An Analysis of Audio Fingerprint Complexity for Music Identification

Christopher Morse (drb6yv)

May 2, 2023

1 Introduction

Music identification software has grown in popularity since the turn of the century, most notably with the release of *Shazam* [1] and *SoundHound* [2]. This technology is still incredibly relevant today, as Shazam was acquired by Apple in 2018 [3]. The general concept behind music identification is fairly straightforward: a user queries a huge database of encoded songs with a sample, and an algorithm finds and reports the closest match from the database. These popular software companies, however, provide fairly limited information regarding which algorithms are used. The main insight into early music identification algorithms comes from a 2003 paper published by one of Shazam's founders, Avery Li-Chun Wang [4]. This paper outlines high-level details about their algorithm at the time.

The primary insight from this paper is that the original version of Shazam used a technique called *audio fingerprinting*. The idea behind *audio fingerprinting* is that directly matching raw audio signals for music identification is intractable, since background noises are inevitable. Instead, raw audio signals are converted into a connected map of frequencies through time, and comparisons are made between these maps. This is much more resilient to background noise, since many of the component frequencies will not be as affected by noise [5].

2 Problem and Motivation

One problem is that there are many ways to to fingerprint audio signals, and the methods that give the best accuracy or are the most robust to noise remain unclear. Another problem is that the fine details regarding the algorithms behind Shazam or SoundHound are unknown, as their code is unavailable and their described techniques are either private or incomplete. Therefore, this report has two primary goals. First, it is necessary to recreate Shazam's pipeline based on how it was described in the 2003 paper. Second, this pipeline will be used to evaluate various fingerprinting parameterizations over noisy data to determine how each parameter affects accuracy, along with the tradeoff between complexity and noise resiliency for each parameter.

3 Approach

My implementation of Shazam's music identification pipeline consists of four main components: *Frequency Analysis*, *Keypoint Extraction*, *Fingerprinting*, and *Fingerprint Matching*.

3.1 Frequency Analysis

The first step is to perform frequency analysis over a signal. To this end, I implemented the Short-Time Fourier Transform (STFT) algorithm, which accepts a raw signal as input and produces a spectrogram that contains the signal's component frequencies over time. Each time I ran a signal through the STFT algorithm, I used a Hanning window of length 756 and a hop length of 378.

3.2 Keypoint Extraction

After a spectrogram has been extracted from an input signal, the next step in the pipline is to detect keypoints (see Figure 1a). This is accomplished through several image processing techniques. First, to establish the spacing between keypoints, a given spectrogram image is partitioned into regions (or "bins") of some size $n \times n$. This bin size is a parameter that is provided by the user. Within each bin, the highest intensity pixel is preserved and all other pixels are set to zero. After this process, there is exactly one delegated key point per bin. To determine which keypoints are most valuable, the low-intensity key points are pruned away if they do not reach a given intensity threshold, α. This threshold is another parameter that is provided by the user. When complete, this step reports a set of key points that are spaced and thresholded according to the given parameters.

3.3 Fingerprinting

The next step in the pipeline is to convert the set of keypoints into a unique fingerprint (see Figure 1b). This is accomplished through a variation of the *k-Nearest Neighbors* algorithm. For each keypoint (the "anchor point"), the k neighboring keypoints that are nearest to it (by Euclidean distance) and occur *after* the anchor point in the time domain are clustered together (and hereby referred to as a "local fingerprint"). This variation is used to prevent a sample from having fingerprints that have been cut off at its start, which would lead to a greater quantity of mismatches. The k parameter is provided by the user, and directly defines how many keypoints to include within each local fingerprint. This process is then repeated for all keypoints until the full fingerprint is complete.

