A Minor Project Report on

“Music Genre Classification using Audio Dataset”

Submitted by
Nitin Sharma (202IT017), Tarushi Jat (202IT029)

under supervision of
Dr. Anand Kumar M

in partial fulfillment for the award of the degree of

MASTER OF TECHNOLOGY
in
INFORMATION TECHNOLOGY

Department Of Information Technology

National Institute Of Technology, Surathkal

ACKNOWLEDGEMENT

We would like to express our deep gratitude to **Dr Anand Kumar M**, our project supervisors, for his patient guidance, enthusiastic encouragement and useful critiques of this project work. I would also like to thank him for his advice and assistance in keeping our progress on schedule.

We would also like to extend our thanks to our batch mates for pitching in with their suggestions and finally we wish to thank our respective parents for their support and encouragement throughout our study.

ABSTRACT

The huge growth of the digital music databases people begin to realize the importance of effectively managing music databases relying on music content analysis. Music classification serves as the fundamental step towards rapid growth and is useful in music indexing. Our goal is to develop a deep learning model that can classify audio music files into different genres using their frequency spectrum. As an output our model will classify an input audio file into one of the following 10 genres : *Blues, Classical, Country, Disco, Hip-Hop, Jazz, Metal, Pop, Reggae and Rock*. Here convolutional neural networks are used for training and classification. Feature Extraction is the most crucial task for audio analysis. Mel Frequency Cepstral Coefficient (MFCC) is used as a feature vector for sound samples. Our result shows the accuracy of our model on test data is around 80% and this will greatly improve the classification of music into different genres.

Keywords: *Music Genre Classification, CNN, Audio, Frequency.*

CONTENTS

INTRODUCTION

Large number of people download and buy music online. Users usually buy albums based on singer or genre. Problem is that there is loads of musical data available which are not labelled and you cannot fetch information regarding their genre.

Music classification is considered as a very challenging task as selection of appropriate features is a very hard task. While unlabeled data is readily available music tracks with appropriate genre tags is very less. Music genre classification is composed of two basic steps: feature extraction and classification. In the first stage, various features are extracted from the waveform. In the second stage, a classifier is built using the features extracted from the training data. The types of features extracted varies from person to person.

The Neural Network is very widely used in classification problems and it is helpful in training huge database. We propose a novel approach for the automatic music genre classification using Convolution Neural Networks (CNN).

The features from the music that we are extracting are called as **Mel Frequency Cepstral Coefficients (MFCC).** They are obtained by taking the **Fourier transforms of the audio signals**, then taking the logarithmic of the power values and then taking the cosine transforms. These extracted features then acts as the inputs to the neurons for training. Here we are analyzing music from ten various genres. We are doing implementation in python programming using Collaboratory as base. The average accuracy obtained by using the MFCC feature vectors and CNN model is 81%.

LITERATURE SURVEY

Musical Genre classification has attracted many researchers in past. **Tzanetakis and Cook** pioneered their work on music genre classification using machine learning algorithm. They created the **GTZAN dataset** which is till date considered as a standard for genre classification. **Vishnupriya S and K Meenakshi** proposed a Convolutional Neural Network Model to perform the classification of musical genre taking **Million Song Dataset (MSD)** and they achieved an accuracy of 76% with it. Changsheng Xu have shown how to use support vector machines (SVM) for this task.

Here we are going to implement an CNN model for our musical dataset and try to achieve higher accuracies.

METHODOLOGY

A. Dataset Details

We have used GTZAN Music Genre Classification Dataset. Dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres (Blues, Classical, Country, Disco, Hip-Hop, Jazz, Metal, Pop, Reggae and Rock), each represented by 100 tracks. This dataset was originally created by Cook and Tzanetakis, which they have used for their work in Music Genre Classification in 2002.

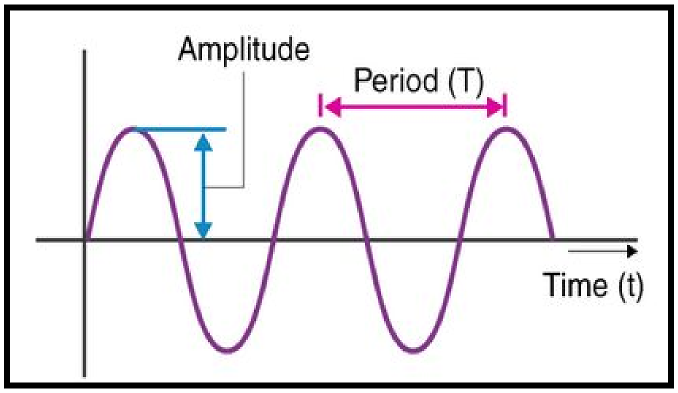
Table 1. Distribution of the dataset

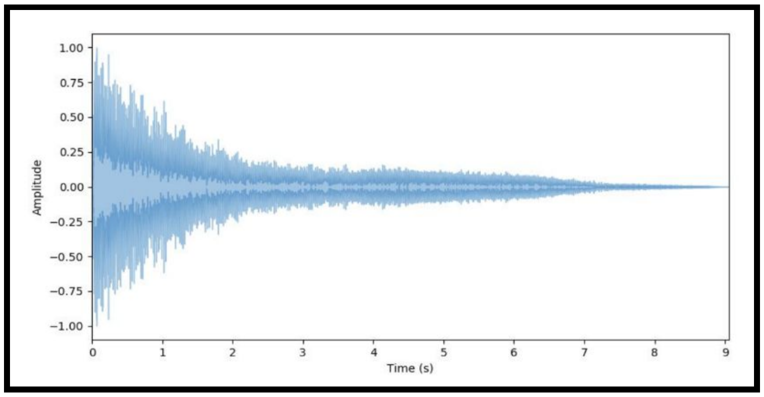
Genre	Number of Tracks
Blues	100
Classical	100
Country	100
Disco	100
Hip-Hop	100
Jazz	100
Metal	100
Pop	100
Reggae	100
Rock	100

