**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

**Subject:**

Signals and Systems

**Project Report**

**SUBMITTED TO:**

Ma’am Aleena

Sir Furqan

**SUBMITTED BY:**

**Student Name**

Muhammad Mussa Kazim **Reg# 404047**

Bazil bin Aamir **Reg# 432243**

Muhammad Taha **Reg#417609**

Talha Iftikhar **Reg#410998**

**DE- 44 Department C&SE – Syn B**

**Submission Date: 31/12/24**

**PROJECT TITLE: AUDIO CLASSIFICATION**

**PROJECT OVERVIEW:**

This project involved developing a system to differentiate between various sounds using signal processing techniques. We have implemented algorithms to classify audio signals into different categories (e.g., speech, music, noise) using neural networks. This project has provided us with foundational experience in signal processing, including filtering, pitch detection, and time and frequency domain analysis.

# PROJECT OBJECTIVES:

1. Understand Sound Signals: Learn about ECG signals and their significance in monitoring heart activity, focusing on identifying R-peaks within the QRS complex.
2. Signal Pre-processing: Apply filtering and data augmentation techniques to classify the signal into voice, music and noise through pitch detection and information.
3. Pitch Detection: Implement algorithms to accurately detect and shift pitch.
4. Audio Classification: Classify the sounds into three broad categories as

* Speech.
* Music.
* Noise.

# PROJECT SPECIFICATIONS:

**Tools and Resources:**

**• Software:**

* Python (with libraries like NumPy, SciPy, TensorFlow) for signal processing.
* MATLAB for Preprocessing of dataset.

**• Datasets:** Musan audio Dataset (for model training and model testing).

**Data Collection:**

For the implementation of our Project we have used publicly available Musan dataset (which has roughly 2000 audio samples for human speech, noise and music files) for our Model Training and Model Testing in Python.

**Preprocessing:**

First we Preprocess our Musan dataset using MATLAB. We apply a band pass filter of 500-6000 Hz for Speech files in our dataset because Human Audible Speech ranges between these frequency ranges. We apply a Low Pass Filter of 12000 Hz on Music Files (Frequencies above 12000 Hz become noise to Human Ears) and no Filter for Noise. We have applied Filters on these audio files in order to remove background noise (if any) from these files. This Filtered dataset is then saved into another Folder.

**Code:**

folderPath = 'C:\Downloads\musan\Speech'; % Replace with your folder path

filteredFolder = 'C:\Downloads\musan\Speech\_filtered'; % Folder to save filtered .wav files

% Create the folder for filtered .wav files if it does not exist

if ~exist(filteredFolder, 'dir')

mkdir(filteredFolder);

end

% Get all .wav files in the folder

files = dir(fullfile(folderPath, '\*.wav'));

numFiles = length(files);

% Batch size (number of files to process at a time)

batchSize = 50;

% Process files in batches

for startIdx = 1:batchSize:numFiles

endIdx = min(startIdx + batchSize - 1, numFiles); % Ensure we don't go beyond the last file

% Process each file in the current batch

for i = startIdx:endIdx

filePath = fullfile(folderPath, files(i).name);

% Display progress

fprintf('Processing file %d of %d: %s\n', i, numFiles, files(i).name);

% Read audio

[audio, Fs] = audioread(filePath);

% Apply filter (using the existing LP filter object)

filteredAudio = filter(BP, audio);

% Convert stereo to mono if necessary

if size(filteredAudio, 2) > 1

filteredAudio = mean(filteredAudio, 2); % Convert stereo to mono

end

% Normalize the audio

filteredAudio = filteredAudio / max(abs(filteredAudio));

% Save the filtered audio to the new folder

filteredFilePath = fullfile(filteredFolder, files(i).name);

audiowrite(filteredFilePath, filteredAudio, Fs);

end

% Clear variables to free up memory after each batch

clear audio filteredAudio;

disp(['Processed batch from ', num2str(startIdx), ' to ', num2str(endIdx)]);

end

disp('Filtering and saving complete.');

**Features Extraction**:

After Preprocessing our data using MATLAB, we extract features of our audio dataset using Python Library **Librosa.**

Librosa is a powerful Python library built to work with audio and perform analysis on it. It is used to work with audio data at scale for a wide range of applications such as detecting voice from a person to finding personal characteristics from an audio.

