Music Informatics Coursework 2 Report

Yiannis Emmanouilidis

I. Introduction and Aim

The purpose of this work is to develop and evaluate a system which carries out the task of audio identification. In this problem, we have a database of audio files and a query, which should be a noisy audio excerpt of one of the database files, and the goal is to identify which is the correct corresponding database file for a query. In this work, the database consists of 200 30 second-long classical and pop audio clips that are taken from the GTZAN dataset, with the queries being 213 10 second-long excerpts from these database files with added noise such as conversation and background noise from different kinds of spaces. The approach examined here takes the original dataset and, through a series of steps (Short-Time Fourier Transform calculation, peak picking based on energy values, and hashing to further quantise the frequency dimension), obtains and saves a fingerprint for each audio file in the database, which is a compact representation which should ideally retain essential informative features of the original audio. Then, after similar processing is also done on the queries, the use of shifted inverted lists enables the efficient identification of the closest match between each fingerprint and shifted versions of the processed query, followed by the identification of the most appropriate database file for the query via a scoring process for each file. In the following discussion, the proposed method for tackling this problem is first put in context of other approaches and compared against them, then, it is described in detail, and this is followed by an explanation of its subsequent evaluation, and the strengths and weaknesses that come into view as a result.

II. Related Work

One of the foundational systems for audio identification is the Philips fingerprinting system, which extracts 32-bit fingerprints based on energy differences in frequency and time, achieving a high degree of robustness against common audio degradations [3]. This model's limited ability to handle substantial pitch shifts was improved upon by Chu et al. through the introduction of a Peak-point based Philips fingerprint (PPF), which dynamically computes energy bands using peak points in the spectrogram, achieving robustness against a wider range of pitch shifts [4].

Another prominent approach is that of the Shazam system, which focuses on extracting numerous compact key signatures representing peak pairs in the time-frequency spectrogram, with identification being achieved by accumulating matches of these keys over time [7]. The pairing method is a useful update on the single-point comparison method to be discussed later, as considering pairs of peaks enables increased specificity of hashes and thus a higher computation speed. Another attempt at handling not just pitch modifications, but also the additional challenge of time-scale modifications, was made in the Panako system by Six and Leman [5]. Contrary to the implementation in this report, Panako makes use of Constant-Q spectrograms and forms fingerprints by combining triplets of local maxima from the spectrograms. In the construction of these triplets, time ratios are used to ensure time-scale invariance and frequency differences for pitch-shift invariance.

More recently, neural network-based approaches have also gained traction in audio fingerprinting. These methods leverage self-supervised contrastive learning to learn embedding spaces that are resilient to signal distortions. Chang et al. proposed a neural audio fingerprinter trained with a contrastive learning framework to generate low-dimensional embeddings from short audio segments, involving data augmentation techniques to simulate real-world distortions and achieving promising results [6]. Bhattacharjee et al. took a Graph Neural Network approach, constructing a k-nearest neighbour graph and applying graph convolution to capture both local and global information [1]. This achieved superior performance on large-scale datasets and demonstrated robustness against ambient noise and reverberation. These neural network-based methods offer the advantage of learning noise-invariant patterns directly from data, reducing the need for manual feature engineering.

III. Method

A. Fingerprinting

First, I will detail the way the chosen files from the GTZAN dataset are manipulated by the fingerprintBuilder function to obtain the fingerprint database. The original audio is converted into a time-frequency representation using the Short-Time Fourier Transform. Next, we want to identify salient features in the resulting spectrogram, which we do by applying peak picking on the logarithm of the spectrogram by using the peak_local_max function from the skimage library. This produces a constellation map — a sparse collection of coordinates of high-energy peaks. A hashing method is then applied to the constellation coordinates to reduce the frequency resolution by grouping frequencies into broader bins. The motivation for this is two-fold: to help reduce computational time and to reduce sensitivity to slight variations in frequency in case that is a potential issue (the question of whether this is indeed a problem is discussed later). Hashing is done by dividing the frequency component of each constellation coordinate by a given bin size (initial value selected was 4).

The outcome of the above - a list of hashed coordinates in the form of (frequency bin, time) pairs - was stored to create the fingerprint database, this being the last step in the fingerprintBuilder function. An example of an STFT spectrogram with its corresponding constellation map and hashed constellation map is shown to give a visual example of this aspect of the system.