B. Understanding Audio Data

Audio Signal: A complex signal composed of multiple “single - frequency” sound waves. When sound is recorded, we only capture the resultant amplitude of those multiple waves. A signal is a variation of a **certain quantity over time**.

For audio, the quantity that varies is air pressure.



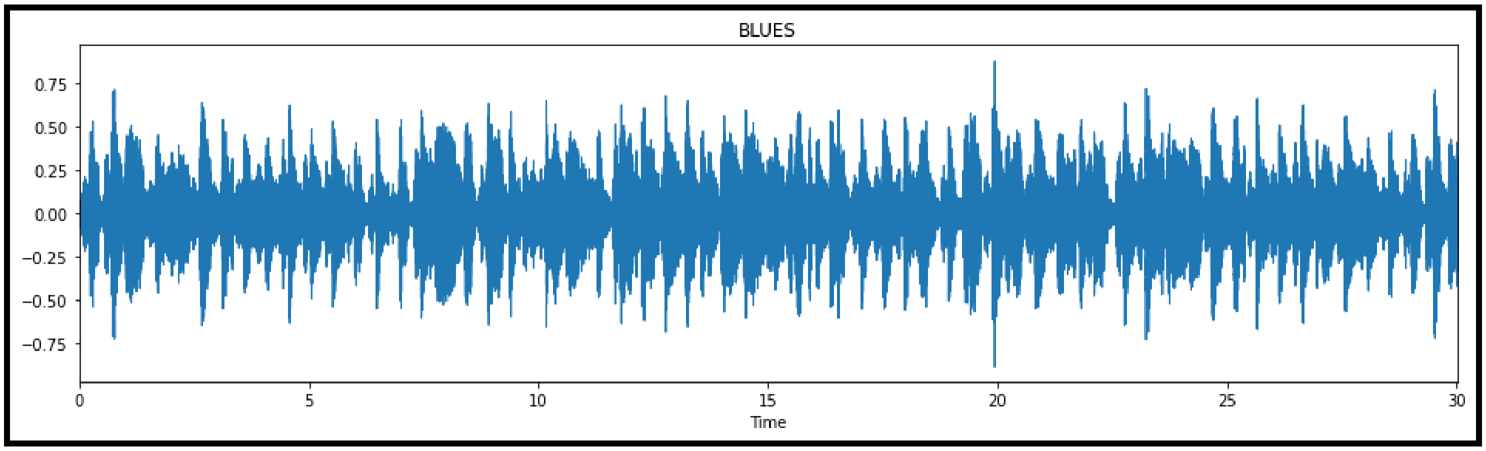


Two common ways to represent sound :

- *Time domain:* each sample represents the variation in air pressure.
- *Frequency domain:* at each time stamp we indicate the amplitude for each frequency.

To capture this audio data, we take samples of the air pressure over time. The rate at which we sample the data can vary, but is most commonly 44.1kHz, or 44,100 amplitude samples per second. This data that we have captured is called Audio Waveform. This audio waveform can be plotted using amplitude and sampling-rate that we can get using python’s librosa library’s load() function.

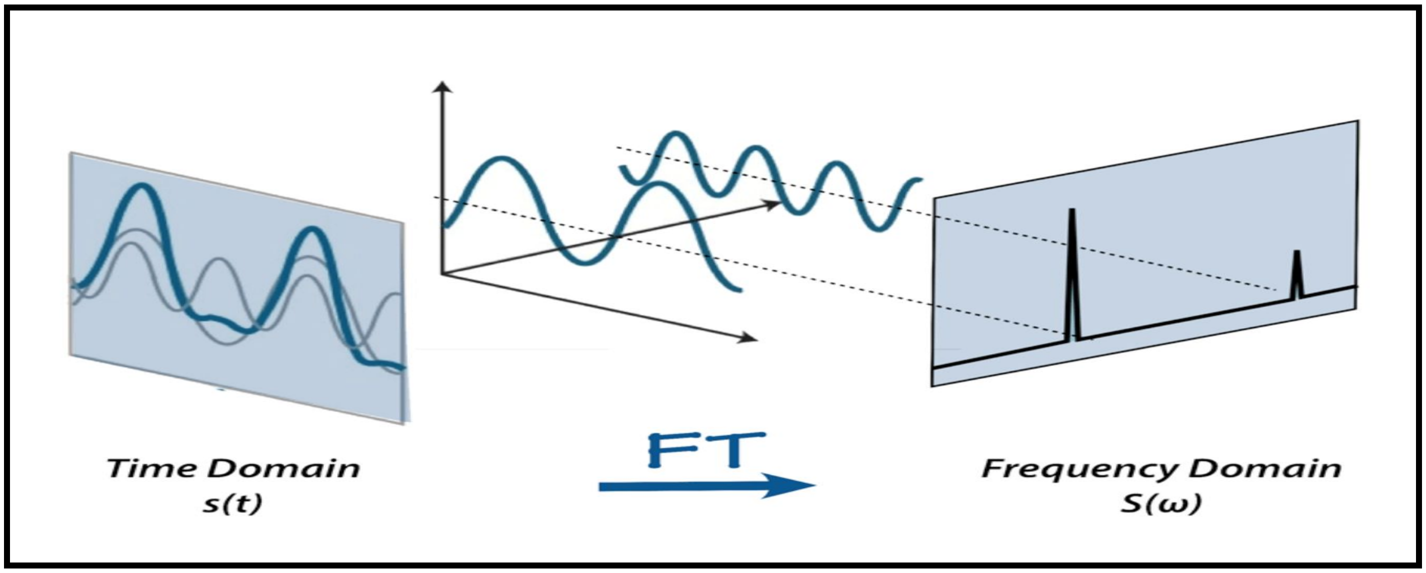
This is how our audio waveform from blues genre looks like:



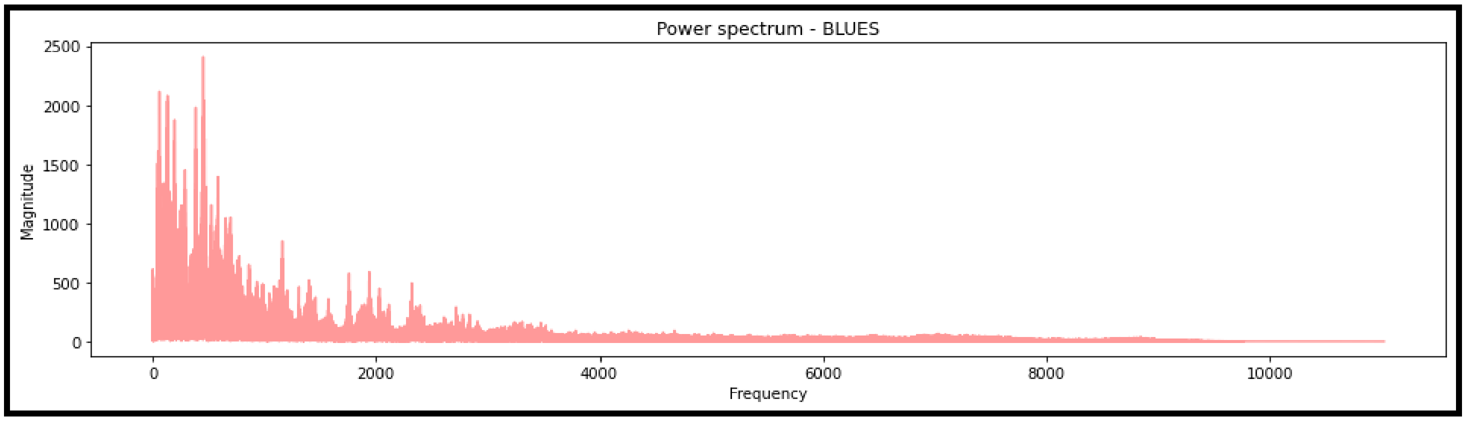
When the sound wave is recorded we only capture the resultant amplitudes of those waves.And these amplitudes are not very informative because they only talk about loudness of audio recording (Amplitude = 0 means silence). To understand audio signals, it is necessary to transform it into frequency-domain. Therefore, to capture other important details, we perform various kinds of transformations.

Fast Fourier Transform

An audio signal is composed of several single-frequency sound waves. When taking samples of the signal over time, we only capture the resulting amplitudes. The **Fourier transform** is a mathematical formula that allows us to decompose a signal into its individual frequencies and the frequency's amplitude.



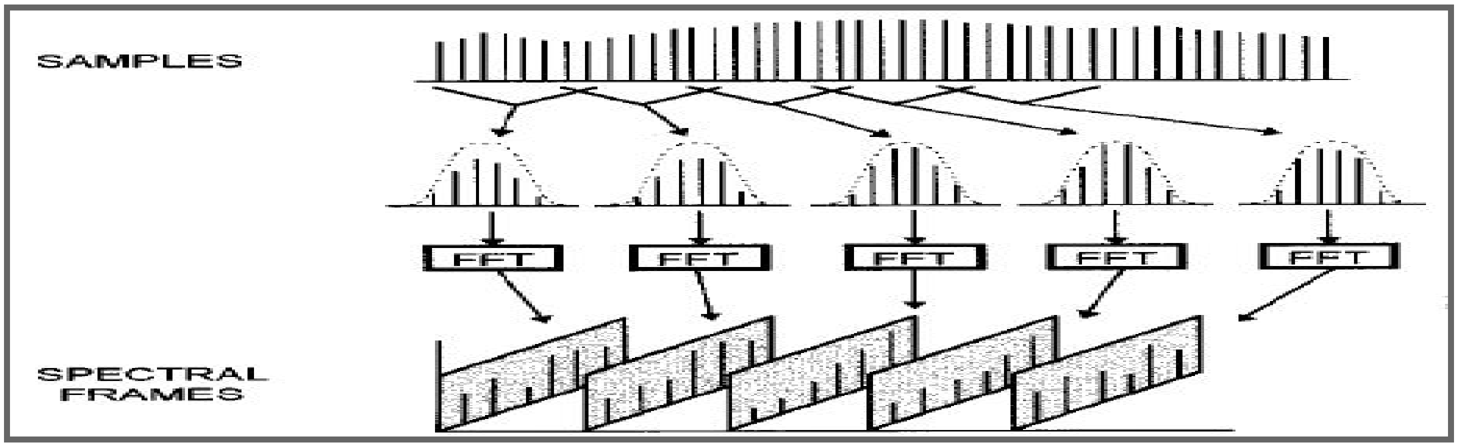
It converts the signal from the time domain into the frequency domain. The result is called a **spectrum**. Basically, Fourier Transform decomposes complex periodic sound into a sum of sine waves oscillating at different frequencies. Following is the spectrum we got after applying Fourier Transformation of audio from blues genre:



But, when we apply FFT, we lose time information. And, if we lose time information then our system won't be able to tell which music sound played first if we use only frequencies as a feature. And thus we will go for the next transformation.

Spectrograms & Short Term Fourier Transform(STFT)

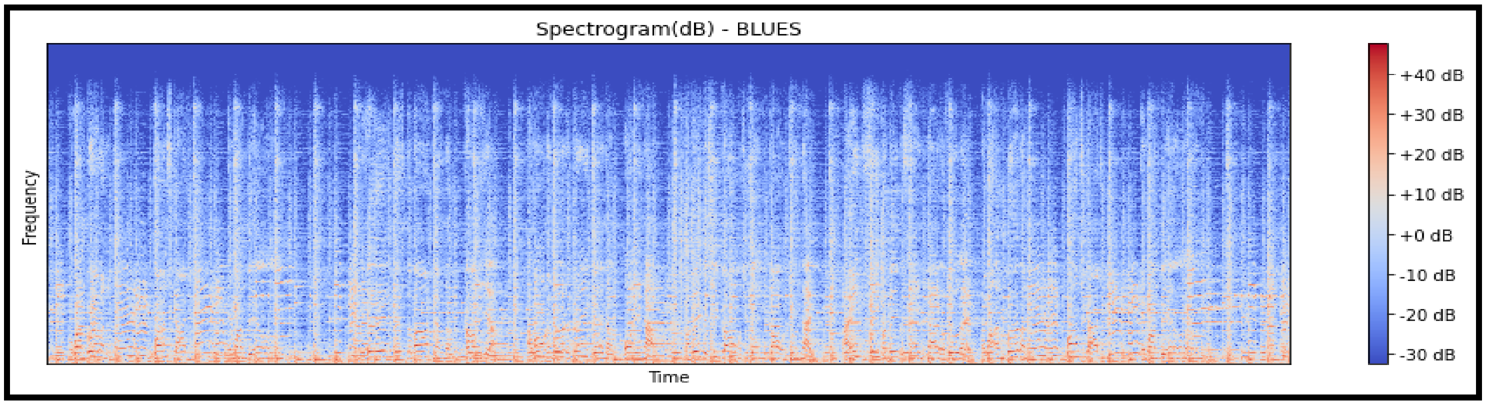
Spectrograms are introduced because we were losing time information after applying Fast Fourier Transformation. A spectrogram shows how the **frequency content of a signal changes over time** and can be calculated from the time domain signal. Short Term Fourier Transform comes to our rescue here.

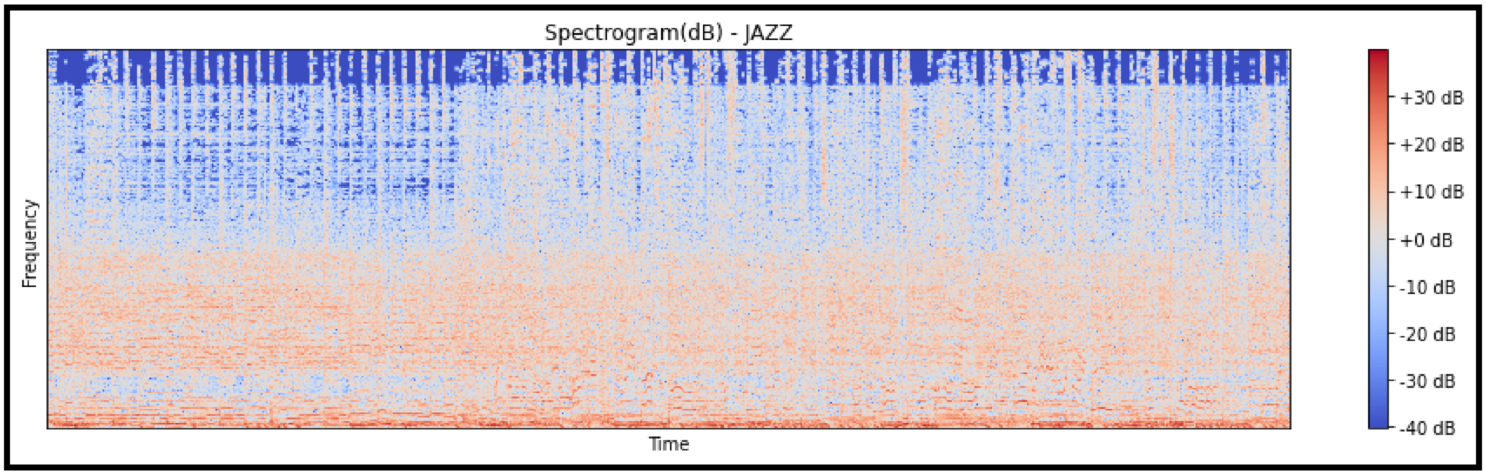


A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. It’s a representation of frequencies changing with respect to time for given music signals.

