Project Report

On

Web Based Music Genre Classification

Submitted by: Raj Mishra 19BTCSE026

Department of Computer Science & Engineering

Sam Higginbottom University of Agriculture, Technology & Sciences

(2023)

1. INTRODUCTION

Music is a universal language that has the power to evoke emotions, memories, and moods. With the increasing availability of digital music libraries, there is a growing need for efficient and accurate music genre classification systems. Music genre classification is the task of automatically identifying the genre of a given piece of music based on its audio features. It has important applications in music recommendation systems, music search engines, and music streaming services. In recent years, machine learning techniques have been widely adopted for music genre classification, with promising results. In this project, we propose a method for classifying music into 10 different genres, including rock, pop, classical, country, jazz, hip-hop, disco, reggae, blues, and metal. Our approach utilizes a combination of feature extraction techniques and machine learning algorithms, and we evaluate our method on a publicly available dataset. Our results show that our proposed method can achieve high accuracy in music genre classification, which can be used to improve the performance of existing music recommendation systems and enhance the user experience in music streaming services.

1.1 PURPOSE

The purpose of this project is to develop an effective and accurate method for music genre classification. We aim to classify music into ten different genres using a combination of feature extraction techniques and machine learning algorithms. Our method has the potential to improve the performance of music recommendation systems and enhance the user experience in music streaming services. By accurately identifying the genre of a given piece of music, we can provide users with more personalized music recommendations and enable them to discover new music that aligns with their preferences. Additionally, our proposed method can be applied to a variety of other applications, such as music search engines and audio-based content analysis. Overall, our project has the potential to advance the field of music information retrieval and contribute to the development of more intelligent and efficient music systems.

1.2 OBJECTIVE

The main objectives of the system include:

Accurate Genre Classification: Develop and implement a robust classification algorithm that can accurately identify the genre of audio files into following genres: Blues, Classical, Country, Disco, Hiphop, Metal, Pop, jazz, Reggae, and Rock.

Real-time Processing: Enable the system to process audio files in real-time, ensuring minimal latency and providing users with instant genre classification results.

User-friendly Interface: Design an intuitive and user-friendly interface that allows users to easily upload their audio files, view classification results, and provide feedback on the accuracy of the predictions.

1.3 PRODUCT SCOPE

The product scope of this project is to develop a software system for music genre classification. The system will take an input audio file and output a predicted genre label based on the audio features of the file. The system will be capable of classifying music into ten different genres, including rock, pop, jazz, classical, country, hip-hop, disco, reggae, blues, and metal.

The software system will consist of several components, including audio preprocessing, feature extraction, feature selection, and machine learning models. The audio preprocessing component will be responsible for converting the input audio file into a suitable format for feature extraction. The feature extraction component will extract relevant audio features, such as spectral features, statistical features, and temporal features. The feature selection component will select the most informative features for classification. The machine learning models component will include several models, such as K-nearest Neighbour, MFcc to classify music into different genres.

The software system will be designed to be user-friendly and easily integrated into other applications. It will be implemented in a programming language such as Python and will utilize open-source libraries for audio processing and machine learning. The system will be evaluated on a publicly available dataset and compared against existing state-of-the-art methods for music genre classification.

1.4 USER CLASSES AND CHARACTERISTICS

There are two primary user classes for the music genre classification system: end-users and developers.

1. **End-users:** The end-users are individuals who will be using the music genre classification system to discover and listen to music. These users may include **music enthusiasts, casual listeners, and professionals in the music industry**. The end-users are characterized by their musical preferences, which may vary widely based on their age, gender, culture, and personal experiences. The end-users may have different levels of technical proficiency and may require a user-friendly interface that is easy to navigate.
2. **Developers:** Developers are individuals who will be responsible for developing and maintaining the music genre classification system. These users may include **software**

engineers, data scientists, and researchers in the field of music information retrieval. The developers are characterized by their technical expertise and their ability to work with programming languages, audio processing libraries, and machine learning algorithms. The developers may have different levels of experience and may require access to the system's source code, documentation, and technical support.

The music genre classification system will cater to two main user classes, end-users and developers, who have different characteristics and needs. The system will be designed to be accessible and user-friendly for the end-users, while also providing the necessary flexibility and technical capabilities required by the developers. By considering the needs of these two user classes, we can ensure that the music genre classification system is effective, efficient, and meets the requirements of its intended audience.

1.5 OPERATING ENVIRONMENT

The Operating environment required for the music genre classification system will depend on the development requirements of the system. Here is an overview of the software and hardware components required for both environments:

Software Requirement:

Operating System: Windows 10

Code Editor: Visual Studio Code

Integrated Development Environment (IDE): Python IDE version 3.8

Audio Processing Libraries: librosa, pydub, soundfile, or any other preferred audio processing library

Machine Learning Libraries: scikit-learn, TensorFlow, Keras, or any other preferred machine learning library

Web Application Frameworks: Django web application framework

Database Management System: SQLite3

Hardware Requirements:

Processor: Intel Pentium 2.40GHz or equivalent

RAM: 4 GB of RAM or higher

Storage: 100 GB disk space or higher

The specific software and hardware components required will depend on the requirements of the system and the preferences of the developers. The development and deployment environments must be carefully configured to ensure that the system is reliable, efficient, and scalable.

1.6 ASSUMPTIONS AND DEPENDENCIES

Assumptions and dependencies play a crucial role in determining the feasibility and success of the music genre classification system. Here are some key assumptions and dependencies that must be considered during the system's design and implementation:

1. **Availability of Training Data:** The music genre classification system relies heavily on the availability of high-quality training data that accurately reflects the target audience's preferences. The availability and quality of training data can significantly impact the system's accuracy and performance. It is assumed that a suitable dataset is available, and the data can be obtained or collected without any legal or ethical issues.
2. **Audio Quality:** The music genre classification system's accuracy is heavily dependent on the quality of audio input data. It is assumed that the audio input data provided to the system is of sufficient quality, without background noise or distortion, to ensure the best possible accuracy.
3. **Algorithm Selection:** The performance of the music genre classification system is highly dependent on the selection of appropriate audio processing and machine learning algorithms. It is assumed that suitable algorithms are available and can be adapted to the requirements of the system.
4. **User Input:** The music genre classification system's usability is heavily dependent on user input and feedback. It is assumed that the system will receive appropriate user input to improve the system's performance and accuracy.
5. **Computational Resources**: The music genre classification system involves complex audio processing and machine learning algorithms that require significant computational resources. It is assumed that the required computational resources are available, and the system's performance will not be limited by hardware limitations.
6. **Integration with Other Systems:** The music genre classification system may need to be integrated with other systems or applications. It is assumed that suitable integration protocols and interfaces are available and can be adapted to the requirements of the system.

The design and implementation of the music genre classification system depends on several assumptions and dependencies, including the availability and quality of training data, audio quality, algorithm selection, user input, computational resources, integration with other systems, and legal and ethical considerations. Careful consideration of these assumptions and dependencies will ensure that the system is accurate, efficient, scalable, and compliant with relevant regulations and standards.

2. REQUIREMENT SPECIFICATION

Chapter 2 of the music genre classification system project is the requirement specification. This chapter outlines the system's functional and non-functional requirements, which serve as the basis for system design, development, and testing. Here are the details of the requirement specification:

I. Functional Requirements:

Audio Input: The music genre classification system must be able to accept audio input in format, MP3.

