Shazam Part I

1. Introudction

The Shazam final project integrates all materials you've learned this semester. In Part I, we will design a Beta version of Shazam: the Beta version identifies a clip, which has no noise/interference. The Beta version and actual version of Shazam follow a similar procedure. As shown in Fig. 1, the basic procedure is:

1. Construct a database of features for each full-length song.
2. When a clip (part of one of the songs in the database) is identified, calculate the corresponding features of the clip.
3. Search the database for a match with the features of the clip.

The features for each song (and clip) will be pairs of proximate peaks in the spectrogram of the song or clip. We start by finding the peaks in the log-spectrogram; plotting the locations of these peaks gives a "constellation map." Each peak has a time-frequency location (t, f) and a magnitude A. We then form pairs of peaks that are within a pre-specified time and frequency distance of each other and record the details of these pairs in the database. For example, if we obtained a pair (f_1,t_1) and (f_2,t_2), we might record (f_1,f_2,t_1,$t_2 - t_1$, songid). Notice the matching is not amplitude-based because the amplitudes of peaks are not robust.

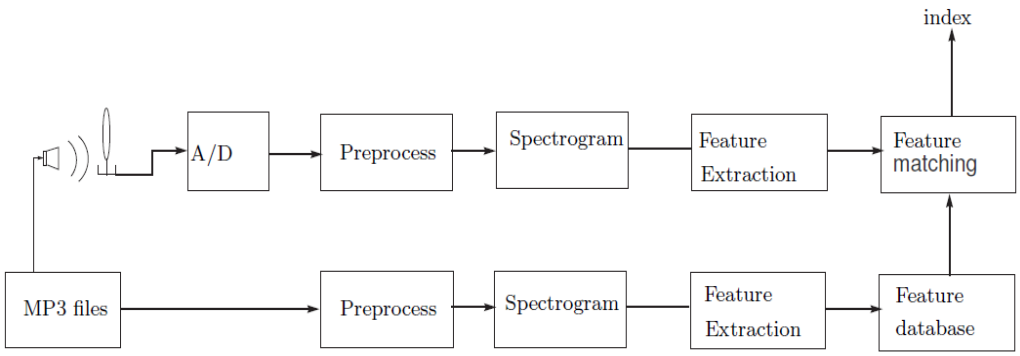


Figure 1: Shazam Diagram

Each song is summarized ("fingerprinted") by a big table of its extracted features:

$$\left|\begin{array}{cccc|c} f_1 & f_2 & t_1 & t_2 - t_1 & \text{songid} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_j & f_k & t_j & t_k - t_j & \text{songid} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_m & f_n & t_m & t_n - t_m & \text{songid} \end{array}\right|$$

When we are given a clip to identify, we do the same feature extraction for the clip. The result is a small table of clip features:

$$\begin{array}{|c|c|c|c|} f_1 & f_2 & t_1 & t_2 - t_1 \\ \vdots & \vdots & \vdots & \vdots \\ f_j & f_k & t_j & t_k - t_j \\ \vdots & \vdots & \vdots & \vdots \\ f_m & f_n & t_m & t_n - t_m \end{array}$$

We do not know the start time of the clip within its corresponding song. The times t_j that appear in the clip table are relative to the start of the clip. The clip itself starts at some offset t_0 from the beginning of the music. Finding a match to the clip in the data base is a matter of matching the constellation map of the clip to the constellation maps of the songs by effectively sliding the former over the latter until a position is found in which a significant number of points match. Using pairs of peaks as features gives us three quantities that we expect to independent of the unknown offset time: $(f_1, f_2, t_2 - t_1)$. The song with the most triples $(f_1, f_2, t_2 - t_1)$ in common with the clip table is likely to be the source of the clip.

2. Overview

Part I of Shazam is composed of several Matlab files. There must be a main.m file that combines what you will build in this instruction. It is suggested that you follow the sequence and specifications laid out in this instruction but you have the freedom to design your project as long as you maintain functionalities. You are required to write total of 5 separate Matlab files (make_table.m, make_database.m, hash.m, matching.m, main.m).

- make_table.m extracts peaks and creates a table from a song in the database.
- make_database.m uses make_table.m to create an efficient hash table for all the songs in the database. It also creates a table of song names.
- hash.m combines the parameters from make_table.m into a unique feature that can be used to differentiate between songs.
- matching.m compares two hash tables of the clip and the database.
- main.m takes an input clip and identifies it.

