MUSIC GENRE CLASSIFICATION USING DEEP LEARNING

Project report submitted in partial fulfillment for the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Priyanshi (191342)

Under the supervision of

Dr. Pardeep Kumar

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CERTIFICATE

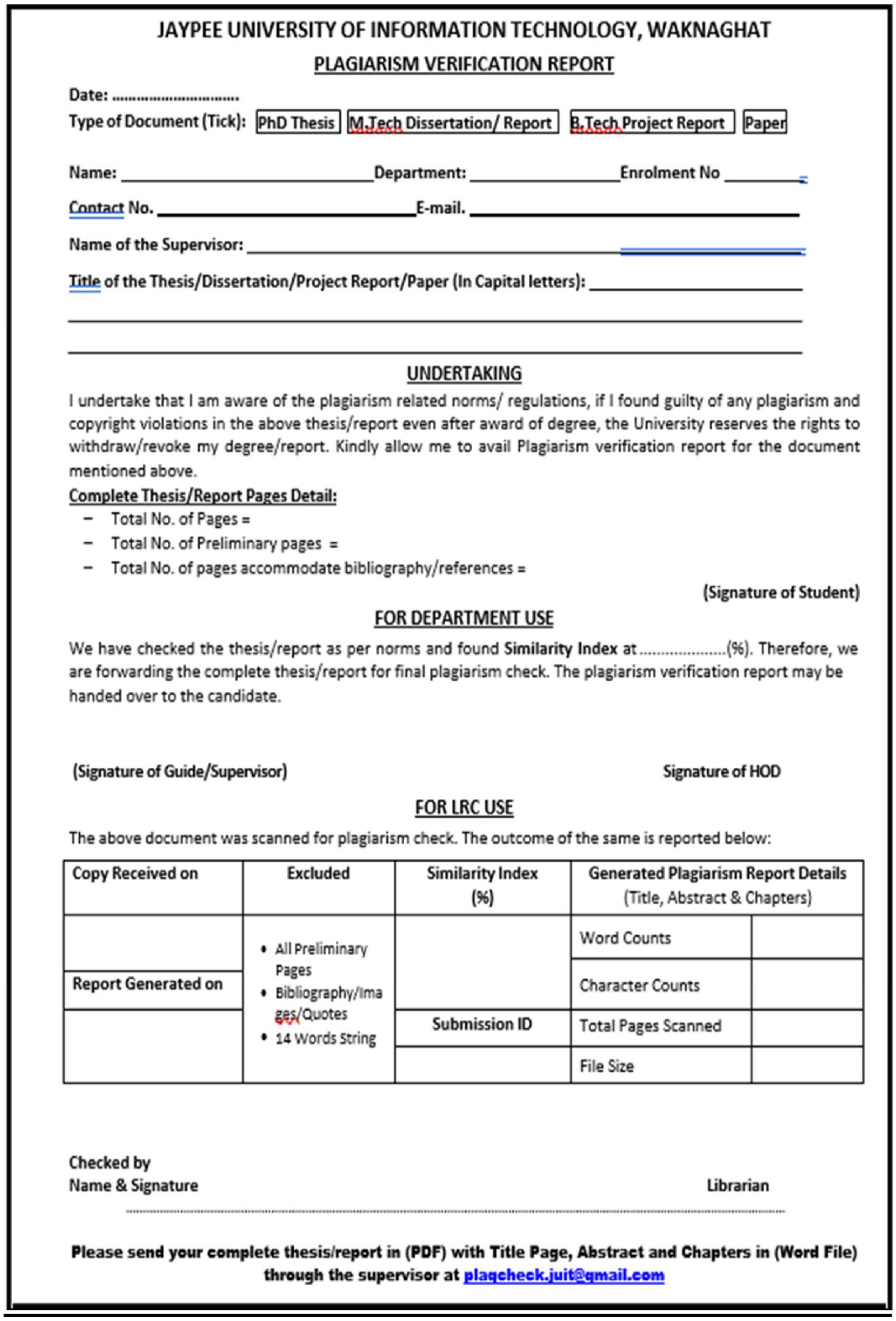
This is to certify that the work which is being presented in the project report titled "Music genre classification using Deep Learning" in partial fulfilment of the requirements for the award of the degree of B. Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by "Priyanshi,191342" during the period from January 2022 to May 2022 under the supervision of Dr. Pardeep kumar, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Priyanshi
(191342)

The above statement made is correct to the best of my knowledge.

(Dr.PardeepKumar)
Associate Professor
Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat, Solan, HP.

PLAGIARISM CERTIFICATE



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: ____________________________ **Department:** ________________ **Enrolment No** _________

Contact No. ____________________________ **E-mail.** ____________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _______________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at....................(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Dr. Pardeep Kumar**, Department of CSE Jaypee University of Information Technology,Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of "**Music genre classification using Deep Learning**" to carry out thisproject. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

It is my radiant sentiment to place on record my best regards, deepest sense of gratitude to all those individuals who have helped me straightforwardly or in a roundabout way in making this internship easier. In this unique situation, I might want to thank the various team members, colleagues , who have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Priyanshi
(191342)

TABLE OF CONTENT

LIST OF ABBREVIATIONS

S.No	**Abbreviation**	**Full Form**
1.	ANN	Artificial Neural Network
2.	CNN	Convolutional Neural Network
3.	DNN	Deep Neural Network
4.	ML	Machine Learning
5.	DL	Deep Learning
6.	dB	decibela
7.	FFT	Fast Fourier Transform
8.	STFT	Short-time Fourier transform
9	MFCC	Mel-frequency cepstral coefficients

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Today, people's daily lives include music in a very significant way. The complexity of the creation of music has gradually lessened, more people are producing music and uploading it to streaming services. People now spend a lot of time searching for particular music because of the massive music streaming industry. Therefore, in today's society, the ability to quickly classify musical genres is crucial. People have various musical tastes because there are several music genres and these genres differ from one another. As a result, categorizing music and recommending new music to users is a crucial and modern challenge. One solution to this problem can be classifying music by their genre . There are a number of approaches for music classification and recommendation, in this study, focus is on extracting representative features obtained by a novel neural network model , have been proposed. Audio properties obtained via these networks have been used to classify music genres on a dataset. MFCC was basis of our feature extraction ,Librosa library of python was used to implement that ,two neural network were implemented DNN and CNN and the dataset used was GTZAN dataset.

KEYWORDS: Music, Genre, Classification, MFCC, Librosa, Spectral features, GTZAN genre dataset

Chapter 01: INTRODUCTION

1.1 Introduction

When compared to image processing and other classification methods, audio processing is one of the data science's most difficult challenges.One such application is music genre classification, it is a piece of software that determines the genre of music in audio format.These devices are utilized for things like automatically labeling music for services like Spotify and Billboard, as well as choosing appropriate background music for events. As classifying music manually requires listening to each track for the full duration, the application is crucial and needs automation to cut down on manual error and time. Therefore, machine learning and deep learning algorithms are what we will make use of in this report to automate the procedure.

Manual genre classification is carried out by people with personal musical expertise. Traditional computer techniques have not yet been successful in automating this process since the distinctions between musical genres are arbitrary and ill-defined. The uncertainty of genre classification, on the other hand, makes machine intelligence ideally adapted to this task. Given enough audio data, which can be easily gathered from publicly available music on the internet, machine learning can recognize and anticipate these ill-defined patterns.

A concept described as music genre classification, which divides music into numerous categories or genres, enables the general public to distinguish between two genres based on their composition or the beats they contain . As more and more musical subgenres appear all over the world, the idea of classifying music according to genre has recently gained a lot of popularity. Music fans can listen to tracks based on their preferences and depend on the technology of music genre classification for everything from K-Pop to Jazz.

Machine learning-based music genre classification is a relatively new idea that has recently come to light. Although the world has been aware of music genres for many years, computers are now able to classify music genres in the modern world where everyone is enjoying music. Machine learning has made it easier to categorise music genres using a variety of methods.IN this study, we'll talk about the two machine learning method that are used to identify musical genres first method used is Deep Neural Network(DNN) that is nothing but Artificial Neural Network(ANN) with additional hidden layer and second and lastly CNN is used. The CNN method in machine learning suggests that computers perceive and process visual images as a basis for predicting future datasets. CNN enables computers to analyze spectrograms (visual graphs of musical frequencies), which in turn aids machines in determining the genre of a particular piece of music when it comes to music genre categorization using neural networks.The dataset used is GTZAN dataset it has sample of 10 different genre, figure 1.1 display the waveplot of all 10 genre, then for the purpose of feature extraction MFCCs is used ,that's indentify and divide the audio into some feautres like amplitude ,frequency etc , its provide total of 39 feautres that are related to amplitude and frequency then was stored as Json file so that it can used as input for the deep learning models as CNN basically works with images. In short DNN and CNN will be used to classify music gerne and will thest there accuracy on the dataset.

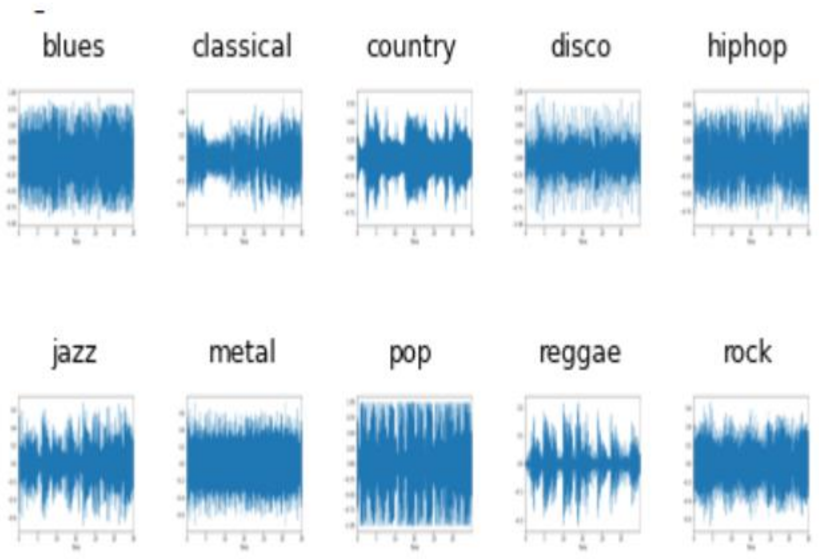


Figure 1.1 waveplot of different genre

The goal of this project is to develop a proof-of-concept music genre classifier that can accurately determine the genre and confidence level of Western music from four candidate genres (classical, jazz, rap, rock, metal, country, etc.) using deep learning approach.

1.2 Problem Statement

Improving music recommendation services and enabling customized user experiences by developing a model to quickly and accurately classify music genre based on audio data.

1. To create a machine learning model that categorizes music according to its genre.
2. To determine the differences in accuracy between this machine learning model and the previous models and come to the appropriate conclusions.

1.3 Objectives

The goal of using machine learning to classify music genres is to create models that can accurately and efficiently sort music into different categories based on audio features. This has a few key benefits. First, it can improve music recommendation systems by offering personalized suggestions to users, which can enhance their listening experience. By accurately classifying music genres, the model can suggest similar songs or artists within the preferred genre, helping users discover new music they might enjoy.

Secondly, music genre classification can help organize and manage large music libraries more effectively. By automatically labeling songs with genre tags, music platforms can offer users intuitive search and browsing options, making it easier to find and access music based on their preferred genres. This objective aims to enhance user convenience and accessibility to diverse music options.

Categorizing musical genres helps in the study of market trends and analysis in the music industry. Experts in the field can gather data on user preferences and distribution of music genres across different platforms, which helps them make informed decisions on marketing strategies, content selection, and artist promotion. By employing machine learning, music classification serves different objectives such as enhancing the user experience, organizing music libraries and offering industry insights. in solving complex classification problems

1.4 Methodology Used

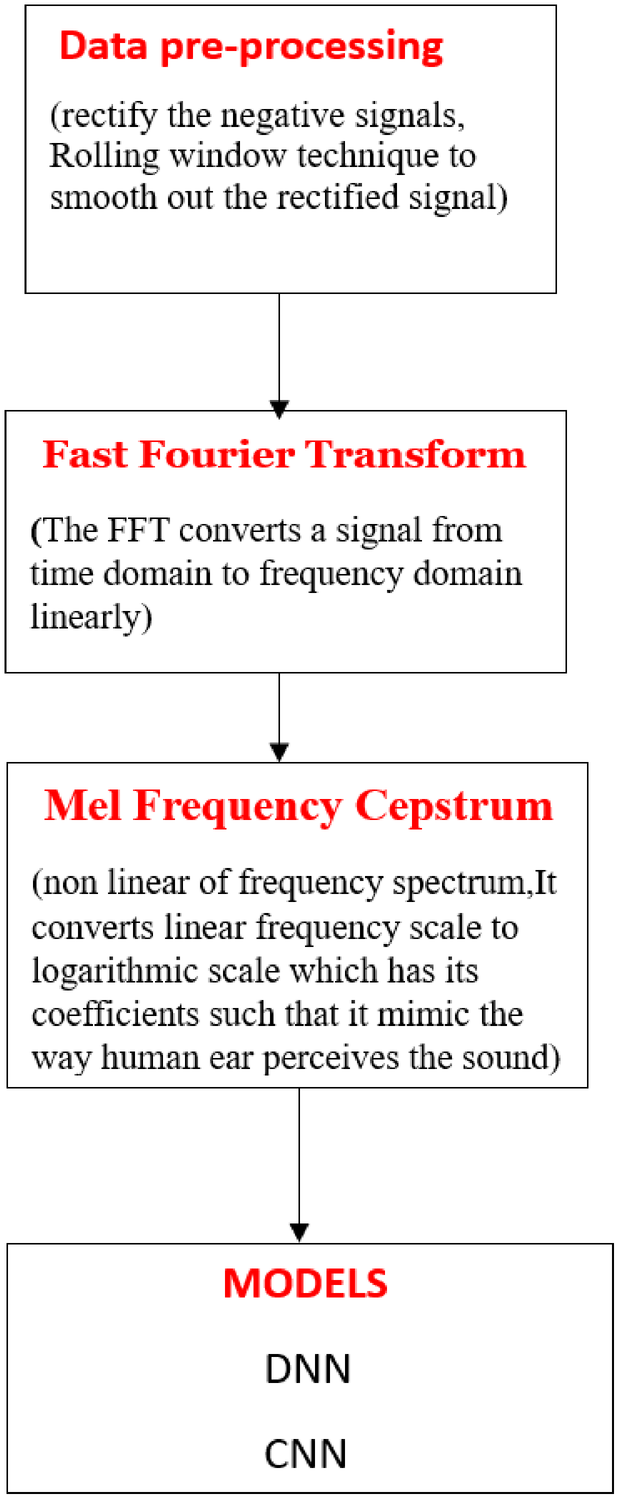

Data pre-processing
(rectify the negative signals, Rolling window technique to smooth out the rectified signal)
Fast Fourier Transform
(The FFT converts a signal from time domain to frequency domain linearly)
Mel Frequency Cepstrum
(non linear of frequency spectrum,It converts linear frequency scale to logarithmic scale which has its coefficients such that it mimic the way human ear perceives the sound)
MODELS
DNN
CNN

Figure 1.2 Methodology used

1.5 Language Used

Guido Van Rossum developed Python, a high-level, interpretative general-purpose programming language, which was released in 1991. Python's design heavily prioritises readability of its code and makes liberal use of whitespace. Its language features and object-oriented approach are designed to aid programmers in writing clear, logical code for both small and large projects. Python has garbage collection and dynamic typing. Procedural, object-oriented, and functional programming paradigms can all be used to create programmes.

1.6 Technical Requirements (Hardware)

Processor: Intel core i5 or above. 64-bit, quad-core, 2.5 GHz minimum per core

Ram: 4 GB or more

Hard disk: 10 GB of available space or more.

Display: Dual XGA (1024 x 768) or higher resolution monitors

Operating system: Windows

1.7 Organization

The remaining part of the academic paper is ordered as follows:

Chapter 2 we have presented the literature survey which depicts the various approaches used by authors to classify music gerne .

Chapter 3 highlights the methodology and system development of the project as well as tried to explain the dataset i.e GTZAN dataset in the best possible manner It illustrates the project's several computational, experimental, and mathematical aspects. We have also concentrated on the hardware and software platforms required for the model's implementation.

In chapter 4 the project's performance analysis is done, that outlines the project's reliability.

Chapter 5 presents the conclusions of the project and the observations seen in the results. It also provides the applications of the project and the future scope of the same.

Chapter 02: Literature Survey

Music genre classification is a fundamental task in music information retrieval (MIR) that aims to automatically categorize music tracks into specific genres.Deep learning algorithms for music genre classification have been the subject of research in recent years. In order to create more precise and reliable genre classification models, these strategies are able to extract high-level features from audio data. This overview of the literature examines many studies that have investigated Machine learning for the classification of musical genres in more detail. We can learn more about these techniques' potential and how they might enhance the precision of genre classification in the music industry by looking at these studies.

Various studies have been conducted to classify music on the basis of their genre. One such study by Nikki Pelchat; Craig M Gelowitz [1] used neural network to perform music genre classification ,dataset was made from a music library of 1880 music that were divided in seven genre. The research was implemented by dividing songs into time segments and using those as time representations.
Segments are determined by the spectrograms for each one. These spectrograms were each assigned a music genre before being utilized as inputs for a neural network. The neural network used four convolutional layers, a fully connected layer, a softmax function, and a one-hot array of genre classifications to determine the likelihood that each genre would be recognized. On the testing data, the initial results were 67% accuracy.

Another study by Rene Josiah M. Quinto, Rowel O. Atienza, and Nestor Michael C. Tiglao [2] did classification sub-gerne using different types of machine learning models. The following most popular machine learning techniques for genre classification were used, Neural Networks, SVM, and KNN, achieved maximum accuracy of 79.39%, 81.67%, and 77.43%, respectively, on a dataset of three jazz music subgenres preprocessed using their Mel-Frequency Cepstral Coefficients (MFCC) as in [6]. A single-layered Long Short-Term Memory (LSTM) network achieved an accuracy of 80.30%, and adding a multi-layer per increased the accuracy to 89%.Similarly in [5] Jash Mehta, Deep Gandhi, Govind Thakur, Pratik Kanani compared four transfer learning architectures, Resnet34, Resnet50, VGG16, and AlexNet, which were ImageNet's top-performing models at various points in time, to classify various songs according to their genre.In [3] Prasenjeet Fulzele, Rajat Singh, Naman Kaushik, Kavita Pandey made a A Hybrid Model For Music Genre Classification Using LSTM And SVM, LSTM Neural Network is used take advantage of the time-dependent character of the dataset. The prediction accuracy of the individual models improved as a result of the hybrid model comprising these two classifiers. The findings of the solo LSTM and SVM models are contrasted with those of the hybrid model, which is applied to the GTZAN music dataset. The proposed model's accuracy of 89% was higher than the LSTM and SVM classifiers' individual accuracy levels, reaffirming the effective use of both classifier.

The study by Yu-Huei Cheng; Pang-Ching Chang; Che-Nan Kuo [4] implemented a music genre classification model using CNN's audio benefits and characteristics. Librosa is used in the pre-processing to transform the original audio files into the matching Mel spectrums. The suggested CNN model is then trained using the converted Mel spectrum. The 10 classifiers use majority voting to make decisions, and the average accuracy on the GTZAN dataset was 84%. Deep Neural Networks, also known as Multilayer Perceptrons (MLP), appear to have the best performance in the research produced by Haggblade et al. [7], with an average accuracy of 96%.Support Vector Machine (SVM) and K-Nearest Neighbours (KNN) tied for second place with an average accuracy of almost 87% on the GTZAN dataset. Kozakowski and Michalak's Deep Sound [8], which utilized both convolutional neural network models (CNN) and LSTM Networks, is another study using the GTZAN dataset. They classified 10 genres with a 67% accuracy rate. When compared to other studies that used the same dataset and achieved > 70%, this result is quite low.

In [9] Akshi Kumar,Arjun Rajpal and Dushyant Rathore, for the classification process, two different types of data—lyrics and album artwork—are used. To process the lyrical data, two Natural Language Processing techniques—bag-Of-Words and Term Frequency-Inverse Document Frequency (TFIDF)—are employed. To identify the genre, we use Deep Convolutional Neural Network (CNN) on the album artwork and machine learning methods including Random Forest, Support Vector Machine (SVM), Naive Bayes, Linear Support Vector Classifier (Linear SVC), and eXtreme Gradient Boosting (XGBoost) on lyrical data. On the basis of lyrical data received from bag-Of-Words, machine learning techniques are used, and a mean precision of 75.66% and a mean f-score of 75.22% are obtained. A mean precision of 76.85% and a mean f-score of 77.38% are obtained by using the same set of algorithms to lyrical data acquired from TFIDF. For bag-Of-Words and TFIDF, respectively, XGBoost beats all other methods, providing maximum precision of 79.30% and 80.16% and maximum f-scores of 79.6% and 84.09%, respectively. A precision of 82.46% and an f-score of 81.84% are obtained when deep neural networks are applied to album art. A hybrid machine proposed in [11] feature extraction is done using Deep Belief Network (DBN) on Discrete Fourier Transforms (DFTs). and used a nonlinear Support Vector Machine (SVM) with a radial basis function kernel that lead to an accuracy of 84.3%.

Nirmal M. R. carried out the research [13]. They did use a spectrogram, which shows how the frequency spectrum of a signal changes over time. A signal's Short Time Fourier Transform (STFT) is shown in a spectrogram. Graphs called spectrograms can show data in both the time and frequency domains. The results of two models used to implement the convolutional neural network are compared in [13]. The user-defined CNN model and MobileNet had classification accuracy of 67% and 40%, respectively.

In [14] Yu et al proposed to construct an other RNN engineering - abidirectional RNN (BRNN) - to group music classes utilizing Gated Intermittent Unit (or GRU). After reducing the number of gate categories, GRU outperforms Long-Short Term Memory (LSTM) in terms of performance in the BRNN architecture . One of the stacked bidirectional RNN (BRNN) layers in this model moved forward from the beginning to the end of the sequence, while the other moved backward . To detect vanishing gradients without requiring any computational complexity, shortcut connections overlooking a hidden BRNN layer were demonstrated .