We have extracted a total of 5 features from our audio.

* **Pitch**

Pitch refers to the degree of highness or lowness of a sound, and it is determined by the frequency of the soundwave. Higher frequency results in a higher pitch, while lower frequency results in a lower pitch.

* **Zero Crossing Rate (ZCR)**

The zero-crossing rate (ZCR) is the rate at which a signal changes from positive to zero to negative or from negative to zero to positive. Its value has been widely used in both speech recognition and music information retrieval, being a key feature to classify percussive sounds.

* **Root Mean Square Value (RMS)**

RMS (Root Mean Square) is a crucial metric in audio engineering, representing the average power or loudness of an audio signal. It provides invaluable insight into the dynamics of sound, helping to bridge the gap between technical measurements and human auditory perception.

* **Energy**

Energy is an ability to do work. Sound energy is a form of kinetic energy caused by the physical vibration of air particles or molecules. The particles collide with other neighboring particles causing them to vibrate. These vibrations travel in a straight line. When they reach our ears, we perceive them as sound.

* **Mel-frequency cepstral coefficients**

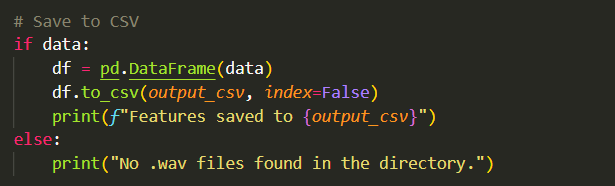
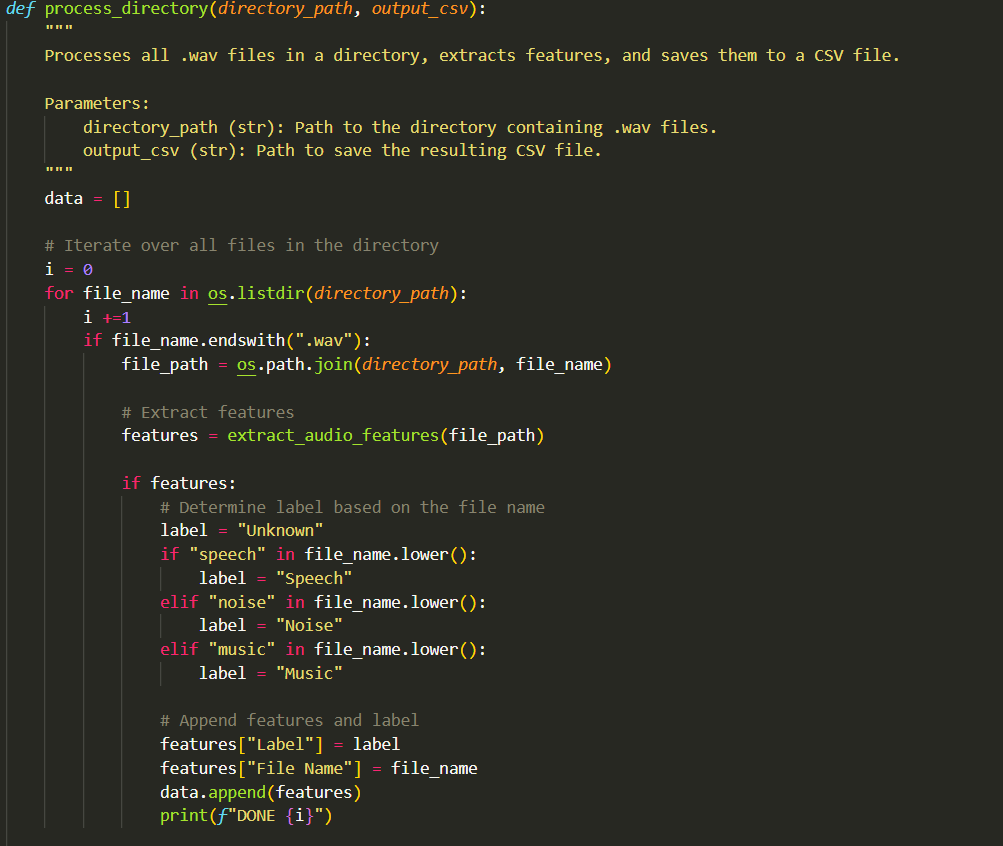
Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC.[1] They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal spectrum. This frequency warping can allow for better representation of sound, for example, in audio compression that might potentially reduce the transmission bandwidth and the storage requirements of audio signals.

