REPORT: Bird Species Classification using Deep Learning

ABSTRACT

The report delves into the application of machine learning and deep learning techniques for the categorization of bird sounds, a useful tool for monitoring bird population health and researching biodiversity. The report delves into the process of sound recognition, from feature extraction to classification, as well as the use of spectrograms and neural networks. Additionally, it explains the methodology used to construct a tailored neural network for bird sound classification, which includes data preprocessing, constructing binary and multi-class classification models, and investigating transfer learning. Furthermore, it addresses hyper parameters and optimization algorithms used to enhance model accuracy, and data augmentation techniques to bolster robustness and generalization, and ultimately present the computational results. Lastly, the report compares the performance of the custom neural network with that of a pre-trained neural network.

INTRODUCTION

For a long time, scientists have been interested in the recognition of bird sounds. Birds are an intriguing subject to examine since they have a distinctive way of communicating through their noises. Now that machine learning and deep learning techniques have been developed, it is possible to utilize these tools to automatically identify and categorize bird sounds. This might be used for a variety of things, like tracking the health of bird populations and researching an area's biodiversity.

The objective of the current study is to create a special neural network that can categorize bird sounds. Spectrograms of bird noises will be used to train and evaluate our neural network. We will also investigate how various network designs and hyper parameters perform to determine how they impact classification accuracy. The performance of our unique neural network will next be compared to that of a pre-trained neural network modified for our data.

THEORETICAL BACKGROUND

The process of sound recognition can be divided into two main stages: feature extraction and classification. Feature extraction involves transforming the raw sound signal into a form that can be used by the classification algorithm.. The use of spectrograms is one common technique for feature extraction in sound recognition. A spectrogram is a visual representation of the spectrum of frequencies of a sound signal as it varies with time. Machine learning techniques are used in classification to categorize the features taken from the sound stream. One popular method for classification is the use of neural networks.

The neural network machine learning algorithm was modeled after the structure of the human brain. They are composed of layers of interconnected nodes, or neurons, that perform basic mathematical operations on the input data. Neural networks can learn to change the weights of the connections between neurons to recognize patterns in the incoming data. There are many different types of neural networks, each with its own benefits and drawbacks. The focus of this study will primarily be on feed-forward neural networks, which are the most basic type of neural network.

An input layer, one or more hidden layers, and an output layer make up feed forward neural networks. The input layer sends the input data to the output layer through the hidden layers, such as the spectrogram of a bird sound. Each layer of the network transforms the input data linearly before applying a non-linear activation function. The weights of the connections between the neurons in the network are changed during training using the gradient descent optimization approach back propagation.

In addition to the neural network's architecture, a variety of hyper parameters can be adjusted to enhance performance. There are four of these: the learning rate, the regularization parameters, and the number of hidden layers and neurons. It can be difficult to determine the best set of hyper parameters, and experimentation that goes wrong is common. However, employing neural networks to do sound recognition tasks can be done with great accuracy with some careful tuning.

METHODOLOGY:

Study started by preprocessing the data, which involved sub sampling the sound clips, selecting 2-second windows with bird calls, and producing spectrograms for each window. Followed by, split the data into training and validation sets and built our custom neural network models using the Keras deep learning library in Python. A binary classification model for two of the bird species, which involved training the neural network to distinguish between the two species, was built.

The traditional approach of using just one clip from a single audio recording for each species could be limiting. It does not account for variations in the sounds produced by the same species in different environments or at different times of the day. By using multiple high pitches in each spectrogram and clipping 3 seconds in each audio file, that can capture a more comprehensive representation of the sound produced by a species. This, in turn, increases the chances of correctly identifying the species present in an audio recording.

To illustrate this point, let's consider an example. Suppose we have 12 species that wanted to be classified in our audio recordings. Using the traditional approach, we would have only one clip from each species, resulting in just 12 sets of data for all the species. However, by using the approach described above, we can have multiple high pitches in each spectrogram, resulting in thousands of audio clips for each species. This translates to 12k sets of data for all 12 species. Here, the value of 'k' represents the number of high pitches in the audio clip that corresponds to a particular species.

The variables 'n_fft', 'hop_length', and 'n_mels' determine the spectrogram's parameters, such as the number of FFT points, hop length, and number of Mel frequency bins. The resulting spectrograms are then saved as grayscale PNG images in the designated directory. The spectrograms can now be used as input data for a deep learning model, which makes this step crucial. We may use convolutional neural networks to categorize the sounds by presenting the audio data as a picture.

Binary classification is comprised of several essential steps. Initially, the data was subjected to preprocessing, encompassing the sub sampling of sound clips and the selection of 2-second windows specifically containing bird calls. Subsequently, spectrograms were generated for each window, facilitating the transformation of audio data into visual representations. Following this, the dataset was divided into training and validation sets. The training set was utilized to train the binary classification model, while the validation set played a crucial role in evaluating the model's performance and making any necessary refinements. By adhering to this methodology, the team successfully developed an effective model capable of distinguishing between the two bird species.

Finally, a multi-class classification model for all 12 species, which involved training the neural network to classify each sound clip into one of the 12 species categories was designed. To explore transfer learning, a pre-trained neural network model was adapted, VGG16, to our data by fine-tuning the model's weights on our dataset. Then comparisons were made against the performance of this adapted model to our custom neural network models.

To improve the accuracy of the model, various optimization algorithms such as Adam, SGD, etc., are used along with learning rates to fit the model better. Data augmentation techniques such as rotation, flipping, resizing, and cropping are also employed to help the model learn the data in different perspectives, thereby improving its robustness and generalization ability. By doing this, the model can better understand the variability in the data and learn more accurate representations of the sounds, resulting in improved classification performance.

COMPUTATIONAL RESULTS:

We will start our analysis by considering any two species first and use binary cross entropy loss function to classify between those two species along with accuracies, neural network architecture as follows:

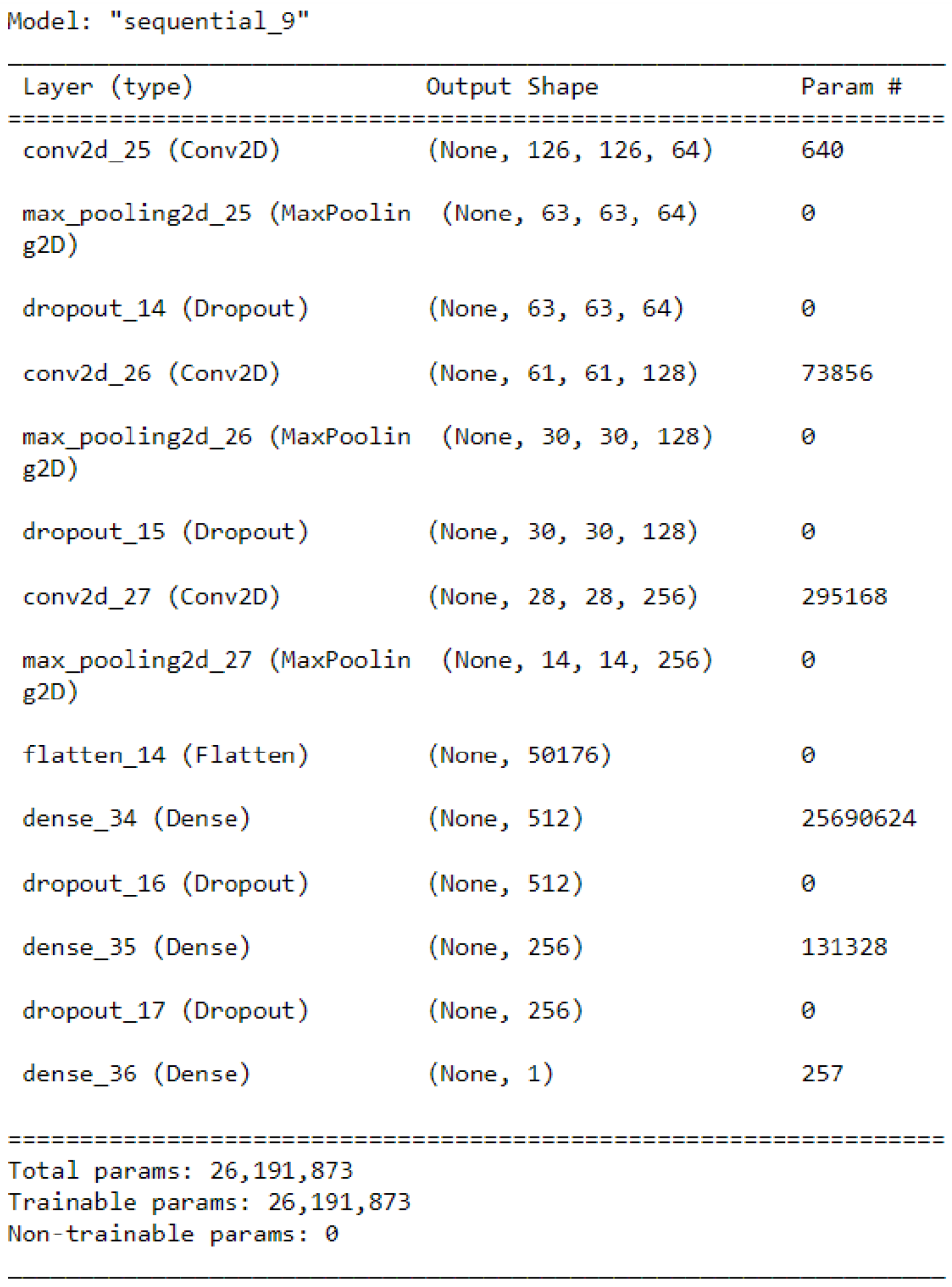
METRIC	SIMPLE NEURAL NET	HYPER PARAMETER TUNING
LOSS	27.31	1.55
TRAINING ACCURACY	45.45	63.64
VALIDATION ACCURACY	37.50	78.05

And then the model is fitted for all the 12 species in our data and the results are as follows:

METRIC	SIMPLE NEURAL NET	HYPER PARAMETER TUNING
LOSS	8.91	62.51
TRAINING ACCURACY	95.57	45.43
VALIDATION ACCURACY	21.43	38.64

