MUSIC CLASSIFICATION SYSTEM BASED ON GENRES USING DEEP LEARNING ALGORITHMS

A PROJECT REPORT

Submitted by

ABHAY KUMAR PATEL [RA1911029010024]
ADITHYA.S.MENON [RA1911029010025]

Under the Guidance of

Dr. B. Balakiruthiga
Assistant Professor, Department of Networking and Communications

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in COMPUTER NETWORKING

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR-603 203

MAY 2023

SRM INSTITUTE OF SCIENCE & TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled **"MUSIC CLASSIFICATION SYSTEM BASED ON GENRE USING DEEP LEARNING ALGORITHMS"** is the bonafide work of **Mr. ABHAY KUMAR PATEL and Mr. ADITHYA S MENON** who carried out the project work under my/our supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

[illegible] 16/8

SIGNATURE

Dr. B. BALAKIRUTHIGA
SUPERVISOR
Assistant Professor
Department of Networking and Communications

Dept. of Networking and Communications
Kattankulathur 603 203
SRM IST

[illegible] 16/8/23

SIGNATURE

Dr. ANNAPURANI.K
HEAD OF THE DEPARTMENT
Department of Networking and Communications

R. [illegible] 18/8/23

Signature of the Internal Examiner

[illegible] 18

Signature of the External Examiner

Department of Networking and Communications
SRM Institute of Science &Technology
Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must besigned and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

<u>To be completed by the student for all assessments</u>

Degree/ Course	**: B.TECH/CSE-CN**
Student Name	**: ABHAY KUMAR PATEL, ADITHYA.S.MENON**
Registration Number	**: RA1911029010024, RA1911029010025**

Title of Work : MUSIC CLASSIFICATION SYSTEM BASED ON GENRES USING DEEP LEARNING ALGORITHMS

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g.fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook /University website

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

We are aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

Abhay Kumar Patel (RA1911029010024)
Adithya.S.Menon (RA1911029010025)

(RA1911029010024)
Abhay

(RA1911029010025)
[illegible]

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr T.V.Gopal**, for his valuable support.

We wish to thank **Dr Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work. We are incredibly grateful to our Head of the Department**, Dr. Annapurani.K.,** Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our program coordinators **Dr. K. Venkatesh,** Associate Professor, and Panel Head, **Dr. T. Balachander,** Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for their inputs during the project reviews and support. We register our immeasurable thanks to our Faculty Advisor**, Dr. R. Naresh,** Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Dr. B. Balakiruthiga,** Associate Professor, Department of Networking and Communications, SRM Institute of Science and Technology, for providing me with an opportunity to pursue my project under his mentorship. He provided me with the freedom and support to explore the research topics of my interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Networking and Communications Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Abhay Kumar Patel (RA1911029010024)

Adithya.S.Menon (RA1911029010025)

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	vii
	LIST OF TABLES	viii
	LIST OF FIGURES	ix
	LIST OF ABBREVIATIONS	xi
1.	**INTRODUCTION**	1
	1.1 OVERVIEW	1
	1.2 RESEARCH OBJECTIVE	1
	1.3 PROBLEM STATEMENT	2
	1.4 INNOVATION OF IDEA	2
	1.5 PURPOSE OF PROJECT	2
	1.6 ARTIFICIAL INTELLIGENCE – A BRIEF INTRODUCTION	3
	1.7 MACHINE AND DEEP LEARNING – AN OVERVIEW	4
	1.8 FLOW OF CHAPTERS	8
	1.9 MOTIVATION	9
2	**LITERATURE REVIEW**	11
3	**PROPOSED METHODOLOGY**	14
	3.1 OVERVIEW	14
	3.2 CONVOLUTIONAL NEURAL NETWORKS	14
	3.2.1 REASONS TO USE CNN	17
	3.2.2 DISADVANTAGES	18
	3.3 RECURRENT NEURAL NETWORKS - GRU	18
	3.4 THE FUZZY KNN MODEL	21
4	**MODULES AND IMPLEMENTATION**	24
	4.1 ARCHITECTURE DIAGRAM – CNN & CRNN	24
	4.2 ARCHITECUTER DIAGRAM – FUZZY KNN	25
	4.3 USE CASE DIAGRAM	26

	4.4 MODULES UNDERTAKEN – CNN & CRNN	27
	4.5 MODULES UNDERTAKEN – FUZZY KNN	30
	4.6 IMPLEMENTATION – CNN & CRNN	31
	4.6.1 CREATING THE JSON FILE	31
	4.6.2 LOADING THE JSON FILE AND SETTING UP INPUTS	35
	4.6.3 PREPARING THE DATASET	36
	4.6.4 CONSTRUCTING THE NEURAL NETWORK MODEL	37
	4.6.5 MAIN FUNCTION	38
	4.6.6 PREDICTING THE RESULT	39
	4.6.7 PREDICTING AN UNKNOWN SONG	40
	4.6.8 RECOMMENDING SIMILAR SONGS	41
	4.7 IMPLEMENTATION – FUZZY KNN	42
	4.7.1 CREATING THE BINARY FILE	42
	4.7.2 DISTANCE FUNCTION	43
	4.7.3 FUNCTION TO GET THE NEIGHBORS	43
	4.7.4 PREDICTION OF SONGS	45
5	**RESULT AND DISCUSSION**	46
	5.1 ACCURACY OF CNN	46
	5.2 ACCURACY OF CRNN	47
	5.3 ACCURACY OF FUZZY KNN	50
6	**CONCLUSION AND FUTURE WORK**	52
	REFERENCES	54
	APPENDIX	56
	PLAGIARISM REPORT	74
	PAPER SUBMISSION PROOF	80

ABSTRACT

In our current era there are lot of song of different genre and it is difficult to classify all the songs present easily by humans and it is important to classify the genre of songs as everyone like different type of songs and we want to make it easy for users to choose their favourite genre easily and it is very difficult to do it all manually so we are using the machine learning concept to do it by the help of machine which will be very easy and effortless for us to do.

In recent years deep neural network have been shown to be effective in many classification tasks. So we are proposing to make a music genre classification system for classifying the music. For making this system we are using deep learning algorithms like CNN, RNN, KNN and their hybrid models like CRNN and Fuzzy-KNN. The models will be individually evaluated in terms of the accuracy percentage metrics. The traditional models and it is hybrids are taken into consideration to observe how each model overcomes the shortcomings of the previously used models. Our deep learning model will be trained solely using the spectrograms received by the audio samples extracted from the GTZAN dataset. We will feed sufficient amount of audio data to train our model for its proper functioning and accuracy. And then with the result of the genre we suggest similar music to the user using various python libraries.

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
5.1	The Dataset	46
5.2	Accuracy of the Training and Testing data of CNN model	47
5.3	Accuracy of the Training and Testing data of CRNN model	47
5.4	Parameters for Training and Classification	49

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
1.1	A representation of Supervised Learning	6
1.2	A representation of Unsupervised Learning	6
1.3	A representation of Reinforced Learning	7
3.1	Convolutional Neural Networks architecture	16
3.2	Update Gate architecture	19
3.3	Reset Gate architecture	20
3.4	Formula for Current memory that will use the reset gate to store relevant information	23
3.5	The Formula used for Fuzzy KNN implementation	24
4.1	Flow of Modules	25
4.2	Flow of Modules – Fuzzy KNN	26
4.3	UML USE CASE Diagram	31
4.4	Including necessary libraries	32
4.5	Making the dictionary	32
4.6	Mapping the genres	33
4.7	Traversing through the dataset files	33
4.8	Appending the MFCC and Labels into the JSON file	34
4.9	Transferring all the data into the JSON file	34
4.10	Execution of the JSON file transfer	35
4.11	Loading the JSON file	36
4.12	Preparing the dataset by splitting the data	37
4.13	The structure of the Neural Network Model	38
4.14	The main function	39
4.15	The prediction function	40
4.16	Including necessary libraries	40
4.17	Predicting the unknown sample	41
4.18	Suggesting the music	42
4.19	The python libraries for the program	42

4.20	Extraction of features from the data	43
4.21	The ‘distance’ function	43
4.22	The ‘getNeighbors_fuzzy’ function	44
4.23	The ‘getNeighbors_fuzzy’ function (continued)	45
4.24	The Prediction code	48
5.1	Model predicting 8 out of 10 random songs from the testing data correctly	49
5.2	The graph plotted between Training and Testing accuracies and error percentages	50
5.3	The Fuzzy KNN model accuracy	50
5.4	The sub-genres predicted by the CRNN model	51
5.5	The sub-genres predicted by the Fuzzy KNN model	63
7.1	Loading dataset	64
7.2	Function to plot graph	64
7.3	Preparing dataset	65
7.4	Creating the neural network	66
7.5	Main Function	66
7.6	Function to predict songs	67
7.7	Testing on random songs	67
7.8	Extracting features of unknown songs	68
7.9	Predicting unknown songs	69
7.10	Importing the Libraries	70
7.11	The function to split the dataset	71
7.12	The function to calculate the distances	71
7.13	The function to calculate and display the nearest classes after applying fuzzy algorithm	72
7.14	The function to calculate and display the nearest classes after applying fuzzy algorithm (continued)	72
7.15	The function to calculate the accuracies	73
7.16	The function to predict the genres	73

LIST OF ABBREVIATIONS

ABBREVIATIONS	**Expansion**
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CRNN	Convolutional Recurrent Neural Network
GRU	Gated Recurrent Unit
i.e	That is
ML	Machine Learning
LSTM	Long Short Term Memory
w.r.t	With Respect T

CHAPTER 1

INTRODUCTION

Machine learning is now widely employed in our daily lives and is quite popular in the modern world. Based on the specifications of the application that the user has requested, a range of machine learning and deep learning algorithms/techniques are suitable for usage in their respective disciplines and domains. Using audio samples supplied by the dataset or the user, it tries to predict the musical genre.

1.1 OVERVIEW

In the current day and age, machine learning is very popular and is now being used very widely in our day-to-day life. There are various different kinds of machine learning and deep learning algorithms/techniques which are suitable to be used in their own field and domain according to the requirements of the application which is demanded according the user. There are primarily four major learning algorithms in machine learning. - Supervised learning where we use fully labelled dataset to create a mathematical model, Unsupervised learning where we use unlabelled dataset without any requirement in our mind, Semi-supervised learning where we use both labelled and unlabelled data and finally Reinforcement learning where the learning is accomplished by completing the task successfully, It is like a reward system. It learns by positive and negative feedback. For this project we use supervised learning along with neural networks to make a classification model that can train itself through multiple layers of neurons. So by using all these concepts we are aiming to create a music suggestion system based of genre classification.

1.2 RESEARCH OBJECTIVES

This project's primary purpose is to develop a machine learning (ML) model capable of suggesting songs depending on the genre of the input audio. We must sift through numerous classification models based on machine learning in order to extract the genre of the music and implement the finest python libraries for song recommendation. We use Convolutional Neural Networks, Recurrent Neural Networks and Fuzzy K-Nearest Neighbour network to classify the songs for our objective problem.

1.3 PROBLEM STATEMENT

This project is a Python programme that can classify the genre of user-provided music and can recommend songs based on that classification. This application will use Deep Learning algorithms to identify each song based on the information retrieved from the music, accurately displaying the genre and classifying songs further into sub-genres for better results. We are also comparing the different deep learning methods and observing which is the best one for the recommendation of the songs. We will use the different features and methods for extracting the datas and testing and training them after which we will be able to give the result of the song ,i.e, the genre and sub genre of songs.

1.4 INNOVATION IDEA

Several variables influence the decision, but most have to do with how the song “sounds.” For example, a pop song might feel faster and “funkier” than, say, a romantic song; based on specific parameters that define these features, music streaming services cluster similar sounding or feeling songs into the same genre. Thus, classifying music based on genres can help suggest the next songs to a listener, curate playlists of new recommendations, or filter undesirable content. Classifying music based on their genres can also help maintain libraries of music and keep them very organized, which can help with the sales of the music albums and individual tracks.

1.5 PURPOSE OF THE PROJECT

This project's primary objective is to construct Neural Networks to extract features from diverse audio samples and train on them to reliably classify any music into distinct genres and recommend songs based on it. Additionally, use the most effective model for extracting the features of the music samples from the datasets. To train the chosen model with multiple samples so that it can have a clear classification notion to separate the music into its various genres. And finally, to employ Deep Learning techniques and learn more about them through the creation of this project.

1.6 ARTIFICIAL INTELLIGENCE – A BRIEF INTRODUCTION

The concept that enables computer systems to perform and give out results that normally require intelligent traits found in human beings like problem-solving, making decisions or understanding a real-time language is called Artificial Intelligence. Large sum of data, commonly referred to as datasets can be used in AI to train the computer systems to form a system that mimics the human brain and recognise patterns and make certain predictions or even learn a human language.

There are various kinds of Artificial Intelligence:

- Rule-based AI: This type of AI have computer system to obey a set of rules that is programmed before-hand for the program to function, and any result that is obtained is by working under the constraints of the mentioned rules. It is often implemented in systems that are primarily focused on providing suggestions or recommending options to the user based on specific rules and regulations it must follow.
- Machine Learning: This is a type of program that is developed by implementing various algorithms that is uses to learn trends from datasets and improve it's performance as time goes on. It is commonly deployed in case of image recognition, recognition of speech etc.
- Deep Learning: This type of AI is in fact a subset of Machine Learning. This specific type of AI is developed by construction of neural networks that mimic the neurons of the human brain to a close degree. It learns using the neural networks and the way it learns can be tailored to the need of the objective of the program. Deep Learning is more effective in handling large datasets in order to perform tasks like recognition of speech or images.
- Natural Language Processing (NLP): This particular kind of AI is solely focused on the interpretation of human language. It uses various algorithms to perform tasks like parsing, tokenization etc. to interpret and learn the human language. It is also used to make predictions based on language input. This is mainly used in applications like virtual assistant, live chat bots etc.

1.7 MACHINE AND DEEP LEARINING – AN OVERVIEW

Machine Learning and Deep Learning are both a part of the super set which is Artificial Intelligence. ML and DL both have similarities and differences. Similarities include:

- In both ML and DL the models are first trained and then the program learns from the training to make predictions based on the input.
- In both the cases the data has to undergo a step called data pre-processing, which refers to conversion of data into a form that is standard and comprehensible for the program extracting features from the data that can be used for training and prediction.
- The usages of variety of algorithms are another aspect where the programs look similar. Both systems make use of complex algorithms to train their respective models and improve their performance with time.
- Both machine learning and deep learning improve in efficiency as the training data size gets increased. The programs will have more patterns to recognise and in turn get a better hold on the learning and prediction skills.

Overall, the similarities are very evident and it can be seen how Deep Learning is considered as a subset of Machine Learning. But the differences between them are what make either of them a more suitable option in certain scenarios. The main differences are:

- The big difference comes in the algorithm usage. While both these programs use algorithms to function, the way they are implemented is what sets them apart. Machine Learning algorithms are structured in a simple manner. They use relatively quite simple algorithms and formulas to analyse the data and perform tasks. Meanwhile, the algorithms used for deep learning tasks are complex in nature. They are often neural networks designed in a complex manner to analyse the data and identify features to perform tasks.
- The quantity of data required differs depending on the program. Deep learning programs require a lot of data to analyse as the neural networks that are used are typically very complex and have multiple layers that often perform same tasks repeatedly on the program by reducing the data being analysed each time. Therefore compared to the machine learning problems, deep learning problems require vast amount of data for functioning.
- Feature extraction is a step where the program makes use of various properties or features of the data given to learn patterns or trends that can be used to train the

model and predict the output when a data is entered. In Machine Learning, the feature extraction process is done manually. There is no facility where the features can be automatically identified by the program and can be used to train models. Meanwhile, deep learning modules have the option of automatically extracting the features it can use for recognizing patterns and train models.

- Machine learning algorithms do not require a lot of time to perform their tasks as the data it analyses is smaller in comparison to deep learning and the algorithms are also less complex in comparison to deep learning. Whereas in deep learning, the neural networks more often than not have multiple layers to analyse the data. And the data being analysed itself is of vast quantity. Therefore the training time is huge compared to the machine learning programs.
- Deep learning models are known to have better performance and accuracy in terms of functioning when compared to machine learning models for complex tasks such as recognition of images and speech.

Overall, machine learning is favoured in situations where the parameters of the task are small and the data being used is fairly simpler, while deep learning is preferred in complex tasks where proper feature extraction is needed and the data required is huge in comparison. Which is exactly why our program makes use of deep learning algorithm as the multi-layered neural networks assist in better performance and accuracy.

There are various types of learning techniques used by machine learning and deep learning that play a major role in selecting the right algorithm for the project. There are three main ways, a program learns:

- Supervised Learning: In this type of learning, the result of the input is known before-hand. The way deep learning algorithms learn in this case is my reducing the contrast between the known output and the output it is predicting to a very small degree. Supervised learning is very common learning method and is widely used in various classification problem. Supervised learning algorithms include decision trees, support vector machines, neural networks, and linear regression, among others. These algorithms can be applied to a wide range of problems, including image recognition, speech recognition, natural language processing, and recommendation systems.

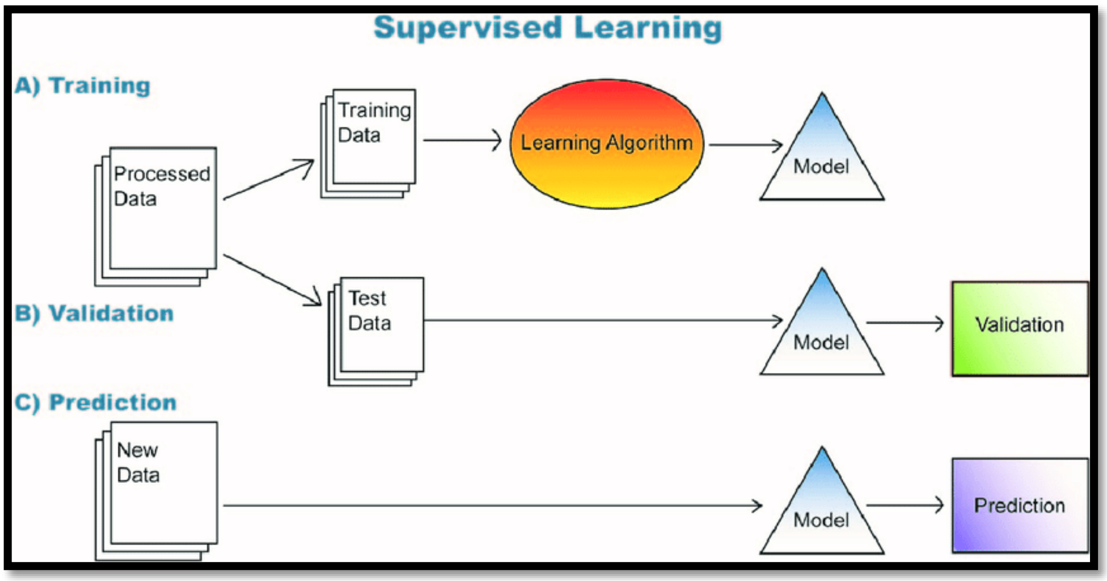


Figure 1.1: A representation of Supervised Learning

- Unsupervised Learning: In the case of unsupervised learning the outputs of the data are not labelled/known to the program or the user. The way deep learning algorithms learn is by reducing the dimensions of the data or by clustering the data. The most common uses of unsupervised learning is for programs where suggestions are given based on the trends of the data.

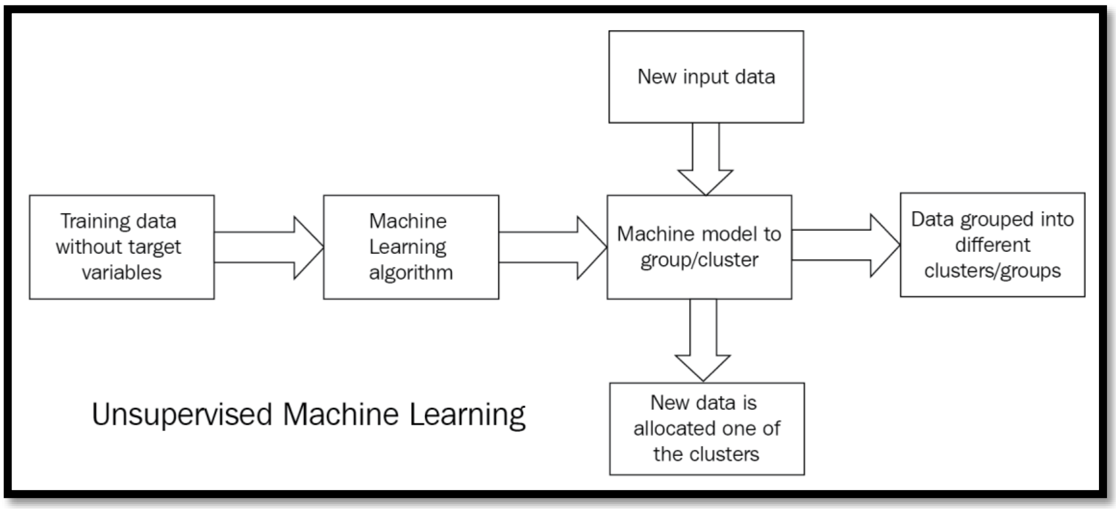

New input data
Training data without target variables
Machine Learning algorithm
Machine model to group/cluster
Data grouped into different clusters/groups
Unsupervised Machine Learning
New data is allocated one of the clusters

Figure 1.2: A representation of Unsupervised Learning

- Reinforced Learning: In the case of reinforced learning, the whole system works on the basis of feedback method. The system is highly reactive towards the feedback or acknowledgment it gets from the environment. The algorithm receives responses or feedbacks for every task it performs, which can be perceived as a reward or an alert. Based on the nature of the feedback, the system learns to repeat or discontinue certain actions. This method is commonly used in cases where autonomous machinery are used. It is also proven effective for playing games and other robotic uses.

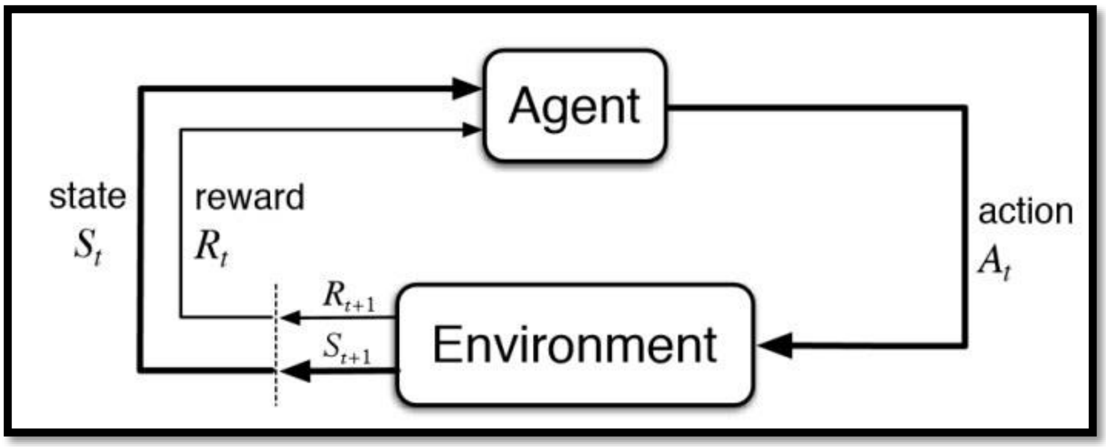


Figure 1.3: A representation of Reinforced Learning

It should also be noted that there are also approaches where the learning methods are fused together to form hybrid versions of learning, for example semi-supervised learning. For our project and considering it’s aim, we move forward with the supervised learning to achieve our goal as our main objective is classification based.

1.8 FLOW OF CHAPTERS

- CHAPTER 1: It will consist of the Overview of the project, in which we will see what we have done in this project and what is our aim/goal and what we expect from this project to do when it is completed. It takes a look at the subject matter of the project where we will define our problem statement in which we will describe about our ai which we want to achive and the solution which we thinks is the best. The idea of innovation of the project. What factors influenced us to pursue this project and the motivation behind it all and how it can be useful for us.
- CHAPTER 2: It consists of the literature review where we have studied the different works of different authors and their approach for solving this problem from which we inferred something which we can use for us but in a better way and their disadvantages which we want to overcome. This chapter covers the previous works that were referenced to implement this project and their objectives and limitations. It also sheds light on what our project gained from it and what it will do to improve upon it.
- CHAPTER 3: This chapter consists of the proposed methodology that we are going to be implementing in the project for solving the problem statement which we have given for our task. It is about the look into the concepts and algorithms that we will be implementing and why we would be using the mentioned concepts and algorithm for the project which are descried in a detailed and proper manner.
- CHAPTER 4: This chapter contains the entire programs implementation which shows the functioning of our code that how different libraries are used and different algorithms are used for solving our problem statement which is given by us. The screenshots of the code and important sections of the program are provided with explanation for all the modules for the better understanding of the person who wants to explore and know about our project, we have also added comments with our code so that anyone can esily understand it and some important libraries are also explained where we describes their main features and their functioning.
- CHAPTER 5: This is the chapter were the results and discussions lie which we have got by running our program and observing the different results obtained after the execution of our code, our program is trained and after that we gets the result after running the code. The screenshots of all the results of the project is provided with

explanation for all the results in such a way that anyone can see and understand the results in a proper and detailed way.

- CHAPTER 6: This is the final chapter of the this project. This is the conclusion and future works chapter where we have given a brief summary of what we have achieved from this project and what can we do to preserve this work for the future which can be helpful for others means the continuous update we can make to keep it up to date so that it can be useful in future also not like other works where they don’t think about the future perspective and only concentrates on the present scenariao which is not a good way of doing project work. We have discussed about different observations which we have got and the advantages which we feel we have in comparison to the other previous works and the methods by using which we can further improve it and keep up with the present world so that it can be useful.

1.9 MOTIVATION

Ever since the era of music streaming services, we have had the benefits of enjoying the enormous selection of music the globe has to offer– and as a requirement, thanks to these services insightful suggestions on the type of music that consumers adore are in demand. Spotify, YouTube, and other comparable platforms analyse our music listening habits and attempt to serve us similar stuff. To manually classify a song into a specific genre, one would need to hear the entire song or a substantial section of it which is time consuming and laborious which is not efficient for us to do. With Deep Learning algorithms, the application can accurately classify music into respective genres in a fraction of the time and effort normally required which will greatly reduce the human effort and will make it more eeficient.

As the music industry continues to evolve and continues to produce more different types and styles of songs by each passing day, the demand for more advanced and accurate classification system will undoubtedly increase. The use of deep learning algorithms will play a critical role in meeting this demand by providing highly personalized and accurate music recommendations to consumers. The use of deep learning algorithms will play critical role in meeting this demand by providing highly personalized and accurate music recommendations to consumers.

In conclusion, the application of deep learning algorithms in the classification of music into genres and the sub-genres {as we know that now a days there are no pure genre music but the mixed music with different sub-genres} is a significant breakthrough in the music industry. It has the potential to revolutionize the way we interact with music, providing highly personalized recommendations of songs to the users and saving time and effort in the process of finding suitable playlist as here user can select anyone song after which he will get the suggestions of the songs according to the genres he has selected for listening. The future of the music industry looks bright, with deep learning algorithms at the forefront of innovation and advancement.

