DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING

Major Project Phase-II Report

Audio-Music Fingerprint Recognition

By
Chinmaya Murthy - ENG19CS0077
Chirayu S M - ENG19CS0079
Dayanand Kavalli - ENG19CS0080
Divya J - ENG19CS0088

Under the supervision of

Dr. Girisha G S

Prof. and Chairman, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING, SCHOOL OF ENGINEERING DAYANANDA SAGAR UNIVERSITY, BANGALORE

(2022-2023)

DAYANANDA SAGAR UNIVERSITY

School of Engineering
Department of Computer Science & Engineering
Kudlu Gate, Bangalore – 560068
Karnataka, India

CERTIFICATE

This is to certify that the Phase-II project work titled **"AUDIO-MUSIC FINGERPRINT RECOGNITION"** is carried out by **Chinmaya Murthy(ENG19CS0077), Chirayu S M (ENG19CS0079), Dayanand Kavalli(ENG19CS0080), Divya J(ENG19CS0088),** bonafide students of Bachelor of Technology in Computer Science and Engineering at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering, during the year **2022-2023**.

Prof Guide Name	**Dr. Girisha G S**	**Dr. Udaya Kumar Reddy K R**
Assistant/Associate/ Professor Dept. of CS&E, School of Engineering Dayananda Sagar University	Chairman CSE School of Engineering Dayananda Sagar University	Dean School of Engineering Dayananda Sagar University
Date: 27/03/2023	Date: 27/03/2023	Date: 27/03/2023

Name of the Examiner **Signature of Examiner**

1.

DECLARATION

We, **Chinmaya Murthy (ENG19CS0077), Chirayu S M (ENG19CS0079), Dayanand Kavalli (ENG19CS0080), Divya J (ENG19CS0088),** are students of the eighth semester B. Tech in **Computer Science and Engineering**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the Major Project Stage-II titled **"AUDIO-MUSIC FINGERPRINTING RECOGNITION"** has been carried out by us and submitted in partial fulfillment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2022-2023**.

Student **Signature:**

Name1: Chinmaya Murthy

USN: ENG19CS0077

Name2: Chirayu SM

USN: ENG19CS0079

Name3: Dayanand Kavalli

USN: ENG19CS0080

Name4: Divya J

USN: ENG19CS0088

Place: Bangalore

Date: 27/03/2023

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
DL	Deep Learning
GUI	Graphical User Interface
PHP	Pre-Processor Hyper text
MySQL	My Structured Query Language

LIST OF FIGURES

ABSTRACT

Audio fingerprinting is the process of representing an audio signal in a compact way by extracting relevant features of the audio content. Some of the significant applications of acoustic fingerprinting include content-based audio retrieval-broadcast monitoring etc... It permits observing of the sound free of its arrangement and without the requirement for metadata. Works by analyzing a song's frequency patterns and finding a match within its database of songs. The proposed method will be able to recognize the music when it is played in any app. It will be able to record the music in the background for some seconds and will recognize it. After recognizing the application will be providing you with the link to that particular song.

CHAPTER 1

INTRODUCTION

CHAPTER 1 INTRODUCTION

Audio fingerprinting is the process of representing an audio signal in a compact way by extracting relevant features of the audio content. Some of the significant applications of acoustic fingerprinting include content-based audio retrieval, broadcast monitoring, etc. It permits observing of the sound free of its arrangement and without the requirement for metadata. Works by analyzing frequency patterns and finding a match within its database of songs. This application identifies songs using an audio fingerprint based on a time-frequency graph called a spectrogram. It uses a smartphone or computer's built-in microphone to gather a brief sample of audio being played. It works by analyzing the captured sound and seeking a match based on an acoustic fingerprint in a database of millions of songs. If it finds a match, it sends information such as the artist, song title, and so on.