MFCCs are commonly derived as follows:

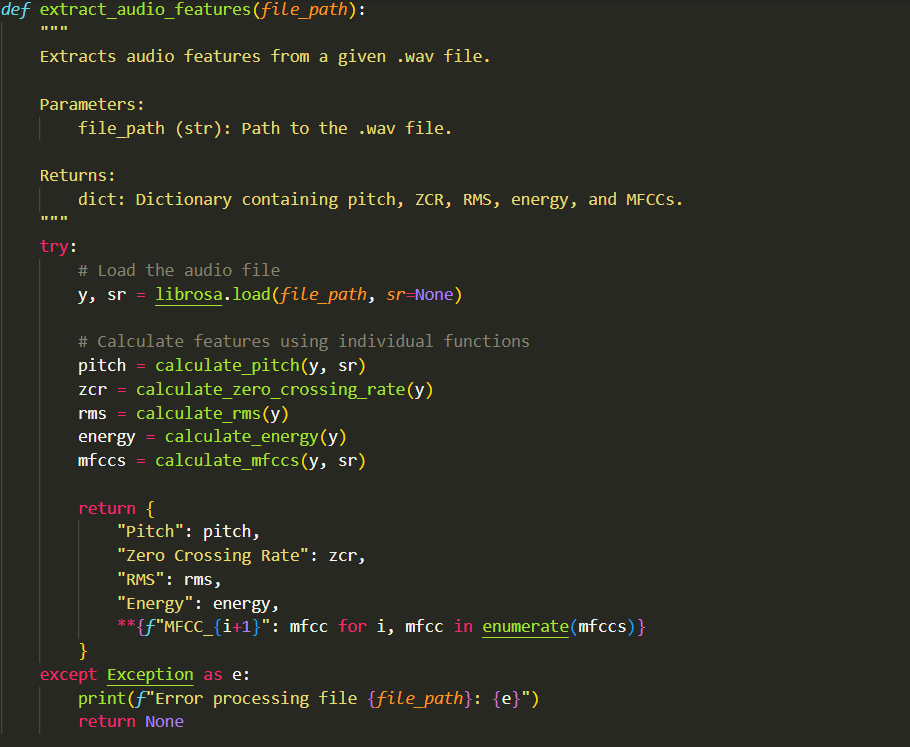
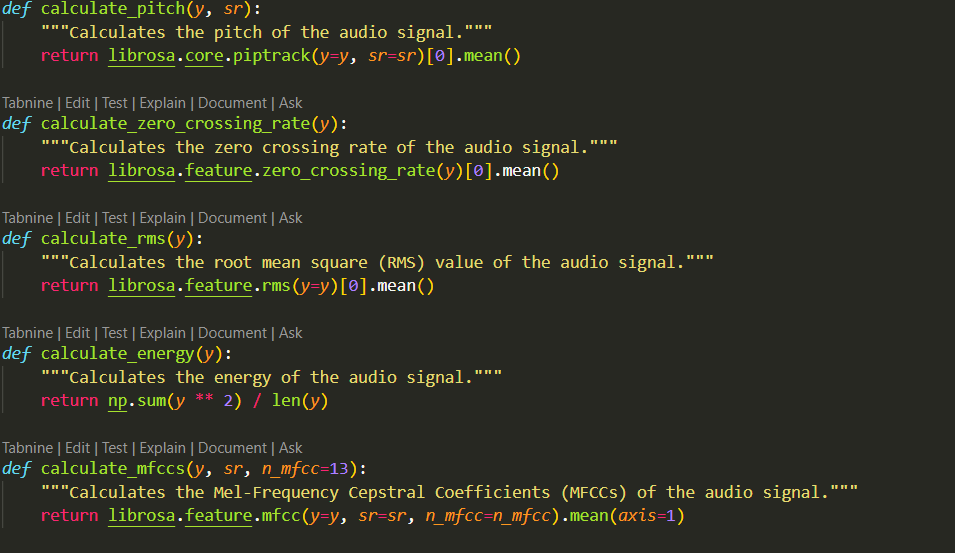
1. Take the Fourier transform of (a windowed excerpt of) a signal.
2. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows or alternatively, cosine overlapping windows.
3. Take the logs of the powers at each of the mel frequencies.
4. Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
5. The MFCCs are the amplitudes of the resulting spectrum.