CHAPTER 2

LITERATURE REVIEW

This chapter covers the details of previous works that are either loosely based on the domain of our project or are a predecessor version of our project. Here we see the various works done by the researchers that lack in features that our project provides. The contrast between their works and our work is highlighted and added as reference.

Music Recommender System based on genre using Convolutional Recurrent Neural Networks is the work done by Adiyansjah, Alexander A S Gunawan, Derwin Suhartono. The purpose of this study is to develop a system for recommending music based on the similarity of audio signal characteristics. The research methodology is based on comparing the similarity of audio signal characteristics. Convolutional recurrent neural networks (CRNNs) are utilised for feature extraction and similarity distance to examine feature similarity. Using Mel-spectrograms for audio representation and CRNNs for feature extraction, these outcomes are followed. Finally, recommendations will be generated by comparing the features' similarities. Since this research paper revealed that CNN and CRNN perform better for different genres, we chose CNN as an option for our collaboration with MFCC on this project. [1]

A Novel Music Genre Classification using Convolutional Neural Network – 2021 is the work done by Lakshman Kumar Puppala, Siva Sankar Reddy Muvva, Sudarshan Reddy Chinige. The purpose of this project is to develop a novel method for music genre classification using the GTZAN dataset. Initially, the GTZAN dataset is used for training and evaluating the machine learning model. Features like MFCC are extracted from the dataset and fed into the model and are trained for the best accuracy. We solve the problem of over fitting in this project and also add a suggestion system on top of this project, to extend the features of this project. [2]

Music Genre Classification using Machine Learning Algorithms: A comparison – 2019 is the work done by Snigdha Chillara, Kavitha A S, Shwetha Neginhal, Shreya Haldia, Vidyullatha K S. Objective is to develop a machine learning model that classifies music into its respective genre, compare the accuracy of this model to that of existing models, and draw the appropriate conclusions. A CNN is a feed-forward network, which means that input examples are fed into the network and transformed into an output; with supervised learning, the output would be a label or even a name applied to the input. In other words, they map raw data to categories by identifying patterns that indicate, for instance, that an input image should be labelled "folk" or "experimental." We studied the research done in this paper and selected and implemented the best algorithm for the project and explored how much accuracy it can potentially give. [3]

Advancements in Image Classification using Convolutional Neural Network – 2018 is the work done by Farhana Sultana, Abu Sufian, Paramartha Dutta. Objective is to give a review of the advancements of the CNN in the area of image classification. Multiple models of CNN were considered to conduct training and testing. These models were trained on a single Dataset. The latest advancements in CNN were implemented in this project. The capabilities of CNN are tested thoroughly and is made one of the foundations of this project. [4]

Neural Network Music Genre Classification is the work done by N. Pelchat and C. M. Gelowitz. The goal is to use images of spectrograms generated by time slices of songs as input for a neural network in order to classify the songs according to their respective musical genre. To slice up the spectrograms obtained by the songs into a 128 x 128 spectrogram and feed it to a neural network where 70% will be training data, 20% will be validation data and rest 10% will be testing data. Then using the NN, classification is done on the spectrograms into various musical genres. We implemented a recommendation system on top of this project. As well as solved over fitting problem of the project. [5]

Music Genre Classification Using Duplicated Convolutional Layers in Neural Networks is the work done by Hansi Yang and Wei-Qiang Zhang. In this work they uses a uses duplicated convolutional layers to capture both local and global features in the audio signals. The authors explore the use of various audio features, such as Mel-frequency cepstral coefficients (MFCCs) and spectral contrast, and compare the performance of their proposed architecture with other state-of-the-art approaches. The author has also used the GTZAN dataset. Which contains audio clips from ten different genres. It has the accuracy rate of 96.20%. [6]

CHAPTER 3

PROPOSED METHODOLOGY

This chapter covers the methodology that we propose for this project. This methodology is based on the research done by us in the literature review. Taking into consideration all the previous works done, we propose a workflow that the modules follow to work in harmony and produce efficient results.

3.1 OVERVIEW

The methodology used will aim to make the most accurate and insightful predictions using Supervised machine learning techniques. Supervised machine learning uses labelled data that will be further pre-processed and fed to the model for training and testing purposes.

Our literature review shows that GTZAN and Million Song Database (MSD) are the two most widely used audio datasets. This project will utilise the GTZAN dataset, a collection of 10 genres containing 100 30-second audio files each. We will be using Convolutional Neural Network for identifying the genre of the input (song).

Further we can use libraries like pywhatkit, pyautogui, OS etc. to further take the output we get from the CNN (genre) and feed it to the python library and suggest songs related to that genre on various platforms.

3.2 CONVOLUTIONAL NEURAL NETWORK

A Convolutional Neural Network is one of the many marvels of the Deep Learning field that has the ability to assign importance to various distinguishing features of an image and distinguish it from other images. A CNN requires less pre-processing than other techniques. CNN's filters can be learned and remembered automatically, whereas in other methods they must be manually created.

It is inspired by the organization and functionality of the visual cortex in the brain, where neurons are arranged in layers that analyze small local regions of the visual field. CNNs consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers. In the convolutional layers, a set of filters is applied to the input image to extract local features, such as edges and textures. The pooling layers then downsample the feature maps to reduce the dimensionality of the data. The fully connected layers then use the extracted features to classify the input image into one of several categories.

CNNs have been used in various applications, such as object recognition, face recognition, and image classification. They have also been adapted for other types of data, such as text and speech, through techniques such as 1D and 2D convolutions.

Convolution: It is known as the process of applying specific multidimensional weights or kernels on an input image. In the case of music genre images, spectrograms are the input images and kernels will be used and the kernels and spectrogram will undergo dot product. Kernels can be used to detect features in images. In this case the kernels will detect MFCC features in different samples.

In a CNN, convolutional layers apply a set of learnable filters to the input image or feature map, which extract local features from the input. The filters are typically small in size and slide across the input image or feature map, producing a feature map that captures the presence and location of specific features in the input.

The convolution operation is defined as the element-wise multiplication of the filter and a small region of the input, followed by a summation of the resulting values. This process is repeated for all regions of the input that are covered by the filter, producing a feature map that summarizes the presence and location of the feature represented by the filter

The parameters of the filters are learned during the training of the network, using backpropagation to adjust the weights of the filters to minimize the error in the network's output. Convolution is a powerful operation for extracting local features from images and other types of data.

Pooling: This is another common process in the CNN procedure that shrinks the image and accumulates the features. Pooling layers scale down the map's dimensions. Overall it scales down the parameters for the programme model to learn, as well as the amount of computation performed and the time required to execute such computations in the network. The pooling layer provides a summary of the features that exist in a particular area of the convolution layer's output feature map.

The pooling layer operates on each feature map separately and replaces a rectangular neighborhood of the input with a single value. The most common types of pooling are Max Pooling and Average Pooling.

In Max Pooling, the maximum value in each neighborhood is selected as the output value. This operation helps to preserve the most important features in the input feature map. In Average Pooling, the average of the values in each neighborhood is taken as the output value. This operation helps to smooth the feature map and reduce the effect of noise

The size of the pooling window and the stride used to move the window also affect the output dimensionality of the pooling layer. A smaller window size and a larger stride lead to a more aggressive reduction in spatial dimensions, while a larger window size and a smaller stride lead to a more conservative reduction.

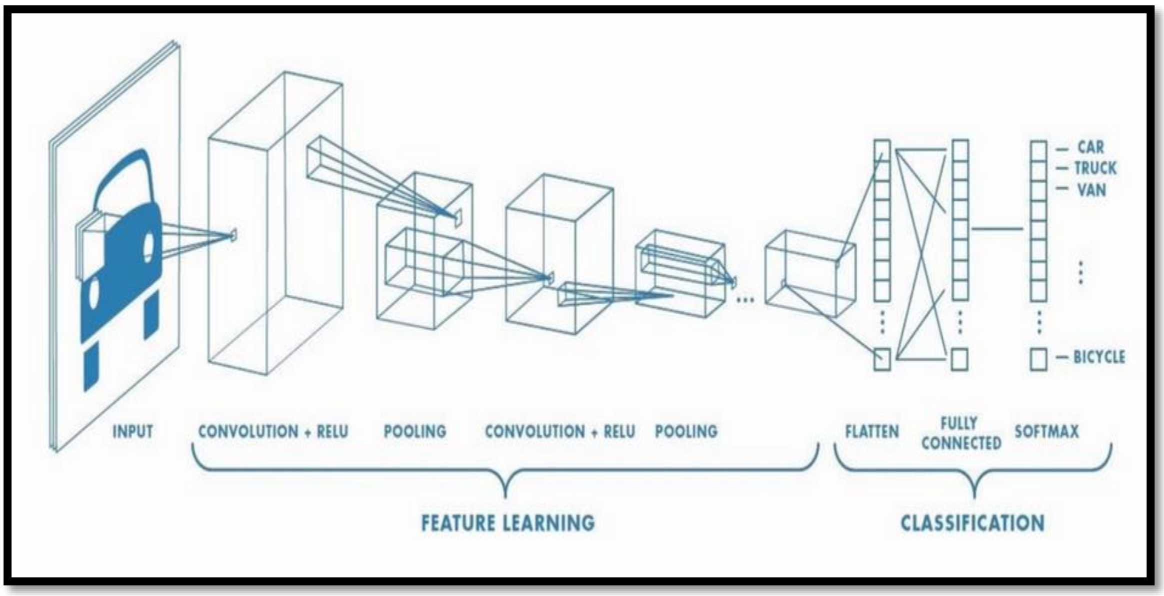


Figure 3.1: Convolutional Neural Networks architecture

In the figure 3.1 the convolutional operation is performed by executing a dot product between the feature map and the kernel.

The formula for the resultant output from the convolutional layer is:

$$(n \times n) \, . \, (f \times f) = (n - f + 1) \times (n - f + 1) \tag{3.1}$$

Where input image has the size of n x n. The kernel size is f x f and the output will be $(n - f + 1) \times (n - f + 1)$. (3.2)

The soft max formula is as follows:

$$\sigma(\overleftarrow{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.3}$$

Where -

- $\overleftarrow{z}$ is the vector given as input to the softmax function.
- z_i is the are the input vector elements. They can acquire any real number like a positive, 0 else even a negative value.
- e^{z_i} is the standard exponential function that is applied on each and every element in the input vector.
- $\sum_{j=1}^{K} e^{z_j}$ is the normalization part of the formula. It makes sure that all values will be in the range of (0.1) and all output values will give a sum on 1.
- K is the number of classes present in the multiclass classifier.

3.2.1 REASONS TO USE CNN

- Music is a time-varying signal with frequency and temporal variations. CNNs are great at detecting changes and patterns in the temporal and frequency domains, which makes them ideal for evaluating audio inputs.
- CNNs are capable of extracting high-level properties from raw audio signals and using them to represent audio data in a more understandable and usable way. When compared to traditional feature extraction methods, this can result in improved classification results.
- CNNs have the ability to learn hierarchical representations of their input data. In the case of music categorization, this means that the network may learn to distinguish low-level components such as individual notes, and then use those attributes to recognise higher-level elements such as chords and melodic lines.
- CNNs may be trained on large datasets of audio signals, such as music databases or audio recordings, and then used to novel classification problems. This can reduce the

need for large amounts of labelled data when starting a new project while simultaneously enhancing classification accuracy and speed.

- Acc. to the literature survey done, it was found out from the research papers that it has a high accuracy and is appropriate for using in Music Genre Classification.

3.2.2 DISADVANTAGES

- CNNs require a large amount of data to perform well. If you just have a little amount of labelled data for your music classification assignment, a CNN may not be the best choice since it will overfit to the training data and will not generalise well to new samples.
- CNNs are frequently referred to as "black boxes" since it can be difficult to understand how the network predicts. If interpretability is important for your music classification effort, you should choose a different approach.
- CNNs are complex models that may need a significant amount of computing power to train and assess. If you have limited computer resources or need a model that can work in real-time, a CNN may not be the best option.
- While CNNs have shown to be useful in many music classification tasks, they are not always the best choice. Other types of models, for example, may be more useful if your job requires you to recognise certain instruments or musical styles.
- Extensive training data is necessary.
- Overall, while CNNs are a strong tool for music categorization, depending on the exact needs of your work, they may not always be the ideal solution. Before settling on a solution, it is critical to thoroughly analyse the trade-offs and limits of various models and methods.

3.3 RECURRENT NEURAL NETWORKS- GRU:

- Recurrent Neural Network is a generalised feed-forward neural network with an internal memory. RNN performs the same function for each data input, but the output of the current input depends on the output computed in the past; hence the name "Recurrent." The output is copied and transferred back into the recurrent network following its computation. For decision-making, it considers both the current input and the output calculated from the previously fed input.Gated Recurrent Units or GRU is a variant of GRU that, unlike Long Short Term Memory (LSTM), has only 3 gates within it. The GRU architecture has been shown to perform well in applications such as natural language processing, speech recognition, and video analysis, where it can effectively model long-term dependencies in the data. Compared to LSTMs, GRUs have fewer parameters, making them faster to train and more memory-efficient, while still achieving similar or better performance in some tasks.

- There is an Update gate (z) that determines how much of the previously acquired information about the computed outputs is needs to be passed along into the future.

- There is a Reset Gate (r) that determines how much of the previously acquired information about the computed output is not necessary to remember i.e. could be forgotten.

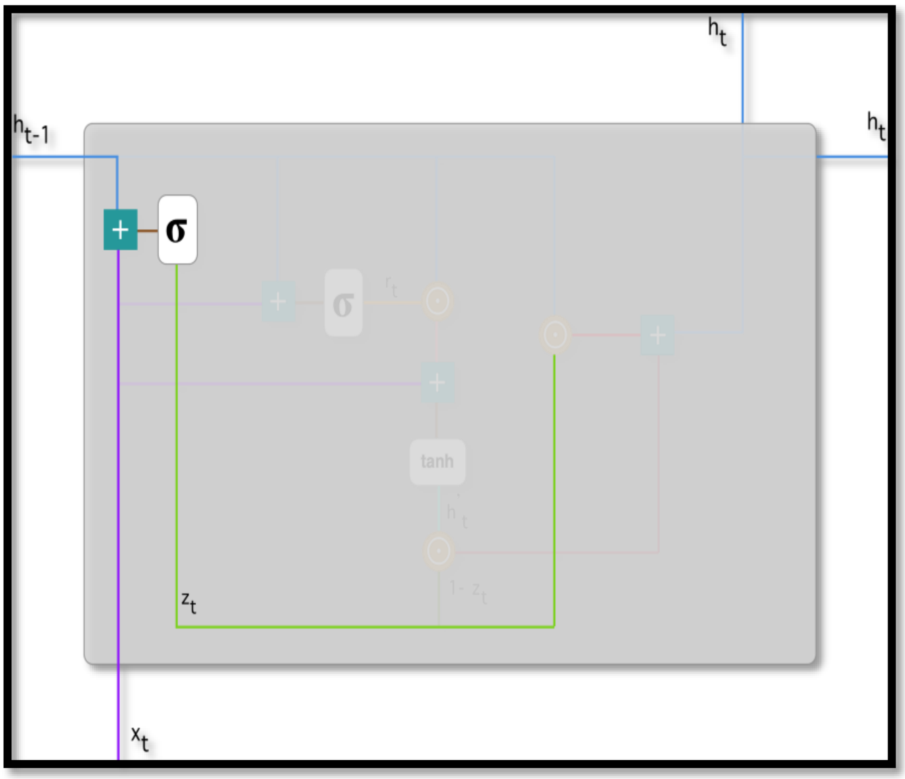


Figure 3.2: Update Gate architecture

- From figure 3.2 the computational formula for the update gate for a time step t is described below :
    - $z_t = \sigma(W_z x_t + U_z h_{t-1})$ (3.4)
    - Where the sigma function represents the sigmoid activation it uses.
    - W_z is the weights for the current unit.
    - x_t is the output that was previously found out, that is considered as input during the current time step.
    - U_z is the weights of the previous units being taken as input.
    - h_{t-1} is the hidden state which holds the information about the previous units.
    - The formula basically means that the update gates multiply the weight of the current unit with the current input and adds multiplication of the previous inputs and the previous weights and the entire result is multiplied with the sigmoid function to give the output as the update input for the next unit in the GRU, z_t.

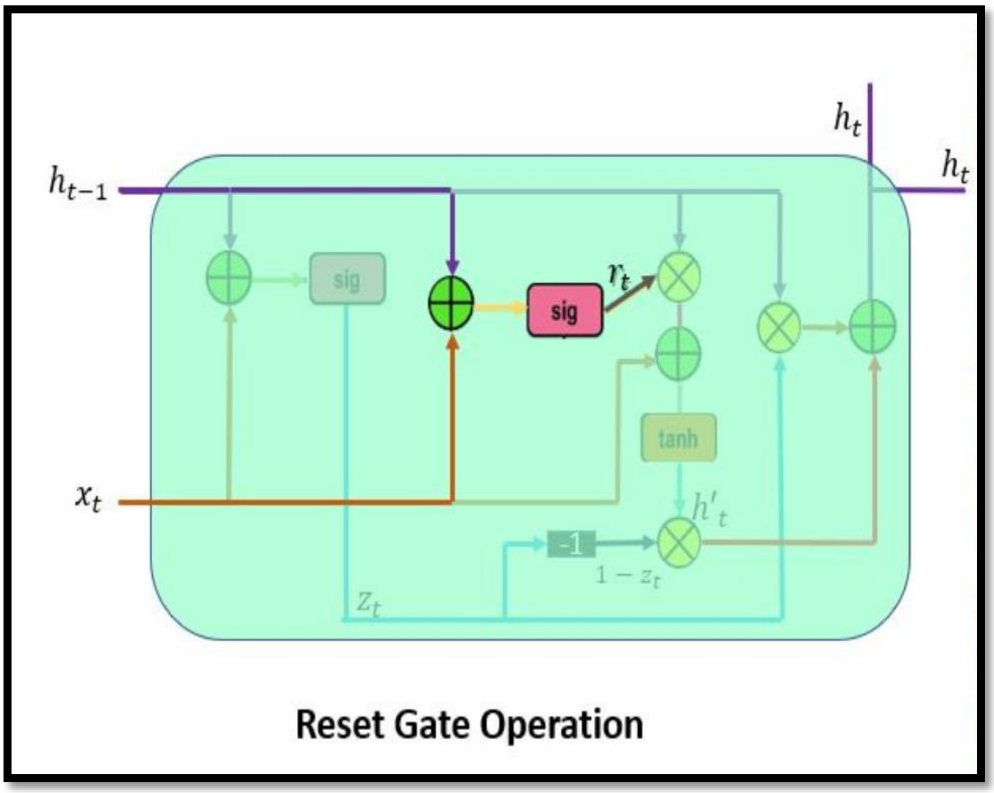


Figure 3.3: Reset Gate architecture

- From the figure 3.3 the formula for the Reset gate in a time step t is given in :
    - $r_t = \sigma(W_r x_t + U_r h_{t-1})$ (3.5)
    - Where the sigma function represents the sigmoid activation it uses.
    - W_r is the weights for the current unit.
    - x_t is the output that was previously found out, that is considered as input during the current time step.
    - U_r is the weights of the previous units being taken as input.
    - h_{t-1} is the hidden state which holds the information about the previous units.
    - The formula basically means that the update gates multiply the weight of the current unit with the current input
    - And adds multiplication of the previous inputs and the previous weights
    - And the entire result is multiplied with the sigmoid function to give the output as the update input for the next unit in the GRU, r_t.

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

(3.6)

Figure 3.4: Formula for Current memory that will use the reset gate to store relevant information.

- As shown in figure 3.4 the information is updated using Hadamard (element-wise) product between r_t and Uh_{t-1} and added with Wx_t and a tan function is used on top of it. That determines what to remove from the previous time steps.
- We split the data into training data, testing data and validation data. 55% will be training data, 25% will be testing data and the rest 20% will be out validation data. Accuracy of the model will be evaluated as well. To avoid overfitting, Dropout will be used from Keras.

3.4 FUZZY K-NEAREST NEIGHBOUR :

- The K Nearest Neighbour (KNN) algorithm is a classification and regression machine learning technique.
- It is a non-parametric method, which means it makes no assumptions about the data's underlying distribution.
- Each example in the dataset is represented as a point in n-dimensional space using the KNN method, where n is the number of features in the dataset.
- To forecast a new example, the algorithm finds the K closest cases in the training set based on Euclidean distance in the n-dimensional space, and then uses the class labels of those examples to predict the new example.
- By allowing for a more flexible and nuanced method to computing distances between data points, fuzzy logic may be utilised to improve the efficiency of the K Nearest Neighbour (KNN) algorithm.
- The distance between two data points in classical KNN is determined using a deterministic distance metric, such as Euclidean distance.
- However, in many real-world scenarios, the relationship between data points is ambiguous and probabilistic rather than completely binary or deterministic.
- The K Nearest Neighbour program is divided into functions for each operation it is required to do. The K value is taken as 5 by default and can be changed later. The extracted MFCC features is used to calculate the distance of the music.
- We may assign a fuzzy membership value to each example in the training set by introducing fuzzy logic into the KNN algorithm, expressing the degree of similarity between the example and the class it belongs to.
- We may use these fuzzy membership values to weight the contribution of each example in the K nearest neighbours to the forecast when creating predictions for new instances.
- The input’s distance from each of the music from the training set is calculated and stored as a list and passed onto the next phase that is assigning the degree of association to the input to all the 15 genres using the distances.
- This weighted technique enables more advanced and context-dependent categorization, which is especially useful when there is overlap or ambiguity between groups.

- Fuzzy KNN can tolerate uncertainty and noise in data while also better capturing the local structure of the data, making it particularly effective in pattern recognition and image processing applications.
- The Fuzzy algorithm is implemented to provide association to the genres and those results are stored in another list, which is used to print the genre with the highest degree and the runner-ups, to inform the user about the genre and sub-genres of the song.

$$\mu_i(P) = \frac{\sum_{j=1}^{k} \mu_{ij}\left(\frac{1}{d\left(P_i, X_j\right)^{\frac{2}{m-1}}}\right)}{\sum_{j=1}^{k} \left(\frac{1}{d\left(P_i, X_j\right)^{\frac{2}{m-1}}}\right)} \tag{3.7}$$

Figure 3.5: The Formula used for Fuzzy KNN implementation

- Using the formula shown in Figure 3.3, we assign the membership for every class, in this case, genres of songs.

CHAPTER 4

MODULES AND IMPLEMENTATION

The module is a collection of source files and build settings that let you divide the project into discrete units of functionality. One module can use another module as a dependency. Here, we have used six modules obtain the result of the project. these modules include genre names extraction, feature and label extraction, training CNN model using 3 convolutional layers and max pooling layers, implementing RNN model for training.

4.1 ARCHITECTURE DIAGRAM – CNN & CRNN

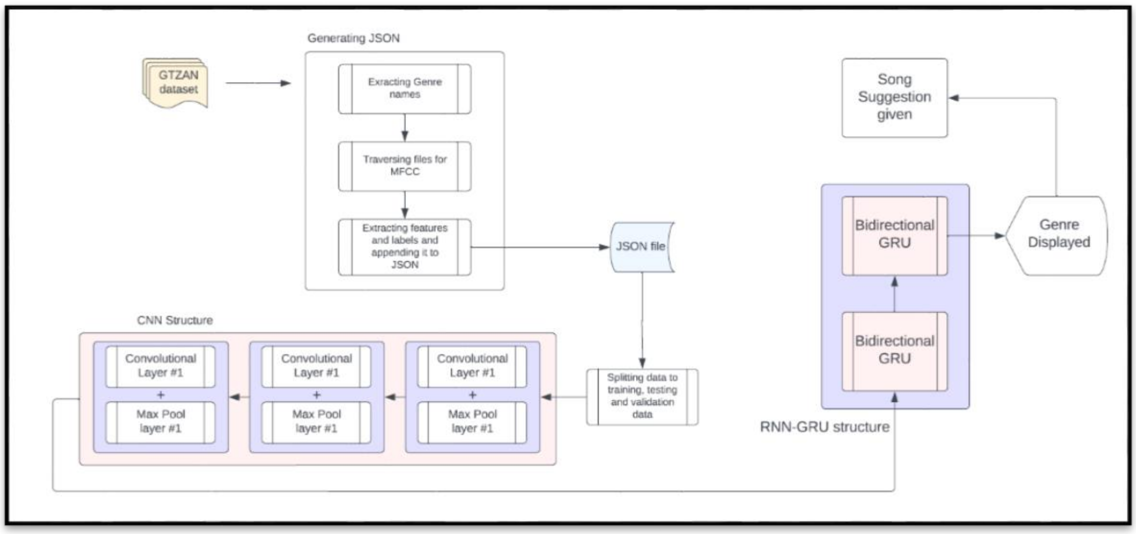


Figure 4.1 Flow of modules

The architecture diagram depicts the flow of modules as shown in figure 4.1, the modules covered as a part of the major project are genre names extraction, feature and label extraction, training CNN model using 3 convolutional layers and max pooling layers, implementing RNN model for training and Song Suggestion. This flow describes how we are going to build the final module right from scratch. The various modules covers respective parts in building the model in a step by step manner. Hence this architecture diagram is necessary to understand the flow of modules and their respective share in building up the complete model.

4.2 ARCHITECTURE DIAGRAM – Fuzzy KNN

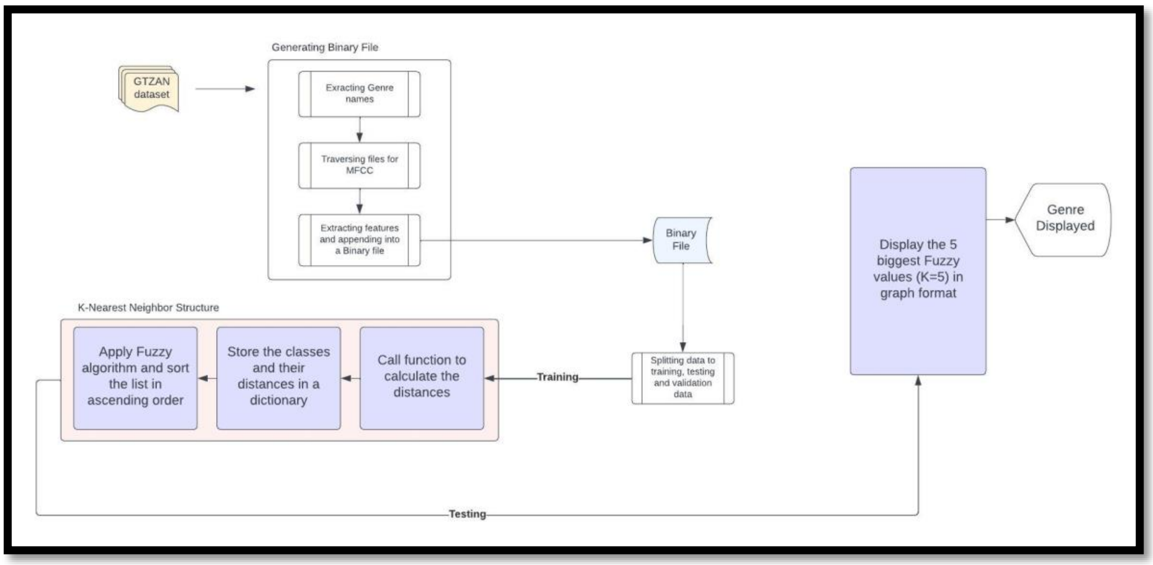

Generating Binary File
GTZAN dataset
Exracting Genre names
Traversing files for MFCC
Extracting features and appending into a Binary file
Binary File
Display the 5 biggest Fuzzy values (K=5) in graph format
Genre Displayed
K-Nearest Neighbor Structure
Apply Fuzzy algorithm and sort the list in ascending order
Store the classes and their distances in a dictionary
Call function to calculate the distances
Training
Splitting data to training, testing and validation data
Testing

Figure 4.2 Flow of modules – Fuzzy KNN

The architecture diagram depicts the flow of modules for the Fuzzy KNN algorithm as shown in figure 4.2. This figure highlights the basic structure of the Fuzzy KNN model. The stages covered in this architecture are the Feature extraction from the songs. Then comes the distance calculation section where the programmed function proceeds to calculate the input distance from the dataset. Then comes the Fuzzy membership association function that assigns a value for each distance. Using ‘5’ as the value of K, the top 5 values are taken and prediction is done.