Genre Classification: The system must be able to classify audio files into 10 different genres, including rock, pop, classical, hip hop, disco, jazz, reggae, blues, classical and country.

Real-time Processing: The system must be able to process audio input in real-time, with minimal latency and delay.

Result Output: The system must be able to output the genre classification result to the user interface in a clear and concise format.

II. Non-functional Requirements:

Performance: The system must be able to classify audio files accurately and quickly, with a high level of precision and recall.

Scalability: The system must be scalable to handle a large volume of audio input files and user requests without performance degradation.

Usability: The system must be user-friendly and accessible to a wide range of users, including those with different levels of technical proficiency.

Reliability: The system must be reliable and available at all times, with minimal downtime or system failures.

Compatibility: The system must be compatible with a wide range of hardware and software environments, including different operating systems and web browsers.

Maintainability: The system must be maintainable and easy to update and modify as required.

The requirement specification for the music genre classification system outlines the functional and non-functional requirements that the system must meet. These requirements serve as the basis for system design, development, and testing to ensure that the system is accurate, efficient, scalable, and compliant with relevant regulations and standards.

2.1 USER INTERFACES

The user interface (UI) for the music genre classification system is an essential component that enables users to interact with the system, input audio files, and view the classification results. The UI must be user-friendly, intuitive, and responsive, allowing users to navigate the system and understand the classification results easily. Here are some key features of the UI:

1. **Audio Input:** The UI must allow users to upload audio files in format .MP3 and display the uploaded files details, such as the file name.
2. **Genre Classification Results:** The UI must display the genre classification results clearly and concisely, indicating the predicted genre and the corresponding confidence level.
3. **System Status:** The UI must display the system status, indicating whether the system is processing audio files, idle, or experiencing any issues or errors.
4. **Responsive Design:** The UI must be responsive and adapt to different screen sizes and resolutions, ensuring that the system is accessible from a wide range of devices, including desktops, laptops, tablets, and mobile phones.

The user interface for the music genre classification system must be user-friendly, intuitive, and responsive, allowing users to interact with the system, input audio files, and view the classification results easily. The UI must provide users with feedback mechanisms, display system status, allow user management, provide help and support, and be accessible to users with disabilities.

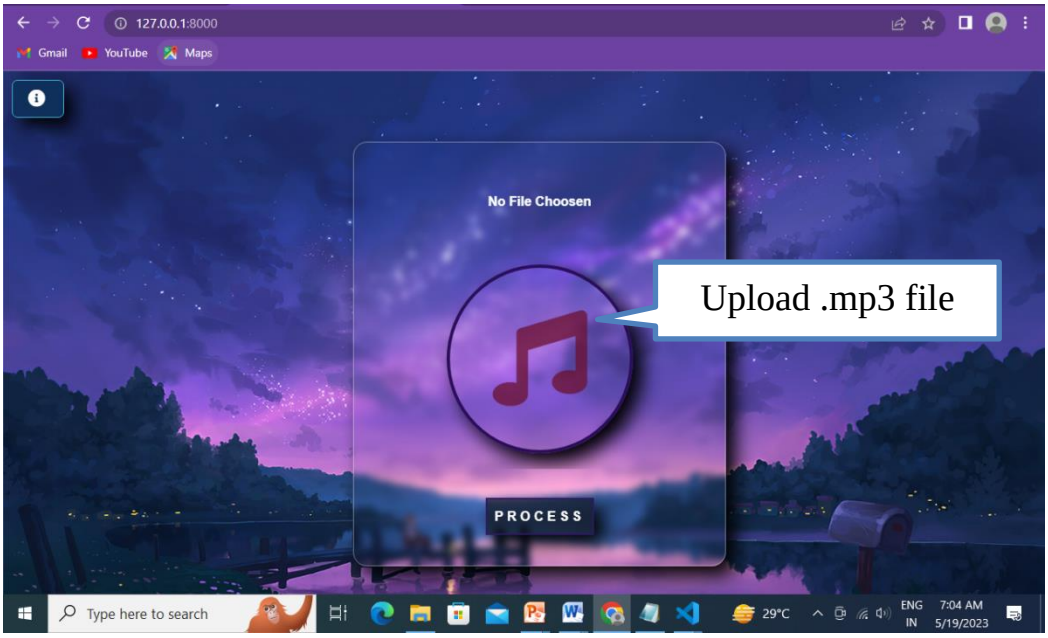

Fig 1.1: Basic UI of the Web-Application

The UI design ensures responsiveness, adapting to different screen sizes and resolutions, making it accessible from various devices, including desktops, laptops, tablets and mobile phones.

2.2 HARDWARE INTERFACE

Audio Input Devices: The system should be compatible with various audio input devices such as microphones, external audio interfaces, or built-in audio input devices of computers or mobile devices.

Display Devices: The system should be able to display the user interface on different types of display devices, including monitors, laptops, tablets, and mobile phones.

2.3 SOFTWARE INTERFACE

1. **Operating System:** The system should be compatible with popular operating systems like Windows, macOS to ensure broad accessibility.
2. **Audio Processing Libraries:** The system may interface with audio processing libraries like librosa, pydub, or soundfile to handle audio file manipulation, feature extraction, and preprocessing tasks.
3. **Machine Learning Libraries:** The system may interface with machine learning libraries such as scikit-learn, TensorFlow, or Keras to implement genre classification algorithms.
4. **Web Technologies:** If the system includes a web-based user interface, it may interface with web technologies like HTML, CSS, and JavaScript, along with web frameworks such as Django or Flask for server-side development.
5. **File System Interface:** The system should have an interface to interact with the file system to access and manage audio files, including reading file metadata and storing classification results.

2.4 COMMUNICATIONS INTERFACES

1. **Web Interface:** If the system includes a web-based user interface, it may utilize HTTP or HTTPS protocols for communication between the client (web browser) and the server hosting the application.
2. **File Transfer:** The system may require communication interfaces for uploading and audio files. This can involve standard protocols such as FTP (File Transfer Protocol) or HTTP-based file transfer mechanisms.

2.5 RELATED WORK

Different genres have distinct musical styles, and the identification of musical styles or musical genres has been extensively researched since its inception. People in other countries

have been using artificial methods to judge music genres and styles since the 1990s. The **"Music Chromosome Project"** is the most well-known. The main goal of this project is for music experts to divide music into different types based on their knowledge and understanding of music technology. However, when faced with massive amounts of data, artificial methods are immature due to the limited technical conditions and different experts have slightly different understandings of different music genres, so the project has spent a lot of financial and material resources. Driven by this situation, people began to try the research of automatic classification algorithms for music genre recognition.

Later, American researchers proposed a classification algorithm. This method mainly calculates the mean, variance, and autocorrelation coefficient from massive music data, so as to further analyze the characteristics of music, such as loudness and pitch, people can easily feel. The obtained features are then identified and classified by using some classifiers. Subsequently, this algorithm has been greatly promoted, and people have begun to try to use some improved algorithms to classify music genres based on this algorithm. **In 2002, Tzanetakis and Cook** provided a new classification algorithm. This method first extracts acoustic features, which mainly contains three types of acoustic features, music timbre, rhythm, and pitch content. Since the extracted acoustic features are generally of higher dimensionality, the feature selection algorithm is used to reduce the dimensionality of the features to facilitate calculation, and at the same time, some insignificant redundant information is removed. Finally, some models and corresponding algorithms are used to identify and classify music genres.

