## 1. ABSTRACT

This report examines the application of neural networks in the classification of bird species based on their sounds. A dataset comprising 10 audio clips for each of the 12 selected bird sounds from the Seattle area, sourced from Xeno-Canto, is employed to train the neural network. The primary objective is to predict the species of birds based on their distinct sounds. The binary classification task involves distinguishing between the amecro and norfli birds using their sounds. Additionally, a multi-class classification task is performed to classify all 12 bird species, namely amecro, barswa, bkcchi, blujay, daejun, houfin, mallar3, norfli, rewbla, stejay, wesmea, and whcspa. The report also addresses encountered limitations, compares the performance of various network structures and hyperparameters, and highlights the potential of neural networks for bird species identification.

## 2. INTRODUCTION

Identification of bird species poses a significant challenge, demanding extensive expertise and knowledge. Recent advancements in machine learning techniques have facilitated the classification of bird species based on their distinct sounds. In this study, a neural network is employed to categorize bird species using audio clips of their sounds. The objective is to develop a robust neural network capable of accurately predicting species based on their unique sound patterns. Both binary and multi-class classification tasks are performed on the dataset, and the encountered limitations are thoroughly discussed. Additionally, a comparative analysis of diverse network structures and hyperparameters is conducted to identify the most suitable model for this application. The findings of this study shed light on the potential of neural networks in bird species identification and their potential implications for conservation efforts.

### 2.1. Dataset Description

The provided dataset comprises original sound clips recorded in the Seattle area, obtained from Xeno-Canto, a crowd-sourced bird sounds archive. For each of the 12 bird species, 10 high-quality sound clips with durations under 60 seconds were selected. These sound clips can be used to generate spectrograms, which serve as input for training our model.

### 2.2 Problem Types

Two distinct problem types are addressed in this report: binary classification and multi-class classification. In the binary classification task, we focused on two specific bird species, amecro and norfli. A neural network was constructed to effectively differentiate between these species, enabling us to concentrate on the unique characteristics that set them apart and develop an accurate classification model.

Conversely, in the multi-class classification task, the objective was to classify all 12 bird species present in the dataset. This presented a more intricate challenge, as it required distinguishing between numerous species with diverse characteristics. Consequently, a more robust neural network architecture and training process were employed, while also considering factors such as class imbalance and overfitting.

By addressing these two problem types, we gained valuable insights into various aspects of bird species classification, enabling us to evaluate the performance of different neural network architectures and hyperparameters.

# 3. THEORETICAL BACKGROUND

## 3.1 Neural Networks:

Neural networks are a class of machine learning algorithms that draw inspiration from the structure and functioning of the human brain. They consist of interconnected nodes, known as neurons, organized into multiple layers. These networks process information and make predictions based on the patterns they learn from the data. In the context of bird species identification, neural networks can be trained to recognize and learn the unique patterns and features associated with each species based on their distinctive sounds.

### 3.1.1 Types of Neural Networks:

Neural networks encompass various types, each with its own architecture and characteristics.

**Feedforward Neural Networks (FNNs):**

Feedforward Neural Networks, also known as Multilayer Perceptron (MLPs), are the foundational type of neural network. They consist of layers of interconnected neurons, with information flowing strictly in one direction, from input to output. FNNs are widely used for tasks such as regression, classification, and pattern recognition.

**Convolutional Neural Networks (CNNs):**

Convolutional Neural Networks (CNNs) are specifically designed for processing grid-like data, such as images or spectrograms. They are widely used in computer vision tasks, including image classification, object detection, and image segmentation. CNNs leverage specialized layers such as convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to extract local patterns and features from the input data, while pooling layers down sample the spatial dimensions. CNNs have demonstrated remarkable performance in analyzing and classifying complex visual data, making them applicable to bird species identification based on their unique sounds.

**Recurrent Neural Networks (RNNs):**

Recurrent Neural Networks are designed to handle sequential data, where the order and temporal dependencies of the data points are important. RNNs leverage recurrent connections to retain memory of past inputs, enabling them to process sequences of varying lengths. RNNs are commonly used in tasks such as natural language processing, speech recognition, and time series analysis.

**Long Short-Term Memory (LSTM) Networks:**

LSTM Networks are a specialized type of RNN that effectively address the vanishing gradient problem associated with traditional RNNs. LSTMs incorporate memory cells and gating mechanisms to selectively retain and update information over long sequences. They excel in capturing long-term dependencies and have been successfully applied in speech recognition, language modeling, and other tasks involving sequential data.

### 3.1.2 Compilation Techniques:

In the context of neural networks, compilation refers to the configuration and optimization of the model before training. Several compilation techniques are employed to enhance the efficiency and performance of the neural network.

One crucial aspect of compilation is the choice of the loss function. The loss function quantifies the discrepancy between the predicted outputs of the network and the true labels. The selection of an appropriate loss function depends on the specific task at hand. For example, binary classification tasks often use binary cross-entropy loss, while multi-class classification tasks commonly employ categorical cross-entropy loss.

Another compilation technique is the choice of the optimizer, which determines the algorithm used to update the network's weights during training. Optimizers play a vital role in guiding the learning process and finding the optimal set of weights that minimize the loss function.

Popular optimizers include stochastic gradient descent (SGD), Adam, and RMSprop, each with its own characteristics and performance trade-offs.

Furthermore, compilation involves specifying the evaluation metrics used to assess the performance of the trained model. These metrics can include accuracy, precision, recall, F1-score, and others, depending on the nature of the classification problem. They provide valuable insights into the model's ability to correctly classify bird species based on their sounds.

Additionally, compilation techniques may also involve adjusting other hyperparameters of the neural network, such as learning rate, batch size, and regularization techniques (e.g., dropout or L2 regularization). These hyperparameters significantly impact the training process and the generalization ability of the model.

By carefully selecting and fine-tuning the compilation techniques, the neural network can be optimized to achieve better accuracy, convergence speed, and overall performance in the task of bird species identification based on their sounds

## 3.2 Activation function:

Activation functions play a crucial role in neural networks by introducing non-linearity to the outputs of individual neurons. They enhance the network's ability to learn complex patterns and make accurate predictions. In this project, two commonly used activation functions are employed: the sigmoid function and the SoftMax function. The sigmoid function maps its input to a value between 0 and 1, which is useful for binary classification problems. The SoftMax function, on the other hand, transforms the inputs into a probability distribution across a set of output classes, making it well-suited for multi-class classification problems.

## 3.3 Transfer Learning:

Transfer learning is a machine learning technique that leverages pre-trained models as a starting point for new tasks. It involves utilizing the knowledge and representations learned from solving one problem and applying them to a related problem, reducing the amount of data and computation required for training. Transfer learning has proven to be effective in scenarios where there is limited data available for a specific task. By utilizing pre-existing models trained on large datasets, the network can benefit from the general features and patterns learned from those datasets and adapt them to the target task, such as bird species identification.

## 3.4 Spectrograms:

Spectrograms are visual representations of the frequency content of a signal as it changes over time. They are widely used in the analysis of audio signals and provide valuable insights into the frequency components present in a sound. Spectrograms are particularly useful for identifying patterns in sound signals, as they allow us to visualize the distribution of frequencies over time. In machine learning applications, spectrograms are commonly employed as input features for models performing tasks such as speech recognition, sound classification, and music analysis. By converting audio signals into spectrograms, we can extract relevant features that capture the unique characteristics of different types of sounds, facilitating accurate classification and identification of bird species.

# 4. METHODOLOGY

## 4.1. Data Preprocessing

The methodology for the preprocessing of bird audio files involves several steps. First, the folder path where the bird audio files are stored is defined. Then, the audio files are resampled to a new sample rate of 22050. Next, the minimum length of a "loud" part of the sound clip is defined as 0.5 seconds, and the window size for selecting bird calls is set to 2 seconds. Spectrogram parameters are also defined, including an n_fft of 2048 and a hop_length of 512. The audios and labels lists are initialized for storing the spectrograms and bird names. The script loops through each bird folder and, it loads the audio file, computes the energy of the audio signal, finds the indexes of the "loud" parts of the sound clip, and extracts the audio signal within the bird call window. Then, it computes the spectrogram of the bird call audio and adds the spectrogram to the list of audios and the bird’s name to the list of labels. The spectrograms are padded with zeros to ensure they all have the same shape, and the padded audios are converted to a numpy array. The labels are prepared by creating a label map, and the data is split into training and testing sets with a test size of 0.2 and a random state of 42. I considered only amecro and norfli data for binary classification and for multi-class classification, I considered data of all birds.

## 4.2 Binary Classification:

In the Binary Classification task, two bird species, amecro and norfli, are being classified. Two different models were employed for this binary image classification task. Model 1 consists of a convolutional neural network (CNN) architecture comprising two convolutional layers with max pooling, a flatten layer, two fully connected layers, and a sigmoid activation function for binary classification. On the other hand, Model 2 is an extension of Model 1, incorporating a dropout layer between the flatten and fully connected layers. Both models were compiled using the Adam optimizer, binary cross-entropy loss, and accuracy metric.

### 4.3 Multi-Class Classification:

The Multi-class classification problem focuses on the classification of bird species from audio data. To prepare the labels, the code maps each bird species to a numerical label and applies one-hot encoding to represent the labels. The data is then split into training and testing sets using the train_test_split function from scikit-learn. Two models were utilized for this classification task.

The first model is a sequential CNN model, consisting of two convolutional layers, two max-pooling layers, one flatten layer, and two dense layers. The input shape for the CNN is defined as (128, 87, 1), where 128 and 87 correspond to the dimensions of the input spectrogram, and 1 represents the number of channels. ReLU activation function is applied to the convolutional layers, while the output layer employs a softmax activation function. The model is compiled using categorical cross-entropy loss and the Adam optimizer. The accuracy metric is used to evaluate the model's performance. In the second model, a dropout layer is added to the architecture of the first model.

### 4.4 Transfer learning

The methodology involves loading the pre-trained VGG16 model without the top layer, freezing its layers, and adding a new top layer for the audio classification task. The model is then compiled using categorical cross-entropy loss and the Adam optimizer. The data is resized to add a channel dimension, and the model is trained for 10 epochs using a batch size of 512. The model is evaluated on the test set using the evaluate function, and the test loss and accuracy are printed. Overall, the methodology involves using transfer learning to adapt a pre-trained image classification model to an audio classification task, resizing the data, and training the model using the generator.

## 5. COMPUTATIONAL RESULTS

### 5.1 Binary Classification:

| | Train Accuracy | Test Accuracy | Training Loss | Testing Loss |
|---|---|---|---|---|

|  |  |  |  |  |
|---|---|---|---|---|
| Model 1 | 99.57% | 99.16% | 1.76% | 1.78% |
| Model 2(with dropout) | 99.36% | 99.16% | 1.65% | 1.75% |

The given table represents the performance metrics of two binary classification models - Model 1 and Model 2 (with dropout). Both models have been evaluated based on their training and testing accuracy and loss. Model 1 has a higher training accuracy of 99.57% compared to Model 2's training accuracy of 99.36%. However, both models have a similar testing accuracy of 99.16%, which suggests that the models are performing well on unseen data. In terms of loss, both models have a low training loss, with Model 2 (with dropout) having a slightly lower training loss of 1.65% compared to Model 1's training loss of 1.76%. Similarly, both models have a similar testing loss of 1.78% for Model 1 and 1.75% for Model 2 (with dropout). Overall, both models have performed well with similar testing accuracy and testing loss. However, Model 2's use of dropout regularization may help prevent overfitting and improve the model's generalization performance on unseen data.