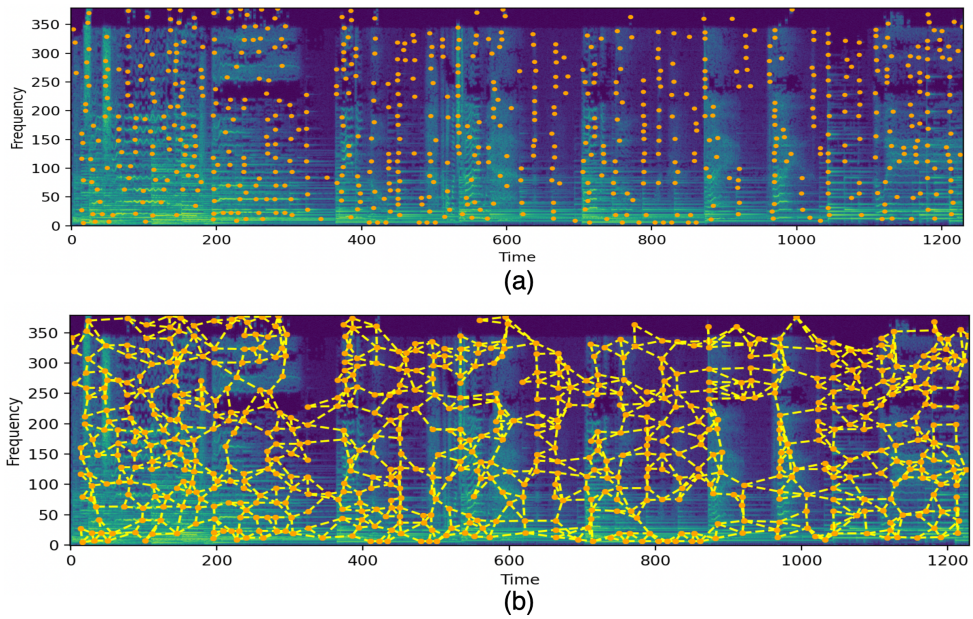

Figure 1: Keypoint Extraction (a) and Fingerprinting (b).

3.4 Fingerprint Matching

Once fingerprints have been extracted from each desired signal, the next necessary step is to match one fingerprint to another. For efficiency, each fingerprint is encoded as a set of *hashed* local fingerprints. Each local fingerprint is stored in a hashtable as follows:

$$h_\omega = hash(f_\omega, \{(f_1, \Delta t_1), (f_2, \Delta t_2), ..., (f_k, \Delta t_k)\}) : t_\omega,$$

where f_ω, t_ω are the frequency and time values of the anchor point, respectively, f_j, t_j are those of some neighbor j, and $\Delta t_j = (t_j - t_\omega)$ is the time offset between the anchor point and a neighbor. The value associated with the hash key is the anchor point's position in the time domain, t_ω.

For the matching procedure, assume that we are given a query fingerprint $q = \{h_0, h_1, ..., h_M\}$ and a database fingerprint $d_k = \{h'_0, h'_1, ..., h'_N\}$. To determine the degree to which q matches with d_k, this algorithm first counts the number of local fingerprint hash keys that are shared between them, i.e.:

$$count(q, d_k) = |q \cap d_k|$$

For each matched local fingerprint, to ensure that they are time-aligned, it is necessary to find the relative offset between that of the database and that of the query, i.e. $(t'_\omega - t_\omega)$. Once complete, the matching score between the fingerprints q and d_k is given by the highest number of matches with the same time-alignment. This matching procedure is repeated for all fingerprints in the database, and the song with the highest number of time-aligned matches is the algorithm's final prediction.

4 Study

In this section, a study is provided that is rooted in the fact that fingerprint complexity is determined by the quantity and size of its local fingerprints. By adjusting the parameters discussed earlier, audio fingerprints can be of various levels of complexity. Therefore, through this study, I seek to answer the following research questions:

- **RQ1:** How does each fingerprinting parameter affect overall accuracy?
- **RQ2:** Are high complexity fingerprints less robust against noise?

Through the exploration of these research questions, we will achieve a greater understanding of how fingerprinting algorithms should be parameterized.