In Python, We have processed all .wav files in a directory, extracts features, and saves them to a CSV file.

Feature Extraction is done as follows:



Functions called is Feature Extraction Function are defined as follows:



**Audio Classification**:

To classify the sounds into three broad categories as Speech, Music and Noise, we have trained our Deep Learning AI Model using TensorFlow along with Python Libraries such as NumPy and Pandas on Jupyter Notebook.

* First, we have loaded our audio file data and extracted its all features.

A screen shot of a computer

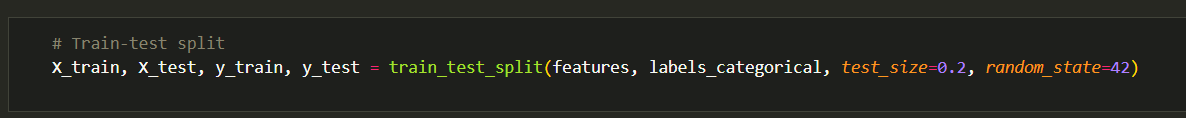
Description automatically generated

* Then we have scaled all parameters of the audio to help our model understand the features more efficiently.

A screen shot of a computer program

Description automatically generated

* We have dedicated 20% of our Musan Dataset for Model Testing, the rest 80% is dedicated for Model Training.



* Then we have set our Model to Train, we have set epoch to 100 in order to train the model 100 times

A screen shot of a computer code

Description automatically generated

* After Training our Neural Network Model, we have got 98.85% Model Training Accuracy.

A screen shot of a computer

Description automatically generated

* Then we tested our Trained Model and We have got 94.35% Model Testing Accuracy

A screen shot of a computer program

Description automatically generated

* In the end we have saved our Trained Model.

