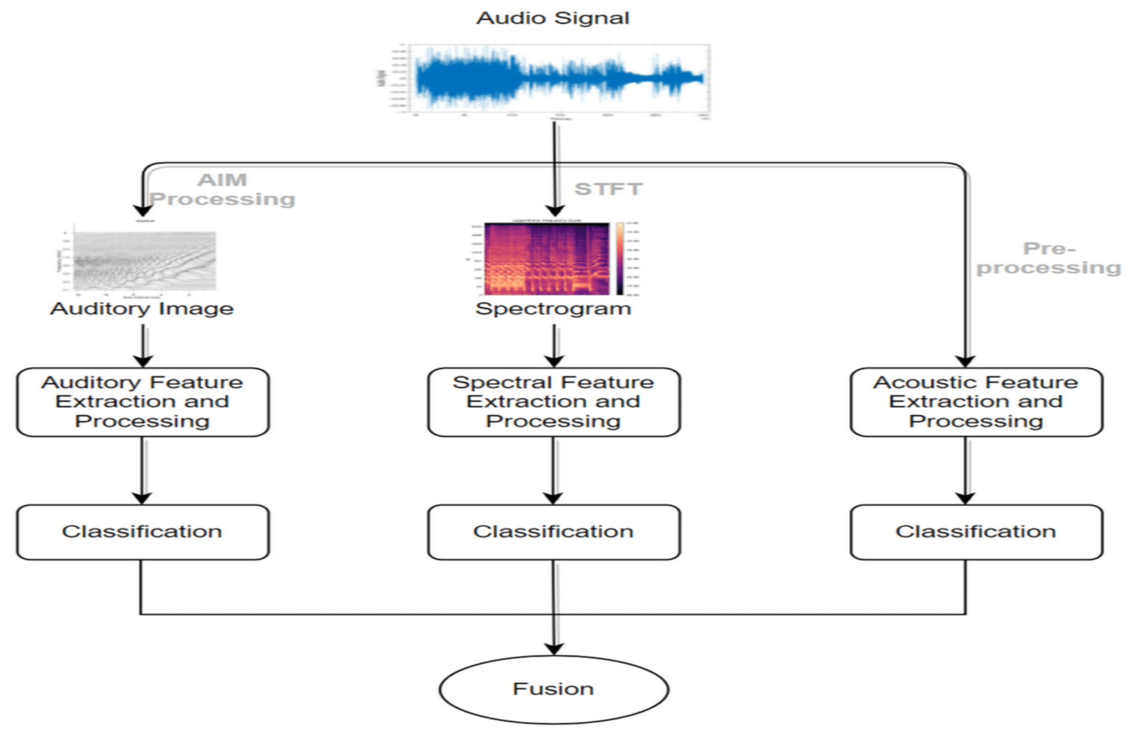
Music Genre Classification using Deep Learning

Course Name: **Tools and Technologies**



Audio Signal
AIM Processing
STFT
Pre-processing
Auditory Image
Spectrogram
Auditory Feature Extraction and Processing
Spectral Feature Extraction and Processing
Acoustic Feature Extraction and Processing
Classification
Classification
Classification
Fusion

SUBMITTED BY:
ATIG PUROHIT(2005090)
SANCHARI RAY(2005264)
SUJATRO BARAL(2005279)
ARGHYA HAZRA(2005440)
SWAPNIL DUTTA(20051229)

DATED: 24.04.2023

TABLE OF CONTENTS

1. LIST OF FIGURES :

Fig.1: Original waveform of an audio file.

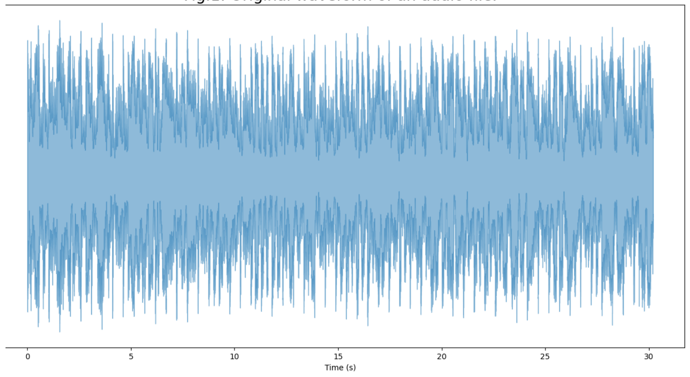


Fig. 2.1: Power spectrum of the above audio file

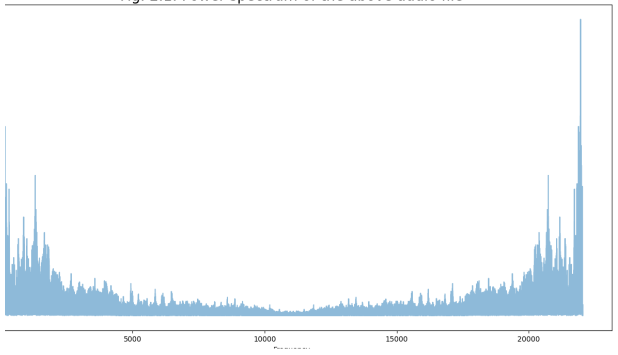


Fig. 2.2: Left Half of the Power spectrum

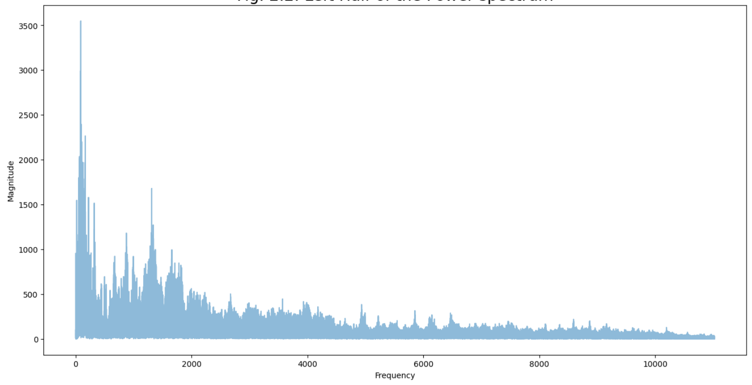


Fig.3: Spectrum of the original audio wave.

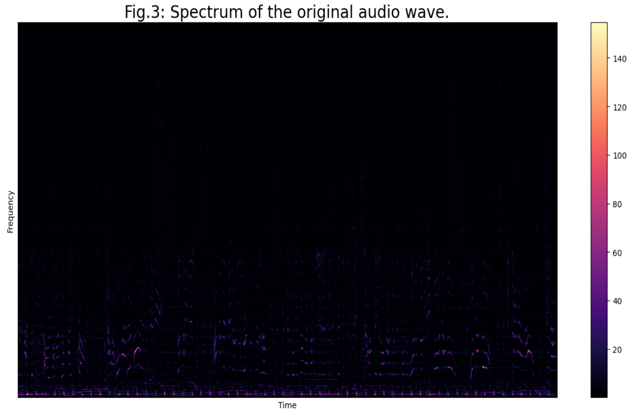


Fig.3.2: Log Spectrum of the original audio wave.

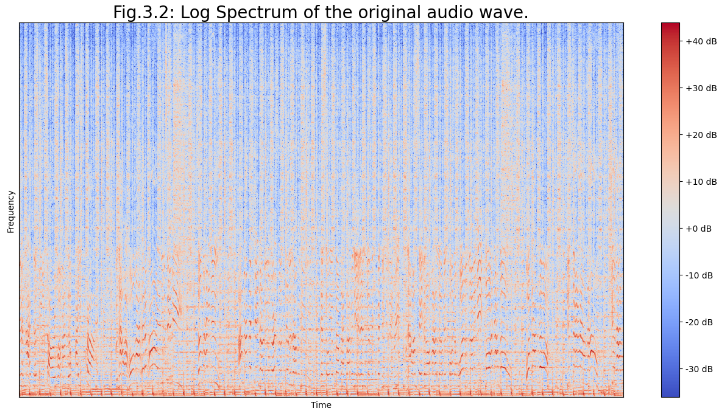
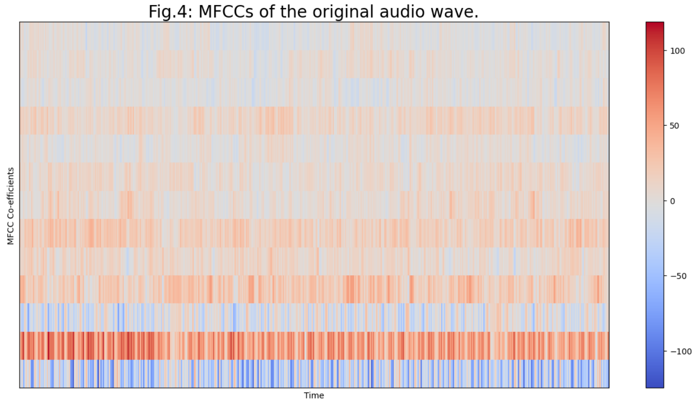


Fig.4: MFCCs of the original audio wave.