**5.2 Multi-Class Classification:**

|  | Train Accuracy | Test Accuracy | Training Loss | Testing Loss |
|---|---|---|---|---|
| Model 1 | 90.95% | 88.23% | 31.15% | 48.59% |
| Model 2(with dropout) | 92.22% | 89.22% | 26.28% | 35.89% |

The given table compares the performance of two models in terms of their train and test accuracy and losses.

Model 1 achieves a train accuracy of 90.95% and a test accuracy of 88.23%. The training loss is 31.15% while the testing loss is 48.59%. This suggests that the model is overfitting on the training data, as it performs much better on the training data than the test data. The high training loss and testing loss further indicate that the model is not generalizing well to unseen data.

Model 2, which includes a dropout layer, achieves a higher train accuracy of 92.22% and a slightly higher test accuracy of 89.22%. The training loss is lower at 26.28% and the testing loss is also significantly lower at 35.89%. This suggests that the model is able to generalize better to unseen data, as indicated by the lower testing loss. The addition of the dropout layer has helped to reduce overfitting and improve the model's generalization ability.

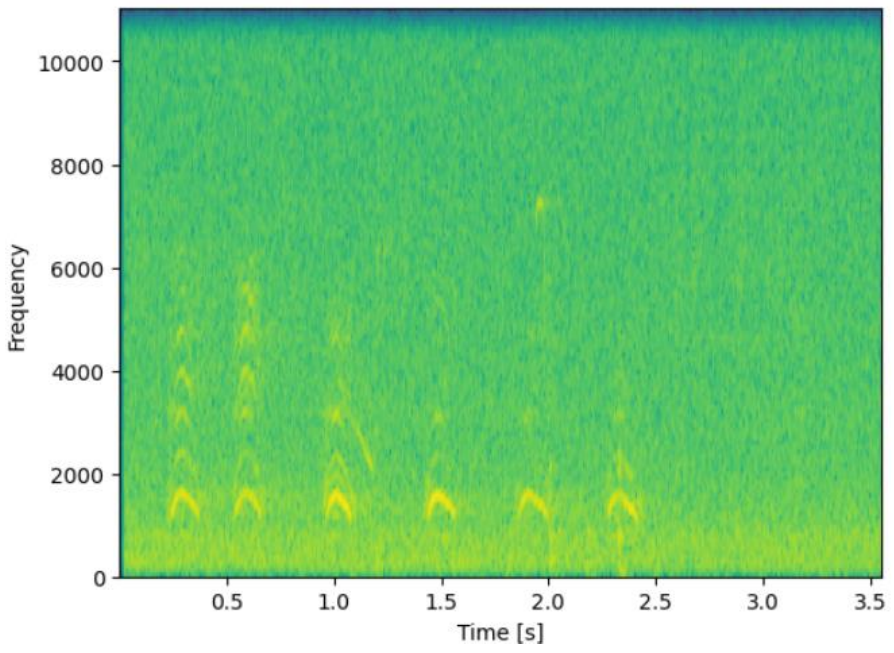
**5.3 Transfer Learning:**

| | Train Accuracy | Test Accuracy | Train Loss | Test loss |
|---|---|---|---|---|
| VGG16 | 94.27% | 92.94% | 26.96% | 32.54% |

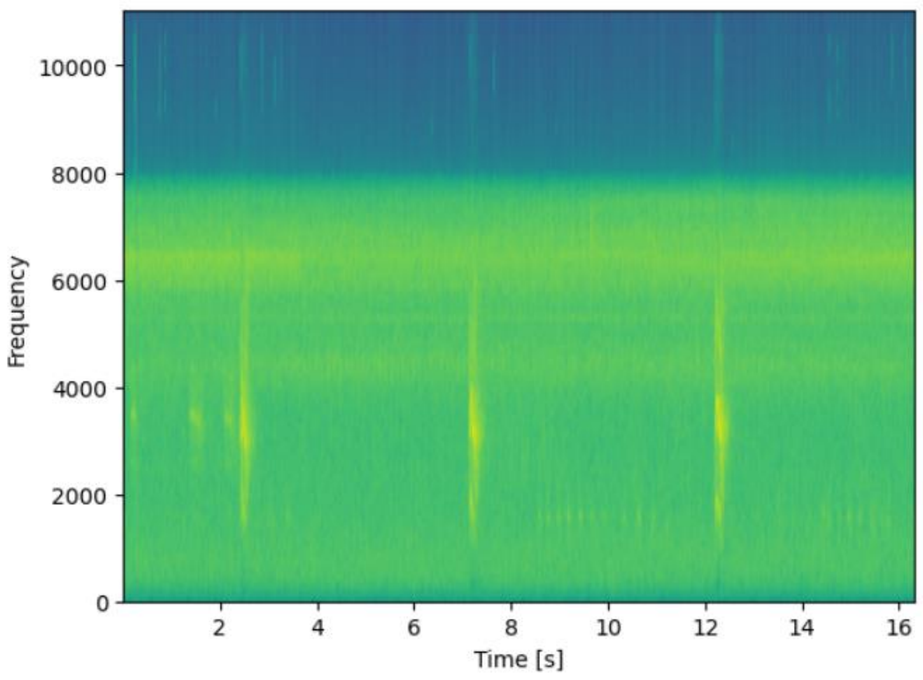
The results show that the transfer learning model achieved a train accuracy of 94.27% and a test accuracy of 92.94%, with a train loss of 26.96% and a test loss of 32.54%. It gives better accuracy compared to both of the multi-class models.

## 6. DISCUSSION

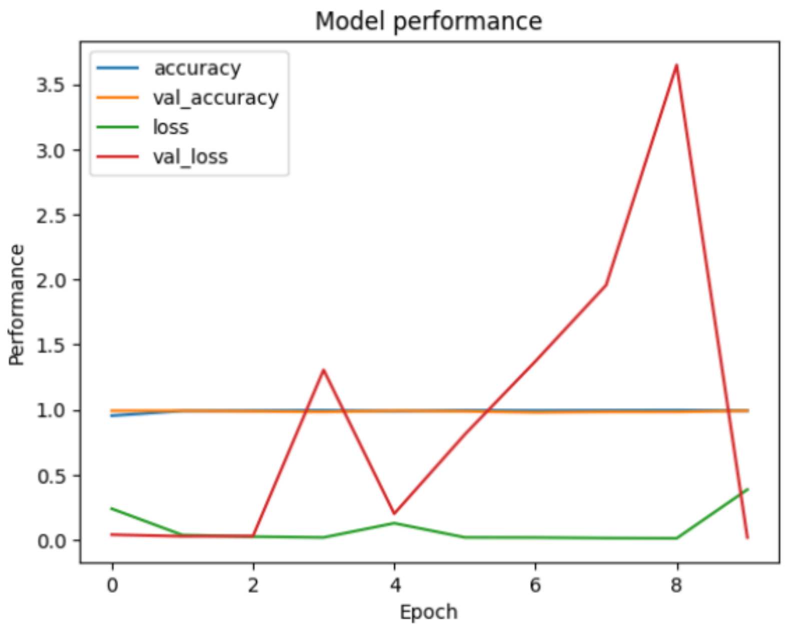
### 6.1 Binary Classification:



Spectrogram of amecro sound clip

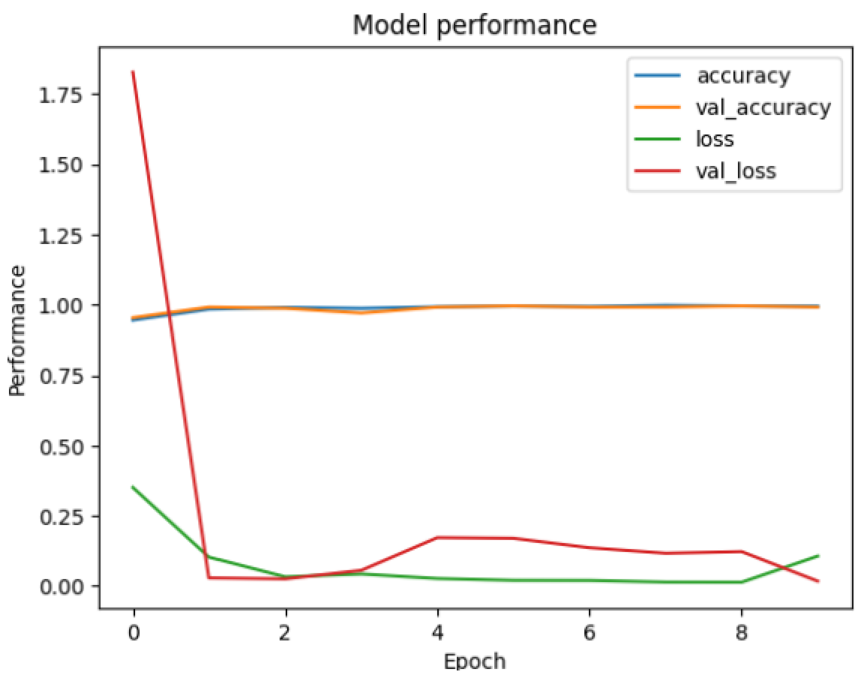


Spectrogram of norfli sound clip.



Model 1 result

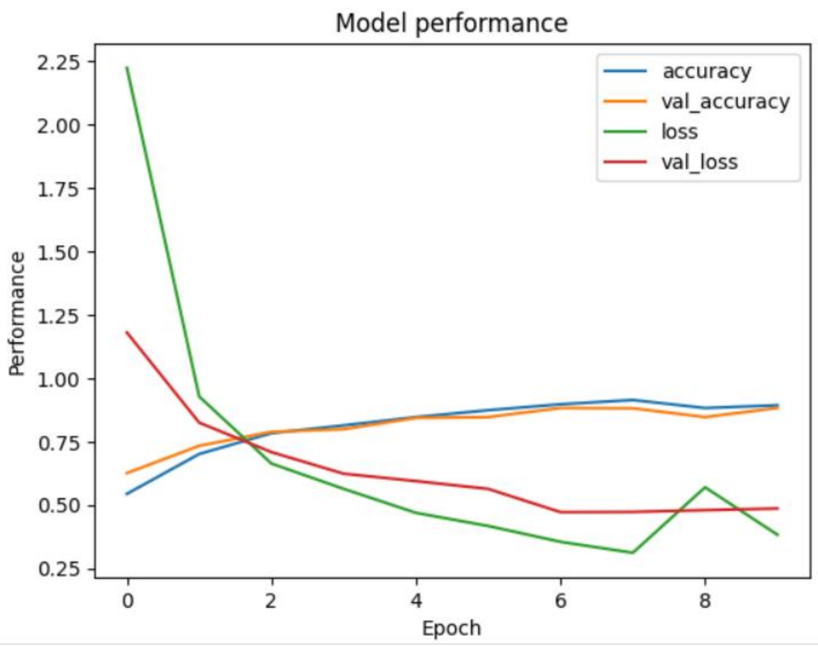
We can see the validation loss is increasing with increase in epochs which may suggest overfitting.