With the development of computer technology, machine learning has also begun to be applied to the classification of music genres**. In 2003, Xu et al. studied the classification of music genres by using different music characteristics.** By comparing the -nearest neighbor method, conditional random field, and Markov model algorithms, they found that the recognition effect classification algorithm of SVM is the most effective. **With the application of wavelet transform theory, Li et al. used statistical methods to calculate the statistical values of wavelet coefficients, combined with classification models commonly used in machine learning, such as LDA, GMM, and KNN, to obtain good classification results. In 2011, in order to obtain more essential music features, Panagakis et al. proposed an unsupervised dimensionality reduction method for the first time.** Through experimental results, it was found that this method has a significant effect in extracting music features compared to previous methods.

3. SYSTEM DESIGN

3.1 DESIGN AND IMPLEMENTATION CONSTRAINTS

The design and implementation of the music genre classification system must take into account several constraints that may impact the system's performance, functionality, and usability. Here are some key design and implementation constraints to consider:

1. **Dataset Availability**: The accuracy of the music genre classification system heavily relies on the availability and quality of the dataset used for training the machine learning algorithms. Access to a comprehensive and diverse dataset can be a significant constraint for the system's design and implementation. The dataset must be representative of the target audience, cover a wide range of music genres, and be adequately labeled with genre information.
2. **Computational Resources:** The music genre classification system involves complex audio processing and machine learning algorithms that require significant computational resources to train and run. The available hardware and software resources may limit the system's performance and scalability. Careful consideration of hardware and software optimization techniques may be required to ensure that the system operates efficiently and effectively.
3. **Algorithm Selection:** The music genre classification system's accuracy is highly dependent on the selection of appropriate audio processing and machine learning algorithms. Careful consideration must be given to algorithm selection and parameter tuning to ensure that the system is accurate and robust.
4. **User Interface Design:** The music genre classification system's usability heavily depends on the user interface design. The interface should be intuitive, user-friendly, and accessible to a wide range of users, including those with different levels of technical proficiency. The interface design may be limited by available software development tools and frameworks.
5. **System Integration:** The music genre classification system may need to be integrated with other systems or applications. Careful consideration must be given to the integration requirements, including data formats, communication protocols, and security measures.
6. **Privacy and Data Protection:** The music genre classification system may process sensitive user data, such as audio files and user preferences. The system must comply with privacy and data protection regulations, including data storage, access, and security measures.

The design and implementation of the music genre classification system must take into account several constraints, including dataset availability, computational resources, algorithm selection, user interface design, and system integration, and privacy and data protection regulations.

Careful consideration of these constraints will ensure that the system is accurate, efficient, scalable, and compliant with relevant regulations and standards.

3.2 ARCHITECTURE OF WEB APPLICATION

The system design for the music genre classification project involves various components and considerations. Here's an overview of the key aspects:

Client-Server Architecture: The system can follow a client-server architecture, where the client interacts with a web-based user interface and the server performs the genre classification.

The front end consists of HTML, JavaScript and, bootstrap. JavaScript used for event handling and animation. Bootstrap is a responsive front-end framework. The front end sends the HTTP requests to the Django backend where it works with the URL's, views and models.

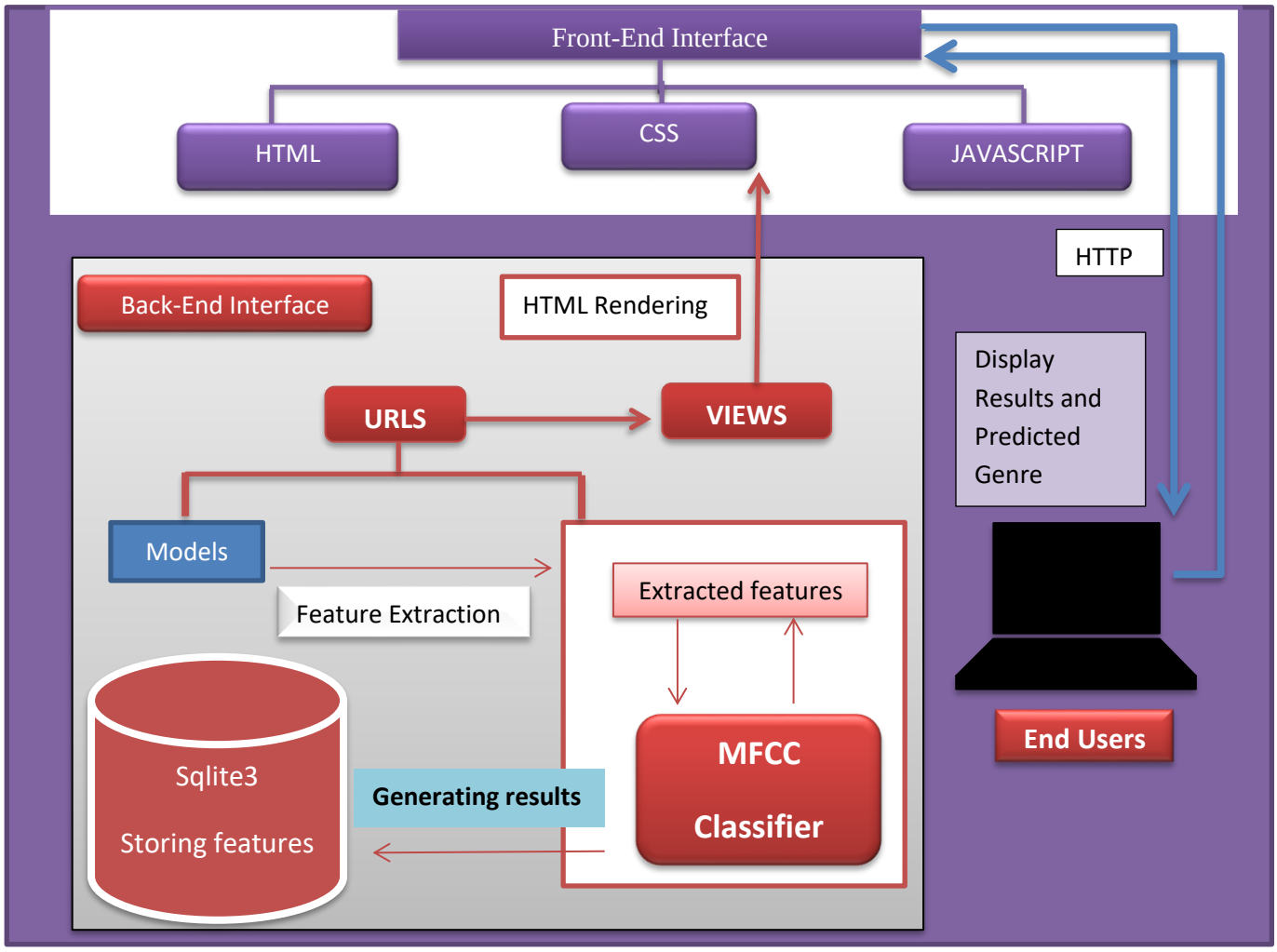

Fig 1.2: System Architecture for Web Application

User Interface:

Web Interface: Develop a user-friendly web interface that allows users to upload audio files, view genre classification results, and access system features. The interface should be responsive and adaptable to different screen sizes.