2.ABSTRACT

Deep learning-based music genre classification is a well-known study area in the world of music information retrieval. Genre classification will be valuable when there are some interesting facts or problems, finding the most specific song which has been listened to many times. Due to its capacity to automatically learn hierarchical representations of musical data, deep learning techniques, in particular Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have been frequently used to classify the musical genres. Many works that compare the application of this approach from different states use the evaluation and performance of both the handcrafted dataset and the GTZAN dataset.
We provide a thorough analysis of current developments in deep learning-based music genre categorization in this study, covering a range of network designs, feature extraction techniques, and training approaches.We reviewed different feature extraction methods that can be used in conjunction with deep learning models, including spectrogram-based features, mel-frequency cepstral coefficients (MFCCs), and others.Additionally, we talk about the drawbacks and shortcomings of the current approaches, namely the uneven distribution of the data and their interpretability, and we suggest potential remedies for future studies. Finally, we offer a critical assessment of the state-of-the-art techniques now in use and indicate potential future paths for developing the study of music genre classification using Deep Learning.

3.DESCRIPTION OF THE REFERENCE PROJECT

In our referred project, the music genre classification is done using KNN and K-Means clustering methods.

1. Libraries that were used in our referred project are:

numpy
pandas
scipy.io.wavfile
os
math
pickle
random
Operator

2.Defining a function to get the distance between feature vectors and finding the neighbors:

```
 def getNeighbors(trainingSet, instance, k):
     distances = []
     for x in range (len(trainingSet)):
         dist = distance(trainingSet[x], instance, k )+ distance(instance, trainingSet[x], k)
         distances.append((trainingSet[x][2], dist))
     distances.sort(key=operator.itemgetter(1))
     neighbors = []
     for x in range(k):
         neighbors.append(distances[x][0])
     return neighbors
```

3.Identifying the nearest neighbors:

```
def nearestClass(neighbors):
    classVote = {}

    for x in range(len(neighbors)):
        response = neighbors[x]
        if response in classVote:
            classVote[response]+=1
        else:
            classVote[response]=1

    sorter = sorted(classVote.items(), key = 
operator.itemgetter(1), reverse=True)
    return sorter[0][0]
```

4.Defining a function for model evaluation:

```
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range (len(testSet)):
        if testSet[x][-1]==predictions[x]:
            correct+=1
    return 1.0*correct/len(testSet)
```

5.Extracting features from the dataset and dumping these features into a binary .dat file "my.dat":

```
directory = "__path_to_dataset__"
f= open("my.dat" ,'wb')
i=0

for folder in os.listdir(directory):
    i+=1
    if i==11 :
        break
    for file in os.listdir(directory+folder):
        (rate,sig) = wav.read(directory+folder+"/"+file)
        mfcc_feat = mfcc(sig,rate ,winlen=0.020, appendEnergy =
False)
        covariance = np.cov(np.matrix.transpose(mfcc_feat))
        mean_matrix = mfcc_feat.mean(0)
        feature = (mean_matrix , covariance , i)
        pickle.dump(feature , f)

f.close()
```

6.Training and test splitting on the dataset:

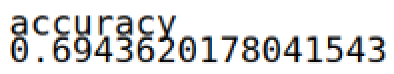
```
dataset = []
def loadDataset(filename , split , trSet , teSet):
    with open("my.dat" , 'rb') as f:
        while True:
            try:
                dataset.append(pickle.load(f))
            except EOFError:
                f.close()
                break

    for x in range(len(dataset)):
        if random.random() <split :
            trSet.append(dataset[x])
        else:
            teSet.append(dataset[x])

trainingSet = []
testSet = []
loadDataset("my.dat" , 0.66, trainingSet, testSet)
```

7.Making prediction using KNN and getting its accuracy on test data:

```
leng = len(testSet)
predictions = []
for x in range (leng):
    predictions.append(nearestClass(getNeighbors(trainingSet ,testSet[x] , 5)))

accuracy1 = getAccuracy(testSet , predictions)
print(accuracy1)
```

<u>OUTPUT:</u>

Accuracy is around 69%.



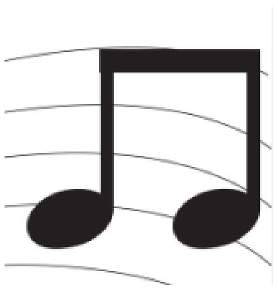
```
accuracy
0.6943620178041543
```

4.PROPOSED ENHANCEMENTS

Enhancements we used are:

1.CNN(**Convolutional Neural Networks**):

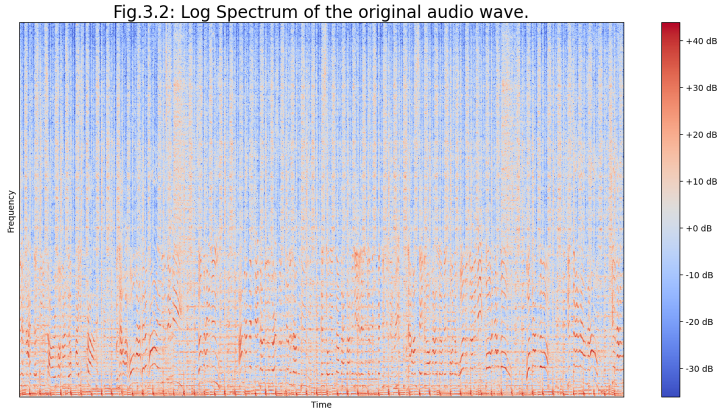
Different tasks, including music tagging, genre classification, and user-item latent feature prediction, are utilized for recommendation in music genre classification.
Convolutional neural networks are used to identify these tasks. Convolutional kernels can be used to recover the source from CNNs as they always process the data at different levels of the features hierarchy. The supplied functions that are taught during supervised training are completed using knowledge that is acquired from hierarchical sources.
CNN is utilized to save the user's time when looking for songs. Its great accuracy is what makes it most popular.
CNN is more accurate than K-NN and Support vector machines.



Original audio

—>

Fig.3.2: Log Spectrum of the original audio wave.

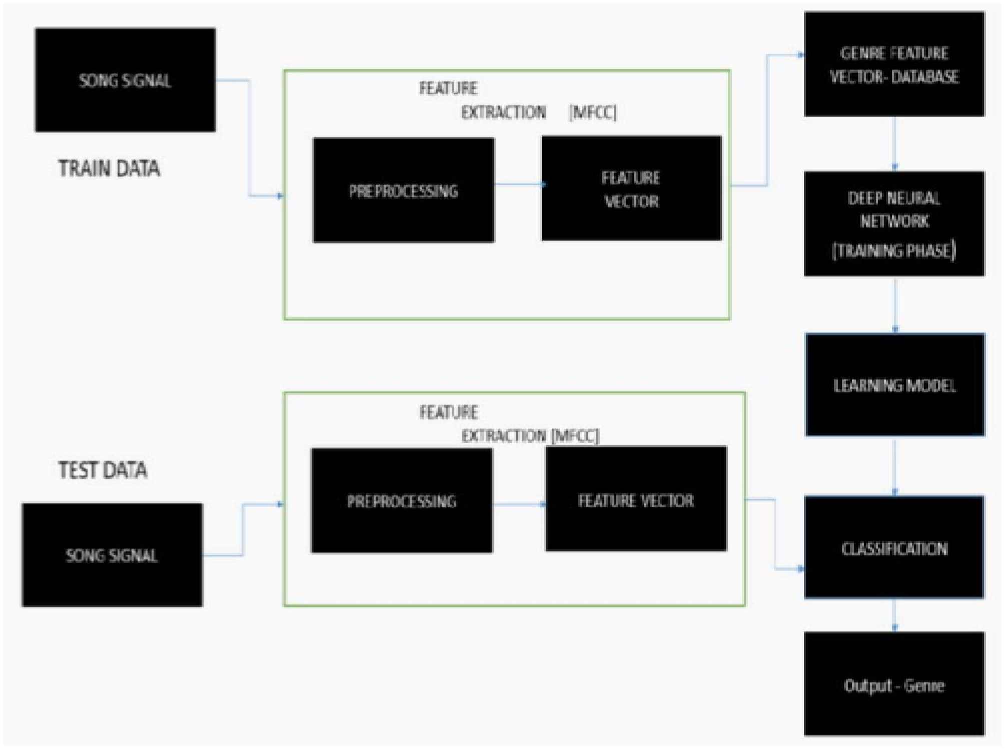


In the above figure, the song audio file which is in waveform is converted into mel-spectrogram. The mel-spectrogram compresses the audio file and then the feature extraction and classification takes place.