<u>How Spectrogram Preserves Time Information?</u>

When we perform STFT on any audio data, the first step is to break the entire audio signal into smaller frames (or windows) and then for each window we calculate FFT. This way we will be getting frequencies for each window and window number will represent time information. Generally, window size is between 20ms to 30ms long. These frames must overlap with each other such that we do not lose any information. Generally, overlapping of 25% to 75% is done. Following is the Spectrogram after applying STFT on our one of the audio file from blues, jazz genre:





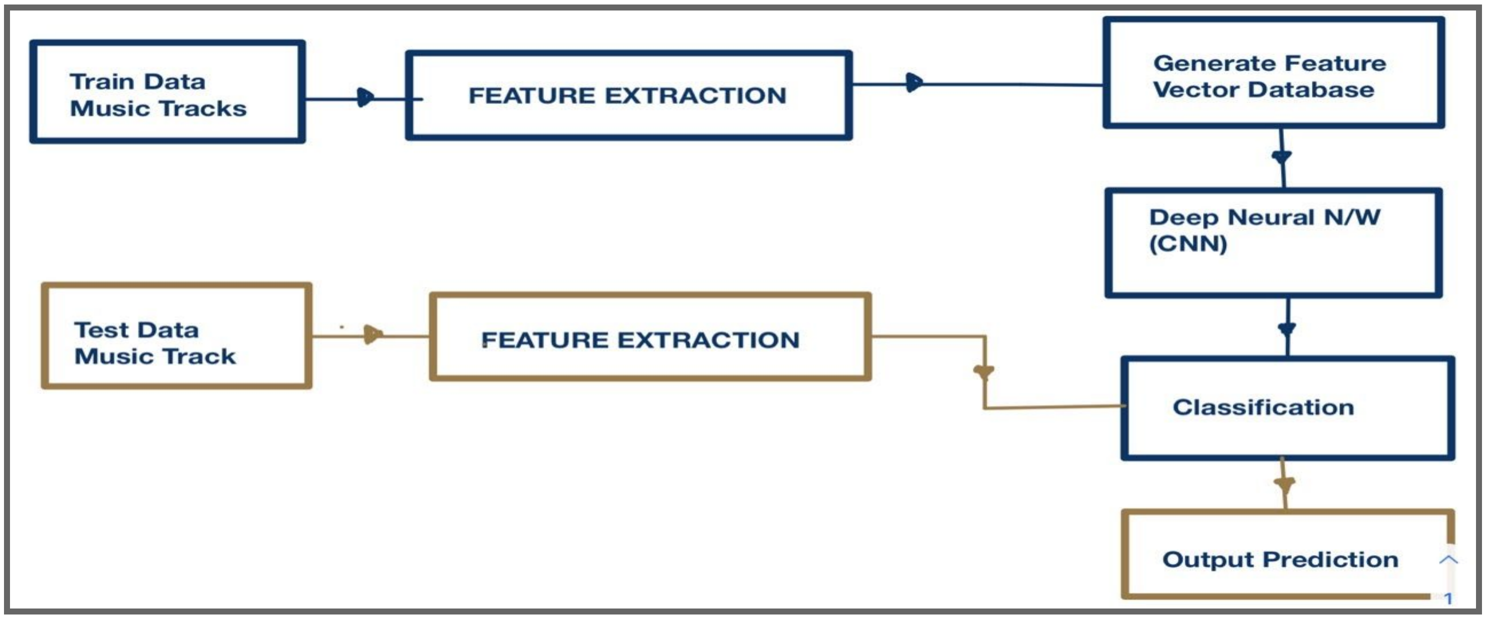
As a result of **STFT**, we got a **spectrogram** which gives us information about **magnitude** as a **function of frequency and time.** Songs that sound drastically different, results in drastically different spectrograms.

Steps involved in extracting features from audio data :

1. We took samples of air pressure over time to digitally represent an audio **signal.**
2. We mapped the audio signal from the **time domain to the frequency domain** using the **fast Fourier transform**, and we performed this on overlapping windowed segments of the audio signal.
3. We converted the y-axis (frequency) to a log scale and the color dimension (amplitude) to decibels to form the **spectrogram**.
4. We mapped the y-axis (frequency) onto the **mel scale** to form the **mel spectrogram**.

C. Pre-Processing & Feature Extraction from Audio Files

Our dataset has 1000 music files and each file is 30 seconds long. Now, to train our CNN model, we have extracted MFCC features from these music files. Entire database of extracted features was created and stored in a ***.json*** file. Following are the operations performed to extract MFCC features from audio file:-



1. Each music file, which is of 30 seconds, is loaded one by one from the respective genre.

2. For each file, we capture number of amplitude samples as:

No. of amplitude samples (in 30 sec file) = sample rate * track duration

= 22050 * 30 = 661500

Sample Rate: number of amplitude samples captured per second.