1.1. FIGURES AND TABLES

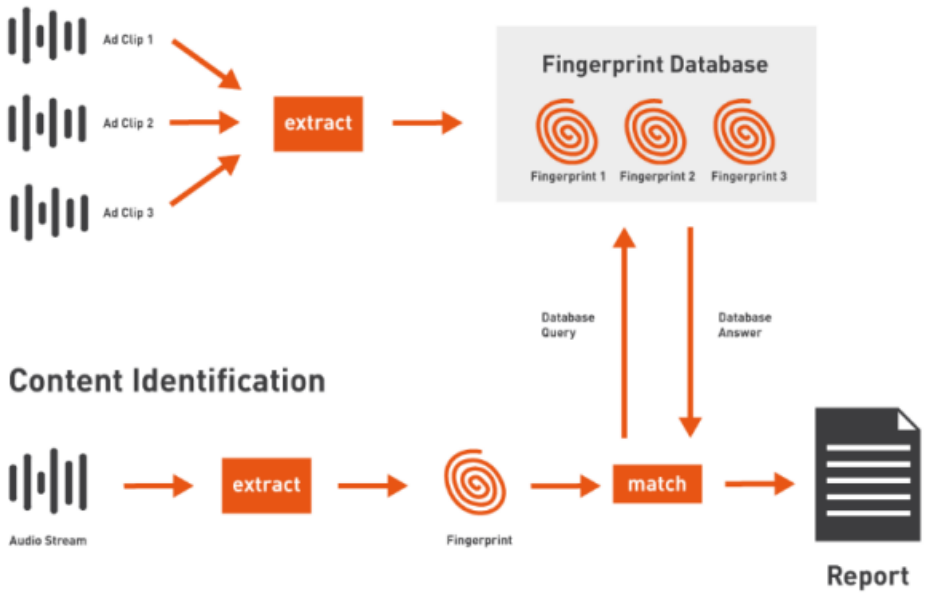


Figure 1.1., Music Fingerprint Extraction

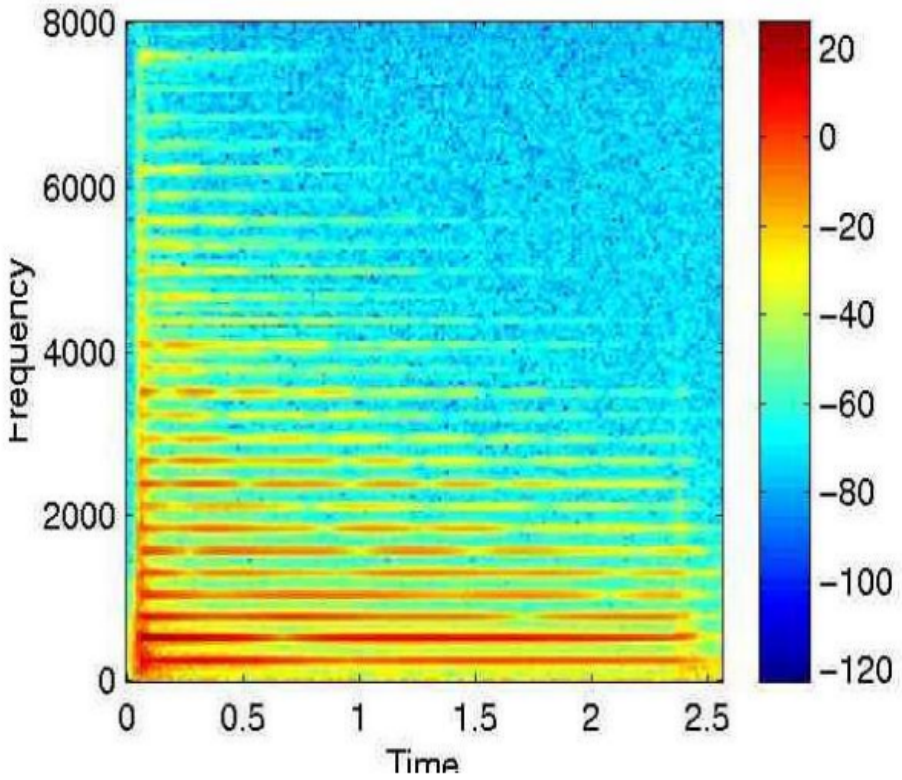


Figure 1.2., Spectrogram

1.2. SCOPE

The project distills samples of a song into fingerprints and matches these fingerprints against fingerprints from known songs dataset. From this project, we can search for any unknown songs within seconds.

CHAPTER 2
PROBLEM DEFINITION

CHAPTER 2 PROBLEM DEFINITION

You hear a familiar song on social media or any streaming application. You listened to this song a thousand times long ago, and the sentimentality of the song really touches your heart. You desperately want to hear it tomorrow, but you can't remember its name! Fortunately, in our amazing futuristic world, you have a phone with music recognition software installed, and you are saved. To overcome the need of the audience for listening to a full song by having only a sentence or tune. To get more information about a song such as which language and singer.

CHAPTER 3

LITERATURE REVIEW

CHAPTER 3 LITERATURE REVIEW

- G Deepsheka, R Keerthana, M Mourina, and B Bharati [1] have presented a paper on Recurrent neural network-based Music Recognition using Audio Fingerprinting where The model is built by using LSTM(Long short-term memory) and Songs have been created which contains the details about the hash value, offset and song details. The MySqldb package is used for interfacing with MySql databases. The approach presented for cover song detection uses Long Short-Term Memory (LSTM) neural network.

- Chahid Ouali, Vishwa Gupta, and Pierre Dumouchel [2] have presented a paper on Fast Audio Fingerprinting System Using GPU and a Clustering-Based Technique using Hashing-based similarity computation where the implementation of the energy difference fingerprint and Shazam (we used the implementation of Shazam by Dan Ellis with the default parameters). To generate audio fingerprints, this system converts the audio signal into 2-D binary images derived from the spectrogram.

- Meinard Müller, Andreas Arzt, Stefan Balke, Matthias Dorfer, and Gerhard Widmer [3] have presented a paper on Cross modal Music Retrieval and Applications using cross-modal retrieval, which can be performed based on a retrieval-by-embedding paradigm where they have created a music data algorithmically accessible, traditional music processing tries to extract suitable features that capture relevant key aspects while suppressing irrelevant details.

- Prem Seetharaman Zajar Rafii [4] presented a paper on Cover song identification with 2d Fourier transformer sequences where they used 2D Fourier Transformation and presented an approach for cover song identification that uses a time-series representation of audio based on the magnitude 2DFT. The audio is represented as a sequence of magnitude 2D Fourier transforms. The 2D Fourier Transform on Musical Signals, Fingerprint with CQT(Constant-Q-Transform), and Adaptive Thresholding. The approach is state-of-the-art on a recent cover song dataset and expands on previous work using the 2DFT for music representation and work on live song recognition.

CHAPTER 4
PROJECT DESCRIPTION

CHAPTER 4 PROJECT DESCRIPTION

4.1. System Design:

Audio fingerprinting is a content summarization technique that links short snippets of unlabeled audio content to the same contents in the database. The most well-known application is the music fingerprinting system that enables users to identify unknown songs from the microphone or streaming audio input. Some of the significant applications of acoustic fingerprinting include content-based audio retrieval, broadcast monitoring, etc. It permits observing of the sound free of its arrangement and without the requirement for metadata. Works by analyzing a song's frequency patterns and finding a match within its database of songs.

4.2. ASSUMPTIONS AND DEPENDENCIES:

Below are some of the processes of music recognition:

1. Analog vs. digital sound:

Every sound that exists in the air or any other medium is analog. Analog waves are continuous and quite complex. Digital sound, on the other hand, has to have a minimum unit just because we can't afford to store an infinite amount of data.