Model 2(with dropout layer) result

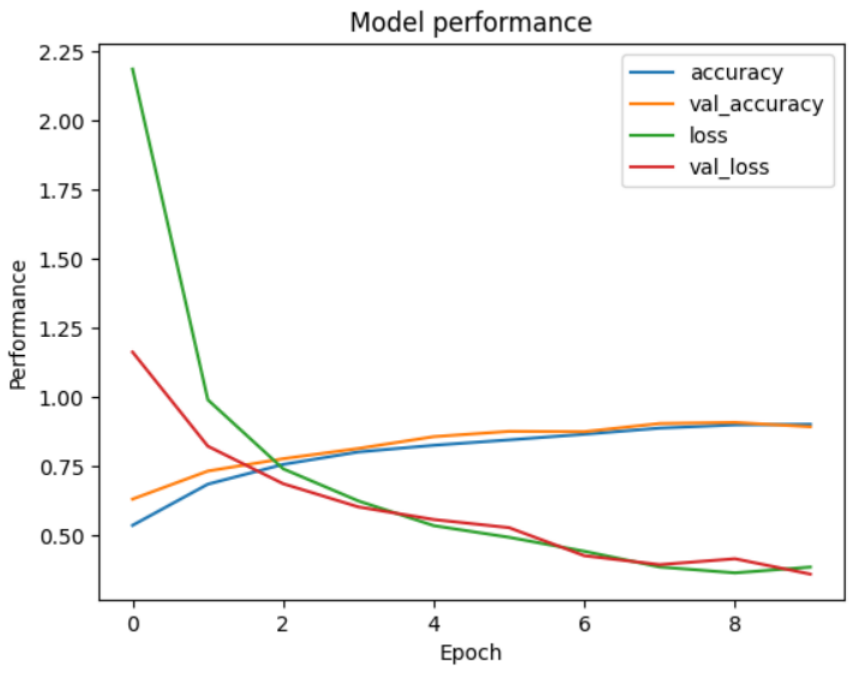
We can see the loss of both the training and validation data is decreasing and the accuracy is increased. It may suggest Model 2's use of dropout regularization may help prevent overfitting and improve the model's generalization performance on unseen data.

### 6.2 Multi-Class Classification



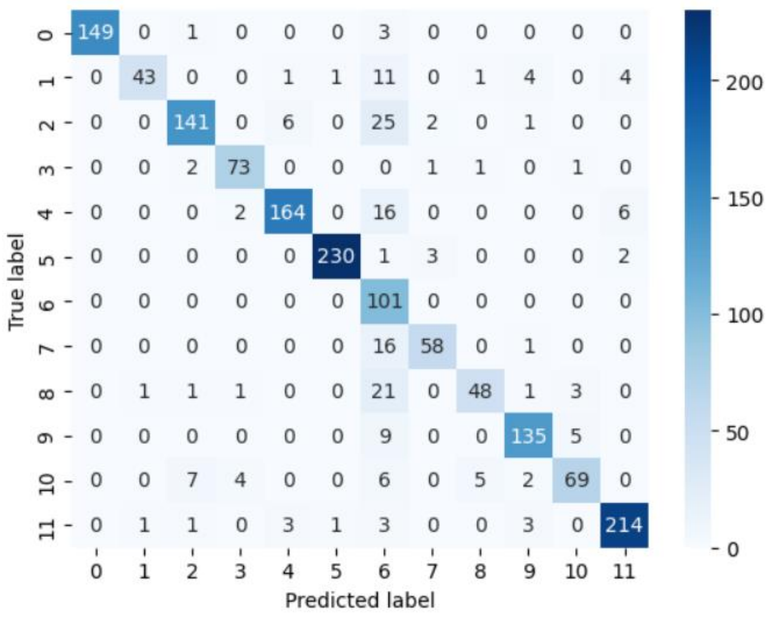
Model 1

We can see the validation loss is higher than the training loss which may suggest overfitting.



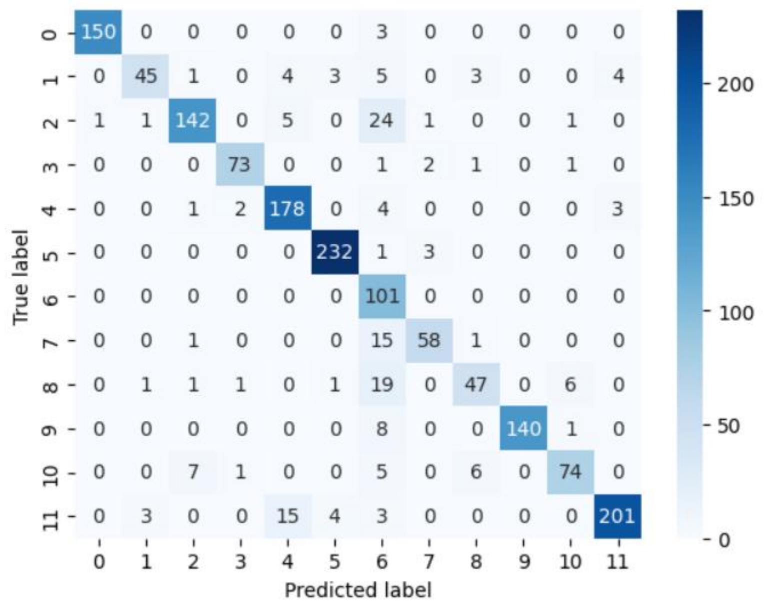
Model 2(with dropout layer)

We can see model2 has much less loss which can suggest that the addition of the dropout layer has helped to reduce overfitting and improve the model's generalization ability.



|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 149 | 0 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 43 | 0 | 0 | 1 | 1 | 11 | 0 | 1 | 4 | 0 | 4 |
| 2 | 0 | 0 | 141 | 0 | 6 | 0 | 25 | 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 2 | 73 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 2 | 164 | 0 | 16 | 0 | 0 | 0 | 0 | 6 |
| 5 | 0 | 0 | 0 | 0 | 0 | 230 | 1 | 3 | 0 | 0 | 0 | 2 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 101 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 58 | 0 | 1 | 0 | 0 |
| 8 | 0 | 1 | 1 | 1 | 0 | 0 | 21 | 0 | 48 | 1 | 3 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 | 0 | 135 | 5 | 0 |
| 10 | 0 | 0 | 7 | 4 | 0 | 0 | 6 | 0 | 5 | 2 | 69 | 0 |
| 11 | 0 | 1 | 1 | 0 | 3 | 1 | 3 | 0 | 0 | 3 | 0 | 214 |

Model 1 Confusion Matrix



| True label \ Predicted label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 150 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 45 | 1 | 0 | 4 | 3 | 5 | 0 | 3 | 0 | 0 | 4 |
| 2 | 1 | 1 | 142 | 0 | 5 | 0 | 24 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 73 | 0 | 0 | 1 | 2 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 2 | 178 | 0 | 4 | 0 | 0 | 0 | 0 | 3 |
| 5 | 0 | 0 | 0 | 0 | 0 | 232 | 1 | 3 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 101 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 1 | 0 | 0 | 0 | 15 | 58 | 1 | 0 | 0 | 0 |
| 8 | 0 | 1 | 1 | 1 | 0 | 1 | 19 | 0 | 47 | 0 | 6 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 140 | 1 | 0 |
| 10 | 0 | 0 | 7 | 1 | 0 | 0 | 5 | 0 | 6 | 0 | 74 | 0 |
| 11 | 0 | 3 | 0 | 0 | 15 | 4 | 3 | 0 | 0 | 0 | 0 | 201 |

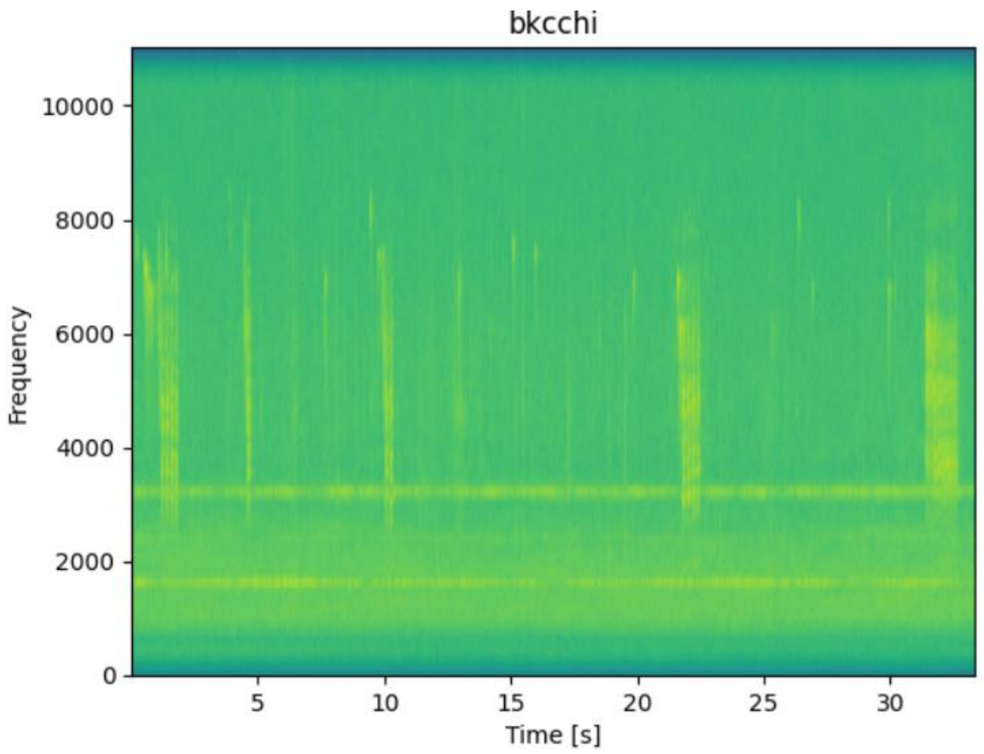
True label
Predicted label
200
150
100
50
0

