A Short Introduction to Audio Fingerprinting with a Focus on Shazam

MUS-17

Simon Froitzheim

July 5, 2017

Introduction

Audio fingerprinting is the process of encoding a (potentially) unlabeled piece of audio in any format (called the input file) into a so-called *audio fingerprint*. It is usually required for this process to work in a compact, discriminative, robust and efficient way, such that the resulting fingerprint can be easily stored, indexed and compared with other fingerprints. The most important applications for audio fingerprinting are *content-based audio identification (CBID)*, *content-based integrity verification (CBV)* and *watermarking support* [1]:

On the one hand, content-based audio identification, which also forms the focus of this report, describes the action of identifying audio tracks with missing metadata, solely based on the content of the files themselves: This is usually achieved by computing a fingerprint for every known audio piece beforehand, resulting in a fingerprint database with the related metadata; then, when confronted with an unlabeled input file, the corresponding fingerprint is computed and subsequently matched against the fingerprint database, linking it to the formerly missing metadata in case of a match with the database.

Content-based integrity verification on the other hand aims for an evaluation of the integrity of an audio piece, i.e. CBV could check among other integrity abnormalities for potential storage errors, transmission errors and quality fluctuations (possibly due to noise) of the given audio file with respect to a known template file that is considered to be an optimum quality version of the audio file that is to analyze.

Digital watermarking uses watermarking signals to identify ownership of copyright. This is realized by embedding additional data that is difficult to remove into the input file. Audio fingerprinting can then be used to find a fitting watermarking signal and embedding position [2].

The aim of this paper is to give an orientation about the challenges of audio fingerprinting and state-of-the-art techniques and to additionally briefly present a famous fingerprinting algorithm that is used by the popular app *Shazam*[1].

It is structured as follows: I will first go in detail about the common requirements and challenges for audio fingerprinting, continuing with a short overview over popular techniques. I will then proceed with details about Shazam, before concluding the report.

Requirements and Challenges

There are several (application-dependent) requirements for audio fingerprinting procedures with focus on CBID [1]:

- Most importantly, the fingerprints should be *discriminative* in order to avoid false

[1] `https://www.shazam.com` (Lastly called up by the 3rd July, 2017)

matches.

- Almost as important is the *robustness* to both *cropping* and *distortions*: Cropping results in an audio piece that is much shorter than the complete track, possibly only seconds long; this means that an audio fingerprinting algorithm should get exact results from very short excerpts. Audio distortions are versatile and range from e.g. compression and background noise to pitching and speed changes; even in the presence of such distortions, the track has to be correctly identified.

- The resulting fingerprints need to be as *compactly* stored as possible; there are great numbers of fingerprints to manage.

- When computing a fingerprint, *efficiency* is also of high relevance: The best fingerprint has not much use when computation time exceeds certain limits.

As common with computer science tasks like CBID, there is a trade-off between reliability and efficiency present when trying to meet the presented requirements.

State-of-the-Art

The first audio fingerprinting approaches that remain of relevance to this day emerged in the early two thousands and almost all of them use the same general framework [1]. Audio fingerprinting was and still is to this day of high commercial interest for music companies in particular and there are many commercial applications and services (mainly mobile) that use such techniques, including, among others, *Gracenote*[2] and Shazam. The provided services usually enable users to identify unknown music tracks that they hear in their everyday life and thus potentially increase the number of music purchases.

[2]`http://www.gracenote.com/` (Lastly called up by the 3rd July, 2017)

The most relevant approaches that are still used for commercial applications up to this date are the *Philips Technique* [3], the Shazam algorithm [4] and *Google Waveprint* [5]. There have also appeared more modern approaches in recent years, most notably *MASK* [6, 7] which promise a substantial performance improvement, but aren't as field-tested as the previously mentioned state-of-the-art procedures.

Shazam

While the Shazam algorithm is much older than some of the aforementioned procedures, it has lead Shazam to great commercial success and represents a suitable entry point to the field of audio fingerprinting, as it has many similarities to most of the other algorithms and thus serves as an excellent choice for an exemplary fingerprinting algorithm.

Application Details

Shazam is a free mobile app that recognizes music, TV program and other media based on short audio snippets recorded with the user's phone. It also features special camera interaction for interactive experiences and additional content and can be connected to most popular internet services like Google, Facebook, Spotify, and others. The heart of the application constitutes of the Shazam algorithm presented in a publication by Wang [4] on which this section is also based on.