Tzanetakis and Cook [17] applied time-recurrence (TF) examination techniques like Mell Recurrence Cepstral Coefficient (MFCC), Otherworldly Centroid and wavelet changes in highlight extraction step on GTZAN informational index, which is a benchmark informational collection utilized in music data recovery music handling studies. Tzanetakis and his colleagues accurately classified the GTZAN with these acoustic features. Tzanetakis and others gotten one more score as 79.5% involving Daubechies wavelet coefficient histogram as a best performing element and backing vector machine (SVM) as a classifier [20].

The deep communication network-based music genre recognition and classification algorithm is examined in this paper [25]. The network will automatically begin to learn and analyze the input characteristic data as long as the bottom characteristics of the music signal, such as multilinear principal component (MPC), fast Fourier transform (FFT), Malta Fairs, and convention center (MFCC), are present. This will allow the network to obtain more appropriate abstract characteristics that capture the essence of each genre of music. As a result, music genre recognition and classification performs better and is more intelligent, and the complexity of feature extraction is greatly reduced [29].

In [31] In the 10-genre classification task, MFCC (Mel-frequency cepstral coefficients), texture, beat, and other human-engineered features typically achieve an accuracy of 61%. They utilized a "partition and-prevail" strategy to tackle the issue: They predicted each three-second segment of the music signal's spectrogram, combined the predictions for each segment, and so forth. 6] utilized the mel-spectrogram as the contribution to the CNN to additionally diminish the component of the spectrogram. For Expectation and Preparing 1000s of music tracks (changed over completely to Mel-Spectrogram) are equally separated into preparing, approval, and testing sets in a 5:2:3 proportion. During testing, all music (Mel-Spectrogram) is partitioned into 3-second fragments with half cross-over. The prepared brain network then, at that point, predicts the probabilities of every class for each section in [6]. In the 10-genre classification, their model is the first to achieve human-level accuracy of 70%.

Few of the research papers studied for this project have been reviewed below in Table 1.Few of the research papers studied for this project have been reviewed below in Table 1.

Few of the research papers studied for this project have been reviewed below in Table 1.

Table I: Literature Review

S. No.	Author(s)	Contribution	Dataset Used	Limitations
1	Nikki Pelchat, Craig M Gelowitz [1]	Neural Network Music Genre Classification	music library of 1880 songs categorized into seven genres was used.	The model Was overfit.
2	Rene Josiah M. Quinto, Rowel O.Atienza, Nestor Michael C. Tiglao [2]	Jazz music sub-genre classification using deep learning	Dataset of jazz sub-genres	Difficulty in capturing subtle and subjective differences between closely related sub-genres.
3	Prasenjeet Fulzele, Rajat Singh, Naman Kaushik, Kavita Pandey [3]	A Hybrid Model For Music Genre Classificatio n Using LSTM And SVM	GTZAN dataset	It didn't evaluated the model's performance using a metrics like recall, F1 score , AUC-ROC.
4	Yu-Huei Cheng, Pang-Ching Chang, Che-Nan Kuo [4]	Convolutional Neural Networks Approach for Music Genre Classification	GTZAN dataset	Lack of interpretability , making it challenging to understand how and why the model makes specific predictions.

5	Jash Mehta, Deep Gandhi, Govind Thakur, Pratik Kanani [5]	Music Genre Classification using Transfer Learningon log-based MEL Spectrogram	GTZAN dataset	Performance Variation across genres as it is challenging to classify accurately due to their overlapping characteristics such as rock and country

Chapter 03: System Development

3.1 Computational

High configuration GPUs are used for training the model. These are available online and also available on one's system. The training time is dependent on the GPU. GPUs with higher memory like 4-16 GB are recommended for such applications. Software's like Jupyter notebook are preferred but applications like PyCharm and VScode can also be used along with python libraries like NumPy, Keras and TensorFlow. For our project, we used Jupyter to write and execute code in python language, and it is well suited to machine learning, deep learning, data analysis and education. Experiments were run on Jupyter with Intel® Xeon ® processor at 2.20GHz using 12.72 GB of RAM coupled with a Nvidia Tesla T4.

3.2 System Design

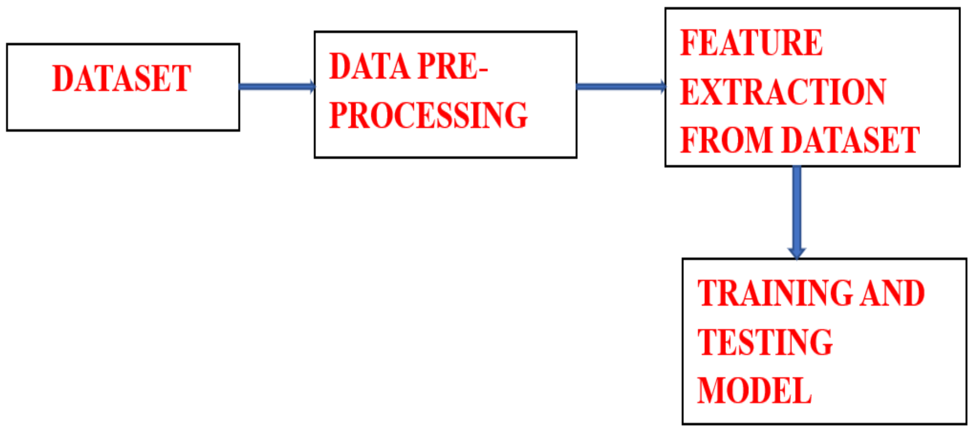


Figure 3.1 system design

3.3 Dataset Used

The GTZAN genre classification dataset, a very well-liked audio collection dataset, is the one we used. It has about 1000 audio tracks in all, divided into ten main categories.

The.wav format (extension) is used for all audio files.

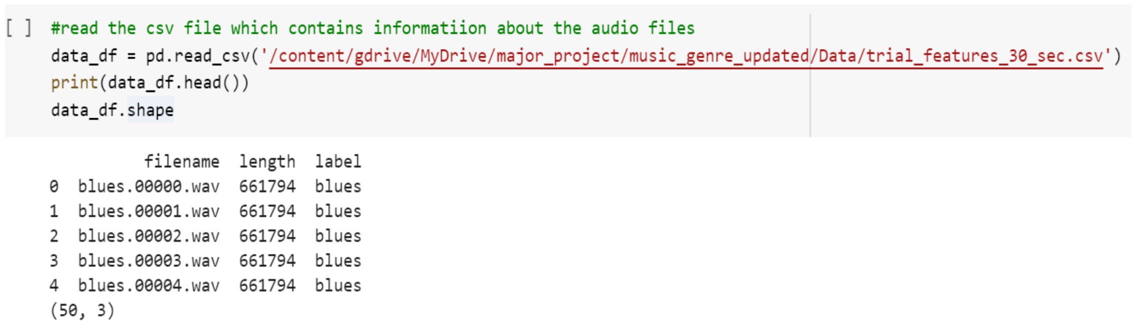
The 10 genres are as follows :

- Blues
- Classical
- Country
- Disco
- Hip-Hop
- Jazz
- Metal
- Pop
- Reggae
- Rock

3.4 Date Set Features

The GTZAN dataset is a collection of audio clips from ten different music genres, including blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. Each genre has 100 audio clips that are 30 seconds long. The audio files are in the WAV format, with a sampling rate of 44.1 kHz and a bit depth of 16 bits.

Researchers often use this dataset to develop and evaluate music genre classification algorithms. To do this, they extract various audio features like Mel-frequency cepstral coefficients (MFCCs), spectral features, chroma features, etc. The dataset is organized into separate folders for each music genre, and it's common to split it into training and testing sets for model development and evaluation. The recommended split is 70% for training and 30% for testing, ensuring a fair representation of genres in both sets.Despite some limitations, like mislabeled audio clips and a relatively small size, the GTZAN dataset remains a popular and valuable resource for exploring audio signal processing and music genre classification techniques.



```
[ ]  #read the csv file which contains informatiion about the audio files
     data_df = pd.read_csv('/content/gdrive/MyDrive/major_project/music_genre_updated/Data/trial_features_30_sec.csv')
     print(data_df.head())
     data_df.shape

              filename  length  label
     0  blues.00000.wav  661794  blues
     1  blues.00001.wav  661794  blues
     2  blues.00002.wav  661794  blues
     3  blues.00003.wav  661794  blues
     4  blues.00004.wav  661794  blues
     (50, 3)
```

Figure 3.2 Data set features shown through code

3.5 Data-Flow Diagram (DFD)

Figure 3.3 presents the proposed methodology. In this study, we will classify music in ten different genre that are present in the dataset .

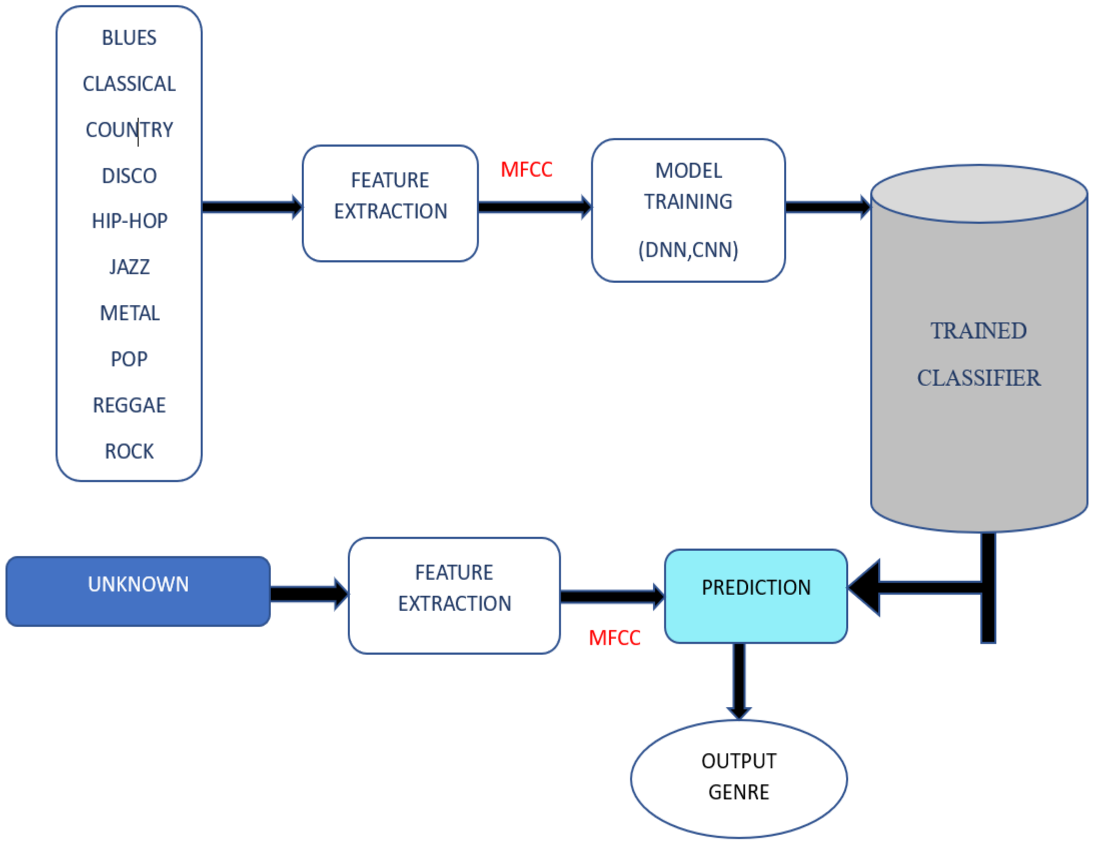


Figure 3.3 Data flow diagram

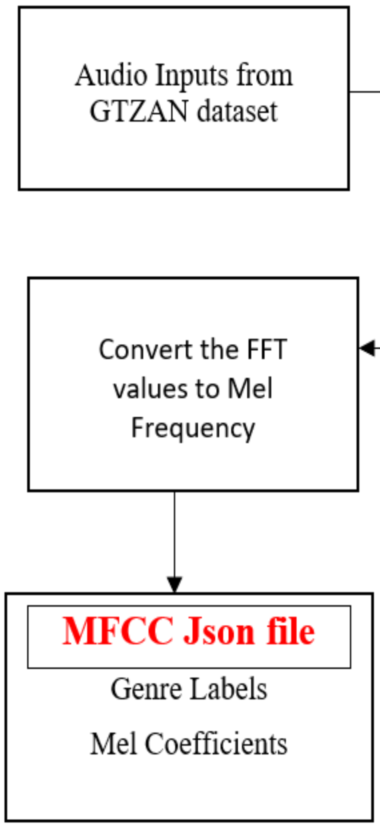
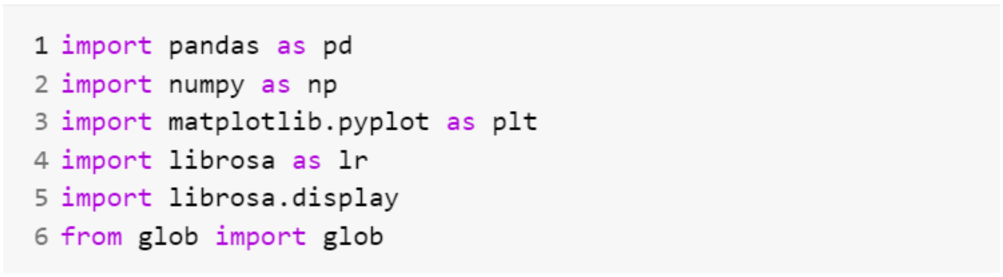

Audio Inputs from GTZAN dataset
Splitting Audio Files into Smaller Time Samples
Convert to Fast-Fourier transform(FFT)
Convert the FFT values to Mel Frequency
MFCC Json file
Genre Labels
Mel Coefficients

Figure 3.4 step involved in feature extraction

3.6 Screenshots and explanation of the various stages of the Project

A. Importing libraries required and uploading the dataset

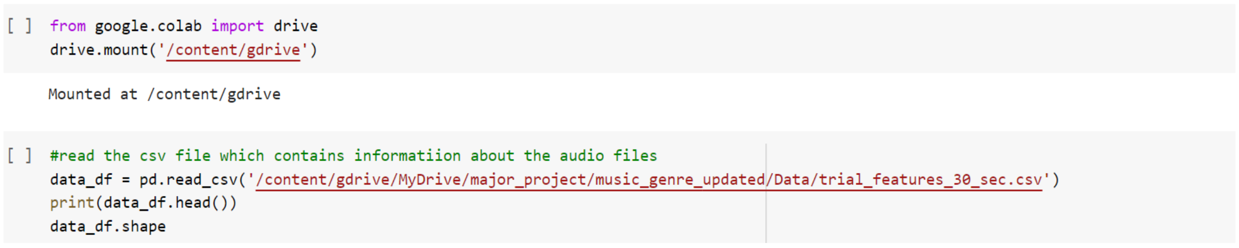


```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import librosa as lr
import librosa.display
from glob import glob
```

Figure 3.5 Importing libraries

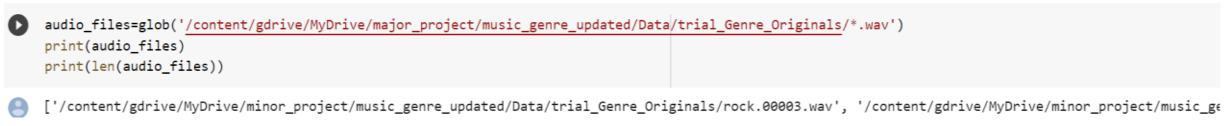
Librosa Python package designed more for music analysis and audio signal analysis. It contains all of the necessary building blocks for a MIR (Music Information Retrieval) system.

Glob :glob (short for global) is used to return all file paths that match a specific pattern.



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

     Mounted at /content/gdrive

[ ]  #read the csv file which contains informatiion about the audio files
     data_df = pd.read_csv('/content/gdrive/MyDrive/major_project/music_genre_updated/Data/trial_features_30_sec.csv')
     print(data_df.head())
     data_df.shape
```

Figure 3.6 Read csv file



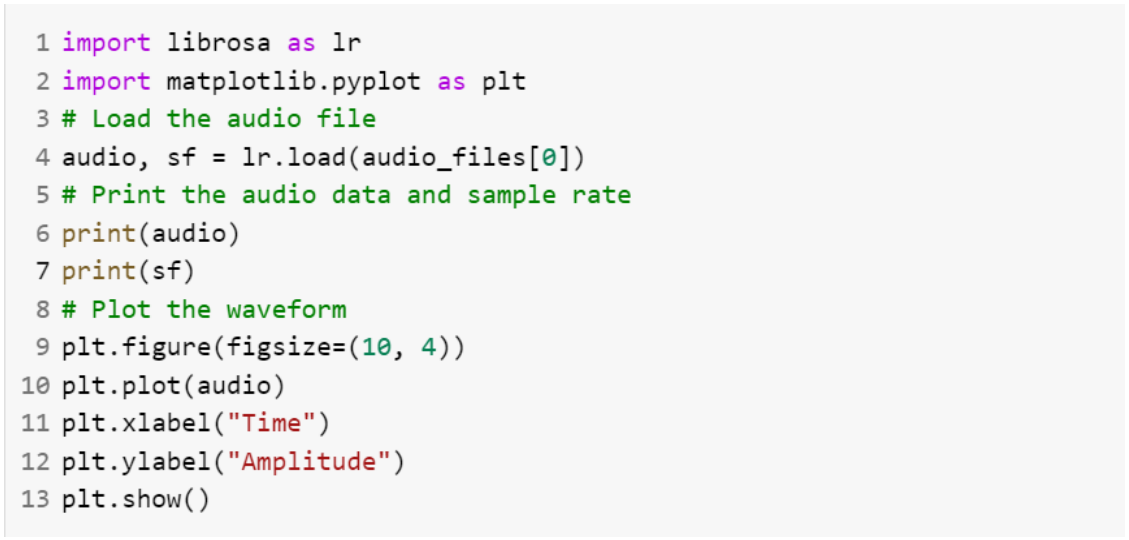
```
audio_files=glob('/content/gdrive/MyDrive/major_project/music_genre_updated/Data/trial_Genre_Originals/*.wav')
print(audio_files)
print(len(audio_files))

['/content/gdrive/MyDrive/minor_project/music_genre_updated/Data/trial_Genre_Originals/rock.00003.wav', '/content/gdrive/MyDrive/minor_project/music_ge
```

Figure 3.7 Loading audio files

Using glob we got all audio files of the dataset in the audio_files.

B. Data Pre-processing

In figure 3.9 we can see the waveplot of one of the audio file from the dataset ,its audio of a blue genre . Now we know its not possible to work on this file as it is so we will try to extract some features from the audio files and on the basis of those fearture we will try to distinguish between all 10 different genre that we have in our dataset. So firstly we tried to get the min , max and mean frequency of the audio that can be seen in Figure .



```python
import librosa as lr
import matplotlib.pyplot as plt
# Load the audio file
audio, sf = lr.load(audio_files[0])
# Print the audio data and sample rate
print(audio)
print(sf)
# Plot the waveform
plt.figure(figsize=(10, 4))
plt.plot(audio)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.show()
```

Figure 3.8 Code for waveplot

```
22050
```

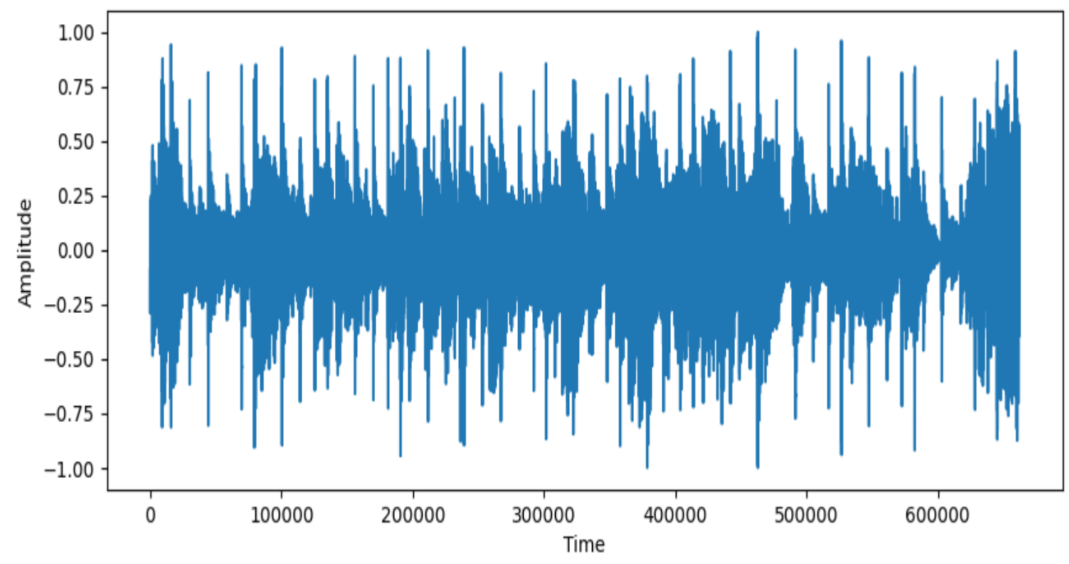
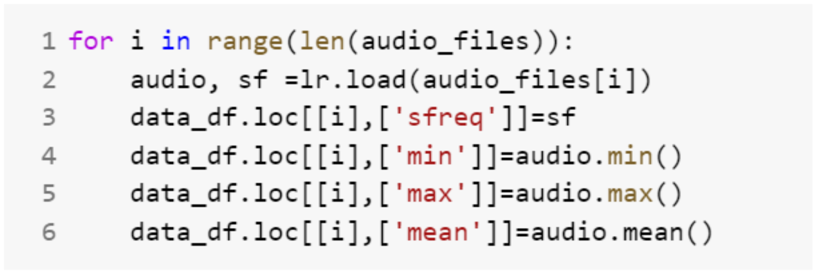
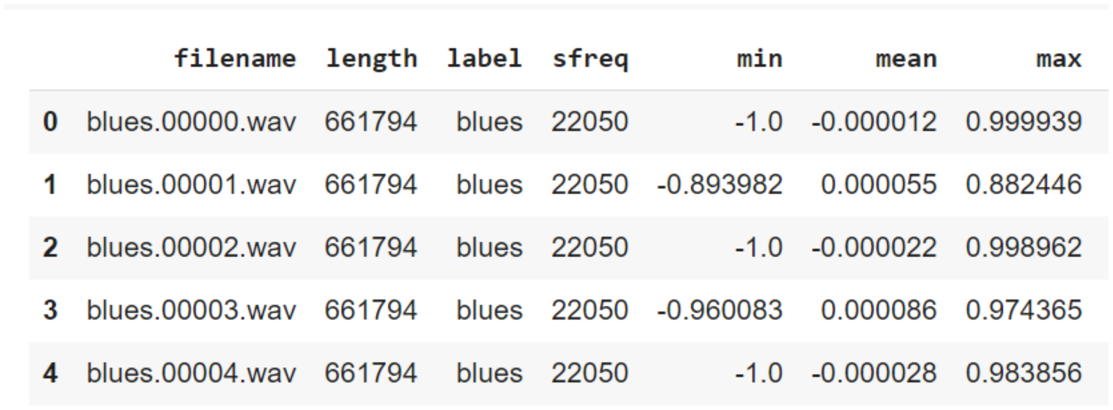


Figure 3.9 Waveplot