4.1 Experiment Setup

For my experiment database, I used the MagnaTagATune Dataset [6], which contains over 25 thousand songs (30 seconds each) of diverse genres and instrumentation. I then randomly selected and extracted audio from 4k of the songs as 5 second clips. Afterward, I created 8 different test sets, each with varying levels of random Gaussian noise added to each audio file ($\sigma = [0, 0.01, 0.1, 1, 10, 100, 100, 10000]$). Over each parameterization discussed in the following sections, I fingerprinted the entire database and evaluated each setting with all 8 test sets, to report the accuracy across each noise level.

4.2 RQ1

For this experiment, I seek to determine how each fingerprinting parameter affects overall accuracy.

4.2.1 k Parameter

First, I vary the size of each local fingerprint (i.e. number of keypoints) while maintaining all other parameters as constants ($bin\_size = 20, \alpha = 30$). Fingerprint size is determined by the value of k from the *k-Nearest Neighbors* algorithm in the *Fingerprinting* step, where $k = i$ means that each local fingerprint contains $i+1$ keypoints. To this end, four parameterizations are evaluated, covering the values $k = [1, 2, 3, 4]$. Note that a large value for k will result in higher fingerprint complexity, since each local fingerprint will include many keypoints. Figure 2a shows a plot of the accuracy over progressively noisier test sets. It is shown that larger local fingerprints actually leads to substantially lower accuracy, even in the case with zero noise. Therefore, these results suggest that local fingerprints should simply be pairs of keypoints for the highest accuracy. We can also observe that all parameterizations are fairly robust against low levels of noise, as they do not face a significant decline in accuracy until $\sigma = 100$.

4.2.2 bin_size Parameter

Next, I vary the size of each bin, a parameter that is used in the *Keypoint Extraction* step, while maintaining all other parameters as constants ($\alpha = 30, k = 2$). As discussed earlier, the bin_size parameter determines the spacing of keypoints throughout a given spectrogram. I then construct four parameterizations to vary this parameter, covering the values $bin\_size = [10, 20, 40, 60]$. Note that a small bin size will result in higher fingerprint complexity, since keypoints will be very densely extracted. As shown in Figure 2b, accuracy decreases as the bin size is increased, across all noise levels. This suggests that keypoints should be fairly dense across a spectrogram because, when spaced too far apart, the resulting fingerprints are not sufficiently representative of the spectrogram.

4.2.3 α Parameter

Finally, I vary the intensity threshold α, a parameter from the *Keypoint Extraction* step that determines the required intensity of each keypoint. I vary this parameter over the values $\alpha = [120, 90, 30, 10]$, with the other parameters as constants ($bin\_size = 20, k = 2$). Note that a low value for α will result in higher fingerprint complexity, since fewer keypoints will be pruned. As shown in Figure 2c, pruning too many keypoints greatly reduces accuracy. Otherwise, accuracy is fairly similar between parameterizations. This suggests that the most representative keypoints hold the highest intensities.

4.3 RQ2

In the second part of my study, considerations from the previous section are used to combine parameters into low and high complexity fingerprinting parameterizations. To this end, the following parameterizations are used:

- Low Complexity: ($bin\_size = 60, \alpha = 120, k = 1$)
- Mid-Low Complexity: ($bin\_size = 40, \alpha = 90, k = 2$)
- Mid-High Complexity: ($bin\_size = 20, \alpha = 30, k = 3$)
- High Complexity: ($bin\_size = 10, \alpha = 10, k = 4$)

Figure 2d shows the accuracy results for all parameterizations, across each test set. As shown, the lowest complexity setting was too simple for matching to be effective, and therefore resulted in the poorest

