Audio Fingerprinting with Python

Garvin Pokhra
Electrical and Electronics Engineering
National Institute of Technology Karnataka
Surathkal, India
garvinpokhra121@gmail.com

Nakshatra Gopi
Electrical and Electronics Engineering
National Institute of Technology Karnataka
Surathkal, India
nakshatragopi.191ee228@nitk.edu.in

Samiran Vanmali
Electrical and Electronics Engineering
National Institute of Technology Karnataka
Surathkal, India
samiranv14@ieee.org

Pranav M Koundinya
Electronics and Communication Engineering
National Institute of Technology Karnataka
Surathkal, India
pranavmkoundinya.201ec247@nitk.edu.in

Raghuram Kannan
Electronics and Communication Engineering
National Institute of Technology Karnataka
Surathkal, India
raghuramkannan400@gmail.com

***Abstract*—In this project, we have implemented a model to recognize a song from a database using the hashing method. We first record the audio file and convert it from time domain to frequency domain. Then we perform STFT to produce the spectrogram of the audio sample. From this 2-D array of STFT coefficients a matrix consisting of Peaks is created. The audio file is then fingerprinted using the hash function. The fingerprints from the unknown sample are matched against a large set of fingerprints stored in the database. Finally the fingerprints of the recording are compared with those in the database to return the best match. We have implemented our model in the Python programming language and MySQL. It is because Python offers versatility in handling and accessing the various data structures, which is extremely essential to our project.**

I. Introduction

Apps like Shazam can recognize a song from a large cluster of audios. This can't be done just by using brute force to compare an audio sample to every song in the database. This process seems to be unrealistic because of computational and storage constraints. In this paper, we have attempted to save the effort by the use of hashing functions to explore audio fingerprinting. An audio fingerprint is a condensed digital summary generated from an audio signal that can be used to identify an audio sample or locate similar items in an audio database. Here, the audio file is fingerprinted, a process in which reproducible hash tokens are extracted. A hash is a unique, encoded string of a pair of distinguishable peaks in an audio signal. Both database and sample audio files are subjected to the same analysis. The fingerprints from the unknown sample are matched against a large set of fingerprints stored in the database. This is supplemented by MySQL serving as a database management system to store the hashes. The fingerprints of the recording are compared with those in the database to return the best match. We have also worked on a search algorithm to identify the audio fingerprint of the given audio sample from a database of songs.

II. Implementation

A. Recording the audio file

An audio file is an acoustic signal which stores information as a function of amplitude against time. The audio file is recorded using a microphone at a specific sampling rate(the number of samples per second). In this project this is set to 44100. The total number of samples increases with the duration of the audio file. This audio file, prior to performing any operations on it, is loaded onto an array, where the array elements represent amplitude and subscripts/indices represent time.

B. Moving to the Fourier domain

The representation of the audio file in the time domain has little use. So, before proceeding any further in the process, the audio file is required to be represented in its frequency domain. This is achieved by taking the Fourier Transform of the file at its fundamental sampling rate. The Fourier Transform of any signal $x(t)$ is given by

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \qquad (1)$$

For a discrete-time signal such as an audio signal, however, the Fast Fourier Transform(FFT) is applicable. The plot of an audio file in the time domain, and its frequency domain counterpart is shown in Fig.1 and Fig.2 respectively. The frequency-domain representation of an audio file has, apart from facilitating audio processing, one more advantage over time-domain representation; it helps to separate the noise components in the file. Since noise comprises waves of higher

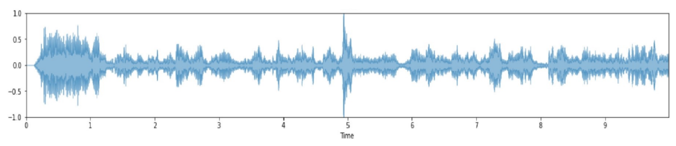


Fig. 1.