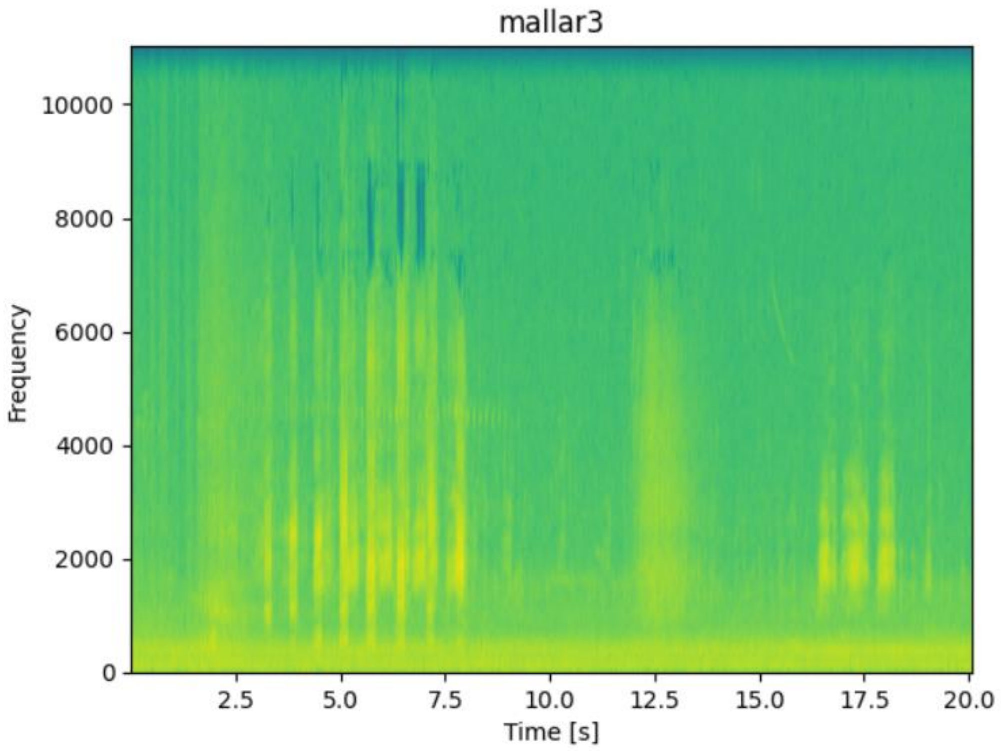
Model 2 Confusion matrix

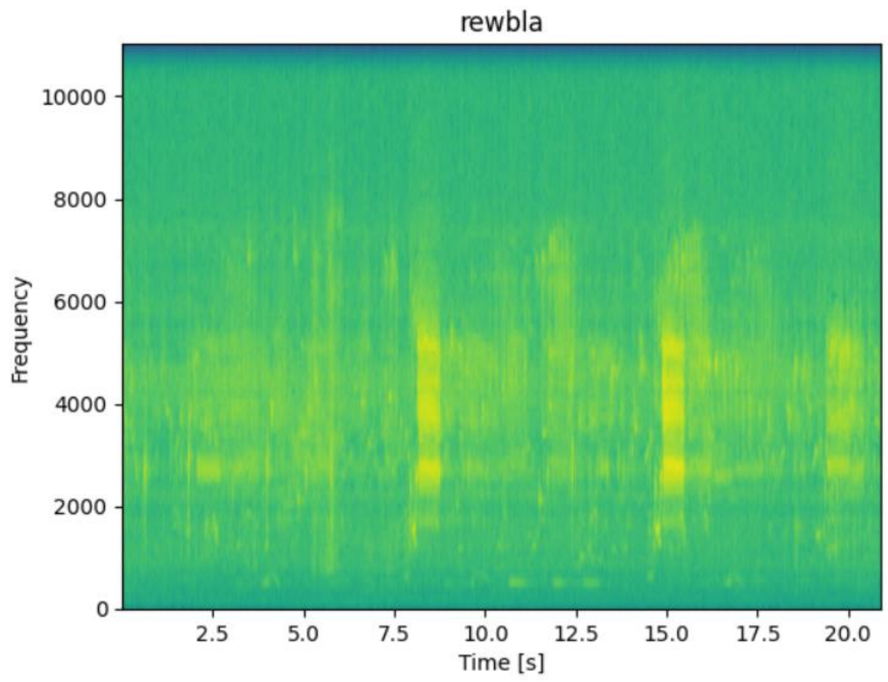
If we see the confusion matrix, most misclassified labels are labels of class 2 and class 8 are misclassified as class 6.

```
0: amecro
1: barswa
2: bkcchi
3: blujay
4: daejun
5: houfin
6: mallar3
7: norfli
8: rewbla
9: stejay
10: wesmea
11: whcspa
```

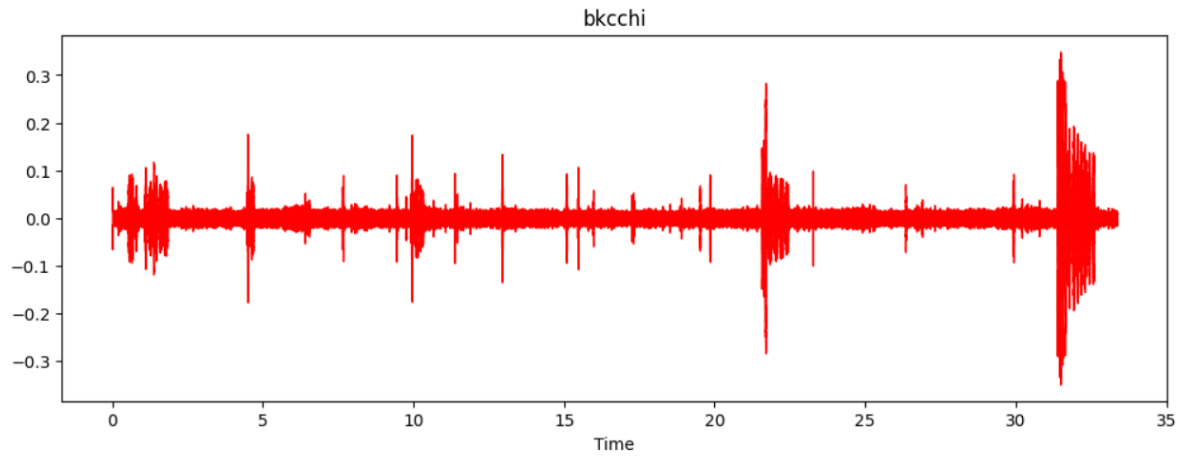
So, bkcchi and rewbla are often misclassified as mallar3. Let us see the spectrogram and wave plots of these 3 birds audio clips.

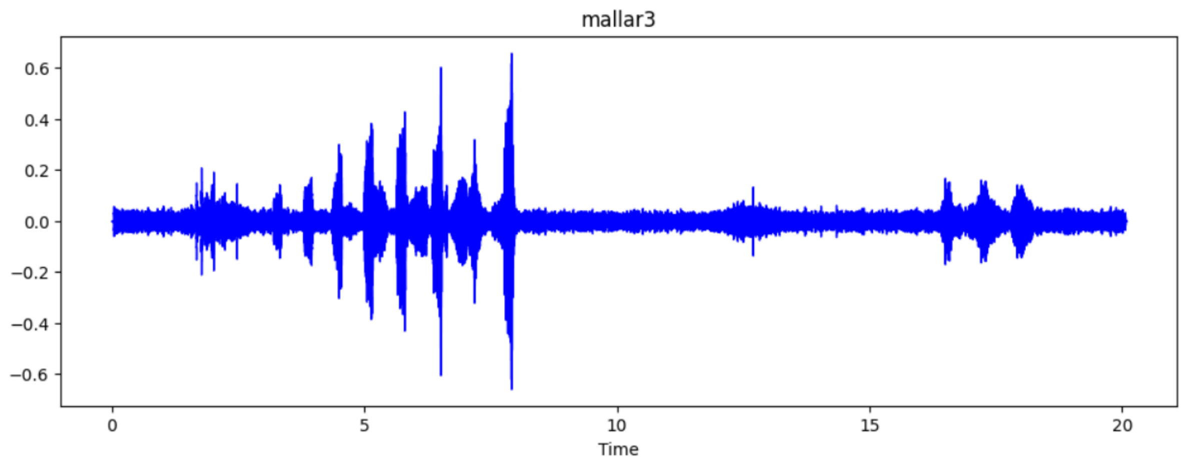

bkcchi
Frequency
Time [s]
0
2000
4000
6000
8000
10000
5
10
15
20
25
30

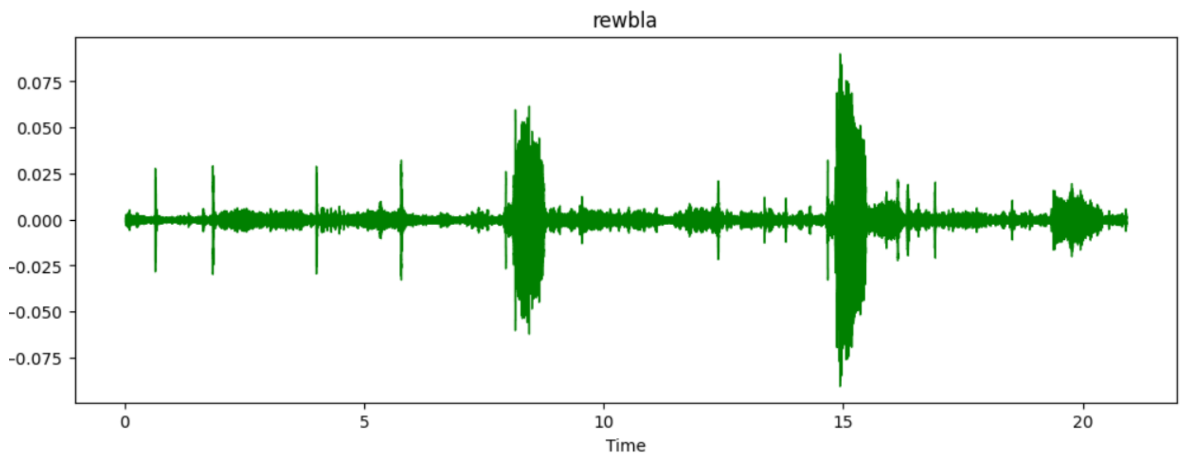

mallar3
Frequency
Time [s]
0
2000
4000
6000
8000
10000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Some of the loud sounds of the 3 audio clips looks kind of similar. Let us see the wave plots to see more clearly.







We can see few of the loud piches looks kind of similar and taking 2 second windows may give similar spectrograms leading to little noise and misclassifications in the model.

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

Running some of the neural networks took a lot of time. Multi-class classification took almost 20 minutes for each model. I increased the batch_size inorder to run it fastly. VGG16 transfer learning took almost one hour time even though I increased the batch size.

**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?**

Some bird species have similar or overlapping calls, making it difficult to differentiate between them. The characteristics of the bird calls, such as pitch, duration, and pattern, could be factors that contribute to the difficulty in distinguishing between them.

Especially some of the bkcchi and rewbla species are misclassified as mallar3 sometimes due to similar spectrograms of 2 second windows.

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

Other models that could be used for this task include decision trees, support vector machines, and random forests. A neural network makes sense for this application because it can learn and extract features from complex data such as spectrograms, which can be difficult for traditional machine learning models to accomplish. Neural networks also have the advantage of being able to generalize well to unseen data, making them suitable for classification tasks such as this one.

## 7. CONCLUSION

In conclusion, the study focused on three different classification tasks: Binary Classification, Multi-Class Classification, and Transfer Learning, all of which aimed to classify different bird species based on their images or audio data. The study used various deep learning models such as CNN, Sequential CNN, and pre-trained VGG16, and evaluated their performance based on accuracy and loss metrics. The results showed that both binary classification models performed well on testing data, with Model 2 having a slight edge due to its use of dropout regularization. In multi-class classification, Model 2 with dropout performed significantly better than Model 1, indicating that dropout regularization can help improve the model's generalization ability. Lastly, the transfer learning approach using pre-trained VGG16 performed best with an accuracy of 92.94%, demonstrating the effectiveness of transfer learning in audio classification tasks. Overall, the study provided valuable insights into using deep learning techniques for bird species classification and provided a foundation for further research in the field.

## 8. BIBLIOGRAPHY

1. Alfonseca, E., Ortega, J. M., Sánchez, J., & Hernández, F. (2006). Automatic bird sound recognition using hidden Markov models with embedded duration models. Journal of VLSI signal processing systems for signal, image, and video technology, 45(3), 263-276.