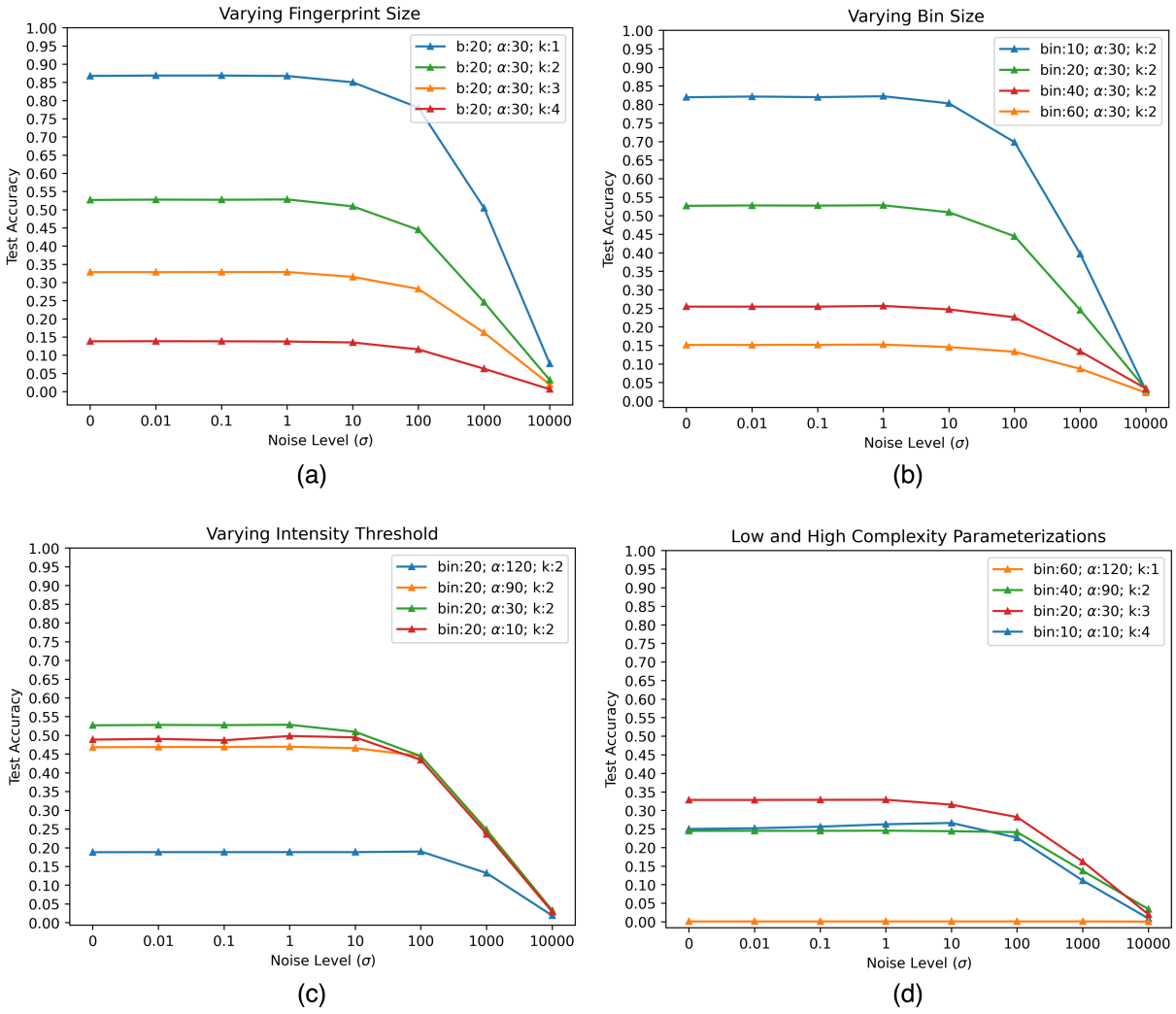

Figure 2: Accuracy results from varying fingerprint size (a), bin size (b), and intensity threshold (c). Subfigure (d) shows results from low to high complexity fingerprinting parameterizations.

accuracy. The results also show that the high complexity settings exhibit slightly more sensitivity to noise levels between $\sigma = 10$ and $\sigma = 100$, in that they begin to drop in accuracy sooner and more quickly than the *mid-low* complexity setting.

