AUTOMATIC MUSIC IDENTIFICATION SYSTEM

Christian J. Steinmetz
Centre for Digital Music, Queen Mary University of London

ABSTRACT

The task of identifying the original recording given a short snippet recorded from a mobile device is a challenging task. This process requires not only robustness to distortions, but also requires an efficient and scalable search and indexing methodology that enables operating on catalogs of upwards of millions of tracks. In this project, we implement the popular method for audio identification based upon audio fingerprints, which are built from spectral constellation maps with an efficient combinatorial hashing algorithm. We outline our implementation and perform a simple parameter search for the optimal trade-off between accuracy and computational overhead. Our evaluation using recordings made from mobile devices based upon the GTZAN dataset reveals that our system can achieve 81% Top-1 accuracy, while remaining computationally efficient.

1. INTRODUCTION

A system with the ability to identify recordings based only upon short audio excerpts has a number of industrial applications. These include identifying copyrighted content on web platforms, tracking plays of music for compilation of music charts, as well as the identification of music playing over the radio, in a pub, or heard anywhere by a listener with a mobile phone. While such a system can bring significant value, the task brings with it many challenges. The identification system must be lightweight and handle querying a database with upwards of millions of items, which means compressed representations of the query and database items are likely needed. On the other hand, due to the potential for strong signal distortions in the query, such as additive noise and the corruption of information across the frequency range, these representations must be robust. They must also be relatively specific, so as not to cause confusion between similar items in the database.

While this task had been unsolved for some time, in 2003, Wang introduced the now popular algorithm based upon audio fingerprints with spectral constellations [1]. This algorithm was behind the success of Shazam, a free smartphone app for identifying the song based upon a short recording. This method, while relatively simple, proved to be very successful, even in the case of significant signal distortions, a critical feature for deployment in situations with smartphone recordings in public venues. The key behind this method lies in the use of spectral peak picking to build audio fingerprints. These spectral peaks within the magnitude frequency response are largely robust to signal distortions. This is true for additive noise, since while the noise may alter the spectrum and the overall magnitude of different frequency components, the location of peaks within the spectrum are likely to remain the same. This is true also for the removal of frequency content, either due to highpass, lowpass, or bandpass filtering, since spectral peaks are located not only across time but also across the frequency range. Finally, this method can function quite well even with short snippets, since a a few seconds of audio potentially contains multiple identifying peaks along the frequency axis.

In this project we will create a simple implementation of the original algorithm as described in the original paper [1]. In the following section we will outline our implementation and the process of selecting a set of optimal parameters for the algorithm based upon an investigation of the parameter search space. We will then present a basic evaluation of the system and point out its strengths and weaknesses. We will conclude by identifying potential methods for improving our implementation.

2. FINGERPRINTING

We begin the description of our implementation with the creation of audio fingerprints. The motivation for the creation of audio fingerprints is the generation of unique, yet reproducible representations to identify audio entities. Due to the high dimensionality of audio recordings, the underlying goal of creating a fingerprint is to design some process by which we can generate a low dimensional, yet unique and descriptive identity for every entity in our database. In practice, this is achieved by locating peaks in the magnitude spectrum after performing the Short-time Fourier transform (STFT). This method is both simple and robust to signal distortions, making it ideal for our task.

While apparently simple, the selection of parameters for this process has a significant impact both on accuracy of retrieval, as well as the computational and memory costs. For that reason, careful selection of the algorithm parameters is required to construct a functioning system that can be realized. In the following sections we will describe the basic functionality of the algorithm, and then outline our process for parameter selection.

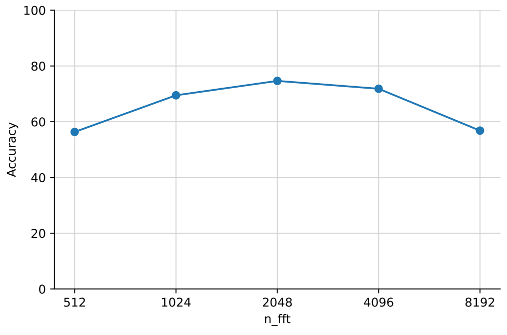

Figure 1. Accuracy based on FFT size.