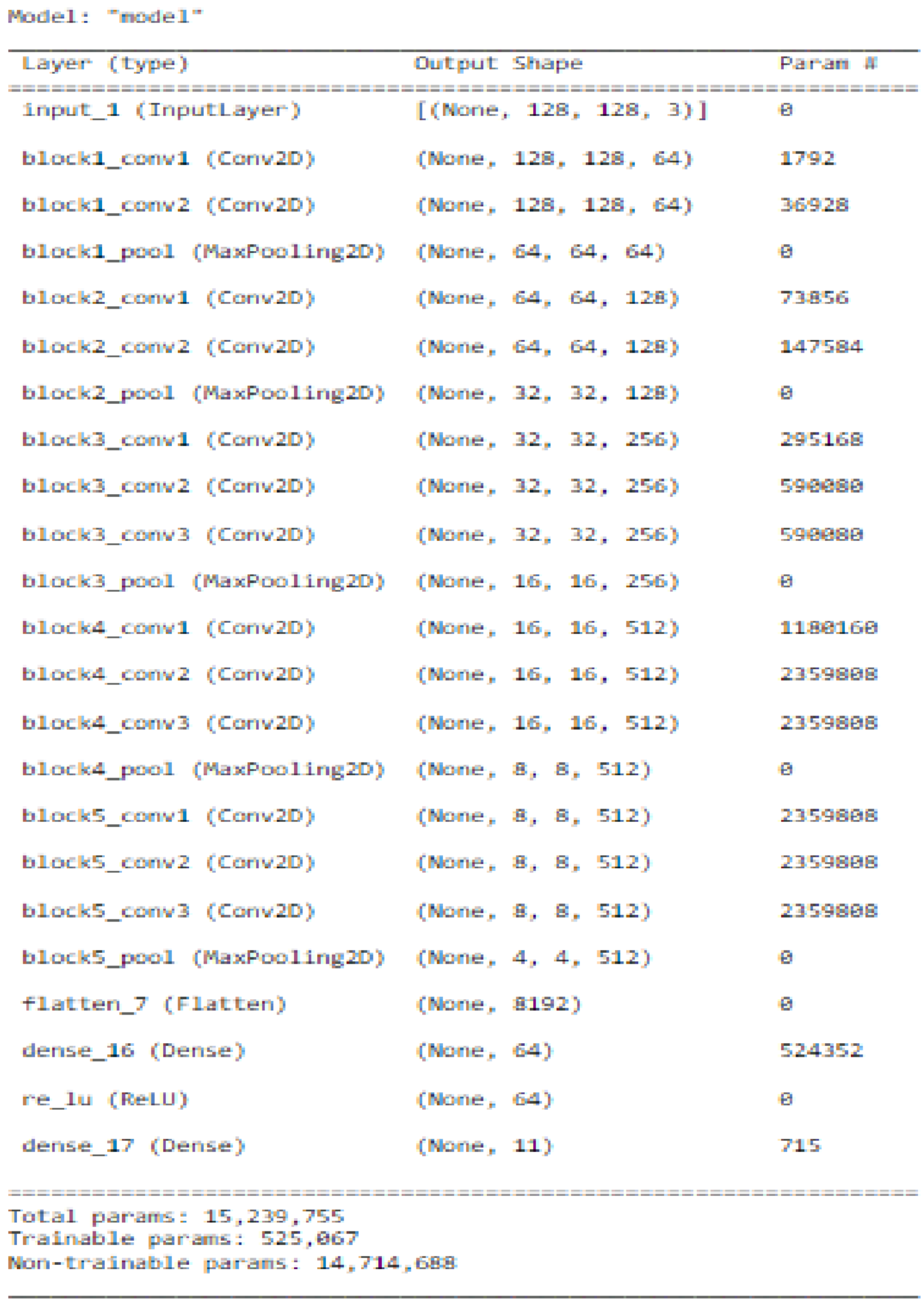
The below model architecture is a sequential convolutional neural network with six layers, including three convolutional layers, three max pooling layers, and three dropout layers. The model takes as input a 3-channel image with a shape of 126x126. The output layer has a sigmoid activation function, indicating that it is a binary classification model. The total number of trainable parameters is over 26 million.

Neural Network Architecture (2-species -- After Parameter tuning):



```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 126, 126, 64)      640

 max_pooling2d_25 (MaxPoolin  (None, 63, 63, 64)       0
 g2D)

 dropout_14 (Dropout)        (None, 63, 63, 64)        0

 conv2d_26 (Conv2D)          (None, 61, 61, 128)       73856

 max_pooling2d_26 (MaxPoolin  (None, 30, 30, 128)      0
 g2D)

 dropout_15 (Dropout)        (None, 30, 30, 128)       0

 conv2d_27 (Conv2D)          (None, 28, 28, 256)       295168

 max_pooling2d_27 (MaxPoolin  (None, 14, 14, 256)      0
 g2D)

 flatten_14 (Flatten)        (None, 50176)             0

 dense_34 (Dense)            (None, 512)               25690624

 dropout_16 (Dropout)        (None, 512)               0

 dense_35 (Dense)            (None, 256)               131328

 dropout_17 (Dropout)        (None, 256)               0

 dense_36 (Dense)            (None, 1)                 257

=================================================================
Total params: 26,191,873
Trainable params: 26,191,873
Non-trainable params: 0
_________________________________________________________________
```

Neural Network Architecture (12-species -- After Hyper-Parameter tuning):



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 flatten_7 (Flatten)         (None, 8192)              0

 dense_16 (Dense)            (None, 64)                524352

 re_lu (ReLU)                (None, 64)                0

 dense_17 (Dense)            (None, 11)                715

=================================================================
Total params: 15,239,755
Trainable params: 525,067
Non-trainable params: 14,714,688
_________________________________________________________________
```

The above given, is, a Sequential model designed for image classification. It consists of three Convolutional layers with MaxPooling layers in between. The output of these layers is flattened and fed into a Dense neural network with two hidden layers, followed by a final Dense layer with a softmax activation function that outputs the predicted probabilities foreach class.

The challenging part was detecting the among the species: 'Red-Winged Black Bird' and 'Barn Swallow'. Because their spectrograms has the similar frequency steep notes recorded at periodic gaps and also there entire no of high pitch notes and low pitch notes are the same.

We have been investigating various methods to enhance the precision of our picture classification model in our analysis. To improve the performance of our model, we chose to test transfer learning, a well-liked deep learning technique. In order to distinguish different objects and settings, we decided to use the VGG16 (imagenet) model, a pre-trained convolutional neural network (CNN) that has been trained on an extensive collection of photos.

We aimed to achieve better results than training a model from scratch by using this model as a starting point and fine-tuning it on our dataset. However, we discovered that the model was not doing as well as we had planned after training and analyzing it. With this model, we were only achieving slightly around 25% accuracy, which was below what we had anticipated. Further investigation revealed that the VGG16 model was unable to accurately classify our dataset's unique properties, which were crucial for accuracy. Surprisingly, with an accuracy of 38.86%, this fundamental CNN model produced noticeably superior results.

As a result, the model was making incorrect predictions and not performing as well as we had hoped. Hence we can proclaim, a simpler model consisting of just 2 convolutional layers and a dense layer is the best model.

MODEL	TRAINING ACCURACY	TEST ACCURACY	LOSS
VGG16 (imagenet)	31.15	27.31	2.20

In summary, this model uses Convolutional and Dense layers to extract features from images and classify them into one of eleven classes. Dropout layers are also included to prevent over fitting during training. Overall, this model has a relatively large number of trainable parameters, suggesting that it has a high capacity to learn complex features and classify images accurately.]

DISCUSSION:

In my data preprocessing section, I aim to improve the accuracy of species identification in audio recordings by utilizing various high pitches in every spectrogram and truncating each audio file to 3 seconds for every species. This approach not only increases the amount of training data available but also ensures that we have a better grip at catching any 3-second clip belonging to any particular species.

By having more training data, we can improve the accuracy of our models in identifying species in audio recordings. The use of multiple high pitches and 3-second clips ensures that our models are trained on a more diverse set of data. As a result, we can achieve better test accuracies and, ultimately, better performance in our species identification task.

Our study revealed that our model was overfitting the data because the training accuracy was low and the validation accuracy was nearly equal. We modified the model in a number of ways to overcome this problem. In order to avoid overfitting, we first increased the number of convolutional layers and dropout layers. In order to select the most effective optimizer for our data, we also tested various optimizers.

Nevertheless, despite these adjustments, we did not notice a considerable increase in the precision of our model. Because of this, we decided to augment our data by rotating, turning, and cropping the images. By performing this procedure, we have extracted a great deal of information from a single image, so more data is supplied to the model, giving model a space for improvement. This strategy gave the model more robust and diverse training data, which helped it find the underlying patterns in the data. Because of this, we noticed a noticeable rise in our model's accuracy, which went from 20% to about 40%.

And further, decided to implement transfer learning using the pre-trained VGG16 model on the ImageNet dataset. We experimented with different activation functions such as ReLU and Leaky ReLU and found that the model architecture shown in image (31/131r) performed the best among all other models trained on our data.In addition to tuning our model architecture and augmenting our data, we also utilized the early stopping technique to improve our model performance. Early stopping allows us to monitor our model's performance on the validation set during training and stop training when the validation loss stops improving. This helps prevent overfitting and allows our model to generalize better to new, unseen data.

Early stopping was implemented, by using the EarlyStopping callback provided by TensorFlow. We set the monitor parameter to 'val_loss' to monitor the validation loss during training, and set the patience parameter to 5, which means that training will stop if the validation loss does not improve for 5 consecutive epochs. By using early stopping, we were able to prevent our model from overfitting and achieve better performance on our validation set.

Overall, by combining various techniques such as data augmentation, model architecture tuning, and early stopping, the performance of our model have been significantly improved and the model have been more robust to variations in the input data.

CONCLUSION:

A binary classification model for two of the species and a multi-class classification model for all 12 species have been created, and explored transfer learning by adapting a pre- trained neural network model to their data. And also the report discusses the theoretical background of sound recognition, which involves feature extraction and classification. We used spectrograms as a popular method for feature extraction in sound recognition, and feed forward neural networks as a popular method for classification. They also discussed the hyper parameters that can be tuned to improve the performance of neuralnetworks, such as the number of hidden layers and neurons, learning rate, and regularization parameters.

In this Study, a custom neural network to classify bird species based on their sounds was been successfully built. Both binary and multi-class classification tasks were explored and been compared to transfer learning using a pre-trained neural network model. While our neural network performed reasonably well, and several limitations and challenges were identified in the process, including dataset size and over fitting. The report describes a study that aimed to build a custom neural network to classify bird sounds based on their species. And the data is, from the Birdcall competition, selecting 12 bird species and using 10 sound clips for each species.

REFERENCES:

[1] Bird Competition data: Xeno Canto Bird Recordings (Extended A-M). Retrieved from https://www.kaggle.com/datasets/rohanrao/xeno-canto-bird-recordings-extended-a-m

[2] Xeno-Canto. (n.d.). The world's largest community-driven sound library of bird songs and calls. Retrieved from https://xeno-canto.org/.

APPENDIX

In [2]:
```python
import os
import librosa
import numpy as np
from PIL import Image
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
```

Data Preprocessing and data cleaning

In [ ]:
```python
# Define the path to the directory containing the audio files
data_dir = 'C:/Users/GOKUL/OneDrive/Desktop/original_clips'

# Define the path to the directory where the spectrograms will be saved
spectrogram_dir = 'C:/Users/GOKUL/OneDrive/Desktop/spectrograms2'