2. Spectrogram generation:

A spectrogram is a matrix where the rows denote frequency bins and the columns denote frames (time). The magnitude of a specific frequency inside a certain time interval is expressed using a colour map. The input songs are transformed into Mel spectrograms using log-scale with a window size of 3 seconds and an overlap of 50%, with n fft = 2048 (frame length), hop length = 512, and no. of coefficients = 13 (number of Mel bands).

3. Proposed system design:
   Figure below depicts the stages used in our methodology.
   The music database is initially constructed. After that, each song must go through preprocessing.



The librosa library in Python is used for Feature Vector Extraction. Use this particular programme only for audio analysis.
Each audio file is taken, and the feature vector is then retrieved from there. MFCC refers to the feature vector that was extracted. By recording the general contour of the log-power spectrum on the Mel Frequency scale, the MFCCs encode the timbral characteristics of the music signal.

5.LIBRARIES USED:

1.NUMPY:

NumPy is a Python library for numerical computing that provides a powerful array object and a set of tools for working with these arrays. It is widely used for scientific computing, data analysis, and machine learning applications. NumPy offers efficient mathematical operations on arrays, including linear algebra, Fourier transforms, and statistical analysis. Additionally, NumPy can be integrated with other Python libraries and provides tools for reading and writing data in a variety of formats.

2.MATPLOTLIB:

Matplotlib is a Python library used for creating static, animated, and interactive visualizations in Python. It provides a wide range of 2D and 3D plotting functions for creating line plots, scatter plots, bar plots, histograms, and more. Matplotlib is highly customizable, allowing users to control the appearance of their plots with a range of formatting options and styles. It is widely used in scientific computing, data analysis, and visualization applications. Matplotlib can also be used in combination with other Python libraries to create interactive data visualizations for the web.

3.MATH:

The math library is a standard Python library that provides a set of mathematical functions for performing mathematical operations in Python. It includes functions for basic mathematical operations such as logarithms, trigonometric functions, and exponential functions. Additionally, the math library provides tools for working with floating-point numbers, including functions for rounding, truncating, and comparing numbers. The library is useful for a wide range of scientific and mathematical applications, such as physics, engineering, and data analysis.

4.LIBROSA:

Librosa is a Python library designed for analyzing and processing audio signals. It provides tools for feature extraction, time-series analysis, and transformation of audio signals into numerical representations, such as spectrograms and mel-frequency cepstral coefficients (MFCCs). Librosa is widely used in applications such as music information retrieval, sound classification, and speech recognition. Additionally, the library can be used to load and manipulate audio files in various formats. Librosa is an open-source library and is compatible with other Python scientific computing libraries, such as NumPy, SciPy, and Matplotlib.

5. OS:

The OS library is a standard Python library that provides a way to interact with the operating system. It allows users to perform tasks related to file management, directory operations, and process management. The OS library provides a range of functions for creating, deleting, and modifying files and directories, as well as for navigating file systems and retrieving file and directory information. Additionally, the library provides functions for running system commands and launching external processes. The OS library is useful for a wide range of applications, such as automating file management tasks, interacting with the file system, and launching system processes.

6.JSON:

The JSON library is a standard Python library that provides functions for encoding and decoding JSON (JavaScript Object Notation) formatted data. It allows for the conversion of Python objects, such as lists and dictionaries, into JSON formatted strings, and vice versa. JSON is a lightweight data interchange format that is commonly used for data storage and communication between web services. The JSON library provides a simple and efficient way to work with JSON data in Python, and is often used in web development and data analysis applications.

7.RANDOM:

The random library is a standard Python library that provides functions for generating random numbers and sequences. It includes a range of functions for generating random integers, floating-point numbers, and sequences such as lists and tuples. Additionally, the random library provides functions for shuffling sequences, selecting random items from a sequence, and generating random strings. The library is useful for a wide range of applications, including simulations, games, and cryptography. The random library provides a simple and efficient way to generate random data in Python.

8.SKLEARN:

The scikit-learn (sklearn) library is a popular Python library used for machine learning applications. It provides a range of tools for data preprocessing, feature selection, classification, regression, clustering, and model selection. Sklearn is built on top of other scientific computing libraries such as NumPy, SciPy, and Matplotlib, and integrates well with other machine learning libraries in Python. The library is easy to use and provides a simple and consistent API for building machine learning models. Sklearn is widely used in data science, scientific research, and industry applications for building and deploying machine learning models.

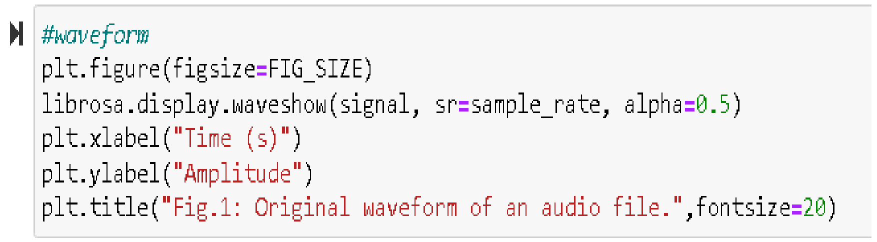
9.TENSORFLOW

TensorFlow is a popular Python library used for machine learning and deep learning applications. It provides a platform for building and training neural networks and other machine learning models. TensorFlow is built on top of computational graphs, which allow for efficient execution of mathematical operations on large datasets. It includes a range of tools for building and training deep neural networks, including functions for creating layers, defining loss functions, and optimizing models. Additionally, TensorFlow provides tools for working with various data formats, such as images and text, and for deploying models to various platforms. TensorFlow is widely used in research and industry applications for building and deploying large-scale machine learning models.

6.WORK DONE:

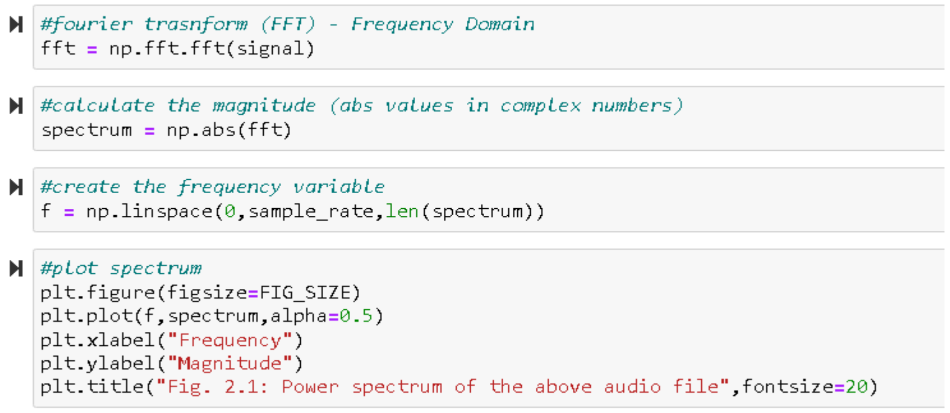
1.Preprocessing of Data:

a.Plotting the waveform:



```
#waveform
plt.figure(figsize=FIG_SIZE)
librosa.display.waveshow(signal, sr=sample_rate, alpha=0.5)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title("Fig.1: Original waveform of an audio file.",fontsize=20)
```

b.Plotting the Power Spectrum:



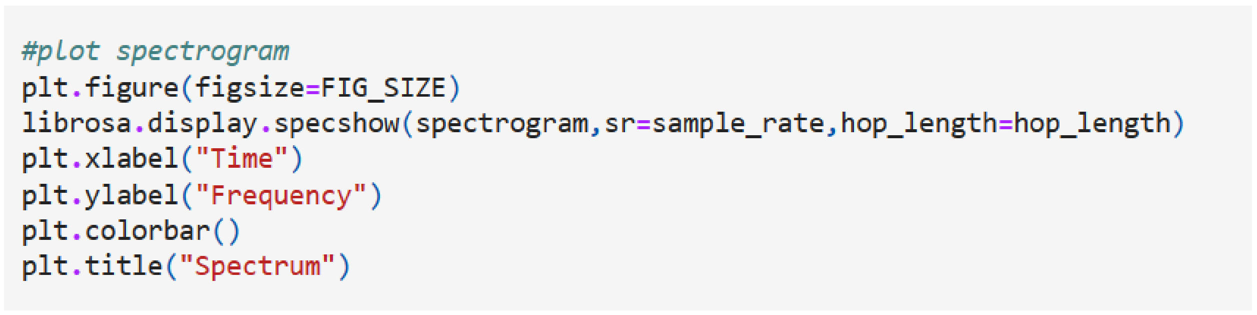
```
#fourier trasnform (FFT) - Frequency Domain
fft = np.fft.fft(signal)
```

```
#calculate the magnitude (abs values in complex numbers)
spectrum = np.abs(fft)
```

```
#create the frequency variable
f = np.linspace(0,sample_rate,len(spectrum))
```

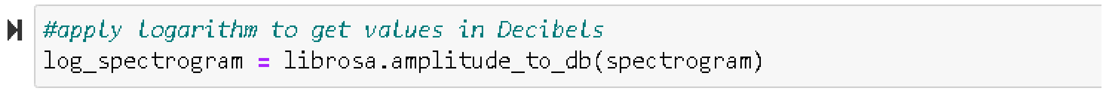
```
#plot spectrum
plt.figure(figsize=FIG_SIZE)
plt.plot(f,spectrum,alpha=0.5)
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.title("Fig. 2.1: Power spectrum of the above audio file",fontsize=20)
```

c.Plotting the Spectrogram:

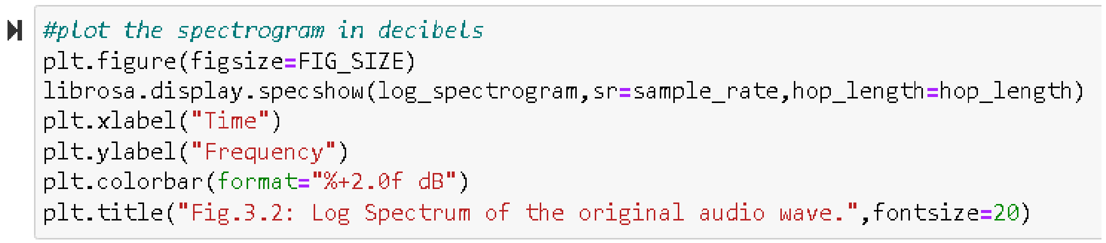


```
#plot spectrogram
plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(spectrogram,sr=sample_rate,hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar()
plt.title("Spectrum")
```

d.Plotting the Log-Spectrogram:

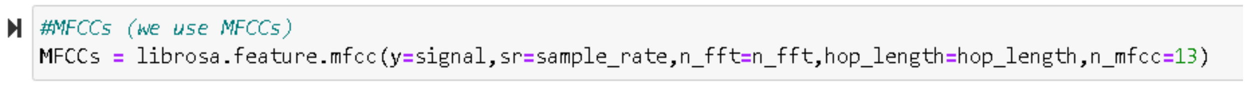


```
#apply logarithm to get values in Decibels
log_spectrogram = librosa.amplitude_to_db(spectrogram)
```

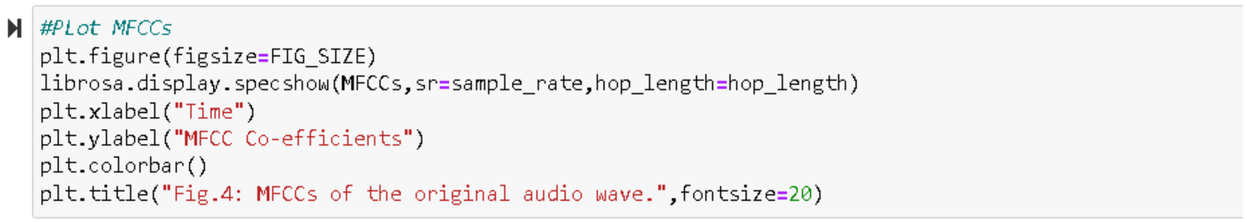


```
#plot the spectrogram in decibels
plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(log_spectrogram,sr=sample_rate,hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar(format="%+2.0f dB")
plt.title("Fig.3.2: Log Spectrum of the original audio wave.",fontsize=20)
```

e.Plotting the MFCCs:



```
#MFCCs (we use MFCCs)
MFCCs = librosa.feature.mfcc(y=signal,sr=sample_rate,n_fft=n_fft,hop_length=hop_length,n_mfcc=13)
```



```
#Plot MFCCs
plt.figure(figsize=FIG_SIZE)
librosa.display.specshow(MFCCs,sr=sample_rate,hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("MFCC Co-efficients")
plt.colorbar()
plt.title("Fig.4: MFCCs of the original audio wave.",fontsize=20)
```

f. Making of the dataset:

In this step we traverse through all the audio files in all the genres and generate the required MFCCs
and label them properly according to their respective genres, and dump everything in a JSON file.

```
import json
import os
import math
import librosa

DATASET_PATH = "D:/6thSem/TNTL/project 2/archive/Data/genres_original"
JSON_PATH = "data_10.json"
```

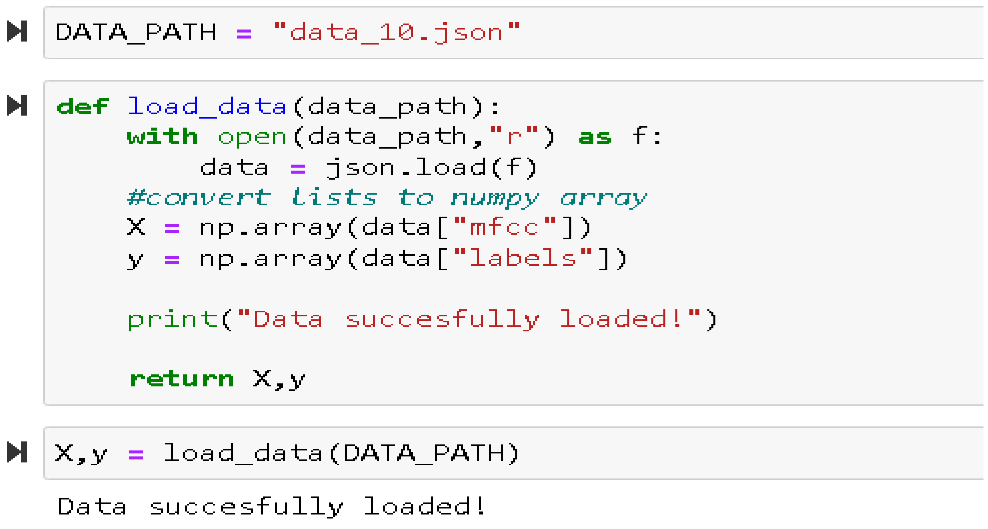
```
SAMPLE_RATE = 22050
TRACK_DURATION = 30 # measured in seconds
SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION


def save_mfcc(dataset_path, json_path, num_mfcc=13, n_fft=2048,
hop_length=512, num_segments=5):
    """Extracts MFCCs from music dataset and saves them into a json file
along witgh genre labels.

        :param dataset_path (str): Path to dataset
        :param json_path (str): Path to json file used to save MFCCs
        :param num_mfcc (int): Number of coefficients to extract
        :param n_fft (int): Interval we consider to apply FFT. Measured in
# of samples
        :param hop_length (int): Sliding window for FFT. Measured in # of
samples
        :param: num_segments (int): Number of segments we want to divide
sample tracks into
        :return:
        """

    # dictionary to store mapping, labels, and MFCCs
    data = {
        "mapping": [],
        "labels": [],
        "mfcc": []
    }

    samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment /
hop_length)

    # loop through all genre sub-folder
    for i, (dirpath, dirnames, filenames) in
enumerate(os.walk(dataset_path)):

        # ensure we're processing a genre sub-folder level
        if dirpath is not dataset_path:
            # save genre label (i.e., sub-folder name) in the mapping
            semantic_label =
dirpath.split("/")[-1]#[genres_original,blues]

#D:\6thSem\TNTL\porject2\archive\Data\genres_original\blues\disco01.wav
            data["mapping"].append(semantic_label)
            print("\nProcessing: {}".format(semantic_label))

            # process all audio files in genre sub-dir
            for f in filenames:

        # load audio file
                file_path = os.path.join(dirpath, f)
                signal, sample_rate = librosa.load(file_path,
sr=SAMPLE_RATE)

                # process all segments of audio file
                for d in range(num_segments):

                    # calculate start and finish sample for current
segment
                    start = samples_per_segment * d
                    finish = start + samples_per_segment

                    # extract mfcc
                    mfcc = librosa.feature.mfcc(y=signal[start:finish],sr=
sample_rate, n_mfcc=num_mfcc, n_fft=n_fft, hop_length=hop_length)
                    mfcc = mfcc.T

                    # store only mfcc feature with expected number of
vectors
                    if len(mfcc) == num_mfcc_vectors_per_segment:
                        data["mfcc"].append(mfcc.tolist())
                        data["labels"].append(i-1)
                        print("{}, segment:{}".format(file_path, d+1))

    # save MFCCs to json file
    with open(json_path, "w") as fp:
        json.dump(data, fp, indent=4)
```

```
if __name__ == "__main__":
    save_mfcc(DATASET_PATH, JSON_PATH, num_segments=10)
```

Since our data has been preprocessed we can move on to model selection and training phase.