- *A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication associates between the actors and the use cases and generalization among the use case as shown below in figure.*

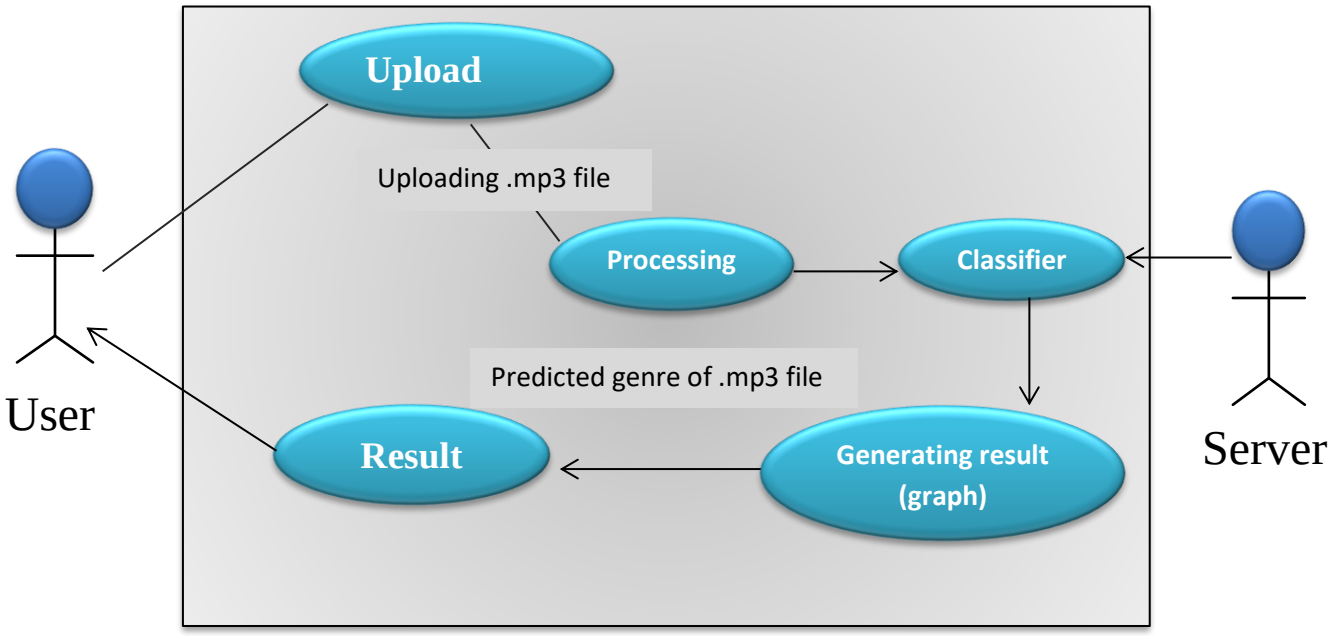

Fig 1.3: Use case diagram for overall system

Backend Components:

File Handling: Implement a file handling module to handle audio file uploads and storage.

Feature Extraction: Develop a feature extraction module using MFCC or other audio feature extraction techniques to extract relevant features from audio files.

Genre Classification Model: Train a machine learning model (e.g., KNN, MFCC, SVM) using a labeled dataset of audio files to classify music genres. The model should be capable of predicting the genre based on the extracted features.

Database Management: If required, integrate a database management system (e.g., SQLite3) to store audio file metadata, and classification results.

Integration and Communication:

APIs: Design and implement APIs to facilitate communication between the user interface, backend components, and the database.

Data Flow: Define the flow of data between different system components, ensuring seamless integration and efficient processing.

Error Handling: Implement error handling mechanisms to handle exceptions and provide informative error messages to users.

Performance and Scalability:

Performance Optimization: Optimize the system components, algorithms, and data structures to achieve efficient and fast genre classification.

Scalability: Design the system to handle a growing number of users and increasing volumes of audio files without compromising performance.

Deployment:

Choose a suitable hosting environment, such as cloud platforms or dedicated servers, to deploy the system.

Configure the necessary server infrastructure, including web servers, application servers, and database servers.

The system design should be modular, allowing for flexibility and future enhancements. Each component should be well-documented, with clear interfaces and dependencies. Regular testing, maintenance, and updates should be performed to ensure the system's reliability and accuracy in music genre classification.

- **Audio File Upload:** The system should allow users to upload audio files in the .MP3 format for genre classification.
- **Genre Classification:** The system should classify the uploaded audio files into specific music genres using machine learning algorithms.
- **Genre Display:** The system should display the predicted genre for each uploaded audio file along with the corresponding confidence level or probability.
- **Error Handling:** The system should have proper error handling mechanisms in place to handle exceptions, errors, or unexpected inputs gracefully and provide informative error messages to users.
- **Scalability:** The system should be designed to handle a growing number of users and increasing volumes of audio files without compromising performance or reliability.

- **Modularity and Extensibility:** The system should be modular and extensible, allowing for easy integration of new machine learning models, audio processing techniques, or additional features in the future.

3.3 TRAINING/TESTING OF THE GTZAN DATASETS

Training phase **Classifier models Includes:**

- **KNN(k nearest neighbor)**
- **SVM(Support Vector Machine)**
- **MFCCs(*Mel-frequency cepstral coefficients)***

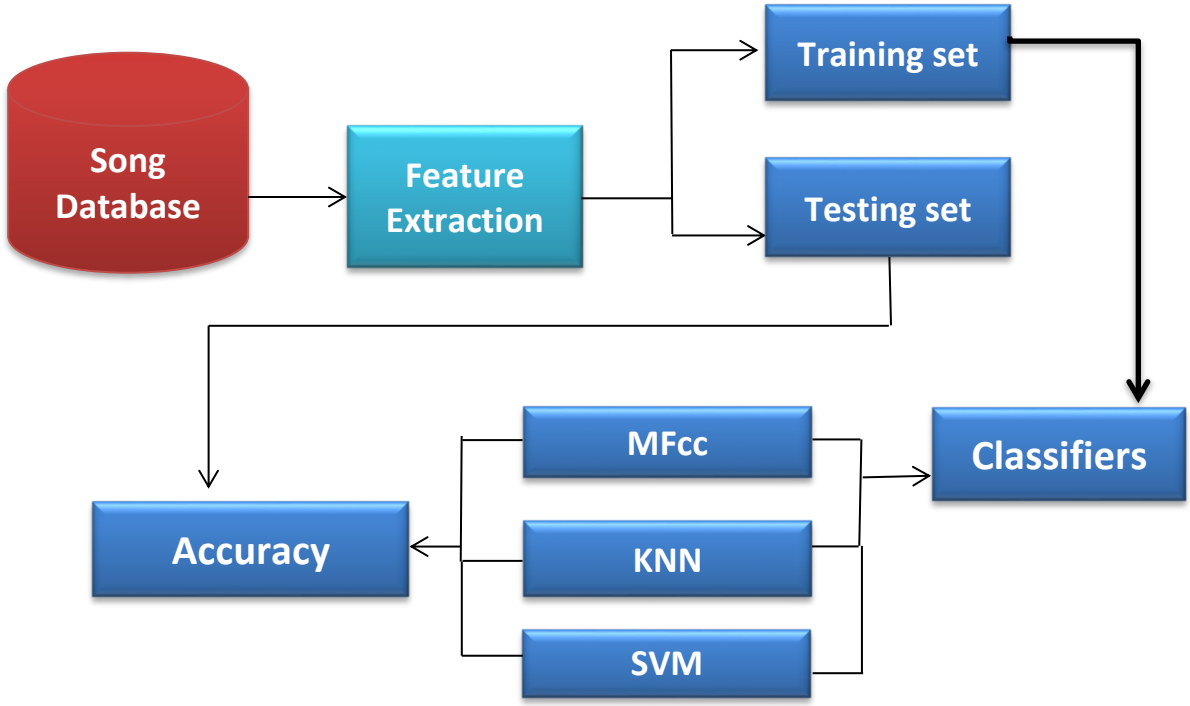

Fig 1.4: Training and Testing Phase Data processing

3.3.1 Dataset

We have used the GTZAN dataset. It contains 10 music genres; each genre has 1000 audio clips in .au format. The genres are - blues, classical, country, disco, pop, reggae, rock, metal, and hip-hop. Each genre contains 100 songs. In the GTZAN dataset, each song is 30 s long, with a 22,050 Hz sample rate, mono mode, AU file format, and 16-bit audio files. The dataset incorporates samples from variety of sources like CDs, radios, microphone recordings etc.