# Define the parameters for the spectrogram
n_fft = 2048
hop_length = 512
n_mels = 128

# Define the pitch threshold
pitch_threshold = 1500

# Loop over each species directory
for species_dir in os.listdir(data_dir):
    species_path = os.path.join(data_dir, species_dir)
    if not os.path.isdir(species_path):
        continue

    # Get the species name from the directory name
    species_name = species_dir.split('.')[0] if '.' in species_dir else species_di

    # Create a directory for the species if it doesn't exist
    spectrogram_path = os.path.join(spectrogram_dir, species_name)
    if not os.path.exists(spectrogram_path):
        os.makedirs(spectrogram_path)

    # Loop over each audio file in the species directory
    for audio_file in os.listdir(species_path):
        audio_path = os.path.join(species_path, audio_file)
        if not audio_file.endswith('.mp3'):
            continue

        # Load the audio file and split into non-silent intervals
        y, sr = librosa.load(audio_path, sr=22050, mono=True)
        intervals = librosa.effects.split(y, top_db=20)

        # Clip each interval where the pitch is high for 3 seconds
        for interval in intervals:
            interval_y = y[interval[0]:interval[1]]
            pitches, magnitudes = librosa.core.piptrack(y=interval_y, sr=sr)
            mean_pitch = np.mean(pitches[magnitudes > np.max(magnitudes) * 0.5])
            if mean_pitch > pitch_threshold:
                start = interval[0]
                end = min(interval[1], start + sr * 3)
                clipped_y = y[start:end]
                S = librosa.feature.melspectrogram(y=clipped_y, sr=sr, n_fft=n_fft
                log_S = librosa.power_to_db(S, ref=np.max)
                img = Image.fromarray(log_S, mode='L')

                # Save the spectrogram image to the spectrogram directory
                img_path = os.path.join(spectrogram_path, os.path.splitext(audio_f
                img.save(img_path)
```

Train-Test Splitting with image of shape (128,128,1)

In [58]:
```
import os
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
from PIL import Image

# Define the path to the directory containing the spectrogram images
spectrogram_dir = 'C:/Users/GOKUL/OneDrive/Desktop/spectrograms2'

# Define the list of classes to classify
class_names = ['houfin', 'blujay']

# Define the input shape of the model
input_shape = (128, 128, 1)

# Define the batch size and number of epochs
batch_size = 32
epochs = 10

# Define a function to load the spectrogram images and corresponding labels
def load_data():
    x = []
    y = []
    for i, class_name in enumerate(class_names):
        class_path = os.path.join(spectrogram_dir, class_name)
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            img = Image.open(img_path)
            img = img.resize((input_shape[0], input_shape[1]))
            img = np.array(img.convert('L'))
            img = np.expand_dims(img, axis=-1)
            x.append(img)
            y.append(i)
    x = np.array(x)
    y = np.array(y)
    return x, y

# Load the spectrogram images and corresponding labels
x, y = load_data()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_st
```

Model building: Convolutional Neural Net(Keras architecture)

In [10]:
```
# Define the model architecture
model_two = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_two.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy

# Train the model
model_two.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_d
```

```
Epoch 1/10
1/1 [==============================] - 1s 1s/step - loss: 2.2403 - accuracy: 0.636
4 - val_loss: 417.1382 - val_accuracy: 0.6250
Epoch 2/10
1/1 [==============================] - 0s 233ms/step - loss: 762.9246 - accuracy:
0.4545 - val_loss: 175.8427 - val_accuracy: 0.6250
Epoch 3/10
1/1 [==============================] - 0s 235ms/step - loss: 317.3696 - accuracy:
0.4545 - val_loss: 93.8714 - val_accuracy: 0.3750
Epoch 4/10
1/1 [==============================] - 0s 225ms/step - loss: 50.6819 - accuracy:
0.5455 - val_loss: 80.6337 - val_accuracy: 0.3750
Epoch 5/10
1/1 [==============================] - 0s 233ms/step - loss: 49.2564 - accuracy:
0.5455 - val_loss: 47.5741 - val_accuracy: 0.3750
Epoch 6/10
1/1 [==============================] - 0s 332ms/step - loss: 29.0356 - accuracy:
0.5455 - val_loss: 15.5416 - val_accuracy: 0.3750
Epoch 7/10
1/1 [==============================] - 0s 265ms/step - loss: 9.7628 - accuracy: 0.
5455 - val_loss: 8.1634 - val_accuracy: 0.6250
Epoch 8/10
1/1 [==============================] - 0s 235ms/step - loss: 14.5117 - accuracy:
0.4545 - val_loss: 7.4877 - val_accuracy: 0.6250
Epoch 9/10
1/1 [==============================] - 0s 310ms/step - loss: 12.4257 - accuracy:
0.4545 - val_loss: 4.2490 - val_accuracy: 0.6250
Epoch 10/10
1/1 [==============================] - 0s 268ms/step - loss: 6.0874 - accuracy: 0.
4545 - val_loss: 2.5939 - val_accuracy: 0.3750
```

Out[10]:
```
<keras.callbacks.History at 0x2a44e33ff40>
```

Fine tuning our model using several optimizers and different learning rate

Adding a drop-out layer

In [12]:
```
model_two_v_01 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
```

```
model_two_v_01.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc

# Train the model
model_two_v_01.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validat
```

```
Epoch 1/10
1/1 [==============================] - 2s 2s/step - loss: 4.9000 - accuracy: 0.636
4 - val_loss: 483.0956 - val_accuracy: 0.3750
Epoch 2/10
1/1 [==============================] - 0s 261ms/step - loss: 314.9265 - accuracy:
0.5455 - val_loss: 21.1761 - val_accuracy: 0.6250
Epoch 3/10
1/1 [==============================] - 0s 311ms/step - loss: 152.0582 - accuracy:
0.7273 - val_loss: 31.6272 - val_accuracy: 0.3750
Epoch 4/10
1/1 [==============================] - 0s 271ms/step - loss: 224.3699 - accuracy:
0.0909 - val_loss: 57.4148 - val_accuracy: 0.3750
Epoch 5/10
1/1 [==============================] - 0s 250ms/step - loss: 89.9900 - accuracy:
0.5455 - val_loss: 3.7804 - val_accuracy: 0.5000
Epoch 6/10
1/1 [==============================] - 0s 266ms/step - loss: 59.6786 - accuracy:
0.6364 - val_loss: 7.4519 - val_accuracy: 0.2500
Epoch 7/10
1/1 [==============================] - 0s 249ms/step - loss: 34.3840 - accuracy:
0.5455 - val_loss: 24.8960 - val_accuracy: 0.3750
Epoch 8/10
1/1 [==============================] - 0s 274ms/step - loss: 31.9560 - accuracy:
0.7273 - val_loss: 37.1292 - val_accuracy: 0.3750
Epoch 9/10
1/1 [==============================] - 0s 258ms/step - loss: 29.2150 - accuracy:
0.4545 - val_loss: 21.1494 - val_accuracy: 0.3750
Epoch 10/10
1/1 [==============================] - 0s 279ms/step - loss: 23.0634 - accuracy:
0.4545 - val_loss: 14.3219 - val_accuracy: 0.3750
```

Out[12]:
```
<keras.callbacks.History at 0x2a44e678e20>
```

In [19]:
```
# Evaluate the model on the test set
test_loss, test_acc = model_two_v_01.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
1/1 [==============================] - 0s 59ms/step - loss: 14.3219 - accuracy: 0.
3750
Test accuracy: 0.375
```

Adding a convolutional layer

In [14]:
```
# Define the model architecture with an additional convolutional layer
model_two_v_02 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
# Compile the model with the Adam optimizer and binary crossentropy loss
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# Compile the model
model_two_v_02.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['

# Train the model
model_two_v_02.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validat
```

```
Epoch 1/10
1/1 [==============================] - 1s 1s/step - loss: 5.2269 - accuracy: 0.545
5 - val_loss: 95.1515 - val_accuracy: 0.3750
Epoch 2/10
1/1 [==============================] - 0s 243ms/step - loss: 39.0628 - accuracy:
0.5455 - val_loss: 44.3024 - val_accuracy: 0.3750
Epoch 3/10
1/1 [==============================] - 0s 244ms/step - loss: 14.4704 - accuracy:
0.6364 - val_loss: 14.0465 - val_accuracy: 0.6250
Epoch 4/10
1/1 [==============================] - 0s 243ms/step - loss: 22.2087 - accuracy:
0.4545 - val_loss: 9.6645 - val_accuracy: 0.6250
Epoch 5/10
1/1 [==============================] - 0s 250ms/step - loss: 23.5204 - accuracy:
0.2727 - val_loss: 0.7181 - val_accuracy: 0.6250
Epoch 6/10
1/1 [==============================] - 0s 248ms/step - loss: 4.0793 - accuracy: 0.
5455 - val_loss: 7.9529 - val_accuracy: 0.3750
Epoch 7/10
1/1 [==============================] - 0s 294ms/step - loss: 2.1414 - accuracy: 0.
6364 - val_loss: 5.9877 - val_accuracy: 0.3750
Epoch 8/10
1/1 [==============================] - 0s 411ms/step - loss: 2.2914 - accuracy: 0.
6364 - val_loss: 1.1076 - val_accuracy: 0.2500
Epoch 9/10
1/1 [==============================] - 0s 262ms/step - loss: 2.4053 - accuracy: 0.
7273 - val_loss: 2.8902 - val_accuracy: 0.6250
Epoch 10/10
1/1 [==============================] - 0s 246ms/step - loss: 7.1873 - accuracy: 0.
4545 - val_loss: 0.7794 - val_accuracy: 0.6250
```

Out[14]:
```
<keras.callbacks.History at 0x2a45bd6b010>
```

In [26]:
```
# Evaluate the model on the test set
test_loss, test_acc = model_two_v_02.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
1/1 [==============================] - 0s 58ms/step - loss: 0.7794 - accuracy: 0.6
250
Test accuracy: 0.625
```

Adding all three convoutionla layers, drop-out layer, with added learning rate

In [59]:
```
# Define the model architecture with additional convolutional layers and dropout
model_two_v_03 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model with the Adam optimizer and binary crossentropy loss
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)