3. Submission

- **Due Time: TBD.**
- You will work in a two students group in this final project. Students in the same group will receive the same credit. You may work with students outside of your lab session.
- Grading Policy: We will test your submitted files with 10 random .mat files in the database provided. If we see that you've successfully matched 5 or more songs, you will receive full credit. If you match under 5, then there will be a penalty to your final Shazam competition score.
- You will submit a zip file named with the netIDs of the two menbers in your group (ex. cw733_kjj34.zip) containing all 5 Matlab files with hashTable.mat and songNameTable.mat.
- In Blackboard, we will provide a part1Test.m file that is similar to the one we will use for grading your program. **Make sure** to run and **check** before you submit.

4. make_table.m

First, we will learn how to preprocess the .mat files, how to draw the spectrogram and how to extract the features. Download the zip file from Blackboard. In folder named "songDatabase", there are total 50 .mat files of songs. For explaining make_table.m, we will use only one song, sample.mat.

Let's start to write a function `make_table`. A suggested function header could be

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

where `songName` is the song name and other parameters will be specified later. Given the name of the song, sample.mat for example, `make_table` will take the name as input and construct a table of features for that song. The function will do the following (details will be discussed in the following sub-section):

1. Load in the sample.mat file in the folder "songDatabase" and resample it.
2. Take the (log magnitude) spectrogram of the song using spectrogram.
3. Find the local peaks of the spectrogram.
4. Threshold the result of step 3 to end up with peak rate peaks/sec (30 peaks per second).
5. For each peak, find "fan-out" pairs in the "target window" and then add to the table.

Step 1. Preprocessing

Read the sample.mat file. The signal contains two channels, but for our purposes it is sufficient to take only one channel – use the first channel. Since the loaded mat song has sample frequency `Fs` of 44100 Hz, we have a lot more data than we need. **Resample the signal at 8000 Hz using the command `resample`.** This command performs an interpolation of the signal at the new sampling points and returns the result. And we will work on the resampled signal from now on.

What to do: Given a song name, sample.mat for example, resample it at 8000 Hz and store the signal.

Hint: Use the following syntax for loading mat file: `load(songName,'-mat');`

Check: What is the length after you resample the file sample.mat in the database? It should be 240001.

Step 2. Spectrogram

Now we construct the spectrogram of the first song in the database, sample.mat. Call the spectrogram command as follows:

```
[S,F,T] = spectrogram(resampledSong,window,noverlap,nfft,Fs);
```

where:

`resampledSong` is the resampled song.

`window` is an integer that indicates the length of the chunks you want to take the fft of.

`noverlap` is the number of samples you would like to have as overlap between adjacent chunks.

`nfft` is the length of the fft, i.e. the resolution of frequencies, which in our case, it is the same as window.

`Fs` is the sampling rate of the signal, in our case 8000 Hz.

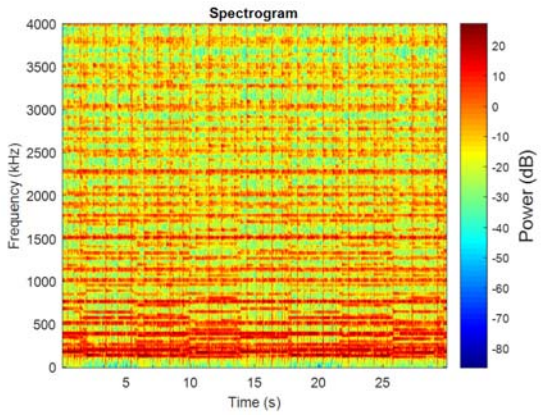
The function returns the spectrogram in the matrix `S` with just the positive frequencies. The frequency vector for the vertical axis is returned in `F` and the time vector for the horizontal axis is returned in `T`.

Compute the spectrogram with **window length of $64*10^{-3}$s, an overlap of $32*10^{-3}$s, and nfft of $64*10^{-3}$**. Note that the number of samples in a window is simply the window length multiplied by our sampling rate. Be sure to specify both these parameters as an integer number of samples.
Then take the absolute value of `S`, add one, and take the log 10: `log_S = log10(abs(S)+1);`. We shall use the log magnitude spectrogram.
What to do: Given the resampled signal y, get the log 10 spectrogram S with window size $64*10^{-3}$s, *overlap* $32*10^{-3}$s, and nfft $64*10^{-3}$s.