2. Fourier transform:

It is a formula that transforms a sound wave into a graph of frequencies that the sound is made of, and their intensities.

3. Spectrogram and audio fingerprint algorithm: Spectrogram is a visual representation of frequencies as they vary in time. In other words, it's a three-dimensional graph.

CHAPTER 5
REQUIREMENTS

CHAPTER 5 REQUIREMENTS

5.1. FUNCTIONAL REQUIREMENTS:

Functional requirements refer to the functionalities that are applicable to a software. This is a website that will allow users to solve various techniques.

- **User Authentication:** this is a security process that covers all of the human-to-computer interactions that require the user to register and log in.
- **Data Storage:** Once the audio fingerprint is created, it gets stored in the database in the form of a hash table.
- **Pattern Search:** the ability to differentiate between two closely related covers of the same song.
- **Noise Reduction:** audio stream as a spectrogram, then picks out the peak point in the audio stream via the spectrogram graph representation -Peak points are points of less background noise.

5.2. NON-FUNCTIONAL REQUIREMENTS:

Non-functional requirements define the performance of a software system and how scalable and reliable the system is and it includes scalability, recoverability, availability, capacity, etc.

- **Accuracy:** Since we will give the priority to the accuracy of the software, the performance of the Music Recommenders will be based on its accuracy on the recommendation.
- **Failure:** System components may fail independently of others.

- **Openness:** The system should be extensible to guarantee that it is useful for a reasonable period of time.
- **Security:** Sensitive information should be kept in safe

5.3. SYSTEM REQUIREMENTS

- **Software requirements:**
  - Net beans
  - Java
  - VS Code

- **Hardware requirements:**
  - OS - Windows

CHAPTER 6
METHODOLOGY

CHAPTER 6 METHODOLOGY

Fingerprint Hashing:

A song usually contains a ton of anchor points and storage of such a large file is insufficient. Likewise, there exists a limitation of having comparative pinnacles. This can be settled by consolidating tops into fingerprints through hash capability. Input and outputs are viewed in hash works with the goal that a superior hash capability could be obtained to process the input and output the whole number.

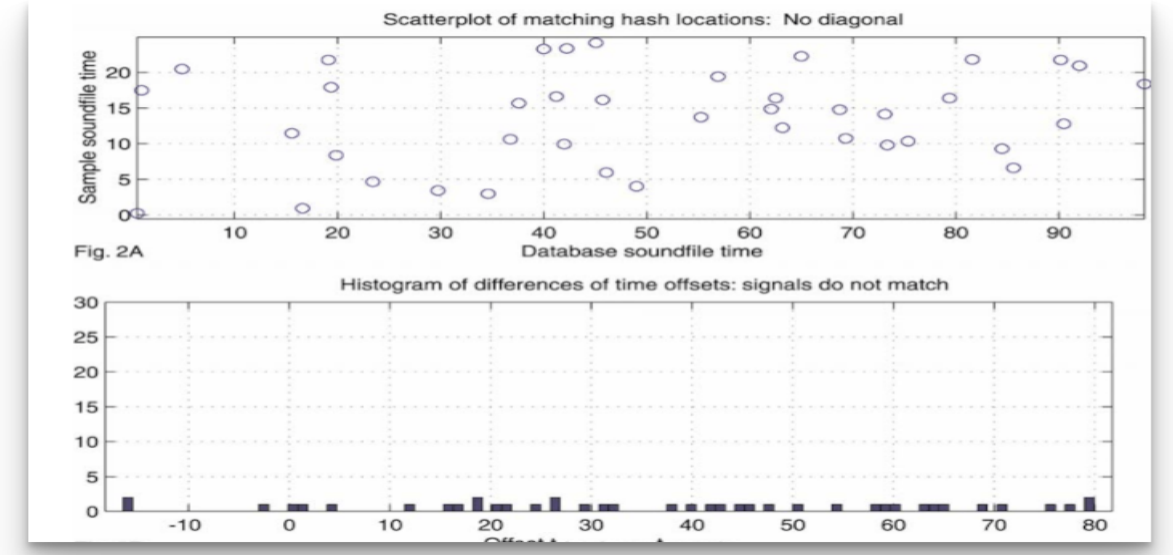


Figure 6.1., Fingerprint Hashing

We can calculate a histogram of the number of matches per offset as shown in fig 3. The height of the tallest peak in the histogram is the best score for that match.

FFT(Fast Fourier Transform)

Fast Fourier Transform (FFT) is a common method used in audio fingerprinting to extract frequency information from an audio signal. It works by breaking down an audio signal into its constituent frequencies, allowing it to be compared to other audio signals based on similarities in their frequency content. This technique is often used in applications such as music identification and content-based music retrieval.

Music identification: Audio fingerprinting is a technique used to identify a particular piece of music by analyzing its unique acoustic characteristics. The process involves creating a digital representation of the audio, known as a fingerprint, which is then compared to a database of pre-existing fingerprints to find a match.

Music Retrieval: Music retrieval in audio fingerprinting involves the use of algorithms to analyze and match digital representations of music. This process typically involves the creation of a digital fingerprint or signature of a piece of music, which is then used to identify and retrieve that music from a database.

CHAPTER 7
EXPERIMENTATION

CHAPTER 7 EXPERIMENTATION

We used the standard SciPy peak-finding methods to find peaks in the spectrum. However, the very noisy makeup of the spectrum means that there are too many peaks to identify regions of actual interest that will be useful in developing fingerprints of the song. And we created our own dataset.