```
for i in range(len(audio_files)):
    audio, sf =lr.load(audio_files[i])
    data_df.loc[[i],['sfreq']]=sf
    data_df.loc[[i],['min']]=audio.min()
    data_df.loc[[i],['max']]=audio.max()
    data_df.loc[[i],['mean']]=audio.mean()
```

Figure 3.10 Adding frequency to dataset

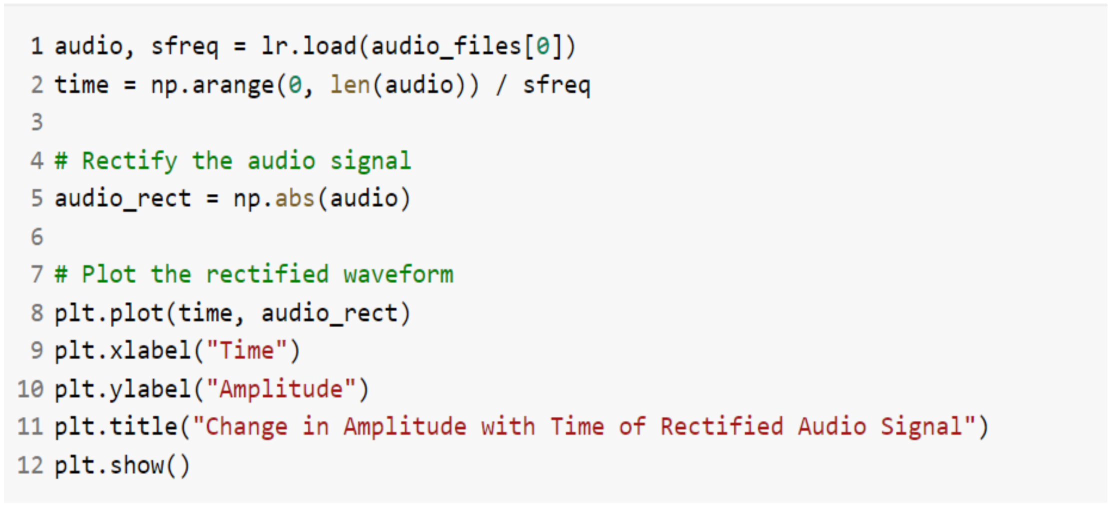


	filename	length	label	sfreq	min	mean	max
0	blues.00000.wav	661794	blues	22050	-1.0	-0.000012	0.999939
1	blues.00001.wav	661794	blues	22050	-0.893982	0.000055	0.882446
2	blues.00002.wav	661794	blues	22050	-1.0	-0.000022	0.998962
3	blues.00003.wav	661794	blues	22050	-0.960083	0.000086	0.974365
4	blues.00004.wav	661794	blues	22050	-1.0	-0.000028	0.983856

Figure 3.11 Updated dataset 1.

Reactifying negative values in signals that shown in Figure 3.9 .
Classifying music genres and other audio analysis tasks frequently involve the preprocessing step of rectifying negative values in a signal. Any negative numbers in the signal are converted to their positive equivalents during the procedure, thus erasing the sign information.

Rectifying negative numbers is important because it can capture the entire energy or size of the signal without capturing the phase or polarity information. Music genre classification typically places more focus on the spectral and temporal characteristics of the audio than the phase or polarity of the signal. By rectification of the signal, we can simplify the representation and ensure that the energy content is precisely represented. The figure shows the rectified waveplot of one of the audio file from the dataset.



```python
audio, sfreq = lr.load(audio_files[0])
time = np.arange(0, len(audio)) / sfreq

# Rectify the audio signal
audio_rect = np.abs(audio)

# Plot the rectified waveform
plt.plot(time, audio_rect)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Change in Amplitude with Time of Rectified Audio Signal")
plt.show()
```

Figure 3.12 Code to rectify negative signal

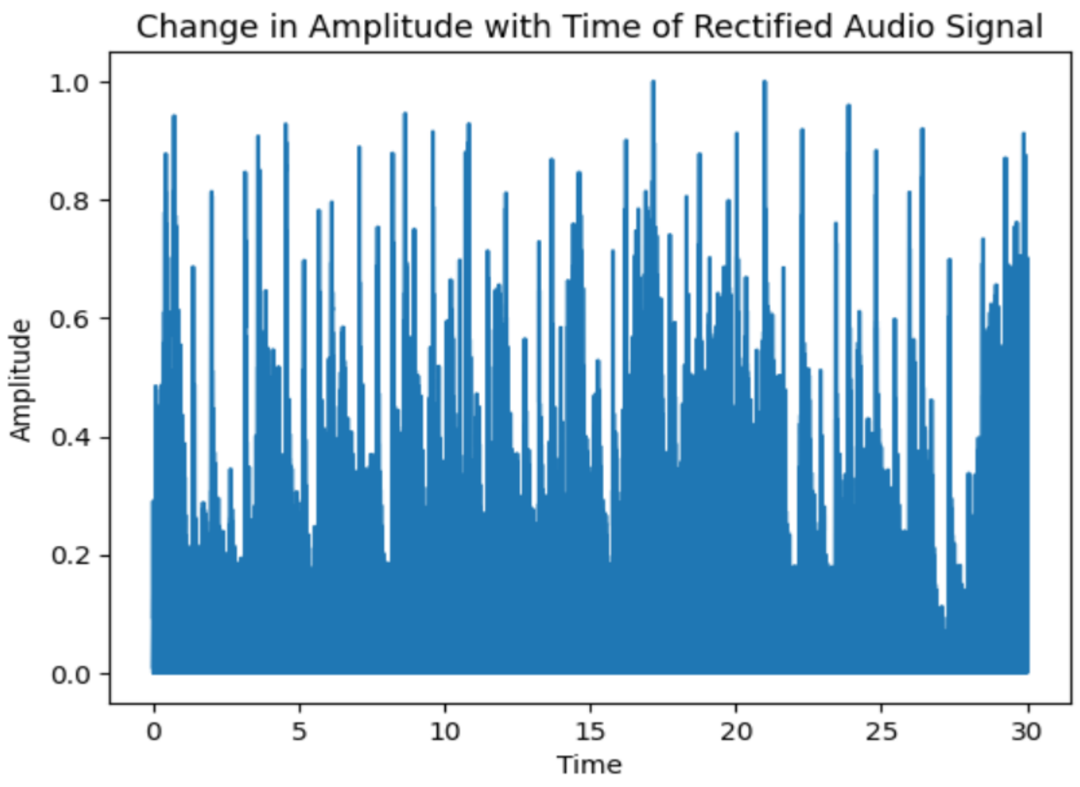
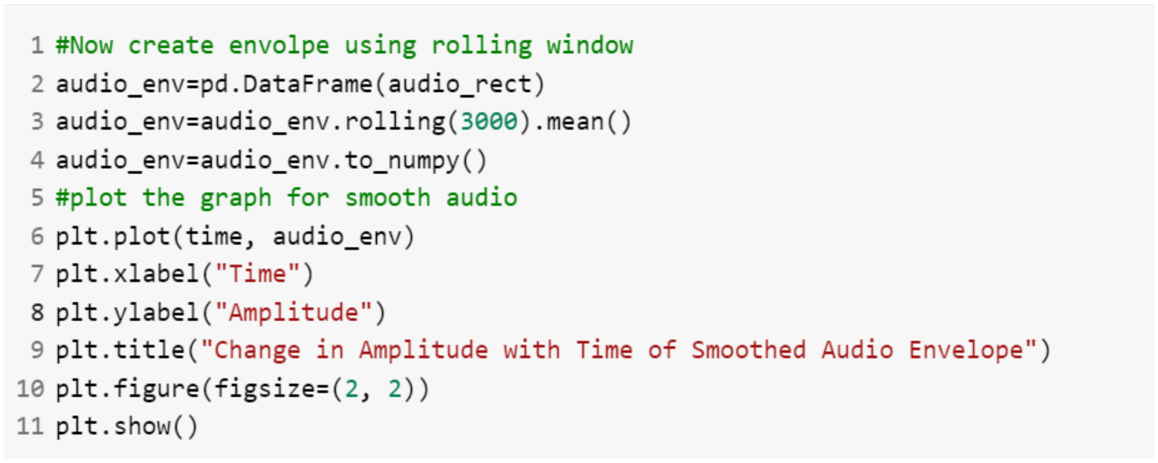


Figure 3.13 Rectified signal

The rolling window technique is applied to smooth out the rectified audio signal. which lessens the effect of short-term fluctuations and highlights longer-term trends or patterns in the data. The width of the window used for smoothing is decided by selecting a window size of 3000 samples. Depending on the qualities of your audio data and the required level of smoothing, you can change this value.



```
#Now create envolpe using rolling window
audio_env=pd.DataFrame(audio_rect)
audio_env=audio_env.rolling(3000).mean()
audio_env=audio_env.to_numpy()
#plot the graph for smooth audio
plt.plot(time, audio_env)
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Change in Amplitude with Time of Smoothed Audio Envelope")
plt.figure(figsize=(2, 2))
plt.show()
```

Figure 3.14 Code to smooth the signal

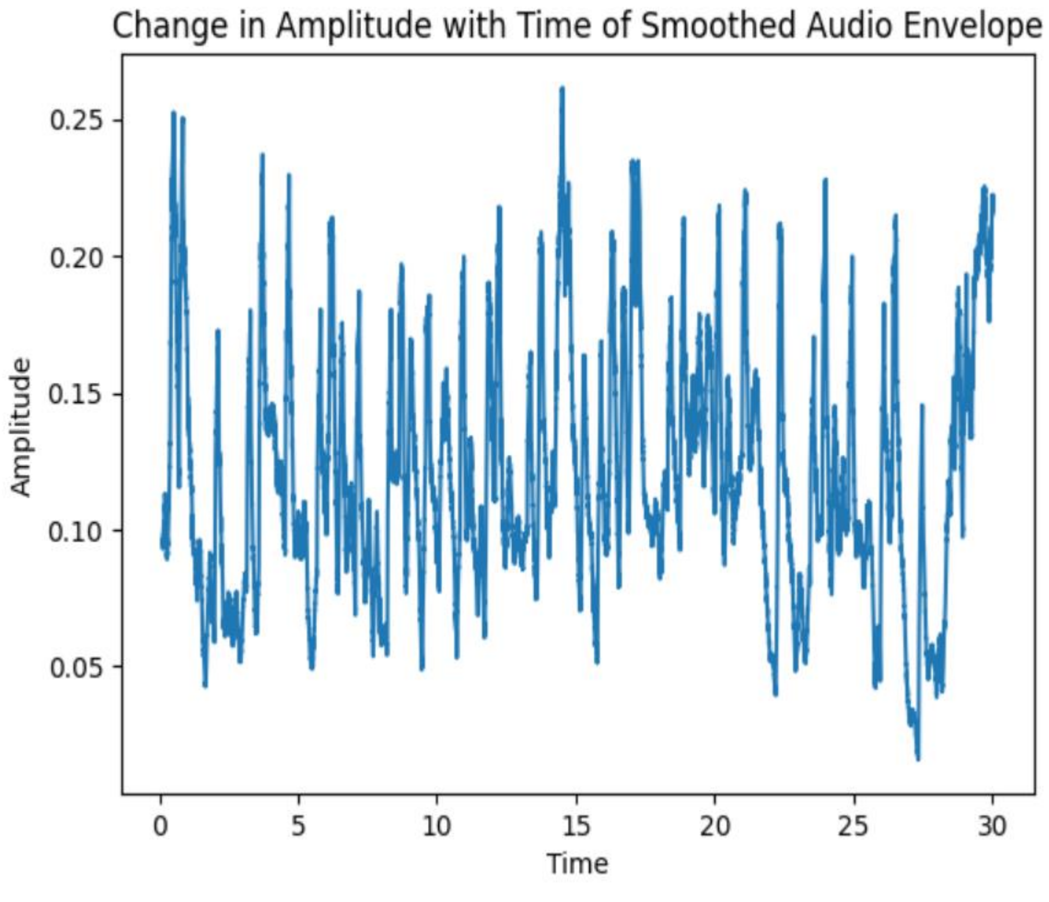
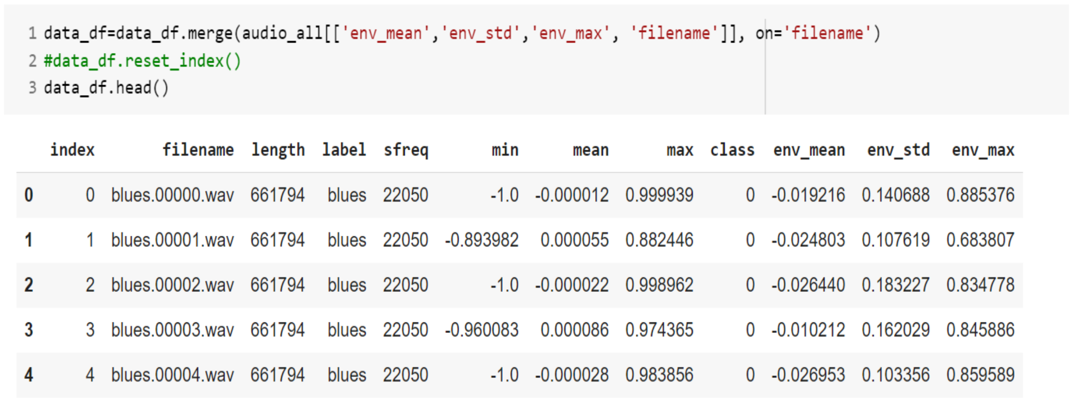


Figure 3.15 Smooth audio

The rectified audio signal's envelope is smoothed out using the rolling window method. The fundamental concept is to average a value over a window of a certain size and move the window with the signal.

To make use of the rolling window capabilities, the rectified audio signal is transformed into a pandas DataFrame. A 3000 sample window size is used with the rolling window. This means that each sample in the DataFrame is considered in a window of 3000 samples, and the mean value of that window is calculated. During this rolling process, the window is moved across the signal one sample at a time. For additional processing and plotting, the resulting DataFrame is changed back into a numpy array.



```
data_df=data_df.merge(audio_all[['env_mean','env_std','env_max', 'filename']], on='filename')
#data_df.reset_index()
data_df.head()
```

	index	filename	length	label	sfreq	min	mean	max	class	env_mean	env_std	env_max
0	0	blues.00000.wav	661794	blues	22050	-1.0	-0.000012	0.999939	0	-0.019216	0.140688	0.885376
1	1	blues.00001.wav	661794	blues	22050	-0.893982	0.000055	0.882446	0	-0.024803	0.107619	0.683807
2	2	blues.00002.wav	661794	blues	22050	-1.0	-0.000022	0.998962	0	-0.026440	0.183227	0.834778
3	3	blues.00003.wav	661794	blues	22050	-0.960083	0.000086	0.974365	0	-0.010212	0.162029	0.845886
4	4	blues.00004.wav	661794	blues	22050	-1.0	-0.000028	0.983856	0	-0.026953	0.103356	0.859589

Figure 3.16 Updated dataset 2.

Figure 3.16 shows the dataset with min ,max and mean frequency of the signal on that was not rectified or smoothen as well as it contains env_mean , the average value of each characteristic across all samples in the DataFrame is shown in this column. It might be helpful for examining the average behaviour or core tendency of the audio elements. env_std provides variability or dispersion of the results within each sample . Understanding the distribution or degree of variance in the audio attributes across various samples might be helpful.

Our next aim is to convert signal from time domain to frequency domain . We can analyse and manipulate a signal based on its frequency content by translating it from the time domain to the frequency domain. In addition to offering computational efficiency for real-time processing, it enables feature extraction, facilitates filtering and manipulation operations, and offers insights into the spectral properties. The conversion to the frequency domain is a crucial step in many audio processing applications because of these advantages.

Fast Fourier Transform is referred to as FFT. The Fast Fourier Transform is a mathematical approach for quickly computing the Discrete Fourier Transform (DFT) of a signal or sequence.

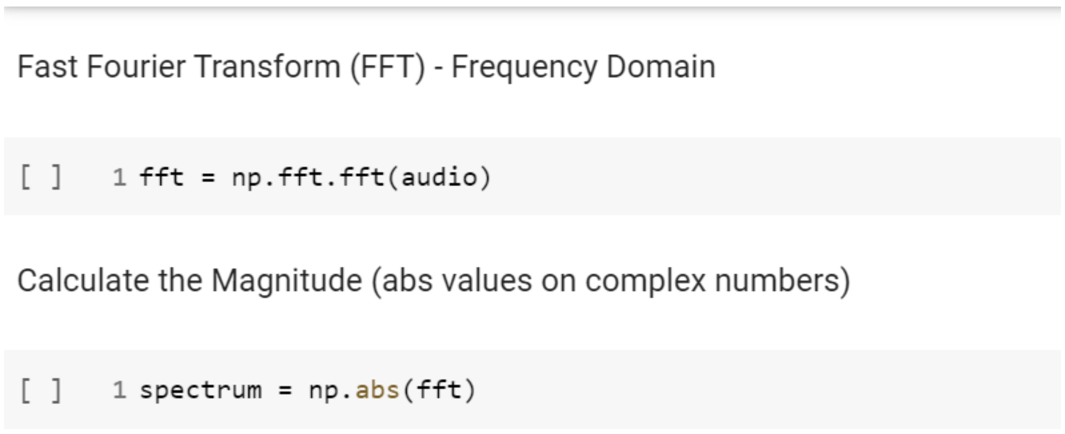
The Discrete Fourier Transform (DFT) is defined as follows for a sequence x[n] of length N:

X[k] = ∑[n=0 to N-1] x[n] * exp(-j * 2π * k * n / N)

where:

X[k] represents the complex-valued spectrum of the signal at frequency bin k.
x[n] is the input sequence, representing the discrete-time samples of the signal.
N is the length of the input sequence, which determines the number of frequency bins in the resulting spectrum.
j is the imaginary unit (√(-1)).
k represents the frequency bin index, ranging from 0 to N-1.

The audio signal is converted from the time domain to the frequency domain via the FFT technique. It breaks down the signal into its individual frequencies and gives details on each frequency's amplitude and phase.


Fast Fourier Transform (FFT) - Frequency Domain

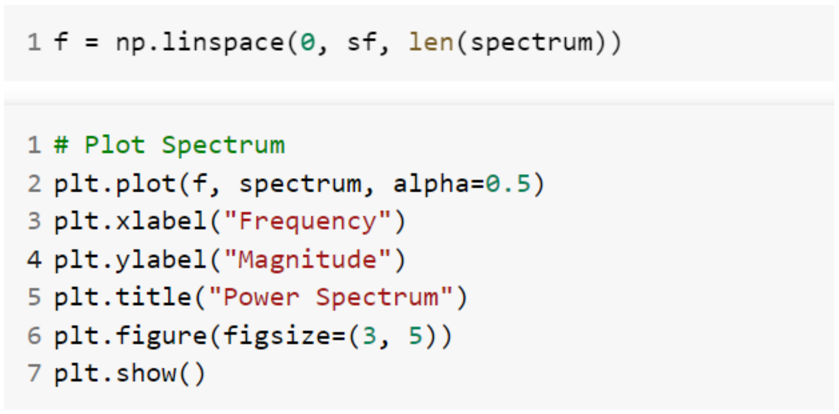
```
[ ]   fft = np.fft.fft(audio)
```

Calculate the Magnitude (abs values on complex numbers)

```
[ ]   spectrum = np.abs(fft)
```

Figure 3.17 Applying fourier transform

FFT is performed on the “audio” file , the results of FFT operation is stored in variable fft , this will contain the audio signal's complex-valued spectrum. “spectrum” variable is the magnitude spectrum i.e a real valued array .Information on the strength or intensity of the various frequencies included in the signal can be determined from the magnitude spectrum.



