AUDIO IDENTIFICATION

Shivam
ec24172@qmul.ac.uk
MSc. Sound and music computing, Queen Mary University of London

--

1. Introduction

Audio fingerprinting is a technique used to identify audio clips by extracting unique and compact representations from sound files. It's widely applied in services like Shazam[1] for song recognition. Shazam solved the challenge of quickly identifying music from short samples in large databases while maintaining high accuracy. When users record a 15-second clip via phone, the algorithm matches it against the database and returns the song details via SMS. The service also lets users view their tagged tracks online and purchase music or ringtones.

Several competing music recognition services have emerged in recent years. Musiwave offers a mobile-based identification system on Spain's Amena network using Philips' audio fingerprinting technology.[2-4] Neuros integrates Relatable's algorithm into its MP3 players, enabling users to record 30-second radio clips for later online identification.[5,6] Similarly, Audible Magic's Clango service employs the Muscle Fish algorithm to identify songs from internet radio streams.[7-9]

The goal of this project is to implement a simplified version of such a system capable of identifying songs from short, 213 noisy snippets of 10 seconds each from different sections of the songs using audio fingerprinting methods. We work with a subset of the GTZAN genre classification[10] dataset, focusing on 100 songs of the classical and pop categories each.

The system converts audio signals into spectrograms and identifies high-energy peaks that stand out in the time-frequency domain. By pairing these peaks using time-offset anchors, the system generates robust, noise-tolerant hashes that represent the audio's fingerprint. This method is especially effective for matching pop snippets, where percussive elements and clear transients dominate. However, it shows relatively lower accuracy for classical music, likely due to its softer dynamics and less distinct peak structures.

2. Implementation

2.1 Creating Hashes

2.1.1 Peak Detection

We begin by converting the audio into a spectrogram using the Short-Time Fourier Transform (STFT), using a Hann window to reduce spectral leakage. The analysis window size (2048 samples) defines the frequency resolution, while the hop length (the stride between consecutive windows) governs temporal resolution.

Using a smaller hop size (e.g., 256 samples) yields finer time resolution, allowing more precise peak localization in time. However, this also introduces more spurious peaks due to increased sensitivity to transient noise. A larger hop size (512 samples) reduces temporal resolution slightly but offers more stability and noise robustness.

To extract prominent spectral features, we identify local maxima using `peak_local_max` [11]in the magnitude spectrogram, producing a constellation map which is a sparse set of high-energy points in time-frequency space. These peaks represent the most perceptually significant components of the audio and form the basis for fingerprinting. Two key parameters control peak selection were a) **Minimum distance**: This enforces a minimum separation between detected peaks in the time-frequency domain. It helps avoid clustering of nearby peaks caused by local fluctuations, ensuring that only the strongest peak within a neighborhood is retained. For creating the fingerprints, b)**Relative threshold**: This sets a floor for peak detection based on the spectrogram's dynamic range. Only points that exceed a certain fraction of the global maximum are considered. This suppresses low-energy noise and ensures that only salient features contribute to the fingerprint.

The values of minimum distance was set to 10 and the value of relative threshold was set to 0.0001 in this research. These values were tested to generate the most meaningful hashes by providing a correct balance between the number of spectral peaks detected and their significance in the audio fingerprint. A minimum distance of 10 ensures that peaks are sufficiently separated in the time-frequency plane, avoiding redundant or closely clustered points that could lead to noisy fingerprints.

Fig 2.1 shows the peaks detected in the audio file classical.00003.wav.