4.3 USE CASE DIAGRAM

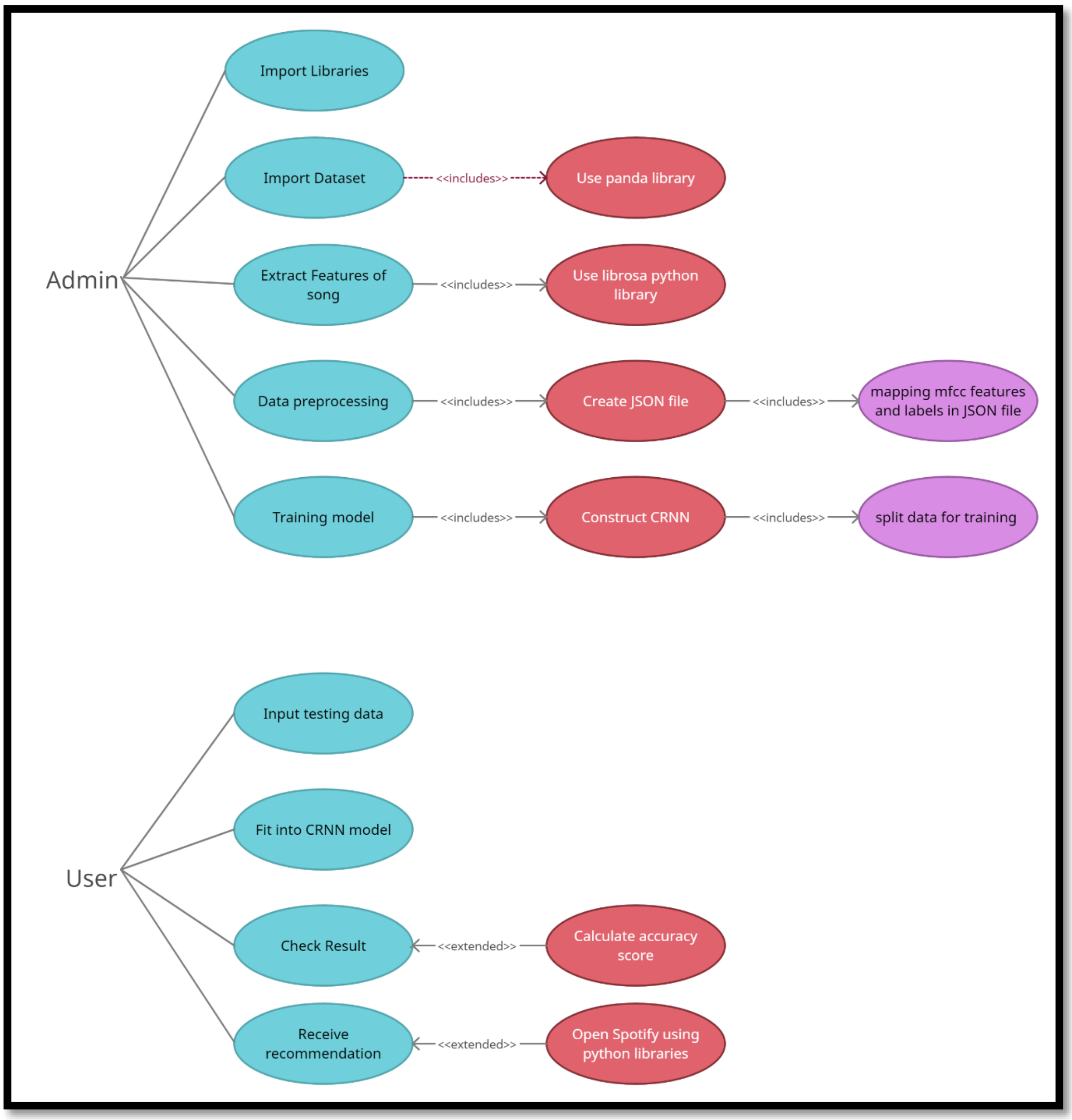


Figure 4.3 UML USE CASE Diagram

This is the UML – Use Case Diagram upon which our whole model is based as shown in figure 4.2. This illustrates the relationship between use cases, actors, and systems at a high level so as to build a music suggestion system and how a user is going to interact with the system upon which the desired features will be displayed and accordingly the user will be entered the values to receive suggestions based on the input.

4.4 MODULES UNDERTAKEN – CNN & CRNN:

1. Data Pre-processing:
    a. Data loading is the starting point of the whole program.
    b. Data loading is the process that consists of loading all the libraries into the program.
    c. The libraries used are :
        i. Os – this python library is used for traversing through the various files and folders in the GTZAN dataset. A for loop is used along with os.walk feature to traverse through the files and folders.
        ii. Json – this python library is used to dump all the dictionary data (mfccs and labels) into the JSON file.
        iii. Librosa – librosa is the main python library being used for this project. This is a python library extensively used for audio and sound analysis. Using this library, we load the audio files and extract the MFCC data.
        iv. Math – this is a library used for mathematical operations. This lib is used only for the simple purpose of rounding off the expected number of MFCCS per segment.
    d. Defining Data Dictionary:
    e. This module focuses on constructing the basic structure of the extracted features that it will form after being dumped into the JSON file. The fields of the file are defined here. The fields are as follows:
        i. Mapping: this will consist the genre names.
        ii. MFCC: will contain the MFCC values of each sample recorded.
        iii. Label: it is basically the index of the genre. It ranges from 0-9 as there are 10 genres. So, all samples of 1st genre will have label of 0, and this will be followed till 10th genre.
    f. Appending Mapping data:
    g. From the above module, we know that there is a field called mapping in the dictionary.
    h. This field consists of the names of the genres.
    i. This can be obtained by the directory path of each genre.
    j. Separating the directory path on '/' character, we can separate the name of the genre from the path of the directory.
    k. Then we can append the name on to the JSON file.

l. Extracting MFCC features for each segment:
m. The next step in the process is to slide from segment to segment of each track and the MFCC features of the tracks are extracted and added to the MFCC field of the dictionary.
n. This is done by defining the start and end point of each sample and extracting the MFCC within these points on the track.
o. Checking expected number of MFCCs:
p. The next step is to calculate the expected number of MFCCs in a segment. At times the audio may have more or less vectors than expected. So only when the MFCC data length in each segment is equal to the expected MFCC length, we need to append it to the dictionary. This is done by dividing the number of samples in a segment by the hop length.
q. Dumping data into JSON:
r. This is the final step where the mapping data, labels and the MFCC features are all appended into the JSON and exported to the system. This data will be used as input to the neural network to classify the audio into different musical genres.
s. Constructing the Neural Network Model:
t. Import the python libraries.
u. A function to load the JSON file and take the MFCCs as input is defined.
v. Splitting the dataset into training, testing and validation dataset.
w. Building the KERAS Sequential model:
    i. 3 layers of convolutional neural networks.
    ii. Each layer with (3,3) sized kernel and a ReLU activation function.
    iii. Batch Normalization is done before execution of each layer.
    iv. Then the output is flatten in form, to be passed into the RNN layers to make the whole project a CRNN project (CNN + RNN).
    v. For RNN implementation, we use Bidirectional GRU layers.
    vi. The output is flattened after the CNN and RNN implementation.
    vii. Then it is passed on to a dense layer and a dropout layer for avoiding over fitting.
    viii. Finally the output of all the steps above is passed on to the soft max activation function which will give a probability output showing which genre is more accurate according to the probability.

2. Fitting the CRNN model:
    a. Split the training, testing and validation data as 55%, 20%, 25%.
    b. Compile the model using keras compiler, using the accuracy metrics to show the accuracy as the end result.
    c. Fit the model with the training, testing and validation data with batch size as 32 and having 30 epochs.
    d. After compiling get a summary of the process.
    e. Get the accuracy of training and testing data from the end result after 30 epochs and print the graph.
3. Predicting songs using newly trained model:
    a. Get a random song to feed to the model as input.
    b. Load the song using Librosa to get the 30 second sample from it.
    c. Extract the MFCCs from the 30s clip by looping through the segments.
    d. Add an extra axis to the MFCC shape and use the model.predict() to predict the genre of music.
    e. Create a list of genres as key value pairs with the respective labels and match the label with the output that was given by the model prediction.
    f. If found the same, print the genre.
4. Suggesting new songs based on the output:
    a. Import pywhatkit, os, and time libraries.
    b. Use the pywhatkit library and OS libraries to open Spotify and enter the genre of music presented.
    c. Spotify recommends the songs falling into this genre.
    d. The time library is used to give some delay in execution of each command to not overload the system with too many tasks to perform altogether.

4.5 MODULES UNDERTAKEN – Fuzzy KNN:

1. Feature Extraction:
    a. The dataset is imported into the workstation.
    b. The file path of the dataset is saved as directory variable.
    c. The OS library is used to access the file directory.
    d. A loop is programmed to go through each file in a genre folder and calculate the MFCC features and covariance of the song.
    e. The features are dumped into a binary file and stored locally in the system.
2. Train-test splitting:
    a. The dataset is prepared. Now the next stage is to split the data into training and testing sets.
    b. The train test split is done manually in this program.
    c. A function is programmed to intake 4 variables:
        i. Filename – the exact file path of each input
        ii. Split – this is the percentage of dataset to be used for training.
        iii. Trset – this is the list that will hold the training data.
        iv. Teset – this list will hold the testing data.
    d. This function is called and the necessary parameters are fed into the program.
3. Construction of the required functions:
    a. The Fuzzy KNN program requires the various functions that can be called to perform operations on the training & testing data and then eventually on the input that will be given to the program.
    b. The functions are:
        i. Distance function – takes 3 parameters, instance 1 which is the training data, instance 2 which is the individual testing data and the k value, which is initialized as 5. This function calculates the distance between the input and the training data and returns the value.
        ii. Getneighbors_fuzzy function – takes 3 parameters like the distance function, the training data, the input and the k value. This function calls the distance function and calculates the distances and with the returned distances, it implements the Fuzzy algorithm and assigns

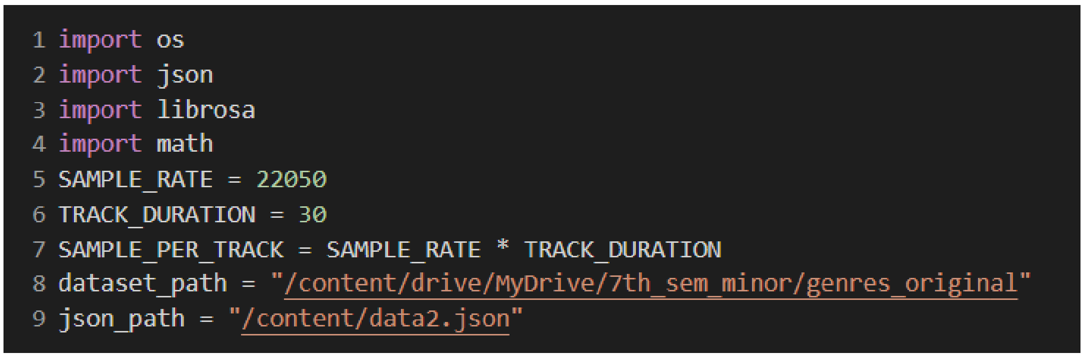
membership value for each class. It chooses the top 5 values and displays the result in a graphical manner.

iii. Accuracy function – this function displays the accuracy of the function by simply keeping tabs on the number of genres that have been correctly predicted.

4. Prediction:
   a. The module deals with the prediction of songs.
   b. The Features are extracted for individual input and the functions mentioned above are called and the parameters are passed.
   c. The output are displayed in graph format.

4.6 IMPLEMENTATION – CNN & CRNN:

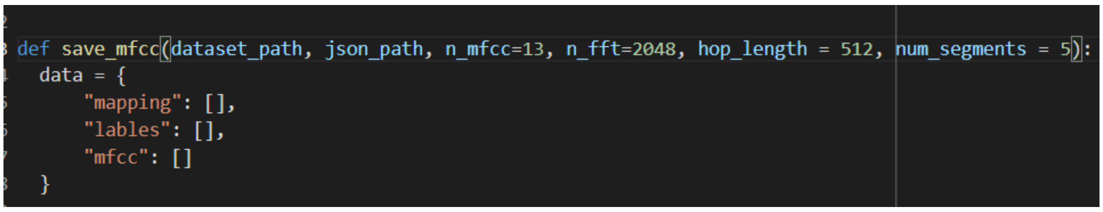
4.6.1 CREATING THE JSON FILE:



```python
import os
import json
import librosa
import math
SAMPLE_RATE = 22050
TRACK_DURATION = 30
SAMPLE_PER_TRACK = SAMPLE_RATE * TRACK_DURATION
dataset_path = "/content/drive/MyDrive/7th_sem_minor/genres_original"
json_path = "/content/data2.json"
```

Figure 4.4: Including necessary libraries

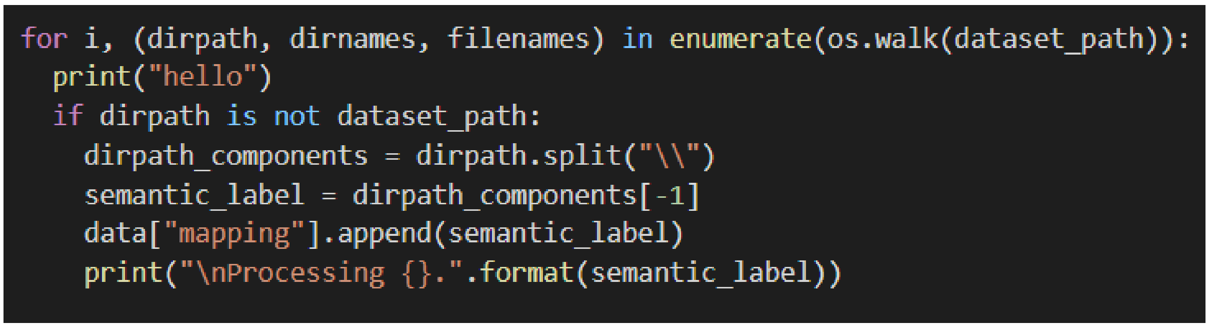
- As shown in figure 4.3 all python libraries are imported.
- Os is used for traversing the dataset directory.
- Json is for performing json operations.
- Librosa is for audio analysis like MFCC extraction.
- Math is for performing ceil operation.
- We also specify the constants and pre-defined variables beforehand.



```
def save_mfcc(dataset_path, json_path, n_mfcc=13, n_fft=2048, hop_length = 512, num_segments = 5):
    data = {
        "mapping": [],
        "lables": [],
        "mfcc": []
    }
```

Figure 4.5: Making the dictionary

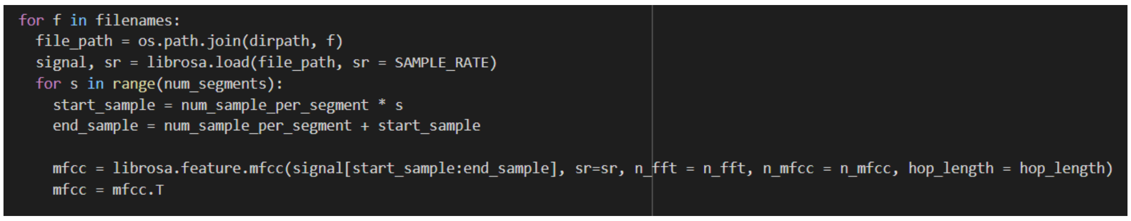
- As shown in figure 4.4 save_mfcc function is created to preprocess the MFCC data and store it into the file.
- A dictionary is created with three fields: mapping data, labels data and MFCC data.
- Mapping data will store the names of the genre.
- Labels will hold the index of the genre.
- MFCC will hold the preprocessed mfcc data of each track.



```
for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):
  print("hello")
  if dirpath is not dataset_path:
    dirpath_components = dirpath.split("\\")
    semantic_label = dirpath_components[-1]
    data["mapping"].append(semantic_label)
    print("\nProcessing {}.".format(semantic_label))
```

Figure 4.6: Mapping the genres

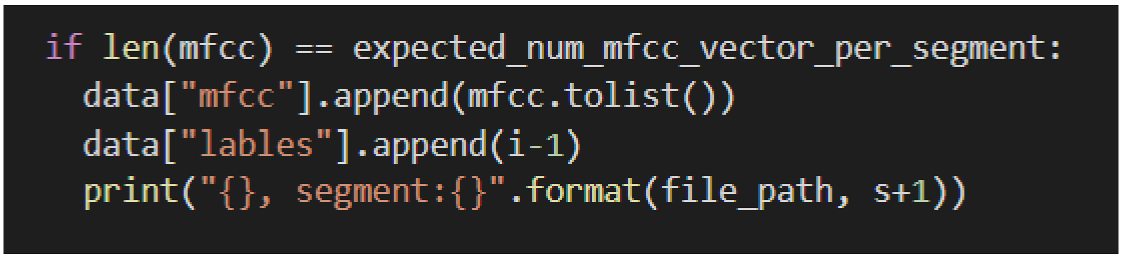
- As shown in figure 4.5 using os.walk we traverse through the directories of the dataset.
- Using dirpath.split(‘‘\\’’) we split the path of the file at ‘/ ’ character to separate the name of the genre.
- We store that name in the variable semantic_label and append it to the dictionary.



```
for f in filenames:
  file_path = os.path.join(dirpath, f)
  signal, sr = librosa.load(file_path, sr = SAMPLE_RATE)
  for s in range(num_segments):
    start_sample = num_sample_per_segment * s
    end_sample = num_sample_per_segment + start_sample

    mfcc = librosa.feature.mfcc(signal[start_sample:end_sample], sr=sr, n_fft = n_fft, n_mfcc = n_mfcc, hop_length = hop_length)
    mfcc = mfcc.T
```

Figure 4.7: Traversing through the dataset files

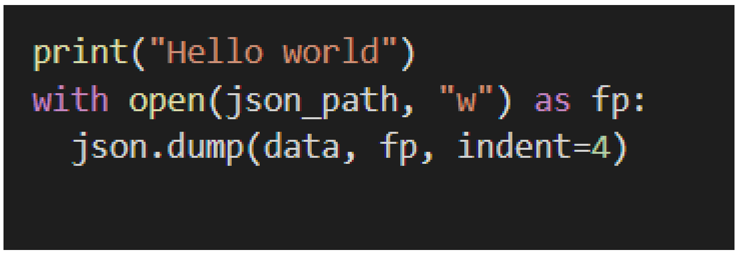
- Using the for loop shown in figure 4.6 we go through each audio file in the GTZAN dataset.
- We load the audio for feature extraction using librosa.load.
- We define the start and end of the segment in each audio, and it increments as the for-loop goes on.
- Then the mfcc is extracted for each segment using librosa library again.
- The mfcc numpy array is transposed.



```
if len(mfcc) == expected_num_mfcc_vector_per_segment:
  data["mfcc"].append(mfcc.tolist())
  data["lables"].append(i-1)
  print("{}, segment:{}".format(file_path, s+1))
```

Figure 4.8: Appending the MFCC and Labels into the JSON file

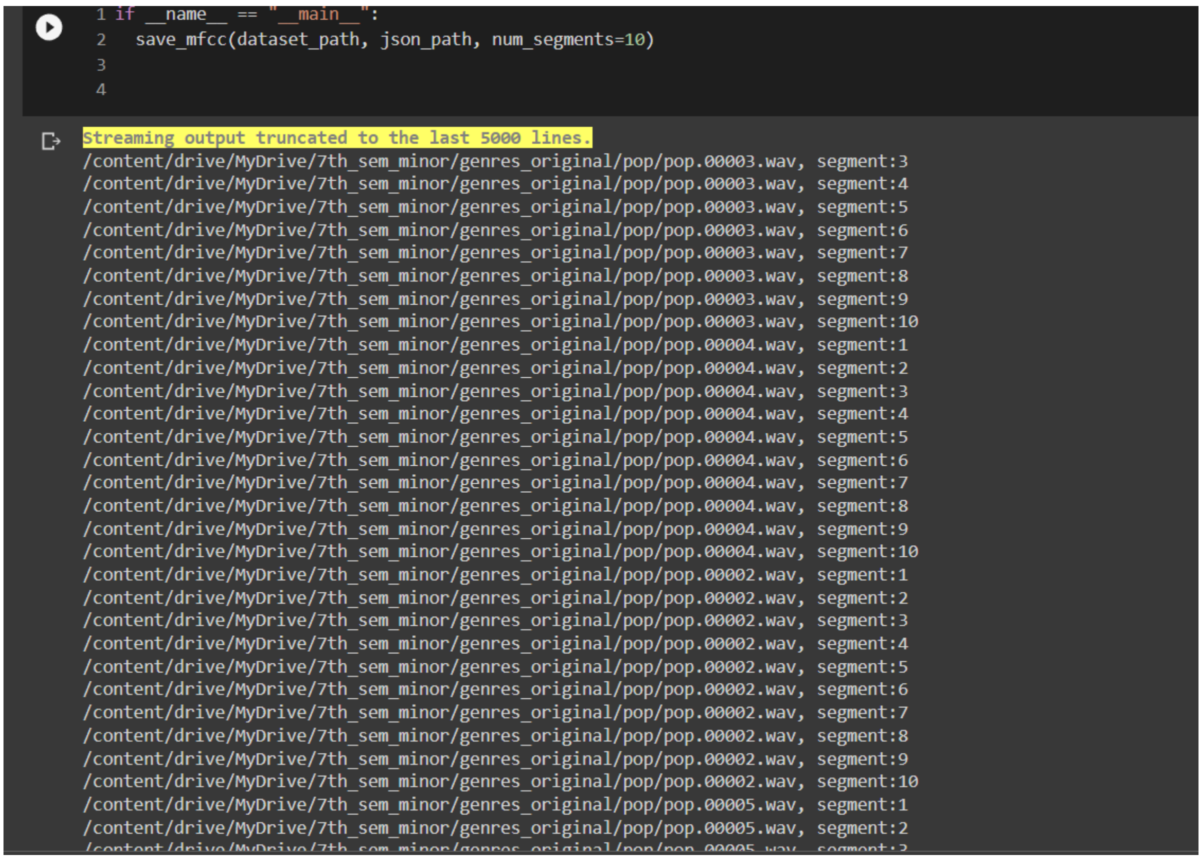
- As shown in figure 4.7 the expected number of MFCC vectors is calculated by number of samples in a segment / hop length(=512).
- We only append the mfcc if the length of the mfcc matches the length expected of the mfcc. We can’t simply append as the MFCC is a numpy array. So we change it to a list.
- The labels are obtained by decreasing the index “i” by one.



```
print("Hello world")
with open(json_path, "w") as fp:
  json.dump(data, fp, indent=4)
```

Figure 4.9: Transferring all the data into the JSON file

- The json library comes into play here as shown in figure 4.8.
- Using json.dump we can add all the mapping data, mfcc data and the labels into the file.
- This data can further implemented for the training and testing process of the deep learning model.



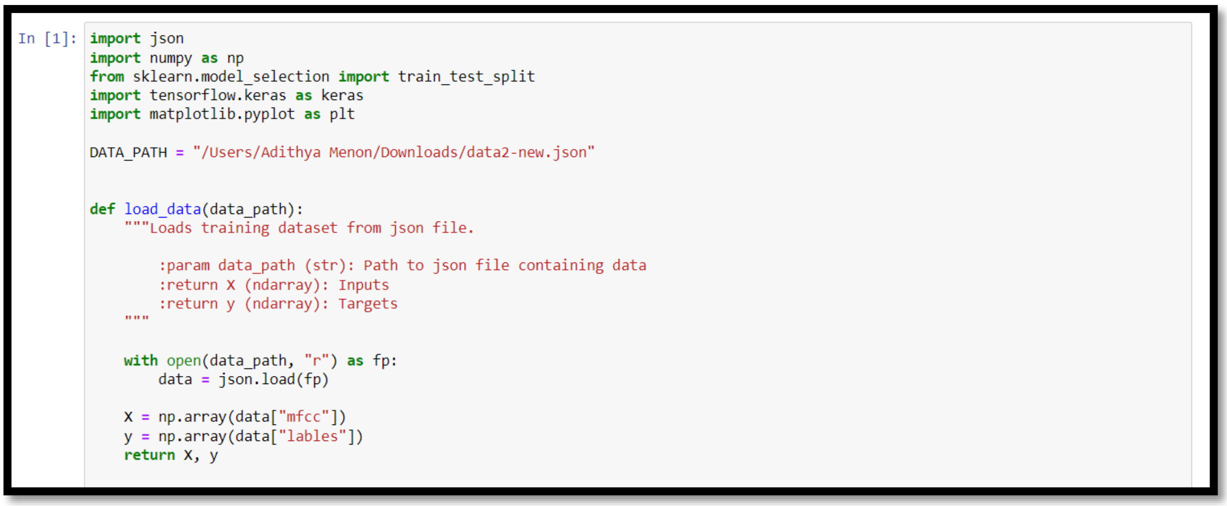
```
if __name__ == "__main__":
  save_mfcc(dataset_path, json_path, num_segments=10)


```

```
Streaming output truncated to the last 5000 lines.
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:3
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:4
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:5
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:6
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:7
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:8
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:9
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00003.wav, segment:10
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:1
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:2
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:3
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:4
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:5
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:6
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:7
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:8
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:9
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00004.wav, segment:10
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:1
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:2
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:3
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:4
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:5
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:6
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:7
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:8
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:9
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00002.wav, segment:10
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00005.wav, segment:1
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00005.wav, segment:2
/content/drive/MyDrive/7th_sem_minor/genres_original/pop/pop.00005.wav, segment:3
```

Figure 4.10: Execution of the JSON file transfer

- As shown in figure 4.9 the program traverses through all the files in the GTZAN dataset and the status of the traversing is displayed on the screen.
- After all the pre-processing the JSON file is ready with the dictionary data.
- The file can be exported to the local system as well.