2. Gao, F., Zhang, S., & Wu, Q. (2019). Bird species identification using deep convolutional neural networks. Journal of signal processing systems, 91(5), 523-534.

3. He, X., & Chen, J. (2017). Bird sound classification based on convolutional neural network. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 221-225). IEEE.

4. Xeno-Canto. (2021). Xeno-Canto: Sharing bird sounds from around the world. https://www.xeno-canto.org/

5. Birdcall Competition Data, Kaggle, https://www.kaggle.com/c/birdsong-recognition/data

6. scikit-learn: Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. https://scikit-learn.org/stable/about.html#citing-scikit-learn

7. Keras, https://keras.io/

8. Iqbal H. Sarker's article: Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions, https://link.springer.com/article/10.1007/s42979-021-00815-1

9. Schmidhuber, J. (2014). Deep Learning in Neural Networks: An Overview, https://arxiv.org/abs/1404.7828

10. Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2021). Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark, https://arxiv.org/abs/2109.14545

11. Sharkawy, A. (2020). Principle of Neural Network and Its Main Types: Review. Journal of Advances in Applied & Computational Mathematics, 7(1), 8-19. DOI: 10.15377/2409-5761.2020.07.2, https://www.researchgate.net/publication/343837591_Principle_of_Neural_Network_and_Its_Main_Types_Review#:~:text=Their%20main%20and%20popular%20types%20such%20as%20the,are%20included%20as%20well%20as%20the%20training%20process.

# 9.APPENDIX

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# for displaying the audio
from IPython.display import Audio

# for loading the audio file
import librosa
import librosa.display
import random


from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
os.chdir('/content/drive/MyDrive/5322/')
```

## Accessing Audio Files

### Audio File 1 - amecro

```
audio_recording="original_clips/amecro/XC304860.mp3"
data,rate=librosa.load(audio_recording)
print(type(data),type(rate))
```

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

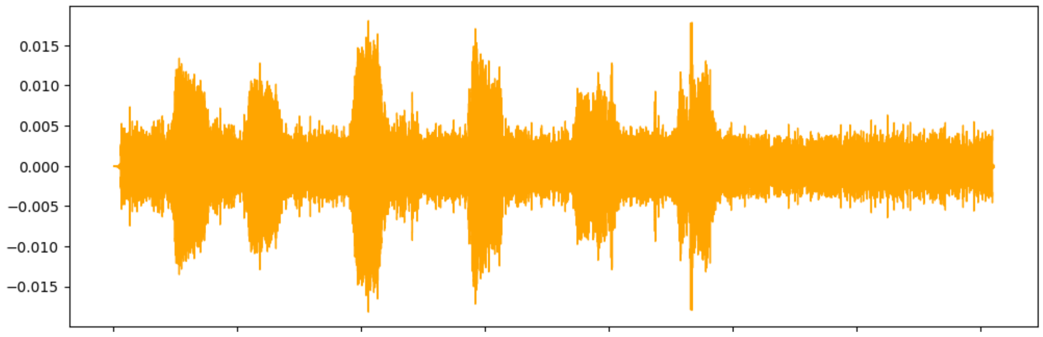
```
print(f"Sample Rate: {rate} Hz ")
print(f"Number of Samples : {data.shape[0]}")
length = data.shape[0] / rate
print("Length of the audio file is {:.2f} seconds".format(length))
```

```
    Sample Rate: 22050 Hz
    Number of Samples : 78336
    Length of the audio file is 3.55 seconds
```

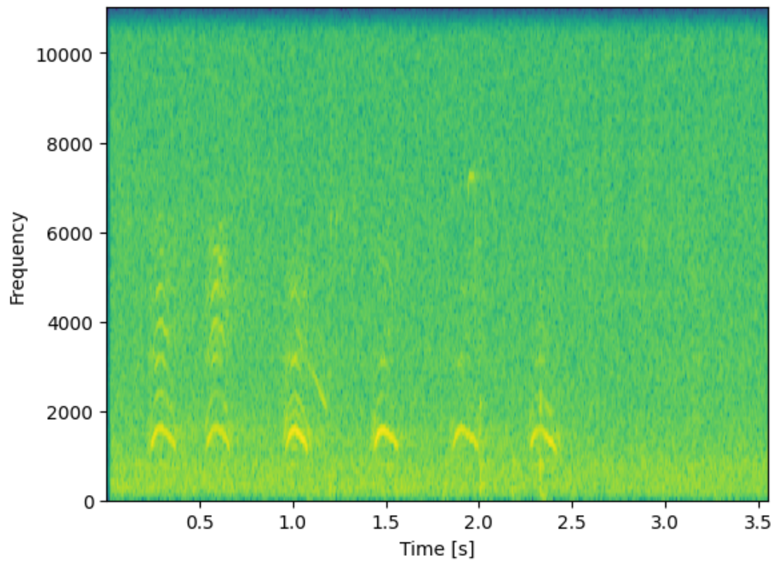
```
Audio(data,rate=rate)
```

0:00 / 0:03

```
plt.figure(figsize=(12,4))
librosa.display.waveshow(data,color="orange")
plt.show()
```



```
plt.specgram(data, Fs=rate)
plt.xlabel("Time [s]")
plt.ylabel("Frequency");
```



## Audio File 2 - norfli

```
audio_recording="original_clips/norfli/XC57036.mp3"
data,rate=librosa.load(audio_recording)
print(type(data),type(rate))
```

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

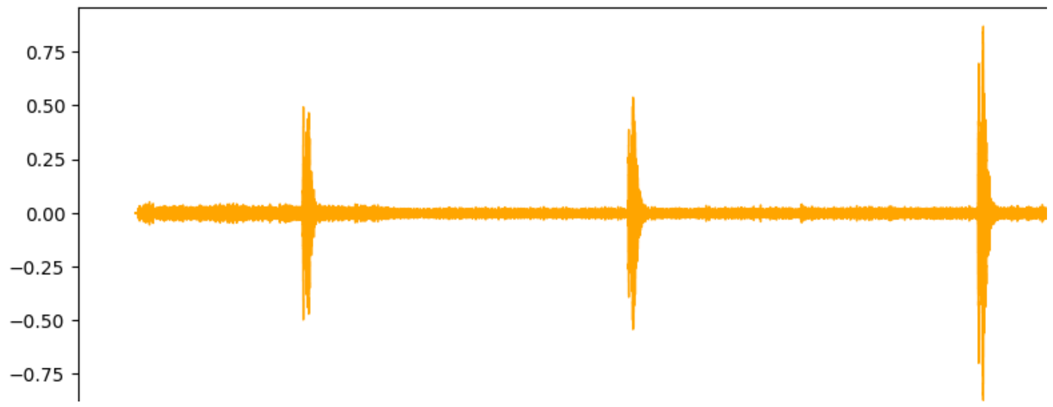
```
print(f"Sample Rate: {rate} Hz ")
print(f"Number of Samples : {data.shape[0]}")
length = data.shape[0] / rate
print("Length of the audio file is {:.2f} seconds".format(length))
```

```
    Sample Rate: 22050 Hz
    Number of Samples : 360000
    Length of the audio file is 16.33 seconds
```

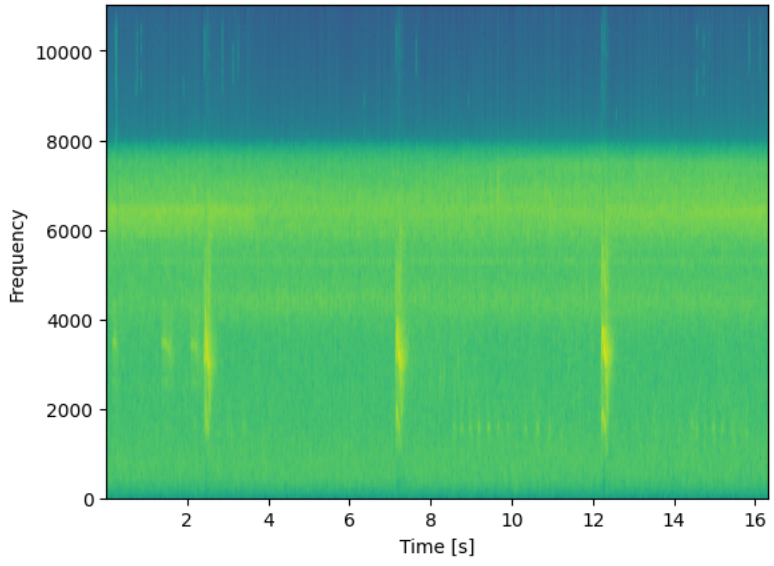
```
Audio(data,rate=rate)
```

0:00 / 0:16

```
plt.figure(figsize=(12,4))
librosa.display.waveshow(data,color="orange")
plt.show()
```



Show code



## BINARY CLASSIFICATION

**Amecro VS Norfli**

## Data Preprocessing

```
# Define the folder path where the bird audio files are stored
folder_path = 'original_clips/'
# Define the new sample rate to which the audio will be resampled
new_sample_rate = 22050
# Define the minimum length (in seconds) of a "loud" part of the sound clip
min_loud_part_length = 0.5
# Define the window size (in seconds) for selecting bird calls
bird_call_window_size = 2
# Define the spectrogram parameters
n_fft = 2048
hop_length = 512
# Initialize the lists for storing the spectrograms and bird names
audios = []
labels = []
# Loop through each bird folder
for bird_folder in os.listdir(folder_path):
  if bird_folder == 'amecro' or bird_folder == 'norfli':
    # Get the full path of the bird folder
    bird_folder_path = os.path.join(folder_path, bird_folder)
    for audio_file in os.listdir(bird_folder_path):
      # Get the full path of the audio file
      audio_file_path = os.path.join(bird_folder_path, audio_file)
      # Load the audio file and resample to the new sample rate
      y, sr = librosa.load(audio_file_path, sr=new_sample_rate)
      # Compute the energy of the audio signal
      energy = librosa.feature.rms(y=y, frame_length=n_fft, hop_length=hop_length)
      # Find the indexes of the "loud" parts of the sound clip
      loud_indexes = np.where(energy > np.max(energy)*0.5)[1]
      # Loop through each "loud" part of the sound clip
      for loud_idx in loud_indexes:
        # Compute the start and end times of the bird call window
        start_time = loud_idx*hop_length/sr
        end_time = start_time + bird_call_window_size
        # Extract the audio signal within the bird call window
        bird_call_audio = y[int(start_time*sr):int(end_time*sr)]
        # Compute the spectrogram of the bird call audio
        spectrogram = librosa.feature.melspectrogram(y=bird_call_audio, sr=sr, n_fft=n_fft, hop_length=hop_length)
        # Add the spectrogram to the list of audios
        audios.append(spectrogram)
        # Add the bird name to the list of labels
        labels.append(bird_folder)
      # Uncomment the following lines to display the spectrograms
      # plt.figure(figsize=(10, 4))
      # librosa.display.specshow(librosa.power_to_db(spectrogram, ref=np.max), y_axis='mel', x_axis='time')
      # plt.colorbar(format='%+2.0f dB')
      # plt.title('Mel spectrogram')
      # plt.tight_layout()
      # plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/librosa/core/spectrum.py:256: UserWarning: n_fft=2048 is too large for input signal of length=16
  warnings.warn(
```