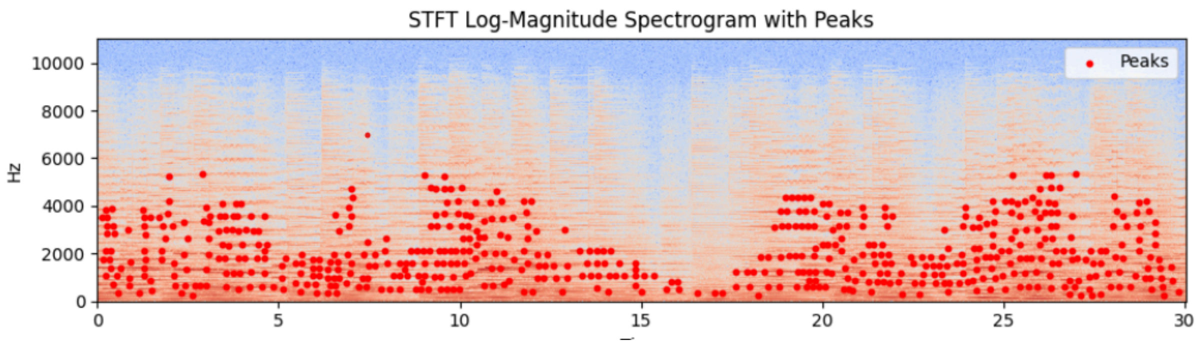

Fig2.1 (STFT Log mel spectogram with peaks)

2.1.2 Hash Creation by using Anchors

To generate robust and compact audio fingerprints, we construct **anchor-target pairs** by examining the constellation map of each track(Fig 2.2). Each prominent spectral peak (the **anchor**) is linked with other nearby peaks (**targets**) that lie within a defined target zone: a region bounded in both time and frequency.

The parameters of this target zone critically affect fingerprint quality: **i) Maximum time difference** sets how far ahead in time we search for targets, influencing temporal context.**ii) Frequency bounds** restrict matches to those within a meaningful pitch range, avoiding irrelevant pairings across disparate spectral bands. **iii) Fan-out** limits the number of target peaks per anchor to control hash density and reduce redundancy. The peaks are sorted by their amplitude and top fan-out number of peaks are chosen. A high pass filter was also implemented to check the effect frequencies above and below the cutoff frequency.

From each anchor-target pair, we derive a hash tuple: [f1, f2, t], where f1 is the anchor's frequency bin, f2 is the target's frequency bin and t is the time gap between anchor and target peaks.(Fig 2.3) This representation captures a local spectral relationship that is both distinctive (likely unique to a song) and invariant to small temporal shifts, which makes it resilient against minor edits or noise in the query signal. Additionally, storing the hash alongside the song ID and anchor timestamp allows efficient reverse lookup during query matching.

By combining multiple such hashes per track, we not only obtain a rich and noise-tolerant fingerprint database, but we also get a high acceleration in the search process as described in [1].

2.2 Searching with Sample Queries

Query audio is processed identically: transforming via STFT, and hashing peak pairs. These query hashes are compared against the database using set intersection.

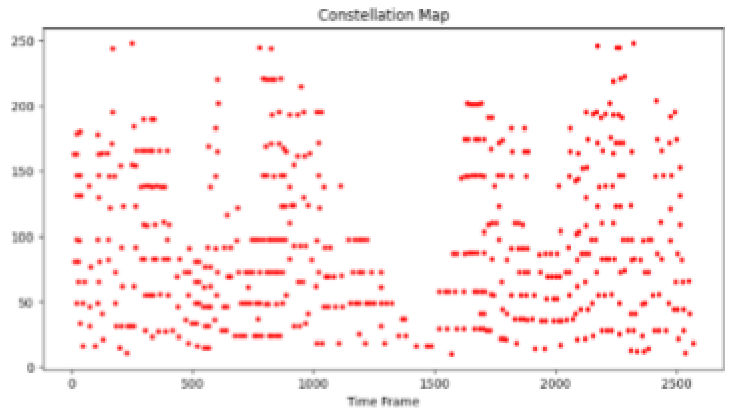

Fig 2.2 Constellation map(Freq. bin vs Time frame)