➢ Characterising Sound with FFT(Fast Fourier transform)

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [8, 6]
plt.rcParams['figure.dpi'] = 140 # 200 e.g. is really fine, but slower
from scipy import fft, signal
import scipy
from scipy.io.wavfile import read
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
import glob
from typing import List, Dict, Tuple
from tqdm import tqdm
import pickle


# Read the input WAV files
# Fs is the sampling frequency of the file
Fs, song = read("./data/1.wav")

time_to_plot = np.arange(Fs * 1, Fs * 1.3, dtype=int)



# Number of sample points
N = 600
# sample spacing
T = 1.0 / 800.0
x = np.linspace(0.0, N*T, N, endpoint=False)
```

```
# Create a signal comprised of 5 Hz wave and an 10 Hz wave
y = np.sin(5.0 * 2.0*np.pi*x) + 0.75*np.sin(10.0 * 2.0*np.pi*x)
yf = fft(y)
xf = fftfreq(N, T)[:N//2]


y = np.sin(5.0 * 2.0*np.pi*x) + 0.75*np.sin(10.0 * 2.0*np.pi*x)
# Add Gaussian Noise to one of the signals
y_corrupted = y + np.random.normal(0, 2.5, len(y))

N = len(song)
fft = scipy.fft.fft(song)
transform_y = 2.0 / N * np.abs(fft[0:N//2])
transform_x = scipy.fft.fftfreq(N, 1 / Fs)[:N//2]

plt.plot(transform_x, transform_y)
plt.xlim(0, 2000)

all_peaks, props = signal.find_peaks(transform_y)

peaks, props = signal.find_peaks(transform_y, prominence=0, distance=10000)
n_peaks = 15

# Get the n_peaks largest peaks from the prominences
# This is an argpartition
# Useful explanation:
https://kanoki.org/2020/01/14/find-k-smallest-and-largest-values-and-its-indices-in-a-nu
mpy-array/
largest_peaks_indices = np.argpartition(props["prominences"], -n_peaks)[-n_peaks:]
largest_peaks = peaks[largest_peaks_indices]

plt.plot(transform_x, transform_y, label="Spectrum")
plt.scatter(transform_x[largest_peaks], transform_y[largest_peaks], color="r", zorder=10,
label="Constrained Peaks")
plt.xlim(0, 3000)

plt.show()


# Some parameters
window_length_seconds = 3
```

```
window_length_samples = int(window_length_seconds * Fs)
window_length_samples += window_length_samples % 2

# Perform a short time fourier transform
# frequencies and times are references for plotting/analysis later
# the stft is a NxM matrix
frequencies, times, stft = signal.stft(
    song, Fs, nperseg=window_length_samples,
    nfft=window_length_samples, return_onesided=True
)

stft.shape

constellation_map = []

for time_idx, window in enumerate(stft.T):
    # Spectrum is by default complex.
    # We want real values only
    spectrum = abs(window)
    # Find peaks - these correspond to interesting features
    # Note the distance - want an even spread across the spectrum
    peaks, props = signal.find_peaks(spectrum, prominence=0, distance=200)

    # Only want the most prominent peaks
    # With a maximum of 5 per time slice
    n_peaks = 5
    # Get the n_peaks largest peaks from the prominences
    # This is an argpartition
    # Useful explanation:
https://kanoki.org/2020/01/14/find-k-smallest-and-largest-values-and-its-indices-in-a-nu
mpy-array/
    largest_peaks = np.argpartition(props["prominences"], -n_peaks)[-n_peaks:]
    for peak in peaks[largest_peaks]:
        frequency = frequencies[peak]
        constellation_map.append([time_idx, frequency])

# Transform [(x, y), ...] into ([x1, x2...], [y1, y2...]) for plotting using zip
plt.scatter(*zip(*constellation_map))
plt.show()
```

```
def create_constellation(audio, Fs):
    # Parameters
    window_length_seconds = 0.5
    window_length_samples = int(window_length_seconds * Fs)
    window_length_samples += window_length_samples % 2
    num_peaks = 15

    # Pad the song to divide evenly into windows
    amount_to_pad = window_length_samples - audio.size % window_length_samples

    song_input = np.pad(audio, (0, amount_to_pad))

    # Perform a short time fourier transform
    frequencies, times, stft = signal.stft(
        song_input, Fs, nperseg=window_length_samples, nfft=window_length_samples,
return_onesided=True
    )

    constellation_map = []

    for time_idx, window in enumerate(stft.T):
        # Spectrum is by default complex.
        # We want real values only
        spectrum = abs(window)
        # Find peaks - these correspond to interesting features
        # Note the distance - want an even spread across the spectrum
        peaks, props = signal.find_peaks(spectrum, prominence=0, distance=200)

        # Only want the most prominent peaks
        # With a maximum of 15 per time slice
        n_peaks = min(num_peaks, len(peaks))
        # Get the n_peaks largest peaks from the prominences
        # This is an argpartition
        # Useful explanation:
https://kanoki.org/2020/01/14/find-k-smallest-and-largest-values-and-its-indices-in-a-nu
mpy-array/
        largest_peaks = np.argpartition(props["prominences"], -n_peaks)[-n_peaks:]
        for peak in peaks[largest_peaks]:
            frequency = frequencies[peak]
            constellation_map.append([time_idx, frequency])
```

```
    return constellation_map


constellation_map = create_constellation(song, Fs)

def create_hashes(constellation_map, song_id=None):
    hashes = {}
    # Use this for binning - 23_000 is slighlty higher than the maximum
    # frequency that can be stored in the .wav files, 22.05 kHz
    upper_frequency = 23_000
    frequency_bits = 10

    # Iterate the constellation
    for idx, (time, freq) in enumerate(constellation_map):
        # Iterate the next 100 pairs to produce the combinatorial hashes
        # When we produced the constellation before, it was sorted by time already
        # So this finds the next n points in time (though they might occur at the same time)
        for other_time, other_freq in constellation_map[idx : idx + 100]:
            diff = other_time - time
            # If the time difference between the pairs is too small or large
            # ignore this set of pairs
            if diff <= 1 or diff > 10:
                continue