4.6.2 LOADING THE JSON FILE AND SETTING UP INPUTS:

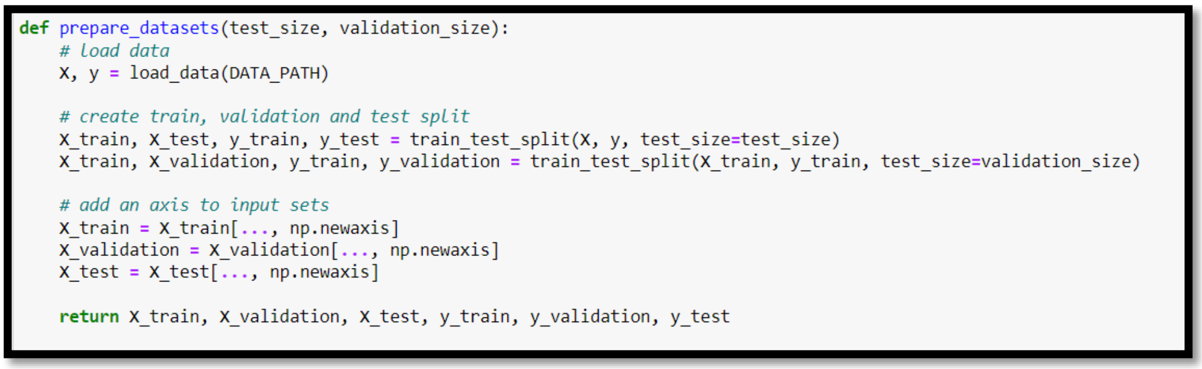


```
In [1]: import json
        import numpy as np
        from sklearn.model_selection import train_test_split
        import tensorflow.keras as keras
        import matplotlib.pyplot as plt

        DATA_PATH = "/Users/Adithya Menon/Downloads/data2-new.json"


        def load_data(data_path):
            """Loads training dataset from json file.

                :param data_path (str): Path to json file containing data
                :return X (ndarray): Inputs
                :return y (ndarray): Targets
            """

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

            X = np.array(data["mfcc"])
            y = np.array(data["lables"])
            return X, y
```

Figure 4.11: Loading the JSON file

- Since it is a JSON file, the json libraries are used to load up the file into the program as shown in figure 4.10.
- This json file includes 3 parameters:
  - Mapping – these are the 10 genre names that are present in the dataset.
  - MFCC – they are the MFCC values that were extracted from the dataset song. There are 13 values for each song. They are to be taken as input for training, testing as well as prediction. Extracting MFCCs and feeding it to the trained model is what gives us the result.
  - Labels – they are the outputs that you get after providing the model with MFCC features of a song. These labels that are present in the JSON file are labels of songs that we already know the genre of. Now the new labels that will be obtained by the prediction will be matched with the labels in the file.

4.6.3 PREPARING THE DATASETS:

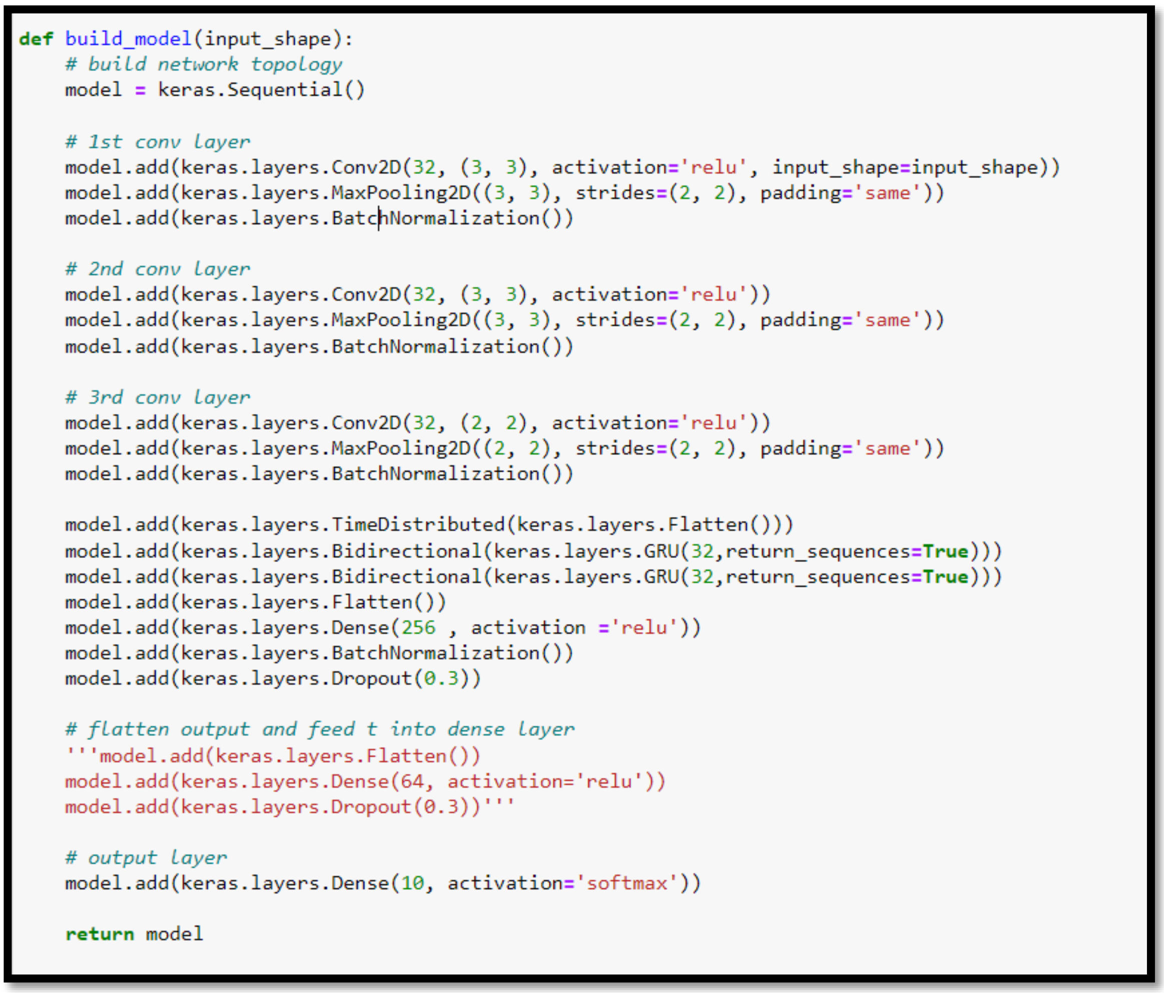


```python
def prepare_datasets(test_size, validation_size):
    # load data
    X, y = load_data(DATA_PATH)

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

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

    return X_train, X_validation, X_test, y_train, y_validation, y_test
```

Figure 4.12: Preparing the dataset by splitting the data

- As shown in figure 4.11 this function prepares the data in the dataset for training the model.
- Test size and validation size is inputted from the main function.
- The train_test_split() function take the parameters as input and splits the data in 3 ways, training set, testing set and validation set.
- We add an axis to the input sets in the end and return it back to the main function.

4.6.4 CONSTRUCTING THE NEURAL NETWORK MODEL:



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

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

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

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

    model.add(keras.layers.TimeDistributed(keras.layers.Flatten()))
    model.add(keras.layers.Bidirectional(keras.layers.GRU(32,return_sequences=True)))
    model.add(keras.layers.Bidirectional(keras.layers.GRU(32,return_sequences=True)))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(256 , activation ='relu'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.3))

    # flatten output and feed t into dense layer
    '''model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.3))'''

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

    return model
```

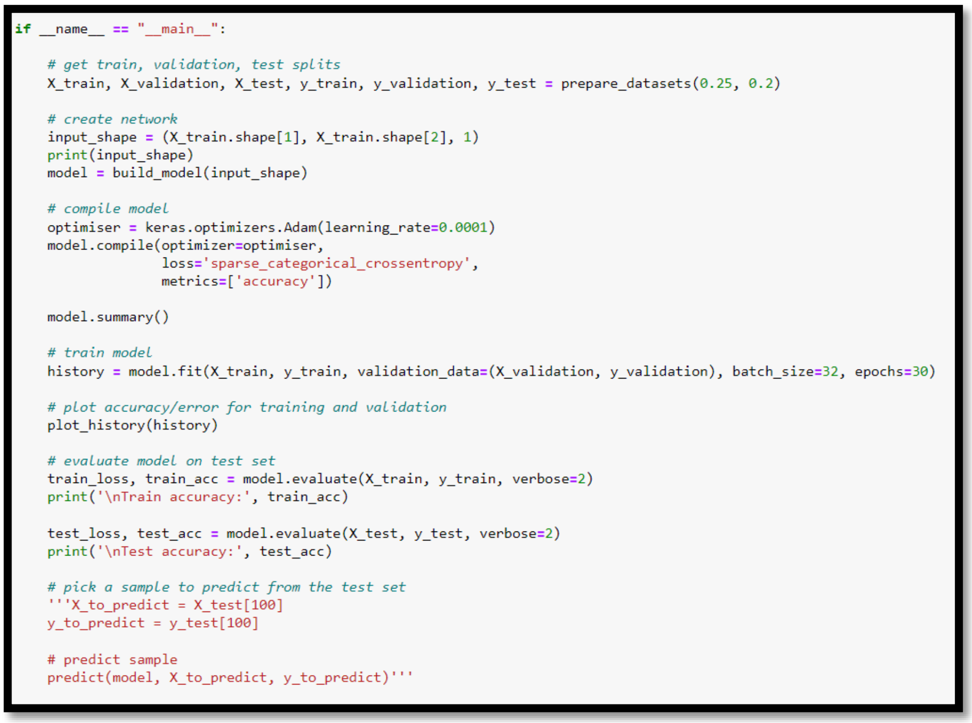
Figure 4.13: The structure of the Neural Network Model

- We use keras sequential model for creating the different layers for the program as given in the figure 4.12.
- Totally 5 layers are available, 3 convolutional layers and 2 bidirectional GRU layers.
- The first and second convolutional layer consists of batch_size 32, a kernel size of (3, 3), activation function as ReLU.
- The second convolutional consists of batch_size 32, and (2,2) sized kernel and ReLU layer as activation function.
- There is a Max pooling layer for each convolution layer, with kernels of (3, 3) for the first and second layers and (2, 2) for the third layer. After every convolutional and max pooling layer there is also a Batch normalization layer. After the max pooling happens the batch

normalization layer takes this output and normalizes the result for every mini batch that will follow.

- Then after the 3 convolutional layers are done, we give the remaining flattened output to the Time distributed layer.
- Then the output is passed on to 2 more bidirectional GRU layers.
- Bidirectional GRU layers use batch size as 32 and have update and forget gates to either retain important information or forget the unimportant information. Making this an improvement on traditional CNN.
- Then the output is flattened and fed into a dropout layer to overcome over fitting problem.
- Finally the whole result is fed into a dense layer with soft max activation function that assigns a probability to each and every possible outcome and the one with the most probability will be selected as the correct result of the model.

4.6.5 MAIN FUNCTION:



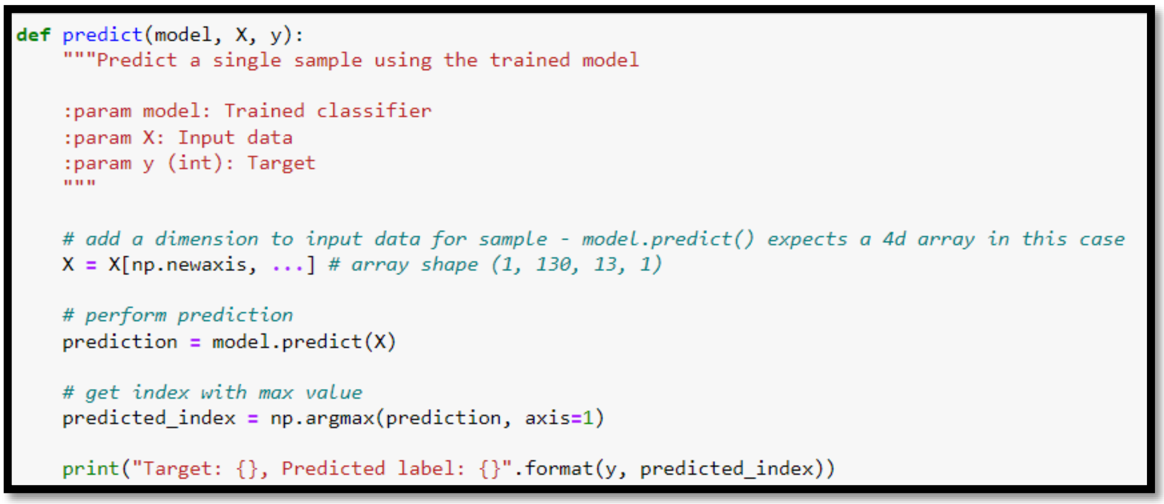
```python
if __name__ == "__main__":

    # get train, validation, test splits
    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.25, 0.2)

    # create network
    input_shape = (X_train.shape[1], X_train.shape[2], 1)
    print(input_shape)
    model = build_model(input_shape)

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

    model.summary()

    # train model
    history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32, epochs=30)

    # plot accuracy/error for training and validation
    plot_history(history)

    # evaluate model on test set
    train_loss, train_acc = model.evaluate(X_train, y_train, verbose=2)
    print('\nTrain accuracy:', train_acc)

    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
    print('\nTest accuracy:', test_acc)

    # pick a sample to predict from the test set
    '''X_to_predict = X_test[100]
    y_to_predict = y_test[100]

    # predict sample
    predict(model, X_to_predict, y_to_predict)'''
```

Figure 4.14: The main function

- First the prepare_dataset() function is called and the parameters for that function are 0.25 and 0.2 indication 25% validation set and 20% testing set, as shown in figure 4.13.

- The prepare_datasets() function returns a 4D output in which the second dimension and the third dimension along with ‘1’,which is the number of channels, is the shape of the input.
- Adam optimizer with learning rate of 0.0001 is taken as the optimizer for the model.
- The model.compile() function is called with the optimiser as a parameter.
- The summary of the compilation is displayed by model.summary().
- A graph is plotted for accuracy of training and testing data. And the final accuracies are also displayed.

4.6.6 PREDICTING THE RESULT:

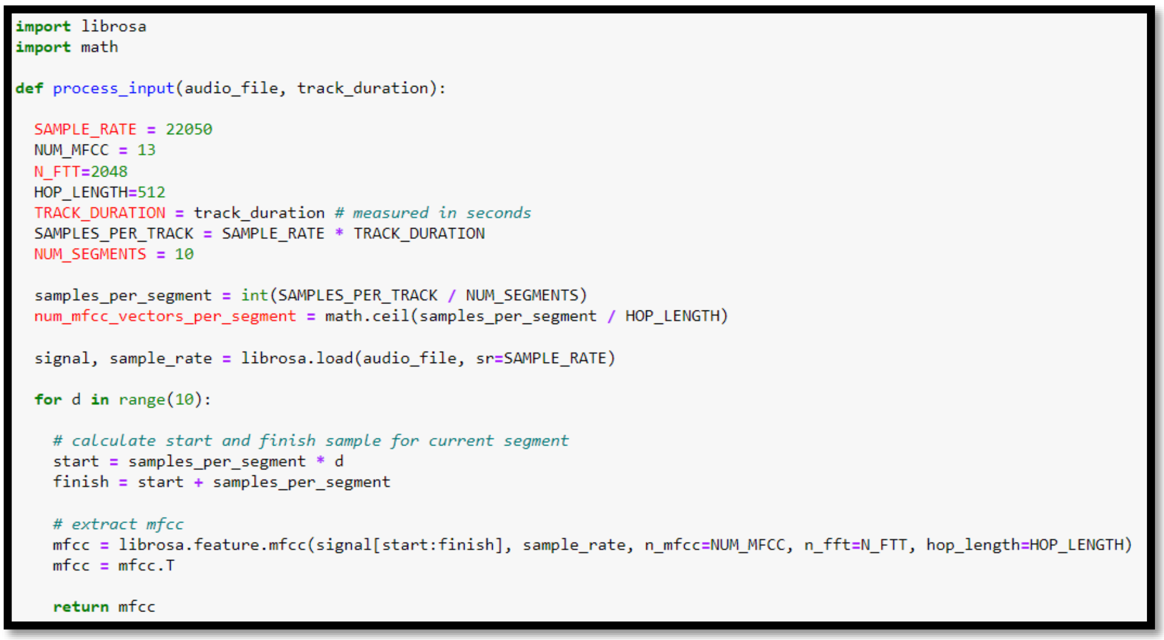


```
def predict(model, X, y):
    """Predict a single sample using the trained model

    :param model: Trained classifier
    :param X: Input data
    :param y (int): Target
    """

    # add a dimension to input data for sample - model.predict() expects a 4d array in this case
    X = X[np.newaxis, ...] # array shape (1, 130, 13, 1)

    # perform prediction
    prediction = model.predict(X)

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

    print("Target: {}, Predicted label: {}".format(y, predicted_index))
```

Figure 4.15: The prediction function

- As given in figure 4.14, the predict() function exists to analyse the features of the input song and accurately predict the output using the freshly trained CNN/CRNN model.
- This function requires 3 parameters: model, X and y. X being the MFCC data for the song and y being the label of that particular song. We know the value of y as this function is used to check if the model predicts the values of songs it has trained on. Therefore the correct label is known and is matched to the predicted label to see if they match.
- We add a new axis to the shape of the X value as the predict function requires us to add the number of batches to predict as well. Here we are just predicting one batch therefore we use np.newaxis to do the needful.
- Then the prediction is done using model.predict(). And the resulting label is printed along with the actual label for us to see if it matches or not.

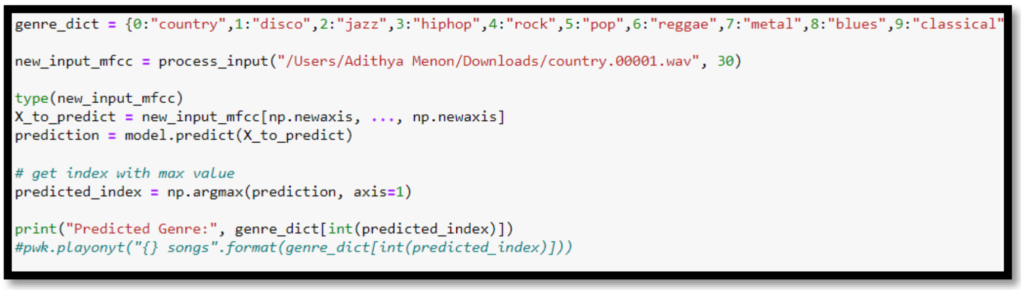
4.6.7 PREDICTING AN UNKNOWN SONG:



```
import librosa
import math

def process_input(audio_file, track_duration):

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

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

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

  for d in range(10):

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

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

    return mfcc
```

Figure 4.16: Including necessary libraries

- As given in figure 4.15, a new song is taken for prediction purposes and a function is written to extract MFCC features from the song.
- Process_input() function takes 2 parameters, audio file and track duration.
- There are few more variables like hop length, number of MFCC and number of fourier-transforms which are specified in the function.
- Librosa is used to load the song and get the sample rate.
- For 10 segments, the 13 MFCC features are taken and the list is transposed and returned.

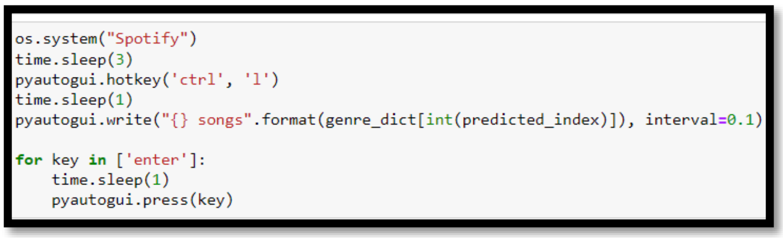


```
genre_dict = {0:"country",1:"disco",2:"jazz",3:"hiphop",4:"rock",5:"pop",6:"reggae",7:"metal",8:"blues",9:"classical"

new_input_mfcc = process_input("/Users/Adithya Menon/Downloads/country.00001.wav", 30)

type(new_input_mfcc)
X_to_predict = new_input_mfcc[np.newaxis, ..., np.newaxis]
prediction = model.predict(X_to_predict)

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

print("Predicted Genre:", genre_dict[int(predicted_index)])
#pwk.playonyt("{} songs".format(genre_dict[int(predicted_index)]))
```

Figure 4.17: Predicting the unknown sample

- A list with key value pairs is created where the genres and its respective labels are given in figure 4.16.
- The process_input() function gets the MFCC of the given song and that feature is taken as input for predicting.
- Since the predict function was used for already known songs, the same commands are written again for predicting unknown songs except without the known label. 2 new axes are added to it as this new input will have only 2D shape and predict expects 4d shape inputs.
- The result we get is a label and we match that label with the genre list we created and the value that matches with the label is the genre that is printed as the result.

4.6.8 RECOMMENDING SIMILAR SONGS:



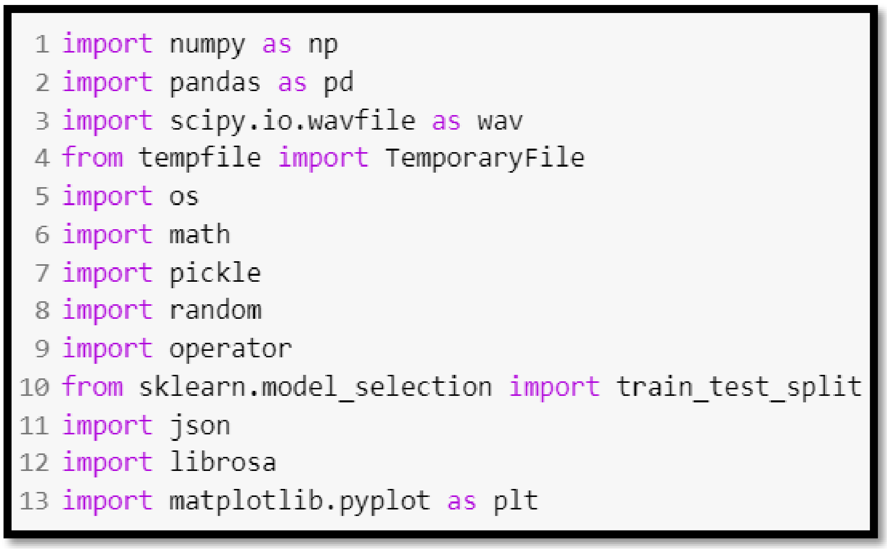
```
os.system("Spotify")
time.sleep(3)
pyautogui.hotkey('ctrl', 'l')
time.sleep(1)
pyautogui.write("{} songs".format(genre_dict[int(predicted_index)]), interval=0.1)

for key in ['enter']:
    time.sleep(1)
    pyautogui.press(key)
```

Figure 4.18: Suggesting the music

- Mainly pyautogui is used for recommendation system, as shown in figure 4.17.
- Pyautogui and OS libraries are used to open Spotify and search for genre of songs that was predicted during the prediction.
- Time library is used to delay commands to avoid task overload.

4.7 IMPLEMENTATION – FUZZY KNN:

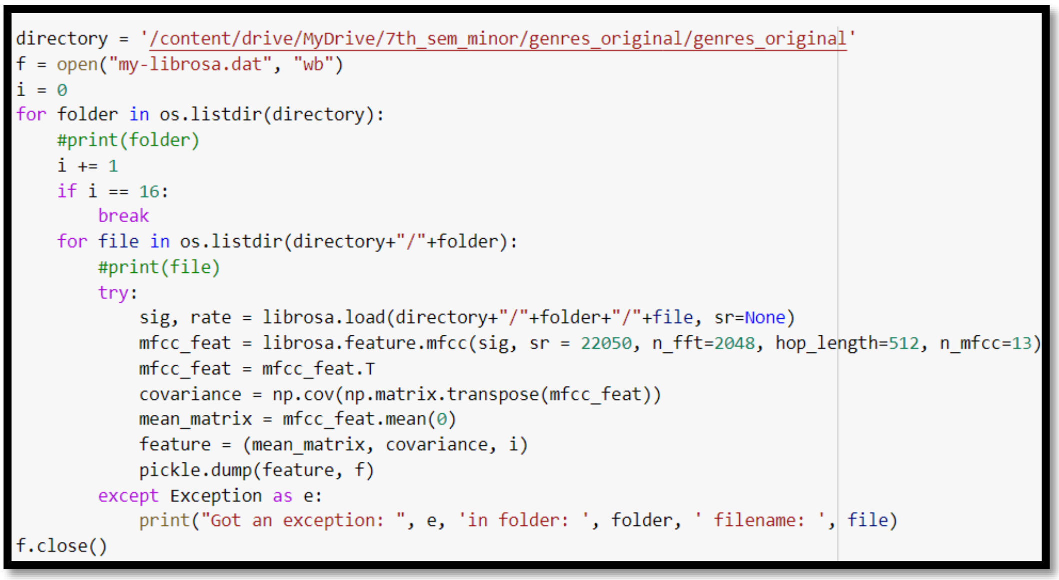
4.7.1 CREATING THE BINARY FILE