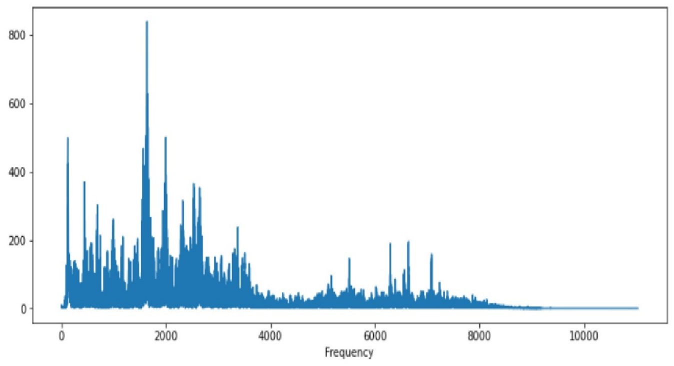


Fig. 2.

frequencies, by restricting our analysis and processing to lower frequencies, we can eliminate noise from the signal.

C. Simultaneous Visualisation of Time and Frequency- The Spectrogram

While the Fast Fourier Transform does, in principle, emphasise the significance of frequency domain representation of a signal, it is the Short-Time Fourier Transform(STFT) that is more useful in practice. STFT is the process of breaking the audio file into chunks of equal intervals, performing the Fourier Transform over each of them, and overlapping the same together. The process of STFT from the original audio file is illustrated in Fig.3 The STFT, just like FFT, has both real and imaginary parts. The spectrogram, however, takes into account only real values. So, a spectrogram is, in practice, a matrix of the magnitude STFT coefficients squared, and frequency components expressed in a particular scale such as linear, logarithmic, Mel, etc. The logarithmic-scale spectrogram of the audio file in Fig.1, is shown in Fig.4.
The spectrogram for a complete song in the database is shown in Fig.5.

D. Mapping the peaks

From the two-dimensional array of STFT coefficients, we need to map the points of local maxima by the method of masking; parsing a kernel of a predefined dimension over the entire array, and setting the points of local maxima to TRUE(1) and all other points to FALSE(0). An example of masking of a 5x10 matrix by a 3x3 kernel is shown in Fig.6.

E. Hashing the Peaks-Towards Fingerprinting

In the matrix consisting of the peaks, each peak is iterated and paired with a set of peaks in its neighbourhood called

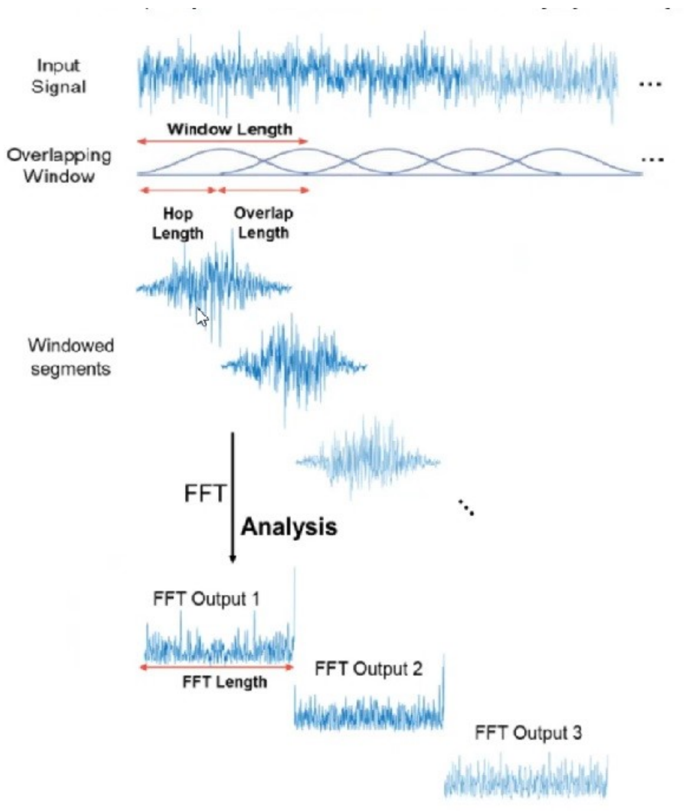


Fig. 3.