# Compile the model
model_two_v_03.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['

# Train the model
model_two_v_03.fit(x_train, y_train, batch_size=batch_size, epochs=20, validation_
Epoch 1/20
6/6 [==============================] - 11s 2s/step - loss: 51.6647 - accuracy: 0.6
141 - val_loss: 8.2076 - val_accuracy: 0.7805
Epoch 2/20
6/6 [==============================] - 10s 2s/step - loss: 44.1906 - accuracy: 0.6
739 - val_loss: 3.1181 - val_accuracy: 0.7805
Epoch 3/20
6/6 [==============================] - 10s 2s/step - loss: 25.8422 - accuracy: 0.5
924 - val_loss: 2.0257 - val_accuracy: 0.7805
Epoch 4/20
6/6 [==============================] - 13s 2s/step - loss: 14.4911 - accuracy: 0.6
630 - val_loss: 1.3051 - val_accuracy: 0.7805
Epoch 5/20
6/6 [==============================] - 12s 2s/step - loss: 7.5635 - accuracy: 0.70
65 - val_loss: 0.5285 - val_accuracy: 0.7805
Epoch 6/20
6/6 [==============================] - 12s 2s/step - loss: 2.8649 - accuracy: 0.65
22 - val_loss: 0.6721 - val_accuracy: 0.7398
Epoch 7/20
6/6 [==============================] - 12s 2s/step - loss: 1.6381 - accuracy: 0.58
70 - val_loss: 0.6876 - val_accuracy: 0.6341
Epoch 8/20
6/6 [==============================] - 12s 2s/step - loss: 1.0334 - accuracy: 0.65
22 - val_loss: 0.6873 - val_accuracy: 0.7073
Epoch 9/20
6/6 [==============================] - 12s 2s/step - loss: 0.8643 - accuracy: 0.63
59 - val_loss: 0.6866 - val_accuracy: 0.6667
Epoch 10/20
6/6 [==============================] - 12s 2s/step - loss: 0.7187 - accuracy: 0.63
59 - val_loss: 0.6868 - val_accuracy: 0.6748
Epoch 11/20
6/6 [==============================] - 12s 2s/step - loss: 0.7847 - accuracy: 0.67
93 - val_loss: 0.6888 - val_accuracy: 0.7236
Epoch 12/20
6/6 [==============================] - 11s 2s/step - loss: 0.6351 - accuracy: 0.73
37 - val_loss: 0.6896 - val_accuracy: 0.7642
Epoch 13/20
6/6 [==============================] - 12s 2s/step - loss: 0.5824 - accuracy: 0.72
83 - val_loss: 0.6901 - val_accuracy: 0.7967
Epoch 14/20
6/6 [==============================] - 13s 2s/step - loss: 0.6852 - accuracy: 0.69
57 - val_loss: 0.6902 - val_accuracy: 0.7886
Epoch 15/20
6/6 [==============================] - 13s 2s/step - loss: 0.6850 - accuracy: 0.66
85 - val_loss: 0.6900 - val_accuracy: 0.7805
Epoch 16/20
6/6 [==============================] - 14s 2s/step - loss: 0.6356 - accuracy: 0.71
74 - val_loss: 0.6900 - val_accuracy: 0.7724
Epoch 17/20
6/6 [==============================] - 13s 2s/step - loss: 0.6256 - accuracy: 0.70
11 - val_loss: 0.6895 - val_accuracy: 0.7642
Epoch 18/20
6/6 [==============================] - 10s 2s/step - loss: 0.6428 - accuracy: 0.71
74 - val_loss: 0.6889 - val_accuracy: 0.7805
Epoch 19/20
6/6 [==============================] - 10s 2s/step - loss: 0.5932 - accuracy: 0.74
46 - val_loss: 0.6896 - val_accuracy: 0.7805
Epoch 20/20
6/6 [==============================] - 10s 2s/step - loss: 0.5841 - accuracy: 0.69
57 - val_loss: 0.6896 - val_accuracy: 0.7805
```

```
Out[59]: <keras.callbacks.History at 0x130145f9030>
```

In [60]:
```
# Evaluate the model on the test set
test_loss, test_acc = model_two_v_03.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
4/4 [==============================] - 1s 268ms/step - loss: 0.6896 - accuracy: 0.
7805
Test accuracy: 0.7804877758026123
```

In [64]:
```
model_two_v_03.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 126, 126, 64)      640

 max_pooling2d_25 (MaxPoolin  (None, 63, 63, 64)       0
 g2D)

 dropout_14 (Dropout)        (None, 63, 63, 64)        0

 conv2d_26 (Conv2D)          (None, 61, 61, 128)       73856

 max_pooling2d_26 (MaxPoolin  (None, 30, 30, 128)      0
 g2D)

 dropout_15 (Dropout)        (None, 30, 30, 128)       0

 conv2d_27 (Conv2D)          (None, 28, 28, 256)       295168

 max_pooling2d_27 (MaxPoolin  (None, 14, 14, 256)      0
 g2D)

 flatten_14 (Flatten)        (None, 50176)             0

 dense_34 (Dense)            (None, 512)               25690624

 dropout_16 (Dropout)        (None, 512)               0

 dense_35 (Dense)            (None, 256)               131328

 dropout_17 (Dropout)        (None, 256)               0

 dense_36 (Dense)            (None, 1)                 257

=================================================================
Total params: 26,191,873
Trainable params: 26,191,873
Non-trainable params: 0
_________________________________________________________________
```

Multi class classification for all 12 species ....

In [19]:
```
from ann_visualizer.visualize import ann_viz
```

In [65]:
```
# Define the path to the directory containing the spectrogram images
spectrogram_dir = 'C:/Users/GOKUL/OneDrive/Desktop/spectrograms2'

# Define the input shape of the model
```

```
input_shape = (128, 128, 1)

# Define the batch size and number of epochs
batch_size = 32
epochs = 20

# Define a function to load the spectrogram images and corresponding labels
def load_data():
    x = []
    y = []
    for i, class_name in enumerate(os.listdir(spectrogram_dir)):
        class_path = os.path.join(spectrogram_dir, class_name)
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            img = Image.open(img_path)
            img = img.resize((input_shape[0], input_shape[1]))
            img = np.array(img.convert('L'))
            img = np.expand_dims(img, axis=-1)
            x.append(img)
            y.append(i)
    x = np.array(x)
    y = tf.keras.utils.to_categorical(y, num_classes=len(os.listdir(spectrogram_di
    return x, y

# Load the spectrogram images and corresponding labels
x, y = load_data()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_st
```

In [66]:
```
# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(len(os.listdir(spectrogram_dir)), activation='softmax')
])

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

# Train the model
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=
Epoch 1/20
22/22 [==============================] - 10s 438ms/step - loss: 323.0537 - accurac
y: 0.1121 - val_loss: 2.4831 - val_accuracy: 0.1157
Epoch 2/20
22/22 [==============================] - 9s 415ms/step - loss: 2.4322 - accuracy:
0.1689 - val_loss: 2.4002 - val_accuracy: 0.2314
Epoch 3/20
22/22 [==============================] - 9s 430ms/step - loss: 1.9097 - accuracy:
0.4527 - val_loss: 2.5253 - val_accuracy: 0.1965
Epoch 4/20
22/22 [==============================] - 10s 443ms/step - loss: 0.9401 - accuracy:
0.7409 - val_loss: 3.0052 - val_accuracy: 0.1900
Epoch 5/20
22/22 [==============================] - 10s 444ms/step - loss: 0.5593 - accuracy:
0.8937 - val_loss: 3.9603 - val_accuracy: 0.2183
Epoch 6/20
22/22 [==============================] - 10s 467ms/step - loss: 0.3844 - accuracy:
0.9141 - val_loss: 4.7987 - val_accuracy: 0.2009
Epoch 7/20
22/22 [==============================] - 11s 494ms/step - loss: 0.2225 - accuracy:
0.9476 - val_loss: 5.1742 - val_accuracy: 0.2031
Epoch 8/20
22/22 [==============================] - 10s 445ms/step - loss: 0.1604 - accuracy:
0.9622 - val_loss: 5.8852 - val_accuracy: 0.2162
Epoch 9/20
22/22 [==============================] - 10s 442ms/step - loss: 0.0652 - accuracy:
0.9854 - val_loss: 6.8339 - val_accuracy: 0.2031
Epoch 10/20
22/22 [==============================] - 11s 485ms/step - loss: 0.0815 - accuracy:
0.9884 - val_loss: 9.0312 - val_accuracy: 0.2445
Epoch 11/20
22/22 [==============================] - 10s 472ms/step - loss: 0.0597 - accuracy:
0.9869 - val_loss: 6.6507 - val_accuracy: 0.2140
Epoch 12/20
22/22 [==============================] - 11s 504ms/step - loss: 0.0778 - accuracy:
0.9869 - val_loss: 6.4734 - val_accuracy: 0.2074
Epoch 13/20
22/22 [==============================] - 10s 467ms/step - loss: 0.0556 - accuracy:
0.9898 - val_loss: 9.2991 - val_accuracy: 0.1900
Epoch 14/20
22/22 [==============================] - 11s 505ms/step - loss: 0.0830 - accuracy:
0.9825 - val_loss: 6.6464 - val_accuracy: 0.1965
Epoch 15/20
22/22 [==============================] - 11s 499ms/step - loss: 0.0308 - accuracy:
0.9956 - val_loss: 6.5754 - val_accuracy: 0.1878
Epoch 16/20
22/22 [==============================] - 10s 447ms/step - loss: 0.0126 - accuracy:
0.9985 - val_loss: 6.4442 - val_accuracy: 0.2052
Epoch 17/20
22/22 [==============================] - 10s 444ms/step - loss: 0.0090 - accuracy:
1.0000 - val_loss: 9.1298 - val_accuracy: 0.2227
Epoch 18/20
22/22 [==============================] - 10s 442ms/step - loss: 0.0024 - accuracy:
1.0000 - val_loss: 8.0829 - val_accuracy: 0.2074
Epoch 19/20
22/22 [==============================] - 10s 443ms/step - loss: 0.0015 - accuracy:
1.0000 - val_loss: 8.5990 - val_accuracy: 0.2118
Epoch 20/20
22/22 [==============================] - 10s 443ms/step - loss: 8.4932e-04 - accur
acy: 1.0000 - val_loss: 8.9125 - val_accuracy: 0.2140
```

```
Out[66]: <keras.callbacks.History at 0x13018133220>
```

In [67]:
```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
15/15 [==============================] - 1s 78ms/step - loss: 8.9125 - accuracy:
0.2140
Test accuracy: 0.21397380530834198
```

In [68]:
```
model.summary()
```

```
Model: "sequential_10"

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 126, 126, 32)      320

 max_pooling2d_28 (MaxPoolin  (None, 63, 63, 32)       0
 g2D)

 conv2d_29 (Conv2D)          (None, 61, 61, 64)        18496

 max_pooling2d_29 (MaxPoolin  (None, 30, 30, 64)       0
 g2D)

 flatten_15 (Flatten)        (None, 57600)             0

 dense_37 (Dense)            (None, 64)                3686464

 dense_38 (Dense)            (None, 12)                780

=================================================================
Total params: 3,706,060
Trainable params: 3,706,060
Non-trainable params: 0
_________________________________________________________________
```

In [69]:
```
# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(len(os.listdir(spectrogram_dir)), activation='softmax')
])