The Shazam Algorithm - Overview

Assuming that a database of fingerprints was already established beforehand, the Shazam algorithm for new individual pieces of unlabeled audio can be divided into five main phases:

1. Spectogram computation of the input file

2. Construction of a constellation map based on the spectogram

3. Combinatorial hashing on determined anchor points and target zones
4. Searching of the database
5. Scoring of possible matches

It is to be noted that the database fingerprints are created in the same way without the above steps four and five. I will go into more detail about the single phases in the following sections. All featured figures are taken from [4].

Spectogram Computation

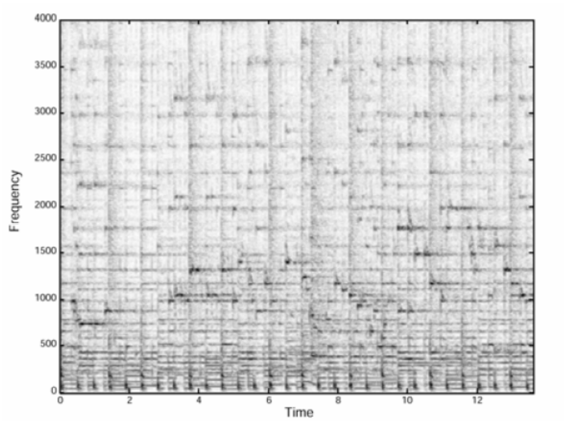

Figure 1: An exemplary spectogram

Spectograms visually represent the signal strength (energy) of a signal over time at various frequencies; darker areas in the spectogram represent a higher signal strength than brighter regions. Spectograms can also be used to depict sound waves and Shazam is exactly doing that in the first phase of the algorithm: The first action taken by the procedure is to compute the spectogram of the input file via *Fourier transform*. Figure 1 shows one such spectogram.

Constellation Map Construction

In the next step, the procedure chooses high energy candidate peaks of the spectogram with respect to amplitude and density, i.e. uniform coverage of the peaks across the whole audio file is maintained. The reason why Shazam searches

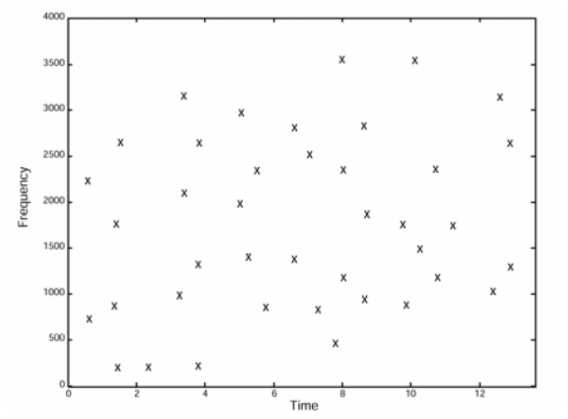

Figure 2: A constellation map, resulting from processing the spectogram shown in Figure 1.

for energy peaks is the fact that the higher signal strengths are much more robust to noise than the lower ones. Additionally, the amplitude component of the spectogram is eliminated in the process, simplifying the representation. The result of this process is called the *constellation map* (cf. Figure 2).

Combinatorial Hashing

With the resulting constellation maps from the previous phase, it is already possible to start matching audio files by sliding the newly computed constellation map across constellation maps from database files. However, the general constellation map is very sparse which would result in very slow matching times. Because of this, Shazam applies combinatorial hashing to achieve a matching speed-up: Each constellation point is taken as an anchor point and for each such anchor point a target zone together with a limited amount of target points from that zone are computed (cf. Figure 3). Each anchor point is then sequentially paired with each of its target points, every pairing forming exactly one hash, consisting of the two frequency components of anchor and target point and the time difference between them. Each hash is then represented by a 32-bit unsigned integer, plus an additional time offset from the beginning of the input file to the anchor point. It is to be noted that the number of target points in each target zone is limited to avoid an extreme number of produced hashes which would strongly hinder efficiency.

Database hashes actually contain more information than the hashes that are produced for unlabeled input audio: The database hashes that are later on used for matching are actually 64-bit structs, as they are additionally containing a track ID. By using hash structs like these as fingerprints, Shazam achieves the aforementioned matching speed-up: Each hash is a much more specific piece of information than just a single constellation point and although there are much more possible hashes than constellation points, the matching process described in the next subsection experiences a tremendous speed-up. The trade-off that the procedure brings with it is an increase in necessary storage space for the great number of hashes in contrast to the number of constellation points.