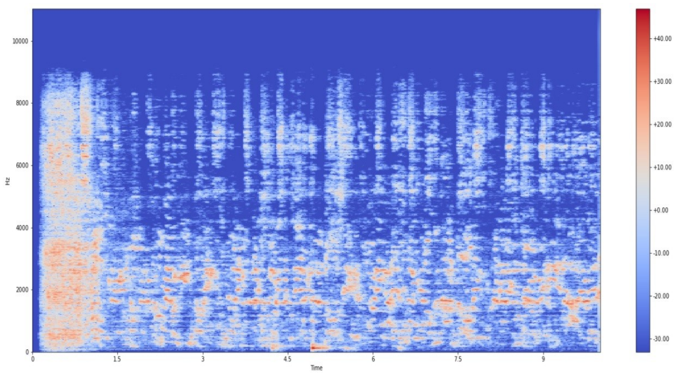


Fig. 4.

the target zone(Fig.7). The parameters in each pair namely, the frequency component of the iterated peak, the frequency component of the peak in the target, and the difference in the time component of the two peaks are passed to the hashing function which generates an encoded string that is unique to that input. This string is called the hash. The ordered pair, or the tuple consisting of hash and the frequency component(offset) of the iterated peak, is referred to as the audio fingerprint. Each instance a hash is generated, the constituents of the fingerprint i.e, the hash and the offset are appended to the database in their respective fields. When this process

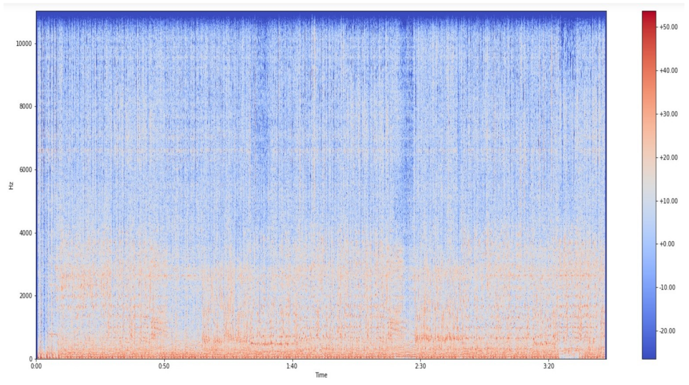
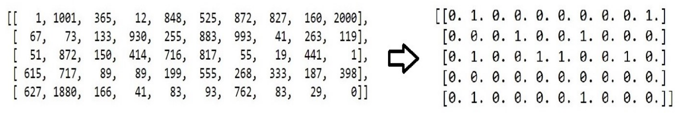