            # Place the frequencies (in Hz) into a 1024 bins
            freq_binned = freq / upper_frequency * (2 ** frequency_bits)
            other_freq_binned = other_freq / upper_frequency * (2 ** frequency_bits)

            # Produce a 32 bit hash
            # Use bit shifting to move the bits to the correct location
            hash = int(freq_binned) | (int(other_freq_binned) << 10) | (int(diff) << 20)
            hashes[hash] = (time, song_id)
    return hashes

# Quickly investigate some of the hashes produced
hashes = create_hashes(constellation_map, 0)
for i, (hash, (time, _)) in enumerate(hashes.items()):
    if i > 10:
        break
    print(f"Hash {hash} occurred at {time}")
```

Output:

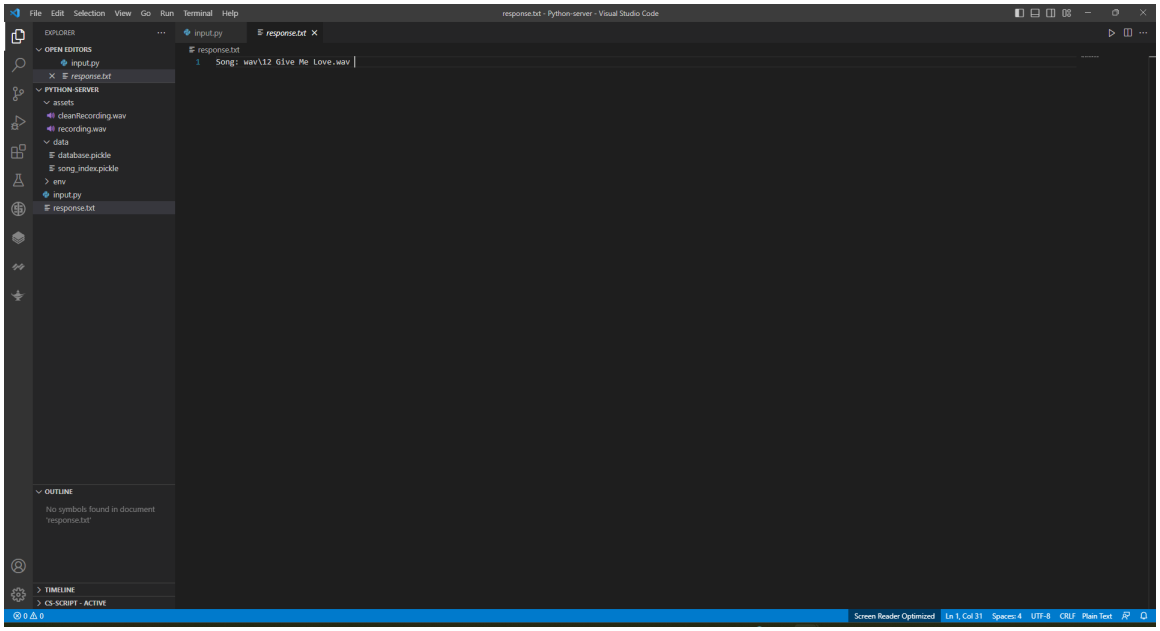


Figure 7.1., Output of the APP

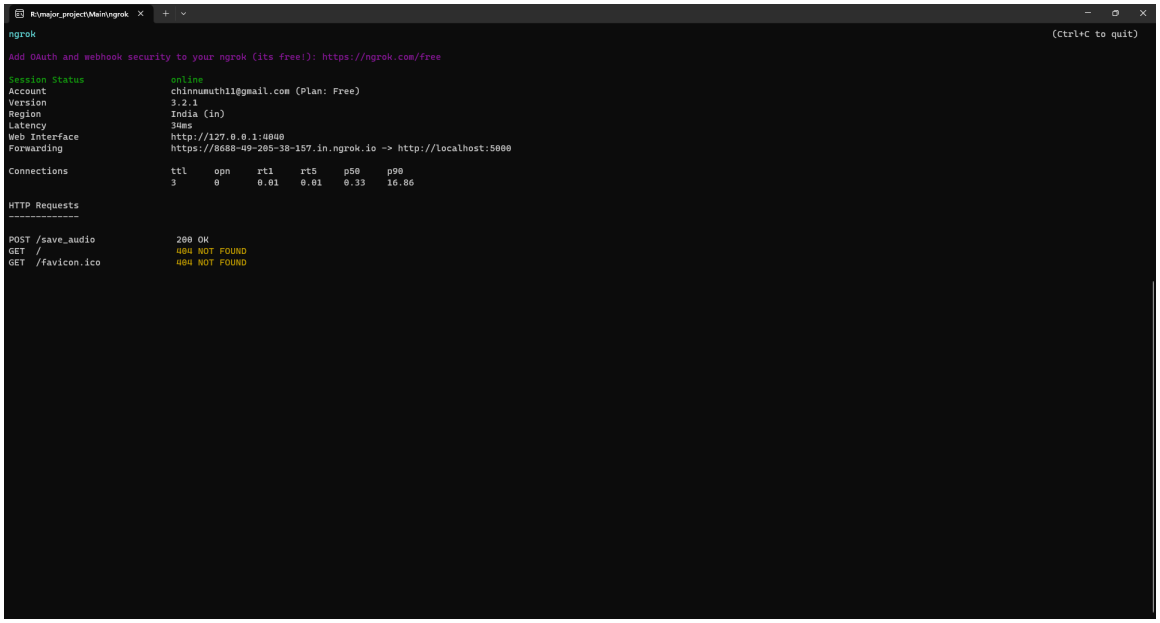


Figure 7.2., Server Connection

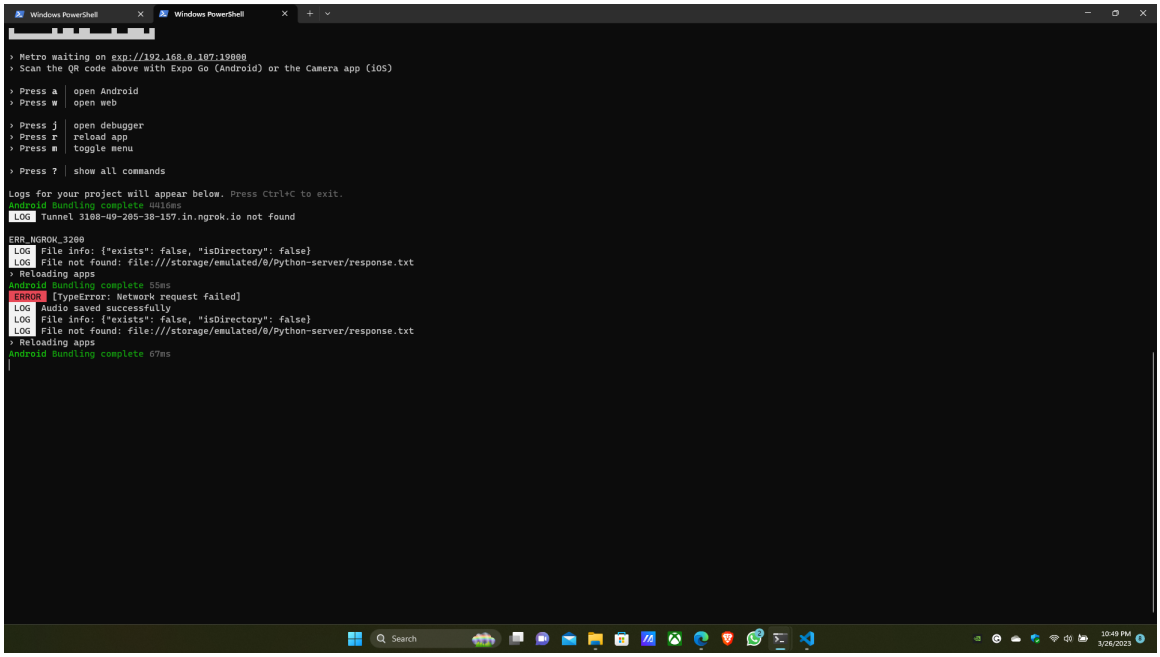


Figure 7.3., React js Server Connection

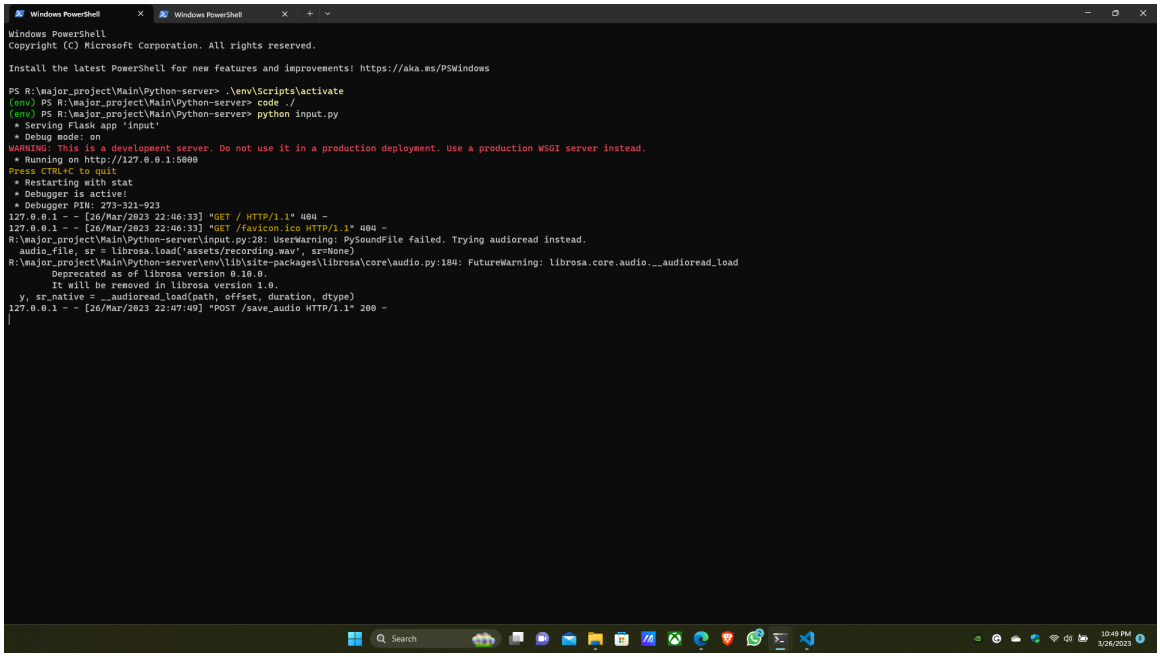


Figure 7.4., Python Server Connection

We can now use the standard SciPy peak-finding methods to find peaks in the spectrum. However, the very noisy makeup of the spectrum means that there are too many peaks to identify regions of actual interest that will be useful in developing fingerprints of the song.

