueral network model is loaded, which is already trained to recognize different bird species form spectogram images. This model is used to make predictions on the prepared spectrograms. The names of bird species are loaded frim a file and the top 5 species with the higest scores for each spectogram are identified. These top predictors are matched with species names, and this information shows which species are most likely for each audio file. The top 5 predictors for each audio file are printed out along with their scores.

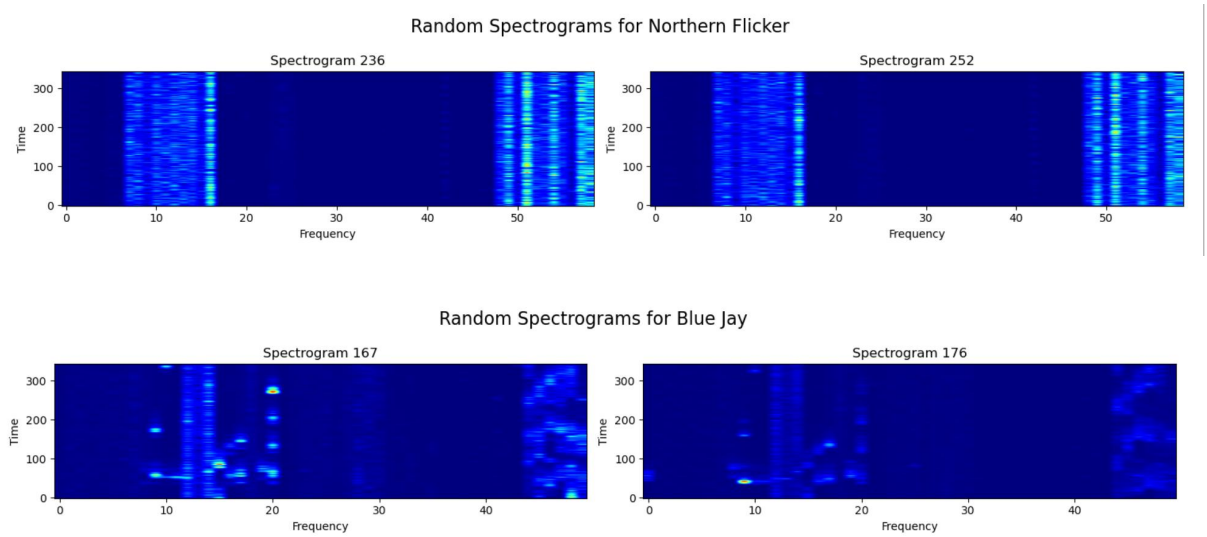
Computational Results

Data visualisation :



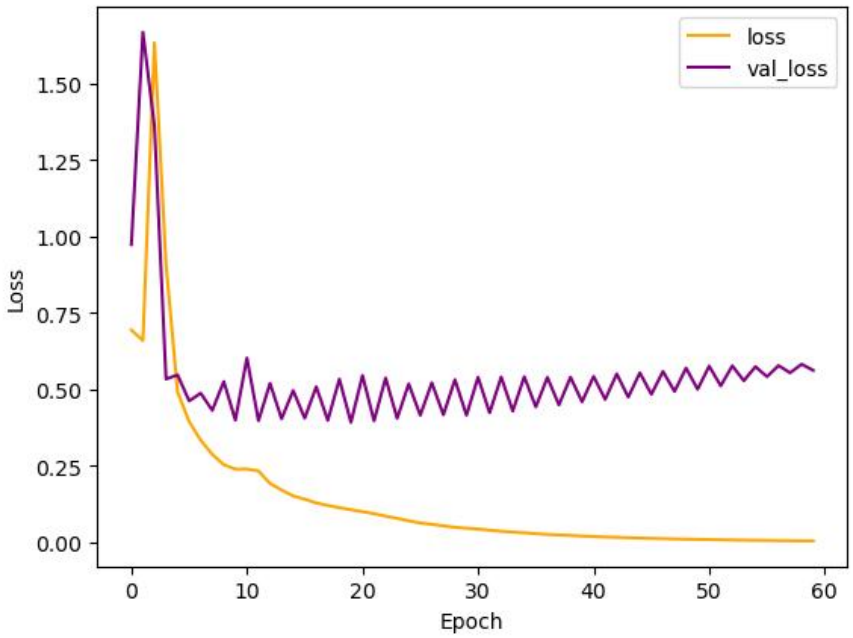


Spectrogram :