```
# Pad the spectrograms with zeros to ensure they all have the same shape
max_length = max([s.shape[1] for s in audios])
padded_audios = []
for s in audios:
  pad_width = max_length - s.shape[1]
  s_padded = np.pad(s, pad_width=((0, 0), (0, pad_width)), mode='constant')
  padded_audios.append(s_padded)

# Convert the padded audios to a numpy array
audios = np.array(padded_audios)

# Prepare the labels
label_map = {bird_species: i for i, bird_species in enumerate(np.unique(labels))}
labels = np.array([label_map[label] for label in labels])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(audios, labels, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

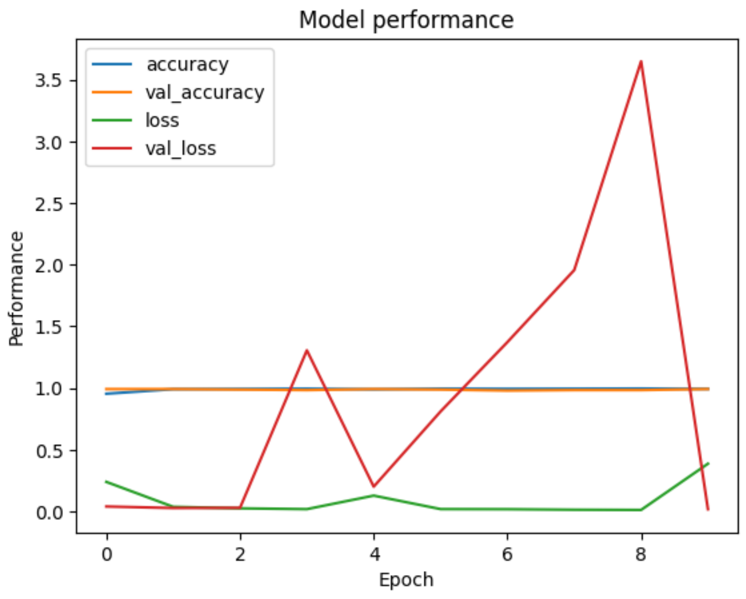
```
(952, 128, 87) (239, 128, 87) (952,) (239,)
```

## Model1

```
# Set the seed for reproducibility
np.random.seed(123)
tf.random.set_seed(123)

b1 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(128,87,1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

b1.compile(optimizer='adam', loss='binary_crossentropy',  metrics=['accuracy'])

# Train the model and store the history
history = b1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10)

# Plot the accuracy and loss
```

```
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()
```

```
Epoch 1/10
30/30 [==============================] - 12s 390ms/step - loss: 0.2392 - accuracy: 0.9538 - val_loss: 0
Epoch 2/10
30/30 [==============================] - 10s 334ms/step - loss: 0.0372 - accuracy: 0.9905 - val_loss: 0
Epoch 3/10
30/30 [==============================] - 11s 360ms/step - loss: 0.0248 - accuracy: 0.9926 - val_loss: 0
Epoch 4/10
30/30 [==============================] - 15s 505ms/step - loss: 0.0184 - accuracy: 0.9947 - val_loss: 1
Epoch 5/10
30/30 [==============================] - 13s 442ms/step - loss: 0.1280 - accuracy: 0.9895 - val_loss: 0
Epoch 6/10
30/30 [==============================] - 12s 389ms/step - loss: 0.0188 - accuracy: 0.9947 - val_loss: 0
Epoch 7/10
30/30 [==============================] - 13s 429ms/step - loss: 0.0178 - accuracy: 0.9937 - val_loss: 1
Epoch 8/10
30/30 [==============================] - 12s 411ms/step - loss: 0.0136 - accuracy: 0.9947 - val_loss: 1
Epoch 9/10
30/30 [==============================] - 12s 414ms/step - loss: 0.0120 - accuracy: 0.9958 - val_loss: 3
Epoch 10/10
30/30 [==============================] - 12s 385ms/step - loss: 0.3866 - accuracy: 0.9926 - val_loss: 0
```



```
train_loss, train_acc = b1.evaluate(X_train, y_train)
print(f'Train loss: {train_loss}, Train accuracy: {train_acc}')
```

```
30/30 [==============================] - 2s 76ms/step - loss: 0.0176 - accuracy: 0.9958
Train loss: 0.017626384273171425, Train accuracy: 0.9957982897758484
```

```
test_loss, test_acc = b1.evaluate(X_test, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_acc}')
```

```
8/8 [==============================] - 1s 72ms/step - loss: 0.0179 - accuracy: 0.9916
Test loss: 0.017858752980828285, Test accuracy: 0.991631805896759
```

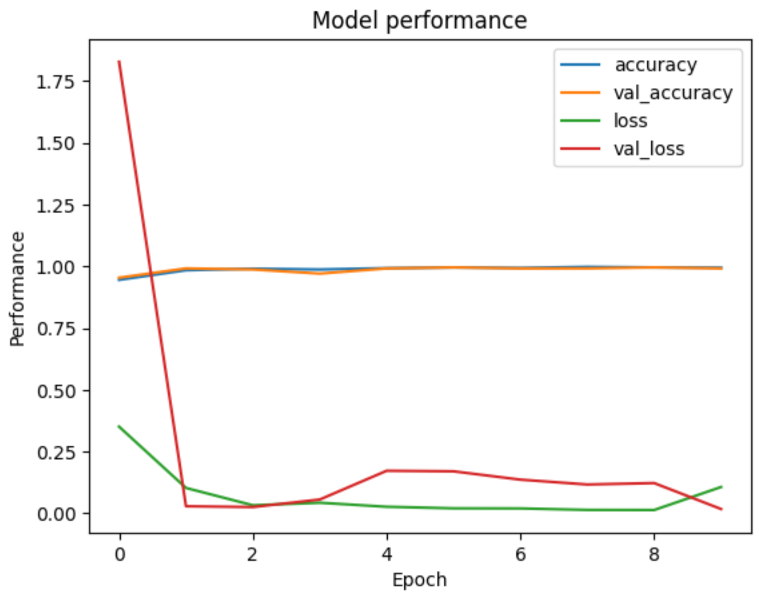
## Model 2

Adding dropout

```
# Set the seed for reproducibility
np.random.seed(123)
tf.random.set_seed(123)
```

```
b2 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(128,87,1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

b2.compile(optimizer='adam', loss='binary_crossentropy',  metrics=['accuracy'])

history = b2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10)

# Plot the accuracy and loss
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()
```

```
Epoch 1/10
30/30 [==============================] - 14s 455ms/step - loss: 0.3510 - accuracy: 0.9454 - val_loss: 1
Epoch 2/10
30/30 [==============================] - 12s 414ms/step - loss: 0.1030 - accuracy: 0.9842 - val_loss: 0
Epoch 3/10
30/30 [==============================] - 12s 410ms/step - loss: 0.0328 - accuracy: 0.9905 - val_loss: 0
Epoch 4/10
30/30 [==============================] - 12s 416ms/step - loss: 0.0430 - accuracy: 0.9874 - val_loss: 0
Epoch 5/10
30/30 [==============================] - 13s 438ms/step - loss: 0.0268 - accuracy: 0.9926 - val_loss: 0
Epoch 6/10
30/30 [==============================] - 12s 415ms/step - loss: 0.0204 - accuracy: 0.9958 - val_loss: 0
Epoch 7/10
30/30 [==============================] - 12s 409ms/step - loss: 0.0200 - accuracy: 0.9937 - val_loss: 0
Epoch 8/10
30/30 [==============================] - 12s 416ms/step - loss: 0.0140 - accuracy: 0.9979 - val_loss: 0
Epoch 9/10
30/30 [==============================] - 15s 493ms/step - loss: 0.0136 - accuracy: 0.9958 - val_loss: 0
Epoch 10/10
30/30 [==============================] - 16s 531ms/step - loss: 0.1067 - accuracy: 0.9947 - val_loss: 0
```



```
train_loss, train_acc = b2.evaluate(X_train, y_train)
print(f'Train loss: {train_loss}, Train accuracy: {train_acc}')
```

```
30/30 [==============================] - 2s 81ms/step - loss: 0.0166 - accuracy: 0.9937
Train loss: 0.016574658453464508, Train accuracy: 0.993697464466095
```

```
test_loss, test_acc = b2.evaluate(X_test, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_acc}')

     8/8 [==============================] - 1s 75ms/step - loss: 0.0176 - accuracy: 0.9916
     Test loss: 0.017574425786733627, Test accuracy: 0.991631805896759
```

## MULTI-CLASS CLASSIFICATION

**12 SPECIES**

- amecro, barswa, bkcchi, blujay, daejun, houfin, mallar3, norfli, rewbla, stejay, wesmea, whcspa

## Data Preprocessing

```
# Define the folder path where the bird audio files are stored
folder_path = 'original_clips/'
# Define the new sample rate to which the audio will be resampled
new_sample_rate = 22050
# Define the minimum length (in seconds) of a "loud" part of the sound clip
min_loud_part_length = 0.5
# Define the window size (in seconds) for selecting bird calls
bird_call_window_size = 2
# Define the spectrogram parameters
n_fft = 2048
hop_length = 512
# Initialize the lists for storing the spectrograms and bird names
audios = []
labels = []
# Loop through each bird folder
for bird_folder in os.listdir(folder_path):
  # Get the full path of the bird folder
  bird_folder_path = os.path.join(folder_path, bird_folder)
  # Loop through each audio clip in the bird folder
  for audio_file in os.listdir(bird_folder_path):
    # Get the full path of the audio file
    audio_file_path = os.path.join(bird_folder_path, audio_file)
    # Load the audio file and resample to the new sample rate
    y, sr = librosa.load(audio_file_path, sr=new_sample_rate)
    # Compute the energy of the audio signal
    energy = librosa.feature.rms(y=y, frame_length=n_fft, hop_length=hop_length)
    # Find the indexes of the "loud" parts of the sound clip
    loud_indexes = np.where(energy > np.max(energy)*0.5)[1]
    # Loop through each "loud" part of the sound clip
    for loud_idx in loud_indexes:
      # Compute the start and end times of the bird call window
      start_time = loud_idx*hop_length/sr
      end_time = start_time + bird_call_window_size
      # Extract the audio signal within the bird call window
      bird_call_audio = y[int(start_time*sr):int(end_time*sr)]
      # Compute the spectrogram of the bird call audio
      spectrogram = librosa.feature.melspectrogram(y=bird_call_audio, sr=sr, n_fft=n_fft, hop_length=hop_length)
      # Add the spectrogram to the list of audios
      audios.append(spectrogram)
      # Add the bird name to the list of labels
      labels.append(bird_folder)
      # Uncomment the following lines to display the spectrograms
      # plt.figure(figsize=(10, 4))
      # librosa.display.specshow(librosa.power_to_db(spectrogram, ref=np.max),
      #                          y_axis='mel', x_axis='time')
      # plt.colorbar(format='%+2.0f dB')
      # plt.title('Mel spectrogram')
      # plt.tight_layout()
      # plt.show()


# Pad the spectrograms with zeros to ensure they all have the same shape
max_length = max([s.shape[1] for s in audios])
padded_audios = []
for s in audios:
  pad_width = max_length - s.shape[1]
  s_padded = np.pad(s, pad_width=((0, 0), (0, pad_width)), mode='constant')
  padded_audios.append(s_padded)
```

```
# Convert the padded audios to a numpy array
audios = np.array(padded_audios)

# Prepare the labels
label_map = {bird_species: i for i, bird_species in enumerate(np.unique(labels))}
labels = np.array([label_map[label] for label in labels])

num_classes = 12
labels = to_categorical(labels, num_classes=num_classes)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(audios, labels, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(6458, 128, 87) (1615, 128, 87) (6458, 12) (1615, 12)
```

```
# Invert the label_map dictionary
num_to_label = {v: k for k, v in label_map.items()}

# Print the numerical label and corresponding bird species for each label
for i in range(num_classes):
    print(f"{i}: {num_to_label[i]}")
```

