Shazam Competition

1. Introduction

The Shazam Beta version builds a simplified version of Shazam program: it pre-processes the songs in the database, extracts the feature and builds a hash table. Then for any clip which we want to identify, the Beta program follows the same procedures, matches in the hash table, and identifies the clip by finding the similar features.

Now in the Shazam competition, we are trying to improve the Beta program such that it is more robust against variant noises and more time efficient. For example, we may add some filters in the preprocess step and use some efficient data structure in the database. The details about the competition procedure is discussed below.

2. Submission

A. You may work in a two students group in this final project. Students in the same group will receive the same credit.

B. You need to create a zip folder named by your netID and your partner's netID. Put all your Matlab scripts of functions and tables listed below in the folder and zip it. Each group needs to upload one zip file named by one netID and your partner's netID in Blackboard.

C. There is no constraint with your Shazam design. But at least, the required Matlab files are:

main.m:

Required Syntax: `function songName = main(testOption,clipName)`

The main function will first decide which test option: 1 for database test and 2 for real-time test. If test option is 1 (database test), then it will receive additional input of the name of clip to identify (clipName). For the other test option of 2 (real-time test), it does not need the additional input of the name of clip, but it will instead record a 30 seconds-long clip coming out from a speaker. With the clip file, the main function loads songNameTable.mat and hashTable.mat. Then the main function calls the matching function (matching.m), which will return an identified song name by the program.

- Tip: For the real-time test, the clip coming from a speaker is at most 30 seconds long. However, it depends on you how long you want to record the clip and use Shazam to identify the clip (so that you can reduce time that your Shazam program requires to identify a clip).

make_table.m:

Suggested Syntax:

```
function table = make_table(testOption,song,gs,deltaTL,deltaTU,deltaF)
```

The make_table function extracts peak pairs and store them in the matrix, named table. Please refer to the Shazam Part I document for the details.

make_database.m:

Suggested Syntax:

```
function make_database(gs,deltaTL,deltaTU,deltaF)
```

make_database function will applies the make_table function to all the songs in the folder "songs" and build an efficient table, such as a hash table. Then it saves the resulting table as hashTable.mat. Also, make_database function saves all the names of songs into songNameTable and save it to songNameTable.mat. Please refer to the Shazam Part I document for the details.

hash.m:

Suggested Syntax:

```
function hashTable = hash(table)
```

Please refer to the Shazam Part I document for the details.

matching.m

Suggested Syntax:

```
function songName =
matching(testOption,clip,hashTable,songNameTable,gs,deltaTL,deltaTU,deltaF)
```

First the matching function applies the make_table function to the clip. Then the table of the clip is converted into the hashTable. Through comparing between the hashTable of the database and the hashTable of the clip, the function calculates a mode or a spike in the histogram. Using this information, the function returns an identified song name.

** Note: The function also measures **a confidence score**, which measures how confidence the program is for identifying the clip. A method for measuring the confidence score is all your choice! Depending on this confidence score, your program shall decide whether to return a predicted song name of the clip or not to return the song name "no decision" (i.e. if confidentiality is low).*

hashTable.mat:

The matrix hashTable in hashTable.mat stores the hash value, t_1 and songID, where songID is the order of the song in the folder. For example: [$h(f_1,f_2,t_2-t_1)$, t_1, songID ; ...]. Please refer to the Shazam Part I document for the details.

songNameTable.mat:

The cell musicTable in musicTable.mat stores the names of songs in order (01.mat, 02.mat, 03.mat, …).

D. Upload your zip file by **5PM Dec 8th Tuesday** in Blackboard.

E. In Blackboard, we have provided test mat files (database_test.m and realtime_test.m) that we will use for testing, including some sample songs that we will use for testing. **Make sure to run and check before you submit.**

3. Grading

The grading will be separated into two parts:

A. Database Test (60%):

We will sample a 15 seconds-long clip from randomly chosen 20 songs in the database with random starting point and add Gaussian noise with different power level. These clips will be pass to your Shazam. Your Shazam should return the name of the clip and we will check whether it is correct or not. Your credit will be based on the following equation:

Database test credit = 0.60 * matching rate

(*matching rate will be determined by: per one run, if correct 1 point; incorrect 0 point; and no-decision: 0.5 point*)

B. Real-time Test (30%):

We will play three 30 seconds-long clips using a speaker. When a clip is playing your code should record the clip and try to match it using your table. Each clip will be 30 seconds long and your main function should give the answer before it ends. If it does not give any response by the end, we will consider this matching as failed (incorrect).

- The first clip will be from 10.mat without adding noise.
- The second clip will be random chosen song with added sinusoid noise that has frequency of 2000Hz.
- The third one will be random chosen song with a sinusoid noise that has some random frequency.

Your credit will be based on the following equation:

Real time test credit = 0.30 * matching rate

(*matching rate will be determined by: per one run, if correct 1 point; incorrect 0 point; and no-decision: 0.5 point*)

C. Time Performance (10%):

For the Database test, we will also measure time performance (i.e., average time it took to identify 20 songs). Shorter time it took to identify wins. Your credit will be based on the following equation:

Time performance credit = time rank

(*time rank will be determined by: 1st group gets full credit of 0.10 and the last group gets 0.02*)

4. Design Considerations

A. Preprocessing: Filters

For pre-processing we can consider filters. For the 2nd part of the competition, we will add some sinusoid noise into a clip. So one possible filter to reduce sinusoid noise could be implementing the Notch filter that we learned in Lab 4. But of course there are many other filters that we could use, such as low-pass filter, bandpass filter, etc. What could be a smart filter for reducing sinusoid noise or even background noise? For the second clip in test Part B, we know the noise frequency. But what about the random frequency in the third clip?

B. Spectrogram: Window Size and overlap.

When we apply the spectrogram to the signal, we fix the window size and the overlap. The larger the window size is, the fewer data we will have. The smaller overlap we set, the less data there will be. Fewer data will surely improve the time performance of the code but it will also reduce the accuracy of the matching.

Required: Try different combination of the window size and the overlap. Use the combination you found that will make sure high accuracy (close to 100%) and the data size is small.

C. Feature Extraction: Spectrogram Local Peaks: Local Troughs and Window Size.

Previously, we used the local peaks to construct the boolean matrix P. An alternative way is to use local troughs. We can also change the window size. The larger the box is, the more peak pairs we will have. But this will increase the size of the table and make the matching slower.

Required: Given the log spectrogram S, get the boolean matrix P in the same size. Optimize the box size such that the matching rate of provided sample is high while the required matching time is short.

Optional: Try to use troughs. What is the matching rate if we use troughs instead of peaks?

D. Thresholding: Adaptive Threshold

In the Beta program, we set a fixed threshold to the entire song to get a peak rate of 30 peaks/sec. Instead, we can use an adaptive threshold, where we advance along the matrix in 1 second chunks of columns and adaptively threshold each of these chunks to approximately get our 30 peaks per second. Or you can do this in your smart way. I know some of you have another way to filter the peaks.

Optional: Find a threshold on the magnitude of the log spectrogram S for each 1 second chunk such that we get 30 peaks in each. Or filter the peaks in your genius way.

E. Efficient data structure

When storing a large size of database, we used a hash table in the Beta program. But there are other data structure that may help to save memory and improve the searching performance.

Required: Given a big table of songs peaks, apply the hash function and store it in matrix hashTable. Save it to hashTable.mat

Optional: Try to use efficient data structure to save the table. For example, we can use hash table.

F. Any other genius method

You can apply whatever steps to the code to make it time efficient while guarantee matching accuracy. Make sure all the functions work well.