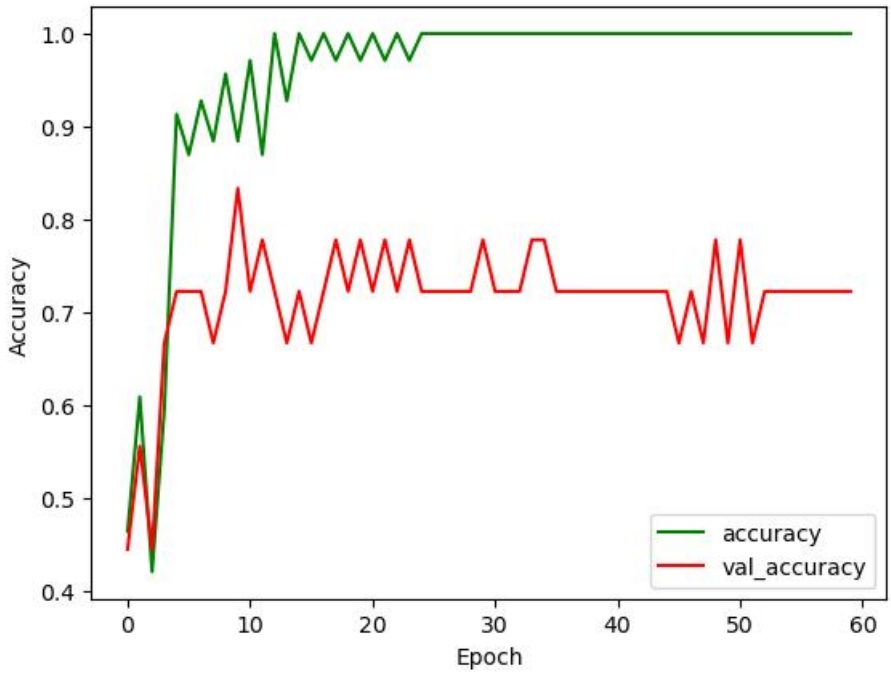


Graphs :

Binary Model :