```python
f = np.linspace(0, sf, len(spectrum))
```

```python
# Plot Spectrum
plt.plot(f, spectrum, alpha=0.5)
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.title("Power Spectrum")
plt.figure(figsize=(3, 5))
plt.show()
```

Figure 3.18 code to plot power spectrum

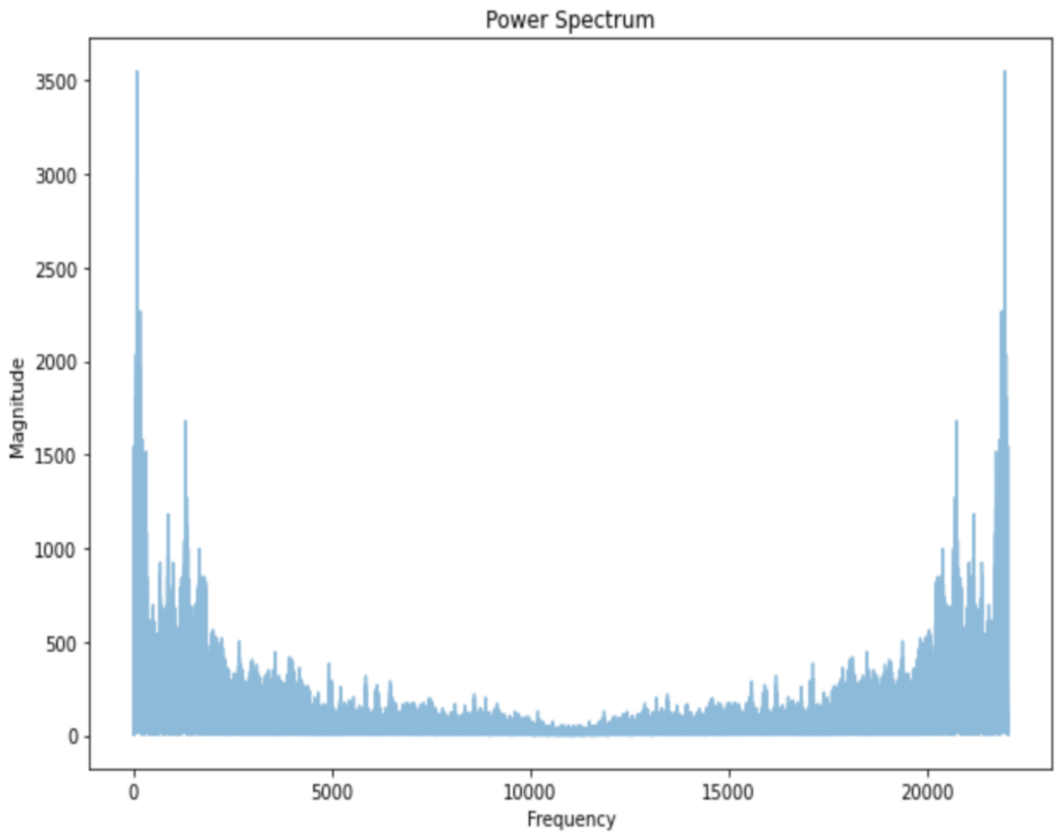
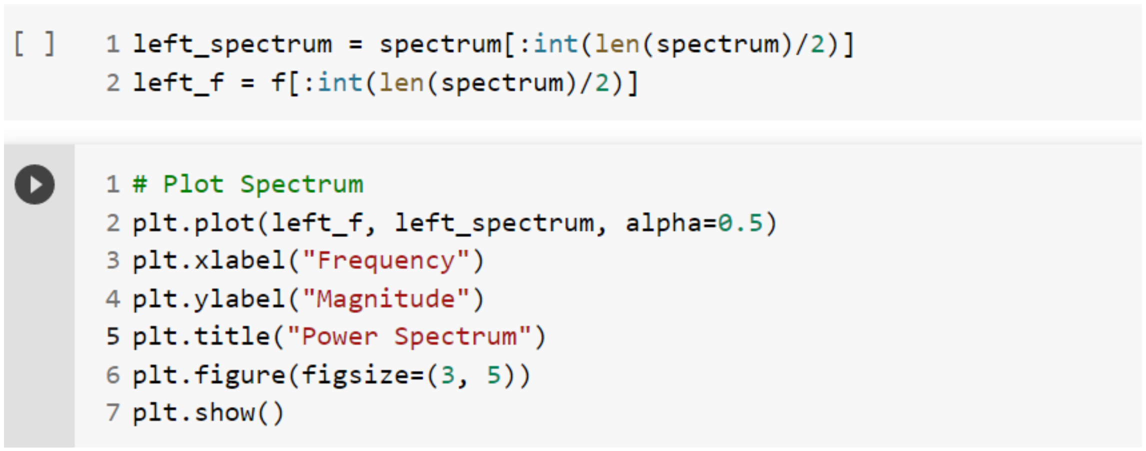


Figure 3.19 Power spectrum



```
[ ]
left_spectrum = spectrum[:int(len(spectrum)/2)]
left_f = f[:int(len(spectrum)/2)]
```

```
# Plot Spectrum
plt.plot(left_f, left_spectrum, alpha=0.5)
plt.xlabel("Frequency")
plt.ylabel("Magnitude")
plt.title("Power Spectrum")
plt.figure(figsize=(3, 5))
plt.show()
```

Figure 3.20 Code to get half power spectrum

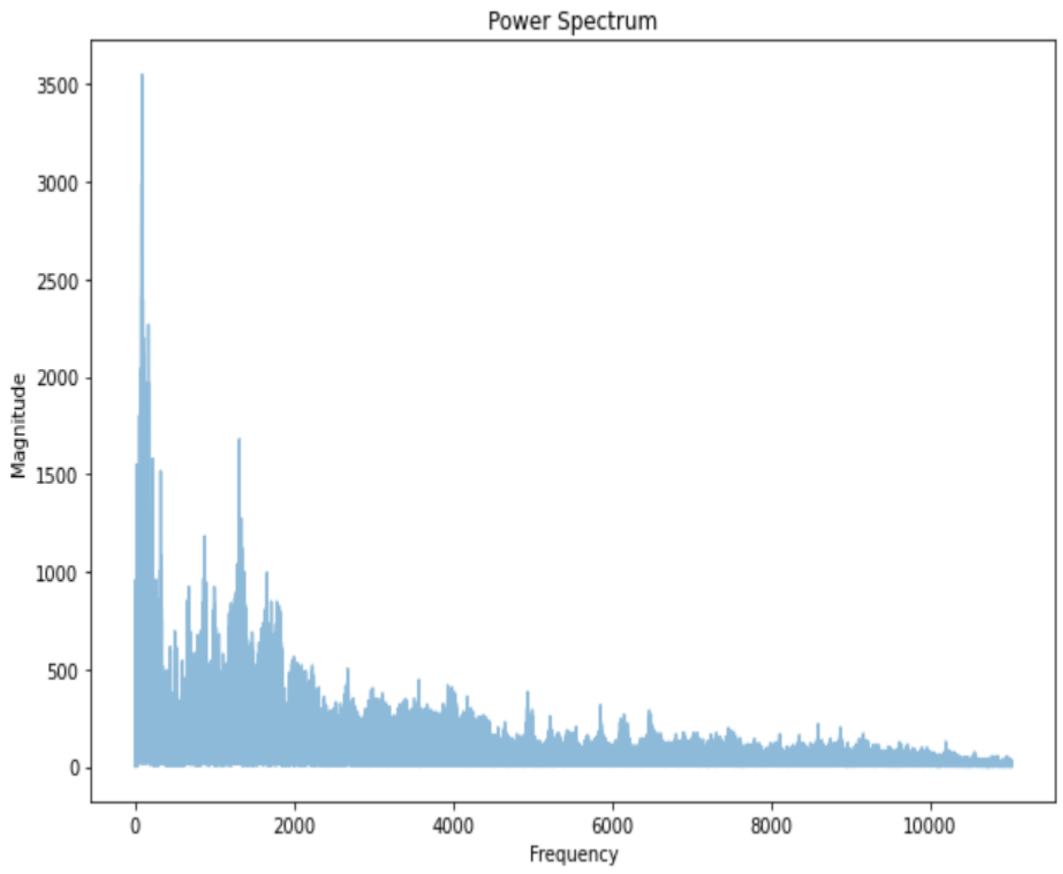
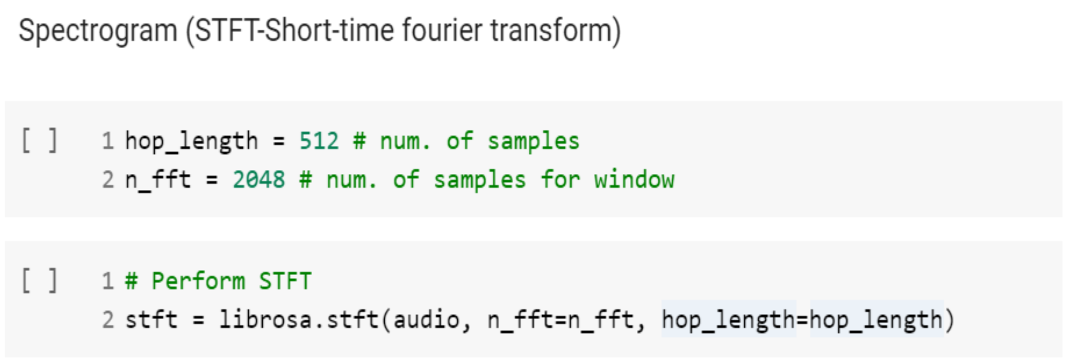


Figure 3.21 Plot of half power spectrum

In figure 3.21 we see that we have taken half of the power spectrum as in real-valued signals, it is usual practice to take half of the power spectrum to reduce analysis complexity and redundancy while keeping the crucial frequency information.

To acquire the frequency representation of the signal at various time intervals, the STFT algorithm splits the audio signal into overlapping frames and applies the FFT to each frame. The resulting STFT is a complex-valued matrix, where each member denotes the phase and amplitude of a single frequency component at a particular time. In comparison to using non-overlapping frames, we are able to depict the signal across time in a way that is more continuous and smooth. The overlap enables a more gradual transition between succeeding frames and improves the ability to accurately capture the temporal dynamics of the data.



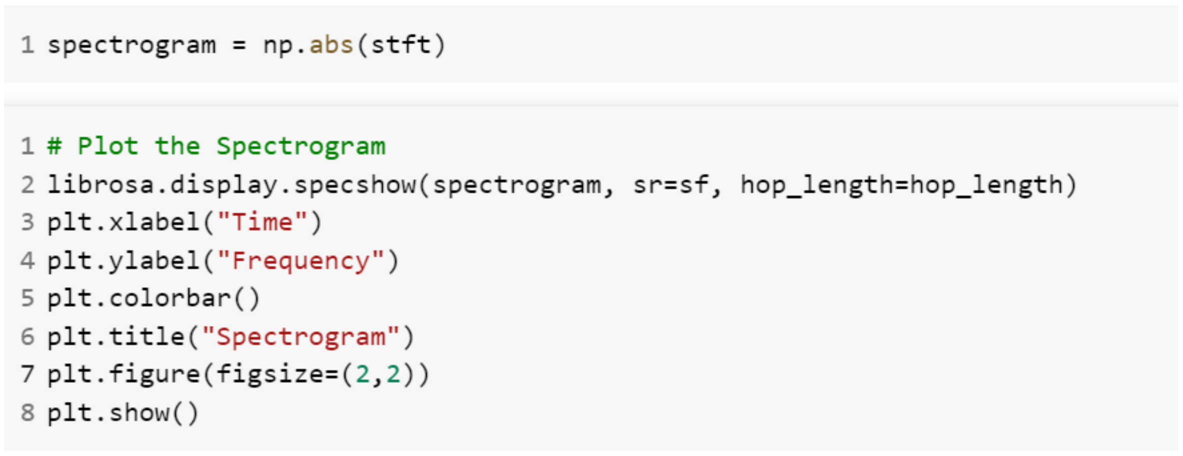
Spectrogram (STFT-Short-time fourier transform)

```
hop_length = 512 # num. of samples
n_fft = 2048 # num. of samples for window
```

```
# Perform STFT
stft = librosa.stft(audio, n_fft=n_fft, hop_length=hop_length)
```

Figure 3.22 Code to perform STFT

STFT makes it possible to perform a wide range of audio signal processing activities and offers insightful information about the spectrum properties and dynamics of audio signals by facilitating time-frequency analysis, spectrogram production, feature extraction, filtering, enhancement, etc.



```
spectrogram = np.abs(stft)
```

```
# Plot the Spectrogram
librosa.display.specshow(spectrogram, sr=sf, hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar()
plt.title("Spectrogram")
plt.figure(figsize=(2,2))
plt.show()
```

Figure 3.23 Code to plot spectogram

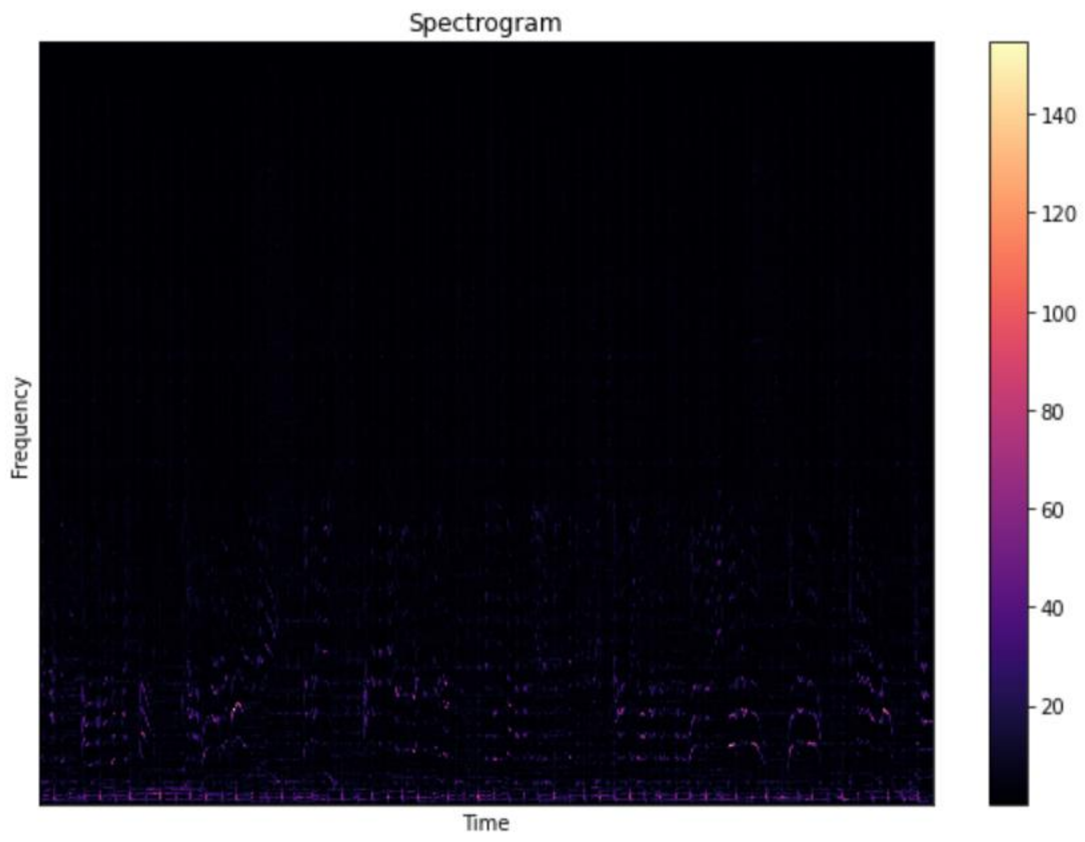
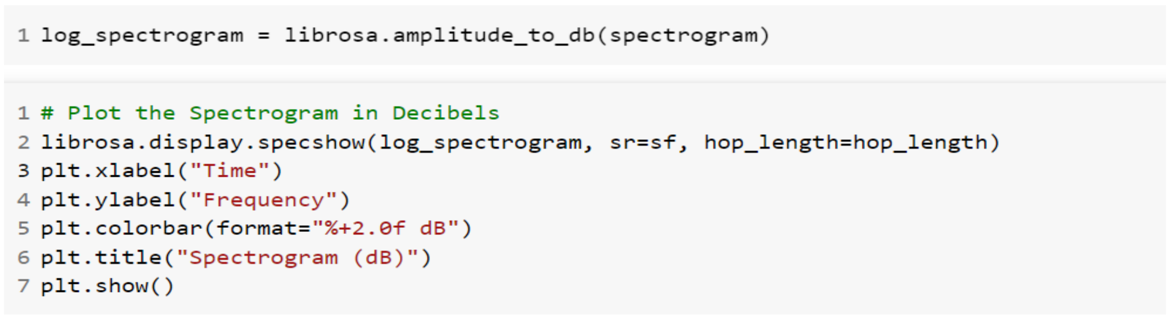


Figure 3.24 Spectogram

The spectrogram graphic makes it possible to see how the audio signal's spectral content changes over time, giving information about the shifting frequency components and the severity of those changes. It is a representation that is frequently used in audio analysis and offers useful data for tasks like pattern recognition, event detection, and feature extraction from the audio signal.

Figure 3.24 is not that clear so we will convert the magnitude spectrogram to a dB scale.



```
log_spectrogram = librosa.amplitude_to_db(spectrogram)
```

```
# Plot the Spectrogram in Decibels
librosa.display.specshow(log_spectrogram, sr=sf, hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("Frequency")
plt.colorbar(format="%+2.0f dB")
plt.title("Spectrogram (dB)")
plt.show()
```

Figure 3.25 Code to convert spectogram to dB scale

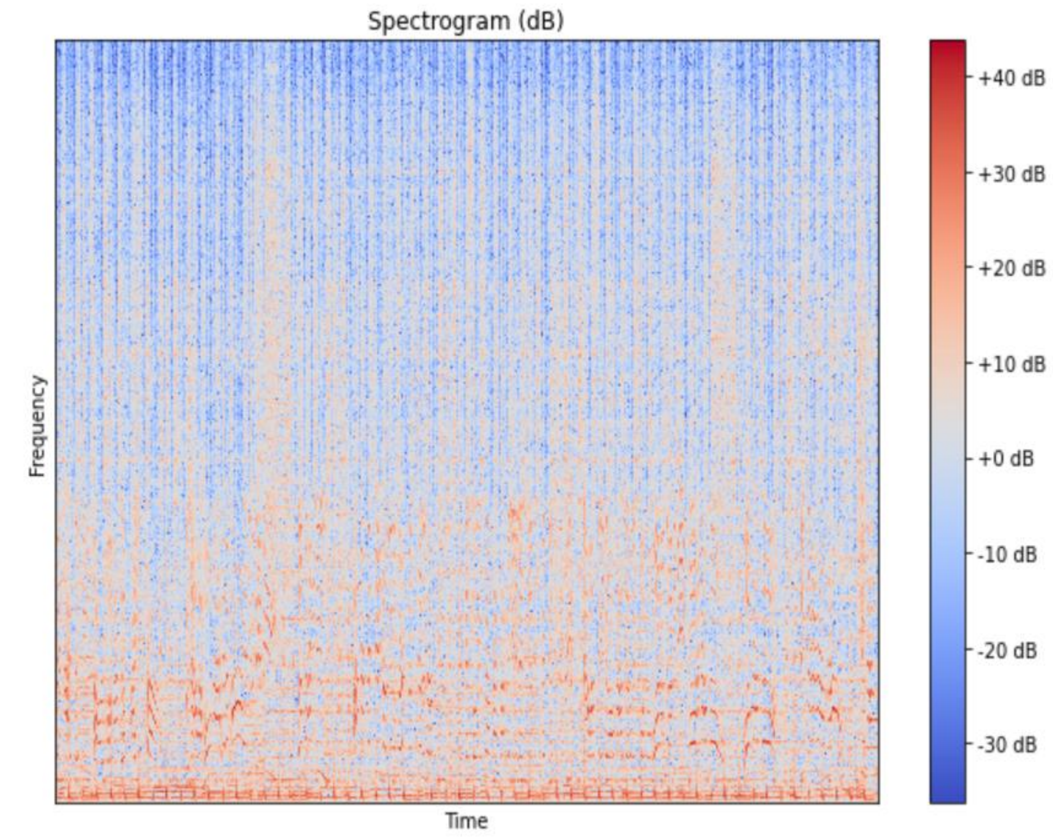
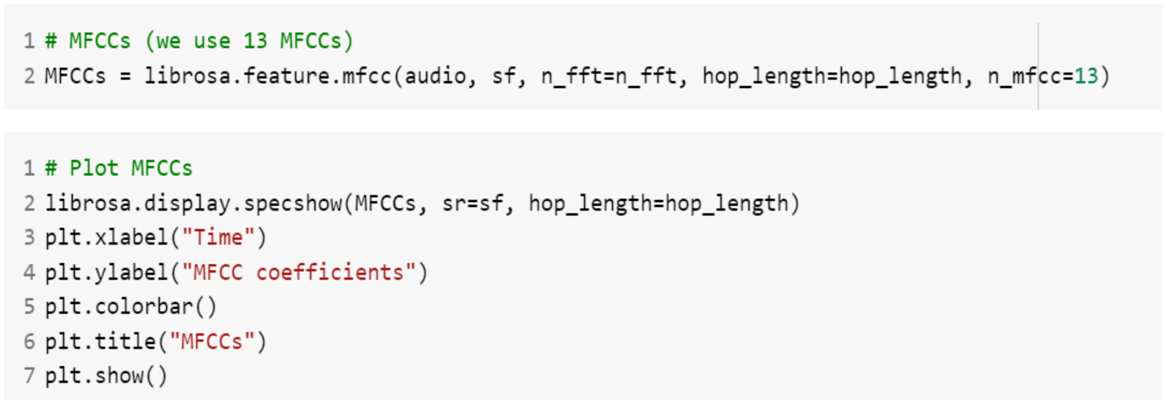


Figure 3.26 Spectogram(dB)

The logarithmic scale emphasises the relative differences in magnitude, making it possible for a more perceptually relevant visualisation when the spectrogram is displayed in dB. When working with a wide range of magnitudes or analysing low-level details, it makes it possible to understand the energy distribution and spectral properties of the audio signal better.

In the analysis and processing of audio signals, MFCCs are a common feature. By depicting the short-term power spectrum on a perceptually inspired frequency scale called the mel-scale, they are able to accurately reproduce the spectral features of the audio source. The audio signal is represented compactly and robustly by MFCCs, which highlight significant spectral features while dismissing the impact of unimportant variations.