```python
import numpy as np
import pandas as pd
import scipy.io.wavfile as wav
from tempfile import TemporaryFile
import os
import math
import pickle
import random
import operator
from sklearn.model_selection import train_test_split
import json
import librosa
import matplotlib.pyplot as plt
```

Figure 4.19: The python libraries for the program

- The figure 4.19 shows the various libraries that were imported for this project.



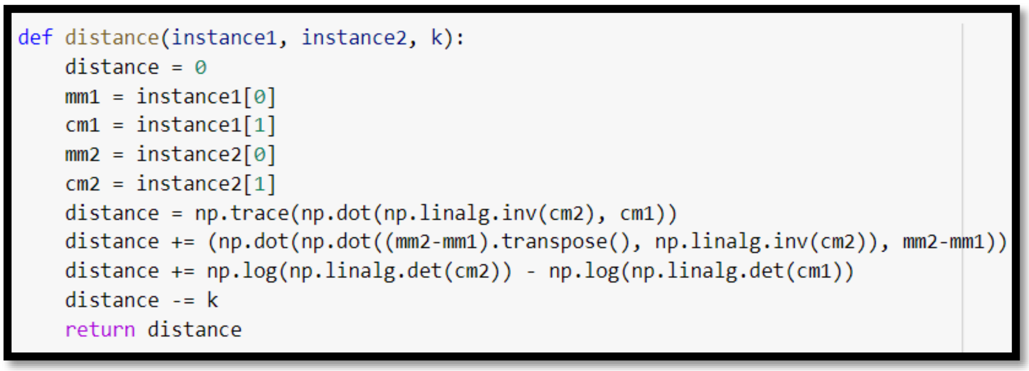
```python
directory = '/content/drive/MyDrive/7th_sem_minor/genres_original/genres_original'
f = open("my-librosa.dat", "wb")
i = 0
for folder in os.listdir(directory):
    #print(folder)
    i += 1
    if i == 16:
        break
    for file in os.listdir(directory+"/"+folder):
        #print(file)
        try:
            sig, rate = librosa.load(directory+"/"+folder+"/"+file, sr=None)
            mfcc_feat = librosa.feature.mfcc(sig, sr = 22050, n_fft=2048, hop_length=512, n_mfcc=13)
            mfcc_feat = mfcc_feat.T
            covariance = np.cov(np.matrix.transpose(mfcc_feat))
            mean_matrix = mfcc_feat.mean(0)
            feature = (mean_matrix, covariance, i)
            pickle.dump(feature, f)
        except Exception as e:
            print("Got an exception: ", e, 'in folder: ', folder, ' filename: ', file)
f.close()
```

Figure 4.20: Extraction of features from the data

- The figure 4.20 shows the whole process of accessing the dataset from the local storage and extracting the features of the files.

- Using the directory path, the OS library traverses through the folders, which are genres in this case and extract the mean mfcc and covariance and dump it into the binary file ‘my-librosa.dat’.

4.7.2 DISTANCE FUNCTION

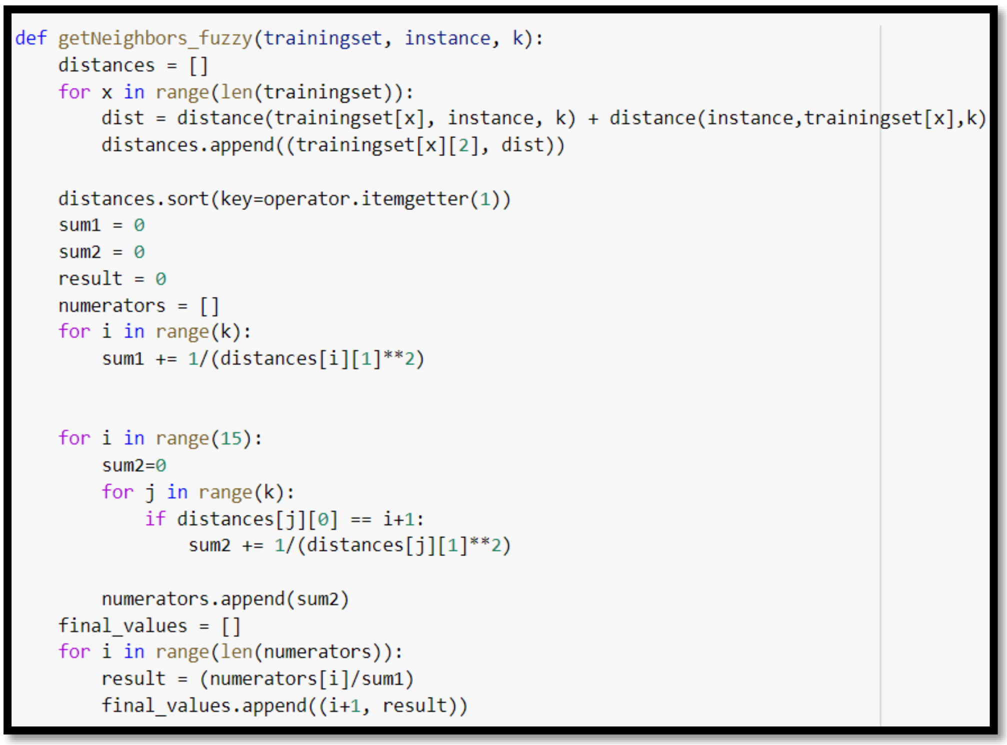


```python
def distance(instance1, instance2, k):
    distance = 0
    mm1 = instance1[0]
    cm1 = instance1[1]
    mm2 = instance2[0]
    cm2 = instance2[1]
    distance = np.trace(np.dot(np.linalg.inv(cm2), cm1))
    distance += (np.dot(np.dot((mm2-mm1).transpose(), np.linalg.inv(cm2)), mm2-mm1))
    distance += np.log(np.linalg.det(cm2)) - np.log(np.linalg.det(cm1))
    distance -= k
    return distance
```

Figure 4.21: The ‘distance’ function

- The figure 4.21 shows the distance function that was implemented for this project.
- This function is responsible for the calculation of distance between the training data and the testing data during the accuracy prediction.
- During the genres prediction of a new input, this function calculates the distance between the whole dataset and the input.

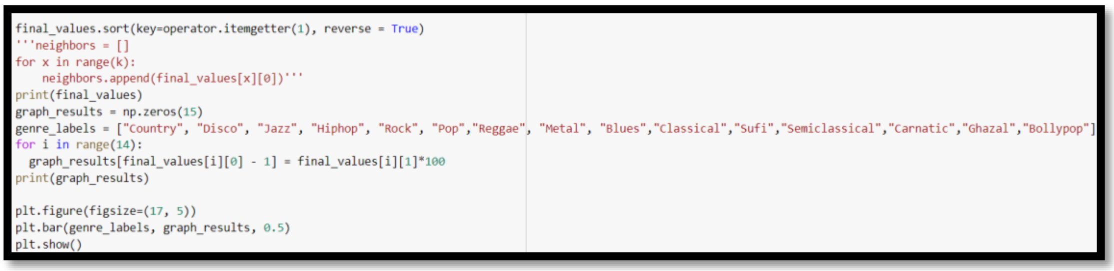
4.7.3 FUNCTION TO GET THE NEIGHBORS



```
def getNeighbors_fuzzy(trainingset, instance, k):
    distances = []
    for x in range(len(trainingset)):
        dist = distance(trainingset[x], instance, k) + distance(instance,trainingset[x],k)
        distances.append((trainingset[x][2], dist))

    distances.sort(key=operator.itemgetter(1))
    sum1 = 0
    sum2 = 0
    result = 0
    numerators = []
    for i in range(k):
        sum1 += 1/(distances[i][1]**2)


    for i in range(15):
        sum2=0
        for j in range(k):
            if distances[j][0] == i+1:
                sum2 += 1/(distances[j][1]**2)

        numerators.append(sum2)
    final_values = []
    for i in range(len(numerators)):
        result = (numerators[i]/sum1)
        final_values.append((i+1, result))
```

Figure 4.22: The ‘getNeighbors_fuzzy’ function

- The figure 4.22 shows the getNeighbors_fuzzy function that was implemented for this project.
- This function is responsible for the calling the distance function shown in the figure 4.21, and calculating the distance of the testing input given w.r.t the training dataset.
- During the genres prediction of a new input, this function calculates the distance between the whole dataset and the input. Then the distances are stored in a list, on which the fuzzy formula is implemented.

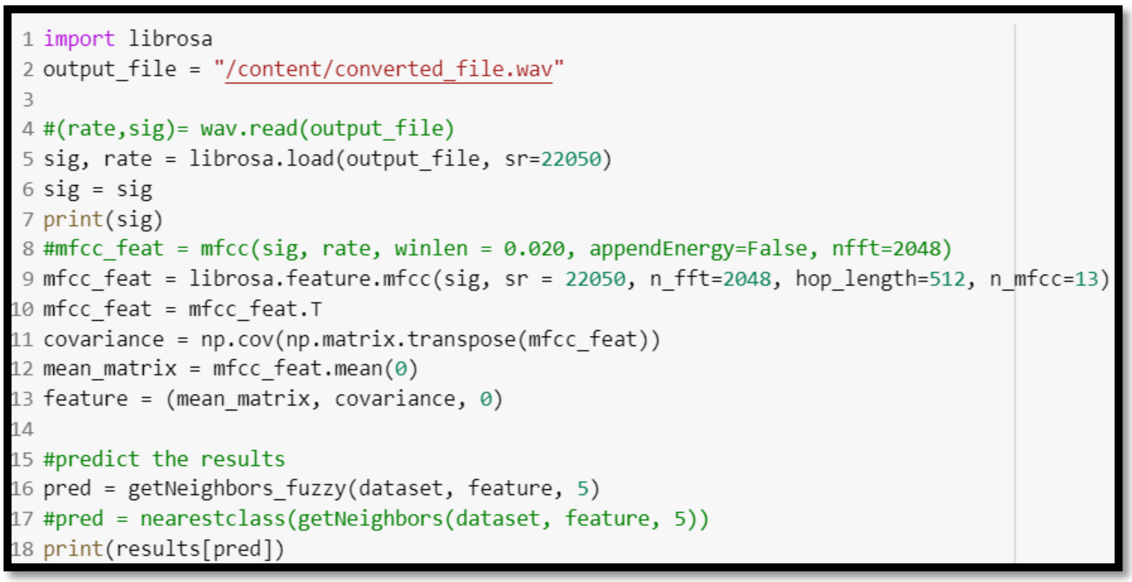


```
final_values.sort(key=operator.itemgetter(1), reverse = True)
'''neighbors = []
for x in range(k):
    neighbors.append(final_values[x][0])'''
print(final_values)
graph_results = np.zeros(15)
genre_labels = ["Country", "Disco", "Jazz", "Hiphop", "Rock", "Pop","Reggae", "Metal", "Blues","Classical","Sufi","Semiclassical","Carnatic","Ghazal","Bollypop"]
for i in range(14):
  graph_results[final_values[i][0] - 1] = final_values[i][1]*100
print(graph_results)

plt.figure(figsize=(17, 5))
plt.bar(genre_labels, graph_results, 0.5)
plt.show()
```

Figure 4.23: The ‘getNeighbors_fuzzy’ function (continued)

- As shown in figure 4.23, the variables sum1, sum2 and result are used to calculate the fuzzy membership value. The values are then sorted in ascending order and the top 5 results are presented in a graph format as the value of K is taken as 5 for this program.

4.7.4 PREDICTION OF THE SONGS



```
import librosa
output_file = "/content/converted_file.wav"

#(rate,sig)= wav.read(output_file)
sig, rate = librosa.load(output_file, sr=22050)
sig = sig
print(sig)
#mfcc_feat = mfcc(sig, rate, winlen = 0.020, appendEnergy=False, nfft=2048)
mfcc_feat = librosa.feature.mfcc(sig, sr = 22050, n_fft=2048, hop_length=512, n_mfcc=13)
mfcc_feat = mfcc_feat.T
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature = (mean_matrix, covariance, 0)

#predict the results
pred = getNeighbors_fuzzy(dataset, feature, 5)
#pred = nearestclass(getNeighbors(dataset, feature, 5))
print(results[pred])
```

Figure 4.24: The prediction code

- The figure 4.24 shows the code written to predict the genres of unknown inputs.
- This section is responsible for importing the new input unknown to the program via a link to the path of the file in local storage.
- The librosa library is used to extract the MFCC of the file. The mean of the mfcc and covariance are passed on to the getNeighbors_fuzzy() function shown in figure 4.22 as parameters along with the dataset and the k value,
- The function displays the 5 sub-genres in the entered file, which shows the percentage-wise overview of sub-genres present in the file.

CHAPTER 5

RESULT AND DISCUSSION

In this chapter we explain about the results which we have got for the proposed model and also showed the dataset where the information about the genres and the no of sample songs is given. We have stated the formulas used for calculating the accuracy of different models.

5.1 ACCURACY OF CNN

TABLE 5.1: The dataset

S no.	**Genres**	**Number of Song samples**
I.	Blues genre	100 songs
II.	Classical genre	100 songs
III.	Country genre	100 songs
IV.	Disco genre	100 songs
V.	Hip-hop genre	100 songs
VI.	Jazz genre	100 songs
VII.	Metal genre	100 songs
VIII.	Pop genre	100 songs
IX.	Reggae genre	100 songs
X.	Rock genre	100 songs
XI.	Sufi genre	100 songs
XII.	Semi-classical genre	100 songs
XIII.	Carnatic genre	100 songs
XIV.	Ghazal genre	100 songs

XV.	Bollywood-pop genre	100 songs

As shown in table 5.1, here were 15 different genres considered for training of the neural network models. Each genre have 100 song samples of 30 second duration each. In case of CNN & CRNN, the MFCC features and the respective labels were extracted and imported to a JSON file to be fed into a NN model (CNN and CRNN). Each song contained 13 MFCC feature. After passing the JSON file MFCC inputs through 3 convolutional layers, 1 dense layer and a soft max activation function the following results were obtained:

TABLE 5.2: Accuracy of the Training and Testing data of CNN model

Training Accuracy	83.64%
Testing Accuracy	71.08%

As shown in table 5.2, the kernel size for CNN was taken as 3 x 3, the dropout was 0.3 and the stride was 2,2. The testing accuracy was found to be 71.08%, which is an impressive accuracy considering we have just trained using 1500 songs and that too of 30 seconds duration each. But there is still scope of improvement and this will be discussed in the section 5.2

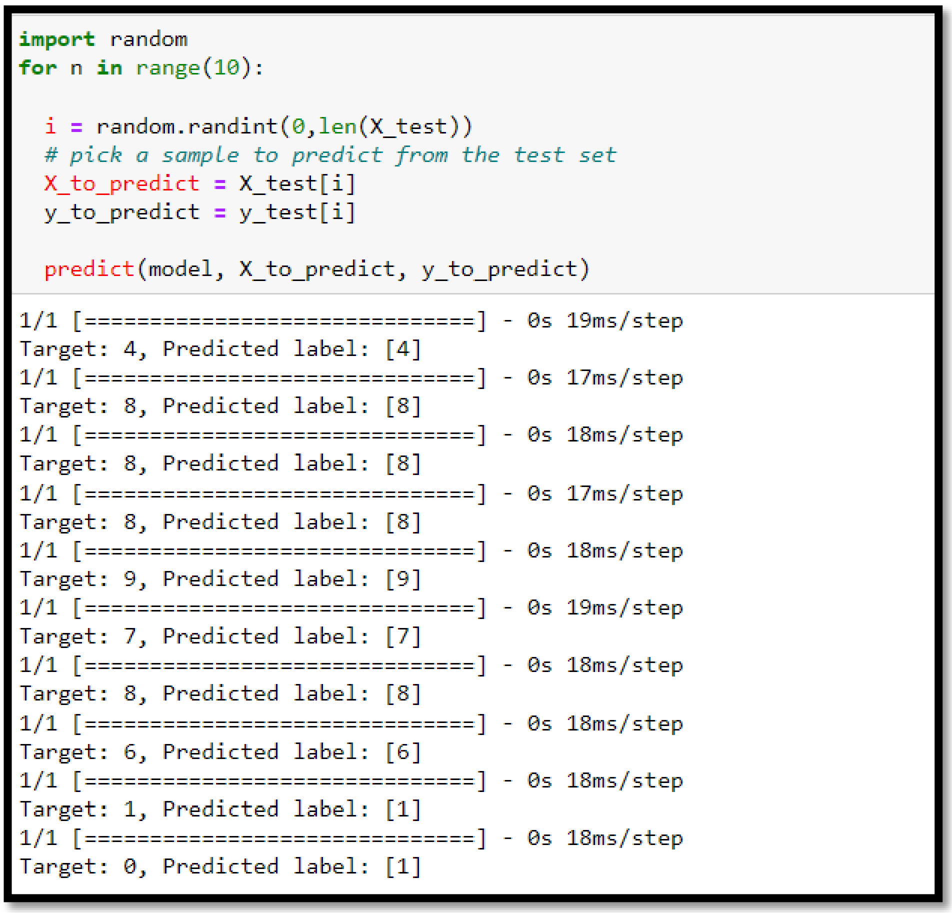
5.2 ACCURACY OF CRNN

Now to see if the results improve on implementing RNN layers with CNN layers (CRNN) we add two other layers after the convolutional layers. We reshape the output from the CNN layers and feed it to 2 bidirectional RNN-GRU layers. And then finally the output is given to a dropout layer to fix the over fitting and fed to a soft max activation function. We finally get the accuracy scores as:

TABLE 5.3: Accuracy of the Training and Testing data of CRNN model

Training accuracy	95.64%
Testing accuracy	77.41%

The accuracy of the CRNN was found to be a big improvement on the CNN results as clearly shown in table 5.3. This was because of the inclusion of RNN along with the CNN model. It is now evident that CRNN favors much more that CNN when it comes to music genre classification.



```
import random
for n in range(10):

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

  predict(model, X_to_predict, y_to_predict)
```

```
1/1 [==============================] - 0s 19ms/step
Target: 4, Predicted label: [4]
1/1 [==============================] - 0s 17ms/step
Target: 8, Predicted label: [8]
1/1 [==============================] - 0s 18ms/step
Target: 8, Predicted label: [8]
1/1 [==============================] - 0s 17ms/step
Target: 8, Predicted label: [8]
1/1 [==============================] - 0s 18ms/step
Target: 9, Predicted label: [9]
1/1 [==============================] - 0s 19ms/step
Target: 7, Predicted label: [7]
1/1 [==============================] - 0s 18ms/step
Target: 8, Predicted label: [8]
1/1 [==============================] - 0s 18ms/step
Target: 6, Predicted label: [6]
1/1 [==============================] - 0s 18ms/step
Target: 1, Predicted label: [1]
1/1 [==============================] - 0s 18ms/step
Target: 0, Predicted label: [1]
```

Figure 5.1: Model predicting 8 out of 10 random songs from the testing data correctly

When tested against 10 random songs, 8 out of 10 time the music genre was correctly predicted as shown in figure 5.1. Additionally a song from the internet was taken for testing and fed into the system and the program was accurately able to figure out the genre of the song. Then the song genre was provided to the suggestion part using python libraries, pyautogui, and suggestions were given.

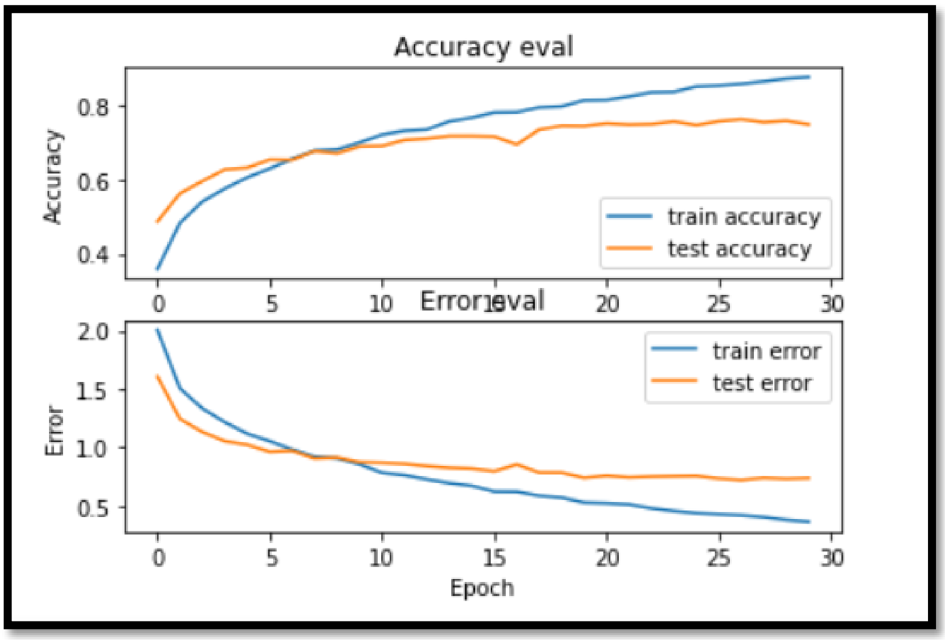


Figure 5.2: The graph plotted between Training and Testing accuracies and error percentages

The graphs in Figure 5.2 shows the graphs between the accuracy of the training as well as the validation set of data as a function of Epochs and the loss in the training and validation set as a function of the number of epochs. The validation set is labelled as test by convention.

TABLE 5.4: Parameters for Training and Classification

Parameters	**Values**
learning_rate	0.001
batch_size	32
epochs	30
dropout	0.3
n_mfcc	130 * 13 * 1

Considering the parameters in table 5.4, the neural network's accuracy is calculated using:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{5.1}$$

In the above formula:

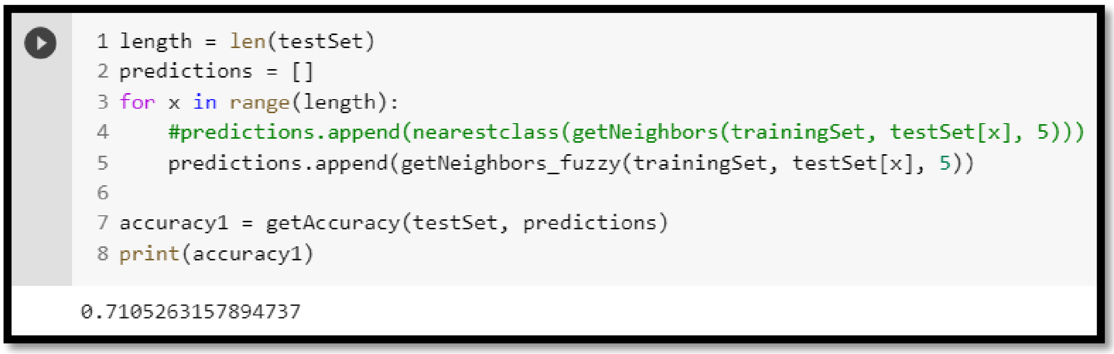
TP – True positive

TN – True negative

FP – False positive

FN – False negative

5.3 ACCURACY OF FUZZY KNN



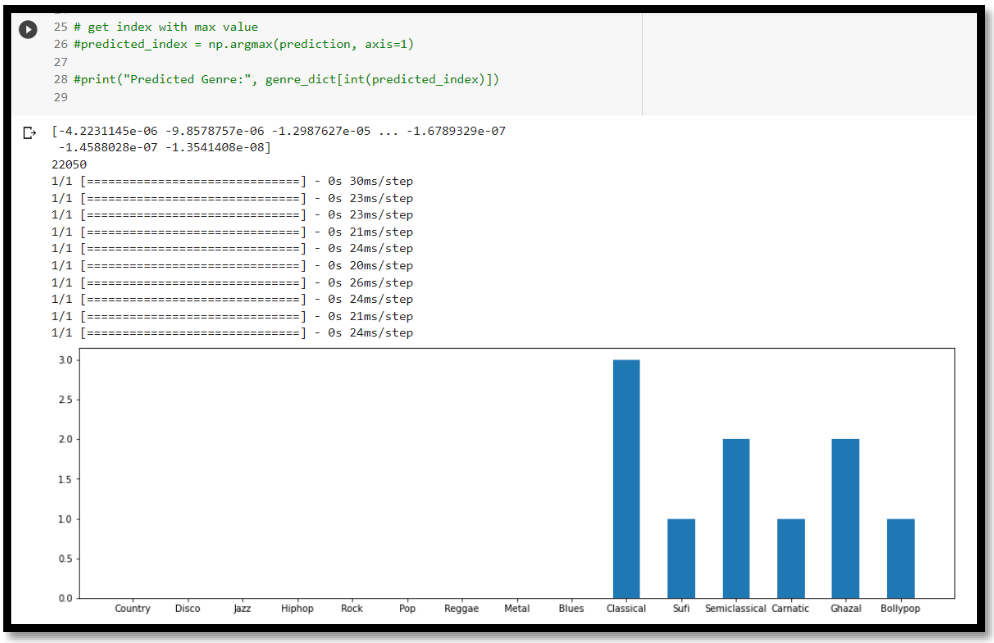
```
length = len(testSet)
predictions = []
for x in range(length):
    #predictions.append(nearestclass(getNeighbors(trainingSet, testSet[x], 5)))
    predictions.append(getNeighbors_fuzzy(trainingSet, testSet[x], 5))

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

```
0.7105263157894737
```

Figure 5.3: The Fuzzy KNN algorithm accuracy

The model as shown in Figure 5.3 is demonstrating an accuracy of 71.05%. This accuracy is being boosted by the Fuzzy algorithm assisting the model to select the closest class to the input.

It should be noted that the accuracies of the model indicate how successfully the model is able to predict a music's genre in the very first try, or which genre is in the majority. However it is to be noted that one could argue that a particular song could be of a different genre than the labelled genre as modern music is more often than not a fusion of multiple genres. Therefore one should also take the model's ability to predict all the genres present in the song and how accurate the result is, into consideration as well.

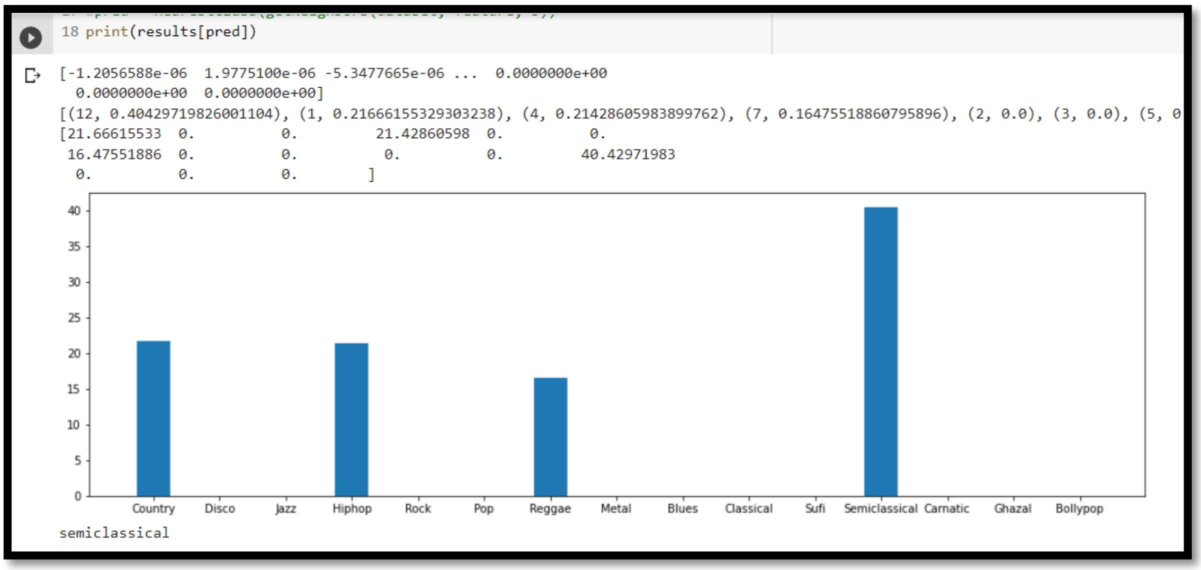


```
# get index with max value
#predicted_index = np.argmax(prediction, axis=1)

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

```

```
[-4.2231145e-06 -9.8578757e-06 -1.2987627e-05 ... -1.6789329e-07
 -1.4588028e-07 -1.3541408e-08]
22050
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
```

Figure 5.4: The sub-genres predicted by the CRNN model

The song used in figure 5.4 to test the model is a Bollywood Sufi song which is classified as a Classical, Sufi, Semi-classical, Ghazal and a Bollywood pop song. The model has accurately analysed the instrumental sounds and predicted the sub-genres accurately.



```
print(results[pred])
```

```
[-1.2056588e-06  1.9775100e-06 -5.3477665e-06 ...  0.0000000e+00
  0.0000000e+00  0.0000000e+00]
[(12, 0.40429719826001104), (1, 0.21666155329303238), (4, 0.21428605983899762), (7, 0.16475518860795896), (2, 0.0), (3, 0.0), (5, 0
[21.66615533  0.          0.         21.42860598  0.          0.
 16.47551886  0.          0.          0.          0.         40.42971983
  0.          0.          0.        ]
```

```
semiclassical
```

Figure 5.5: The sub-genres predicted by the Fuzzy-KNN model

Figure 5.5 shows us the result of Fuzzy KNN classifying the same classical Bollywood pop song used to test the CRNN model. The results while classifying the song mostly as a semi-classical song, still predicts sub genres as Country, hip-hop and reggae which is inaccurate.