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

# Train the model
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=
Epoch 1/20
22/22 [==============================] - 12s 498ms/step - loss: 43.0082 - accurac
y: 0.1106 - val_loss: 2.4821 - val_accuracy: 0.2031
Epoch 2/20
22/22 [==============================] - 11s 488ms/step - loss: 2.4803 - accuracy:
0.1921 - val_loss: 2.4769 - val_accuracy: 0.2031
Epoch 3/20
22/22 [==============================] - 11s 518ms/step - loss: 2.4752 - accuracy:
0.1921 - val_loss: 2.4710 - val_accuracy: 0.2031
Epoch 4/20
22/22 [==============================] - 12s 529ms/step - loss: 2.4699 - accuracy:
0.1921 - val_loss: 2.4650 - val_accuracy: 0.2031
Epoch 5/20
22/22 [==============================] - 12s 537ms/step - loss: 2.4644 - accuracy:
0.1921 - val_loss: 2.4588 - val_accuracy: 0.2031
Epoch 6/20
22/22 [==============================] - 12s 542ms/step - loss: 2.4588 - accuracy:
0.1921 - val_loss: 2.4529 - val_accuracy: 0.2031
Epoch 7/20
22/22 [==============================] - 13s 573ms/step - loss: 2.4534 - accuracy:
0.1921 - val_loss: 2.4468 - val_accuracy: 0.2031
Epoch 8/20
22/22 [==============================] - 12s 552ms/step - loss: 2.4479 - accuracy:
0.1921 - val_loss: 2.4408 - val_accuracy: 0.2031
Epoch 9/20
22/22 [==============================] - 12s 547ms/step - loss: 2.4427 - accuracy:
0.1921 - val_loss: 2.4348 - val_accuracy: 0.2031
Epoch 10/20
22/22 [==============================] - 12s 563ms/step - loss: 2.4374 - accuracy:
0.1921 - val_loss: 2.4293 - val_accuracy: 0.2031
Epoch 11/20
22/22 [==============================] - 13s 584ms/step - loss: 2.4323 - accuracy:
0.1921 - val_loss: 2.4238 - val_accuracy: 0.2031
Epoch 12/20
22/22 [==============================] - 12s 544ms/step - loss: 2.4275 - accuracy:
0.1921 - val_loss: 2.4182 - val_accuracy: 0.2031
Epoch 13/20
22/22 [==============================] - 12s 532ms/step - loss: 2.4225 - accuracy:
0.1921 - val_loss: 2.4131 - val_accuracy: 0.2031
Epoch 14/20
22/22 [==============================] - 12s 545ms/step - loss: 2.4180 - accuracy:
0.1921 - val_loss: 2.4080 - val_accuracy: 0.2031
Epoch 15/20
22/22 [==============================] - 15s 683ms/step - loss: 2.4136 - accuracy:
0.1921 - val_loss: 2.4031 - val_accuracy: 0.2031
Epoch 16/20
22/22 [==============================] - 12s 536ms/step - loss: 2.4093 - accuracy:
0.1921 - val_loss: 2.3981 - val_accuracy: 0.2031
Epoch 17/20
22/22 [==============================] - 15s 695ms/step - loss: 2.4049 - accuracy:
0.1921 - val_loss: 2.3935 - val_accuracy: 0.2031
Epoch 18/20
22/22 [==============================] - 15s 689ms/step - loss: 2.4009 - accuracy:
0.1921 - val_loss: 2.3888 - val_accuracy: 0.2031
Epoch 19/20
22/22 [==============================] - 12s 552ms/step - loss: 2.3969 - accuracy:
0.1921 - val_loss: 2.3844 - val_accuracy: 0.2031
Epoch 20/20
22/22 [==============================] - 12s 544ms/step - loss: 2.3932 - accuracy:
0.1921 - val_loss: 2.3802 - val_accuracy: 0.2031
```

```
Out[69]: <keras.callbacks.History at 0x13014370610>
```

As we can see the validation accuracy is higher than training accuracy so next up we will try to add drop out layer and then access the model.

In [70]:
```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
15/15 [==============================] - 1s 98ms/step - loss: 2.3802 - accuracy:
0.2031
Test accuracy: 0.20305676758289337
```

In [31]:
```
model.summary()
```

```
Model: "sequential_1"

 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 126, 126, 32)      320

 max_pooling2d_2 (MaxPooling  (None, 63, 63, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 flatten_1 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 64)                1605696

 dense_3 (Dense)             (None, 11)                715

=================================================================
Total params: 1,699,083
Trainable params: 1,699,083
Non-trainable params: 0
_________________________________________________________________
```

In [71]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(os.listdir(spectrogram_dir)), activation='softmax')
])
```

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

# Train the model
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=
Epoch 1/20
22/22 [==============================] - 12s 503ms/step - loss: 15.2808 - accurac
y: 0.1412 - val_loss: 2.4802 - val_accuracy: 0.2227
Epoch 2/20
22/22 [==============================] - 11s 517ms/step - loss: 2.4775 - accuracy:
0.1980 - val_loss: 2.4736 - val_accuracy: 0.2227
Epoch 3/20
22/22 [==============================] - 12s 530ms/step - loss: 2.4713 - accuracy:
0.1980 - val_loss: 2.4666 - val_accuracy: 0.2227
Epoch 4/20
22/22 [==============================] - 12s 528ms/step - loss: 2.4649 - accuracy:
0.1980 - val_loss: 2.4597 - val_accuracy: 0.2227
Epoch 5/20
22/22 [==============================] - 12s 537ms/step - loss: 2.4588 - accuracy:
0.1980 - val_loss: 2.4527 - val_accuracy: 0.2227
Epoch 6/20
22/22 [==============================] - 12s 539ms/step - loss: 2.4527 - accuracy:
0.1980 - val_loss: 2.4459 - val_accuracy: 0.2227
Epoch 7/20
22/22 [==============================] - 12s 561ms/step - loss: 2.4468 - accuracy:
0.1980 - val_loss: 2.4391 - val_accuracy: 0.2227
Epoch 8/20
22/22 [==============================] - 13s 612ms/step - loss: 2.4412 - accuracy:
0.1980 - val_loss: 2.4327 - val_accuracy: 0.2227
Epoch 9/20
22/22 [==============================] - 13s 577ms/step - loss: 2.4354 - accuracy:
0.1980 - val_loss: 2.4269 - val_accuracy: 0.2227
Epoch 10/20
22/22 [==============================] - 12s 544ms/step - loss: 2.4301 - accuracy:
0.1980 - val_loss: 2.4207 - val_accuracy: 0.2227
Epoch 11/20
22/22 [==============================] - 12s 546ms/step - loss: 2.4249 - accuracy:
0.1980 - val_loss: 2.4150 - val_accuracy: 0.2227
Epoch 12/20
22/22 [==============================] - 12s 544ms/step - loss: 2.4199 - accuracy:
0.1980 - val_loss: 2.4092 - val_accuracy: 0.2227
Epoch 13/20
22/22 [==============================] - 12s 531ms/step - loss: 2.4150 - accuracy:
0.1980 - val_loss: 2.4041 - val_accuracy: 0.2227
Epoch 14/20
22/22 [==============================] - 12s 544ms/step - loss: 2.4105 - accuracy:
0.1980 - val_loss: 2.3989 - val_accuracy: 0.2227
Epoch 15/20
22/22 [==============================] - 12s 531ms/step - loss: 2.4059 - accuracy:
0.1980 - val_loss: 2.3940 - val_accuracy: 0.2227
Epoch 16/20
22/22 [==============================] - 12s 542ms/step - loss: 2.4016 - accuracy:
0.1980 - val_loss: 2.3889 - val_accuracy: 0.2227
Epoch 17/20
22/22 [==============================] - 12s 546ms/step - loss: 2.3975 - accuracy:
0.1980 - val_loss: 2.3842 - val_accuracy: 0.2227
Epoch 18/20
22/22 [==============================] - 12s 531ms/step - loss: 2.3935 - accuracy:
0.1980 - val_loss: 2.3801 - val_accuracy: 0.2227
Epoch 19/20
22/22 [==============================] - 12s 542ms/step - loss: 2.3899 - accuracy:
0.1980 - val_loss: 2.3758 - val_accuracy: 0.2227
Epoch 20/20
22/22 [==============================] - 12s 555ms/step - loss: 2.3863 - accuracy:
0.1980 - val_loss: 2.3717 - val_accuracy: 0.2227
```

```
Out[71]: <keras.callbacks.History at 0x1301d12c850>
```

In [72]:
```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
15/15 [==============================] - 2s 112ms/step - loss: 2.3717 - accuracy:
0.2227
Test accuracy: 0.22270742058753967
```

As there is not significant improvement in our model by tuning the hyper parameters, We will now do data augmentation thta is rotating the image or resizing the image and so by applying these transformations the model will train better and learn robust variations in the data.so, we can get beter validation accuracy!!

In [73]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the data generator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

# Define the data generator for the validation/test set
test_datagen = ImageDataGenerator(rescale=1./255)

# Define the data iterators for the training and validation/test sets
train_generator = train_datagen.flow(x_train, y_train, batch_size=batch_size)
test_generator = test_datagen.flow(x_test, y_test, batch_size=batch_size)
```