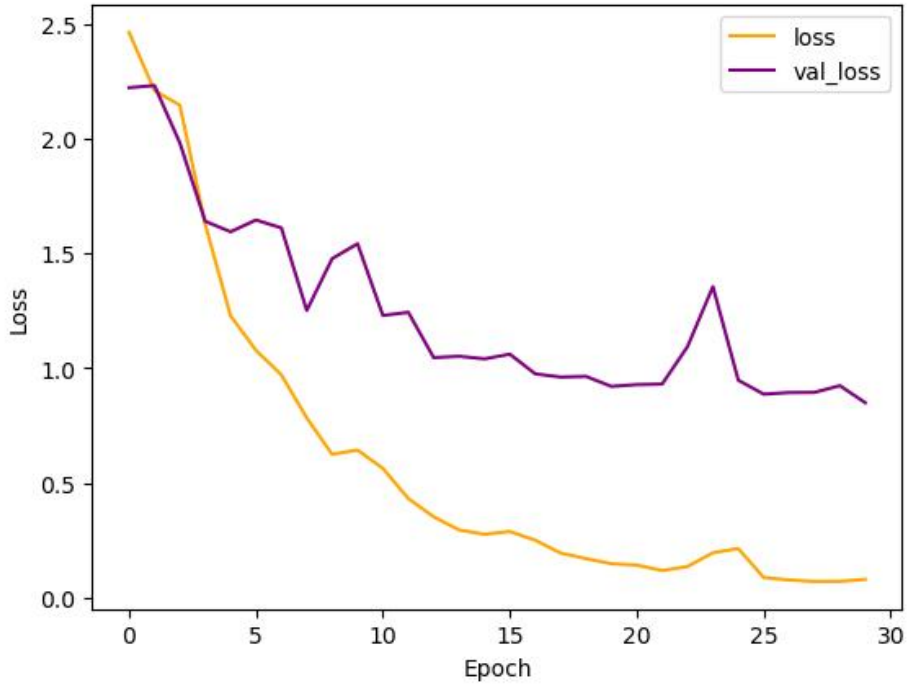


In the above line graph the x-axis is representing “Epoch” and y-axis is “Loss”. It represents loss visualization during the training of binary model. From the above figure it can be depicted that the loss is decreasing with the increase in number of epocs. The means that model is overfitting as it might perform well on unseen data.

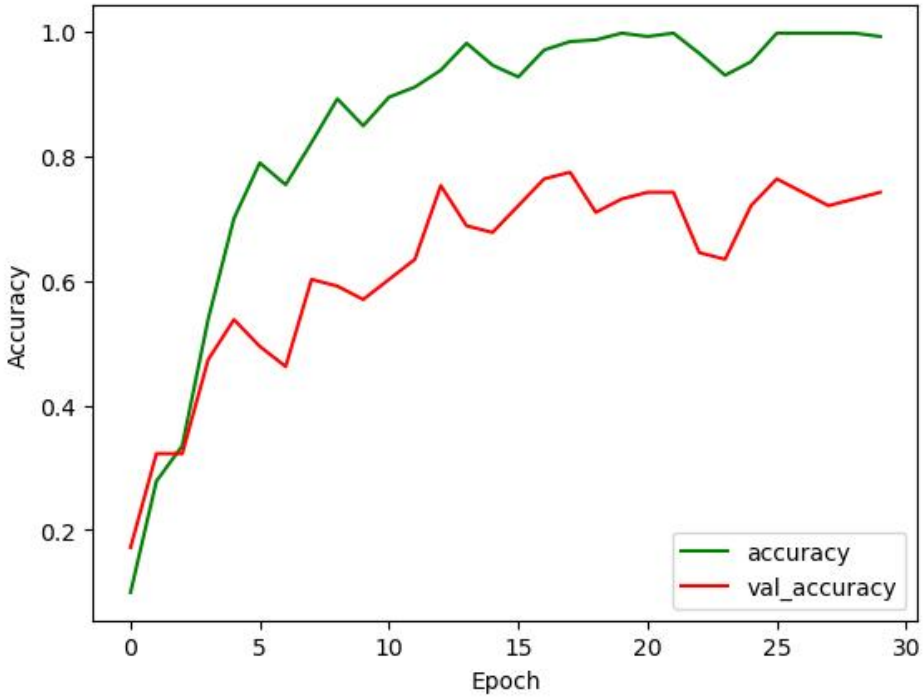


In the above line graph the x-axis is representing “Epoch” and y-axis is “accuracy”. It represents accuracy visualization during the training of binary model. From the above figure it can be depicted that the accuracy increases with the increase in number of epocs. After 20 itcan be seen that there is not increase or decrease in accuracy/ The val_accuracy is also increasing but with a slower rate.

Multiclass Model :



The above line graph represents “epoch” on the x-axis and “Loss” on the y-axis. The yellow line represents the loss on the data for each class, it can be seen that it decreases with the increase in number of epocs.



The above line graph represents “epoch” on the x-axis and “Accuracy” on the y-axis. The green line represents the accuracy on the data for each class, it can be seen that it increases

with increase in number of epocs. The val_accuracy is also increasing but at a slower rate. The model tends to overfit.