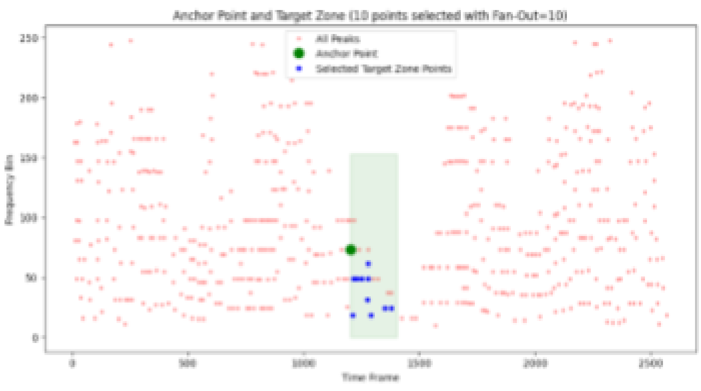

Fig 2.3 Anchor with fan_out=10(Freq. bin vs Time frame)

The songs with the highest number of matching hashes are considered the most likely matches, with the system typically returning the top candidate or top three candidates. As described in [1] the audio fingerprinting system uses combinatorial hashing (pairing points) instead of single points, providing a ~10,000× speedup in searches(for fan out of 10). While this reduces hash survival probability from p to p^2 (since both points in a pair must survive distortion), the system compensates by generating many more hash combinations than original points. This approach effectively balances computational efficiency with matching accuracy.

3.1 Accuracy Metrics

To evaluate the effectiveness of our audio fingerprinting system, we measure **classification accuracy** under two key evaluation setups:

To evaluate the effectiveness of our audio fingerprinting system, we measure **classification accuracy** under two key evaluation setups **Per-Class Accuracy:** We report accuracy separately for the **classical** and **pop** genres. This highlights how well the system performs across distinct audio characteristics. For instance, pop songs often have percussive, repetitive elements, while classical music has a broader dynamic range and less consistent spectral features. These differences impact fingerprint robustness, as seen in lower classical genre accuracy. **2. Overall Accuracy :** This is the proportion of correctly identified tracks across all samples. It provides a holistic view of system performance, averaged over both genres.

We also calculate Top-1 vs Top-3 Accuracy. These reflect the precision of the fingerprint matching algorithm on different levels.

It is important to take a note that in the dataset, there were multiple audio files found to have different names even though they had the same music. The details of a few of these are stored in notes_2.txt in the code repo. These incorrect labels effect the accuracy and can not be ignored.

3.2 Experiments

3.2.1 Parameter tuning

In this section, we present the results of different experimental configurations to evaluate the effect of various parameters on system performance. For each experiment, the key parameters were adjusted, and their impact on accuracy and robustness was observed.

Threshold and min_distance were kept 0.0001 and 10. These values were chosen because they give a sufficient amount of hash with minimal latency. Performance was observed to be degrading with changing values.

Table 3.1 shows the results pf tuning the tuning cutoff frequency of LPF and hop size of peak detection from 30+ runs obtained by tuning other parameters. These include fan out, sampling rate etc. but their tuning did not increase the performance further.

Hop Size	Cutoff (Hz)	Min Dist	Top-1 Overall	Classical Top-1	Pop Top-1	Top-3 Overall	Classical Top-3	Pop Top-3
256	500	7	65.42%	58.88%	72.64%	71.96%	61.68%	83.02%
512	500	10	64.95%	51.40%	79.25%	71.50%	54.21%	89.62%
256	500	10	66.36%	47.66%	85.85%	73.36%	54.21%	93.40%
256	250	10	64.49%	45.79%	83.96%	71.96%	55.14%	89.62%
256	1000	10	64.02%	44.86%	83.96%	71.50%	51.40%	92.45%
256	500	10	65.89%	46.73%	85.85%	72.43%	52.34%	93.40%

Table 3.1 : Results obtained by tuning cutoff and hop size.

Several trends can be observed from the results:

1. **Effect of Cutoff Frequency:** A cutoff of 500 Hz consistently provides better results than 250 Hz or 1000 Hz. At 500 Hz, the balance between removing low-frequency noise and retaining important musical features seems optimal. Raising the cutoff to 1000 Hz sometimes improves pop accuracy slightly but hurts classical performance, possibly due to the loss of low-mid harmonic content crucial for orchestral textures.