3 .We divided this 30 seconds music file into small segments, such that each segment will be 3 seconds long. And number of amplitude samples captured for each segment is:

No. of samples = samples captures for one music file / no. of segments

= 66150 / 10 = 66150 amplitude samples per 3 sec segment

4. Now, to extract features from each segment in a way that we also preserve time domain information, we have used frame (or window) size of 2048 samples and hop length of 512 samples.

5. For each segment we have 66150 samples. We apply FFT on the first 2048 amplitude samples (since our frame size is 2048), then we shift out the window by 512 amplitude samples and again calculate FFT for this frame and so on till the end of segment. This way we are ensuring 75% overlapping between each frame so that we do not lose any information and time domain is preserved with window numbering.

6. Now, we extracted MFCC features from the resultant spectrograms.

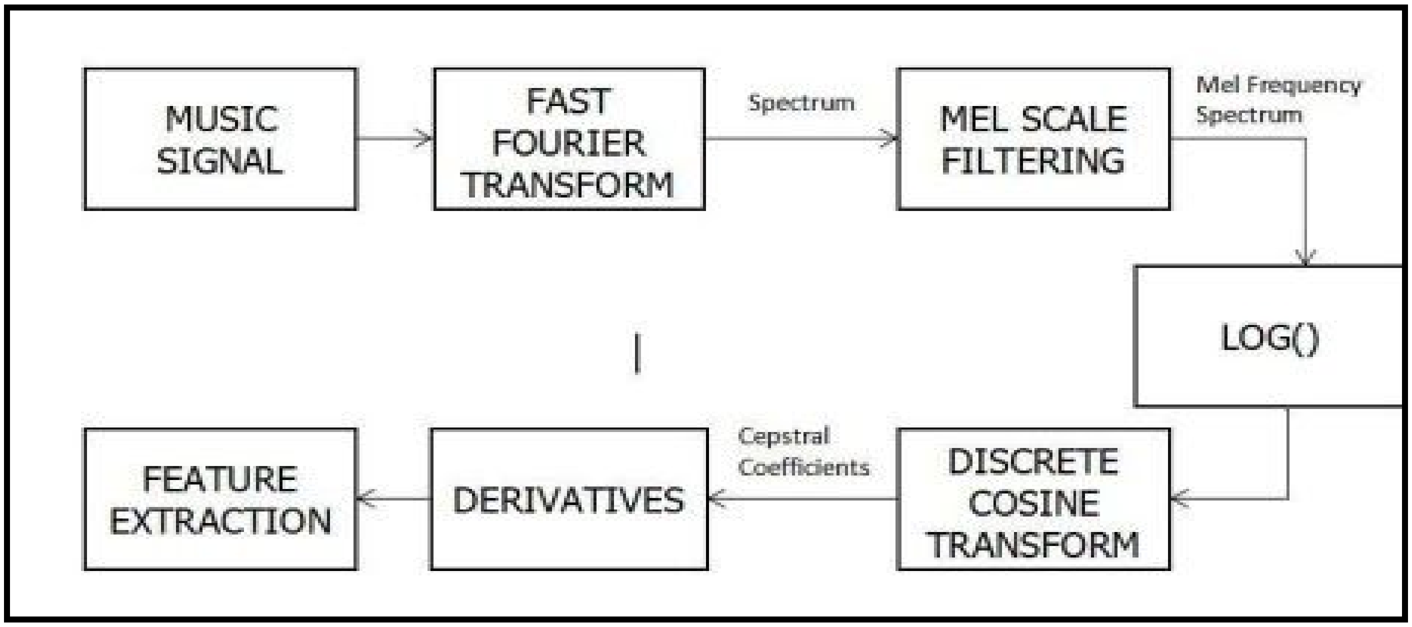
Mel - Frequency Cepstral Coefficient (MFCC)

The Mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope. It models the characteristics of the human voice.

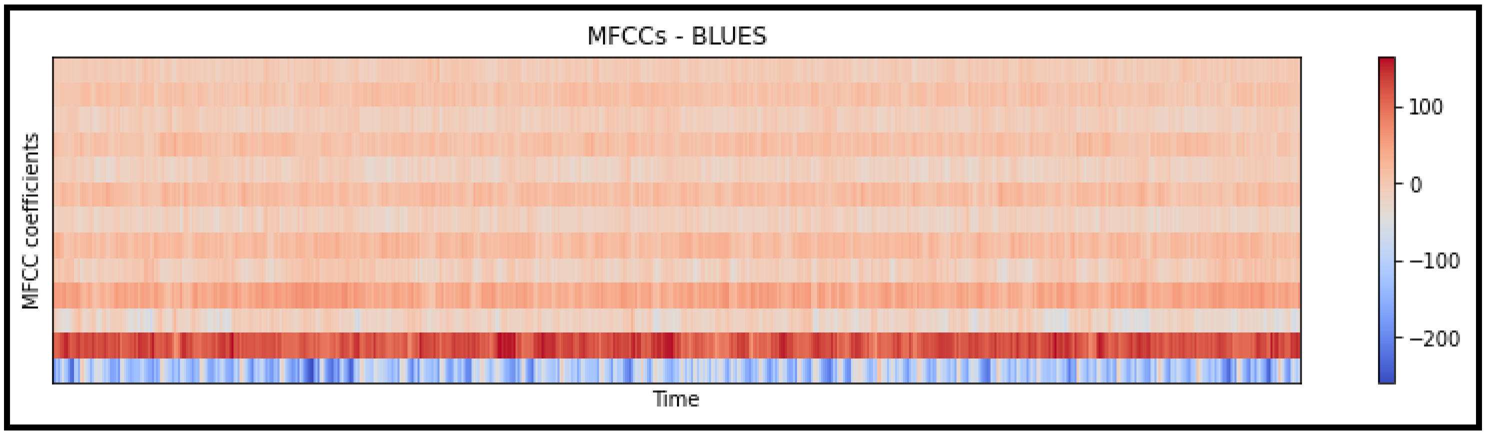
Pitch is one of the characteristics of a speech signal and is measured as the frequency of the signal. ***Mel scale*** is a scale that relates the perceived frequency of a tone to the actual measured frequency. It scales the frequency in order to match more closely what the human ear can hear (humans are better at identifying small changes in speech at lower frequencies). A frequency measured in Hertz (f) is converted to the Mel scale using the following formula :