4.4 Discussion of Findings

In the first part of this study, it is found that the size of each local fingerprint has the greatest effect on accuracy. Even so, the spacing and intensity requirement of keypoints also have a fundamental role in the resulting accuracy. For the greatest accuracy, keypoints should be fairly dense, and only those with the least intensity should be pruned. When forming local fingerprints, these keypoints should form low-complexity groupings (ideally pairs). In the second part of the study, I explore selections of these parameters to create a range of simple to complex settings. These results demonstrate that it is necessary to find a balance between these parameters, and that the highest complexity fingerprints are slightly less robust against noise, since noise may slightly alter some low-intensity local fingerprints (therefore leading to mismatches).

5 Challenges

In the development of this project, I faced a few key challenges. Aside from a fairly high-level overview of Shazam's algorithm from resources found online, there was minimal information regarding the fine details of the algorithms behind Shazam or SoundHound. For this reason, there were several implementation details for which I had to make my best guess regarding how Shazam might actually handle audio fingerprinting. Furthermore, given that I had a database of 25 thousand songs, 8 test sets with 4 thousand clips each, and 13 unique parameterizations, processing and evaluating all of this data (through fingerprinting, matching, etc.) takes a lot of time. To handle this, I submitted my jobs through Slurm to get my results in time.

In my proposal, I suggested an extension of this work to compare traditional fingerprinting approaches to the use of deep generative models for music identification. Inspired by existing work which has used spectrograms as input to variational autoencoders (VAEs) for the purpose of learning speech representations [7], I set out to train a VAE to encode spectrogram images into their latent representations. To identify a (potentially noisy) audio sample, I considered passing it through the encoder and identifying the closest database sample in the latent space as the final prediction. I began to train several VAE architectures of various sizes, but was not able to produce results in time. For this reason, a thorough comparison of noise resiliency between traditional fingerprinting and VAE-based approaches for music identification will be left for future work.

6 Conclusion

This project presents a replication of Shazam's early music identification pipeline, based on the concept of *audio fingerprinting*. Since there many ways to fingerprint audio signals, the goal of this project was to use the replicated pipeline to explore the effects of each parameter on test accuracy and noise resiliency. Through the study provided, it is shown that a balance between these parameters is necessary to achieve the highest accuracy against all levels of noise. Furthermore, it is shown that extremely complex parameterizations are more susceptible to noise-induced reduction in accuracy than those that offer balanced complexity.

References

[1] *Shazam*. Available at: https://www.shazam.com/ (Accessed: May 2, 2023).

[2] *Soundhound Music App* (2023), SoundHound. Available at: https://www.soundhound.com/soundhound (Accessed: May 2, 2023).

[3] Apple Acquires Shazam, Offering More Ways to Discover and Enjoy Music (2023) Apple Newsroom. Available at: https://www.apple.com/newsroom/2018/09/apple-acquires-shazam-offering-more-ways-to-discover-and-enjoy-music/ (Accessed: May 2, 2023).

[4] A. Wang. An Industrial-Strength Audio Search Algorithm. *4th International Conference on Music Information Retrieval*, January 2003.

[5] W. Drevo. Audio Fingerprinting with Python and Numpy. November 2013. *https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/.*

[6] Edith Law, Kris West, Michael Mandel, Mert Bay and J. Stephen Downie (2009). Evaluation of algorithms using games: the case of music annotation. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*

[7] J. Chorowski, R. J. Weiss, S. Bengio and A. van den Oord, "Unsupervised Speech Representation Learning Using WaveNet Autoencoders," in *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 12, pp. 2041-2053, Dec. 2019, doi: 10.1109/TASLP.2019.2938863.