Fig. 5.



```
[[   1, 1001,  365,   12,  848,  525,  872,  827,  160, 2000],
 [  67,   73,  133,  930,  255,  883,  993,   41,  263,  119],
 [  51,  872,  150,  414,  716,  817,   55,   19,  441,    1],
 [ 615,  717,   89,   89,  199,  555,  268,  333,  187,  398],
 [ 627, 1880,  166,   41,   83,   93,  762,   83,   29,    0]]
```

```
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 1. 1. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

Fig. 6.

is performed during learning of the song, the song id of the peak is also added, to aid recognition.

F. Learning the songs and creating the song database

Learning of songs involves developing a database of fingerprints of complete songs. It is considered that the song is free of any noise. Learning the song involves all the aforementioned operations pertaining to a recording. Once this database is generated, we can proceed to the recognition.

G. Recognising the song- The Searching Algorithm

Prior to each instance of song recognising process, a list of empty dictionaries is created. The number of dictionaries in the list corresponds to the number of songs in the database.
Now, the hashes and corresponding offsets of both the song and the recording are imported from the database and stored in appropriate iterable data structures. Each hash from the recording is iterated over those in the database and compared. Whenever a match is found, the corresponding song id and the database offset(i.e, of the song hash from the database) are appended to a list in the form of a tuple. Once a hash from the recording is compared with all the hashes from the database, updating the dictionaries begins. For each tuple in the list, the dictionary corresponding to the song id of the tuple is added a key-value pair, where the key is the numerical difference between database offset and sample offset(i.e of the recording hash), and value is 1, which gets incremented each time the same tuple is repeatedly present in the list. That is, if the same tuple is repeated in the list, then instead of appending the dictionary with the same key-value pair, the previously existing key-value pair is updated by incrementing value by 1. So if a tuple (SongId_i, Diff) is repeated n times in the list, then the 'i'th dictionary would contain the element $\{$Diff : n$\}$.

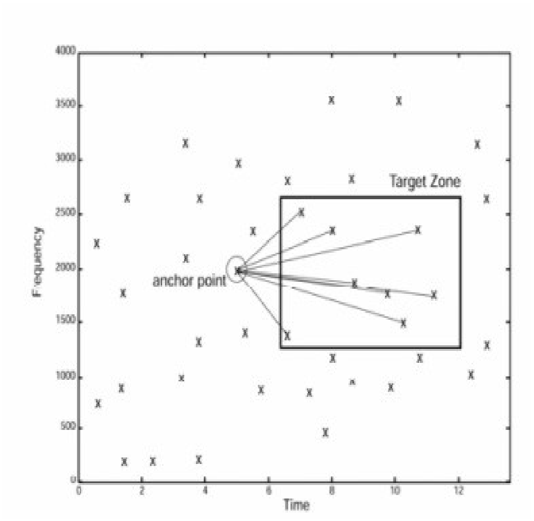


Fig. 7.

Before the iteration for the next hash(of the recording) begins, the list is emptied to accommodate the tuples from the following iteration. This process is carried out in each iteration.

At the end of looping, the maximum value out of all the key-value pairs in a dictionary is taken as the score for that particular dictionary. The song id of the dictionary which has the maximum score is taken to be the matching song for the recording. The name of the of the song is back-traced from the database and is printed along with the appropriate message.

III. Experiments, Trials and Results

For trials conducted over recordings of 10 seconds, the results were fairly accurate and promising. The time taken to recognise the song varied from recording to recording, but on average, was 120 seconds. The trials which yielded negative results were usually for songs that shared similar acoustic characteristics such as beats, frequencies, among others. In such cases, however, the score for the actual correct song was just behind the incorrectly matched song.

Acknowledgment

First, the team would like to thank IEEE NITK for giving us an opportunity to work on this project. The members would like to thank the mentors Mr Garvin Pokhra and Ms Nakshatra Gopi for their continued guidance and support throughout this project. The team would also acknowledge the help and support of the mentors of the Knowledge Sharing Sessions Mr Nishant Nayak, Mr Anirudh Achal and others who helped us understand the required prerequisites for this project and those who gave us the inspiration to do this work. The team would also like to thank Mr Vinay Kumar N for his valuable input during the course of this project.

References

[1] An Industrial-Strength Audio Search Algorithm, Avery Li-Chun Wang, Shazam Entertainment, Ltd.

[2] FPGzAm – Song Identification System, Pranav Sood and Yafim Landa - http://web.mit.edu/6.111/www/f2010/projects/landa_Project_Proposal.pdf.

[3] Audio Fingerprinting with Python and NumPy by Will Drevo. https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/

[4] How Shazam Works - https://youtu.be/kMNSAhsyiDg.

[5] Peter Sobot on An Industrial-Strength Audio Search Algorithm https://youtu.be/WhXgpkQ8E-Q.

[6] Understanding STFT - https://youtu.be/-Yxj3yfvY-4 .

[7] How to use Python with MySQL - https://medium.com/@tattwei46/how-to-use-python-with-mysql-79304bee8753.4

[8] Getting started with MySQL: https://www.w3schools.com/sql/.

[9] Fingerprinting with Hash Functions - https://youtu.be/hssoS0BqPnc