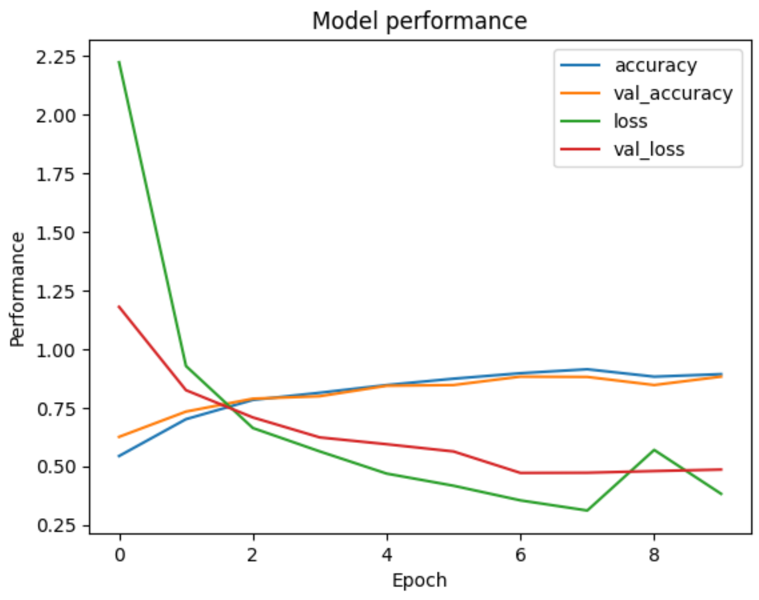
```
0: amecro
1: barswa
2: bkcchi
3: blujay
4: daejun
5: houfin
6: mallar3
7: norfli
8: rewbla
9: stejay
10: wesmea
11: whcspa
```

## Model 1

```
# Set the seed for reproducibility
np.random.seed(123)
tf.random.set_seed(123)

model1 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(128,87,1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(12, activation='softmax')
])

model1.compile(optimizer='adam', loss='categorical_crossentropy',  metrics=['accuracy'])

history = model1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size = 512)
```

```
Epoch 1/10
13/13 [==============================] - 91s 7s/step - loss: 2.2231 - accuracy: 0.5437 - val_loss: 1.1799 - val_accuracy: 0.6254
Epoch 2/10
13/13 [==============================] - 84s 6s/step - loss: 0.9276 - accuracy: 0.7008 - val_loss: 0.8245 - val_accuracy: 0.7331
Epoch 3/10
13/13 [==============================] - 73s 6s/step - loss: 0.6631 - accuracy: 0.7829 - val_loss: 0.7083 - val_accuracy: 0.7882
Epoch 4/10
13/13 [==============================] - 78s 6s/step - loss: 0.5634 - accuracy: 0.8136 - val_loss: 0.6231 - val_accuracy: 0.7988
Epoch 5/10
13/13 [==============================] - 90s 7s/step - loss: 0.4688 - accuracy: 0.8464 - val_loss: 0.5940 - val_accuracy: 0.8433
Epoch 6/10
13/13 [==============================] - 71s 6s/step - loss: 0.4170 - accuracy: 0.8735 - val_loss: 0.5632 - val_accuracy: 0.8464
Epoch 7/10
13/13 [==============================] - 71s 6s/step - loss: 0.3546 - accuracy: 0.8969 - val_loss: 0.4716 - val_accuracy: 0.8824
Epoch 8/10
13/13 [==============================] - 76s 6s/step - loss: 0.3113 - accuracy: 0.9138 - val_loss: 0.4724 - val_accuracy: 0.8811
Epoch 9/10
13/13 [==============================] - 72s 6s/step - loss: 0.5696 - accuracy: 0.8822 - val_loss: 0.4795 - val_accuracy: 0.8464
Epoch 10/10
13/13 [==============================] - 72s 5s/step - loss: 0.3826 - accuracy: 0.8928 - val_loss: 0.4860 - val_accuracy: 0.8824
```

```
# Plot the accuracy and loss
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()
```



```
train_loss, train_acc = model1.evaluate(X_train, y_train)
print(f'Train loss: {train_loss}, Train accuracy: {train_acc}')
```

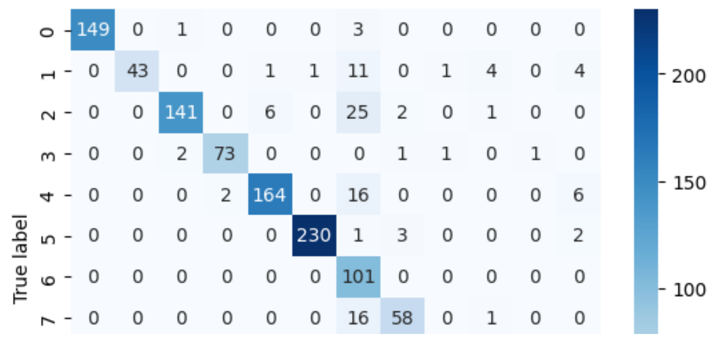
```
202/202 [==============================] - 16s 81ms/step - loss: 0.3115 - accuracy: 0.9096
Train loss: 0.3115084171295166, Train accuracy: 0.909569501876831
```

```
test_loss, test_acc = model1.evaluate(X_test, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_acc}')
```

```
51/51 [==============================] - 5s 95ms/step - loss: 0.4860 - accuracy: 0.8824
Test loss: 0.48595908284187317, Test accuracy: 0.8823529481887817
```

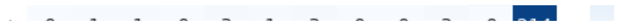
```
# Get predictions for test set
y_pred = model1.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Get true classes for test set
y_true = np.argmax(y_test, axis=1)

# Create confusion matrix
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred_classes)

# Plot heatmap
sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```

```
51/51 [==============================] - 8s 152ms/step
```



## Model2

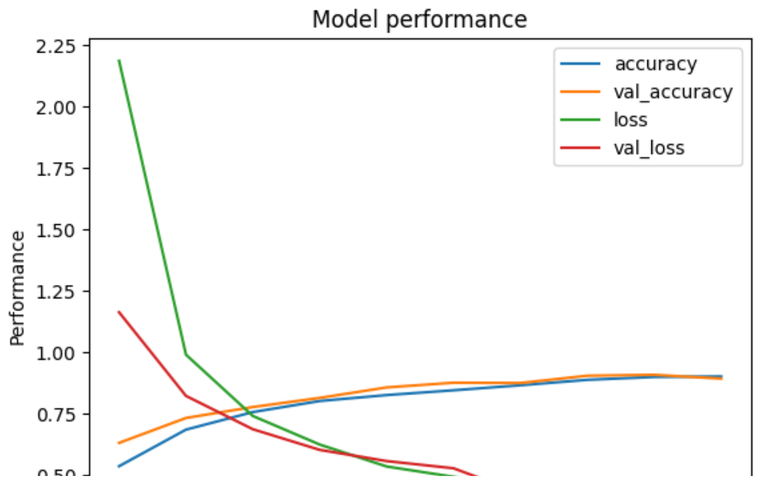
Using dropout



```
# Set the seed for reproducibility
np.random.seed(123)
tf.random.set_seed(123)

model2 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(128,87,1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(12, activation='softmax')
])

model2.compile(optimizer='adam', loss='categorical_crossentropy',  metrics=['accuracy'])

history2 = model2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size = 512)
```

```
Epoch 1/10
13/13 [==============================] - 75s 6s/step - loss: 2.1866 - accuracy: 0.5355 - val_loss: 1.1626 - val_accuracy: 0.6303
Epoch 2/10
13/13 [==============================] - 72s 6s/step - loss: 0.9897 - accuracy: 0.6844 - val_loss: 0.8218 - val_accuracy: 0.7319
Epoch 3/10
13/13 [==============================] - 73s 6s/step - loss: 0.7400 - accuracy: 0.7560 - val_loss: 0.6860 - val_accuracy: 0.7765
Epoch 4/10
13/13 [==============================] - 72s 6s/step - loss: 0.6236 - accuracy: 0.8010 - val_loss: 0.6020 - val_accuracy: 0.8136
Epoch 5/10
13/13 [==============================] - 73s 6s/step - loss: 0.5343 - accuracy: 0.8252 - val_loss: 0.5564 - val_accuracy: 0.8563
Epoch 6/10
13/13 [==============================] - 73s 6s/step - loss: 0.4921 - accuracy: 0.8447 - val_loss: 0.5270 - val_accuracy: 0.8755
Epoch 7/10
13/13 [==============================] - 74s 6s/step - loss: 0.4416 - accuracy: 0.8651 - val_loss: 0.4255 - val_accuracy: 0.8743
Epoch 8/10
13/13 [==============================] - 73s 6s/step - loss: 0.3847 - accuracy: 0.8873 - val_loss: 0.3932 - val_accuracy: 0.9040
Epoch 9/10
13/13 [==============================] - 72s 6s/step - loss: 0.3634 - accuracy: 0.8990 - val_loss: 0.4145 - val_accuracy: 0.9077
Epoch 10/10
13/13 [==============================] - 73s 6s/step - loss: 0.3843 - accuracy: 0.9012 - val_loss: 0.3590 - val_accuracy: 0.8923
```

```
# Plot the accuracy and loss
plt.plot(history2.history['accuracy'], label='accuracy')
plt.plot(history2.history['val_accuracy'], label='val_accuracy')
plt.plot(history2.history['loss'], label='loss')
plt.plot(history2.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()
```



```
train_loss, train_acc = model2.evaluate(X_train, y_train)
print(f'Train loss: {train_loss}, Train accuracy: {train_acc}')
```

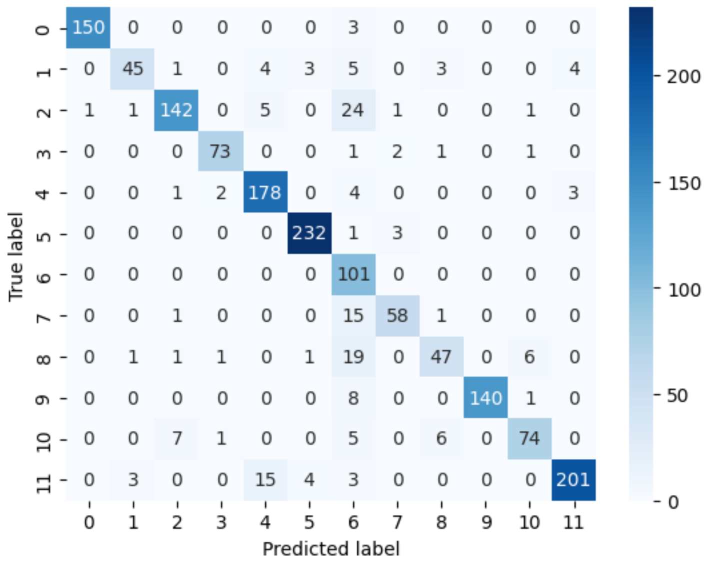
```
    202/202 [==============================] - 18s 87ms/step - loss: 0.2629 - accuracy: 0.9223
    Train loss: 0.2628675103187561, Train accuracy: 0.922266960144043
```