```python
# MFCCs (we use 13 MFCCs)
MFCCs = librosa.feature.mfcc(audio, sf, n_fft=n_fft, hop_length=hop_length, n_mfcc=13)
```

```python
# Plot MFCCs
librosa.display.specshow(MFCCs, sr=sf, hop_length=hop_length)
plt.xlabel("Time")
plt.ylabel("MFCC coefficients")
plt.colorbar()
plt.title("MFCCs")
plt.show()
```

Figure 3.27 Code to plot MFCC’s

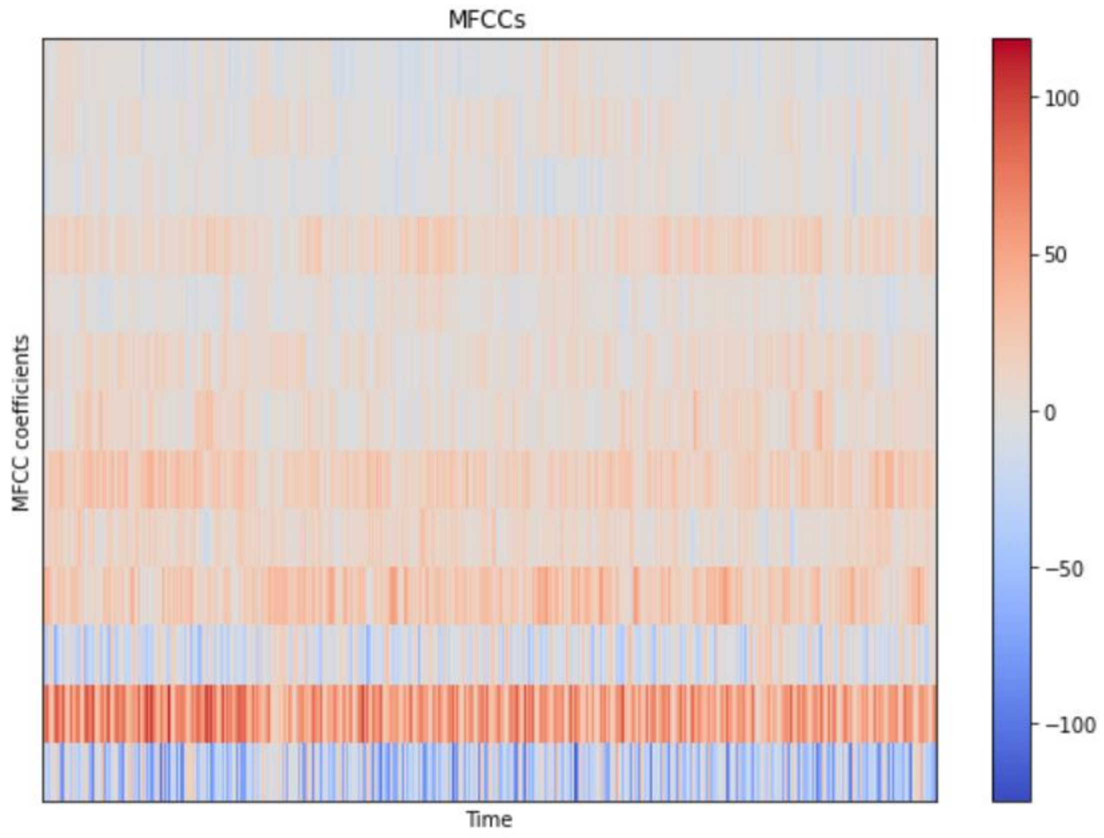
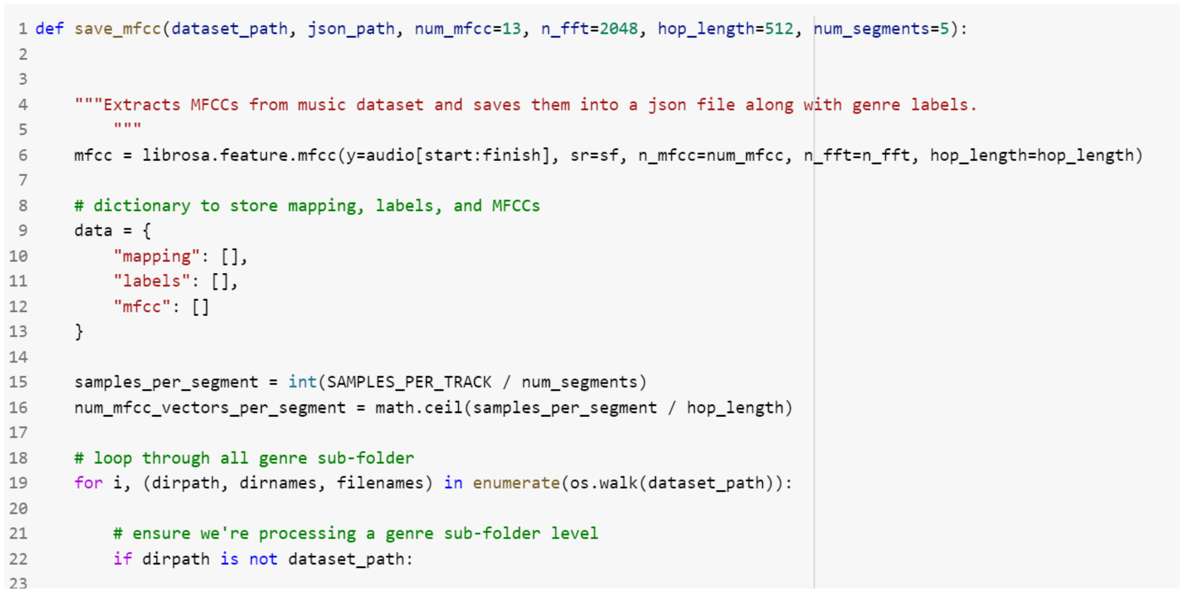


Figure 3.28 MFCC

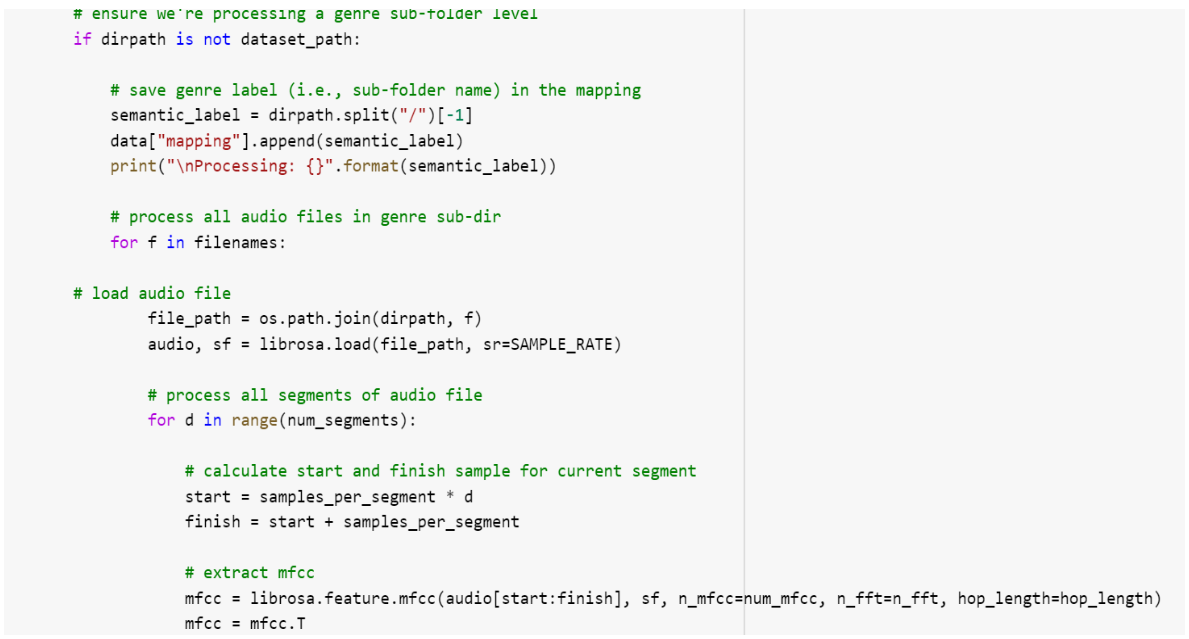
Machine learning models are frequently trained using MFCCs as input features. These models pick up on trends and connections between the MFCCs and the associated musical genres. MFCCs help to accurately classify music genres by extracting relevant spectral information and displaying them in a compact and reliable way.

After pre-processing our data with MFCC’s as the end result we store them in the json file and after that we split ou data into training and testing dataset.



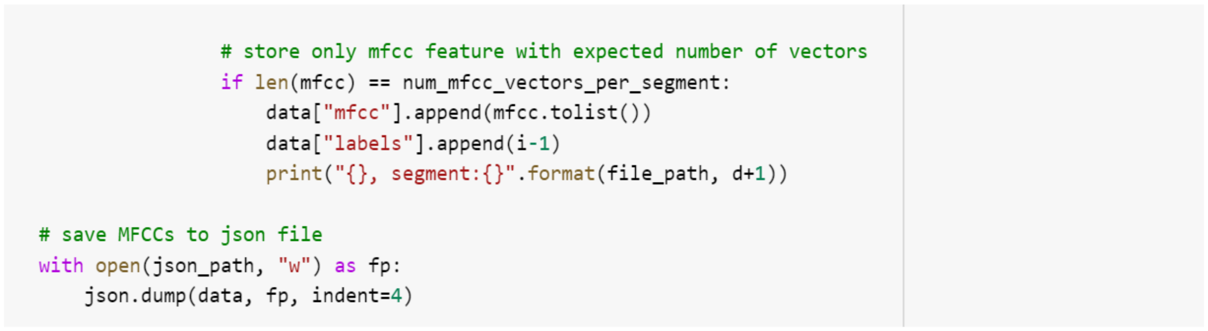
```
def save_mfcc(dataset_path, json_path, num_mfcc=13, n_fft=2048, hop_length=512, num_segments=5):


    """Extracts MFCCs from music dataset and saves them into a json file along with genre labels.
        """
    mfcc = librosa.feature.mfcc(y=audio[start:finish], sr=sf, n_mfcc=num_mfcc, n_fft=n_fft, hop_length=hop_length)

    # 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:

```

Figure 3.29 Code to save MFCC in json part_1



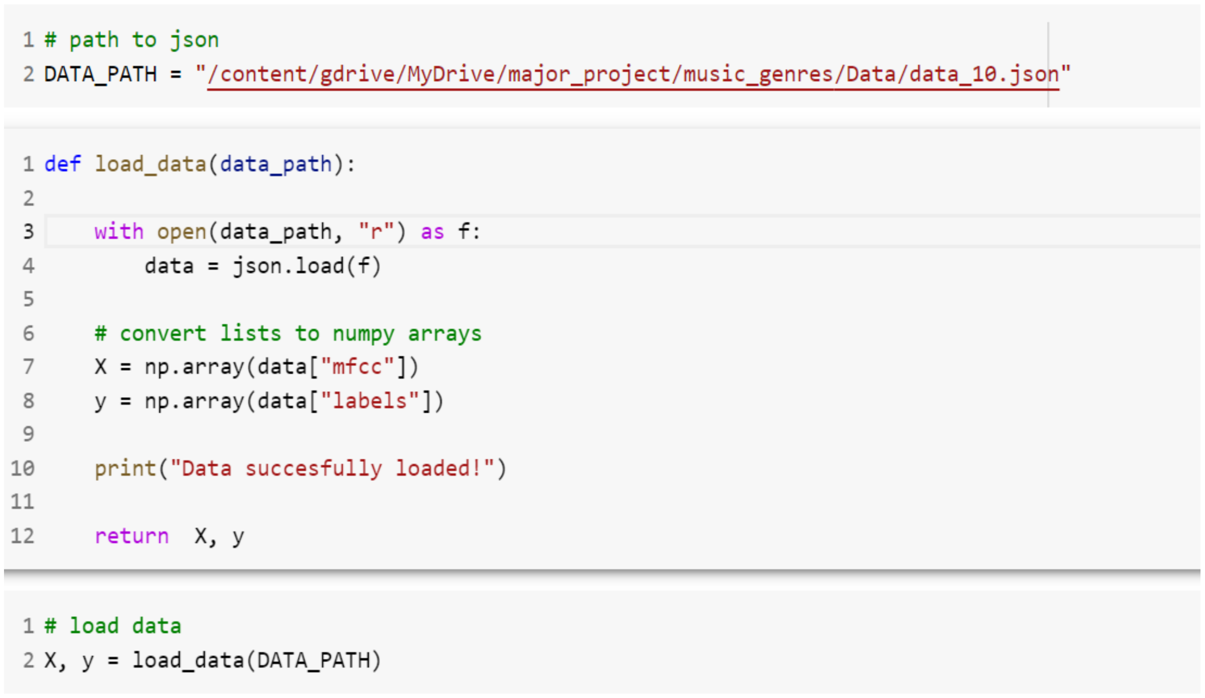
```
        # ensure we're processing a genre sub-folder level
        if dirpath is not dataset_path:

            # save genre label (i.e., sub-folder name) in the mapping
            semantic_label = dirpath.split("/")[-1]
            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)
                audio, sf = 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(audio[start:finish], sf, n_mfcc=num_mfcc, n_fft=n_fft, hop_length=hop_length)
                    mfcc = mfcc.T
```

Figure 3.30 Code to save MFCC in json part_2



```
                # 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)
```

Figure 3.31 Code to save MFCC in json part_3

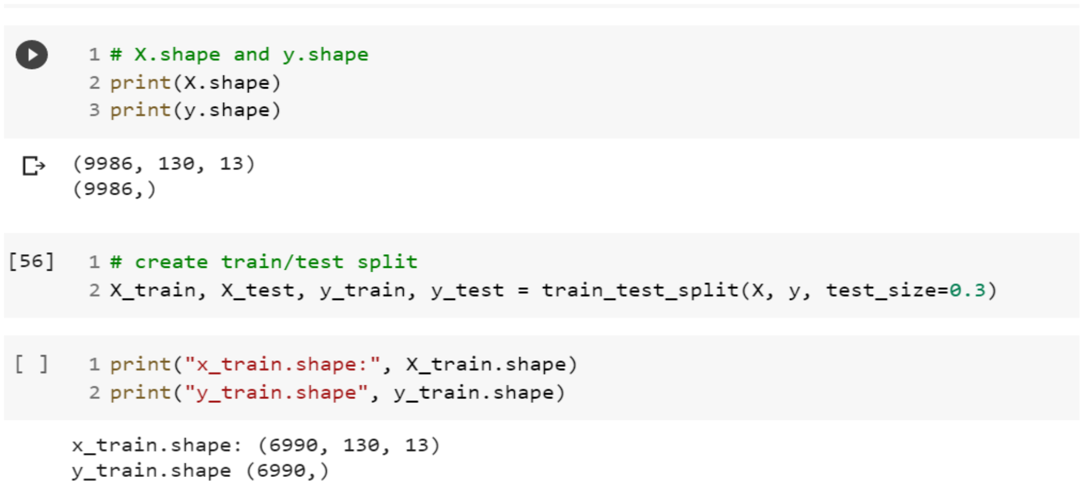


```
# path to json
DATA_PATH = "/content/gdrive/MyDrive/major_project/music_genres/Data/data_10.json"
```

```
def load_data(data_path):

    with open(data_path, "r") as f:
        data = json.load(f)

    # convert lists to numpy arrays
    X = np.array(data["mfcc"])
    y = np.array(data["labels"])

    print("Data succesfully loaded!")

    return  X, y
```

```
# load data
X, y = load_data(DATA_PATH)
```

Figure 3.32 Code to load the data



```
# X.shape and y.shape
print(X.shape)
print(y.shape)
```

```
(9986, 130, 13)
(9986,)
```

```
[56]
# create train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
[ ]
print("x_train.shape:", X_train.shape)
print("y_train.shape", y_train.shape)
```

```
x_train.shape: (6990, 130, 13)
y_train.shape (6990,)
```

Figure 3.33 Code to split dataset

C. Models

Sequential DNN

Deep or shallow artificial neural networks (ANNs) are both possible. An ANN is referred to as a deep neural network when it contains more than one hidden layer in its architecture. With the use of mathematical modelling, these networks handle complex data. Another name for Deep Neural Networks (DNN) is Feed Forward Neural Networks (FFNNS).

DNNs are designed to learn hierarchical representations of data by progressively extracting higher-level features from lower-level ones.

Data in these networks always ever flows forward and never backward, a node can never be accessed more than once. These Networks can classify millions of data points and require a lot of data to train.

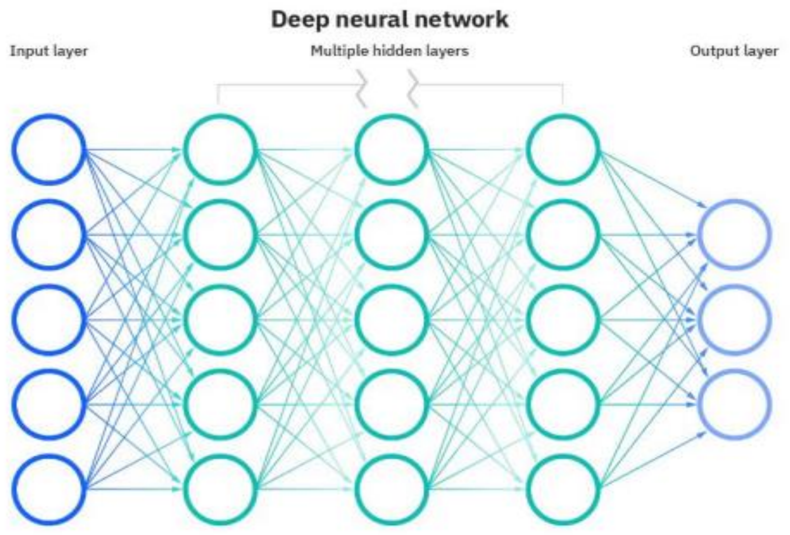


Figure 3.34 Deep neural network

Input, output, and a few hidden layers are all present in deep neural networks. These networks are capable of handling non-linearity as well as unstructured and unlabeled data. Similar to the human brain, they contain a hierarchical organisation of neurons.

Depending on the information they receive, the neurons relay the signal to other neurons. The output will be passed if the signal value exceeds the threshold value and ignored otherwise. As you can see, data is transmitted from one layer to the next, producing output at each stage until it reaches the output layer, which makes a yes-or-no prediction based on probability. Each neuron in a layer has an activitation function , and a layer is made up of numerous neurons. They act as a gateway for the signal to travel to the following linked neuron. The weight affects the input to the next neuron's output and then the final output layer. Initial weights are assigned at random, but when the network is iteratively trained, the weights are optimised to make sure the network predicts correctly.

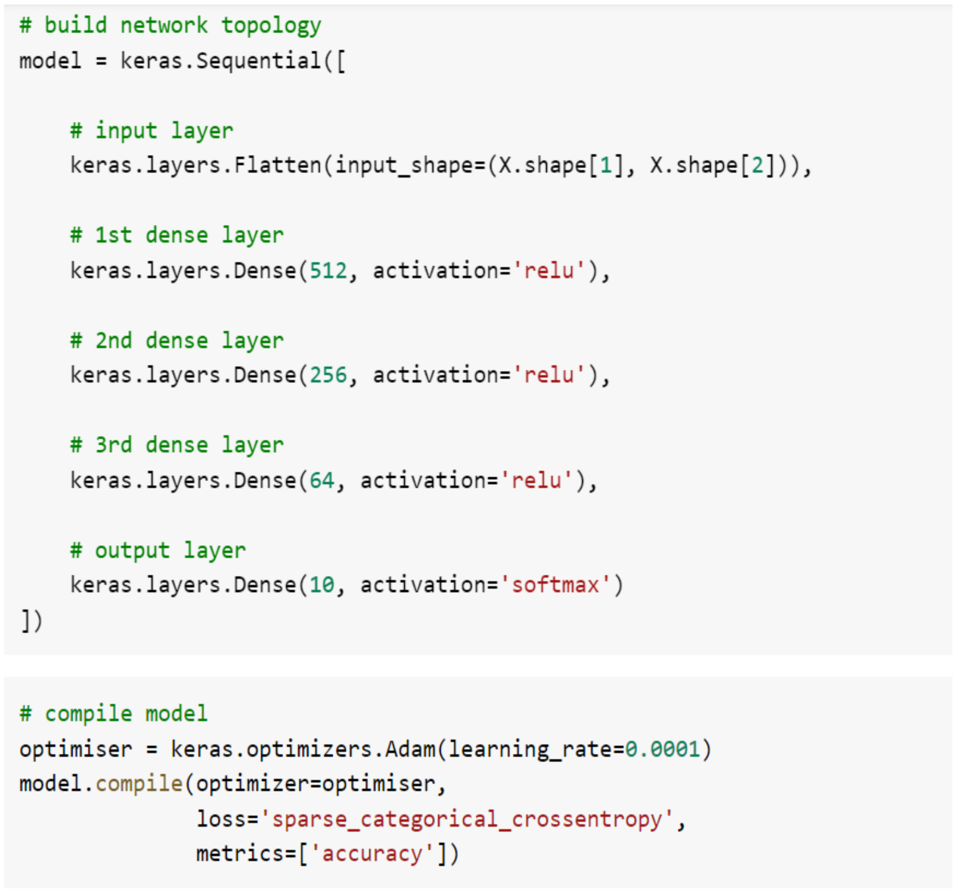
When we want to use deep neural networks (DNNs) to classify audio files into different genres, we need to first transform the audio signals into a format that the DNN can use. We usually do this by converting the signals into spectrograms, mel-frequency cepstral coefficients (MFCCs), or other similar representations that capture important information about the audio.

The DNN architecture we use for this task is designed to identify complex patterns and relationships in the audio data. It has multiple hidden layers, each containing many neurons that allow the network to learn detailed and abstract features from the audio signals.

To train the DNN, we use a dataset of labeled audio samples, where each sample is associated with a specific genre label. During the training process, the DNN learns to map the input audio representations to their corresponding genre labels by adjusting the weights and biases of its neurons. We use techniques like backpropagation and gradient descent to minimize the loss function, which is usually cross-entropy loss.

Once the DNN is trained, we can use it to classify new, unseen audio samples into their respective genre categories. We pass the input audio through the trained network, and the output layer produces a probability distribution over the possible genres. We usually determine the predicted genre label based on the highest probability.

Implementation



```
# build network topology
model = keras.Sequential([

    # input layer
    keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])),

    # 1st dense layer
    keras.layers.Dense(512, activation='relu'),

    # 2nd dense layer
    keras.layers.Dense(256, activation='relu'),

    # 3rd dense layer
    keras.layers.Dense(64, activation='relu'),

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

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

Figure 3.35 DNN model

The model has 3 hidden layers:

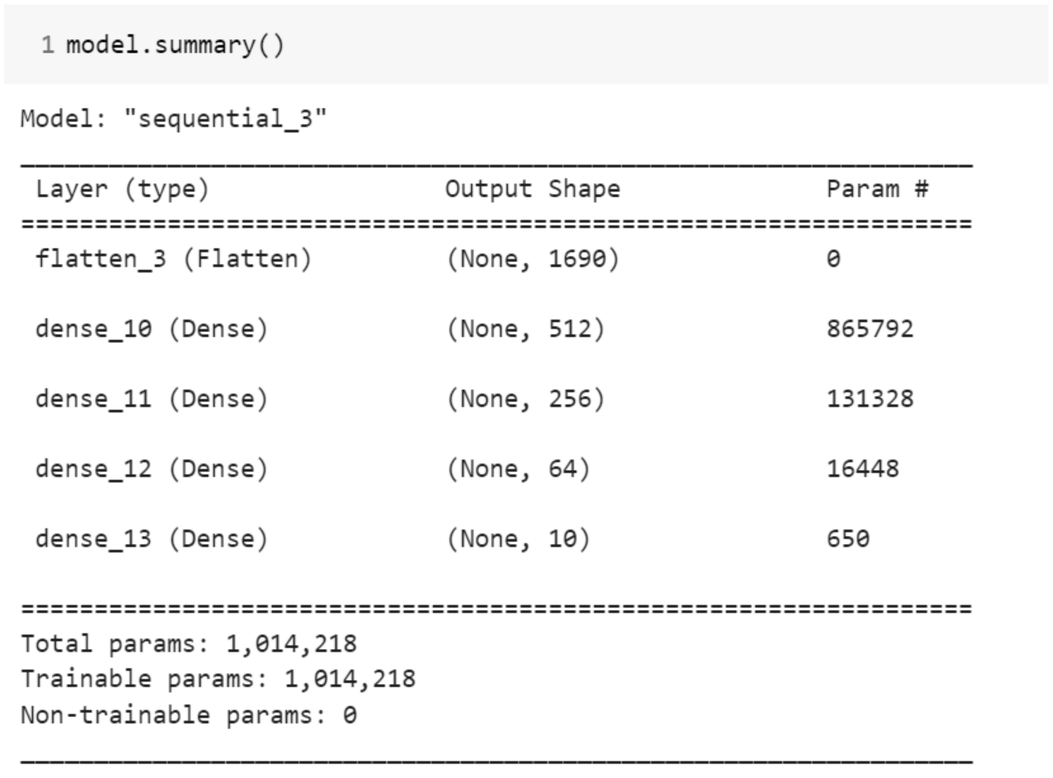
1st Dense Layer with 512 units and ReLU activation function.