Modules that we used:

Numpy - It is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays.

Scipy - It is a scientific computation library that uses Numpy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats, and signal processing.

Tqdm - Tqdm is a Python library used to display smart progress bars that show the progress of your Python code execution.

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

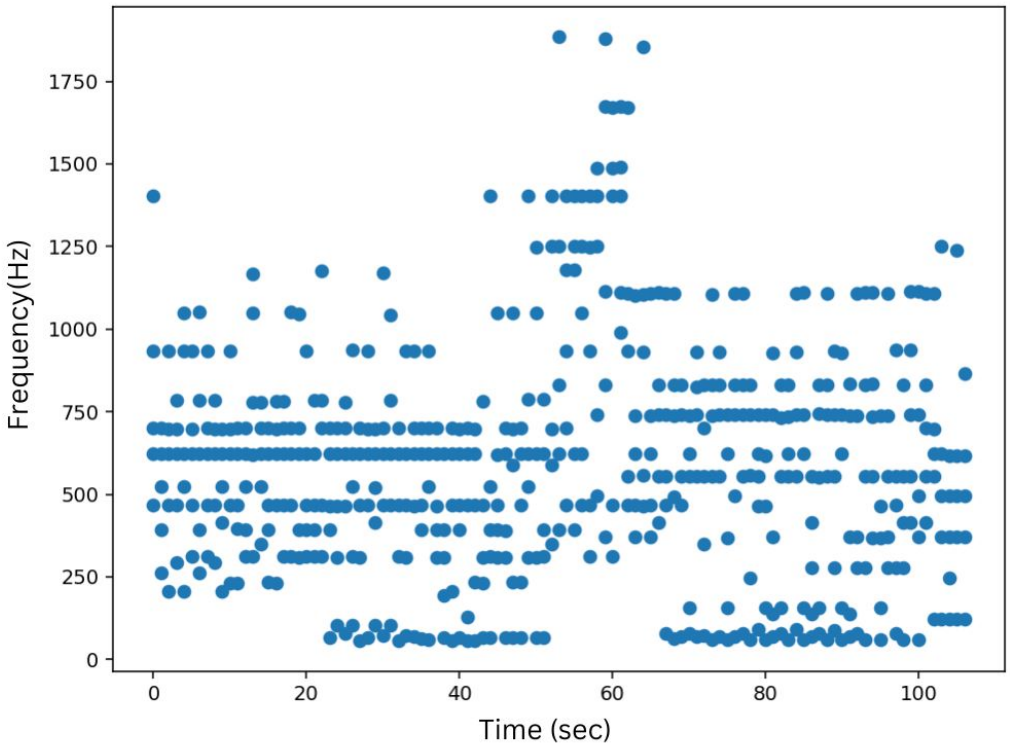


Figure 7.5., Characterizing song

We can then perform the same peak finding as before, and plot the results as a graph of time across the X-axis and the frequency of the peak on the Y-axis as shown in Fig. 7.5. This forms a constellation of points that characterize the song.

CHAPTER 8
TESTING AND RESULTS

CHAPTER 8 TESTING AND RESULTS

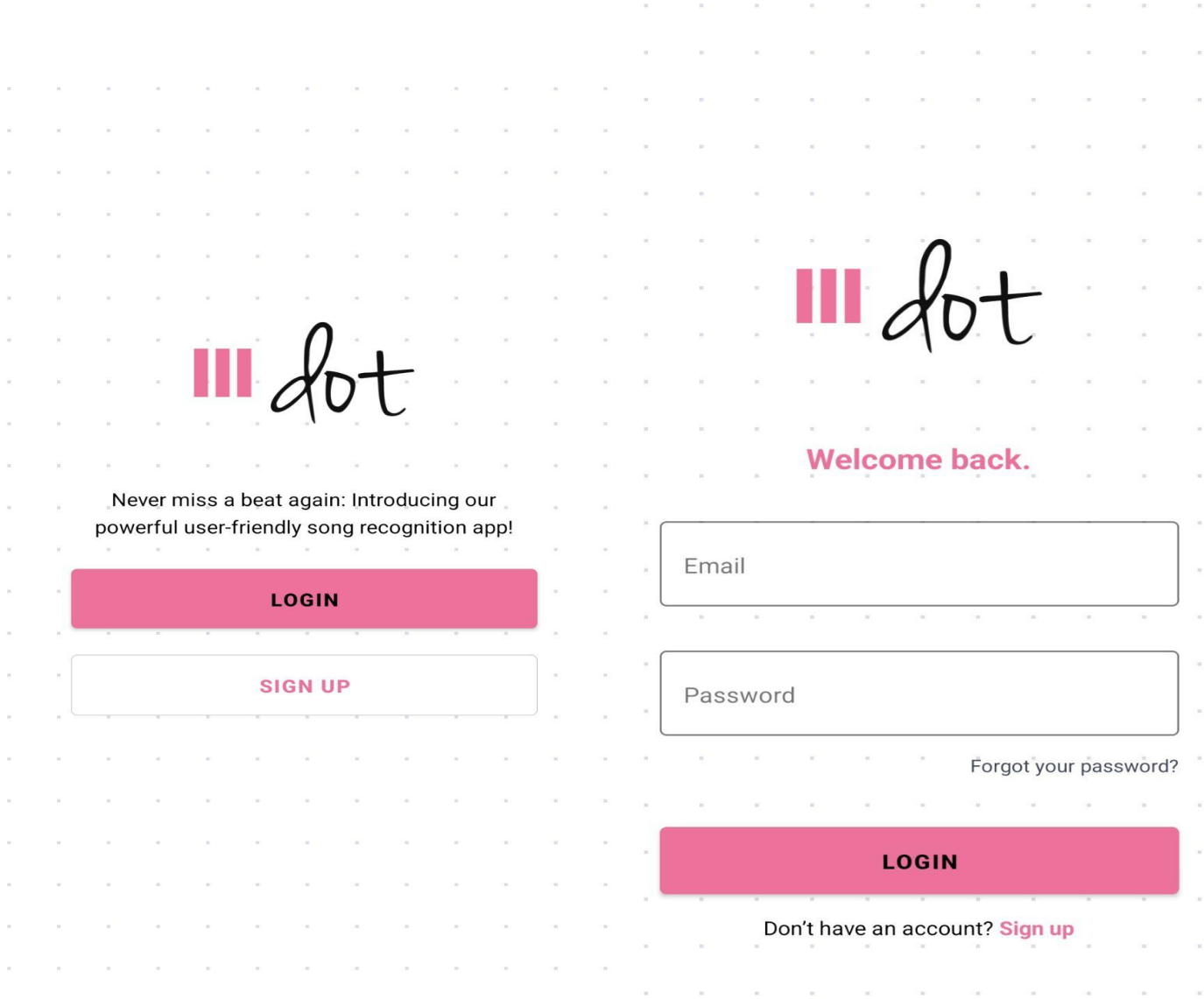


Fig 8.1., Frontend of the APP

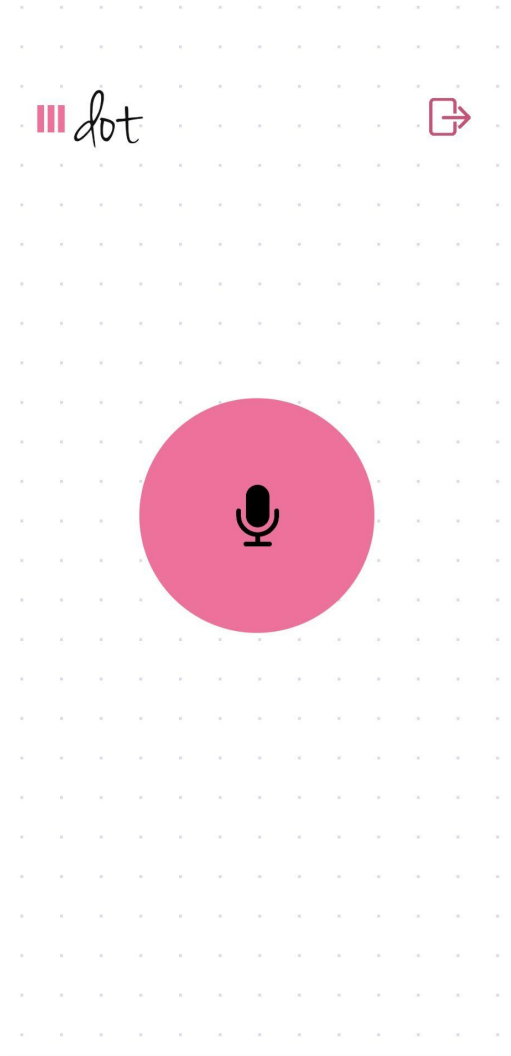


Figure 8.2., Frontend of the APP

CHAPTER 9

CONCLUSION AND FUTURE WORK

CONCLUSION

Audio fingerprinting have numerous applications, including music identification, content-based retrieval, copyright enforcement, and audio surveillance. It has become an important technology in the music industry, where it is used for music recommendation systems, royalty collection, and copyright protection.