A screenshot of a computer

Description automatically generated

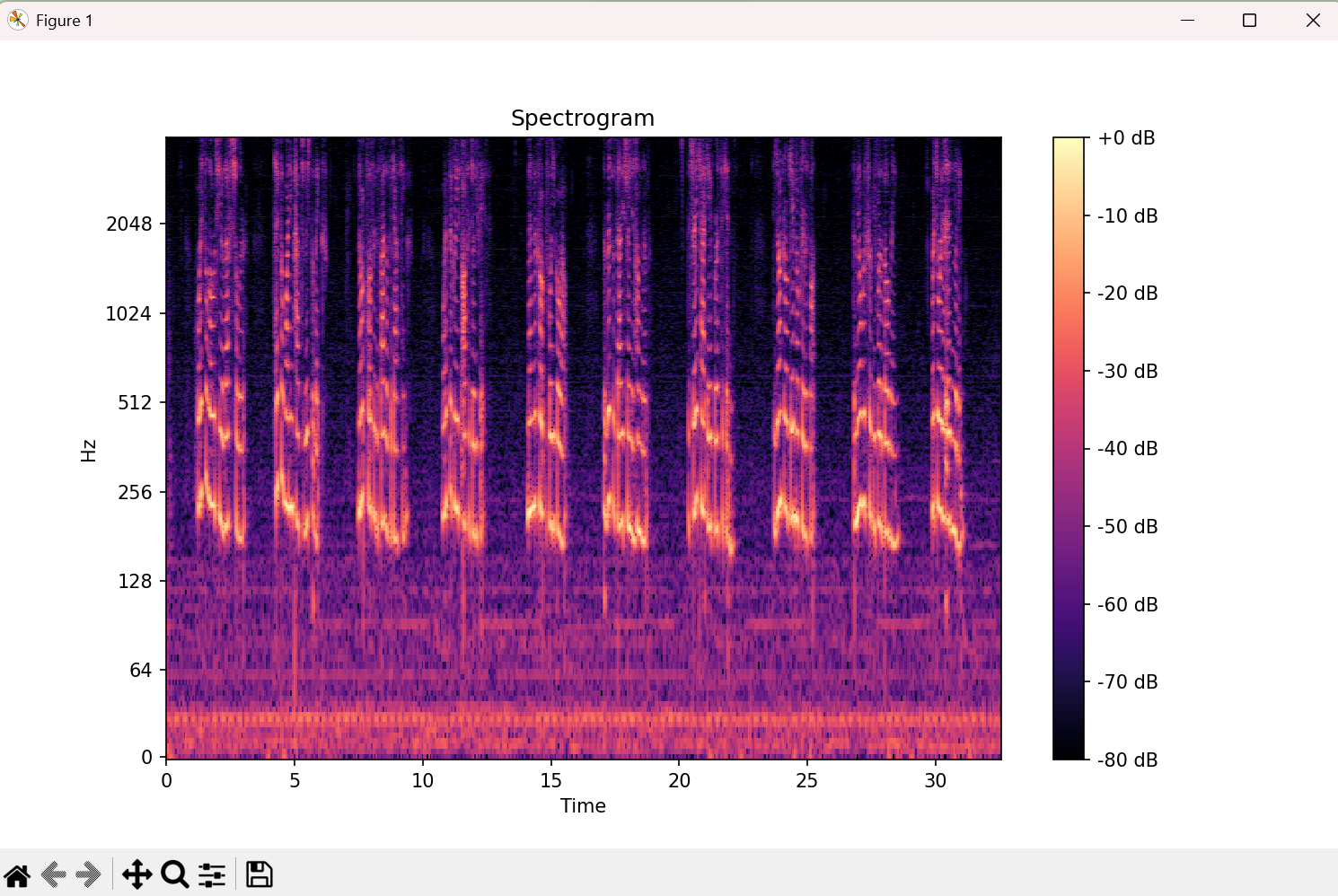
**Graphical User Interface using Flask:**