2nd Dense Layer with 256 units and ReLU activation function.

3rd Dense Layer with 64 units and ReLU activation function.

These 3 layers, along with the input layer and the output layer,

layer with 10 units and softmax activation function. This layer produces the final output probabilities for each of the 10 classes. The softmax activation ensures that the output probabilities sum up to 1.

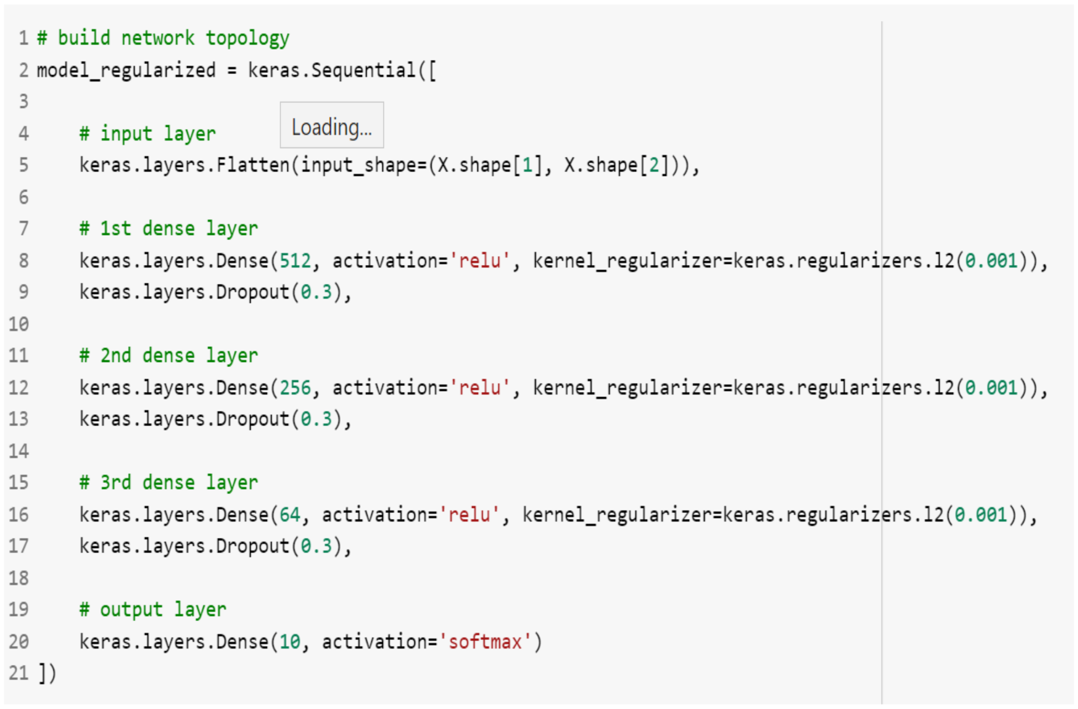


```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_3 (Flatten)         (None, 1690)              0

 dense_10 (Dense)            (None, 512)               865792

 dense_11 (Dense)            (None, 256)               131328

 dense_12 (Dense)            (None, 64)                16448

 dense_13 (Dense)            (None, 10)                650

=================================================================
Total params: 1,014,218
Trainable params: 1,014,218
Non-trainable params: 0
_________________________________________________________________
```

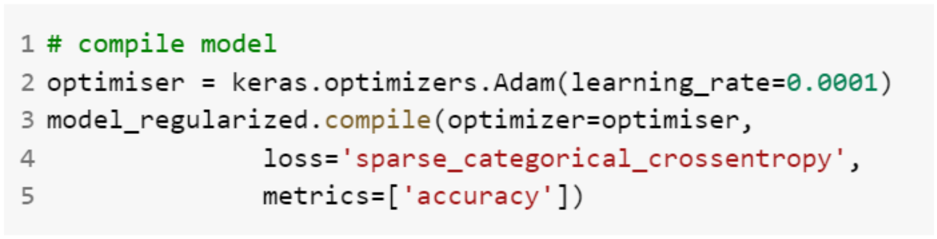
Figure 3.36 DNN model summary

But this model showed overfitting so we regularized the model using l2 regularization. figure shows the implementation of regularized model.



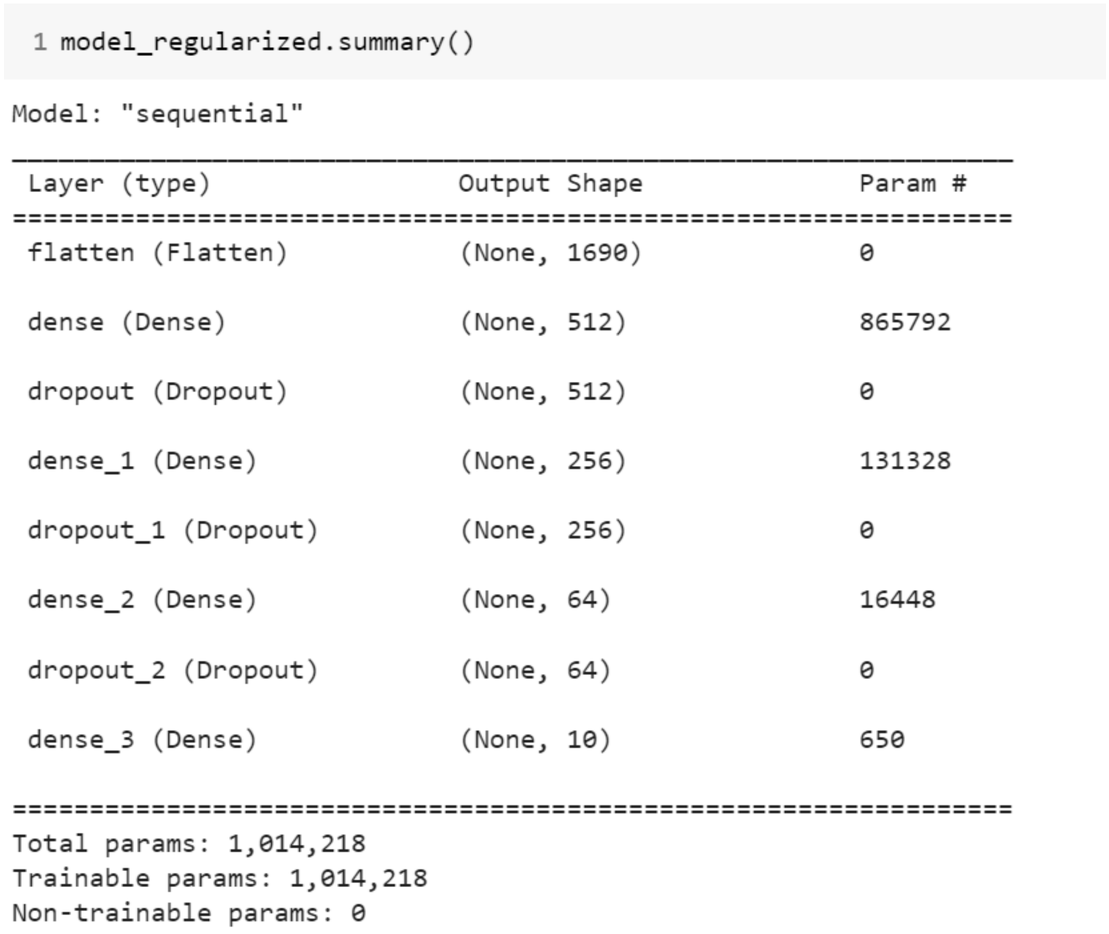
```
# build network topology
model_regularized = keras.Sequential([

    # input layer
    keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])),

    # 1st dense layer
    keras.layers.Dense(512, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # 2nd dense layer
    keras.layers.Dense(256, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

    # 3rd dense layer
    keras.layers.Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),
    keras.layers.Dropout(0.3),

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

Figure 3.37 DNN regularized model part_1



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

Figure 3.38 DNN regularized model part_1



```
model_regularized.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 1690)              0         
                                                                 
 dense (Dense)               (None, 512)               865792    
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 256)               131328    
                                                                 
 dropout_1 (Dropout)         (None, 256)               0         
                                                                 
 dense_2 (Dense)             (None, 64)                16448     
                                                                 
 dropout_2 (Dropout)         (None, 64)                0         
                                                                 
 dense_3 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 1,014,218
Trainable params: 1,014,218
Non-trainable params: 0
```

Figure 3.39 DNN regularized model summary

CNN model

It is a feed-forward artificial neural network. It uses multilayer perceptron’s which are designed to require minimal pre-processing. When applied to NLP for text classification the sentences in the document file are represented as a square matrix, each row of which corresponds to a vector which represents a word (word embedding). Then convolution is performed on the matrix using linear filters. A Convolutional Neural Network (CNN) provides outcomes that can be included into further training phases via matrix multiplication. This method's name is convolution Words in a sentence or a news item are represented as word vectors in the context of NLP. The training of a CNN is then done using these word vectors. By choosing a number of filters and size of the kernel, the training is done. CNN may have several dimensions.

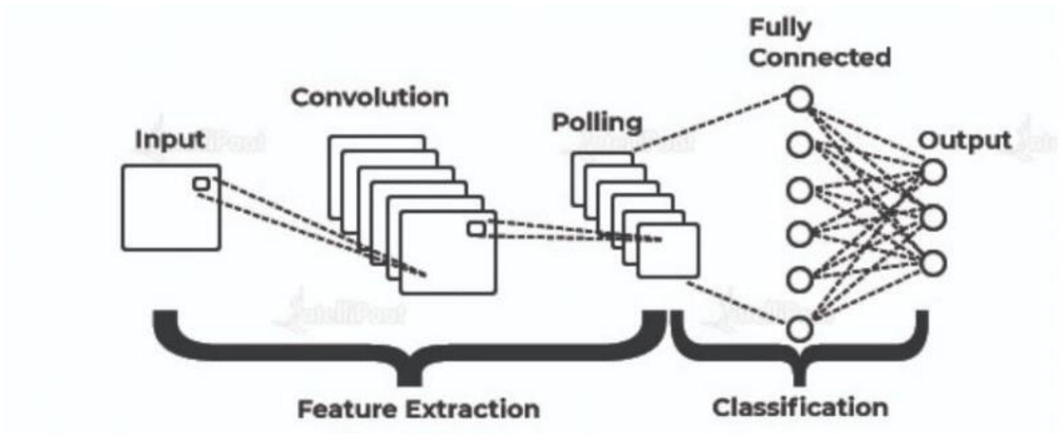


Figure 3.40 CNN

To classify music genres using deep learning, we can use convolutional neural networks (CNNs) which are designed to automatically learn and extract important features from the input data.

For music genre classification, the input data is usually represented as spectrograms or other time-frequency representations of audio signals, which capture the temporal and frequency characteristics of the audio. The convolutional layers in a CNN are responsible for learning local patterns and features from the input data, such as relevant spectral patterns and variations in different genres.

Pooling layers are used to downsample the feature maps obtained from the convolutional layers, which helps to reduce the dimensionality of the data and extract the most important features. Max pooling is commonly used in CNNs for music genre classification.

After the convolutional and pooling layers, the output is flattened and passed through one or more fully connected layers. These layers learn higher-level representations and capture the global dependencies between the learned features.

The final layer of the CNN is the output layer, which typically consists of multiple nodes corresponding to different music genres. The softmax activation

function is commonly used to obtain the probability distribution over the genres, indicating the predicted genre for a given input sample.

Overall, by using CNNs for music genre classification, we can automatically learn and extract relevant features from the spectrogram input, enabling accurate classification based on the learned patterns. CNNs have shown high performance in music genre classification tasks and have been widely used in research and industry applications.

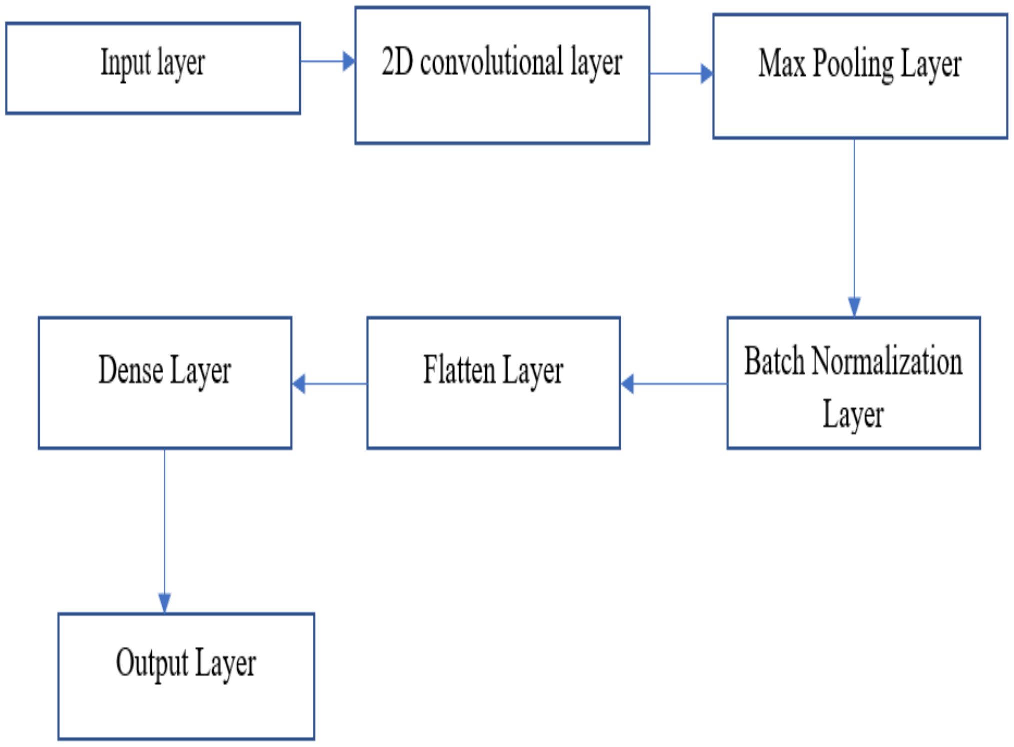
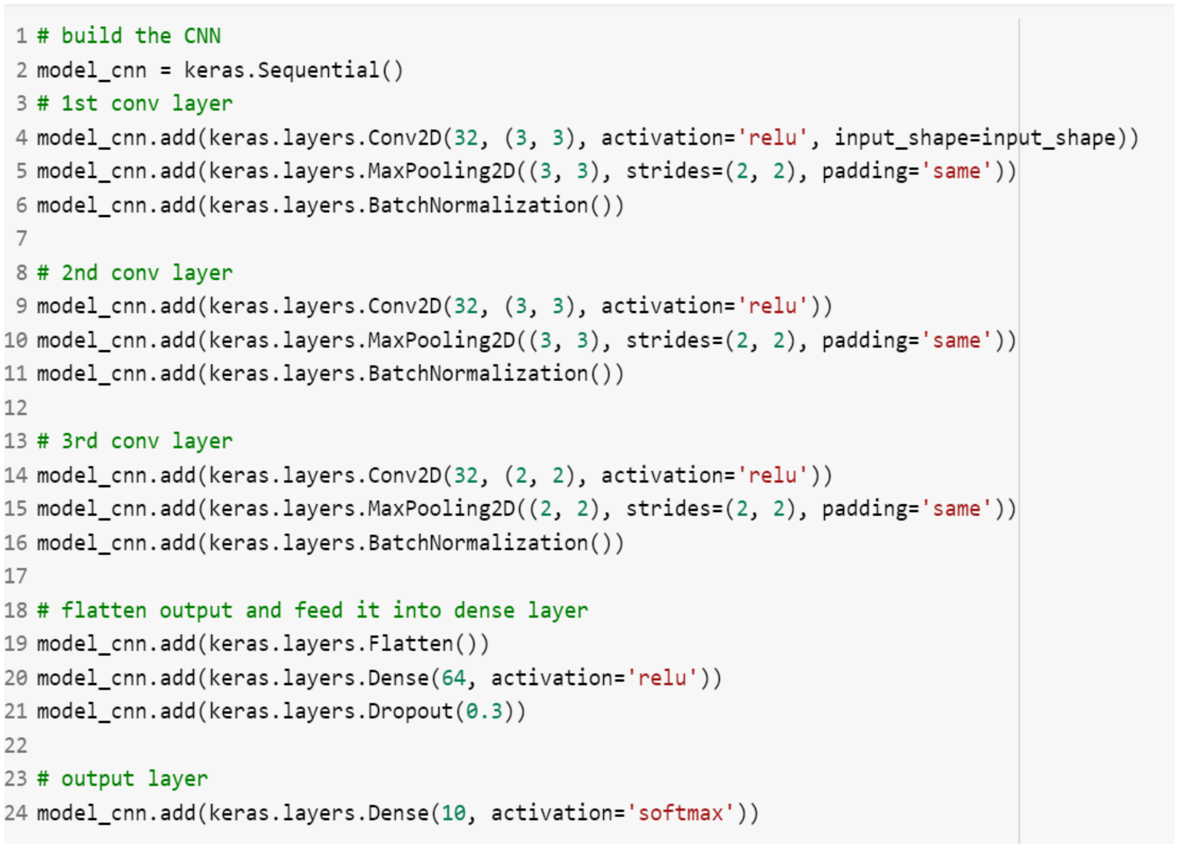


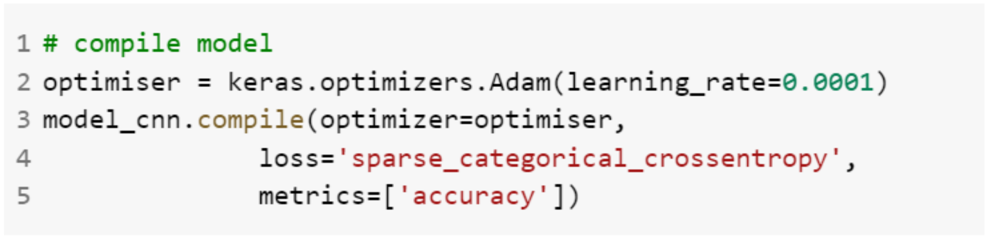
Figure 3.41 CNN layers

Implementation



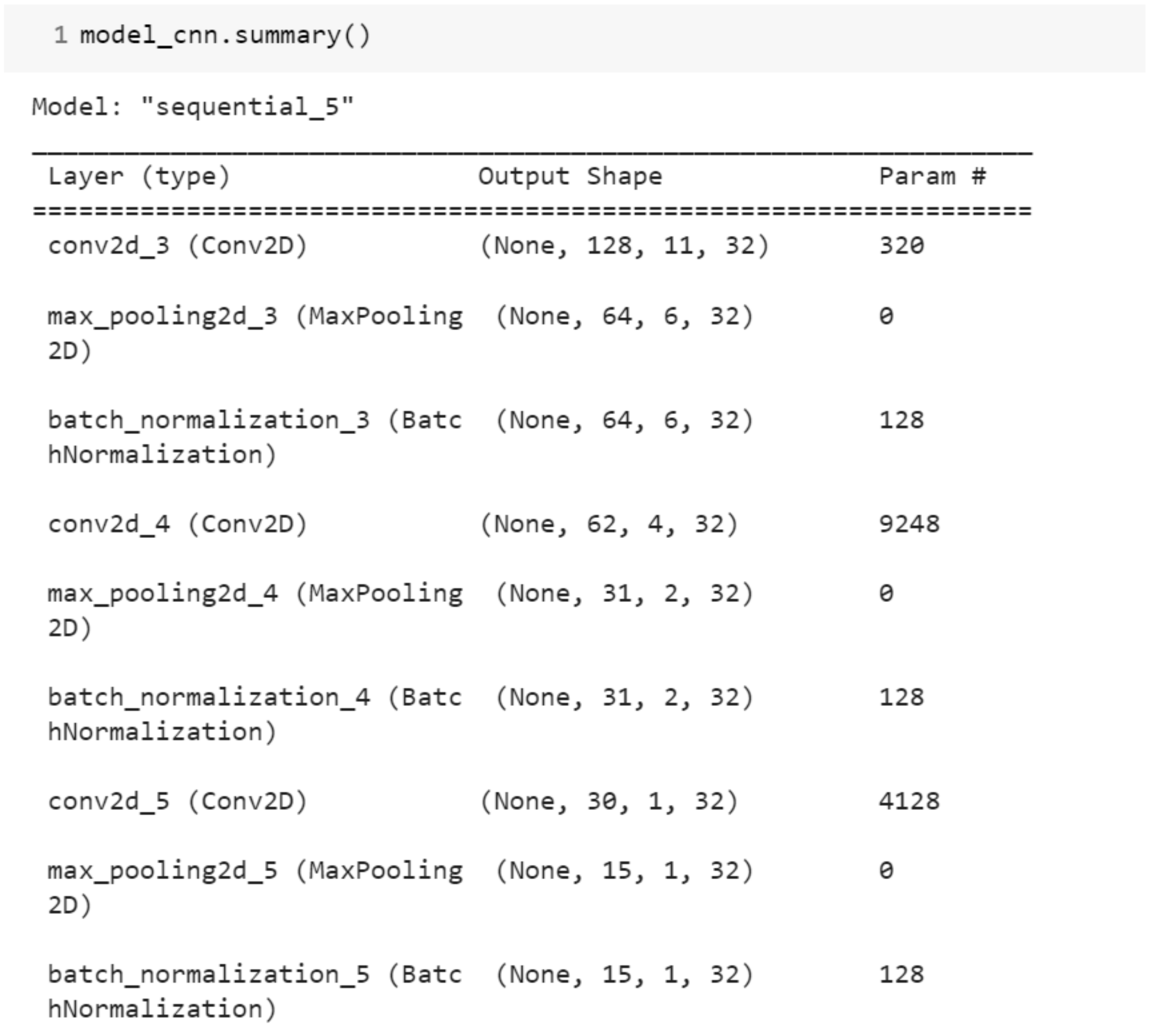
```python
# build the CNN
model_cnn = keras.Sequential()
# 1st conv layer
model_cnn.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model_cnn.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
model_cnn.add(keras.layers.BatchNormalization())