Results :

For Binary Classification Model:

Epocs	Batch size	Test Accuracy	CPU Time
5	80	81.82%	43 s
15	128	86.36%	60 s
30	256	81.82%	65 s
60	128	86.36%	157 s
50	384	86.36%	143 s

For MultiClass Classification Model:

Epocs	Batch size	Test Accuracy	CPU Time
30	250	64.66%	320 s
10	128	66.38%	183 s
20	256	65.52%	224 s
20	128	63.79%	256 s
40	256	64.66%	366 s

Top 5 Predictions for Test File 1:

Species	Prediction Score
amecro	1.00
whcspa	0.00
wesmea	0.00
stejay	0.00
rewbla	0.00

Top 5 Predictions for Test File 2:

Species	Prediction Score
amecro	1.00
whcspa	0.00
wesmea	0.00
stejay	0.00
rewbla	0.00

Top 5 Predictions for Test File 3:

Species	Prediction Score
amecro	1.00
whcspa	0.00
wesmea	0.00
stejay	0.00
rewbla	0.00

Discussion

During the analysis, for binary model the highest achieved accuracy is 86.36%. This high accuracy indicates that the given dataset is well - prepared. It has been achieved by 3 model, Model 1 achived it with 15 epocs and 128 batch size, Model 2 achieved it by setting epocs to 60 and batch size to 128 while Model 3 achieved it by keeping epocs to 50 and batch size to 384. On comparing the computation time for all the three models, Model 1 took relatively lesser time in comparison to other two models which is 60 s, while model 2 and model 3 took 157 s and 143 s respectively. These varied training times shows the excution and learning efficiency of model. Even the accuracy is higher , the model tends to overfit the training data.

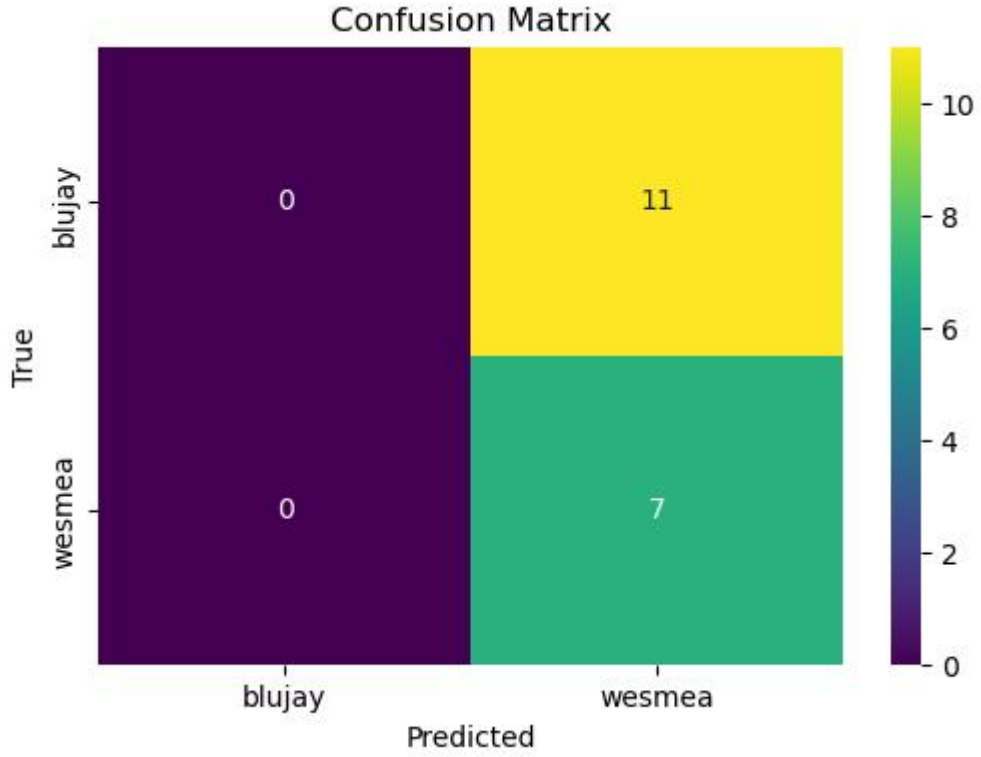
For multi-class model, the highest achieved accuracy is 66.38%. It has been achieved by the model with 10 epocs, 128 batch size and CPU time is 183 s. The seocnd best accuracy is acieved by model 2 with 20 eopocs, 256 batch size and CPU time is 224 s. It can be observed from the reuslts that increase in number of epocs has slightly impacted the accuracy of the models. Also the lower accuracy in multiclass classification in comparison to binary shows that the complexity and difficulty has increased while differentiating multiple bird species.