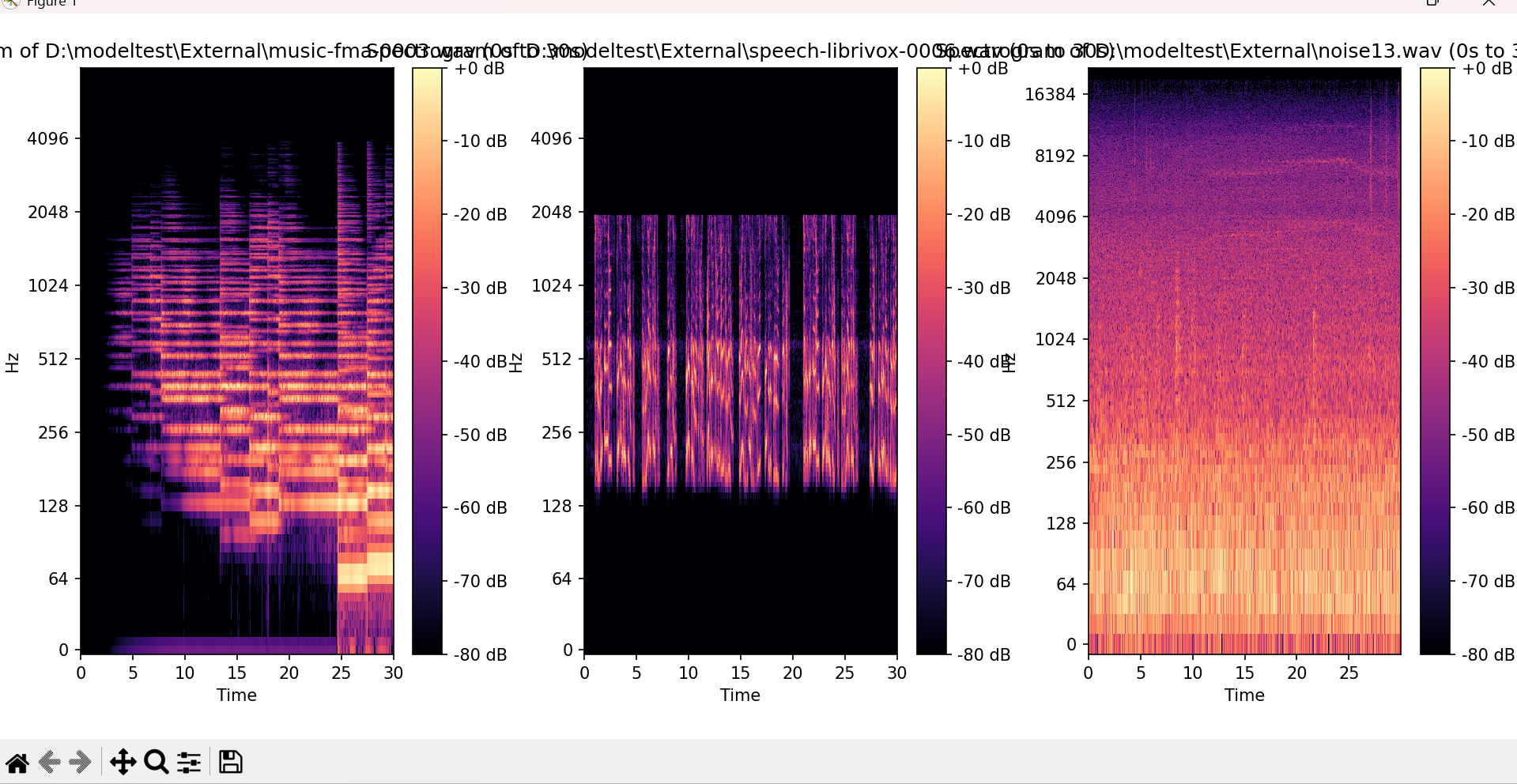
After implementing our Trained Model in main.py , we have then integrated our whole project into Graphical User Interface App using Flask for great user experience. The GUI will tell the predicted Sound Class and will also provide the spectrogram for the file.

A screen shot of a sign

Description automatically generated

**SPECTRUM PLOTTING:**

THE SPECTRUM OF THE CLASSIFIED FILE IS ALSO SHOWN :  
  
**Spectrogram of Music, Speech and Noise File in Testing Data :**

****The above spectrograms represent three distinct audio waveforms: music, speech, and noise. Each spectrogram visualizes the frequency distribution over time, with intensity represented by the color scale. These visualizations serve as references to classify and analyze a new audio file based on its spectral characteristics, aiding in distinguishing between different sound categories.

**Flask Application Workflow for Audio Classification**The Flask application enables the classification of an uploaded audio file. Upon pressing the Classify button, the app performs the following steps:

1. Audio File Processing: The uploaded audio file is preprocessed and converted into a spectrogram using libraries like librosa or matplotlib.
2. Feature Extraction and Classification: The generated spectrogram is analyzed using a pre-trained classification model to identify whether the audio is music, speech, or noise.
3. Result Visualization: The spectrogram of the classified audio file is displayed alongside the reference spectrograms (music, speech, and noise). The classification result is presented clearly, assisting in understanding the categorization.

**Testing and Evaluation:**

After complete integration of our Project we finally test our audio classification application and it classifies audio files with great accuracy.

**Conclusion:**

In conclusion, this project successfully implemented an audio classification system using the MUSAN dataset, categorizing audio samples into speech, music, and noise. By extracting key features such as pitch, MFCCs, zero-crossing rate (ZCR), energy of the signal, and RMS, we were able to achieve an impressive test accuracy of approximately 94% using a simple Multilayer Perceptron (MLP) model. To further enhance the user experience, we developed a user-friendly Flask application, providing an intuitive graphical interface for real-time audio classification. This project demonstrates the potential of combining traditional signal processing techniques with machine learning to efficiently classify audio data, and the Flask application adds accessibility for users to interact with the system seamlessly.