This comparison demonstrates the real difference in performances between the CRNN model and Fuzzy KNN model. All results of CRNN remain relevant to the type of music with respect to the style of vocals and the type of instruments used.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This project has achieved a remarkable feat by demonstrating the accuracy and effectiveness of deep learning technologies and Python libraries in suggesting music based on sample inputs. By implementing Convolutional Recurrent Neural Networks (CRNN) and Fuzzy KNN algorithms, the project has effectively classified music into separate genres with great efficiency. The use of these algorithms and techniques is a testament to their remarkable ability to solve complex problems.

The project's success is further attributed to the features of the songs provided that have helped classify music accurately. The project's classification system is an indirect testimony to the remarkable performance of Convolutional Recurrent Neural Networks when faced with the task of image classification. Additionally, the project's use of the Fuzzy membership association algorithm implemented to the KNN model has also been shown to be highly effective.

The project has provided a comprehensive comparison between CRNN and Fuzzy KNN algorithms, and the difference in results was observed, with CRNN being found to be the better fit for the project. The use of deep learning technologies, coupled with efficient algorithms, has contributed to the project's success.

Moreover, the project's user-friendliness is commendable, as the user can provide any custom sample song to the program, and it will work just fine. This is a clear indication of how useful Neural Networks and Deep Learning techniques can be utilized to their full potential, even in solving everyday problems.

In conclusion, the project's success is a testament to the remarkable capabilities of deep learning technologies and algorithms in solving complex problems. It serves as an excellent example of how these technologies can be leveraged to their full potential and demonstrates the impact that they can have in various fields, including music.

The potential uses of this project are vast, and the project has set a strong foundation for future improvements and advancements. One of the key areas for potential future improvements is the implementation of CNN classification techniques with a mixture of Gated Recurrent Units (GRU) layers. The use of GRU layers in the model will provide a more thorough analysis of the audio and enable even more accurate predictions of the genre.

Additionally, as the model has been found to achieve impressive accuracy scores with just 30 seconds of audio clip, it is expected that with larger amounts of data and longer audio samples, the model's performance will improve even further.

The future uses of this project is vast. The implementation of CNN classification techniques with a mixture of Gated Recurrent Units layers provide us with a thorough analysis of the audio and is able to predict the genre. With just 30 seconds of audio clip the accuracy score is still found to be impressive. So with more amounts of data and longer audio samples one can only imagine how well the model could perform.

This work can be extended more by creating a dedicated application or a software that can be used as an interface for the suggestion system. The application could be an interface that lets the user input the song he/she wants and global results could be shown including features like curating a dedicated playlist for the user, in turn enhancing the user experience of exploring the project. That could not only be an idea worth making an effort for and would imply the concepts of fields that cross the boundaries of just Deep Learning and make other fields of computing collaborate with itself.

REFERENCES

[1] Adiyansjah, Alexander A S Gunawan, Derwin Suhartono, Music Recommender System Based on Genre using Convolutional Recurrent Neural Networks, Procedia Computer Science, Volume 157, 2019, Pages 99-109, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2019.08.146. (https://www.sciencedirect.com/science/article/pii/S1877050919310646)

[2] L. K. Puppala, S. S. R. Muvva, S. R. Chinige and P. S. Rajendran, "A Novel Music Genre Classification Using Convolutional Neural Network," 2021 6th International Conference on Communication and Electronics Systems (ICCES), 2021, pp. 1246-1249, doi: 10.1109/ICCES51350.2021.9489022.

[3] Chillara, S., Kavitha, A. S., Neginhal, S. A., Haldia, S., & Vidyullatha, K. S. (2019). Music genre classification using machine learning algorithms: a comparison. Int Res J Eng Technol, 6(5), 851-858.

[4] F. Sultana, A. Sufian and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), 2018, pp. 122-129, doi: 10.1109/ICRCICN.2018.8718718.

[5] N. Pelchat and C. M. Gelowitz, "Neural Network Music Genre Classification," in Canadian Journal of Electrical and Computer Engineering, volume 43, number 3, pp. 170-173, Summer 2020, doi: 10.1109/CJECE.2020.2970144.

[6] J. Despois, Finding the Genre of a Song With Deep Learning—A.I. Odyssey Part. 1, Dec. 2018, [online] Available: https://hackernoon.com/finding-the-genre-of-a-song-with-deep-learning-da8f59a61194

[7] T-RECSYS: A Novel Music Recommendation System Using Deep Learning, March 2019 T-RECSYS: A Novel Music Recommendation System Using Deep Learning | IEEE Conference Publication | IEEE Xplore

[8] Y. Im, P. Prahladan, T. H. Kim, Y. G. Hong and S. Ha, "Snn-cache: A practical machine learning-based caching system utilizing the inter-relationships of requests", *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*, pp. 1-6, March 2018.

[9] C.-W. Chen, C. D. Boom, J. Garcia-Gathright, P. Lamere, J. McInerney, V. Murali, et al., "The million playlist dataset", *Spotify - RecSys Challenge 2018*, 2018, [online] Available: https://recsys-challenge.spotify.com/dataset.

[10] Bisharad, Dipjyoti, and Rabul Hussain Laskar. "Music Genre Recognition Using Residual Neural Networks." TENCON 2019-2019 IEEE Region 10 Conference (TENCON). IEEE, 2019.

[11] Yang, Hansi, and Wei-Qiang Zhang. "Music Genre Classification Using Duplicated Convolutional Layers in Neural Networks." Interspeech. 2019.

[12] Gessle, Gabriel, and Simon Åkesson. "A comparative analysis of CNN and LSTM for music genre classification." (2019).

[13] M. Ahmed, M. T. Imtiaz and R. Khan, "Movie recommendation system using clustering and pattern recognition network", 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), pp. 143-147, Jan 2018.

[14] I. H. Mwinyi, H. S. Narman, K. C. Fang and W. S. Yoo, "Predictive self-learning content recommendation system for multimedia contents", 2018 Wireless Telecommunications Symposium (WTS), pp. 1-6, April 2018.

[15] A Survey of Music Recommendation Systems with a Proposed Music Recommendation System A Survey of Music Recommendation Systems with a Proposed Music Recommendation System | SpringerLink -July 2019

[16] P. Deshmukh, K. Geetanjali, A Survey of Music Recommendation System (2018) International Journal of Scientific Research in Computer Science, Engineering and Information Technology (ijsrcseit.com)

[17] Ferretti, Stefano. "Clustering of Musical Pieces through Complex Networks: an Assessment over Guitar Solos." IEEE MultiMedia (2018).

[18] Clustering of Musical Pieces Through Complex Networks: An Assessment Over Guitar Solos | IEEE Journals & Magazine | IEEE Xplore

[19] Nam, Juhan, et al. "Deep learning for audio-based music classification and tagging: Teaching computers to distinguish rock from bach." IEEE signal processing magazine 36.1 (2018): 41-51.

[20] Elbir, Ahmet, and Nizamettin Aydin. "Music genre classification and music recommendation by using deep learning." Electronics Letters 56.12 (2020): 627-629.

APPENDIX

This section of the report contains the briefs about the various libraries and packages used to develop this project. The programming language used to develop this project is python. This development of this programme was done by importing multiple python libraries into the workspace of the IDE. The Integrated Development Environment used for deploying this program was Google Colab. The record of the main packages used in this programme are:

1. Numpy: This python package is most famous for performing numerical calculations and working with different arrays and lists. Numpy is short for Numerical Python. Numpy provides a very easy and convenient method to handle and use multi-dimensional lists for computation. It is an essential tool for programs that involve large datasets and lists.
2. Sklearn or Scikit-learn: This is a python package infamous for its plethora of features tailored for machine learning and deep learning applications. This package offers services like model training, dataset splitting, data pre-processing, feature selection etc. Scikit-learn is a Python library that provides an easy-to-use interface on top of established scientific computing libraries such as NumPy, SciPy, and matplotlib.
3. Tensorflow: TensorFlow, developed by Google, is a well-known open-source machine learning framework. It comes with a comprehensive set of tools and libraries for creating and training machine learning models such as deep neural networks. TensorFlow is designed to be highly versatile and scalable, and it may be used to tackle a wide range of problems such as image and audio recognition, natural language processing, and so on. TensorFlow is widely used in industry and academia for image and audio recognition, natural language processing, recommendation systems, and other purposes. It is a popular solution for both research and production use cases because to its scalability and adaptability.
4. Keras: Keras is a TensorFlow-based high-level neural networks API that provides a simplified and user-friendly interface for constructing and training deep neural networks. Keras was developed with the goal of making deep learning more accessible to a wider range of users, including researchers, engineers, and students. Keras offers a simple interface for building and training neural networks. It includes pre-built layers like as convolutional layers, pooling layers, and recurrent layers, as well as activation algorithms, loss functions, and optimizers. These

building blocks may be easily combined to create a number of neural network architectures. Some of its key features are:

- Built-in support for common deep learning tasks, such as image classification, object detection, text generation, and more.
- Integration with other popular deep learning libraries, such as TensorFlow and Theano, allowing for efficient computation on CPUs and GPUs.
- Simple and easy-to-use interface for building and training neural networks, allowing users to quickly prototype and experiment with different architectures and configurations.

5. Matplotlib: Matplotlib is a Python tool for creating data visualisations. It comes with a number of tools for creating graphs, charts, and other visualisations that may be tweaked in a variety of ways. Matplotlib is designed to be adaptable and powerful, and it may be used to create high-quality visualisations for scientific study, data analysis, and other purposes. Matplotlib is a Python tool for creating data visualisations. It comes with a number of tools for creating graphs, charts, and other visualisations that may be tweaked in a variety of ways. Matplotlib is designed to be adaptable and powerful, and it may be used to create high-quality visualisations for scientific study, data analysis, and other purposes. Some of its key features are:
    - Compatibility with various output formats, including PNG, PDF, and SVG
    - Support for a wide range of plots and visualizations, including line plots, scatter plots, bar plots, histograms, and heatmaps.
    - Multiple ways of creating plots, including a MATLAB-style interface and an object-oriented interface.
    - Customizable settings for controlling the appearance of plots, including colors, fonts, labels, and axes.
6. Pandas: Pandas is a popular open-source library for data manipulation and analysis in Python. It provides data structures for efficiently storing and manipulating large and complex datasets, as well as a variety of tools for data cleaning, transformation, and analysis. The core data structure in Pandas is the DataFrame, which is a two-dimensional table-like structure that can store heterogeneous data types. DataFrames can be easily loaded from various sources such as CSV files, Excel spreadsheets, SQL databases, and web APIs. Pandas also provides a wide range of functions and methods for data manipulation, such as filtering, sorting, grouping, and merging data.

It also supports a variety of data visualization tools, making it easy to create charts and graphs from your data. Some of the key features of Pandas are:

- Efficient data handling and manipulation
- Flexible indexing and slicing of data
- Built-in support for data visualization
- Easy integration with other Python libraries and tools

7. Librosa: Librosa is a Python module that analyses and processes audio signals. It is based on NumPy, SciPy, and scikit-learn, and it includes a variety of tools and methods for working with audio data. Librosa also contains audio signal visualisation capabilities like as spectrograms, chromagrams, and beat and tempo information. These visualisations may assist you in understanding the structure and content of audio signals, as well as identifying patterns and characteristics for use in machine learning models. Some of the key features of Librosa are:
    - Data visualization, including plotting of waveforms, spectrograms, and feature distributions.
    - Feature extraction, including extraction of common audio features such as Mel-frequency cepstral coefficients (MFCCs), chroma features, and spectral contrast.
    - Efficient audio signal processing, including time domain transformations such as resampling and filtering, and frequency domain transformations such as Fourier transforms and spectrograms.

IMPORTING ALL THE NECESSARY LIBRARIES FOR THE PROPER FUNCTIONING OF OUR CODE:

```
from google.colab import drive
drive.mount('/content/drive/', force_remount=True)
import numpy as np
import pandas as pd
import scipy.io.wavfile as wav
from tempfile import TemporaryFile
import os
import math
import pickle
import random
import operator
from sklearn.model_selection import train_test_split
import json
import librosa
import matplotlib.pyplot as plt
!pip install python_speech_features
!pip install pydub
from python_speech_features import mfcc
from pydub import AudioSegment
from os import path
```

In the above code lines we are importing all the important libraries for our code to excute in a proper way so that we can complete our task properly if these libraries are not installed then we can not achieve our goal by writing this code as we all know that there are certain requirements which we have to fulfil and certain pre-written codes and other components which we should give to our python code by importing the libraries.

```
from google.colab import drive
drive.mount('/content/drive/', force_remount=True)
```

These two lines are used to mount our google drive account to the Colab runtime environment. We are mounting our drive to colab runtime environment because all our datasets are stored in the drive which we need for the testing training and output of our program and once we will mount it then we can access and work with files and folders stored in our Drive directly from our Colab notebooks.

Once mounted we are able to read or write data files, such as audio or text files, or when we want to save our trained machine learning models or other important data.

It makes it easy for us to work with our local files which are stored in our local device.

```
import scipy.io.wavfile as wav
```

We are using this import function for importing the **wavfile** module from the **scipy.io** package the properties by which we can save our audio in the wav format.

The **wavfile** module provides functions for reading and writing audio files in WAV format, which is a commonly used format for storing uncompressed audio data.

And we are also shortening the name and giving it a alias name which makes it easier for us to write the code.

```
from tempfile import TemporaryFile
```

Here tempfile is the Python module used in a situation, where we need to read multiple files, change or access the data in the file, and gives output files based on the result of processed data. Each of the output files produced during the program execution was no longer needed after the program was done. In this case, a problem arose that many output files were created and this cluttered the file system with unwanted files that would require deleting every time the program ran.

In this situation, tempfiles are used to create temporary files so that next time we don't have to find delete when our program is done with them.

Math function is imported in this code using import math for the means of calculation functions as we need to calculate distance, accuracy and various other things in our code.

```
import pickle
```

The **pickle** module provides functionality for serializing and deserializing Python objects, which means it can convert a Python object into a stream of bytes that can be written to a file or sent over a network, and then convert that stream of bytes back into the original Python object.

We typically use the **pickle** module when we need to save or load complex Python data structures, such as lists, dictionaries, or even entire machine learning models. By serializing these objects into a byte stream, we can save them to a file or database, and then reload them later without losing their internal structure or type.

The pickle module is used for implementing binary protocols for serializing and de-serializing a Python object structure.

Pickling: It is a process where a Python object hierarchy is converted into a byte stream.

Unpickling: It is the inverse of Pickling process where a byte stream is converted into an object hierarchy.

```
import operator
```

We are using this command to import the operator module. This module provides a set of efficient functions that perform various operations on Python objects, such as arithmetic, comparison, and logical operations.

Using the **operator** module we can lead to more efficient and readable code which is very much helpful while doing this this project as it is very long and we need to write a long lines of codes.

```
from sklearn.model_selection
```

Sklearn's model selection is a module which provides various functions to cross-validate our model, tune the estimator's hyperparameters, or produce validation and learning curves.

We are using !pip install python_speech_features command to install the Python Speech Features library in our Python environment. The Python Speech Features library provides a set of useful features for speech signal processing and analysis, including Mel frequency cepstral coefficients (MFCCs), which are commonly used in speech recognition and other speech-related applications.
By installing the Python Speech Features library, we can use these features in our Python code to extract meaningful information from speech signals, such as the speaker's identity, emotions, or spoken words. This library can be particularly useful in applications such as voice recognition, speaker identification, and speech synthesis.
The **!pip** command is a package installer for Python, and it allows us to download and install packages from the Python Package Index using a command-line interface
!pip install pydub We are using this command to install the Pydub library in our Python environment. Pydub is a Python module for working with audio files, providing an easy-to-use interface for manipulating various audio formats, such as MP3, WAV, and AIFF. With Pydub, we can perform a wide range of audio operations, such as slicing and concatenating audio files, changing the volume, adjusting the speed, and applying various effects like reverb, fade-in, and fade-out. Pydub also allows us to export our modified audio files to different formats and can be used for tasks like generating text-to-speech audio or creating music samples.

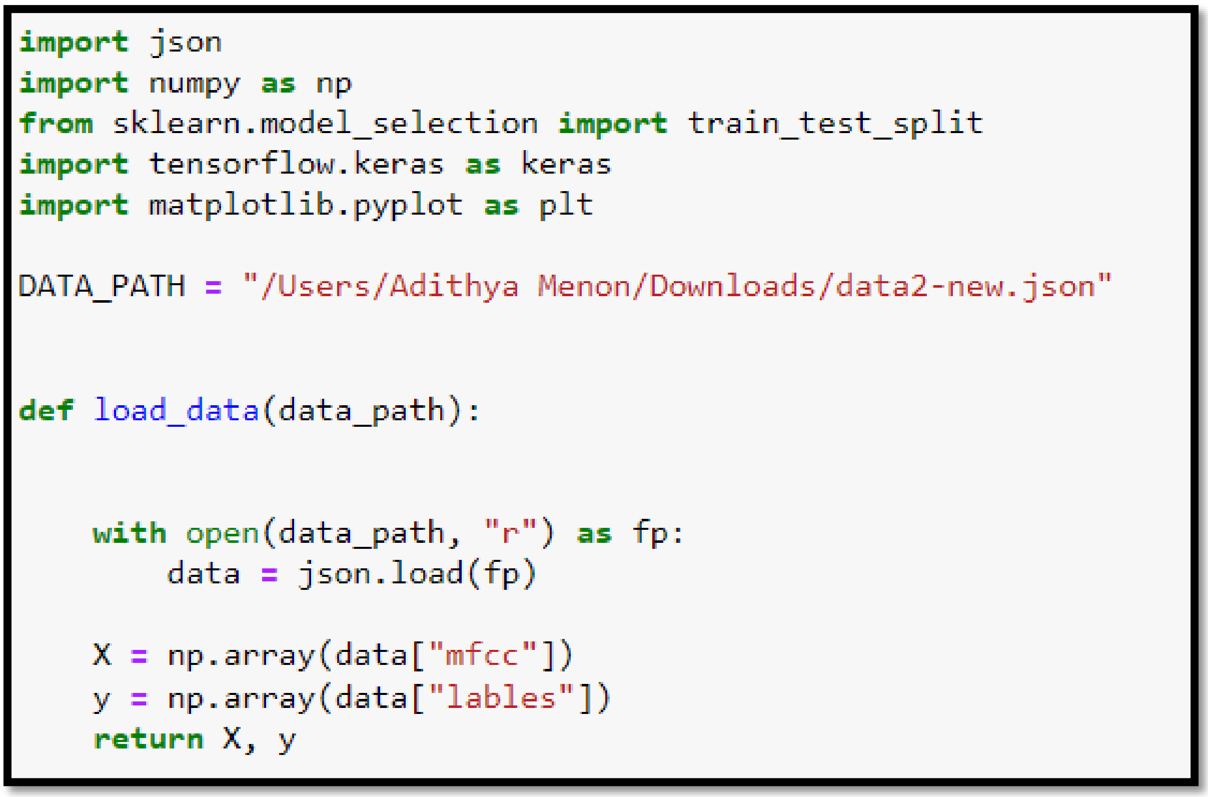
By using the **!pip** command followed by the package name (**pydub** in this case), we can easily install the package and start using its functions and modules in our Python code. With Pydub, we can efficiently manipulate and process audio files in our Python projects, making it a valuable tool for various applications, such as audio editing, music production, and speech recognition.

```
from pydub import AudioSegment
```

We are using this command as we want to import the **AudioSegment** class from the Pydub library. The **AudioSegment** class is the primary interface for working with audio files in Pydub, and it provides a range of methods and attributes for loading, manipulating, and exporting audio files.

Once we import the **AudioSegment** class using the **from pydub import AudioSegment** statement, we can create an instance of the class and load an audio file into it.

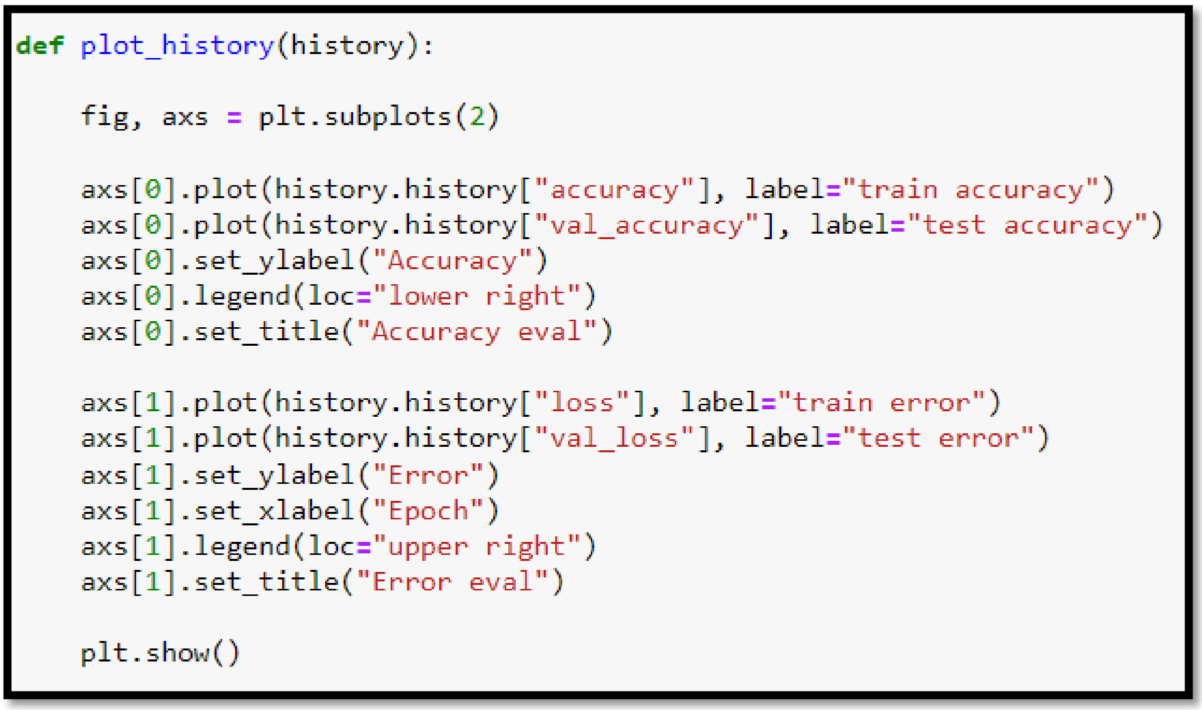
USING CONVOLUTIONAL RECURRENT NEURAL NETWORKS TO MAKE SONG PREDICTIONS:



```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

DATA_PATH = "/Users/Adithya Menon/Downloads/data2-new.json"


def load_data(data_path):


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

    X = np.array(data["mfcc"])
    y = np.array(data["lables"])
    return X, y
```

Figure 7.1: Loading Data

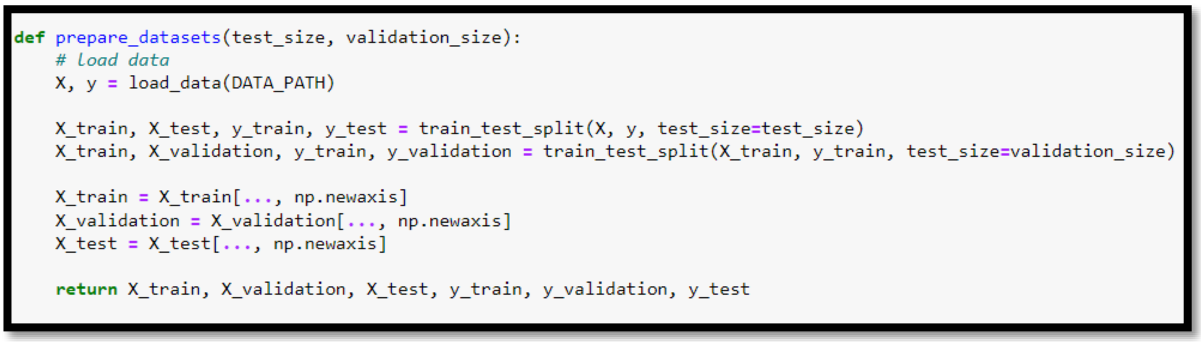
The figure 7.1 shows the libraries json, numpy, sklearn, keras and matplotlib. There is also load_data function where the JSON file is traversed and MFCC and labels are taken for training and testing. The MFCC values are taken in variable X and the labels are taken as y.



```
def plot_history(history):

    fig, axs = plt.subplots(2)

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

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

    plt.show()
```

Figure 7.2: Function to plot graph

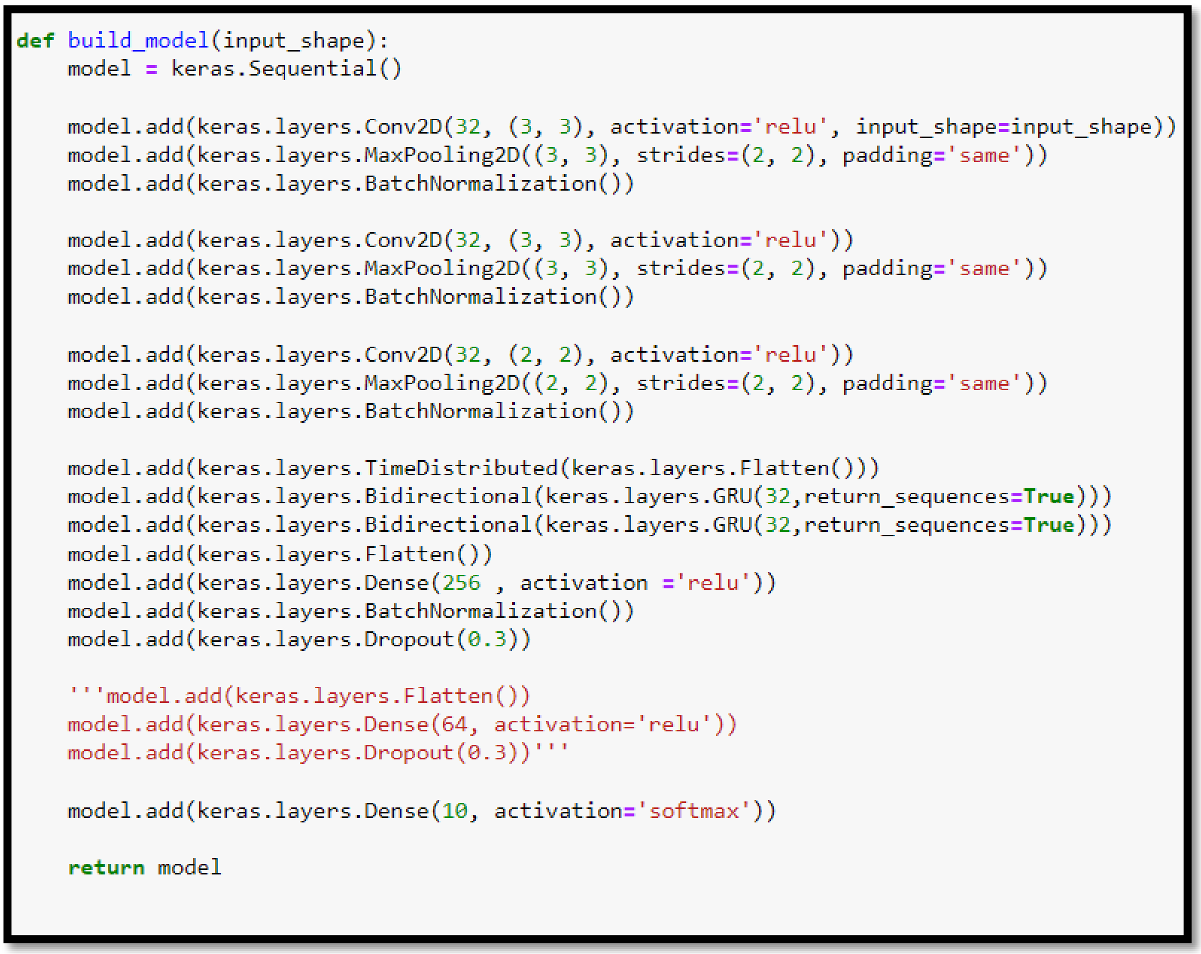
The figure 7.2 shows the function plot_history() which takes the model as the input and creates the graph between Accuracy of testing data and error of testing data as a function of the number of epochs.



```
def prepare_datasets(test_size, validation_size):
    # load data
    X, y = load_data(DATA_PATH)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=validation_size)

    X_train = X_train[..., np.newaxis]
    X_validation = X_validation[..., np.newaxis]
    X_test = X_test[..., np.newaxis]

    return X_train, X_validation, X_test, y_train, y_validation, y_test
```