When analyzing the external test data, the neural network model provided predictions for multiple bird species, demonstrating its capability to handle real-world data. For Test File 1, the model predicted the American Crow (amecro) with absolute certainty (1.00), while assigning zero probabilities to other species like the White-crowned Sparrow (whcspa),

Western Meadowlark (wesmea), Steller's Jay (stejay), and Red-Winged Blackbird (rewbla). This suggests that the model confidently recognized the American Crow's distinctive audio characteristics. Similarly, in Test File 2, the model again predicted the American Crow (amecro) with absolute certainty, while other species were given zero probabilities. This pattern was repeated for Test File 3, indicating that the model consistently recognized the American Crow across different test files.

Some limitations/challenges were encountered during the analysis, for the external data analysis model, it can be observed that it has been identifying single bird with absolute certainity which sets its limitation in recognizing and differentiating between other species. The zero probabilities for other species suggest that the model might need further refinement to improve its efficiency and accuracy in identifying a broader range of bird species. While binary and multiclass gave decent accuracy but these can still be improved to achive best accuracies. In the analysis blue jay and western meadowlark were the most challenging species to identify. The confusion matrix also suggests that the model incorrectly predicted 11 western meadowlark as blue jays. We could have considered SVMs for this analysis as they are effective for high-dimensional spaces, but they might struggle with large and complex datasets typical in audio analysis. While nueral networks are highly suitable for such application because they are good in handling high-dimensional data.

Confusion Matrix :



Conclusion

The report aimed to understand how a neural network can be used to identify different types of birds just from their voice/sound. The study utilized dataset from the Xeno-Canto bird sounds archive. Two different types of models are implemented for this. Binary model which is used to identify two specific birds, Northern Flicker and Blue Jay and multiclass that is used to identify for all 12 bird types. The binary classification model achieved accuracy of 86.36% while the multiclass model achieved an accuracy of 66.38%. These models were trained with different epochs and batch sizes, and their performance was evaluated in terms of accuracy and computational time. When the multi-class model is tested on 3 new audio files of unknown bird sounds, it consistently identified the sounds as coming from the American Crow species with high confidence. However, it struggled to recognize any other species present in those recordings. This suggest that model needs further refinement to accurately identify species. The results revealed that while the models performed well overall, there were limitations and challenges encountered. The binary model showed signs of overfitting, indicating the need for further refinement. Additionally, the model struggled to differentiate between similar species, such as Blue Jay and Western Meadowlark. Overall, this project showed that machine learning techniques like neural networks can be used to automatically figure out what bird species a sound recording is from. This could help people monitor bird populations and protect them better.

Bibliography/References

1. https://www.kaggle.com/datasets/imoore/xenocanto-bird-recordings-dataset

2. What is Deep Learning? by AWS
https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain.

3. 1.17. Neural network models (supervised) by scikit learn
https://scikit-learn.org/stable/modules/neural_networks_supervised.html

4 . What is deep learning? by IBM
https://www.ibm.com/topics/deep-learning#:~:text=Deep%20learning%20is%20a%20subset,AI)%20in%20our%20lives%20today.