2. **Effect of Hop Size:** Lowering the hop size from 512 to 256 generally improves pop accuracy, likely because it increases temporal resolution, capturing more detail in fast-changing music like pop. Classical music, however, does not benefit as much and even shows slight degradation. This might be due to increased sensitivity to noise or overlapping harmonics in more complex classical textures.

Best Configuration : hop_size=256, cutoff=500, min_distance=10

3.2.2 Pop vs Classical Discrepancy

Across all configurations, **pop music consistently outperforms classical music** by a large margin (Approx. 30%+ difference). This disparity may stem from pop tracks having clearer onsets and sparser textures. Also, Classical recordings often being noisier in the dataset or contain overlapping harmonics that confuse peak detection.(eg : classical.00065-snippet-10-0.wav)

3.3 Noise content in the audio files

A major challenge encountered during experimentation was the consistently poor recognition accuracy across several audio files. Upon inspection, it was clear that high noise content significantly affected peak detection and fingerprint generation, particularly in files with ambient noise, low fidelity, or overlapping instruments.

To address this, we initially applied a **high-pass filter (HPF)** as a basic noise reduction strategy as discussed in section 3.2, aiming to suppress low-frequency noise that often contaminates music recordings. However, this approach proved to be ineffective. While HPF helped attenuate some noise, it failed to meaningfully improve peak clarity or recognition results, especially in mid to high frequency noise scenarios.

As a next step, we experimented with a more advanced denoising technique using Resemble Enhance[12], a deep learning-based speech enhancement tool. This was applied to a few selected audio files to evaluate its potential. Interestingly, the model performed reasonably well on some examples — for instance, the pop.00019 snippet saw noticeable improvement in clarity and the model gave fingerprint matching after denoising. However, the results were inconsistent. Since the model is optimized for **speech enhancement(The vocals from pop.00019 were the reason the denoiser worked well on it)**, it struggled with complex polyphonic music and genre-specific textures. In most cases, it either removed key musical content or failed to eliminate non-speech noise effectively. Additionally, the processing time per file was approximately **10 seconds**, which introduces **significant latency** and renders it impractical for real-time or large-scale audio fingerprinting use. Another denoiser library `noisereduce[13]` was experimented with but the accuracy reduced drastically by order of 10% over the whole dataset. Again, in this case as well musical elements important for audio matching were removed by the denoiser. However, hyperparameter tuning of this denoiser was not tested in this research. This tuning can be done in future to test the full potential of the denoiser.

These experiments highlight the limitations of existing denoisers in the music domain. There is a clear need for a **low-latency, music-specific denoising solution** that can preserve harmonic content while suppressing ambient or instrumental noise.

4. Refereces

[1] Shazam - Music Discovery, Charts & Song Lyrics
[2] http://www.musiwave.com
[3] Jaap Haitsma, Antonius Kalker, Constant Baggen, and Job Oostveen., WIPO publication WO 02/065782A1, 22 August 2002, (Priority 12 February, 2001).
[4] Jaap Haitsma, Antonius Kalker, “A Highly Robust Audio Fingerprinting System”, International Symposium on Music Information Retrieval (ISMIR) 2002, pp. 107-115.
[5] Neuros Audio web site: http://www.neurosaudio.com/
[6] Sean Ward and Isaac Richards, WIPO publication WO 02/073520A1, 19 September 2002, (Priority 13 March 2001)
[7] Audible Magic web site: http://www.audiblemagic.com/
[8] Erling Wold, Thom Blum, Douglas Keislar, James Wheaton, “Content-Based Classification, Search, and Retrieval of Audio”, in IEEE Multimedia, Vol. 3, No.3: FALL 1996, pp. 27-36
[9] Clango web site: http://www.clango.com/
[10] GTZAN dataset:
https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
[11] https://scikit-image.org/docs/0.25.x/auto_examples/segmentation/plot_peak_local_max.html
[12] https://huggingface.co/spaces/ResembleAI/resemble-enhance
[13] https://pypi.org/project/noisereduce/