Figure 7.3: Preparing the dataset

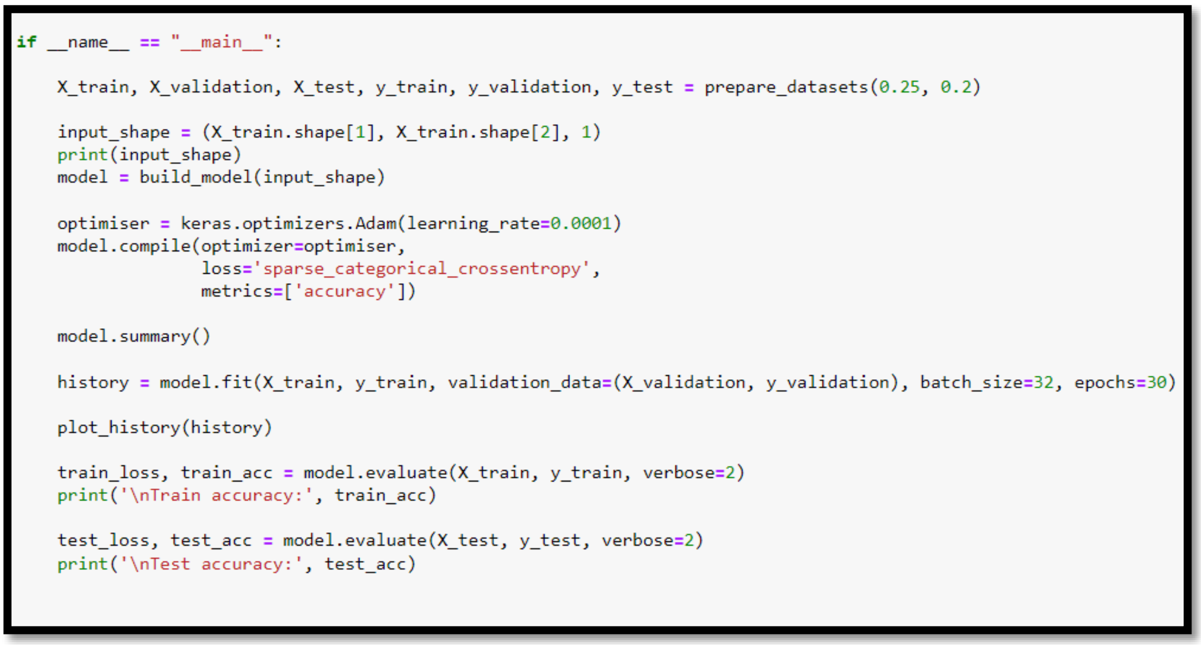
The figure 7.3 contains the function to prepare the dataset. This function takes in the percentage of testing and validation set for splitting the data according to the percentages.



```python
def build_model(input_shape):
    model = keras.Sequential()

    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(keras.layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.Conv2D(32, (2, 2), activation='relu'))
    model.add(keras.layers.MaxPooling2D((2, 2), strides=(2, 2), padding='same'))
    model.add(keras.layers.BatchNormalization())

    model.add(keras.layers.TimeDistributed(keras.layers.Flatten()))
    model.add(keras.layers.Bidirectional(keras.layers.GRU(32,return_sequences=True)))
    model.add(keras.layers.Bidirectional(keras.layers.GRU(32,return_sequences=True)))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(256 , activation ='relu'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Dropout(0.3))

    '''model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.3))'''

    model.add(keras.layers.Dense(10, activation='softmax'))

    return model
```

Figure 7.4: Creating the Neural Network model

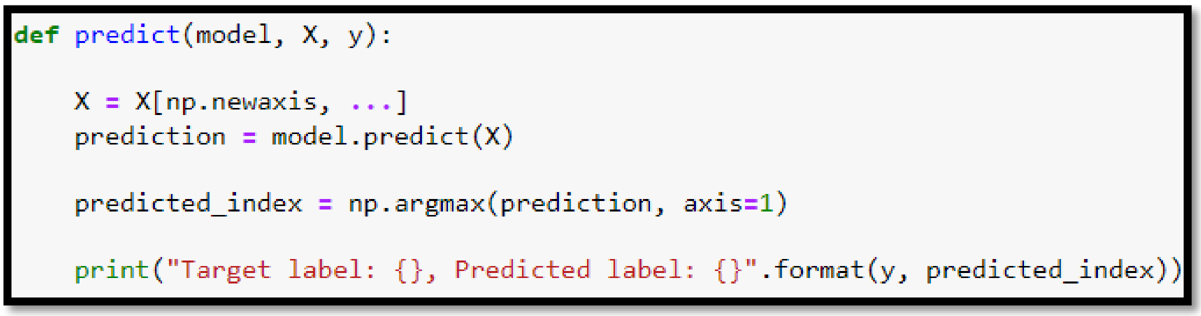
The figure 7.4 gives an overview of the neural networks model that will be used for the training of the data. It contains 3 convolutional layers with 3 3 kernel size and ReLU activation function. Each layer has a batch normalization and a max pool layer. Following that are the GRU layers each being bidirectional and having batch size of 32. The dropout used for solving over fitting is 0.3.



```
if __name__ == "__main__":

    X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.25, 0.2)

    input_shape = (X_train.shape[1], X_train.shape[2], 1)
    print(input_shape)
    model = build_model(input_shape)

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

    model.summary()

    history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32, epochs=30)

    plot_history(history)

    train_loss, train_acc = model.evaluate(X_train, y_train, verbose=2)
    print('\nTrain accuracy:', train_acc)

    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
    print('\nTest accuracy:', test_acc)
```

Figure 7.5: Main Function

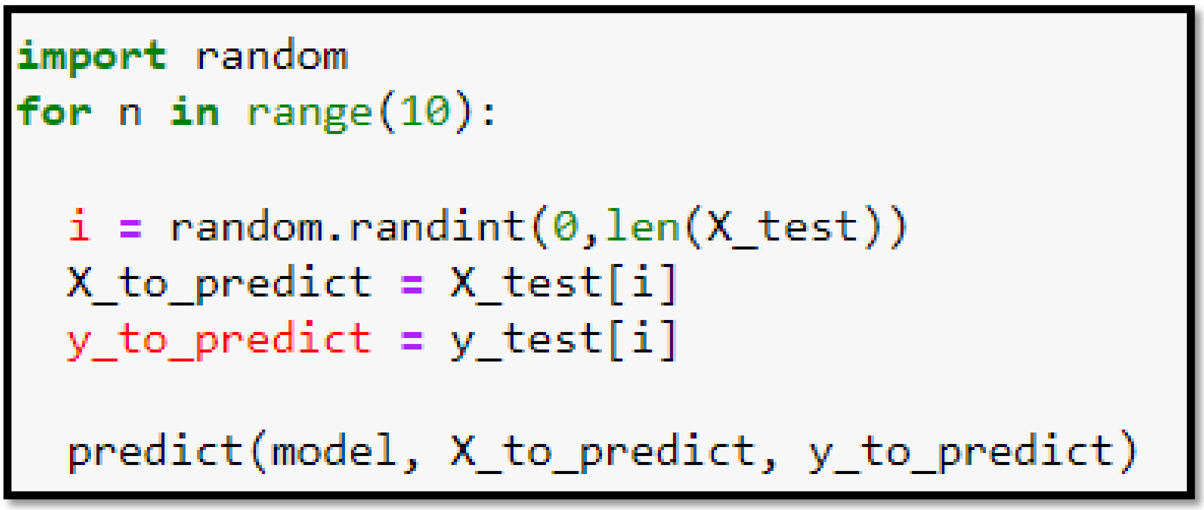
The figure 7.5 is the main function. This is the part of the code where we provide inputs for splitting the data and training the model. Here we specify the testing and validation size, which in turn specifies the training data size.



```
def predict(model, X, y):

    X = X[np.newaxis, ...]
    prediction = model.predict(X)

    predicted_index = np.argmax(prediction, axis=1)

    print("Target label: {}, Predicted label: {}".format(y, predicted_index))
```

Figure 7.6: Function to predict result

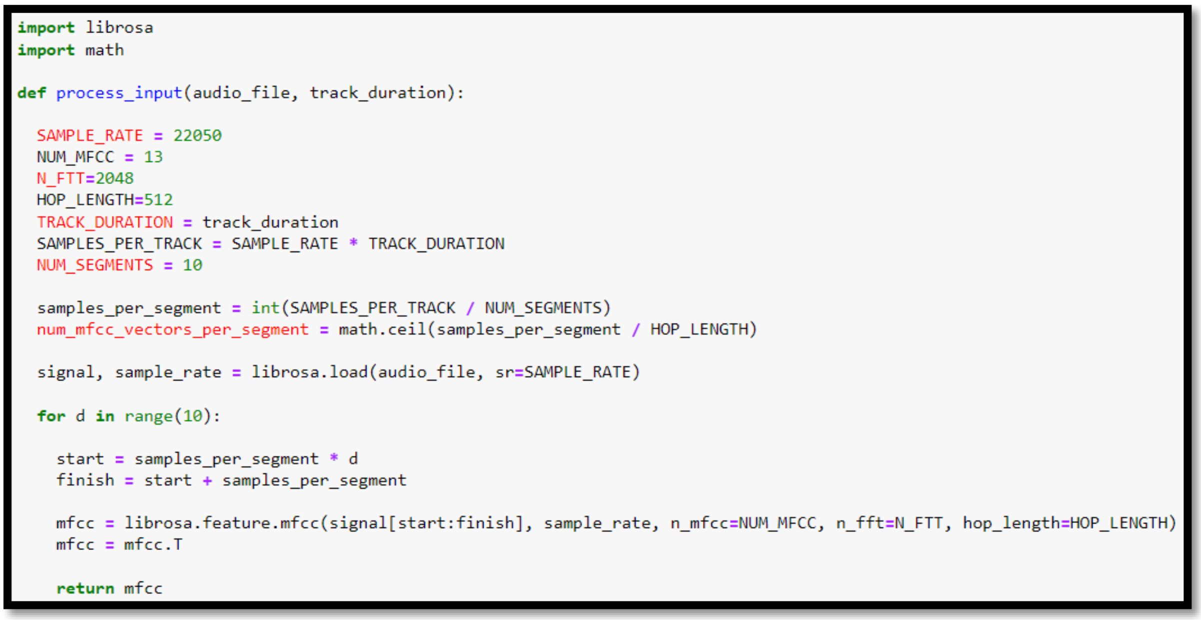
The figure 7.6 shows the predict function. This function is used to take the MFCC features as input for the function and returns the label as an output which can be mapped to the genres to make a prediction.



```python
import random
for n in range(10):

  i = random.randint(0,len(X_test))
  X_to_predict = X_test[i]
  y_to_predict = y_test[i]

  predict(model, X_to_predict, y_to_predict)
```

Figure 7.7: Testing on random songs

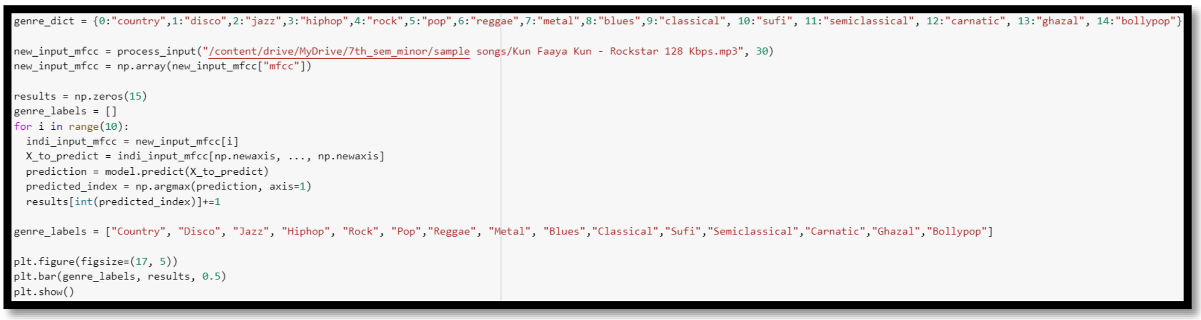
The figure 7.7 show the program taking any 10 random songs and feeding it to the predict model discussed in figure 7.6. This program will print the predicted label and the actual label which can be used as a comparison for how well this model performs with 10 random songs.



```python
import librosa
import math

def process_input(audio_file, track_duration):

  SAMPLE_RATE = 22050
  NUM_MFCC = 13
  N_FTT=2048
  HOP_LENGTH=512
  TRACK_DURATION = track_duration
  SAMPLES_PER_TRACK = SAMPLE_RATE * TRACK_DURATION
  NUM_SEGMENTS = 10

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

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

  for d in range(10):

    start = samples_per_segment * d
    finish = start + samples_per_segment

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

    return mfcc
```

Figure 7.8: Extracting features of unknown song

The figure 7.8 shows the function process_input. This function is the starting stage for predicting songs that are unknown to the program. It takes the unknown song’s address on the system which it gets from the user and it extracts the MFCC features from these songs and returns them for prediction.

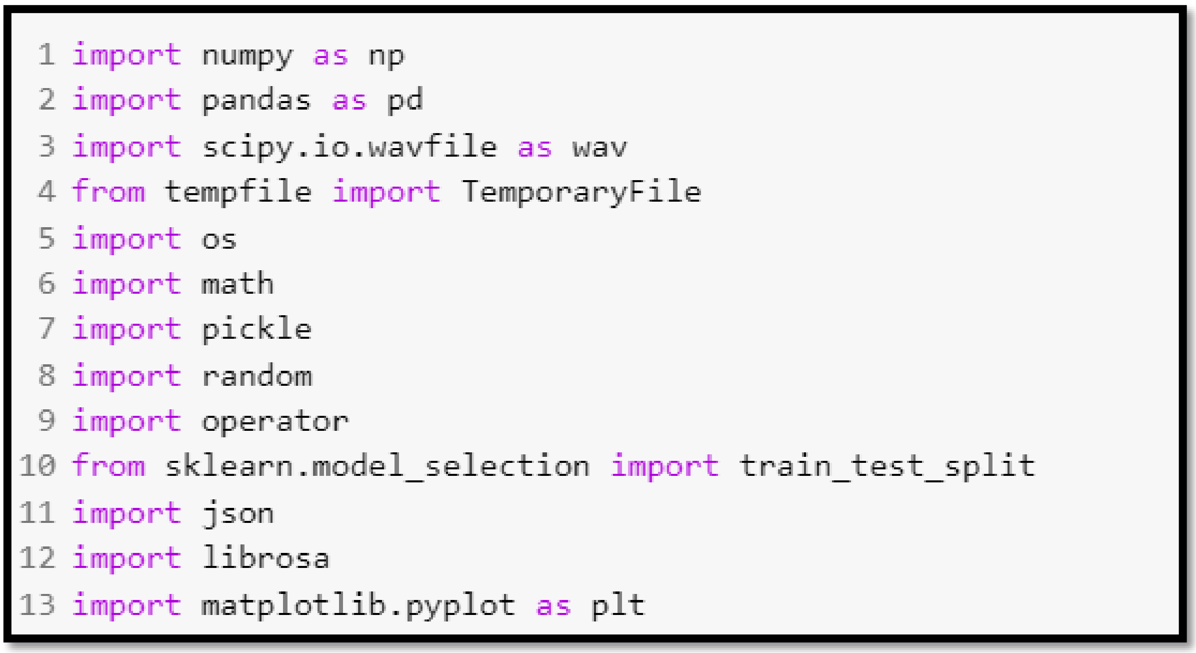


```
genre_dict = {0:"country",1:"disco",2:"jazz",3:"hiphop",4:"rock",5:"pop",6:"reggae",7:"metal",8:"blues",9:"classical", 10:"sufi", 11:"semiclassical", 12:"carnatic", 13:"ghazal", 14:"bollypop"}

new_input_mfcc = process_input("/content/drive/MyDrive/7th_sem_minor/sample songs/Kun Faaya Kun - Rockstar 128 Kbps.mp3", 30)
new_input_mfcc = np.array(new_input_mfcc["mfcc"])

results = np.zeros(15)
genre_labels = []
for i in range(10):
  indi_input_mfcc = new_input_mfcc[i]
  X_to_predict = indi_input_mfcc[np.newaxis, ..., np.newaxis]
  prediction = model.predict(X_to_predict)
  predicted_index = np.argmax(prediction, axis=1)
  results[int(predicted_index)]+=1

genre_labels = ["Country", "Disco", "Jazz", "Hiphop", "Rock", "Pop","Reggae", "Metal", "Blues","Classical","Sufi","Semiclassical","Carnatic","Ghazal","Bollypop"]

plt.figure(figsize=(17, 5))
plt.bar(genre_labels, results, 0.5)
plt.show()
```

Figure 7.9: Predicting unknown songs

The figure 7.9 contains the genre dictionary with the genres and labels being mapped as key-value pairs. This is the part of the program where the user adds the location of the unknown song and passes it to the function process_input for extracting MFCC. The returned MFCC is used to map labels to the genres and displaying the prediction.

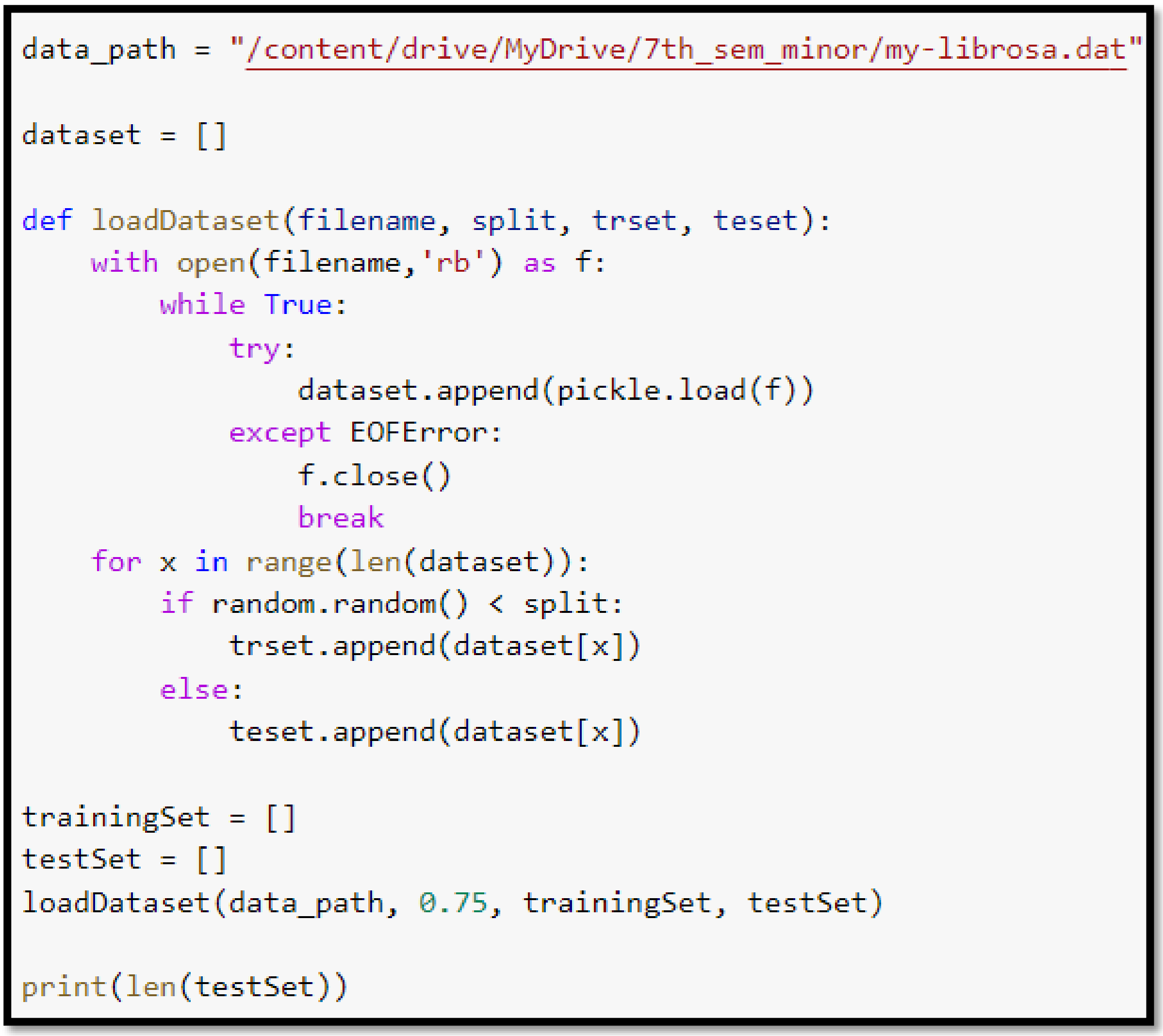
USING FUZZY K-NEAREST NEIGHBORS MODEL TO MAKE SONG PREDICTIONS:



```python
import numpy as np
import pandas as pd
import scipy.io.wavfile as wav
from tempfile import TemporaryFile
import os
import math
import pickle
import random
import operator
from sklearn.model_selection import train_test_split
import json
import librosa
import matplotlib.pyplot as plt
```

Figure 7.10: Importing the libraries

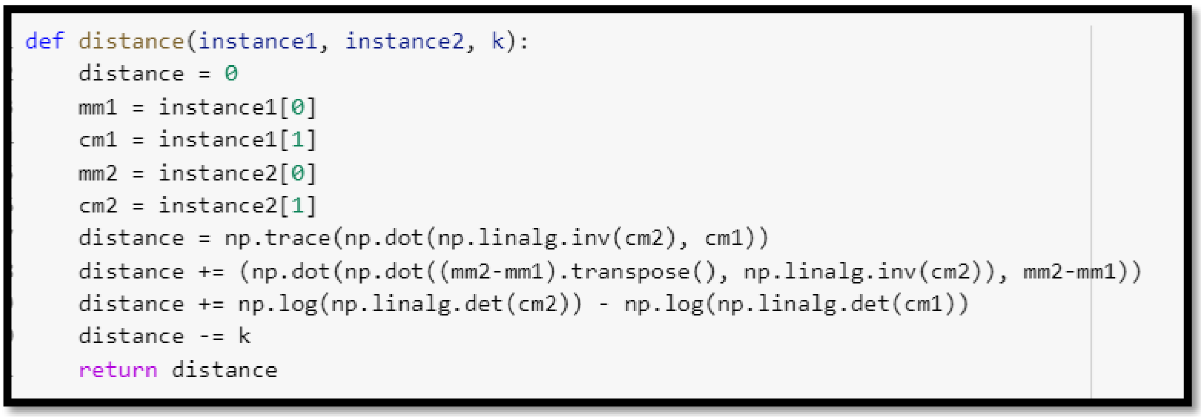
The figure 7.10 contains the various libraries that are implemented for the working of the Fuzzy KNN model.



```
data_path = "/content/drive/MyDrive/7th_sem_minor/my-librosa.dat"

dataset = []

def loadDataset(filename, split, trset, teset):
    with open(filename,'rb') as f:
        while True:
            try:
                dataset.append(pickle.load(f))
            except EOFError:
                f.close()
                break
    for x in range(len(dataset)):
        if random.random() < split:
            trset.append(dataset[x])
        else:
            teset.append(dataset[x])

trainingSet = []
testSet = []
loadDataset(data_path, 0.75, trainingSet, testSet)

print(len(testSet))
```