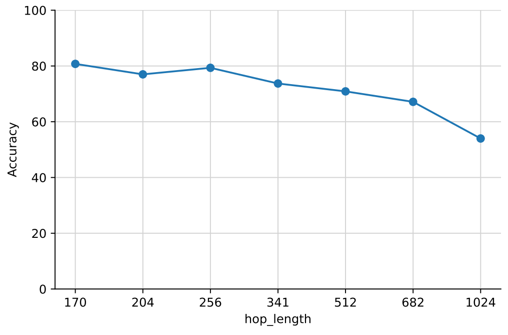

Figure 2. Accuracy based on hop length.

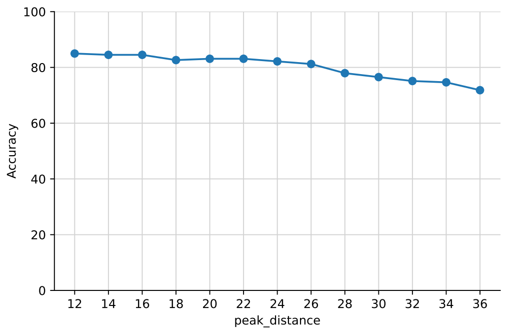

Figure 3. Accuracy based on min. peak distance.

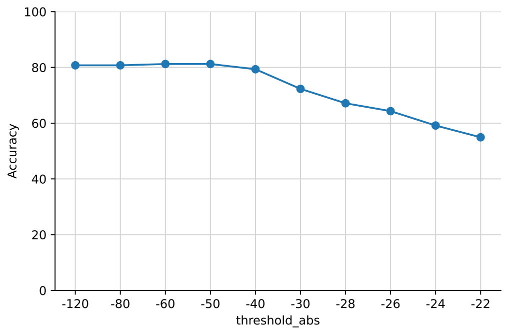

Figure 4. Accuracy based on peak threshold.

2.1 Short-time Fourier transform

We first load the audio signal using soundfile [1] . All operations are carried out on audio signals at 22.05 kHz, since this is the sample rate of the database recordings. If a query is provided at a different sample rate, we first resample the signal before computing the STFT. After peak normalizing the audio signal, we use librosa [2] to compute the STFT using a frame size of 2048 and a hop length of 256. This corresponds to a frame size of 92 ms and a hop length of 11 ms at a sample rate of 22.05 kHz, providing sufficient frequency and time resolution. After computing the STFT, we covert the complex signal to dB.

To determine these STFT parameters we performed a simple sweep across both the STFT frame size and hop length. We show the performance of the algorithm as a function of the FFT size in Figure 1. We held all other parameters constant, as we adjusted the frame size, using a hop length 1/4 the frame size each time. There is a clear peak at a frame size of 2048, which we select as our frame size. Next, we perform a sweep over the hop length, using a constant frame size of 2048, as shown in Figure 2. We found that while a smaller hop length, which increased temporal resolution and improved performance, this came at the cost of more hashes and hence larger memory and compute requirements. To balance this, we selected a hop length of 256, which archived good accuracy with reasonable compute. We note that while this is far from an exhaustive search, as other parameters may have an impact on these results, we see this as a valid starting point.

[1] `https://github.com/bastibe/python-soundfile`

2.2 Spectral peaks

With the magnitude STFT computed, the next critical step involves locating peaks within the spectrogram. While this may appear a simple task, there is a significant number of considerations when designing an appropriate peak picking algorithm. For simplicity, we utilize `peak_local_max()` from the scikit-image [3] library. We consider two parameters, `min_distance`, which sets the minimum distance between any two detected peaks, and `threshold_abs`, which sets the minimum acceptable value for a peak to be considered. While we found that setting a lower minimum distance between peaks increased accuracy, it came at the cost of a significantly larger number of peaks, and hence larger memory and computation cost during the query process.

To formalize these findings, we again performed a sweep across both the threshold and distance parameters. As before, for these tests we held all other parameters constant, using our current best parameter settings. As shown in Figure 3, we found that smaller minimum peak distance resulted in higher accuracy, although the effect was quite small in comparison to the resultant size of the hashes, which grew quite rapidly when using smaller distances. For this reason, we chose a value to balance the accuracy and computational overhead, with `min_distance` of 24.