The dataset is divided into 60% as the training dataset, 30% as the test set, and 10% as the validation set. Table 1 shows the relevant information of the GTZAN dataset.

ITEM	DATA
Genre	10
Length	30 seconds each
Sample rate	22,050 Hz
Song format	wav
Total Songs	1000

Table 1. Contents of GTZAN dataset

Music contains three elements: **pitch, rhythm, and timbre.** Melody and harmony of music can be formed through the combination and transformation of pitch; tempo is related to articulation, which controls the speed and transition of music; timbre is the sound quality of sound perception, used to distinguish different types of sounds to produce notes, and each instrument has its own unique timbre.

Feature Extraction

The inputs to the algorithm are collected using 3 features namely MFCC, Pitch and Tempo. Each feature is extracted using Librosa: a python library.

Framing and Windowing: A continuous voice stream is divided into N-sample frames, with M samples between consecutive frames. Spectrum refers to the end outcome of this phase.

Mel Frequency Wrapping: Any tone with a frequency of f has its pitch evaluated on the Mel scale. This scale uses linear spacing for frequencies under 1000Hz and a logarithmic scale for frequencies beyond 1000Hz.

Cepstrum: Time conversion from log-Mel scale. This yields MFCC features, which are an excellent representation of a signal's local spectral qualities.

Preprocessing

The pre-processing part involved converting the audio from .mp3 or any format to .wav format to make it compatible to python’s wave module for reading audio files.

3.3.2 Extracted features

Musical characteristics: what are the features that we can measure and base our classifications on? The accuracy of music genre classification is highly reliant on proper feature extraction.

Tzanetakis and Cook considered: spectral centroid, spectral rolloff, spectral flux, time domain zero crossings, Mel frequency ceptral coefficients, analysis window, texture window and low-energy feature in their 2002 paper. While this research did not involve machine learning, later research in 2018 still considers many of these features. These features are extracted by applying the Fourier transform on an audio file, like .mp3 or .wav. From this Fourier transform you can also visualize the music by extracting the Mel-frequency cepstral coefficients and plotting sound frequencies on a y-axis and time on the x-axis with sound intensity displayed on a third axis represented by a colored heatmap.

Below mentioned are the features extracted to distinguish between different genre and accurately predict the genre of given audio file, these features are stored in **feature_30_sec.csv** file.

Filename: This column contains the names of the audio files in the dataset. Each row represents a different audio file.

• **Length:** This column represents the length of each audio file in samples. It indicates the number of samples present in the audio file.

• **chroma_stft_mean:** Chroma feature is a representation of the pitch content of an audio signal. This column represents the mean value of the chroma feature computed using the short-time Fourier transform (STFT) for each audio file.

• **chroma_stft_var:** This column represents the variance of the chroma feature. It indicates the spread or variation in the chroma values across different time segments of an audio file.

• **rms_mean:** Root Mean Square (RMS) is a measure of the average energy of an audio signal. This column represents the mean value of the RMS energy for each audio file.

• **rms_var:** This column represents the variance of the RMS energy. It indicates the variation in the RMS energy values across different time segments of an audio file.

• **spectral_centroid_mean:** Spectral centroid represents the "center of mass" of the spectrum and provides information about the brightness of an audio signal. This column represents the mean value of the spectral centroid for each audio file.

• **spectral_centroid_var:** This column represents the variance of the spectral centroid. It indicates the spread or variation in the spectral centroid values across different time segments of an audio file.

• **spectral_bandwidth_mean:** Spectral bandwidth measures the spread of frequencies around the spectral centroid. This column represents the mean value of the spectral bandwidth for each audio file.

• **spectral_bandwidth_var:** This column represents the variance of the spectral bandwidth. It indicates the spread or variation in the spectral bandwidth values across different time segments of an audio file.

• **rolloff_mean:** Spectral rolloff is a measure of the frequency below which a specified percentage of the total spectral energy is concentrated. This column represents the mean value of the spectral rolloff for each audio file.

• **rolloff_var:** This column represents the variance of the spectral rolloff. It indicates the spread or variation in the spectral rolloff values across different time segments of an audio file.

• **zero_crossing_rate_mean:** Zero-crossing rate is a measure of the rate at which the audio signal changes its sign. This column represents the mean value of the zero-crossing rate for each audio file.

• **zero_crossing_rate_var:** This column represents the variance of the zero-crossing rate. It indicates the spread or variation in the zero-crossing rate values across different time segments of an audio file.

• **harmony_mean**: This column represents the mean value of the harmony feature. Harmony is related to the perceived tonal quality of an audio signal.

• **harmony_var:** This column represents the variance of the harmony feature. It indicates the spread or variation in the harmony values across different time segments of an audio file.

• **perceptr_mean:** This column represents the mean value of the perceptual feature. Perceptual features capture subjective perceptual aspects of audio signals.

• **perceptr_var:** This column represents the variance of the perceptual feature. It indicates the spread or variation in the perceptual feature values across different time segments of an audio file.

• **tempo:** Tempo represents the speed or pace of the audio file, measured in beats per minute (BPM).

• **mfcc1_mean to mfcc20_var:** Mel-frequency cepstral coefficients (MFCCs) are widely used in audio signal processing for capturing the timbral characteristics of an audio signal. These columns represent the mean and variance values of the 20 MFCCs for each audio file.

• **Label:** This column represents the genre label of each audio file. It specifies the genre to which the audio file belongs, such as "blues," "rock," "jazz," etc

MODELS	**ACCURACY**
K-nearest Neighbour	**42%**
Support Vector Machines	**31%**

Mel Frequency Cepstral Coefficients	**70%**

Table 2. Model Accuracy on Trained Datasets

3.4 MODEL SELECTION

Selected Model: MFCC (Mel-Frequency Cepstral Coefficients) Workflow:

MFCC (Mel-Frequency Cepstral Coefficients) is a widely used technique for feature extraction in audio signal processing. It provides a compact representation of the spectral characteristics of an audio signal, making it suitable for various applications such as speech recognition, music genre classification, and audio event detection.

Preprocessing:

Load the audio signal: Read the audio file into memory.

Resampling: If necessary, resample the audio signal to a consistent sample rate.

Frame Segmentation: Divide the audio signal into frames of equal duration using a sliding window.

Windowing: Apply a window function (e.g., Hamming window) to each frame to reduce spectral leakage.