2.Training and Testing phase:

a. Loading the data:



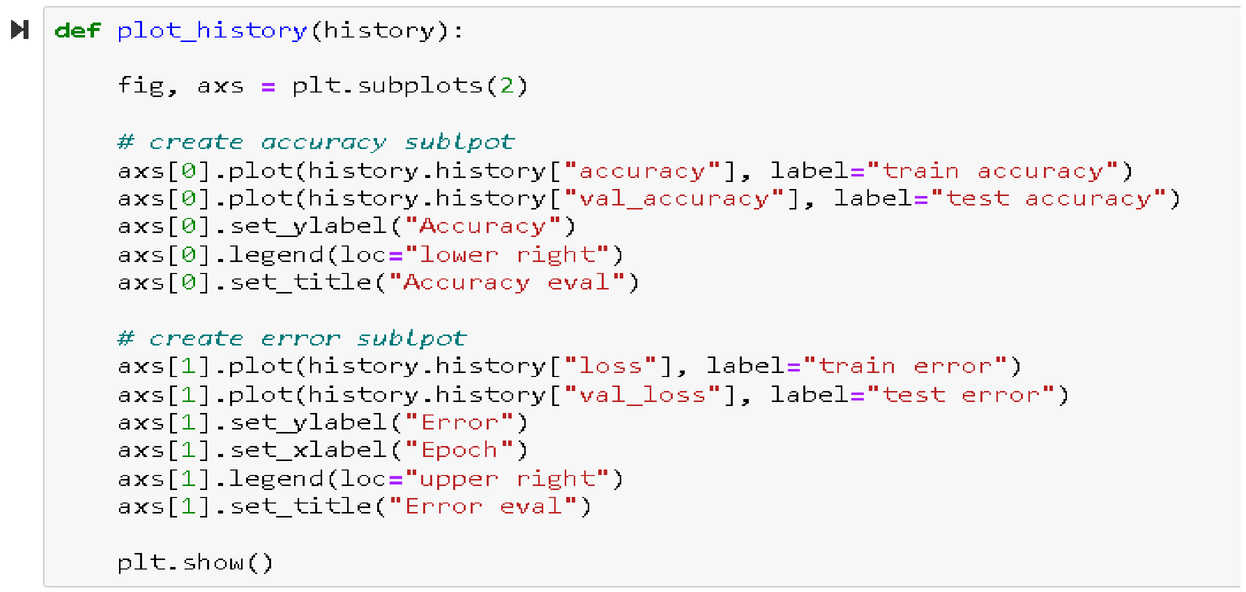
```
DATA_PATH = "data_10.json"
```

```
def load_data(data_path):
    with open(data_path,"r") as f:
        data = json.load(f)
    #convert lists to numpy array
    X = np.array(data["mfcc"])
    y = np.array(data["labels"])

    print("Data succesfully loaded!")

    return X,y
```

```
X,y = load_data(DATA_PATH)
```

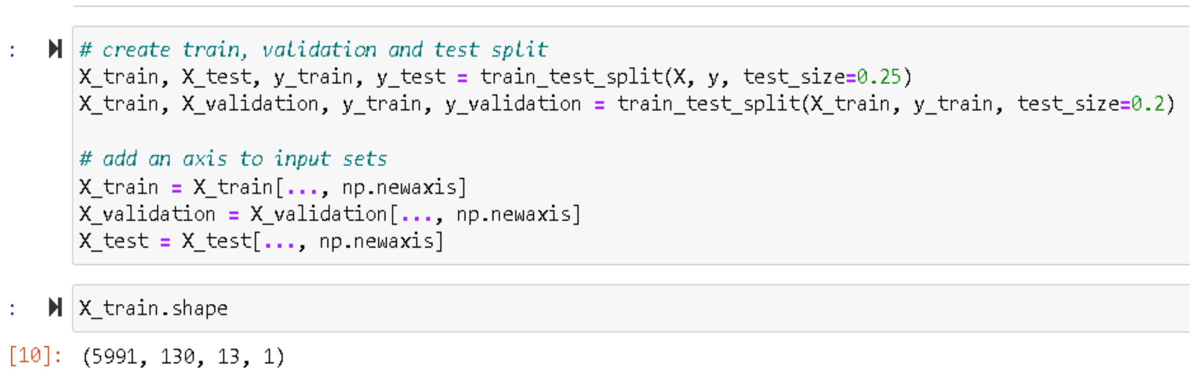
```
Data succesfully loaded!
```

b. Function to evaluate error and accuracy:



```python
def plot_history(history):

    fig, axs = plt.subplots(2)

    # create accuracy sublpot
    axs[0].plot(history.history["accuracy"], label="train accuracy")
    axs[0].plot(history.history["val_accuracy"], label="test accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy eval")

    # create error sublpot
    axs[1].plot(history.history["loss"], label="train error")
    axs[1].plot(history.history["val_loss"], label="test error")
    axs[1].set_ylabel("Error")
    axs[1].set_xlabel("Epoch")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Error eval")

    plt.show()
```

c. Creating train,test and validation split:

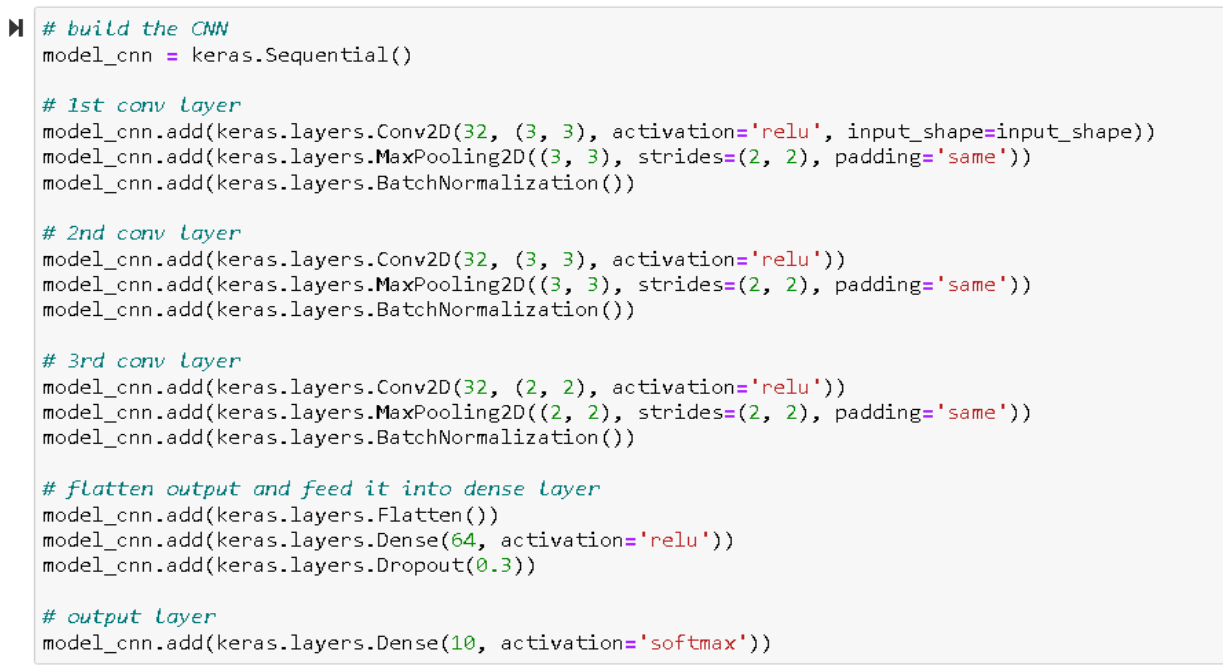


```python
# create train, validation and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.2)

# add an axis to input sets
X_train = X_train[..., np.newaxis]
X_validation = X_validation[..., np.newaxis]
X_test = X_test[..., np.newaxis]
```