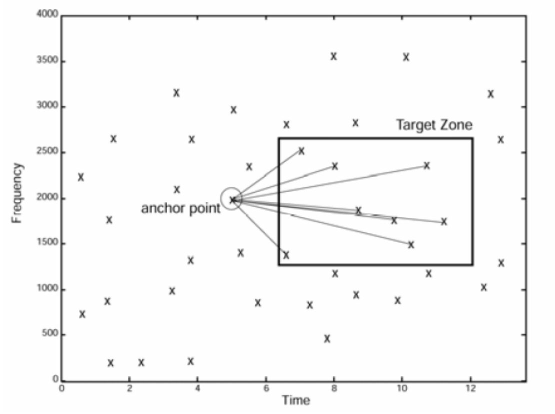

Figure 3: Paradigmatic anchor point and the corresponding computed target zone with target points, extracted from the constellation map seen in Figure 2.

Database Searching

For searching purposes, the hashes generated from the input file are matched against the database hashes; every matching hash results in a time pair consisting of both offset times. This is necessary, because the time offsets of the hashes from the input file most likely differ from

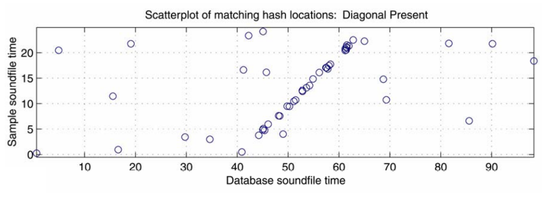

Figure 4: An exemplary scatterplot of matching hash locations in case of a successful match.

the time offsets of the database hashes, as the former are based on a short audio snippet, while the latter ones are computed from the complete tracks. These time pairs are then distributed into bins according to the track ID. Right now, the algorithm has only a number of time pairs according to matching hashes, but these matching hashes don't guarantee that the whole audio files match. Thus, the matching hashes need to be scored.

Scoring of Possible Matches

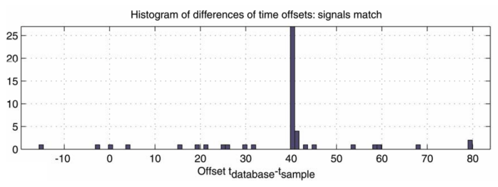

Figure 5: An exemplary histogram of matching hash locations in case of a successful match. The histogram corresponds to the scatterplot seen in 4 and the y-axis shows the input sound file time.

Finally, the resulting bins from the previous phase are scanned for matches on audio file level: Within each bin, the set of time pairs can be represented by a scatterplot of association between the input file and the database sound files. In case that the files match as a whole, the matching hashes should occur at similar relative offsets with respect to the beginning of the file. That means that a sequence of hashes in the input file should also occur somewhere in the matching database file. The detection thus boils down to detecting diagonal lines in these scatterplots; a detected diagonal line represents a successful match (cf. Figure 4). In practice, Shazam is not constructing the described scatterplots for computation, in fact, the procedure computes an histogram of differences of time offsets for every bin and searches for peaks (see Figure 5). This is done for reasons of efficiency.

Conclusion

I have presented a short introduction into the topic of audio fingerprinting, including the explanation of main applications, the mentioning of state-of-the-art algorithms and audio fingerprinting challenges and a closer look at one paradigmatic popular procedure, the Shazam algorithm.

References

[1] E. Batlle, P. Cano, J. Haitsma and T. Kalker (2002). *A Review of Algorithms for Audio Fingerprinting*. IEEE Workshop on Multimedia Signal Processing.

[2] S. Zmudzinski, M. Steinebach and M. Butt (2012). *Watermark Embedding Using Audio Fingerprinting*. Transactions on Data Hiding and Multimedia Security VIII, pp. 63–79.

[3] J. Haitsma and T. Kalker (2002). *A Highly Robust Audio Fingerprinting System*. Journal of New Music Research 32 (2), pp. 211 – 221.

[4] A. Wang (2003). *An Industrial-Strength Audio Search Algorithm*. Proceedings of the 4th International Conference on Music Information Retrieval.

[5] S.Baluja and M. Covell (2007). *Audio Fingerprinting: Combining Computer Vision & Data Stream Processing*. Acoustics, Speech and Signal Processing.

[6] X. Anguera, A. Garzon and T. Adamek (2012). *MASK: Robust Local Features for Audio Fingerprinting*. IEEE International Conference on Multimedia and Expo.

[7] A. Garzon (2011). *Audio Fingerprinting (Master's thesis)*. Retrieved from `http://mtg.upf.edu/node/2326` (Lastly called up by the 3rd July, 2017).