Fourier Transform:

Apply a Fast Fourier Transform (FFT) to convert each frame from the time domain to the frequency domain.

Obtain the power spectrum by calculating the squared magnitude of the complex FFT coefficients.

Mel Filterbank:

Construct a Mel filterbank with a specified number of triangular filters.

Apply the Mel filterbank to the power spectrum to obtain the filterbank energies.

Take the logarithm of the filterbank energies to obtain the log filterbank energies.

Discrete Cosine Transform (DCT):

Apply a Discrete Cosine Transform (DCT) to decorrelate the log filterbank energies.

Retain a subset of the resulting DCT coefficients, typically the lower-frequency coefficients, as MFCCs.

Mean Normalization:

Calculate the mean value of each MFCC coefficient across all frames.

Subtract the mean from each MFCC coefficient to normalize them.

Optional Steps:

Delta and Delta-Delta Features: Compute the first and second derivatives of the MFCCs to capture temporal dynamics.

Feature Concatenation: Concatenate the MFCCs, delta features, and delta-delta features to form a feature vector with increased dimensionality.

Feature Extraction:

Obtain a feature matrix where each row represents the feature vectors extracted from each frame of the audio signal.

The resulting feature matrix, typically consisting of MFCCs and optionally delta and delta-delta features, can be used as input to machine learning algorithms for tasks such as music genre classification, speech recognition, or audio event detection. The MFCC workflow is a common and effective method for extracting compact and discriminative representations of audio signals in various audio processing applications.

Data Flow Diagram for the Classifier Model

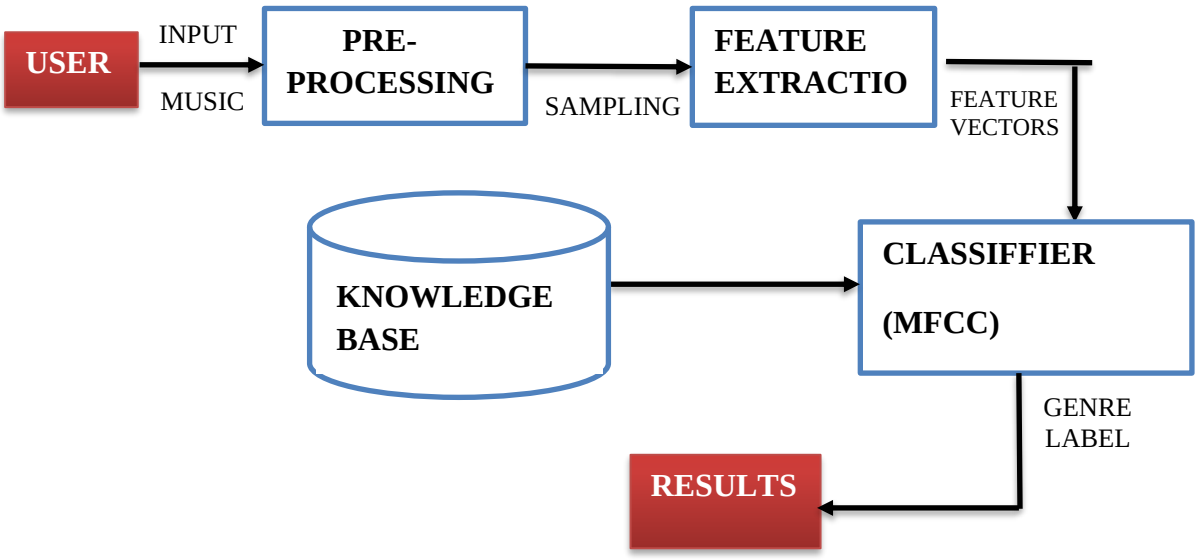

Fig 1.5: Data Flow diagram of MFCC Classifier

The above fig 1.5 shows the data flow diagram for the overall system of music genre classification. We have used the GTZAN dataset. It contains 10 music genres; each genre has 1000 audio clips in .au format. The genres are - blues, classical, country, disco, pop, jazz, reggae, rock, metal, hip-hop. Each audio clip has a length 30 seconds. Then the pre-processing part involved converting the audio from .au format to .wav format to make it compatible to python's wave module for reading audio files.

To classify our audio clips, we chose features like Mel-Frequency Cepstral Coefficients, and also reduce the feature dimensions then, we used different multi-class classifiers and an ensemble of these to obtain our results as a genre label.

MFCC (Mel-frequency Cepstral Coefficients) is used as a feature extraction technique to represent the audio signal in the form of a Mel spectrogram. Let's understand what MFCC is doing in the code:

Mel Spectrogram:

The code starts with an audio file, typically in the WAV format.

The audio file is loaded and processed using the librosa library.

The librosa library is used to compute the mel spectrogram, which represents the frequency content of the audio signal over time.

The mel spectrogram is a 2D matrix where the horizontal axis represents time and the vertical axis represents frequency.

MFCC Calculation:

- The MFCC computation involves several steps:
- The mel spectrogram is divided into overlapping frames of a fixed duration.
- Each frame is transformed to the frequency domain using techniques like the Fourier Transform.
- A filterbank is applied to the frequency domain representation to mimic the human auditory system's response to different frequencies.
- The logarithm of the filterbank energies is computed to represent the amplitudes in a more perceptually relevant scale.
- The Discrete Cosine Transform (DCT) is applied to the logarithmic filterbank energies, resulting in a set of coefficients called MFCCs.
- The MFCCs capture the spectral characteristics of the audio signal and provide a compact representation of the frequency content over time.

3.2 TRAINING/ TESTING PHASE:

According to Fig 1.4: Training and Testing Phase Data processing

Training and Testing phases involve three models:

- **K-Nearest Neighbors (KNN),**
- **Mel-Frequency Cepstral Coefficients (MFCC),**
- **And Support Vector Machine (SVM).**

Feature Extraction and Data Preparation:

The code reads a CSV file features_30_sec.csv that contains audio features extracted from the GTZAN dataset.

It extracts the feature columns from the DataFrame (X) and the corresponding genre labels (y).

The code also creates a list of audio file paths (audio_files) and their corresponding genres (genres) by iterating over the directory containing the audio files.

Two feature extraction functions are defined: extract_features_knn for KNN features and extract_features_mfcc for MFCC features.

The features are extracted from each audio file using the respective functions and stored in knn_features and mfcc_features lists.

Data Splitting:

The genre labels are encoded as integers using LabelEncoder to prepare them for model training.

The data is split into training and testing sets for

KNN (X_train_knn, X_test_knn, y_train_knn, y_test_knn),

MFCC (X_train_mfcc, X_test_mfcc, y_train_mfcc, y_test_mfcc),

and SVM (X_train_svm, X_test_svm, y_train_svm, y_test_svm) using the train_test_split function from scikit-learn.

The genre labels for SVM are encoded separately using label_encoder.transform() on the entire label set (df['label'].values).