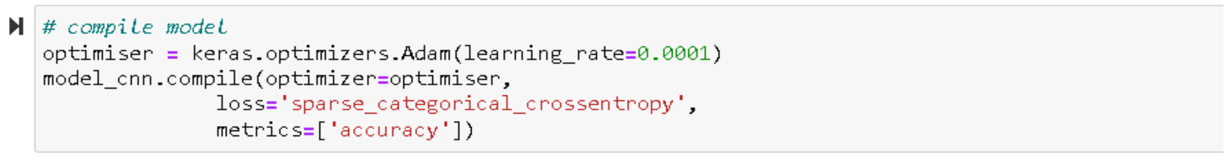
```python
X_train.shape
```

```
[10]: (5991, 130, 13, 1)
```

d. Building the CNN:

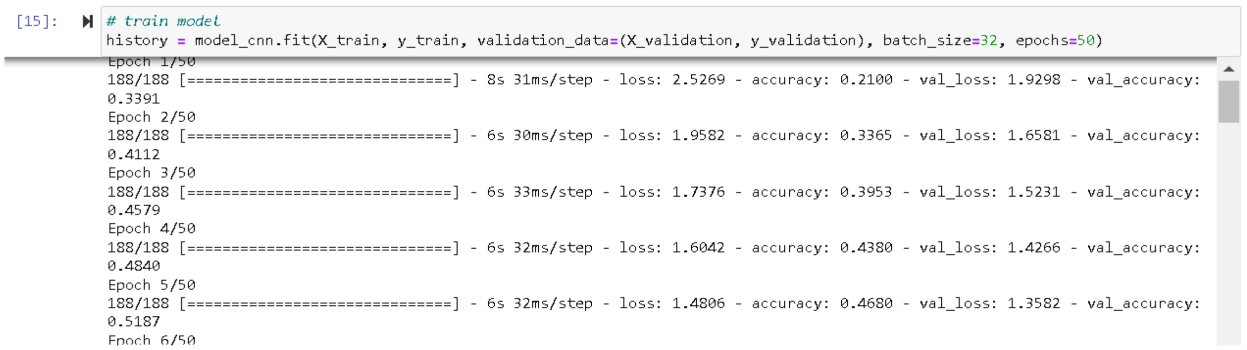


```
# build the CNN
model_cnn = keras.Sequential()

# 1st conv layer
model_cnn.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model_cnn.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model_cnn.add(keras.layers.BatchNormalization())

# 2nd conv layer
model_cnn.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
model_cnn.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model_cnn.add(keras.layers.BatchNormalization())

# 3rd conv layer
model_cnn.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
model_cnn.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
model_cnn.add(keras.layers.BatchNormalization())

# flatten output and feed it into dense layer
model_cnn.add(keras.layers.Flatten())
model_cnn.add(keras.layers.Dense(64, activation='relu'))
model_cnn.add(keras.layers.Dropout(0.3))

# output layer
model_cnn.add(keras.layers.Dense(10, activation='softmax'))
```



```
# compile model
optimiser = keras.optimizers.Adam(learning_rate=0.0001)
model_cnn.compile(optimizer=optimiser,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

e. Training the model:

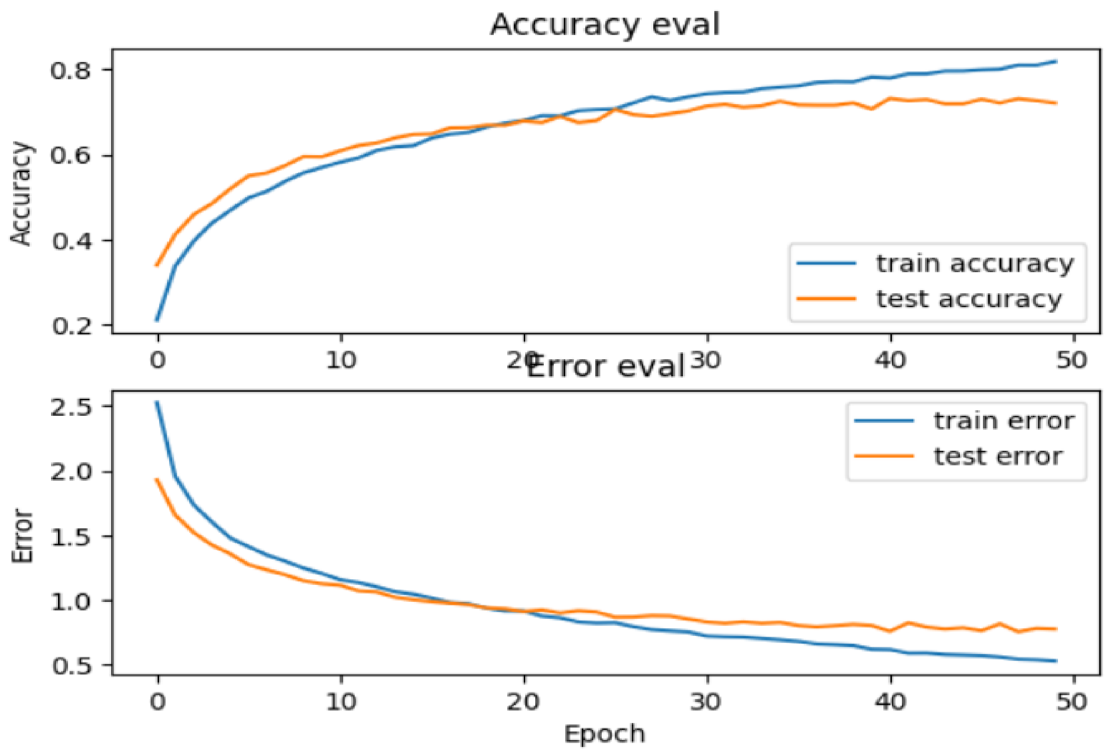