5 Introduction to Deep Learning by geeksforgeeks
https://www.geeksforgeeks.org/introduction-deep-learning/

6 Applied Deep Learning - Part 1: Artificial Neural Networks by Arden Dertat
https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Appendix

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from matplotlib import image
from sklearn.model_selection import train_test_split
import os

from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.preprocessing import OneHotEncoder, StandardScaler,
LabelBinarizer, LabelEncoder
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score

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, LSTM, SimpleRNN
from keras.applications import imagenet_utils, ResNet50
from keras.applications.imagenet_utils import preprocess_input
from keras.datasets import imdb

from scipy.sparse import csr_matrix
import h5py
import numpy as np
f = h5py.File('spectrograms.h5', 'r')
list(f.keys())
norfli = f['norfli']
blujay = f['blujay']
norfli_shapes = norfli.shape
blujay_shapes = blujay.shape
print(f"Shape of Northern Flicker: {norfli_shapes}")
print(f"Shape of Blue Jay: {blujay_shapes}")
norfli_shapes = np.array([norfli[:, :, i].shape for i in range(norfli.shape[2])])
print("Extracted norfli spectrogram shapes:", norfli_shapes.shape)

blujay_shapes = np.array([blujay[:, :, i].shape for i in range(blujay.shape[2])])
print("Extracted blujay spectrogram shapes:", blujay_shapes.shape)
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
```

```
axes[0].hist(norfli_shapes[:, 0], bins=20, color='blue', edgecolor='black')
axes[0].set_title('Distribution of Time Frames Northern Flicker')
axes[0].set_xlabel('Number of Time Frames')
axes[0].set_ylabel('Frequency')

axes[1].hist(norfli_shapes[:, 1], bins=20, color='blue', edgecolor='black')
axes[1].set_title('Distribution of Frequency Bins Northern Flicker')
axes[1].set_xlabel('Number of Frequency Bins')
axes[1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

axes[0].hist(blujay_shapes[:, 0], bins=20, color='green', edgecolor='black')
axes[0].set_title('Distribution of Time Frames Blue Jay')
axes[0].set_xlabel('Number of Time Frames')
axes[0].set_ylabel('Frequency')

axes[1].hist(blujay_shapes[:, 1], bins=20, color='green', edgecolor='black')
axes[1].set_title('Distribution of Frequency Bins Blue Jay')
axes[1].set_xlabel('Number of Frequency Bins')
axes[1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
num_spectrograms_to_visualize = 2

fig, axes = plt.subplots(1, num_spectrograms_to_visualize, figsize=(15, 3))
fig.suptitle("Random Spectrograms for Northern Flicker", fontsize=16)

for i in range(num_spectrograms_to_visualize):
    random_index = np.random.randint(0, len(norfli))
    axes[i].imshow(norfli[random_index], cmap='jet', origin='lower')
    axes[i].set_title(f"Spectrogram {random_index+1}")
    axes[i].set_xlabel('Frequency')
    axes[i].set_ylabel('Time')
    axes[i].set_aspect('auto')

plt.tight_layout()
plt.show()
num_spectrograms_to_visualize = 2
```

```
fig, axes = plt.subplots(1, num_spectrograms_to_visualize, figsize=(15, 3))
fig.suptitle("Random Spectrograms for Blue Jay", fontsize=16)

for i in range(num_spectrograms_to_visualize):
    random_index = np.random.randint(0, len(blujay))
    axes[i].imshow(blujay[random_index], cmap='jet', origin='lower')
    axes[i].set_title(f"Spectrogram {random_index+1}")
    axes[i].set_xlabel('Frequency')
    axes[i].set_ylabel('Time')
    axes[i].set_aspect('auto')

plt.tight_layout()
plt.show()
norfli_data = f['norfli'][:].transpose((2, 0, 1))
blujay_data = f['blujay'][:].transpose((2, 0, 1))

labels_norfli = np.zeros(norfli_data.shape[0])
labels_blujay = np.ones(blujay_data.shape[0])

data = np.concatenate((norfli_data, blujay_data), axis=0)
labels = np.concatenate((labels_norfli, labels_blujay), axis=0)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=42)