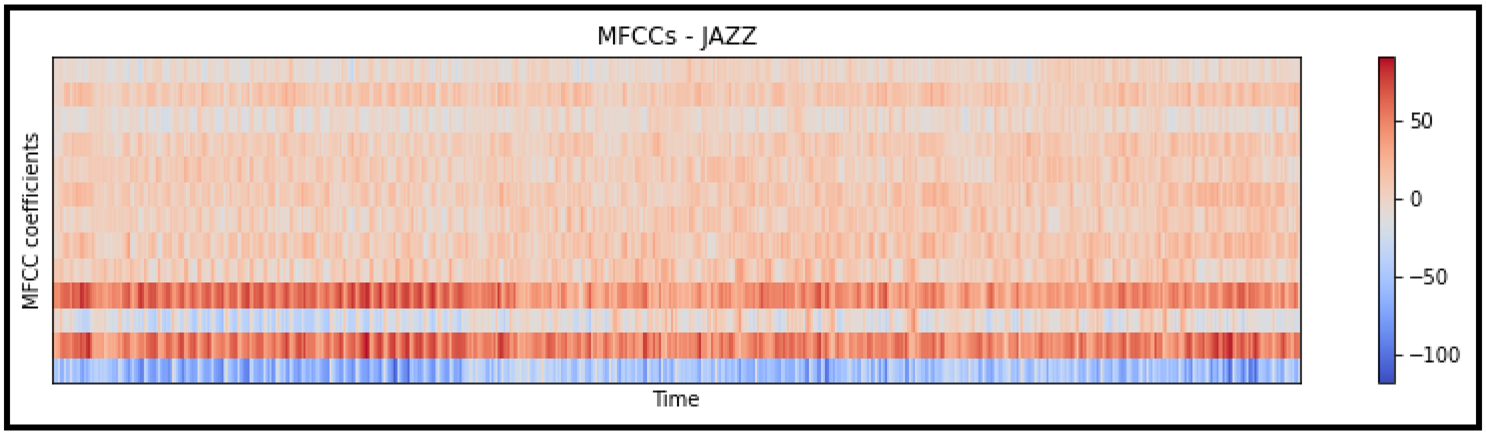
$$\boxed{\mathrm{Mel}(f) = 2595 \log\left(1 + \frac{f}{700}\right)}$$

Any sound generated by humans is determined by the shape of their vocal tract. If this shape can be determined correctly, any sound produced can be accurately represented. The envelope of the time power spectrum of the speech signal is representative of the vocal tract and MFCC (which is nothing but the coefficients that make up the *Mel-frequency cepstrum*) accurately represents this envelope. The following block diagram is a step-wise summary of how we arrived at MFCCs:



After performing all these steps we got the extracted MFCC features in our ***.json*** file. Following are the visual representation of audio file after extracting MFCC:

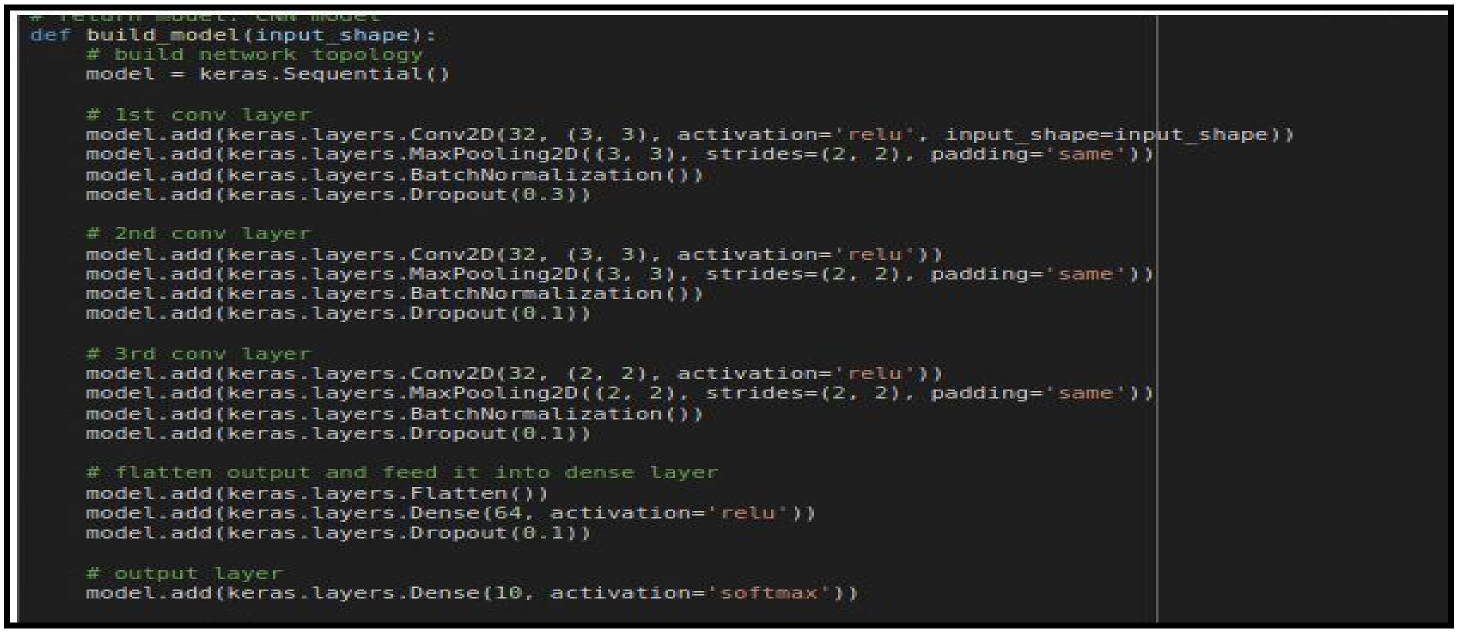




D. Convolutional Neural Network

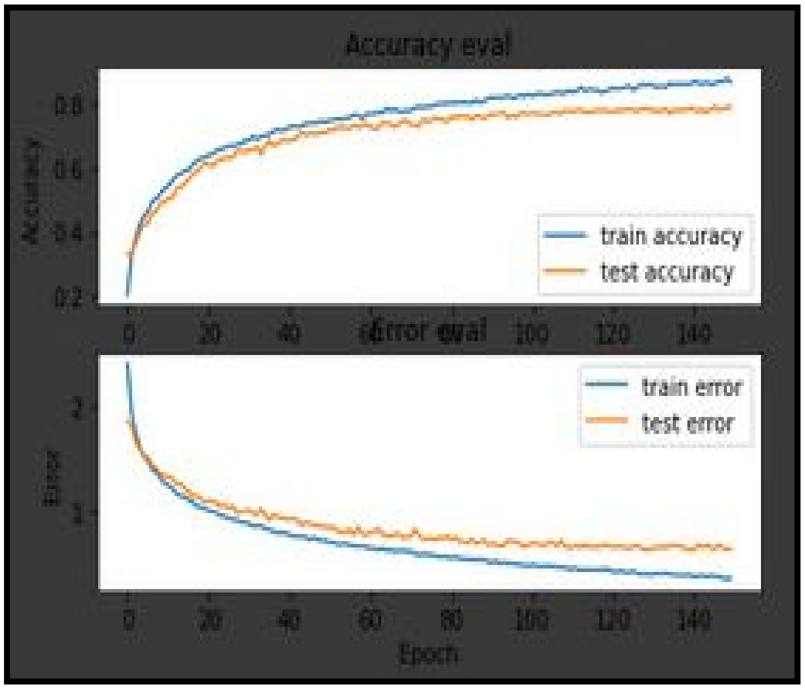
CNN is a Deep Learning algorithm which can take an input image as input, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. A CNN has various layers such as Convolutional layers, ReLU layers, Pooling layers and Fully connected dense layer. CNN is widely used for image classification because it does automatic feature extraction using convolution. Following are the details of our CNN model :

Code Snapshot for CNN model :



```python
def build_model(input_shape):
    # build network topology
    model = keras.Sequential()

    # 1st conv layer
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.3))

    # 2nd conv layer
    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.1))

    # 3rd conv layer
    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.1))

    # flatten output and feed it into dense layer
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.1))

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

Parameter details and outcomes:

	**Train Data : Test Data :: 80 : 20** **Validation data = 20% of Train Data** **Batch :** 32 **Epochs** : 150 **Learning Rate** : 0.0001 **Training Accuracy : 85.91%** **Validation Accuracy : 78.79%** **Testing Accuracy : 81.3%**

CONCLUSION AND FUTURE WORK

This project implemented the application which performs Music Genre Classification using Deep Learning techniques. The application uses a Convolutional Neural Network model to perform the classification. MFCC features of each track from the GTZAN dataset are obtained. This is done by using the librosa package of python. **After training the model for 150 epochs we got the training accuracy of 85.91% and test accuracy of 81.3%.**

The extension of this work would be to consider bigger datasets like **Free Music Archive** which offers 106,000 tracks ***(around 250 GB)*** with labeled genres and also different formats of tracks other than wav format like (mp3, au etc). Also, with time the style represented by each genre will continue to change. So the objective for the future will be to stay updated with the change in styles of genres and extend our software to work on these updated styles. This work can also be extended to work as a music recommendation system depending on the mood of the person.

REFERENCES

- **Cook and Tzanetakis's GTZAN dataset :** *https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification*
- **Vishnupriya S and K Meenakshi[2018] work using CNN :** https://ieeexplore.ieee.org/document/8441340
- **Beici Liang and Minwei Gu[2020] work using Transfer Learning :** https://ieeexplore.ieee.org/abstract/document/9175547
- **Towards datascience blog on Musical Genre Classification :** https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8