Check: Plot the magnitude of the spectrogram using the following code. The result should be similar to:

```
% Step 2 Check
figure
imagesc(T,F,20*log10(abs(S)))
axis xy;
xlabel('Time (s)')
ylabel('Frequency (Hz)')
title('Spectrogram')

colormap jet
c= colorbar;
set(c);
ylabel(c,'Power (dB)','FontSize',14);
```



Step 3. Feature Extraction (Spectrogram Local Peaks)
Next, we find the local peaks of the spectrogram. A local peak has log magnitude greater than that of its neighbors. One way to find the local peaks is to iterate through each point in the spectrogram and compare the magnitude to the magnitude of each of the points in the surrounding g_s x g_s grid. This can be done for all points at the same time by using the command `circshift`. For example, if log_S denotes the log magnitude spectrogram, then the code

```
1.  CS = circshift(log_S,[0,-1]);
2.  localPeak = ((log_S-CS)>0);
```

returns a boolean matrix localPeak with entries 1 for the positions in S that are greater than their neighbor immediately to the right (see help `circshift` for details). If you put this structure in a loop, you can select the points in log_S that are greater than all of their neighbors in the g_s x g_s grid, i.e. the local peaks. The location of these peaks are stored in a Boolean matrix localPeak of the same size as S.

What to do: Given the log spectrogram log_S, get the Boolean matrix localPeak in the same size, where if the entry in log_S is a local peak, the corresponding entry in localPeak is 1, otherwise 0. Put gs = 9.
Check: there should be total 2,078 peaks (use command `nnz`).
Another check: to test whether your code is working properly, put the log spectrogram log_S = [1 1 1 3 ; 1 10 1 2 ; 1 3 1 5 ; 2 11 1 2]. Set gs=3 for this case, then localPeak should be: [0 0 0 1 ; 0 1 0 0 ; 0 0 0 1 ; 0 1 0 0]

Step 4. Thresholding
Among the local peaks, we want to use only the larger peaks. To select the larger peaks and also to control the average rate of peak section (peaks per second), we have to do some sort of selection operation on the peaks. The simplest thing would be to apply a fixed threshold to the detected peaks and keep only those above the threshold. The threshold could be selected to yield (approximately) the desired number of peaks. Assume that we want approximately 30 peaks per second. Find a threshold which yields that rate of peaks.

What to do: Find a threshold on the magnitude of the log spectrogram `S` such that on average there are 30 peaks per second. Apply the threshold to get a new `localPeak`.
Check: threshold value for sample.mat should be around 0.5883.
Another Check: after thresholding, number of non-zeros in `localPeak` for sample.mat should be around 900. You want to make this procedure automatic such that it will work for all songs in the database.

Step 5. Constructing the table

As outlined in the beginning, we want to select pairs of peaks and record the frequency of each peak, the time of the first peak and the time difference between the two peaks.

A peak-pair must satisfy certain constraints: the second peak must fall within a given distance from the frequency of the first peak and the second peak must occur within a certain time interval after the first peak. We will also limit the number of pairs allowed to form from a given peak. This is called the fan-out. For now, let's set set fan-out to be 3.

So a peak located at index $(\mathrm{t}_1,\mathrm{f}_1)$, can only be paired with peaks which have $\mathrm{t}_1 + \Delta_t^l \leq \mathrm{t}_2 \leq t_1 + \Delta_t^u$ and $\mathrm{f}_1 - \Delta_f \leq \mathrm{f}_2 \leq f_1 + \Delta_f$ for some Δ_t^l, Δ_t^u, and Δ_f. See Figure 2.

Note that here f_1 and f_2 are the indices of the frequencies in the spectrogram matrix, but not the exact value in Hz. Therefore Δ_f is the difference in indices. It is the same for the time t. i.e., you need you find the paired peaks of $(\mathrm{t}_1,\mathrm{f}_1)$ within the $(\Delta_t^u - \Delta_t^l) \times 2\Delta_f$ grids.

Our objective is to select appropriate parameter values for Δ_t^l, Δ_t^u, Δ_f and fan-out, and then record the 4−tuples $(\mathrm{f}_1,\mathrm{f}_2,\mathrm{t}_1,\mathrm{t}_2-\mathrm{t}_1)$ in a matrix.