While audio fingerprinting has many advantages, it also has some limitations. For example, it can be susceptible to noise and distortion, and it may not be able to accurately match audio recordings that have been significantly altered or remixed. Additionally, privacy concerns may arise when audio fingerprinting is used for surveillance or tracking purposes.

Overall, audio fingerprinting is a rapidly developing field with many exciting possibilities for the future of audio technology. As the technology continues to improve, we can expect to see more sophisticated and accurate audio fingerprinting algorithms that are capable of handling a wider range of audio signals and applications.

REFERENCES

[1] G. Deepsheka, R. Kheerthana, M. Mourina and B. Bharathi, "Recurrent neural network based Music Recognition using Audio Fingerprinting," 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT), 2020, pp. 1-6, doi: 10.1109/ICSSIT48917.2020.9214302.

[2] M. Mueller, A. Arzt, S. Balke, M. Dorfer and G. Widmer, "Cross-Modal Music Retrieval and Applications: An Overview of Key Methodologies," in IEEE Signal Processing Magazine, vol. 36, no. 1, pp. 52-62, Jan. 2019, doi: 10.1109/MSP.2018.2868887.

[3] C. Ouali, P. Dumouchel and V. Gupta, "Fast Audio Fingerprinting System Using GPU and a Clustering-Based Technique," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 24, no. 6, pp. 1106-1118, June 2016, doi: 10.1109/TASLP.2016.2541303.

[4] P. Seetharaman and Z. Rafii, "Cover song identification with 2D Fourier Transform sequences," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 616-620, doi: 10.1109/ICASSP.2017.7952229

FUNDING AND PUBLISHED PAPER DETAILS

Letter of Acceptance

Details of Accepted Paper

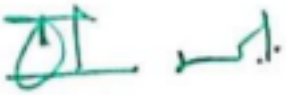
	ICPCSN-0175
	AUDIO-MUSIC FINGERPRINTING RECOGNITION
	Chinmaya Murthy,Chirayu S M,Dayanand Kavalli,Divya J
	Engineering (of Aff.)Computer Science Engineering (of Aff.) Dayananda Sagar University (of Aff.) Bengaluru, India

Greetings!!

We congratulate you on being successfully selected to present the aforementioned article at the "3rd International Conference on Pervasive Computing and Social Networking" on June 19-20, 2023.

Your research manuscript has been accepted after the peer-review process of ICPCSN-2023 for oral presentation and publication in ICPCSN-2023 proceedings.

In this regard, ICPCSN-2023 will give an unforgettable experience in exploring new research opportunities in Pervasive Computing and Social Networking.



Dr. Munusami Viswanathan
Conference Chair
ICPCSN - 2023

Previous Editions
ICPCSN 2021
ICPCSN 2022