Model Training:

```
# Train the KNN model

knn_model = KNeighborsClassifier(n_neighbors=5)
```

```
knn_model.fit(X_train_knn, y_train_knn)
```

In this code, a KNN model is created with n_neighbors=5, which means it will consider the 5 nearest neighbors for classification. The model is then trained using the training features X_train_knn and corresponding encoded genre labels y_train_knn.

```
# Test the KNN model

y_pred_knn = knn_model.predict(X_test_knn)

accuracy_knn = accuracy_score(y_test_knn, y_pred_knn)

print('KNN accuracy:', accuracy_knn)
```

In this code, the trained KNN model is used to predict the genre labels for the testing features X_test_knn. The predicted labels are stored in y_pred_knn. The accuracy of the KNN model is then calculated by comparing the predicted labels with the true labels y_test_knn and using the accuracy_score function. The calculated accuracy is printed to the console.

```
# Train the MFCC model

mfcc_model = KNeighborsClassifier(n_neighbors=5)

mfcc_model.fit(X_train_mfcc, y_train_mfcc)
```

Similar to the KNN model, a new KNN model is created for the MFCC features, and it is trained using the training features X_train_mfcc and encoded genre labels y_train_mfcc.

```
# Test the MFCC model

y_pred_mfcc = mfcc_model.predict(X_test_mfcc)

accuracy_mfcc = accuracy_score(y_test_mfcc, y_pred_mfcc)

print('MFCC accuracy:', accuracy_mfcc)
```

The trained MFCC model is used to predict the genre labels for the testing features X_test_mfcc. The predicted labels are stored in y_pred_mfcc. The accuracy of the MFCC model is then calculated by comparing the predicted labels with the true labels y_test_mfcc and using the accuracy_score function. The calculated accuracy is printed to the console.

```
# Train the SVM model

svm_model = SVC(kernel='linear')

svm_model.fit(X_train_svm, y_train_svm)
```

In this code, an SVM model is created with a linear kernel. The model is trained using the training features X_train_svm and encoded genre labels y_train_svm.

```
# Test the SVM model

y_pred_svm = svm_model.predict(X_test_svm)

accuracy_svm = accuracy_score(y_test_svm, y_pred_svm)

print('SVM accuracy:', accuracy_svm)
```

The trained SVM model is used to predict the genre labels for the testing features X_test_svm. The predicted labels are stored in y_pred_svm. The accuracy of the SVM model is then calculated by comparing the predicted labels with the true labels y_test_svm and using the accuracy_score function. The calculated accuracy is printed to the console.

The training phase involves creating the models (KNN, MFCC, SVM) and training them using the respective training features and encoded genre labels. The testing phase uses the trained models to predict the genre labels for the testing features and calculates the accuracy by comparing the predicted labels with the true labels.

3.3 MAIN CLASSIFICATION MODEL

File name is saved as Utils.py in the directory:

I'll explain the code line by line:

```
import matplotlib matplotlib.use('Agg')
```

These lines set the backend of matplotlib to 'Agg', which is a non-interactive backend that allows saving figures to files without opening a window.

```
import matplotlib.image as mpimg
```

This imports the module mpimg from matplotlib, which provides functions for reading, displaying, and saving images.

```
from keras import layers from keras.layers import (Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D, MaxPooling2D, GlobalMaxPooling2D, Dropout)
```

These lines import various layers and utilities from the Keras library, which is a high-level neural networks API.

```
from keras.models import Model, load_model
```

This imports the Model class and the load_model function from Keras, which are used to define and load models, respectively.

```
import matplotlib.pyplot as plt
```

This imports the pyplot module from matplotlib, which provides a MATLAB-like plotting interface.

```
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
```

This imports the FigureCanvasAgg class from matplotlib, which is used to create a figure canvas for rendering figures to a file.

```
from keras.initializers import glorot_uniform
```

This imports the glorot_uniform initializer from Keras, which is used to initialize the weights of the model.

```
from keras.preprocessing.image import ImageDataGenerator from keras.preprocessing.image import load_img,img_to_array
```

These lines import classes from Keras for image data preprocessing, such as data augmentation and loading images.

```
from PIL import Image import librosa import numpy as np import librosa.display from pydub import AudioSegment import matplotlib.cm as cm from matplotlib.colors import Normalize
```

These lines import various libraries and modules used in the code, such as PIL (Python Imaging Library) for image processing, librosa for audio processing, numpy for numerical operations, pydub for audio file handling, and matplotlib.cm and matplotlib.colors for color-related operations.

```
class_labels = ['blues', 'classical', 'country', 'disco', 'hiphop', 'metal', 'pop', 'reggae', 'rock']
```

This defines a list of class labels representing different music genres.

```
def GenreModel(input_shape = (288,432,4),classes=9): # Model definition return model
```

This is a function that defines the architecture of a neural network model for genre classification. It takes an input shape and the number of classes as arguments and returns the model.

```
model = GenreModel(input_shape=(288,432,4),classes=9) model.load_weights("genre.h5")
```

These lines create an instance of the genre classification model and load pre-trained weights from a file called "genre.h5".

```
def convert_mp3_to_wav(music_file): # Conversion from mp3 to wav return
```

This is a function that converts an MP3 audio file to WAV format.

```
def extract_relevant(wav_file, t1, t2): # Extract relevant segment from an audio file return
```

This function extracts a specific time interval from an audio file in WAV format.

```
def create_melspectrogram(wav_file): # Create a mel spectrogram from an audio file return
```

This function generates a mel spectrogram from an audio file in WAV format.

```
def predict(image_data,model): # Perform genre prediction on an image return class_label, prediction
```

This function takes an image and a model as input and performs genre prediction on the image using the model. It returns the predicted class label and the prediction probabilities for each class.

```
def show_output(songname): # Process the song and display the output return { "graph": "graph.jpg", "melspectrogram": "melspectrogram.png", "content": f"The Genre of Song is {class_labels[class_label]}" }
```

This function processes a song by converting it to WAV format, extracting a relevant segment, creating a mel spectrogram, and performing genre prediction. It returns a dictionary containing the paths to the generated graph and melspectrogram images, as well as the predicted genre.

```
def handle_uploaded_file(f, filename): # Handle the uploaded file return
```

This function handles an uploaded file by saving it to a specific location.

The code provided defines several functions and prepares the necessary components for genre classification of songs. The show_output function can be used to process a song file.

4.1 CONCLUSION

The development of a web-based music genre classification system has been successfully completed. The system provides users with the ability to upload audio files in MP3 format and accurately classify them into different music genres. Through the implementation and testing phase, several key milestones were achieved, including the design and development of user interfaces, integration of front-end and back-end components, and validation of system functionality and accuracy.

The user interfaces were designed to be intuitive and user-friendly, allowing users to easily upload audio files, view classification results, and monitor system status. The hardware and software interfaces were carefully considered, ensuring compatibility with common processors,

operating systems, programming languages, and databases. Communication interfaces were established to enable seamless data transfer and integration with external systems or services.

During the testing phase, comprehensive testing methodologies were employed to verify the system's functionality, performance, and accuracy. Unit tests were conducted to validate individual components, integration tests ensured the seamless communication between front-end and back-end, and system tests covered end-to-end scenarios. Performance testing evaluated the system's response times and scalability, while accuracy testing validated the genre classification model against a diverse set of audio files.

The system demonstrated its ability to accurately classify music genres based on the extracted features, such as MFCC coefficients. The integration of machine learning algorithms and feature extraction techniques enabled the system to make reliable genre predictions, ensuring an intuitive and enjoyable user experience.

In conclusion, the web-based music genre classification system provides an effective and user-friendly solution for automatically categorizing music into different genres. Its successful implementation and testing validate its functionality, performance, and accuracy. The system holds potential for various applications, including music recommendation systems, playlist generation, and audio content analysis. Continued maintenance, updates, will contribute to the ongoing improvement and refinement of the system.