Binary Model
%%time
binary_model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

history = binary_model.fit(X_train, y_train
                    , epochs=5, batch_size=80
                    , validation_split=0.2, verbose=0)
```

```
score = binary_model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(score[1] * 100))
%%time
binary_model2 = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

binary_model2.compile(optimizer='rmsprop'
              , loss='binary_crossentropy'
              , metrics=['accuracy'])

history = binary_model2.fit(X_train, y_train
                    , epochs=15, batch_size=128
                    , validation_split=0.2, verbose=0)

score = binary_model2.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(score[1] * 100))
%%time
model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

history = model.fit(X_train, y_train
                    , epochs=30, 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))
%%time
model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
```

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

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

Multiclass

```
import h5py
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical

f = h5py.File('spectrograms.h5', 'r')
species = list(f.keys())
data = []
labels = []
for species_name in species:
    spectrograms = f[species_name][:].transpose((2, 0, 1))
    data.append(spectrograms)
```

```
    labels.extend([species.index(species_name)] * len(spectrograms))

data = np.concatenate(data, axis=0)
labels = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=42)

num_classes = len(species)

y_train_onehot = to_categorical(y_train, num_classes)
y_test_onehot = to_categorical(y_test, num_classes)
%%time
model_multiclass = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])

model_multiclass.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model_multiclass.fit(X_train, y_train_onehot,
                  epochs=30, batch_size=250,
                  validation_split=0.2, verbose=0)

score = model_multiclass.evaluate(X_test, y_test_onehot, verbose=0)
print("Test Accuracy: {:.2f}%".format(score[1] * 100))
%%time
model_multiclass.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history1 = model_multiclass.fit(X_train, y_train_onehot,
                  epochs=10, batch_size=128,
                  validation_split=0.2, verbose=0)