What to do: For the new localPeak matrix, find the 4-tuples (f_1,f_2,t_1,t_2-t_1) with $\Delta_t^l = 3$, $\Delta_t^u = 6$, $\Delta_f = 9$. These parameters should be able to be set through `deltaTL`, `deltaTU` *and* `deltaF` *outside the function make_table.*

Hint: The Matlab commands find and ind2sub might come in handy.

Hint: For the fan-out, you might want to use find([Your code goes here],fanOut), where fanOut = 3.

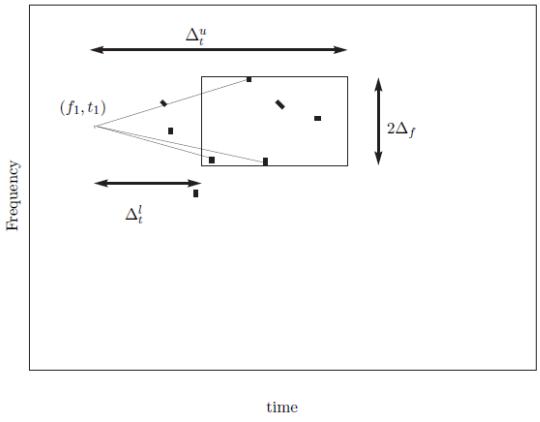


Figure 2: Peak pairs

Step 6. Final Function

Now, combine everything into a function `make_table` which puts all of these steps together. It will take as input the song signal and it will return an npairs ×4 matrix which contains in each row the 4−tuple corresponding to a peak pair:

$$\begin{array}{|c|c|c|c|} f_1 & f_2 & t_1 & t_2 - t_1 \\ \vdots & \vdots & \vdots & \vdots \\ f_j & f_k & t_j & t_k - t_j \\ \vdots & \vdots & \vdots & \vdots \\ f_m & f_n & t_m & t_n - t_m \end{array}$$

When you are constructing this table, use the indices of the spectrogram as the values for f and t instead of using the corresponding Hz and seconds values.

What to do: Store all 4-tuples $(f_1, f_2, t_1, t_2 - t_1)$ *in a matrix called table. You may want to iterate it in P column by column. You do not need to rotate when you are at the edge of the matrix localPeak.*

Check: there should be close to 193 rows in the table.

5. make_database.m

Suggested Syntax:

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

1. Apply `make_table` to each song in the folder "songs" so we have a large table as described in sec 3.3.
2. Apply the `hash` function (described later) to the table to convert it from 4 columns to 3 columns and save it into the table named as `hashTable`. One example could be [h(f_1,f_2,$t_2 - t_1$) t_1 songID ; ...]. The songID is the order of the song in the database. Use `save` command to save the hash table into `hashTable.mat`.
3. Initialize `songNameTable` and store the name of the music into it in order. For example: [01 ; am859_music ; ...]. Save the `songNameTable` into `songNameTable.mat`.

We will learn how to build the database (make_database.m). Now you have the function `make_table` that takes as input a clip and returns a table of peak pairs.

$$\left|\begin{array}{c|c|c|c} f_1 & f_2 & t_1^c & t_2^c - t_1^c \\ \vdots & \vdots & \vdots & \vdots \\ f_j & f_k & t_j^c & t_k^c - t_j^c \\ \vdots & \vdots & \vdots & \vdots \\ f_m & f_n & t_m^c & t_n^c - t_m^c \end{array}\right|$$

And you can run make_table on each song in the database and you can produce a large table.

$$\left|\begin{array}{c|c|c|c|c} f_1 & f_2 & t_1^s & t_2^s - t_1^s & \text{songid} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_j & f_k & t_j^s & t_k^s - t_j^s & \text{songid} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_m & f_n & t_m^s & t_n^s - t_m^s & \text{songid} \end{array}\right|$$

If the clip comes from a particular song, we expect each entry in the clip table to have a corresponding entry in the song table. In particular, if (f_1, f_2, t_1^c, $t_2^c - t_1^c$) is an entry in the clip table and (g_1, g_2, t_1^s, $t_2^s - t_1^s$) is its corresponding entry in the song table, then we should have $f_1 = g_1$, $f_2 = g_2$, and $t_2^c - t_1^c = t_2^s - t_1^s$ because these features are time and shift invariant. So using the triple (f_1, f_2, $t_2^c - t_1^c$) from a clip table entry is a good way to search for a match from the set of song tables.