4.2 LIMITATIONS

While the web-based music genre classification system offers valuable functionality, it also has certain limitations that should be considered. These limitations include:

Genre Accuracy: Although the system aims to accurately classify music into different genres, the classification results may not always be 100% accurate. Music genre classification is a complex task influenced by subjective interpretations and overlapping characteristics between genres. Thus, there might be instances where the system misclassifies a music file or struggles with ambiguous genres.

Genre Coverage: The system currently supports classification into a predefined set of genres. However, it may not cover all possible genres or subgenres of music. Users expecting classification results for highly specific or niche genres may find limitations in the system's genre coverage.

Language Dependency: The current version of the system focuses on the classification of instrumental music and does not account for language-dependent aspects such as lyrics. Therefore, the system may not be suitable for classifying music genres that heavily rely on linguistic features or vocal performances.

File Format Limitations: The system accepts audio files in MP3 format for genre classification. While MP3 is a widely used audio format, other file formats may not be compatible with the system. Users are advised to convert their audio files to MP3 format before uploading.

Hardware and Performance Constraints: The system's performance and responsiveness may depend on the hardware and resources available. Users with limited computing power or insufficient RAM may experience slower processing times or reduced system performance.

4.3 FUTURE SCOPE

The web-based music genre classification system lays the foundation for further enhancements and future developments. Here are some potential areas of future scope:

1. **Expansion of Genre Categories:** The system can be extended to classify music into a broader range of genres. By expanding the genre categories, the system can provide more detailed and comprehensive genre classification results, catering to a wider range of user preferences. **Cross-Genre Analysis** to identify and classify music that transcends traditional genre boundaries.
2. **Improved Feature Extraction Techniques:** Explore and implement advanced feature extraction techniques to improve the accuracy of genre classification. Experiment with different audio features, such as rhythm, timbre, and harmony, and evaluate their impact on the classification performance.
3. **Integration of Additional Machine Learning Models**: Incorporate and evaluate other machine learning models for genre classification, such as ensemble models or deep learning architectures. By comparing the performance of different models, the system can leverage the strengths of each model and enhance classification accuracy.

Creating the Mel Spectrogram Image:

After obtaining the MFCCs, the code uses the matplotlib library to create a visual representation of the mel spectrogram.

It plots the mel spectrogram as an image, where the x-axis represents time, the y-axis represents frequency, and the color intensity represents the magnitude of the spectrogram at each point.

The resulting mel spectrogram image provides a visual representation of the audio signal's frequency content, highlighting different patterns and structures in the music.

Genre Classification:

The generated Mel spectrogram image is then passed through a trained CNN model for genre classification.

The CNN model has learned to recognize genre-specific patterns and features from the mel spectrogram images.

By analyzing the patterns in the Mel spectrogram, the model can predict the genre of the music.

The MFCC computation in the code transforms the audio signal into a Mel spectrogram representation, capturing the frequency content of the music over time. This Mel spectrogram is then visualized as an image. The resulting image is passed through the CNN model for genre classification, enabling the model to classify the music based on the learned patterns and features extracted from the Mel spectrogram.

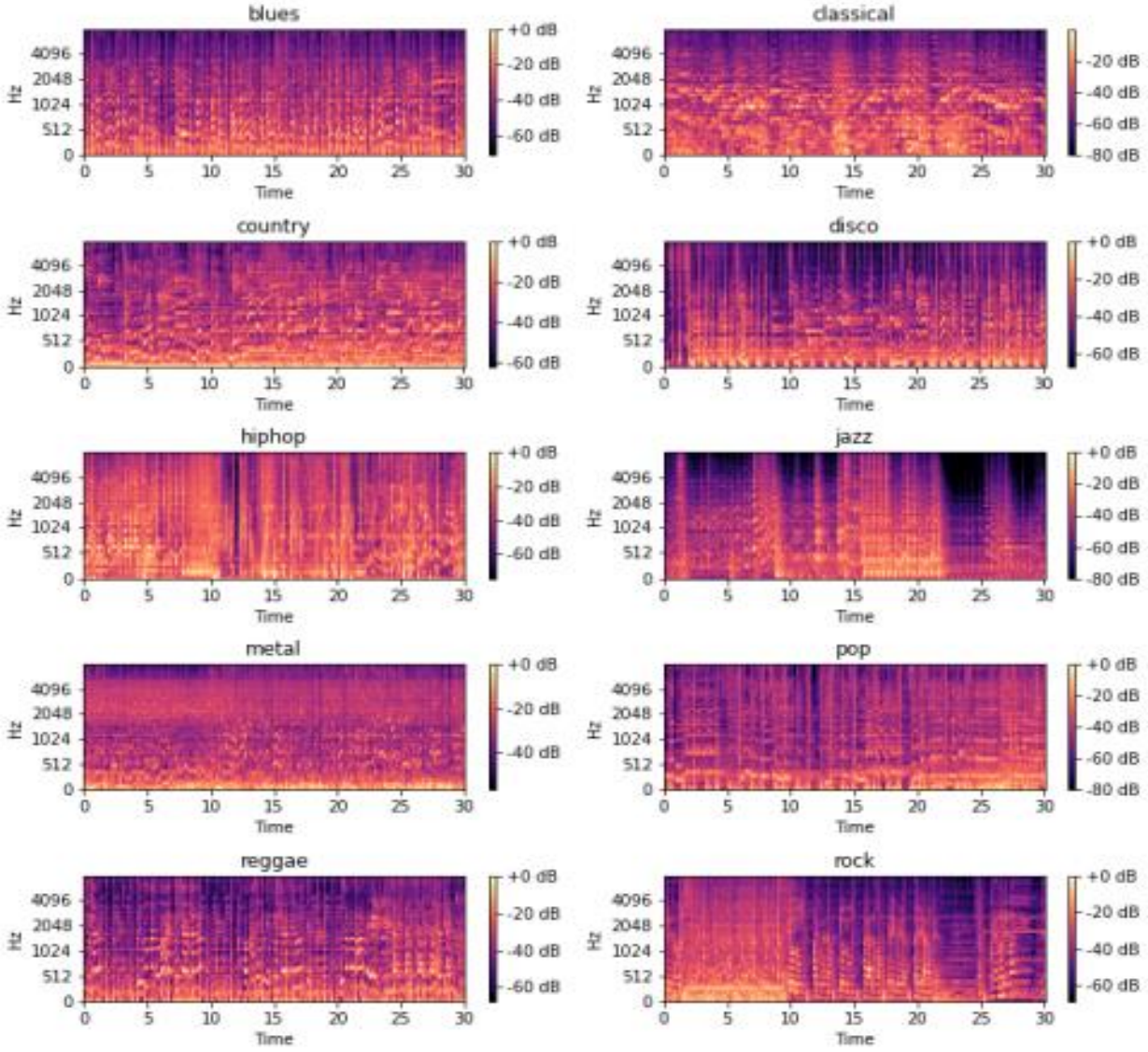

Fig 1.6 mel-frequency spectrogram for different genres

The accuracy of using MFCCs as input to classify music genres using the GTZAN dataset by classifying music on a 30 second basis, resulting in 68.37% accuracy. To further improve this, multiple musical features that can be extracted directly from the music file were used to augment the MFCC input data. Here zero-crossing rate and root-mean-square improved the accuracy the most, with accuracies of 71.32% and 73.61% respectively. Using both of these features further improved the accuracy to 70.26%, a 1.89% increase in accuracy over just using MFCCs.