We also considered the effect of the threshold parameter, and performed another sweep with the results shown in Figure 4. Setting a higher threshold reduced the total number of peaks, which appeared to have a clear impact on accuracy. We found that in practice when using

Name	Value
FFT size	2048
Hop lenth	256
Window	Hann
Min. peak distance	24
Peak threshold	-60 dB
Target zone frequency	256 bins
Target zone time	128 frames
Max peaks	24

Table 1. Final algorithm parameters.

the same algorithm parameters, the number of peaks (and hence hashes) was significantly larger in the database fingerprints than the query fingerprints. This is likely due to the larger noise floor in the query recording, as well as other signal distortions. Therefore, while setting a higher threshold for the database (clean) recordings may produce fingerprints with adequate specificity, the same may not be true for query fingerprints. For this reason, we selected a threshold of -40 dB, which appears to capture enough peaks for adequate performance, while ignoring some superfluous, low-level peaks.

We also investigated the performance of the algorithm as a function of a number of other parameters, but will not discuss the details of this search here. We refer readers to the code provided in the `search.py` script for further details. At the end of this exercise in finding optimal parameters, we created a list of proposed algorithm parameters as shown in Table 1.

2.3 Combinatorial hashing

With the spectral peaks located, we perform a combinatorial hashing process similar to that proposed in [1]. This involves hashing each peak by considering a set of neighboring peaks, and the relative time difference between them. In practice, this is achieved by iterating over each peak and defining a relative target zone in which to select nearby peaks for comparison to the anchor peak. Selecting an appropriate target zone size (considering differences in the time and frequency scales), as well as setting a maximum number of peaks is generally required. This enables control of the total number of hashes generated, which can quickly grow, causing memory concerns.

Hashes for each located peak are then generated as a series of three values, the frequency bin of the anchor f_1, the frequency bin of the neighbor peak f_2, and the relative number of timesteps between the two peaks $t_2 - t_1$. This produces a hash of the form: $(f_1, f_2, t_2 - t_1)$. This hash is then stored along with the song ID and the timestep of the anchor point. The motivation for such hashes is that performing a pointwise correlation between the spectral peaks, or constellation maps will result in significant overhead. Creating hashes enables a further level of compression on the data, while retaining a level of specificity, which can then be efficiently searched.

2.4 Filtering

We also experimented with applying a lowpass filter as well as a mel filterbank to provide better spectral representations before peak picking. The use of a lowpass filter was motivated by the ability to further remove high frequency information which may be more likely to be corrupted by noise. The use of a mel filterbank was motivated by the perceptual relevance of the mel scale. Unfortunately, we did not observe improved performance for either. While the lowpass filter removed some higher frequency components that may contain noise, we found that this had minimal effect on performance. In the case of the mel filterbank, we found that this transformation had a significant negative impact on performance. Even when using a large number of mel bins, such as 256, the accuracy dropped by 30% in comparison the the baseline algorithm. We posit this is likely due to the reduced specificity of the hashes when using fewer frequency bins after the mel filterbank.

3. QUERYING

Once the database fingerprints have been created, we can identify recordings based upon short query snippets. To do so, we first create fingerprints of the query audio signal using the same algorithm parameters as before. This will result in a series of hashes for each query. Our goal is then to search the database using these query hashes to find entities in the database that provide maximal alignment.

3.1 Inverted lists

To make the search process more efficient, we employ the traditional inverted list [4], which enables us to index the database by the hashes, and retrieve all timesteps in which a hash is present in the database entries. This enables us to perform an efficient computation of something similar to the correlation between the query hashes and the database hashes. We create these lists by looping over the hashes of each song and then creating a dictionary that contains the relevant hash as the key, which then points to a list, containing all timesteps in the database song at which this hash was present.

3.2 Search

We then perform a search across the database given the hashes from a query recording. To do so, we loop over the entities in the database, selecting the inverted lists containing the hashes and their locations. For each set of entity hashes in the database, we then loop over the query hashes, looking for a match between them. If we find a match in the inverted list, we then loop over the elements in the inverted list, accumulating the number of matches found at each relative time step. We compute this shifted timestep value for each entry in the inverted list by subtracting the time step of the query hash from the timestep of each match from the database hash in the inverted list. We then assign a score for each song in the database by finding the maximum of the accumulated timesteps at which matches occurred.