We need to apply `make_table` for all the songs in the folder "songs". You may find these code useful:

```
Files = what('/songDatabase);
matFiles = files.mat;
filename = matFiles{1};
toRead = ['songDatabase/',fileName];
```

All the names of files in the folder "songDatabase" are stored in the cell `matFiles`. And the directory of the first song is now stored in `toRead`.

What to do: Apply `make_table` to all 50 songs in the folder "songDatatbase" and store the feature in a large 5 columns table, like [$f_1, f_2, t_1, t_2 - t_1$,songID;...].

6. hash.m

Suggested Syntax:

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

Since each frequency can be represented by a number from 0 to 255 (assuming for convenience that we drop the last (highest) frequency bin) we can represent each frequency with 8 bits. Depending on how large a time window we allow for our target box, we can also represent the time $t_2 - t_1$ by a *n* bit number, for some small *n*. Usually a hash function maps large data sets to smaller data sets, thus different inputs might be mapped to the same hash value. For this lab, to make things easy, we use a simple hash function of the form:

$$h(f_1, f_2, t_2 - t_1) = (t_2 - t_1)2^{16} + f_1 2^8 + f_2$$

(make sure that f_1 and f_2 range from 0 to 255, not 1 to 256. You should subtract f_1 and f_2 by 1).

You can run `make_table` to each song to form a large table and then apply hash function to the large table.

What to do: Given a big table of songs peaks, convert it into the form of $[h(f_1,f_2,t_2 - t_1)\ t_1$ *songID ; ...] and save it as* `hashTable`*.*

7. matching.m

Suggested Syntax:

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

Function `match`: Take the file name of the clip and try to match it to one of the songs in the database and return the matching name. You need to write your own codes.

1. Run `make_table` and `hash` on the input clip to produce a hash table for it. Use the given song "sample.mat".
2. For each h(f_1^c, f_2^c,$t_2^c - t_1^c$), look for it in the `hashTable`. When a match is found, record the time offset $t_0 = t_1^s - t_1^c$ and `songID` in a matrix `matchMatrix` like
   [t_0 songID ; t_0 songID ; ...]
3. After going through the entire clip table, there will be a collection of $t_1^s - t_1^c$ values for each song in the database. The song that the clip matches will have a large mode, or a spike in the histogram of $t_1^s - t_1^c$ values. Determine a match by using `mode`.

To identify a clip, we run the function **make_table** on the clip to generate a clip table of peak pairs. Then we search the database for matches to each entry in the clip table. We want to use the triples (f_1, f_2, $t_2^c - t_1^c$) to search the database. What we need is a fast way to determine if (f_1,f_2,$t_2^c - t_1^c$) matches something in the database and if so, extract what we know about that match.
Hash table will help to ensure fast database lookup. But for simplicity we will just combine entries of the table (f_1,f_2,$t_2^s - t_1^s$) into a more compact one using a hash function h(f_1,f_2,$t_2^s - t_1^s$) which will be detailed later. For each entry in the compact song table, we place the songid, the t_1^s, and the h(f_1,f_2,$t_2^s - t_1^s$).
Then given a peak pair entry in the clip table, say (f_1,f_2,$t_2^c - t_1^c$), we look up the database entry for h(f_1,f_2,$t_2^c - t_1^c$). There will be either hit, or there is one hit with (songid,t_1^s,h(f_1,f_2,$t_2^s - t_1^s$)= h(f_1,f_2,$t_2^c - t_1^c$)) or multiple hits.
All the matches from the clip table to the correct song should occur with the same difference $t_0 = t_1^s - t_1^c$, where t_1^s is the time of the pair in the song, and t_1^c is the time of the pair in the clip. The time $t_0 = t_1^s - t_1^c$ is the offset in time we would need for the clip constellation to line ip with the song constellation. This suggests that we find the songid that has the most matches occurring with the same offset t_0 and assert that our clip comes from that song, or rank the matches by this criterion.

What to do: Given a clip, apply the `make_table` function and apply the hash function to the clip table. Look the clip table up in the database (hashTable) and return the song name.

8. main.m

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

* testOption parameter is not used for Part I but will be used later. This can be any number for the moment.

Write a main function so that you can test your program out using the sample.mat.
In the main.m, you need to load the song. Set the matching parameter gs, deltaTL, deltaTU, and deltaF. Then call the matching function.

Question: What is the name of this clip? It should be 01.mat!