```
[15]:  # train model
       history = model_cnn.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32, epochs=50)
```

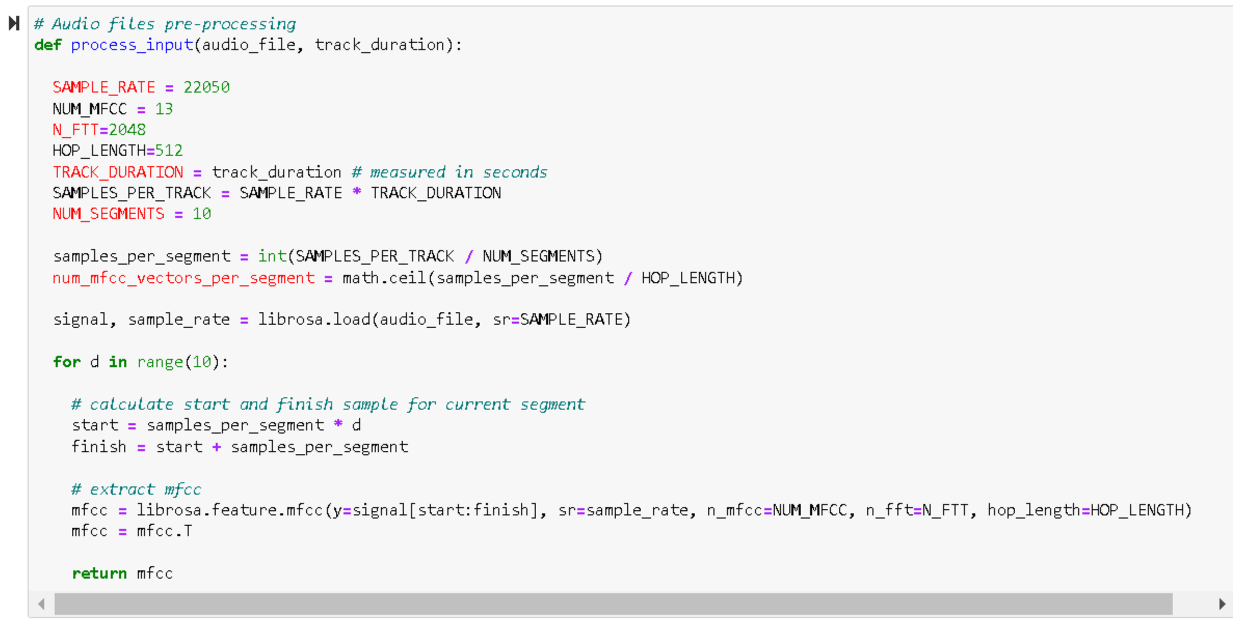
```
Epoch 1/50
188/188 [==============================] - 8s 31ms/step - loss: 2.5269 - accuracy: 0.2100 - val_loss: 1.9298 - val_accuracy:
0.3391
Epoch 2/50
188/188 [==============================] - 6s 30ms/step - loss: 1.9582 - accuracy: 0.3365 - val_loss: 1.6581 - val_accuracy:
0.4112
Epoch 3/50
188/188 [==============================] - 6s 33ms/step - loss: 1.7376 - accuracy: 0.3953 - val_loss: 1.5231 - val_accuracy:
0.4579
Epoch 4/50
188/188 [==============================] - 6s 32ms/step - loss: 1.6042 - accuracy: 0.4380 - val_loss: 1.4266 - val_accuracy:
0.4840
Epoch 5/50
188/188 [==============================] - 6s 32ms/step - loss: 1.4806 - accuracy: 0.4680 - val_loss: 1.3582 - val_accuracy:
0.5187
Epoch 6/50
```

f. Plotting the accuracy:

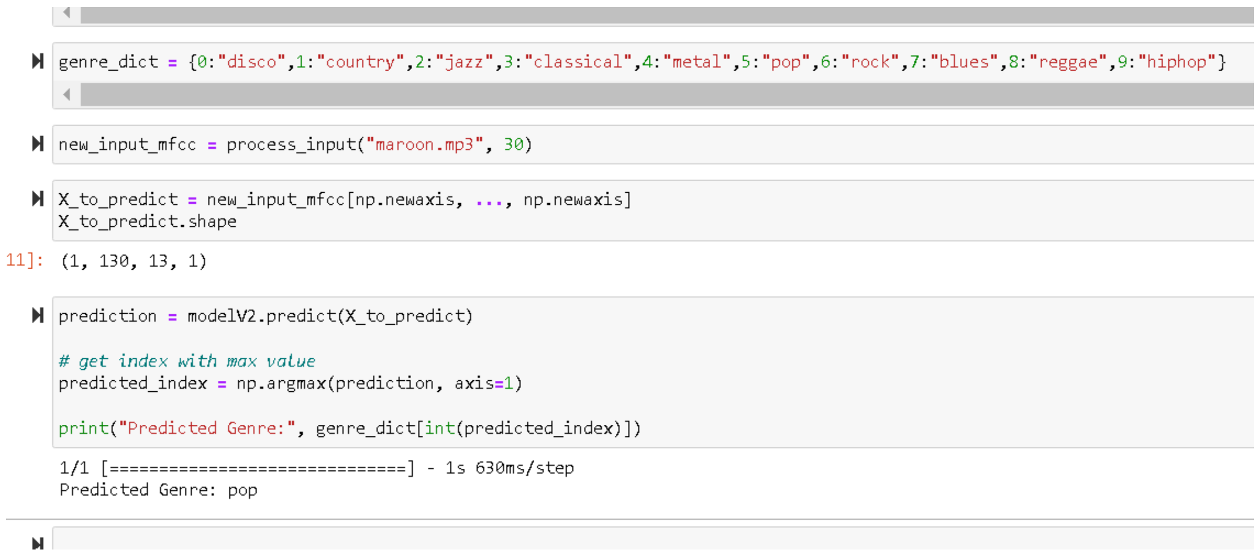
```
# plot accuracy and error as a function of the epochs
plot_history(history)
```



g. Making Predictions:



```python
# Audio files pre-processing
def process_input(audio_file, track_duration):

  SAMPLE_RATE = 22050
  NUM_MFCC = 13
  N_FTT=2048
  HOP_LENGTH=512
  TRACK_DURATION = track_duration # measured in seconds
  SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION
  NUM_SEGMENTS = 10

  samples_per_segment = int(SAMPLES_PER_TRACK / NUM_SEGMENTS)
  num_mfcc_vectors_per_segment = math.ceil(samples_per_segment / HOP_LENGTH)

  signal, sample_rate = librosa.load(audio_file, sr=SAMPLE_RATE)

  for d in range(10):

    # calculate start and finish sample for current segment
    start = samples_per_segment * d
    finish = start + samples_per_segment

    # extract mfcc
    mfcc = librosa.feature.mfcc(y=signal[start:finish], sr=sample_rate, n_mfcc=NUM_MFCC, n_fft=N_FTT, hop_length=HOP_LENGTH)
    mfcc = mfcc.T

    return mfcc
```



```python
genre_dict = {0:"disco",1:"country",2:"jazz",3:"classical",4:"metal",5:"pop",6:"rock",7:"blues",8:"reggae",9:"hiphop"}
```

```python
new_input_mfcc = process_input("maroon.mp3", 30)
```

```python
X_to_predict = new_input_mfcc[np.newaxis, ..., np.newaxis]
X_to_predict.shape
```

```
11]: (1, 130, 13, 1)
```

```python
prediction = modelV2.predict(X_to_predict)