# 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'))
```

Figure 3.42 CNN model part_1



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

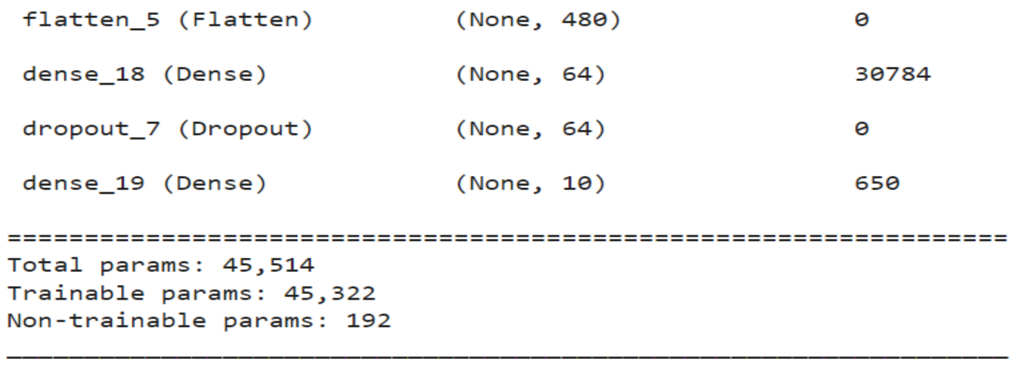
Figure 3.43 CNN model part_2



```
model_cnn.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 128, 11, 32)       320

 max_pooling2d_3 (MaxPooling  (None, 64, 6, 32)        0
 2D)

 batch_normalization_3 (Batc  (None, 64, 6, 32)        128
 hNormalization)

 conv2d_4 (Conv2D)           (None, 62, 4, 32)         9248

 max_pooling2d_4 (MaxPooling  (None, 31, 2, 32)        0
 2D)

 batch_normalization_4 (Batc  (None, 31, 2, 32)        128
 hNormalization)

 conv2d_5 (Conv2D)           (None, 30, 1, 32)         4128

 max_pooling2d_5 (MaxPooling  (None, 15, 1, 32)        0
 2D)

 batch_normalization_5 (Batc  (None, 15, 1, 32)        128
 hNormalization)
```

Figure 3.44 CNN model summary part_1