In [74]:
```
# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(len(os.listdir(spectrogram_dir)), activation='softmax')
])

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

# Train the model with data augmentation
model.fit(train_generator,
          steps_per_epoch=len(x_train)//batch_size,
          epochs=epochs,
          validation_data=test_generator,
          validation_steps=len(x_test)//batch_size)
```

```
Epoch 1/20
21/21 [==============================] - 10s 439ms/step - loss: 2.6485 - accuracy:
0.1817 - val_loss: 2.3771 - val_accuracy: 0.2232
Epoch 2/20
21/21 [==============================] - 10s 466ms/step - loss: 2.3683 - accuracy:
0.2046 - val_loss: 2.3425 - val_accuracy: 0.2299
Epoch 3/20
21/21 [==============================] - 10s 463ms/step - loss: 2.3321 - accuracy:
0.2031 - val_loss: 2.2870 - val_accuracy: 0.3058
Epoch 4/20
21/21 [==============================] - 10s 459ms/step - loss: 2.2662 - accuracy:
0.2595 - val_loss: 2.4429 - val_accuracy: 0.2031
Epoch 5/20
21/21 [==============================] - 10s 481ms/step - loss: 2.2847 - accuracy:
0.2458 - val_loss: 2.5117 - val_accuracy: 0.2388
Epoch 6/20
21/21 [==============================] - 10s 457ms/step - loss: 2.2397 - accuracy:
0.2748 - val_loss: 2.5561 - val_accuracy: 0.1607
Epoch 7/20
21/21 [==============================] - 11s 536ms/step - loss: 2.1901 - accuracy:
0.3008 - val_loss: 2.4570 - val_accuracy: 0.2076
Epoch 8/20
21/21 [==============================] - 11s 532ms/step - loss: 2.1426 - accuracy:
0.2901 - val_loss: 2.3455 - val_accuracy: 0.2478
Epoch 9/20
21/21 [==============================] - 12s 548ms/step - loss: 2.1110 - accuracy:
0.2976 - val_loss: 2.8132 - val_accuracy: 0.1786
Epoch 10/20
21/21 [==============================] - 13s 638ms/step - loss: 2.1259 - accuracy:
0.3023 - val_loss: 3.0366 - val_accuracy: 0.1696
Epoch 11/20
21/21 [==============================] - 13s 622ms/step - loss: 2.0835 - accuracy:
0.3176 - val_loss: 3.2044 - val_accuracy: 0.2031
Epoch 12/20
21/21 [==============================] - 12s 598ms/step - loss: 2.0577 - accuracy:
0.3344 - val_loss: 2.9154 - val_accuracy: 0.2076
Epoch 13/20
21/21 [==============================] - 11s 504ms/step - loss: 2.0379 - accuracy:
0.3252 - val_loss: 2.3663 - val_accuracy: 0.3013
Epoch 14/20
21/21 [==============================] - 10s 465ms/step - loss: 2.0426 - accuracy:
0.3160 - val_loss: 2.5678 - val_accuracy: 0.2433
Epoch 15/20
21/21 [==============================] - 10s 467ms/step - loss: 2.0149 - accuracy:
0.3405 - val_loss: 3.0167 - val_accuracy: 0.2254
Epoch 16/20
21/21 [==============================] - 11s 505ms/step - loss: 1.9937 - accuracy:
0.3481 - val_loss: 3.1409 - val_accuracy: 0.2054
Epoch 17/20
21/21 [==============================] - 11s 514ms/step - loss: 1.9821 - accuracy:
0.3405 - val_loss: 2.9850 - val_accuracy: 0.2433
Epoch 18/20
21/21 [==============================] - 10s 480ms/step - loss: 1.9919 - accuracy:
0.3542 - val_loss: 2.5265 - val_accuracy: 0.2701
Epoch 19/20
21/21 [==============================] - 10s 496ms/step - loss: 1.9674 - accuracy:
0.3603 - val_loss: 3.0700 - val_accuracy: 0.2366
Epoch 20/20
21/21 [==============================] - 10s 487ms/step - loss: 1.8766 - accuracy:
0.3954 - val_loss: 2.4399 - val_accuracy: 0.3549
```

```
Out[74]: <keras.callbacks.History at 0x1301cdfe380>
```

In [75]:
```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
15/15 [==============================] - 1s 83ms/step - loss: 271.4424 - accuracy:
0.3865
Test accuracy: 0.38646286725997925
```

Now lets try to improve the accuracy

In [76]:
```
from tensorflow.keras.layers import Dropout

# Define the model architecture
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape)
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    Dropout(0.5),
    tf.keras.layers.Dense(len(os.listdir(spectrogram_dir)), activation='softmax')
])

# Define the optimizer with a learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# Compile the model with the optimizer
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accu

# Train the model with data augmentation
model.fit(train_generator,
          steps_per_epoch=len(x_train)//batch_size,
          epochs=epochs,
          validation_data=test_generator,
          validation_steps=len(x_test)//batch_size)
```

```
Epoch 1/20
21/21 [==============================] - 19s 704ms/step - loss: 2.4435 - accuracy:
0.1634 - val_loss: 2.3309 - val_accuracy: 0.2054
Epoch 2/20
21/21 [==============================] - 14s 644ms/step - loss: 2.4044 - accuracy:
0.1985 - val_loss: 2.3098 - val_accuracy: 0.2344
Epoch 3/20
21/21 [==============================] - 14s 642ms/step - loss: 2.3585 - accuracy:
0.1924 - val_loss: 2.3034 - val_accuracy: 0.2366
Epoch 4/20
21/21 [==============================] - 14s 673ms/step - loss: 2.3380 - accuracy:
0.1878 - val_loss: 2.3129 - val_accuracy: 0.3147
Epoch 5/20
21/21 [==============================] - 13s 640ms/step - loss: 2.3253 - accuracy:
0.2031 - val_loss: 2.3269 - val_accuracy: 0.2321
Epoch 6/20
21/21 [==============================] - 14s 646ms/step - loss: 2.3196 - accuracy:
0.2031 - val_loss: 2.2556 - val_accuracy: 0.2522
Epoch 7/20
21/21 [==============================] - 13s 638ms/step - loss: 2.3097 - accuracy:
0.2244 - val_loss: 2.2337 - val_accuracy: 0.3415
Epoch 8/20
21/21 [==============================] - 14s 661ms/step - loss: 2.2862 - accuracy:
0.2247 - val_loss: 2.4083 - val_accuracy: 0.1071
Epoch 9/20
21/21 [==============================] - 14s 647ms/step - loss: 2.2761 - accuracy:
0.2351 - val_loss: 2.2746 - val_accuracy: 0.2701
Epoch 10/20
21/21 [==============================] - 14s 669ms/step - loss: 2.2572 - accuracy:
0.2336 - val_loss: 2.1417 - val_accuracy: 0.3527
Epoch 11/20
21/21 [==============================] - 14s 670ms/step - loss: 2.2552 - accuracy:
0.2504 - val_loss: 2.3473 - val_accuracy: 0.1964
Epoch 12/20
21/21 [==============================] - 14s 664ms/step - loss: 2.2251 - accuracy:
0.2595 - val_loss: 2.3190 - val_accuracy: 0.2188
Epoch 13/20
21/21 [==============================] - 14s 644ms/step - loss: 2.2417 - accuracy:
0.2550 - val_loss: 2.5073 - val_accuracy: 0.1317
Epoch 14/20
21/21 [==============================] - 14s 662ms/step - loss: 2.2250 - accuracy:
0.2748 - val_loss: 2.2566 - val_accuracy: 0.1897
Epoch 15/20
21/21 [==============================] - 14s 666ms/step - loss: 2.1678 - accuracy:
0.2932 - val_loss: 2.6301 - val_accuracy: 0.1451
Epoch 16/20
21/21 [==============================] - 14s 654ms/step - loss: 2.1481 - accuracy:
0.3221 - val_loss: 2.6805 - val_accuracy: 0.1585
Epoch 17/20
21/21 [==============================] - 14s 667ms/step - loss: 2.1718 - accuracy:
0.3115 - val_loss: 2.3835 - val_accuracy: 0.1875
Epoch 18/20
21/21 [==============================] - 14s 647ms/step - loss: 2.1729 - accuracy:
0.2840 - val_loss: 3.3241 - val_accuracy: 0.1607
Epoch 19/20
21/21 [==============================] - 14s 646ms/step - loss: 2.1472 - accuracy:
0.3206 - val_loss: 2.7906 - val_accuracy: 0.1808
Epoch 20/20
21/21 [==============================] - 13s 641ms/step - loss: 2.1461 - accuracy:
0.3099 - val_loss: 2.6329 - val_accuracy: 0.1853
```

```
Out[76]: <keras.callbacks.History at 0x130206290f0>
```

In [77]:
```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)


# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
15/15 [==============================] - 2s 122ms/step - loss: 303.8218 - accurac
y: 0.1703
Test accuracy: 0.17030568420886993
```

In [43]:
```
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_16 (Conv2D)          (None, 126, 126, 32)      320       
                                                                 
 max_pooling2d_16 (MaxPoolin  (None, 63, 63, 32)       0         
 g2D)                                                            
                                                                 
 conv2d_17 (Conv2D)          (None, 61, 61, 64)        18496     
                                                                 
 max_pooling2d_17 (MaxPoolin  (None, 30, 30, 64)       0         
 g2D)                                                            
                                                                 
 conv2d_18 (Conv2D)          (None, 28, 28, 128)       73856     
                                                                 
 max_pooling2d_18 (MaxPoolin  (None, 14, 14, 128)      0         
 g2D)                                                            
                                                                 
 flatten_6 (Flatten)         (None, 25088)             0         
                                                                 
 dense_13 (Dense)            (None, 256)               6422784   
                                                                 
 dropout_4 (Dropout)         (None, 256)               0         
                                                                 
 dense_14 (Dense)            (None, 128)               32896     
                                                                 
 dropout_5 (Dropout)         (None, 128)               0         
                                                                 
 dense_15 (Dense)            (None, 11)                1419      
                                                                 
=================================================================
Total params: 6,549,771
Trainable params: 6,549,771
Non-trainable params: 0
_________________________________________________________________
```

Transfer Learning

In [3]:
```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, ReLU
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# Define the path to the directory containing the spectrogram images
spectrogram_dir = 'C:/Users/GOKUL/OneDrive/Desktop/spectrograms2'

# Define the input shape of the model
input_shape = (128, 128, 3)
```

```
# Define the batch size and number of epochs
batch_size = 32
epochs = 20

# Define a function to load the spectrogram images and corresponding labels
def load_data():
    x = []
    y = []
    for i, class_name in enumerate(os.listdir(spectrogram_dir)):
        class_path = os.path.join(spectrogram_dir, class_name)
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            img = Image.open(img_path)
            img = img.resize((input_shape[0], input_shape[1]))
            img = np.array(img.convert('L'))
            img = np.stack((img,) * 3, axis=-1) # Convert to 3 channels
            x.append(img)
            y.append(i)
    x = np.array(x)
    y = tf.keras.utils.to_categorical(y, num_classes=len(os.listdir(spectrogram_di
    return x, y

# Load the spectrogram images and corresponding labels
x, y = load_data()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_st
```

In [4]:
```
# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

# Add new output layers to the pre-trained model
x = base_model.output
x = Flatten()(x)
x = Dense(64)(x)
x = ReLU()(x) # Use ReLU activation function
predictions = Dense(len(os.listdir(spectrogram_dir)), activation='softmax')(x)

# Create the new model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the weights of the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

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

# Create the data generator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True
)

# Fit the data generator to the training data
train_datagen.fit(x_train)
```