```
test_loss, test_acc = model2.evaluate(X_test, y_test)
print(f'Test loss: {test_loss}, Test accuracy: {test_acc}')
```

```
    51/51 [==============================] - 5s 90ms/step - loss: 0.3590 - accuracy: 0.8923
    Test loss: 0.3589573800563812, Test accuracy: 0.8922600746154785
```

```
# Get predictions for test set
y_pred = model2.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# Get true classes for test set
y_true = np.argmax(y_test, axis=1)

# Create confusion matrix
confusion_mtx = tf.math.confusion_matrix(y_true, y_pred_classes)

# Plot heatmap
sns.heatmap(confusion_mtx, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```

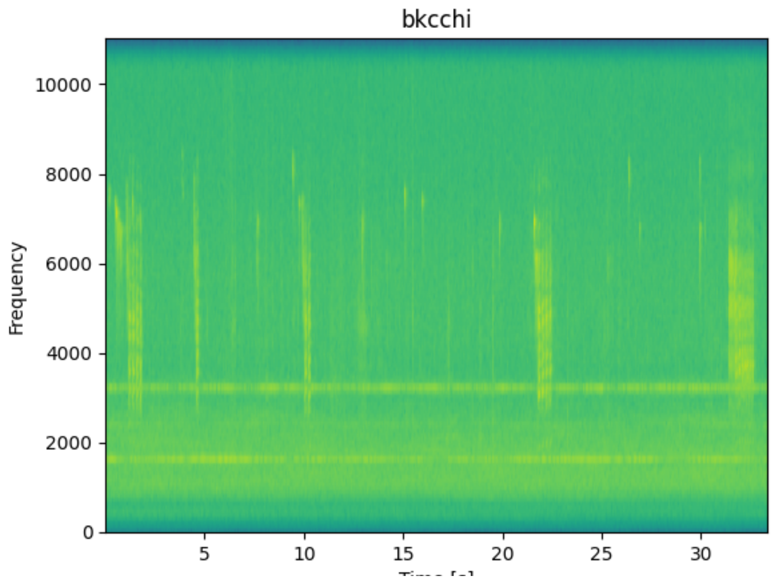
```
    51/51 [==============================] - 6s 109ms/step
```



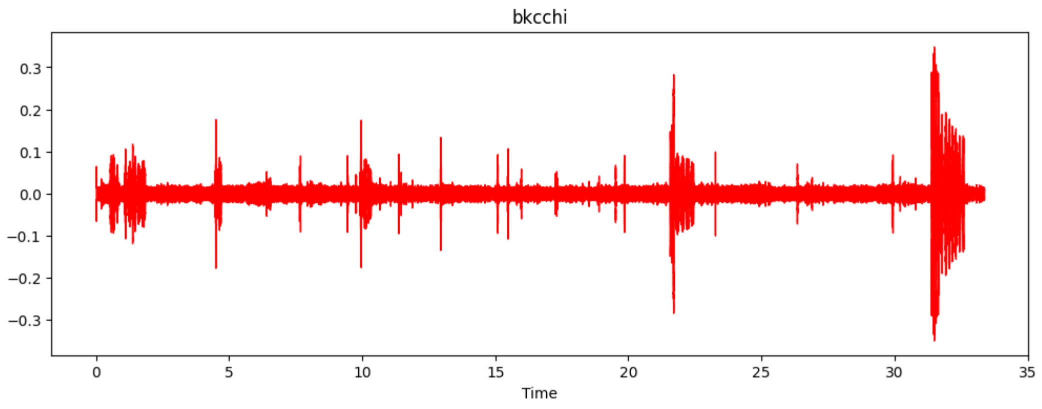
| True \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 150 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 45 | 1 | 0 | 4 | 3 | 5 | 0 | 3 | 0 | 0 | 4 |
| 2 | 1 | 1 | 142 | 0 | 5 | 0 | 24 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 73 | 0 | 0 | 1 | 2 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 2 | 178 | 0 | 4 | 0 | 0 | 0 | 0 | 3 |
| 5 | 0 | 0 | 0 | 0 | 0 | 232 | 1 | 3 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 101 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 1 | 0 | 0 | 0 | 15 | 58 | 1 | 0 | 0 | 0 |
| 8 | 0 | 1 | 1 | 1 | 0 | 1 | 19 | 0 | 47 | 0 | 6 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 140 | 1 | 0 |
| 10 | 0 | 0 | 7 | 1 | 0 | 0 | 5 | 0 | 6 | 0 | 74 | 0 |
| 11 | 0 | 3 | 0 | 0 | 15 | 4 | 3 | 0 | 0 | 0 | 0 | 201 |

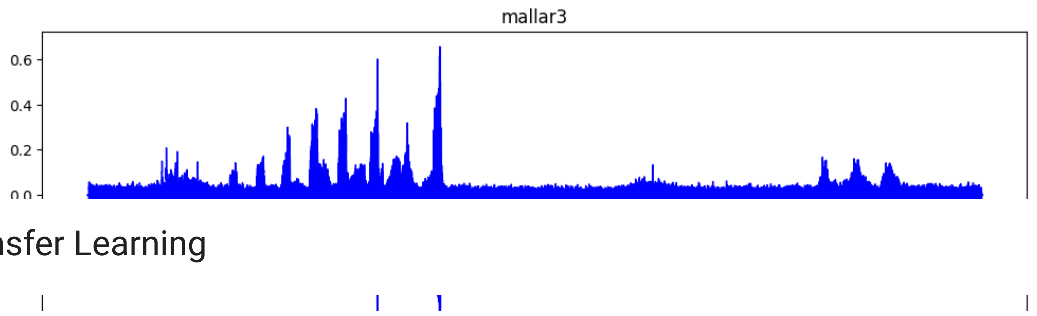
0: amecro 1: barswa 2: bkcchi 3: blujay 4: daejun 5: houfin 6: mallar3 7: norfli 8: rewbla 9: stejay 10: wesmea 11: whcspa

```
audio_recording="original_clips/bkcchi/XC513209.mp3"
data1,rate1=librosa.load(audio_recording)

audio_recording="original_clips/mallar3/XC440476.mp3"
data2,rate2=librosa.load(audio_recording)

audio_recording="original_clips/rewbla/XC470007.mp3"
data3,rate3=librosa.load(audio_recording)


plt.specgram(data1, Fs=rate1)
plt.xlabel("Time [s]")
plt.ylabel("Frequency");
plt.title('bkcchi')
plt.show()

plt.specgram(data2, Fs=rate2)
plt.xlabel("Time [s]")
plt.ylabel("Frequency");
plt.title('mallar3')
plt.show()

plt.specgram(data3, Fs=rate3)
plt.xlabel("Time [s]")
plt.ylabel("Frequency");
plt.title('rewbla')
plt.show()
```



```
plt.figure(figsize=(12,4))
librosa.display.waveshow(data1,color="red")
plt.title('bkcchi')
plt.show()

plt.figure(figsize=(12,4))
librosa.display.waveshow(data2,color="blue")
plt.title('mallar3')
plt.show()

plt.figure(figsize=(12,4))
librosa.display.waveshow(data3,color="green")
plt.title('rewbla')
plt.show()
```





## Transfer Learning

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from skimage.transform import resize

# Load the pre-trained VGG16 model without the top layer
vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=(128, 87, 3))

# Freeze the layers in the pre-trained model
for layer in vgg_model.layers:
    layer.trainable = False

# Add a new top layer for our audio classification task
flatten_layer = Flatten()(vgg_model.output)
dense_layer1 = Dense(128, activation='relu')(flatten_layer)
dense_layer2 = Dense(num_classes, activation='softmax')(dense_layer1)

# Create the new model
model = Model(inputs=vgg_model.input, outputs=dense_layer2)

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Resize the data to add a channel dimension
X_train_resized = []
for sample in X_train:
    resized_sample = resize(sample, (128, 87, 3))
    X_train_resized.append(resized_sample)
X_train_resized = np.array(X_train_resized)

X_test_resized = []
for sample in X_test:
    resized_sample = resize(sample, (128, 87, 3))
    X_test_resized.append(resized_sample)
X_test_resized = np.array(X_test_resized)


# train the model with the generator
model.fit(X_train_resized,y_train,epochs=10, validation_data=(X_test_resized, y_test), batch_size = 512)

# Evaluate the model
score = model.evaluate(X_test_resized, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Epoch 1/10
13/13 [==============================] - 673s 52s/step - loss: 2.6163 - accuracy: 0.4848 - val_loss: 1.3209 - val_accuracy: 0.6118
Epoch 2/10
```

```
13/13 [==============================] - 684s 53s/step - loss: 1.0336 - accuracy: 0.6826 - val_loss: 0.8017 - val_accuracy: 0.7889
Epoch 3/10
13/13 [==============================] - 678s 53s/step - loss: 0.7238 - accuracy: 0.8013 - val_loss: 0.6346 - val_accuracy: 0.8576
Epoch 4/10
13/13 [==============================] - 678s 53s/step - loss: 0.5803 - accuracy: 0.8529 - val_loss: 0.5565 - val_accuracy: 0.8650
Epoch 5/10
13/13 [==============================] - 669s 52s/step - loss: 0.4894 - accuracy: 0.8738 - val_loss: 0.4780 - val_accuracy: 0.8978
Epoch 6/10
13/13 [==============================] - 678s 53s/step - loss: 0.4298 - accuracy: 0.8964 - val_loss: 0.4364 - val_accuracy: 0.9053
Epoch 7/10
13/13 [==============================] - 668s 52s/step - loss: 0.3757 - accuracy: 0.9102 - val_loss: 0.3957 - val_accuracy: 0.9170
Epoch 8/10
13/13 [==============================] - 676s 53s/step - loss: 0.3370 - accuracy: 0.9238 - val_loss: 0.3650 - val_accuracy: 0.9189
Epoch 9/10
13/13 [==============================] - 669s 52s/step - loss: 0.3057 - accuracy: 0.9316 - val_loss: 0.3446 - val_accuracy: 0.9189
Epoch 10/10
13/13 [==============================] - 678s 53s/step - loss: 0.2837 - accuracy: 0.9348 - val_loss: 0.3255 - val_accuracy: 0.9294
51/51 [==============================] - 132s 3s/step - loss: 0.3255 - accuracy: 0.9294
Test loss: 0.32546570897102356
Test accuracy: 0.929411768913269
```

```
#@title
score = model.evaluate(X_train_resized, y_train)
print('Train loss:', score[0])
print('Train accuracy:', score[1])
```

```
202/202 [==============================] - 529s 3s/step - loss: 0.2697 - accuracy: 0.9427
Train loss: 0.26969659328460693
Train accuracy: 0.9427067041397095
```

✓ 9m 24s completed at 10:55 AM