Metric	Ours	Random	$\mathbb{E}$
Top-1	81.2%	0.47%	0.05%
Top-2	85.9%	0.47%	0.1%
Top-3	86.4%	1.41%	1.5%
Top-5	86.8%	2.35%	2.5%
Top-10	87.8%	4.69%	5.0%

Table 2. Algorithm accuracy across query set.

4. EVALUATION

To evaluate our algorithm implementation we consider a set of query recordings. These are short recordings made on consumer devices that contain excerpts from recordings in the database. As described above, our process involves first generating a database of hashes for each of the 200 songs in the database, which originated from the pop and classical subsets of the GTZAN dataset [5]. These fingerprint hashes are then converted to inverted lists. We then generate fingerprints for the 213 query recordings in the query set, and search for matches with optimal alignment in the database entries.

The main metric of interest is the Top-N accuracy, which is defined as the percent of the queries in the query set for which the correct entity is retrieved within the first N results. For example, the Top-1 accuracy requires that the first result from the query is the correct entity. We report the Top-N accuracy for a number of different N, as can be seen in Table 2. As a baseline, we also generate matches by constructing random guesses for each query based on items in the database, as shown in the second column. We also show the expected error with random guessing assuming that all queries are unique.

4.1 Speed

We also considered the speed of fingerprint generation. To accelerate the computation of hashes, we implemented a simple parallel processing pipeline using multiple threads with the `multiprocessing` package in Python. In this way, we could then compute multiple fingerprints in parallel, since the computation of one fingerprint is not dependant on any others. We followed the same implementation for the generation of the query fingerprints, but conducted the search for matches across the database for each query sequentially, for simplicity.

Using a modest eight threads, the generation of the database hashes for 200 items required 42.9 seconds, resulting in a rate of around 4.7 fingerprints per second, which we consider to be relatively fast on consumer hardware. Fingerprinting of the 213 items in the query set is somewhat faster, at 9.9 fingerprints per second, which we believe is due to the fewer number of peaks identified in the queries. This is likely due to the addition of noise and other distortions, which raise the noise floor, and decrease the sensitivity of the algorithm to these peaks. We find that we can complete a single query, searching across the entire database, fairly quickly, computing 8.7 queries per second.

5. DISCUSSION

Our implementation is very simple, and therefore has significant room for improvement. Interesting directions for further work include first performing a more extensive optimization of the algorithm parameters. This could be achieved by a naive grid search over all combinations of a set number of the parameters, using both the accuracy and the fingerprint generation speed as metrics for performance. While our experiments make clear that certain configurations improve the accuracy by a number of percentage points, these configurations often impose significant memory and compute, therefore optimizing for accuracy alone will not produce useful results.

Other ideas include additional pre-processing to remove noise from query recordings, or even dynamically adjusting some of the algorithm parameters after performing an analysis on the query recording to determine aspects related to the level of noise and/or reverberation. Finally, superior spectral representations, such as multi-resolution spectrograms, may also provide interesting results.

6. CONCLUSION

In this project we created an implementation of an automatic music identification system based upon audio fingerprints and constellation maps. We outlined our design process, along with our motivation behind our parameter selection for the algorithm. We demonstrated the performance of our system, with Top-1 accuracy of 81%, given a very simple and efficient implementation running on consumer hardware. Due to the simplicity of our implementation, we highlighted a number of potential directions for future work to refine the algorithm, and potentially bring about further improvements in performance.

7. REFERENCES

[1] A. Wang *et al.*, “An industrial strength audio search algorithm.” in *ISMIR*, vol. 2003, 2003, pp. 7–13.

[2] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, “librosa: Audio and music signal analysis in python,” in *Python in Science Conference*, vol. 8, 2015, pp. 18–25.

[3] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, “scikit-image: image processing in Python,” *PeerJ*, vol. 2, 6 2014.

[4] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems (2nd Ed.).* USA: Benjamin-Cummings Publishing Co., Inc., 1994.

[5] G. Tzanetakis and P. Cook, “Musical genre classification of audio signals,” *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.