In [5]:
```
# Train the model with data augmentation
model.fit(train_datagen.flow(x_train, y_train, batch_size=batch_size),
          steps_per_epoch=len(x_train)//batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)

# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
Epoch 1/20
21/21 [==============================] - 86s 4s/step - loss: 2.4098 - accuracy: 0.
2031 - val_loss: 12.6131 - val_accuracy: 0.1026
Epoch 2/20
21/21 [==============================] - 84s 4s/step - loss: 2.2446 - accuracy: 0.
2336 - val_loss: 14.1127 - val_accuracy: 0.1463
Epoch 3/20
21/21 [==============================] - 85s 4s/step - loss: 2.2062 - accuracy: 0.
2656 - val_loss: 15.0668 - val_accuracy: 0.1266
Epoch 4/20
21/21 [==============================] - 86s 4s/step - loss: 2.1828 - accuracy: 0.
2595 - val_loss: 15.4631 - val_accuracy: 0.1550
Epoch 5/20
21/21 [==============================] - 86s 4s/step - loss: 2.1521 - accuracy: 0.
2702 - val_loss: 20.5614 - val_accuracy: 0.1288
Epoch 6/20
21/21 [==============================] - 87s 4s/step - loss: 2.1312 - accuracy: 0.
2595 - val_loss: 16.7931 - val_accuracy: 0.1463
Epoch 7/20
21/21 [==============================] - 86s 4s/step - loss: 2.1041 - accuracy: 0.
2580 - val_loss: 20.8332 - val_accuracy: 0.1179
Epoch 8/20
21/21 [==============================] - 86s 4s/step - loss: 2.0967 - accuracy: 0.
2519 - val_loss: 22.4251 - val_accuracy: 0.1157
Epoch 9/20
21/21 [==============================] - 85s 4s/step - loss: 2.0857 - accuracy: 0.
2794 - val_loss: 25.6802 - val_accuracy: 0.1245
Epoch 10/20
21/21 [==============================] - 86s 4s/step - loss: 2.0562 - accuracy: 0.
3008 - val_loss: 36.5054 - val_accuracy: 0.0939
Epoch 11/20
21/21 [==============================] - 87s 4s/step - loss: 2.0754 - accuracy: 0.
2901 - val_loss: 26.7220 - val_accuracy: 0.1485
Epoch 12/20
21/21 [==============================] - 85s 4s/step - loss: 2.0331 - accuracy: 0.
2855 - val_loss: 29.5355 - val_accuracy: 0.1441
Epoch 13/20
21/21 [==============================] - 85s 4s/step - loss: 2.0619 - accuracy: 0.
2916 - val_loss: 27.6878 - val_accuracy: 0.1790
Epoch 14/20
21/21 [==============================] - 83s 4s/step - loss: 2.0279 - accuracy: 0.
3069 - val_loss: 29.9908 - val_accuracy: 0.1485
Epoch 15/20
21/21 [==============================] - 79s 4s/step - loss: 2.0518 - accuracy: 0.
2931 - val_loss: 30.3596 - val_accuracy: 0.1550
Epoch 16/20
21/21 [==============================] - 74s 4s/step - loss: 2.0362 - accuracy: 0.
2702 - val_loss: 34.9549 - val_accuracy: 0.1397
Epoch 17/20
21/21 [==============================] - 81s 4s/step - loss: 2.0164 - accuracy: 0.
3008 - val_loss: 33.7288 - val_accuracy: 0.1332
Epoch 18/20
21/21 [==============================] - 77s 4s/step - loss: 2.0111 - accuracy: 0.
3145 - val_loss: 35.3629 - val_accuracy: 0.1310
Epoch 19/20
21/21 [==============================] - 73s 4s/step - loss: 2.0566 - accuracy: 0.
2916 - val_loss: 38.8872 - val_accuracy: 0.1070
Epoch 20/20
21/21 [==============================] - 76s 4s/step - loss: 2.0261 - accuracy: 0.
2962 - val_loss: 40.3995 - val_accuracy: 0.1201
15/15 [==============================] - 30s 2s/step - loss: 40.3995 - accuracy:
0.1201
Test accuracy: 0.12008733302354813
```

```
In [6]: model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0         
                                                                 
 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0         
                                                                 
 flatten (Flatten)           (None, 8192)              0         
                                                                 
 dense (Dense)               (None, 64)                524352    
                                                                 
 re_lu (ReLU)                (None, 64)                0         
                                                                 
 dense_1 (Dense)             (None, 12)                780       
                                                                 
=================================================================
Total params: 15,239,820
Trainable params: 525,132
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
In [8]: from tensorflow.keras.applications.vgg16 import VGG16
        from tensorflow.keras.models import Model
        from tensorflow.keras.layers import Dense, Flatten, LeakyReLU
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```python
# Define the path to the directory containing the spectrogram images
spectrogram_dir = 'C:/Users/GOKUL/OneDrive/Desktop/spectrograms2'

# Define the input shape of the model
input_shape = (128, 128, 3)
# Define the batch size and number of epochs
batch_size = 32
epochs = 20

# Define a function to load the spectrogram images and corresponding labels
def load_data():
    x = []
    y = []
    for i, class_name in enumerate(os.listdir(spectrogram_dir)):
        class_path = os.path.join(spectrogram_dir, class_name)
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            img = Image.open(img_path)
            img = img.resize((input_shape[0], input_shape[1]))
            img = np.array(img.convert('L'))
            img = np.stack((img,) * 3, axis=-1) # Convert to 3 channels
            x.append(img)
            y.append(i)
    x = np.array(x)
    y = tf.keras.utils.to_categorical(y, num_classes=len(os.listdir(spectrogram_di
    return x, y

# Load the spectrogram images and corresponding labels
x, y = load_data()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_st

# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

# Add new output layers to the pre-trained model
x = base_model.output
x = Flatten()(x)
x = Dense(64)(x)
x = LeakyReLU(alpha=0.1)(x) # Use LeakyReLU activation function
predictions = Dense(len(os.listdir(spectrogram_dir)), activation='softmax')(x)

# Create the new model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the weights of the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

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

# Create the data generator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True
)
```

```python
# Fit the data generator to the training data
train_datagen.fit(x_train)

# Train the model with data augmentation
model.fit(train_datagen.flow(x_train,  y_train,  batch_size=batch_size),
          steps_per_epoch=len(x_train)//batch_size,
          epochs=epochs,
          validation_data=(x_test,  y_test))

# Evaluate the model on the test set
test_loss, test_acc  =  model.evaluate(x_test, y_test)

# Print the test accuracy
print()
print('Test accuracy:', test_acc)
```

```
Epoch 1/20
21/21 [==============================] - 67s 3s/step - loss: 2.4390 - accuracy: 0.
2015 - val_loss: 16.9476 - val_accuracy: 0.2293
Epoch 2/20
21/21 [==============================] - 75s 4s/step - loss: 2.2109 - accuracy: 0.
2565 - val_loss: 24.7844 - val_accuracy: 0.1878
Epoch 3/20
21/21 [==============================] - 72s 4s/step - loss: 2.2202 - accuracy: 0.
2305 - val_loss: 25.6214 - val_accuracy: 0.1921
Epoch 4/20
21/21 [==============================] - 73s 4s/step - loss: 2.1601 - accuracy: 0.
2702 - val_loss: 31.7762 - val_accuracy: 0.2031
Epoch 5/20
21/21 [==============================] - 72s 4s/step - loss: 2.1545 - accuracy: 0.
2733 - val_loss: 26.2394 - val_accuracy: 0.2096
Epoch 6/20
21/21 [==============================] - 76s 4s/step - loss: 2.1013 - accuracy: 0.
2748 - val_loss: 31.4533 - val_accuracy: 0.2031
Epoch 7/20
21/21 [==============================] - 77s 4s/step - loss: 2.0537 - accuracy: 0.
2687 - val_loss: 36.6595 - val_accuracy: 0.1943
Epoch 8/20
21/21 [==============================] - 74s 4s/step - loss: 2.0761 - accuracy: 0.
2870 - val_loss: 32.4574 - val_accuracy: 0.2118
Epoch 9/20
21/21 [==============================] - 73s 4s/step - loss: 2.1129 - accuracy: 0.
2702 - val_loss: 33.7099 - val_accuracy: 0.1900
Epoch 10/20
21/21 [==============================] - 75s 4s/step - loss: 2.0804 - accuracy: 0.
2748 - val_loss: 34.6314 - val_accuracy: 0.1638
Epoch 11/20
21/21 [==============================] - 73s 4s/step - loss: 2.0465 - accuracy: 0.
2733 - val_loss: 34.8479 - val_accuracy: 0.1681
Epoch 12/20
21/21 [==============================] - 79s 4s/step - loss: 2.0186 - accuracy: 0.
2794 - val_loss: 42.6757 - val_accuracy: 0.1507
Epoch 13/20
21/21 [==============================] - 73s 4s/step - loss: 1.9980 - accuracy: 0.
3099 - val_loss: 36.3115 - val_accuracy: 0.1681
Epoch 14/20
21/21 [==============================] - 73s 4s/step - loss: 2.0241 - accuracy: 0.
2916 - val_loss: 42.2901 - val_accuracy: 0.1725
Epoch 15/20
21/21 [==============================] - 73s 4s/step - loss: 2.0042 - accuracy: 0.
2931 - val_loss: 45.5757 - val_accuracy: 0.1878
Epoch 16/20
21/21 [==============================] - 75s 4s/step - loss: 2.0270 - accuracy: 0.
2687 - val_loss: 45.6160 - val_accuracy: 0.1572
Epoch 17/20
21/21 [==============================] - 73s 4s/step - loss: 2.0031 - accuracy: 0.
3115 - val_loss: 44.8368 - val_accuracy: 0.1747
Epoch 18/20
21/21 [==============================] - 73s 4s/step - loss: 1.9524 - accuracy: 0.
3282 - val_loss: 50.5928 - val_accuracy: 0.1834
Epoch 19/20
21/21 [==============================] - 73s 4s/step - loss: 1.9484 - accuracy: 0.
3206 - val_loss: 51.7384 - val_accuracy: 0.1507
Epoch 20/20
21/21 [==============================] - 73s 4s/step - loss: 1.9912 - accuracy: 0.
2931 - val_loss: 51.2039 - val_accuracy: 0.1594
15/15 [==============================] - 31s 2s/step - loss: 51.2039 - accuracy:
0.1594

Test accuracy: 0.15938864648342133
```