```
 flatten_5 (Flatten)         (None, 480)               0

 dense_18 (Dense)            (None, 64)                30784

 dropout_7 (Dropout)         (None, 64)                0

 dense_19 (Dense)            (None, 10)                650

=================================================================
Total params: 45,514
Trainable params: 45,322
Non-trainable params: 192
_________________________________________________________________
```

Figure 3.45 CNN model summary part_2

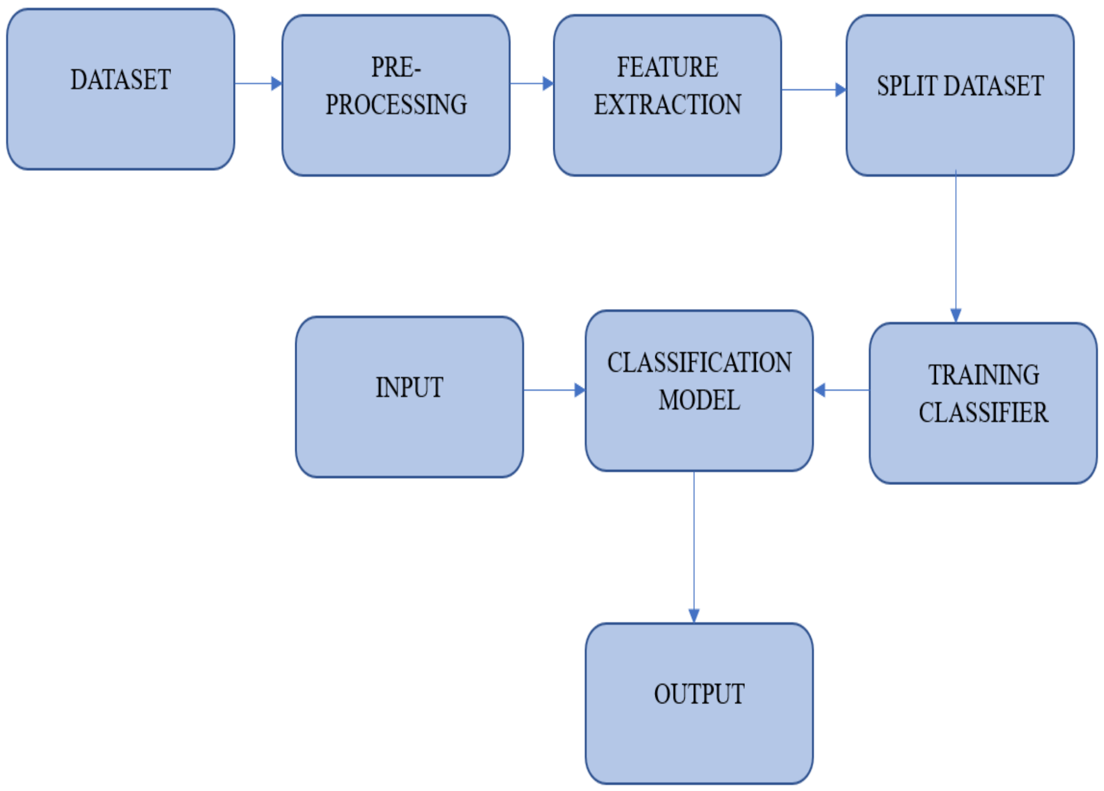


Figure 3.46 Flow of the project

3.7 Mathematics

1. ReLU function

The rectified linear activation function is a straightforward calculation that gives an immediate response of the value entered or 0.0 if the input is 0.0 or less.

$$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

Figure 3.47 Equation for ReLU function

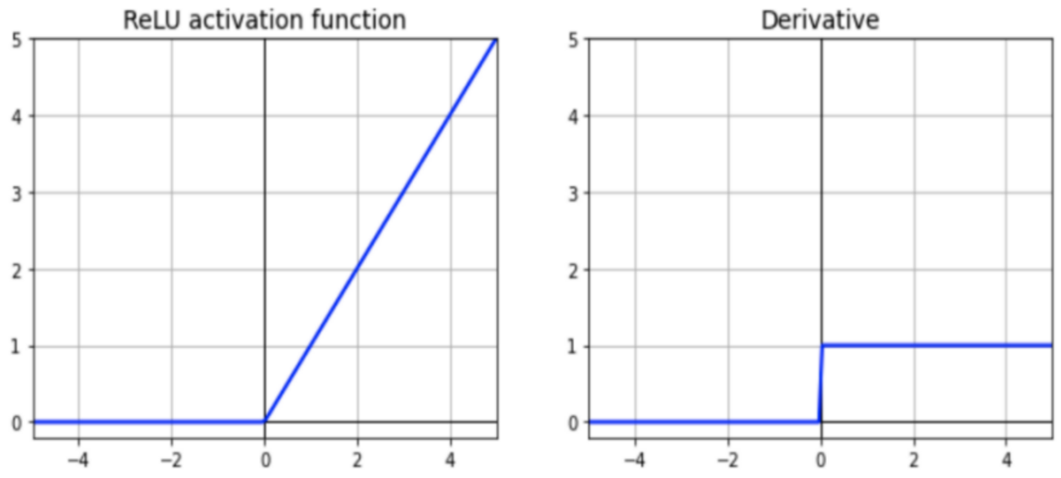


Figure 3.48 Plot of ReLU activation function and its derivative

Rectified Linear Activation Function Benefits

● To account for non-linearities, the ReLU function is divided graphically into two linear halves. If the slope of a function changes, it isn't linear. Negative inputs have 0 slope while all positive inputs have 1 slope.

● This function helps computes quite quickly.

2. Sigmoid Function

A mathematical function called the sigmoid function converts any input value to a number between 0 and 1. It is frequently used in machine learning as an activation function to provide nonlinearity to the model, particularly in logistic regression and neural networks. It is said that the sigmoid function is:

y(x) = 1 / (1 + e^(-x))

where x is the input value and e are Euler's number, a mathematical constant that is roughly equivalent to 2.71828. Since y(x) always has an output value between 0 and 1, it can be used to represent probabilities or binary classifications. When x is a negative number, the sigmoid function moves closer to zero. The sigmoid function approaches 1 when x is positive. The sigmoid function produces 0.5 when x is 0.

The sigmoid function is advantageous in machine learning because it possesses a number of favorable characteristics. Since it is differentiable, any point can be used to calculate its derivative. This qualifies it for optimization techniques like gradient descent. It always increases or always drops as x increases since it is monotonic. The sigmoid function, however, is prone to saturation, which implies that for extremely high or extremely low values of x, the function's gradient becomes extremely small. Due to slow convergence during training, the model may be unable to recognize complex patterns in the data. Alternative activation functions, like the Rectified Linear Unit (ReLU), have been created to address this problem.

3. Adam optimizer

Adam (Adaptive Moment Estimation) is a well-liked stochastic gradient descent (SGD) optimization technique used in machine learning. AdaGrad and RMSProp, two further optimization methods, are combined to create it. The first and second moments, which are the past gradients and past squared gradients, respectively, are maintained by the Adam optimizer. Based on the size of the gradient and the magnitude of the second moment of the gradient, the algorithm adjusts the learning rate for each parameter. As a result, the algorithm converges more quickly and consistently than conventional optimization algorithms. Based on the following equations, the Adam optimizer modifies the model's parameters:

m_t = beta1 * m_(t-1) + (1 - beta1) * g_t

v_t = beta2 * v_(t-1) + (1 - beta2) * (g_t ^ 2)

m_hat = m_t / (1 - beta1^t)

v_hat = v_t / (1 - beta2^t) t

heta_t = theta_(t-1) - alpha * m_hat / (sqrt(v_hat) + epsilon)

where g_t is the gradient at time step t, m_t and v_t is the first and second moments, m_hat and v_hat are the bias-corrected estimates of the first and second moments, theta_t is the updated parameter at time step t, alpha is the learning rate, beta1 and beta2 are the exponential decay rates for the first and second moments, and epsilon is a small value added to the denominator to avoid division by zero.

The Adam optimizer is superior to conventional optimization techniques in a number of ways. It is suitable for issues involving huge datasets and high-dimensional parameter spaces since it is computationally efficient, memory-intensive, and memory-constrained. Additionally, it offers adaptive learning rates, which can help the model's convergence and generalization. To attain the best performance, it is crucial to precisely set the optimizer's hyperparameters, including the learning rate, decay rates, and epsilon.

Chapter 04: Experiments and Results Analysis

The following parameters have been used to assess the prediction results:

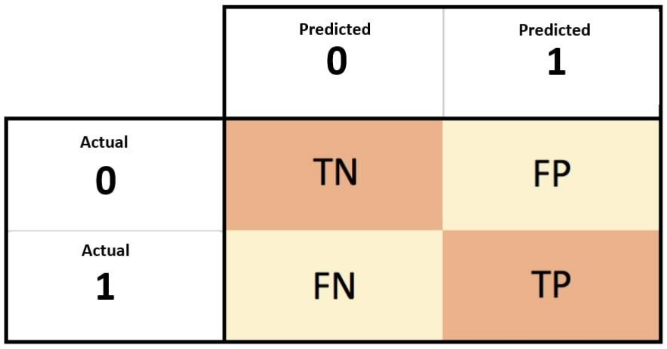
Accuracy: Out of the total number of points in a model, the points which are correctly classified is called accuracy.

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$

Where, TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative

Learning curves: These are charts that display the model's accuracy or loss with time during training and validation. They can be used to determine the ideal number of training epochs as well as to diagnose underfitting or overfitting.

Confusion Matrix : An analysis of a machine learning model's performance on a set of test data is summarized by a confusion matrix. It is frequently used to assess how well classification models work. These models try to predict a categorical label for each input event. The matrix shows how many true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) the model generated using the test data.



	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Figure 4.1 Example of Confusion matrix

AUC-ROC Curve : An indicatorof performance for classification problems. AUC stands for the level or measurement of separability, and ROC is a probability curve. It reveals how well the model can differentiate across classes. The model is more accurate at classifying 0 classes as 0, and classifying 1 classes as 1, the higher the AUC.so we can say, the more AUC, the more effective the model is at classifying.

The ROC curve is a graph that plots the True Positive Rate (TPR) against the False Positive Rate (FPR), with the TPR on the y-axis and the FPR on the x-axis.

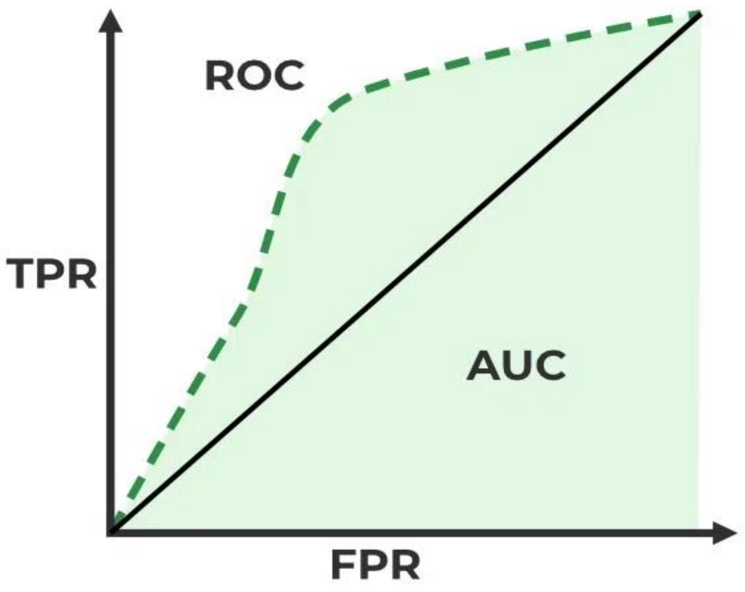


Figure 4.2 Typical ROC

True Positive Rate(TPR): True Positive/positive   or **TPR**=TP/TP+FN

False Positive Rate(FPR): False Positive /Negative   or **FPR**=FP/TN+FP

An AUC near to 1 indicates a high level of separability, which is a sign of a good model. An AUC that is near to 0, the poorest measure of separability, denotes a subpar model. Actually, it suggests that the outcome is being reversed. It forecasts that both 1s and 0s will be 1. Additionally, the model has zero capability for class separation if AUC is 0.5.

When we have a problem where we need to classify data into multiple categories, we can use the One vs. All technique to create graphs called AUC ROC curves. For example if we have four categories A,B,C and D then we will have four ROC curve for each one of them, ROC curve corresponding A will show how good is model at distinguishing A from B,C and D and same for all other categories.

4.1 Discussion on the Results

Two different algorithms are used :

DNN(Deep Neural Network)

CNN(Convolutional Neural Network)

Accuracy using DNN is 72% for testing dataset. Accuracy using CNN is 82% for testing dataset .

In this project, we have trained two models one of them is ANN and other is CNN while we were training our ANN model we came across overfititting and its quite difficult to avoid overfitting we to use regularization ,we used l2 regularization we have performed Hyper Parameter Tuning to enhance the performance of our model . The Hyper Parameter Tuning process involves adjusting the model's hyperparameters to achieve better accuracy and optimize its performance. In our case, we have focused on tuning the regularization and penalty parameters, which are key hyperparameters that significantly affect the model's performance.

Regularization process has been meticulously carried out to ensure that the model's accuracy is maximized, while also avoiding overfitting, which is a common challenge in machine learning. The results before regularization is as shown in Figure 4.3 and the results after the regularization process are depicted in Figure 4.4.

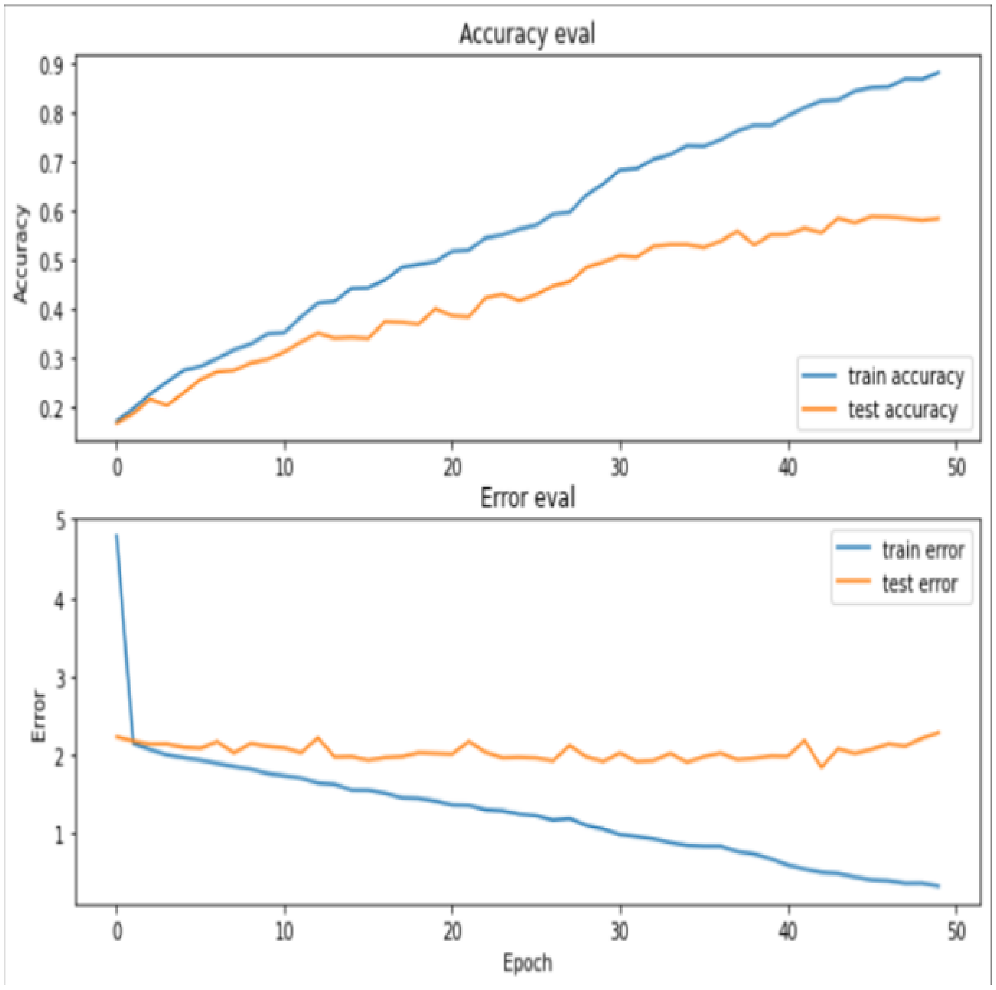


Figure 4.3 Accuracy/Loss VS Epoch Graph of DNN Model

In the above figure we can clearly see that accuray of the training dataset is increasing significantly as the no. of epochs are increasing as well as the error is also decreasing at very significant rate but as soon as we see accuracy for test dataset we can see that accuray is increasing and decreasing and after a point there is no significant change in the accuracy and if its not clear by that we can see that error rarely reduced questioning our trained model , this called for regularizing our model as our model is overfited that means models gives accurate results for the training dataset but when tested on test dataset its performance is not up to the mark.

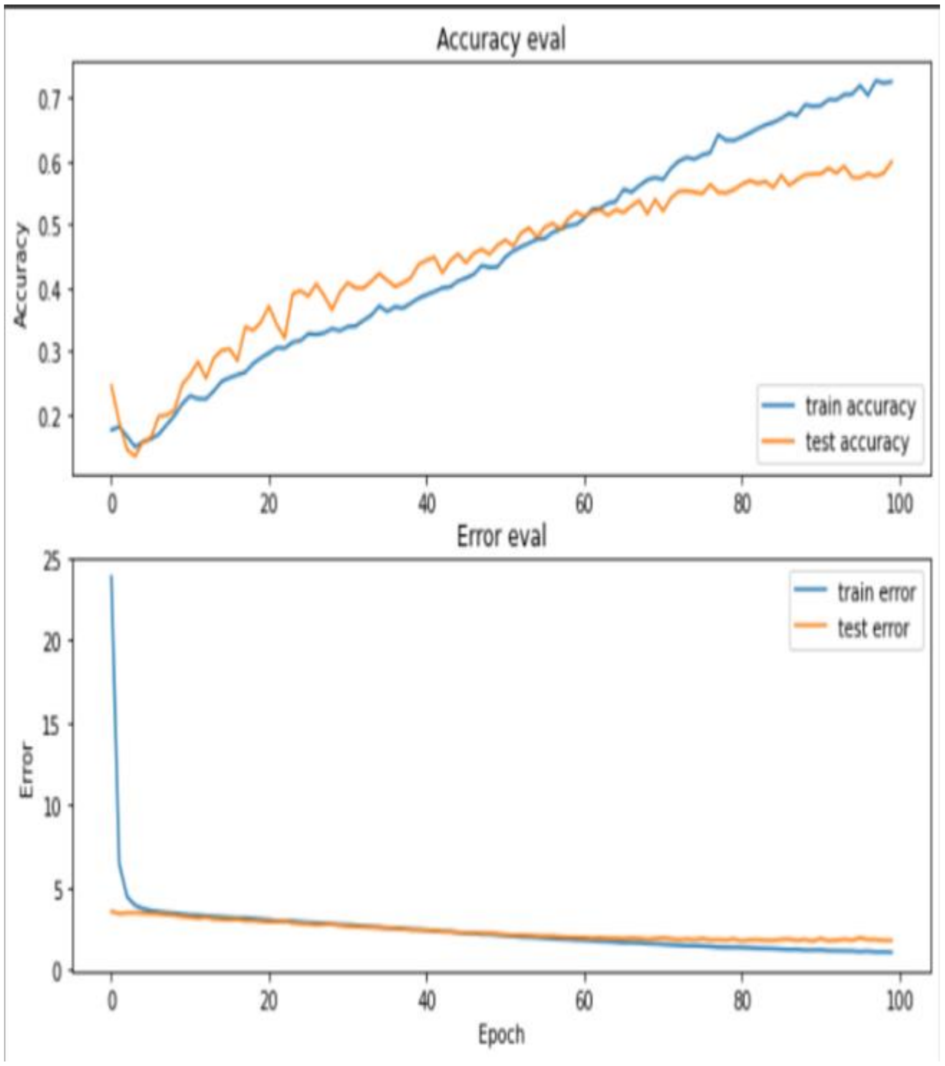


Figure 4.4 : Accuracy/Loss VS Epoch Graph of DNN regularized Model

In the above Figure 4.4 its quite visible that overfitting has been taken care of as the model work fine on both trained and test datset and it was done with the help of l2 regularization . The most popular regularization types are L1 and L2. These add a new term known as the regularisation term and update the general cost function.

Cost function = Loss + Regularization term

Because it is assumed that a neural network with smaller weight matrices results in simpler models, the inclusion of this regularisation term causes the values of weight matrices to drop. As a result, it will also significantly lessen overfitting.L1 and L2 differ on the basis that in case of L1 the weight may decay to zero but in case of L2 it forces to decay toward zero but its never decays to zero.

CNN model

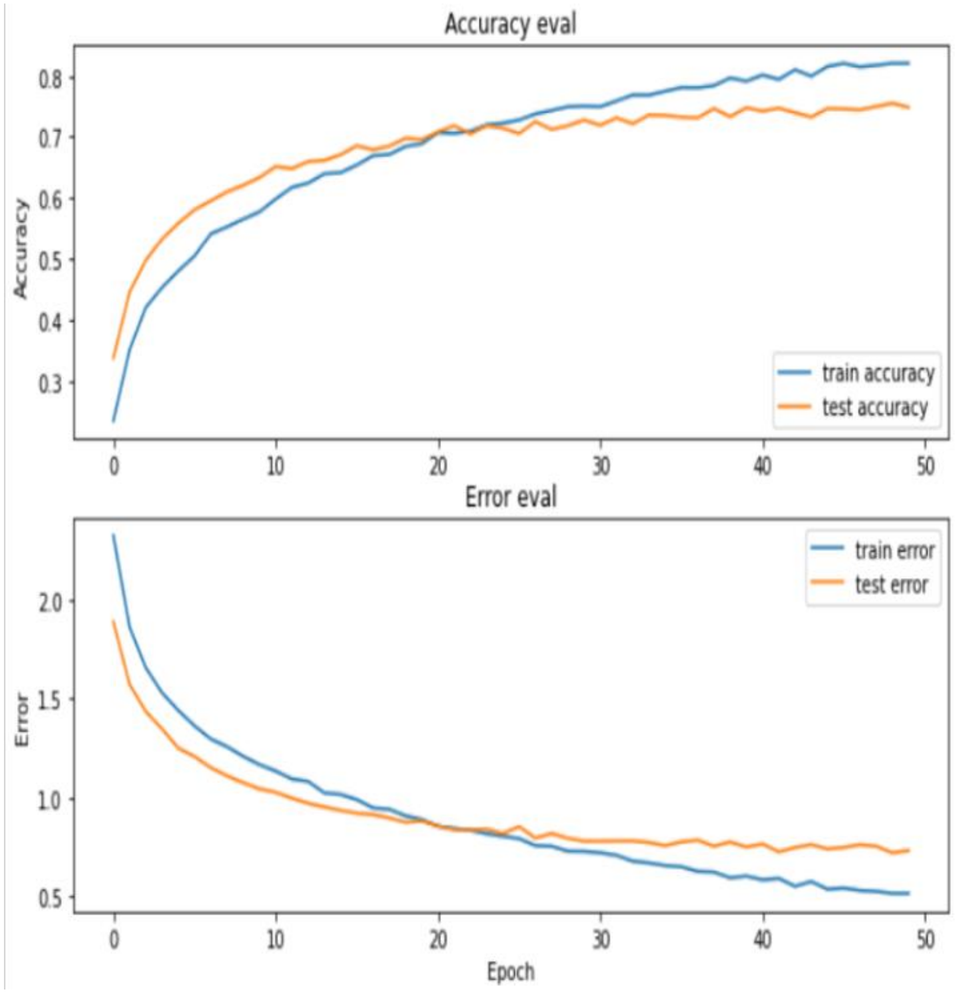
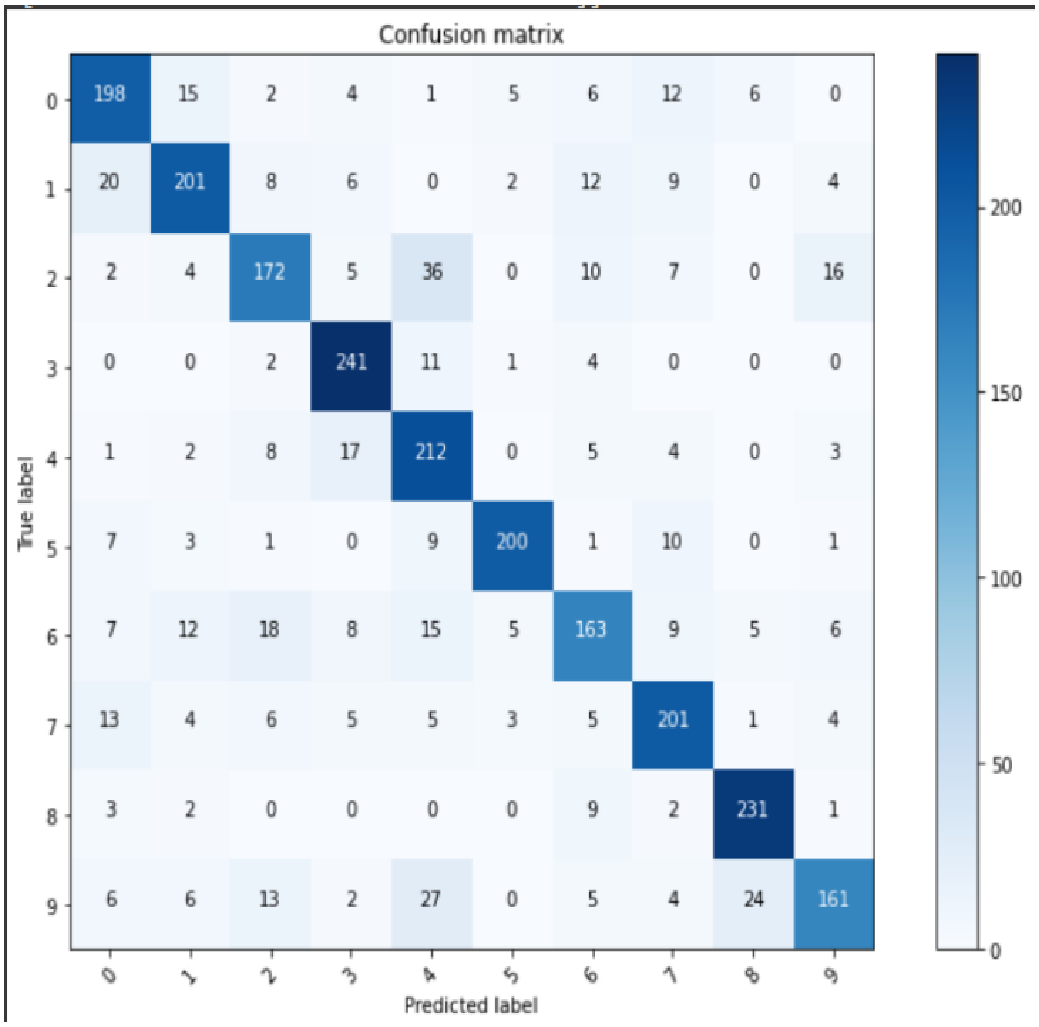


Figure 4.5 Accuracy/Loss VS Epoch Graph of CNN Model

The Figure 4.5 shows the learning curve of the CNN model and its behaviour is almost similar on both test and train dataset .



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	198	15	2	4	1	5	6	12	6	0
1	20	201	8	6	0	2	12	9	0	4
2	2	4	172	5	36	0	10	7	0	16
3	0	0	2	241	11	1	4	0	0	0
4	1	2	8	17	212	0	5	4	0	3
5	7	3	1	0	9	200	1	10	0	1
6	7	12	18	8	15	5	163	9	5	6
7	13	4	6	5	5	3	5	201	1	4
8	3	2	0	0	0	0	9	2	231	1
9	6	6	13	2	27	0	5	4	24	161

Figure 4.6 Confusion Matrix

Figure 4.6 shows the confusion matrix for each 10 gerne. Confusion Matrix is visible between True label and Predicted label. The diagonal elements which are darker are correctly predicted records and the rest are incorrectly classified which can also be visualized.

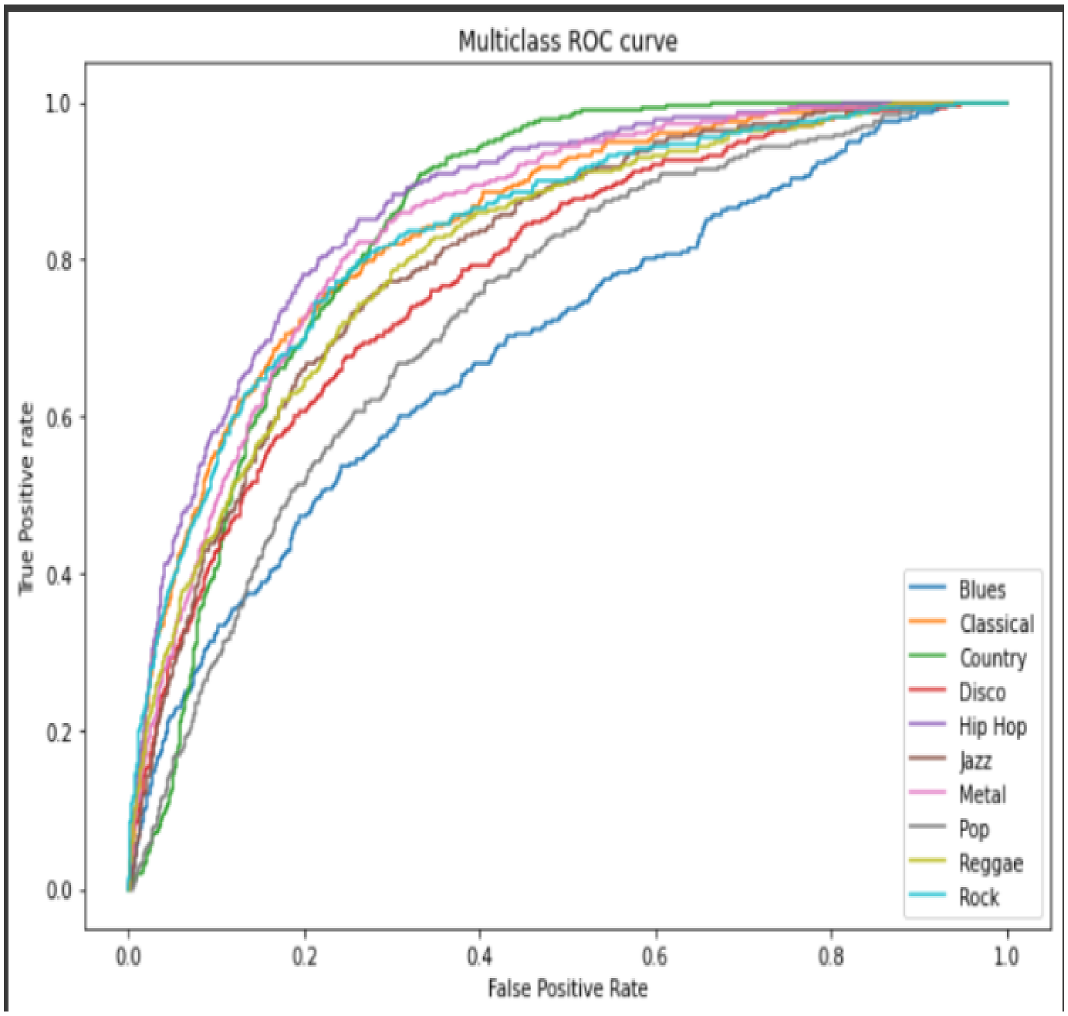


Figure 4.7 ROC-AUC curve

The Figure 4.7 shows ROC-AUC curve,its plays a significant role in evaluating the performance ofa classification model by showing the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) for different classification thresholds. To determine the optimal classification threshold based on specific objectives, we plotted the ROC curves for each class.. By analyzing these curves, we can make more informed decisions about the performance of our classification model.

The ROC Curve displayed in Figure 4.8 provides us with valuable insights into the classification performance of the model. We can observe that the classifier model doesnot work that great for blues, pop and disco while its really good at classifying for country, hip hop and metal rightly ,it is so as the AUC of these is high compared to the others.

Overall, the ROC curve provides us with a clear understanding of the model's classification performance.

Chapter 05: Conclusion

5.1 Conclusion

In conclusion, deep learning model-based trials for classifying music gerne has produced encouraging findings. A 82% accuracy was attained by the CNN model, a 72% accuracy by the ANN model. These models' high accuracy demonstrates that deep learning models are useful for classifying music gerne. It is crucial to remember that accuracy does not, by itself, give a full picture of the model's performance. Further investigation could be conducted in order to enhance the functionality of these models. To get even better results, this can entail experimenting with various architectures or fine-tuning the hyperparameters. Overall, these studies' findings imply that CNN, ANN deep learning models, in particular, have potential for accurate music classification .

5.2 Applications

Music genre classification is an important tool that has many practical applications across various domains. For example, it is crucial for building effective music recommendation systems. By analyzing a user's favorite songs or listening history, recommendation algorithms can suggest relevant music from the same or similar genres, enhancing user satisfaction and engagement.

Moreover, genre classification plays an important role in organizing and curating music libraries. Streaming platforms and music curators use genre labels to create themed playlists or radio stations, catering to specific moods, occasions, or preferences. By accurately classifying music into genres, curators can create playlists that appeal to specific audiences, providing a diverse and engaging music selection.

Music genre classification also aids in music analysis and market research. By analyzing the distribution of genres in music libraries or streaming platforms, music industry professionals can gain insights into popular genres, emerging trends, and consumer preferences. This information can drive decision-making processes such as content curation, marketing strategies, and identifying potential gaps or opportunities in the music market.

Additionally, music genre classification can be applied in areas such as music therapy, content licensing and copyright management, radio broadcasting, and music education. In summary, accurate and robust music genre classification models enable a wide range of applications that enhance user experiences, facilitate content organization, and provide valuable insights to the music industry.

5.3 Future Scope and Limitations

The future of music genre classification looks very promising, and there are many exciting possibilities for advancements and innovations in this field. For example, we can expect to see more sophisticated music recommendation systems that take into account factors like mood, tempo, instrumentation, and lyrics analysis. This could lead to more personalized and precise music recommendations that are tailored to each user's specific preferences and context.

In addition, future genre classification models could focus on identifying and categorizing hybrid genres or subgenres, reflecting the increasingly diverse and eclectic nature of music. We may also see real-time genre detection in live performances or streaming environments, allowing for dynamic and adaptive music experiences that respond to the changing musical characteristics during a performance.

Other potential developments include cross-modal genre classification, which incorporates other data sources like music album covers, artist information, or lyrics, to provide a more holistic understanding of music genres. Furthermore, future genre classification models could be trained to identify and understand emerging genres, facilitating their recognition, categorization, and discovery by music enthusiasts.

However, there are some limitations to consider as well. For example, deep learning models can be complex and difficult to interpret, and they require large labeled datasets to achieve high accuracy. Additionally, accurately representing evolving genres and accounting for subjective interpretations can be challenging, and there may be cultural and regional variations that need to be accounted for in genre classification models.

Overall, the future of music genre classification is exciting and holds a lot of potential, but it's important to strike a balance between technological advancements and contextual understanding to ensure that these models are accurate, inclusive, and adaptable.

References

[1] N. Pelchat and C. M. Gelowitz, "Neural Network Music Genre Classification," 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), Edmonton, AB, Canada, 2019, pp. 1-4, doi: 10.1109/CCECE.2019.8861555.

[2] R. J. M. Quinto, R. O. Atienza and N. M. C. Tiglao, "Jazz music sub-genre classification using deep learning," TENCON 2017 - 2017 IEEE Region 10 Conference, Penang, Malaysia, 2017, pp. 3111-3116, doi: 10.1109/TENCON.2017.8228396.

[3] P. Fulzele, R. Singh, N. Kaushik and K. Pandey, "A Hybrid Model for Music Genre Classification Using LSTM and SVM," 2018 Eleventh International Conference on Contemporary Computing (IC3), Noida, India, 2018, pp. 1-3, doi: 10.1109/IC3.2018.8530557.

[4] Y. -H. Cheng, P. -C. Chang and C. -N. Kuo, "Convolutional Neural Networks Approach for Music Genre Classification," 2020 International Symposium on Computer, Consumer and Control (IS3C), Taichung City, Taiwan, 2020, pp. 399-403, doi: 10.1109/IS3C50286.2020.00109.

[5] J. Mehta, D. Gandhi, G. Thakur and P. Kanani, "Music Genre Classification using Transfer Learning on log-based MEL Spectrogram," 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2021, pp. 1101-1107, doi: 10.1109/ICCMC51019.2021.9418035.

[6] B. Logan et al., “Mel frequency cepstral coefficients for music modeling.,” in ISMIR, 2000

[7] M. Haggblade, Y. Hong, and K. Kao, “Music genre classification,” Department of Computer Science, Stanford University, 2011.

[8] P. Kozakowski and B. Michalak, “Deep sound: Music genre recognition,” 2016

[9] A. Kumar, A. Rajpal and D. Rathore, "Genre Classification using Feature Extraction and Deep Learning Techniques," 2018 10th International Conference on Knowledge and Systems Engineering (KSE), Ho Chi Minh City, Vietnam, 2018, pp. 175-180, doi: 10.1109/KSE.2018.8573325.

[10] B.S. Vogler and A. Othman, "Music Genre Recognition", Benediktsvogler.com,2016.

[11] P. Hamel and D. Eck, "Learning features from music audio with deep belief networks", in 11th International Society for Music Information Retrieval Conference, ISMIR, 2010, pp. 339-344.

[12] Choi, K., Fazekas, G., & Sandler, M. (2017). Convolutional recurrent neural networks for music classification. In Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR), Suzhou, China

[13] Nirmal M R, “Music Genre Classification using Spectrograms”, International Conference on Power, Instrumentation, Control and Computing (PICC) – 2020.

[14] Yang Yu, Sen Luo, Shenglan Liu, Hong Qiao, Yang Liu, and Lin Feng. 2020. Deep attention based music genre classification. Neurocomputing 372 (2020), 84–91.

[15] Zanaty, E. A., & O'Connor, N. E. (2015). Exploring deep learning for music genre classification. In Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR) (pp. 469-475).

[16] Fernández, S., & Herrera, P. (2018). Music Genre Classification Using Convolutional Neural Networks: How Transferable Are Features Learned from Speech? In 2018 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-7).

[17] Tzanetakis, G., Cook, P.: ‘Musical genre classification of audio signal’, IEEE Trans. Speech Audio Process., 2002, 10, (3), pp. 293– 302

[18] Silla Jr., C. N., Koerich, A. L., & Ghosh, J. (2011). Multi-label music genre classification from audio, text, and images using deep features. Journal of the Association for Information Science and Technology, 62(4), 745-762.

[19] Pons, J., Serra, X., & Herrera, P. (2017). Experimenting with musically motivated convolutional neural networks. IEEE Transactions on Multimedia, 19(8), 1750-1760.

[20] Li, T., Tzanetakis, G.: ‘ Factors in automatic musical genre classification of audio signals’. Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Platz, NY, USA, 19–22 October 2003

[21] Panteli, M., Foster, P., & Sandler, M. (2014). Timbre analysis of music audio signals with convolutional neural networks. In Proceedings of the 15th International Society for Music Information Retrieval Conference (ISMIR) (pp. 525-530).

[22] Jiang, D., Deng, L., & Huang, X. (2012). Advances in spectral and timbral features for music genre classification. Journal of the Acoustical Society of America, 132(5), 3346-3357.

[23] Korzeniowski, F., & Widmer, G. (2016). Feature learning for chord recognition: The deep chroma extractor. In Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR) (pp. 334-340).

[24] Han, S., Heo, H., Kim, H., & Kim, J. (2017). Music genre classification based on convolutional neural networks with multi-resolution processing. In Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR) (pp. 270-276).

[25] K. He, X. Zhang, and S. Ren, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition

[26] Turney, P. D. (2006). Measuring semantic similarity by latent relational analysis. In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI).

[27] Nieto, O., Bello, J. P., & Reiss, J. D. (2016). Automatic music genre classification using wavelet packets and the ensemble of classifiers. In 2016 24th European Signal Processing Conference (EUSIPCO).

[28] Liang, J., & Huang, X. (2018). Music Genre Classification Based on Convolutional Neural Network and Content-Based Features. In 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics.

[29] J. Donahue, L. A. Hendricks, S. Guadarrama, S. Venugopalan, S. Guadarrama, and K. Saenko, "Long-term Recurrent Convolutional Networks for Visual Recognition and description," in Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern RecognitionCVPR.

[30] Wang, J., Yang, Y., Xu, J., & Yu, K. (2016). Music Genre Classification via Joint Recurrent and Convolutional Networks. In Advances in Neural Information Processing Systems (NIPS).

[31] Mingwen Dong, "Convolutional Neural Network Achieves Human-level Accuracy in Music Genre Classification", arXiv:1802.09697v1 [cs.SD] – 2018.

[32] Fuentes, M., & Lopez, F. J. (2019). Music Genre Classification Using Convolutional Neural Networks. In Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR).

[33] G. Mishra, A., & Pattnaik, P. K. (2020). Music Genre Classification Using Deep Learning Techniques. In 2020 5th International Conference on Intelligent Computing and Control Systems (ICICCS).

[34] Kim, Y., Lee, K., & Oh, S. H. (2018). Convolutional Recurrent Neural Networks for Music Genre Classification. In 2018 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 204-207).

[35] Hamel, P., & Eck, D. (2010). Learning features from music audio with deep belief networks. In Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR) (pp. 339-344)