score = model_multiclass.evaluate(X_test, y_test_onehot, verbose=0)
print("Test Accuracy: {:.2f}%".format(score[1] * 100))
%%time
model_multiclass.compile(optimizer='rmsprop',
```

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

history1 = model_multiclass.fit(X_train, y_train_onehot,
                epochs=60, batch_size=256,
                validation_split=0.2, verbose=0)

score = model_multiclass.evaluate(X_test, y_test_onehot, verbose=0)
print("Test Accuracy: {:.2f}%".format(score[1] * 100))
model_multiclass.save('multiclass_model.h5')
plt.plot(history.history['loss'], color='orange')
plt.plot(history.history['val_loss'], color='purple')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['loss', 'val_loss'], loc='upper right')
plt.show();

plt.plot(history.history['accuracy'], color='green')
plt.plot(history.history['val_accuracy'], color='red')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['accuracy', 'val_accuracy'], loc='lower right')
plt.show();
part 3
pip install librosa
import os
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
from tensorflow.keras.models import load_model
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
def create_spectrograms(audio_paths):
    spectrogram_list = []
    spectrogram_indices = []

    for i, audio_path in enumerate(audio_paths):
        try:
```

```
            print(f"Processing audio file: {audio_path}")
            data, rate = librosa.load(audio_path, sr=22050)
            loud_parts = librosa.effects.split(data, top_db=20)

            if len(loud_parts) == 0:
                print(f"No loud parts found in {audio_path}")
            else:
                for start, end in loud_parts:
                    duration = (end - start) / rate
                    print(f"Loud segment from {start} to {end}, duration:
{duration:.2f}s")

                    if duration > 0.5:
                        print(f"Processing bird call from {start} to {end}")

                        window = data[start:end]

                        spec = librosa.feature.melspectrogram(y=window, sr=rate,
n_fft=1024, hop_length=512, n_mels=256)

                        if spec.shape[1] < 343:
                            padding = np.zeros((256, 343 - spec.shape[1]))
                            spec = np.hstack((spec, padding))
                        elif spec.shape[1] > 343:
                            spec = spec[:, :343]

                        spec_db = librosa.power_to_db(spec, ref=np.max)
                        spectrogram_list.append(spec_db)
                        spectrogram_indices.append(i)

        except Exception as e:
            print(f"Error processing {audio_path}: {e}")

    return spectrogram_list, spectrogram_indices
def flatten_spectrograms(spectrogram_list):
    flattened_spectrograms = []
    target_size = 256 * 343

    for spec in spectrogram_list:
        flattened_spec = spec.flatten()
        if len(flattened_spec) < target_size:
            padding = np.zeros(target_size - len(flattened_spec))
            flattened_spec = np.concatenate((flattened_spec, padding))
```

```
        elif len(flattened_spec) > target_size:
            flattened_spec = flattened_spec[:target_size]
        flattened_spectrograms.append(flattened_spec)

    flattened_spectrograms = np.array(flattened_spectrograms)
    return flattened_spectrograms

audio_paths = [
    '/Users/nidhitrivedi/Downloads/test_birds/test1.mp3',
    '/Users/nidhitrivedi/Downloads/test_birds/test2.mp3',
    '/Users/nidhitrivedi/Downloads/test_birds/test3.mp3'
]
spectrograms, spectrogram_indices = create_spectrograms(audio_paths)
flattened_spectrograms = flatten_spectrograms(spectrograms)
reshaped_spectrograms =
flattened_spectrograms.reshape(flattened_spectrograms.shape[0], 256, 343, 1)

model = load_model('multiclass_model.h5')

predictions = model.predict(reshaped_spectrograms)

file_path = r"/Users/nidhitrivedi/Downloads/spectrograms.h5"
with h5py.File(file_path, 'r') as f:
    species_names = list(f.keys())

top_n_predictions = np.argsort(predictions, axis=1)[:, -5:]
top_n_scores = np.sort(predictions, axis=1)[:, -5:]

predicted_species = [[] for _ in range(len(audio_paths))]
for i, idx in enumerate(spectrogram_indices):
    prediction_row = top_n_predictions[i]
    scores_row = top_n_scores[i]
    species_list = [(species_names[pred], scores_row[j]) for j, pred in
enumerate(prediction_row)]
    predicted_species[idx].append(species_list)

for i, species_lists in enumerate(predicted_species):
    print(f"Top 5 Predictions for Test File {i+1}:")
    for species in species_lists:
        for bird, value in reversed(species):
            print(f"{bird:<10} {value:.10f}")
    print("\n")
f = h5py.File('spectrograms.h5', 'r')
```

```
spectro_species = []
labels = []

for species in ['wesmea', 'blujay']:
    spectrograms = f[species][:].transpose((2, 0, 1))
    for spectrogram in spectrograms:
        spectro_species.append(spectrogram)
        labels.append(species)

spectro_species = np.array(spectro_species)

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

max_length = max([s.shape[1] for s in spectro_species])
spectrograms_padded = []

for s in spectro_species:
    pad_width = max_length - s.shape[1]
    s_padded = np.pad(s, pad_width=((0, 0), (0, pad_width)), mode='constant')
    spectrograms_padded.append(s_padded)

spectro_species = np.array(spectrograms_padded)
labels = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(spectro_species, labels,
test_size=0.2, random_state=42)

X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]
binary_model = Sequential([
    Flatten(input_shape=(256, 343)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

y_pred = (binary_model.predict(X_test) > 0.5).astype("int32")

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['blujay',
'wesmea'], yticklabels=['blujay', 'wesmea'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```