In [10]:
```
import os
import numpy as np
import tensorflow as tf
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten, Dropout, LeakyReLU
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import EarlyStopping

# Define the path to the directory containing the spectrogram images
spectrogram_dir = 'C:/Users/GOKUL/OneDrive/Desktop/spectrograms'

# Define the input shape of the model
input_shape = (128, 128, 3)

# Define the batch size and number of epochs
batch_size = 32
epochs = 50
early_stop = EarlyStopping(monitor='val_loss', patience=3, verbose=1, mode='min')

# Define a function to load the spectrogram images and corresponding labels
def load_data():
    x = []
    y = []
    for i, class_name in enumerate(os.listdir(spectrogram_dir)):
        class_path = os.path.join(spectrogram_dir, class_name)
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            img = Image.open(img_path)
            img = img.resize((input_shape[0], input_shape[1]))
            img = np.array(img.convert('L'))
            img = np.stack((img,) * 3, axis=-1) # Convert to 3 channels
            x.append(img)
            y.append(i)
    x = np.array(x)
    y = tf.keras.utils.to_categorical(y, num_classes=len(os.listdir(spectrogram_di
    return x, y

# Load the spectrogram images and corresponding labels
x, y = load_data()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_st

# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

# Add new output layers to the pre-trained model
x = base_model.output
x = Flatten()(x)
x = Dense(64)(x)
x = LeakyReLU()(x)
x = Dropout(0.5)(x)
x = Dense(64)(x)
x = LeakyReLU()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(os.listdir(spectrogram_dir)), activation='softmax')(x)

# Create the new model
model = Model(inputs=base_model.input, outputs=predictions)
```

```python
# Freeze the weights of the pre-trained layers except the last 4 layers
for layer in base_model.layers[:-4]:
    layer.trainable = False

# Compile the model
#optimizer = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac


# Fit the model to the training data
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_test, y_test),
                    callbacks=[early_stop])

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)

# Print the test accuracy
print('Test accuracy:', test_acc)
```

```
Epoch 1/50
19/19 [==============================] - 93s 5s/step - loss: 7.5166 - accuracy: 0.
1204 - val_loss: 2.3834 - val_accuracy: 0.2080
Epoch 2/50
19/19 [==============================] - 92s 5s/step - loss: 2.3785 - accuracy: 0.
2124 - val_loss: 2.3633 - val_accuracy: 0.2080
Epoch 3/50
19/19 [==============================] - 105s 6s/step - loss: 2.3452 - accuracy:
0.2258 - val_loss: 2.3397 - val_accuracy: 0.2431
Epoch 4/50
19/19 [==============================]  - 91s 5s/step - loss: 2.3210 - accuracy: 0.
2358 - val_loss: 2.3156 - val_accuracy: 0.2431
Epoch 5/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2962 - accuracy: 0.
2174 - val_loss: 2.2960 - val_accuracy: 0.2431
Epoch 6/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2774 - accuracy: 0.
2241 - val_loss: 2.2792 - val_accuracy: 0.2431
Epoch 7/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2659 - accuracy: 0.
2007 - val_loss: 2.2650 - val_accuracy: 0.2431
Epoch 8/50
19/19 [==============================]  - 91s 5s/step - loss: 2.2563 - accuracy: 0.
2157 - val_loss: 2.2531 - val_accuracy: 0.2431
Epoch 9/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2394 - accuracy: 0.
2140 - val_loss: 2.2427 - val_accuracy: 0.2431
Epoch 10/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2264 - accuracy: 0.
2391 - val_loss: 2.2340 - val_accuracy: 0.2431
Epoch 11/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2253 - accuracy: 0.
2341 - val_loss: 2.2253 - val_accuracy: 0.2431
Epoch 12/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2162 - accuracy: 0.
2358 - val_loss: 2.2205 - val_accuracy: 0.2431
Epoch 13/50
19/19 [==============================]  - 92s 5s/step - loss: 2.2124 - accuracy: 0.
2274 - val_loss: 2.2158 - val_accuracy: 0.2431
Epoch 14/50
19/19 [==============================]  - 91s 5s/step - loss: 2.2158 - accuracy: 0.
2124 - val_loss: 2.2120 - val_accuracy: 0.2431
Epoch 15/50
19/19 [==============================]  - 91s 5s/step - loss: 2.2084 - accuracy: 0.
2492 - val_loss: 2.2096 - val_accuracy: 0.2431
Epoch 16/50
19/19 [==============================]  - 89s 5s/step - loss: 2.2149 - accuracy: 0.
2441 - val_loss: 2.2082 - val_accuracy: 0.2431
Epoch 17/50
19/19 [==============================]  - 85s 5s/step - loss: 2.1905 - accuracy: 0.
2291 - val_loss: 2.2059 - val_accuracy: 0.2431
Epoch 18/50
19/19 [==============================]  - 74s 4s/step - loss: 2.2133 - accuracy: 0.
2525 - val_loss: 2.2040 - val_accuracy: 0.2431
Epoch 19/50
19/19 [==============================]  - 74s 4s/step - loss: 2.2065 - accuracy: 0.
2291 - val_loss: 2.2032 - val_accuracy: 0.2431
Epoch 20/50
19/19 [==============================]  - 77s 4s/step - loss: 2.1976 - accuracy: 0.
2375 - val_loss: 2.2027 - val_accuracy: 0.2431
Epoch 21/50
19/19 [==============================]  - 75s 4s/step - loss: 2.2106 - accuracy: 0.
2140 - val_loss: 2.2024 - val_accuracy: 0.2431
Epoch 22/50
```

```
19/19 [==============================] - 75s 4s/step - loss: 2.1998 - accuracy: 0.
2124 - val_loss: 2.2017 - val_accuracy: 0.2431
Epoch 23/50
19/19 [==============================] - 75s 4s/step - loss: 2.1939 - accuracy: 0.
2090 - val_loss: 2.2009 - val_accuracy: 0.2431
Epoch 24/50
19/19 [==============================] - 75s 4s/step - loss: 2.1857 - accuracy: 0.
2375 - val_loss: 2.2004 - val_accuracy: 0.2431
Epoch 25/50
19/19 [==============================] - 75s 4s/step - loss: 2.1914 - accuracy: 0.
2659 - val_loss: 2.2002 - val_accuracy: 0.2431
Epoch 26/50
19/19 [==============================] - 75s 4s/step - loss: 2.1921 - accuracy: 0.
2258 - val_loss: 2.2005 - val_accuracy: 0.2431
Epoch 27/50
19/19 [==============================] - 75s 4s/step - loss: 2.1974 - accuracy: 0.
2391 - val_loss: 2.2001 - val_accuracy: 0.2431
Epoch 28/50
19/19 [==============================] - 77s 4s/step - loss: 2.2098 - accuracy: 0.
2375 - val_loss: 2.2004 - val_accuracy: 0.2431
Epoch 29/50
19/19 [==============================] - 75s 4s/step - loss: 2.1940 - accuracy: 0.
2291 - val_loss: 2.2009 - val_accuracy: 0.2431
Epoch 30/50
19/19 [==============================] - 78s 4s/step - loss: 2.1938 - accuracy: 0.
2258 - val_loss: 2.2003 - val_accuracy: 0.2431
Epoch 30: early stopping
13/13 [==============================] - 26s 2s/step - loss: 2.2003 - accuracy: 0.
2431
Test accuracy:  0.24310776591300964
```

In [11]:

```
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 flatten_3 (Flatten)         (None, 8192)              0

 dense_7 (Dense)             (None, 64)                524352

 leaky_re_lu_3 (LeakyReLU)   (None, 64)                0

 dropout_2 (Dropout)         (None, 64)                0

 dense_8 (Dense)             (None, 64)                4160

 leaky_re_lu_4 (LeakyReLU)   (None, 64)                0

 dropout_3 (Dropout)         (None, 64)                0

 dense_9 (Dense)             (None, 11)                715

=================================================================
Total params: 15,243,915
Trainable params: 7,608,651
Non-trainable params: 7,635,264
_________________________________________________________________
```

In [12]:
```python
import os
import numpy as np
import tensorflow as tf
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten, Dropout, LeakyReLU
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.callbacks import EarlyStopping

# Define the path to the directory containing the spectrogram images
spectrogram_dir = 'C:/Users/GOKUL/OneDrive/Desktop/spectrograms'

# Define the input shape of the model
input_shape = (128, 128, 3)

# Define the batch size and number of epochs
batch_size = 32
epochs = 50
early_stop = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='min')

# Define a function to load the spectrogram images and corresponding labels
def load_data():
    x = []
    y = []
    for i, class_name in enumerate(os.listdir(spectrogram_dir)):
        class_path = os.path.join(spectrogram_dir, class_name)
        for img_file in os.listdir(class_path):
            img_path = os.path.join(class_path, img_file)
            img = Image.open(img_path)
            img = img.resize((input_shape[0], input_shape[1]))
            img = np.array(img.convert('L'))
            img = np.stack((img,) * 3, axis=-1) # Convert to 3 channels
            x.append(img)
            y.append(i)
    x = np.array(x)
    y = tf.keras.utils.to_categorical(y, num_classes=len(os.listdir(spectrogram_di
    return x, y

# Load the spectrogram images and corresponding labels
x, y = load_data()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, random_st

# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

# Add new output layers to the pre-trained model
x = base_model.output
x = Flatten()(x)
x = Dense(64)(x)
x = LeakyReLU()(x)
x = Dropout(0.5)(x)
x = Dense(64)(x)
x = LeakyReLU()(x)
x = Dropout(0.5)(x)
predictions = Dense(len(os.listdir(spectrogram_dir)), activation='softmax')(x)

# Create the new model
model = Model(inputs=base_model.input, outputs=predictions)
```

```
# Freeze the weights of the pre-trained layers except the last 4 layers
for layer in base_model.layers[:-4]:
    layer.trainable = False

# Compile the model
optimizer = SGD(lr=0.001, momentum=0.9)
model.compile(optimizer=optimizer,  loss='categorical_crossentropy',  metrics=['accu


# Fit the model to the training data
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_test,  y_test),
                    callbacks=[early_stop])

# Evaluate the model on the test set
test_loss,  test_acc  =  model.evaluate(x_test,  y_test)

# Print the test accuracy
print('Test  accuracy:',  test_acc)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the le
gacy optimizer, e.g.,tf.keras.optimizers.legacy.SGD.
Epoch 1/50
19/19 [==============================] - 80s 4s/step - loss: nan - accuracy: 0.190
6 - val_loss: nan - val_accuracy: 0.2080
Epoch 2/50
19/19 [==============================] - 75s 4s/step - loss: nan - accuracy: 0.207
4 - val_loss: nan - val_accuracy: 0.2080
Epoch 3/50
19/19 [==============================] - 80s 4s/step - loss: nan - accuracy: 0.207
4 - val_loss: nan - val_accuracy: 0.2080
Epoch 4/50
19/19 [==============================] - 83s 4s/step - loss: nan - accuracy: 0.207
4 - val_loss: nan - val_accuracy: 0.2080
Epoch 5/50
19/19 [==============================] - 78s 4s/step - loss: nan - accuracy: 0.207
4 - val_loss: nan - val_accuracy: 0.2080
Epoch 5: early stopping
13/13 [==============================] - 26s 2s/step - loss: nan - accuracy: 0.208
0
Test  accuracy:  0.20802004635334015
```

```
In [13]: model.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 flatten_4 (Flatten)         (None, 8192)              0

 dense_10 (Dense)            (None, 64)                524352

 leaky_re_lu_5 (LeakyReLU)   (None, 64)                0

 dropout_4 (Dropout)         (None, 64)                0

 dense_11 (Dense)            (None, 64)                4160

 leaky_re_lu_6 (LeakyReLU)   (None, 64)                0

 dropout_5 (Dropout)         (None, 64)                0

 dense_12 (Dense)            (None, 11)                715

=================================================================
Total params: 15,243,915
Trainable params: 7,608,651
Non-trainable params: 7,635,264
_________________________________________________________________
```

```
In [ ]:
```