Figure 7.11: The function to split the dataset

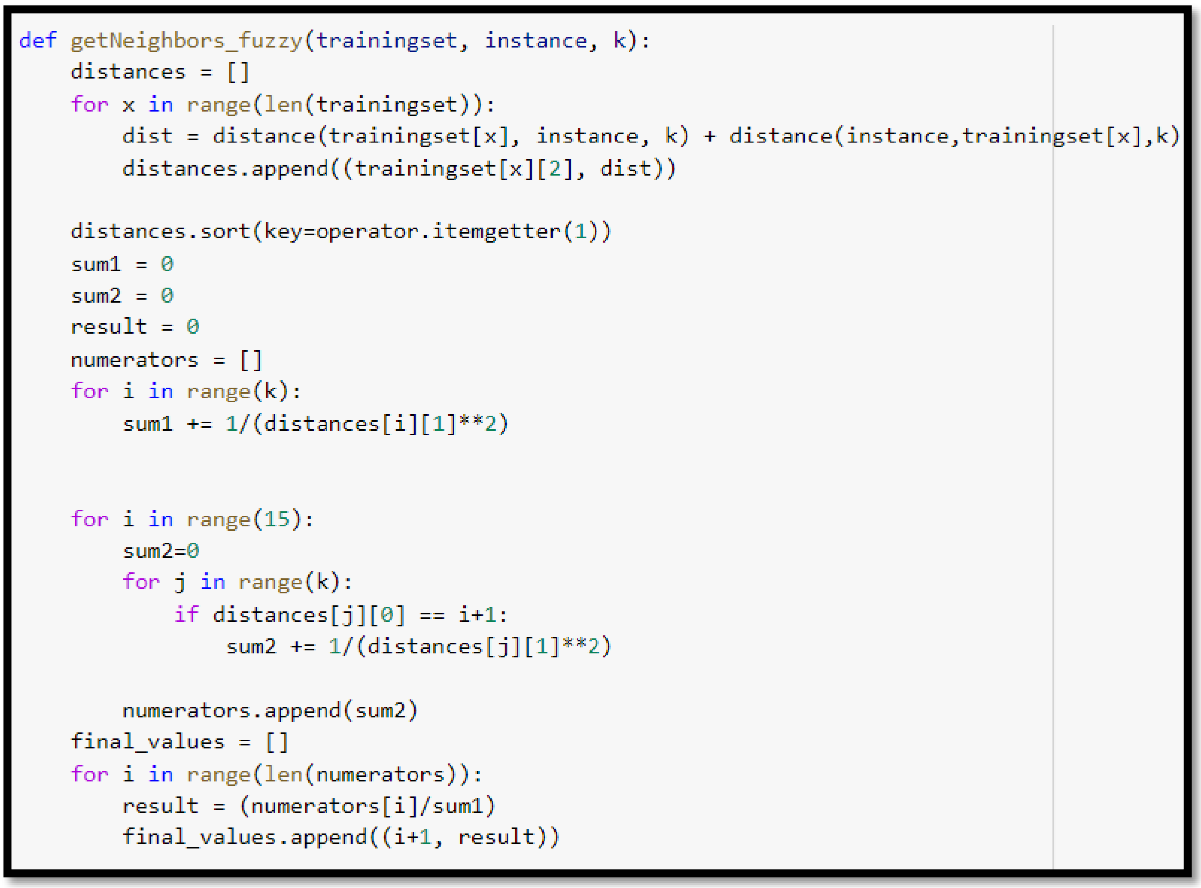
The figure 7.11 displays the code that splits the dataset used into training and testing data. The directory of the dataset is accessed and each file is traversed and split randomly into training and testing data.



```python
def distance(instance1, instance2, k):
    distance = 0
    mm1 = instance1[0]
    cm1 = instance1[1]
    mm2 = instance2[0]
    cm2 = instance2[1]
    distance = np.trace(np.dot(np.linalg.inv(cm2), cm1))
    distance += (np.dot(np.dot((mm2-mm1).transpose(), np.linalg.inv(cm2)), mm2-mm1))
    distance += np.log(np.linalg.det(cm2)) - np.log(np.linalg.det(cm1))
    distance -= k
    return distance
```

Figure 7.12: The function to calculate the distance

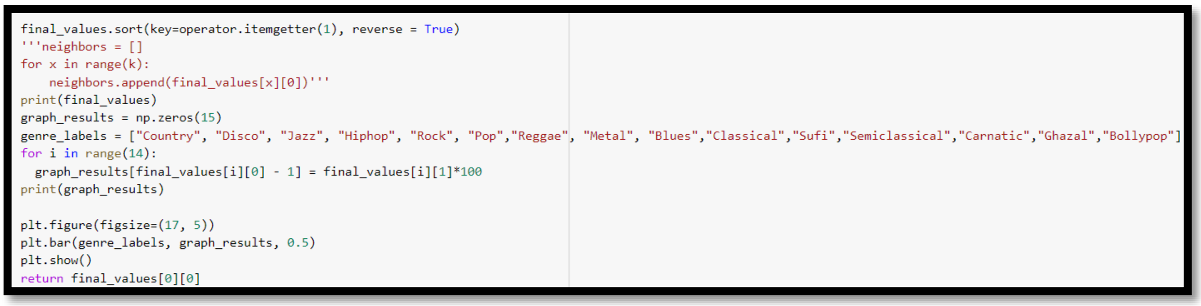
The figure 7.12 displays the function that calculates the distance between the individual test data and the training data as a whole during the accuracy prediction stage. During the genre prediction stage this function returns the distance between any input and the distance between all the songs in the dataset.



```python
def getNeighbors_fuzzy(trainingset, instance, k):
    distances = []
    for x in range(len(trainingset)):
        dist = distance(trainingset[x], instance, k) + distance(instance,trainingset[x],k)
        distances.append((trainingset[x][2], dist))

    distances.sort(key=operator.itemgetter(1))
    sum1 = 0
    sum2 = 0
    result = 0
    numerators = []
    for i in range(k):
        sum1 += 1/(distances[i][1]**2)


    for i in range(15):
        sum2=0
        for j in range(k):
            if distances[j][0] == i+1:
                sum2 += 1/(distances[j][1]**2)

        numerators.append(sum2)
    final_values = []
    for i in range(len(numerators)):
        result = (numerators[i]/sum1)
        final_values.append((i+1, result))
```

Figure 7.13: The function to calculate and display the nearest classes after applying fuzzy algorithm

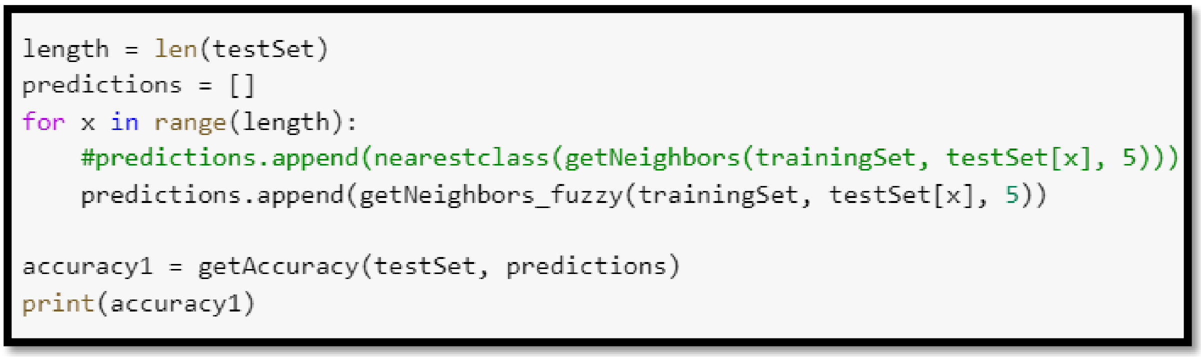
The figure 7.13 displays the function that calls the distance function from figure 7.11 and calculates the distance between testing and training data. During the song prediction stage it calls distance function to calculate the distance between the input and the dataset as a whole. After calling distance function it assigns fuzzy membership to the classes.



```
final_values.sort(key=operator.itemgetter(1), reverse = True)
'''neighbors = []
for x in range(k):
    neighbors.append(final_values[x][0])'''
print(final_values)
graph_results = np.zeros(15)
genre_labels = ["Country", "Disco", "Jazz", "Hiphop", "Rock", "Pop","Reggae", "Metal", "Blues","Classical","Sufi","Semiclassical","Carnatic","Ghazal","Bollypop"]
for i in range(14):
  graph_results[final_values[i][0] - 1] = final_values[i][1]*100
print(graph_results)

plt.figure(figsize=(17, 5))
plt.bar(genre_labels, graph_results, 0.5)
plt.show()
return final_values[0][0]
```

Figure 7.14: The function to calculate and display the nearest classes after applying fuzzy algorithm (continued)

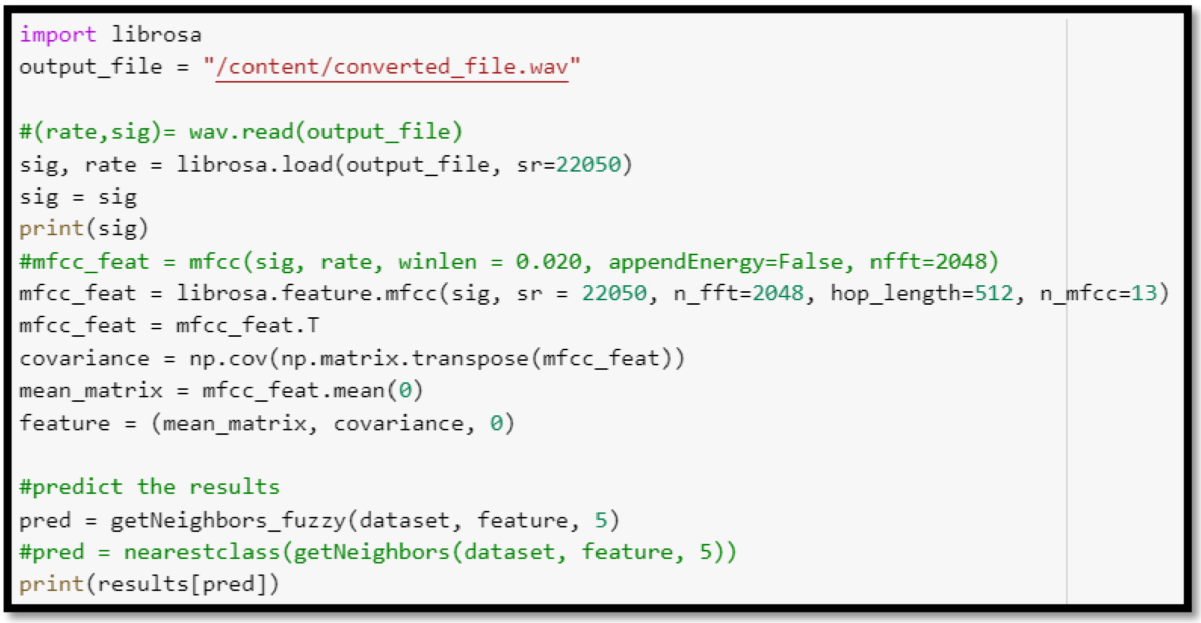
The figure 7.13 displays the function sorting the distances after assigning the fuzzy values to the distances, referred to in figure 7.12. It then selects the 5 greatest values and uses those values to construct the graph for sub-genres, as K is assumed as 5.



```
length = len(testSet)
predictions = []
for x in range(length):
    #predictions.append(nearestclass(getNeighbors(trainingSet, testSet[x], 5)))
    predictions.append(getNeighbors_fuzzy(trainingSet, testSet[x], 5))

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

Figure 7.15: The function to calculate the accuracy

The figure 7.14 displays the function to calculate the accuracy of the Fuzzy KNN algorithm. The algorithm is used to call the getNeighbors_fuzzy function referred to in 7.13. The function calculates the accuracy based on the number of correct predictions the model can make.

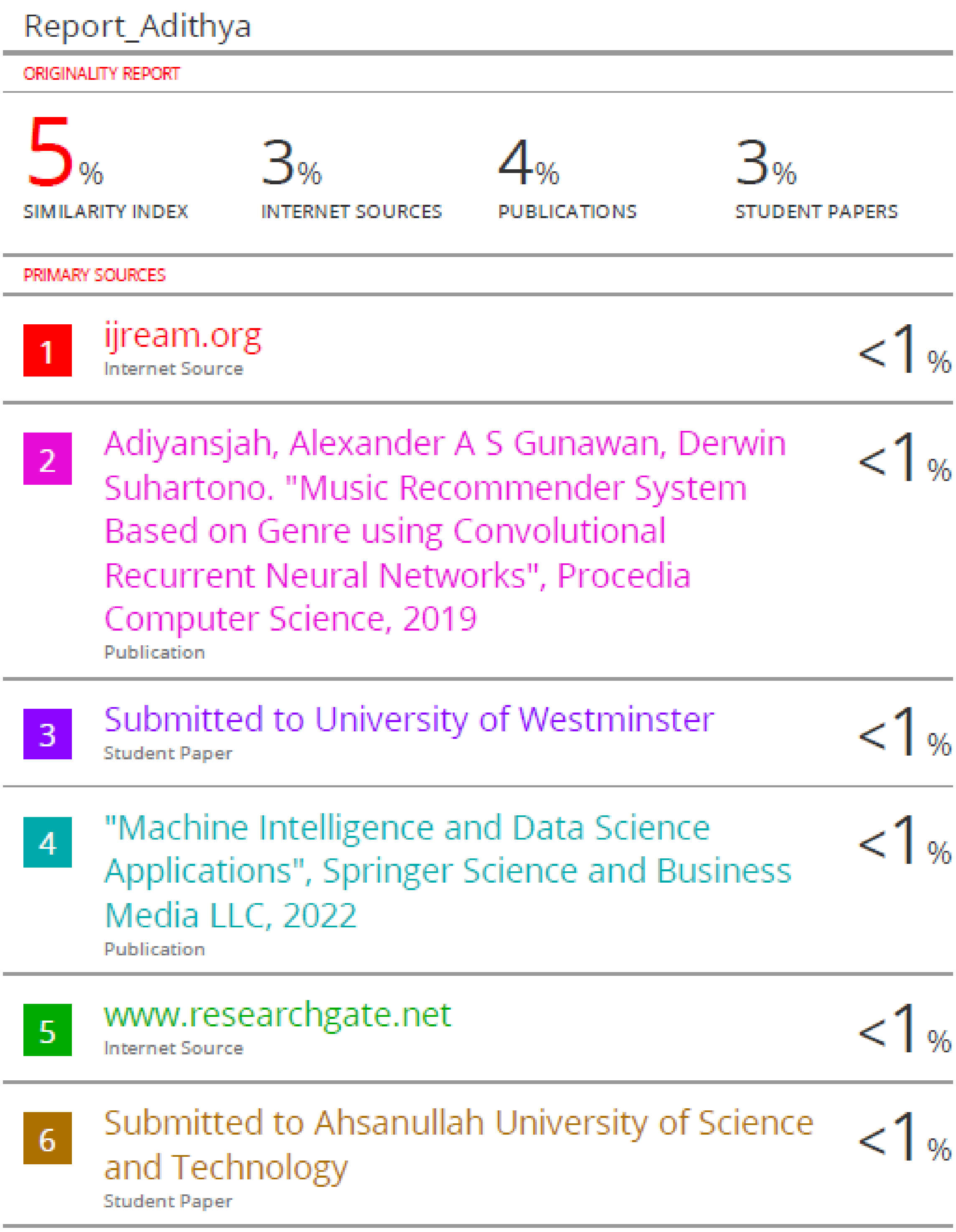


```python
import librosa
output_file = "/content/converted_file.wav"

#(rate,sig)= wav.read(output_file)
sig, rate = librosa.load(output_file, sr=22050)
sig = sig
print(sig)
#mfcc_feat = mfcc(sig, rate, winlen = 0.020, appendEnergy=False, nfft=2048)
mfcc_feat = librosa.feature.mfcc(sig, sr = 22050, n_fft=2048, hop_length=512, n_mfcc=13)
mfcc_feat = mfcc_feat.T
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature = (mean_matrix, covariance, 0)

#predict the results
pred = getNeighbors_fuzzy(dataset, feature, 5)
#pred = nearestclass(getNeighbors(dataset, feature, 5))
print(results[pred])
```

Figure 7.16: The function to Predict genres

The figure 7.16 makes use of the librosa library to extract the input provided through the path link. These are taken as parameters along with the whole dataset of 1500 songs and the K value, which is 5, for the the getNeighbors_fuzzy() function. The function then provides the sub-genres distributed w.r.t their fuzzy values in a graph format.

PLAGIARISM REPORT



Report_Adithya

ORIGINALITY REPORT

5%	3%	4%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ijream.org Internet Source	<1%
2	Adiyansjah, Alexander A S Gunawan, Derwin Suhartono. "Music Recommender System Based on Genre using Convolutional Recurrent Neural Networks", Procedia Computer Science, 2019 Publication	<1%
3	Submitted to University of Westminster Student Paper	<1%
4	"Machine Intelligence and Data Science Applications", Springer Science and Business Media LLC, 2022 Publication	<1%
5	www.researchgate.net Internet Source	<1%
6	Submitted to Ahsanullah University of Science and Technology Student Paper	<1%

7	"Information and Communication Technology for Competitive Strategies (ICTCS 2020)", Springer Science and Business Media LLC, 2022 Publication	<1 %
8	"Proceedings of Second International Conference on Computing, Communications, and Cyber-Security", Springer Science and Business Media LLC, 2021 Publication	<1 %
9	Nikki Pelchat, Craig M Gelowitz. "Neural Network Music Genre Classification", 2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE), 2019 Publication	<1 %
10	pure.tue.nl Internet Source	<1 %
11	Submitted to Piri Reis University Student Paper	<1 %
12	Aditi Mittal, Vipasha Dhiman, Ashi Singh, Chandra Prakash. "Short-Term Bitcoin Price Fluctuation Prediction Using Social Media and Web Search Data", 2019 Twelfth International Conference on Contemporary Computing (IC3), 2019 Publication	<1 %

research.spit.ac.in

13	Internet Source	<1 %
14	tojqi.net Internet Source	<1 %
15	www.sciencegate.app Internet Source	<1 %
16	www.semanticscholar.org Internet Source	<1 %
17	Kai Xu, Dawei Li, Nick Cassimatis, Xiaolong Wang. "LCANet: End-to-End Lipreading with Cascaded Attention-CTC", 2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018), 2018 Publication	<1 %
18	"Emerging Technology in Modelling and Graphics", Springer Science and Business Media LLC, 2020 Publication	<1 %
19	Submitted to Aston University Student Paper	<1 %
20	"Machine Vision Inspection Systems, Volume 2", Wiley, 2021 Publication	<1 %
21	www.mdpi.com Internet Source	<1 %

22	www.ijraset.com Internet Source	<1 %
23	eprints.utar.edu.my Internet Source	<1 %
24	S. Aramuthakannan, M. Ramya Devi, S. Lokesh, R. Manimegalai. "Movie recommendation system via fuzzy decision making based dual deep neural networks", Journal of Intelligent & Fuzzy Systems, 2023 Publication	<1 %
25	essentials.news Internet Source	<1 %
26	Submitted to Associatie K.U.Leuven Student Paper	<1 %
27	mdpi-res.com Internet Source	<1 %
28	medium.com Internet Source	<1 %
29	open.uct.ac.za Internet Source	<1 %
30	www.mecs-press.org Internet Source	<1 %

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

Office of Controller of Examinations

REPORT FOR PLAGIARISM CHECK ON THE SYNOPSIS/THESIS/DISSERTATION/PROJECT REPORTS

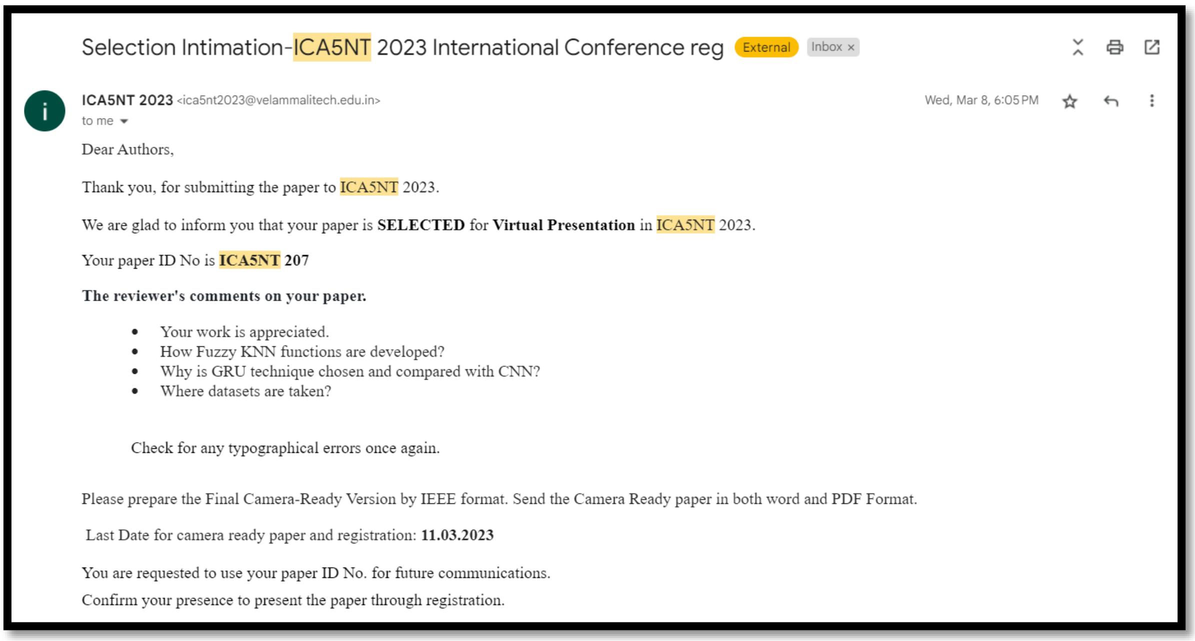
1	**Name of the Candidate(IN BLOCK LETTERS)**	**ABHAY KUMAR PATEL** **ADITHYA S MENON**
2	**Address of the Candidate**	**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR-603203** **Mobile Number :7068253303,6379723689**
3	**Registration Number**	**RA1911029010024 & RA1911029010025**
4	**Date of Birth**	**10/07/2001 & 18/11/2001**
5	**Department**	**DEPARTMENT OF NETWORKING AND COMMUNICATIONS**
6	**Faculty**	**ENGINEERING AND TECHNOLOGY**
7	**Title of the Synopsis/ Thesis/ Dissertation/Project**	**MALWARE ANALYSIS AND DETECTION USING MACHINE LEARNING**
8	**Name and address of the Supervisor / Guide**	**DR. B. BALAKIRUTHIGA** **Mail ID :balakirb@gmail.com** **Mobile Number : 9976767096**
9	**Software Used**	**Turnitin**
10	**Date of Verification**	**01/05/2023**

11	Plagiarism Details: (to attach the final report)			
Chapter	Title of the Chapter	Percentage of similarity index(including self citation)	Percentage of Similarity index (Excluding Self citation)	%of plagiarism after excluding Quotes, Bibliography, etc.,
1	CHAPTER 1	0	1	1
2	CHAPTER 2	0	<1	<1
3	CHAPTER 3	0	1	1
4	CHAPTER 4	0	1	1
5	CHAPTER 5	0	1	1
6	CHAPTER 6	0	<1	<1
7	CHAPTER 7	0	0	0
Thesis abstract		0	0	0
Appendices		0	0	0

I / We declare that the above information has been verified and found true to the best of my / our knowledge.

[illegible] Abhay Signature of the Candidate	[illegible] Signature of the Supervisor / Guide
Signature of the Co-Supervisor/Co-Guide	[illegible] 16/5/23 Signature of the HOD

PAPER SUBMISSION PROOF



Selection Intimation-ICA5NT 2023 International Conference reg External Inbox ×

ICA5NT 2023 <ica5nt2023@velammalitech.edu.in> Wed, Mar 8, 6:05 PM
to me

Dear Authors,

Thank you, for submitting the paper to ICA5NT 2023.

We are glad to inform you that your paper is **SELECTED** for **Virtual Presentation** in ICA5NT 2023.

Your paper ID No is **ICA5NT 207**

The reviewer's comments on your paper.

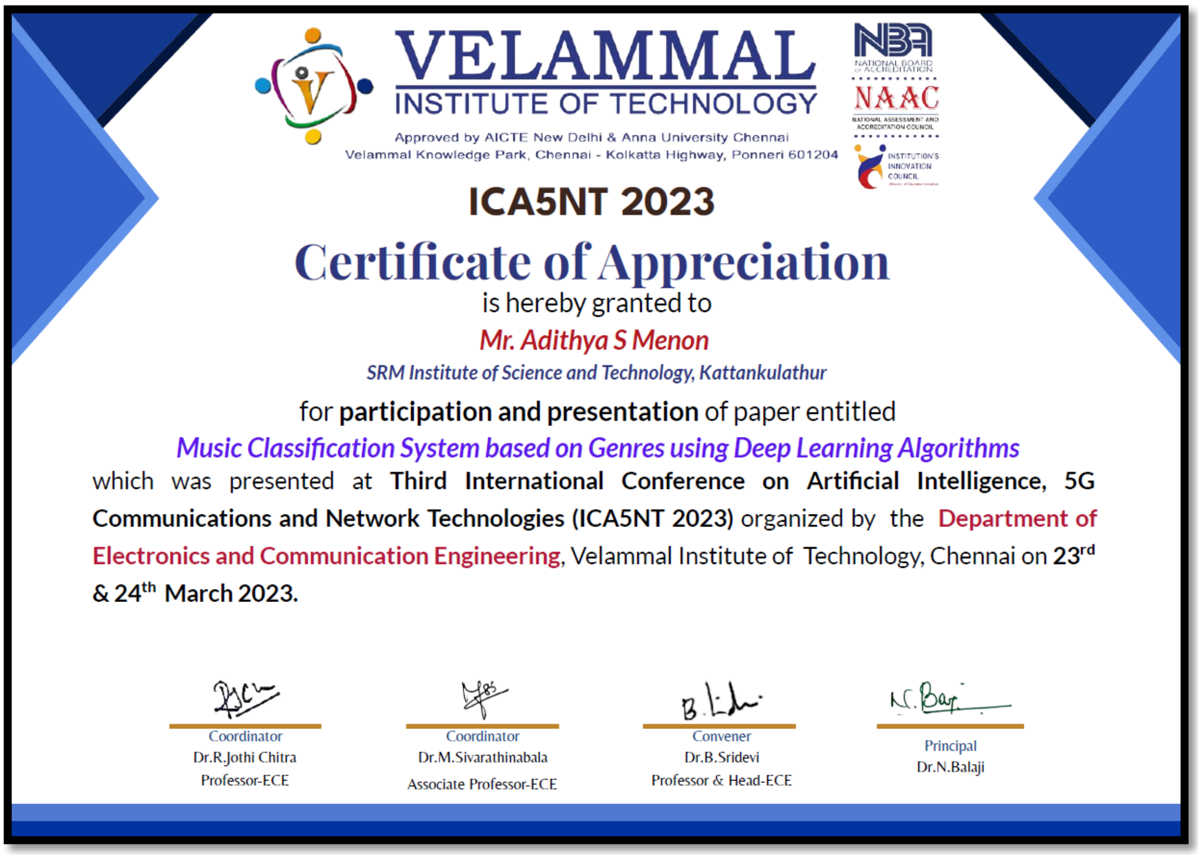
- Your work is appreciated.
- How Fuzzy KNN functions are developed?
- Why is GRU technique chosen and compared with CNN?
- Where datasets are taken?

Check for any typographical errors once again.

Please prepare the Final Camera-Ready Version by IEEE format. Send the Camera Ready paper in both word and PDF Format.

Last Date for camera ready paper and registration: **11.03.2023**

You are requested to use your paper ID No. for future communications.
Confirm your presence to present the paper through registration.



VELAMMAL
INSTITUTE OF TECHNOLOGY
Approved by AICTE New Delhi & Anna University Chennai
Velammal Knowledge Park, Chennai - Kolkatta Highway, Ponneri 601204

ICA5NT 2023

Certificate of Appreciation
is hereby granted to
Mr. Adithya S Menon
SRM Institute of Science and Technology, Kattankulathur

for **participation and presentation** of paper entitled
Music Classification System based on Genres using Deep Learning Algorithms
which was presented at **Third International Conference on Artificial Intelligence, 5G Communications and Network Technologies (ICA5NT 2023)** organized by the **Department of Electronics and Communication Engineering**, Velammal Institute of Technology, Chennai on 23rd & 24th **March 2023.**

Coordinator	Coordinator	Convener	Principal
Dr.R.Jothi Chitra	Dr.M.Sivarathinabala	Dr.B.Sridevi	Dr.N.Balaji
Professor-ECE	Associate Professor-ECE	Professor & Head-ECE	