# get index with max value
predicted_index = np.argmax(prediction, axis=1)

print("Predicted Genre:", genre_dict[int(predicted_index)])
```

```
1/1 [==============================] - 1s 630ms/step
Predicted Genre: pop
```

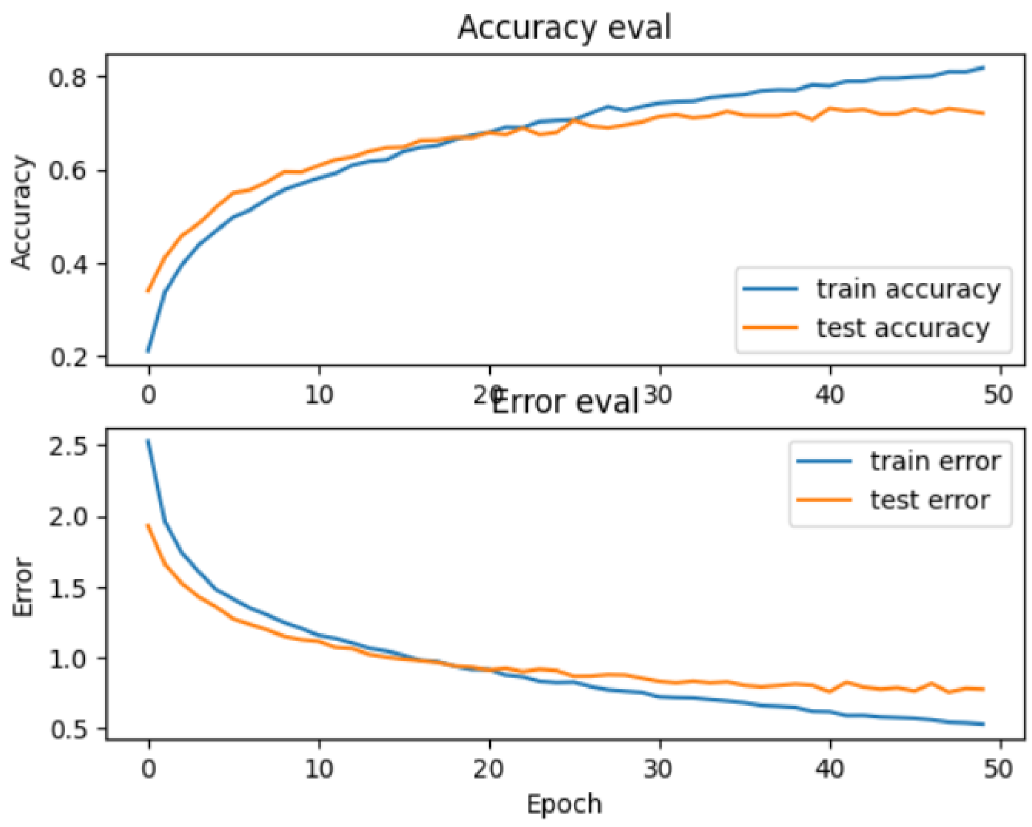
7.RESULTS:

The Anaconda package for Python was used for the evaluation. The deep learning package utilized was Tensorflow. 77.72% was the learning accuracy measured using the Mel Spec feature vector and the MFCC feature vector. Mel Spec requires more time to learn whereas MFCC needs less time to converge. Prediction is quicker once the model has been built.

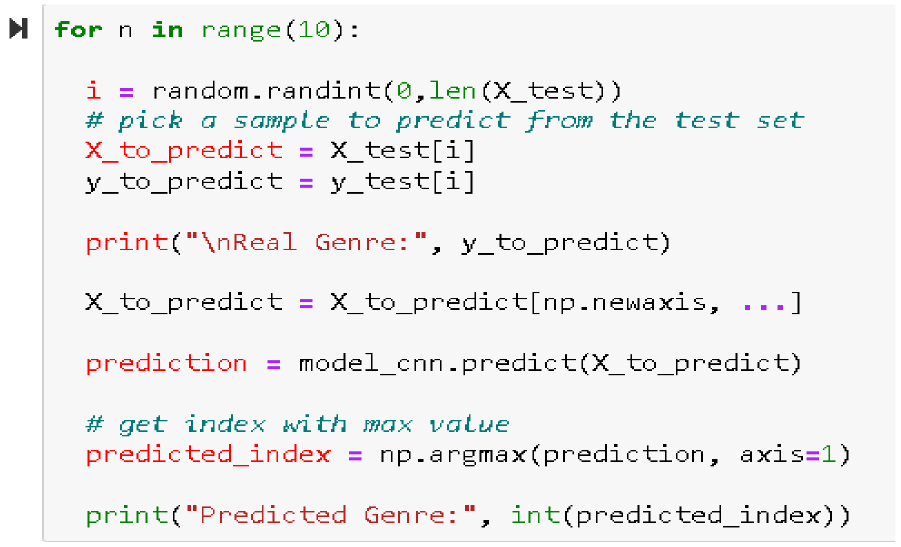
The implemented model has a higher rate and correctly identifies the musical genres of Pop, Disco, and Blues with recall values of 93.8%, 85.8%, and 83.8%, respectively. However, it performs poorly and significantly less well in identifying Metal and Pop music, with recall values of 49.8% and 53.5%, respectively.

Plot for accuracy and error evaluation:

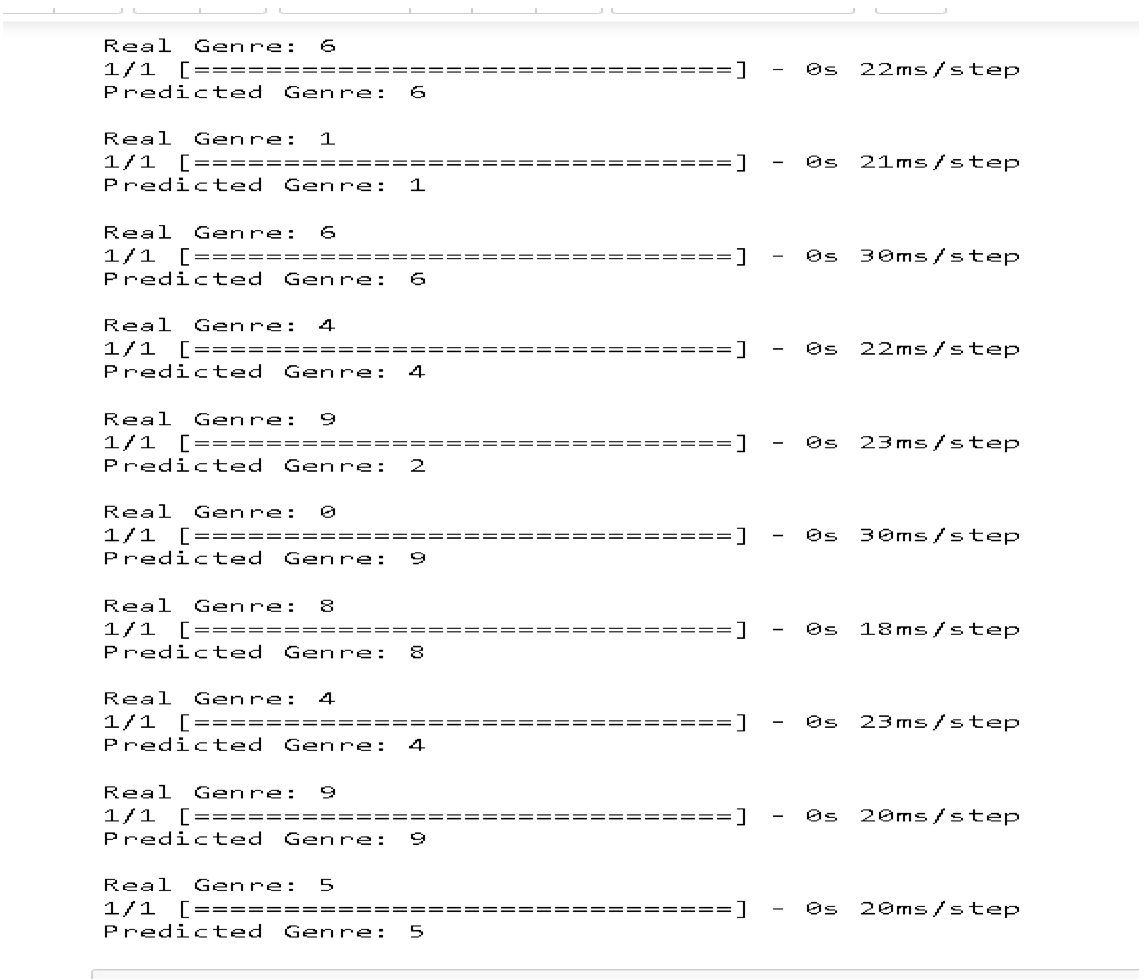
```
# plot accuracy and error as a function of the epochs
plot_history(history)
```



Predictions :



```
for n in range(10):

    i = random.randint(0,len(X_test))
    # pick a sample to predict from the test set
    X_to_predict = X_test[i]
    y_to_predict = y_test[i]

    print("\nReal Genre:", y_to_predict)

    X_to_predict = X_to_predict[np.newaxis, ...]

    prediction = model_cnn.predict(X_to_predict)

    # get index with max value
    predicted_index = np.argmax(prediction, axis=1)

    print("Predicted Genre:", int(predicted_index))
```



```
Real Genre: 6
1/1 [==============================] - 0s 22ms/step
Predicted Genre: 6

Real Genre: 1
1/1 [==============================] - 0s 21ms/step
Predicted Genre: 1

Real Genre: 6
1/1 [==============================] - 0s 30ms/step
Predicted Genre: 6

Real Genre: 4
1/1 [==============================] - 0s 22ms/step
Predicted Genre: 4

Real Genre: 9
1/1 [==============================] - 0s 23ms/step
Predicted Genre: 2

Real Genre: 0
1/1 [==============================] - 0s 30ms/step
Predicted Genre: 9

Real Genre: 8
1/1 [==============================] - 0s 18ms/step
Predicted Genre: 8

Real Genre: 4
1/1 [==============================] - 0s 23ms/step
Predicted Genre: 4

Real Genre: 9
1/1 [==============================] - 0s 20ms/step
Predicted Genre: 9

Real Genre: 5
1/1 [==============================] - 0s 20ms/step
Predicted Genre: 5
```

8.CONCLUSION

We explored the use of CNN in the classification of musical genres.Low data volumes for transfer learning will result in decreased accuracy.
The Mel Spectrum for each track in the GTZAN dataset is obtained. The python package library librosa is capable of carrying out this. The process of categorizing music according to genre is carried out by a software system. The single-frame songs with an average state are improved using a multiframe strategy. The experiment uses 1000 songs from 10 distinct genres, and 8 of these 10 genres match with the highest accuracy rate.
Time and frequency domain, feature extraction, and neural network modeling were the 3 different components we utilized.
In order to improve performance, future research can find techniques to pre-process this noisy data before feeding it to a machine learning model.Finally, greater effort could be put into selecting top-notch databases. This can entail modifying and condensing already-existing datasets, fusing together already-existing datasets, or gathering fresh tracks that aren't already in the public domain. Overall, there is still much opportunity for research and analysis into the topic of music genre classification, which is a rewarding task for both corporations and academic organizations.

9.REFERENCES

[1] Navneet Parab, Shikta Das, Gunj Goda, Ameya Naik,Music Genre Classification using Deep Learning,Oct 2021 International Research Journal of Engineering and Technology (IRJET) ,1462-1463

[2] K S Mounika; S Deyaradevi; K Swetha; V Vanitha,Music Genre Classification,2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA),2021.

[3] https://github.com/mlachmish/MusicGenreClassification

[4] Deep learning with Python by Mark Graph

[5]Mitt Shah; Nandit Pujara; Kaushil Mangaroliya; Lata Gohil; Tarjni Vyas; Sheshang Degadwala ,2022 6th International Conference on Computing Methodologies and Communication (ICCMC).

[6] https://github.com/egarciamartin/music-genre-classification-cnn