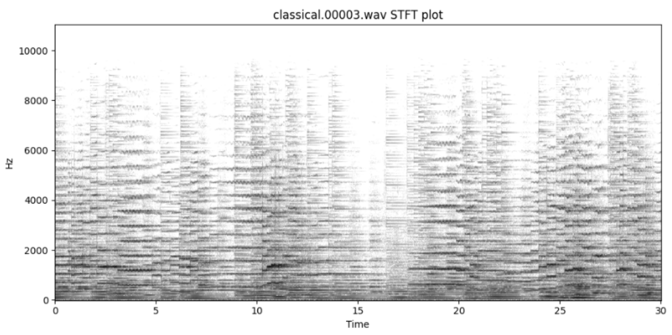

Fig. 1. Spectrogram of the input audio file

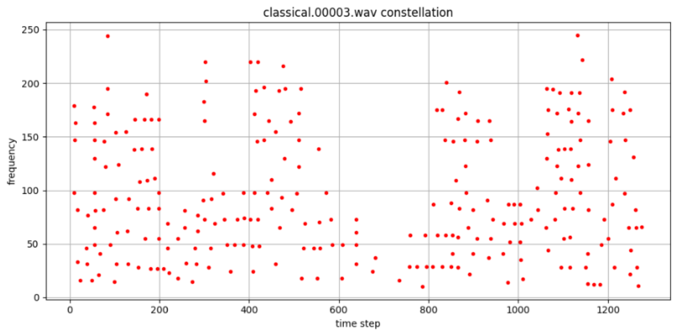

Fig. 2. Constellation of the input audio file

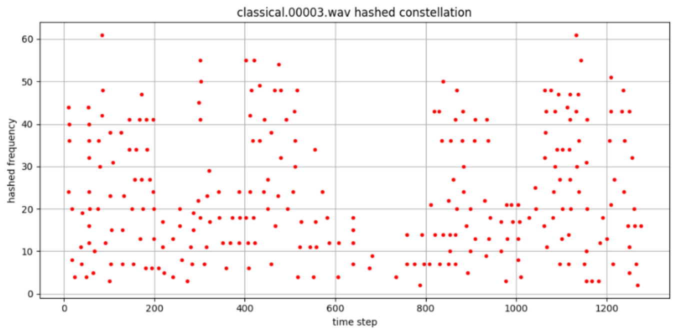

Fig. 3. Hashed constellation of the input audio file

B. Audio Matching

We now turn to the series of functions which will be called in the audioIdentification function. Before the necessary steps towards audio identification can be taken, the queries are processed in a similar way to the database files to get a dictionary which gives the hashed constellation maps for each query. The coordinates are flipped to [freq_bin, time] to ensure consistency with the way time shifts are computed later during matching.

Both the database and the queries have now been processed to obtain constellation maps. One way that audio identification could be done would be to now take each query, and for every file in the database slide the query constellation across the fingerprint constellation time bin by time bin, to find the point across the fingerprint where the largest number of matching peaks is found and to note this score; after this is done for every database file the biggest score would indicate the best guess for audio matching. However, from the literature already discussed, it is known that a more computationally efficient approach to execute this process is to use inverted lists in order to reduce the search space by only looking at hashes that occur in the query (effectively reduces complexity by a factor of the number of inverted lists).

To facilitate this, the hashed coordinates from each database track are converted into an inverted list with the help of the inverted_list_construction function, which takes each fingerprint and builds a dictionary where the keys are frequency bins, and the values are lists of time bins in which those frequency bins appear.

These inverted lists are used in the subsequent audio_matching function, which brings these inverted lists and the queries together to find the best match between query and fingerprint. For each [n,h] pair in a query, the database track constellation is examined to see if it is active for any times t at that frequency. If that is the case, an inverted list L(h)-n is worked out, which in effect takes the times t and shifts them all by n. For each query and database peak at the same frequency bin, this effectively tells us how much later that event occurs in the database relative to the query. These t-n values are collected in the aggregated_shifted_lists, which acts as a scoring mechanism, telling us how many query and fingerprint peaks line up at each time shift. The highest scoring shift is selected to provide the overall score for the likeness between the query and fingerprint in question, and the three highest scoring fingerprints per query are returned as the top 3 audio identification guesses.

The audioIdentification function's role is to then bring together the query preparation, inverted list construction, and audio matching functions to get the output file with the top 3 matches for each query as required.

IV. Results and Evaluation

Based on the outlined method from the previous section, I carried out tests with different hyperparameter choices to identify the ones that yield the best performance. In initial testing, I experimented with changing the following compute_stft function arguments:

- n_fft: This defines the size of the Fourier transform window, i.e., how many samples are used to compute

TABLE I
EVALUATION RESULTS FOR DIFFERENT STFT HYPERPARAMETER COMBINATIONS

nfft	window	hop	Combined Top-1 (%)	Combined Top-3 (%)	Classical Top-1 (%)	Classical Top-3 (%)	Pop Top-1 (%)	Pop Top-3 (%)
1024	256	64	49.30	54.93	32.41	35.19	66.67	75.24
1024	256	128	49.77	54.93	33.33	35.19	66.67	75.24
1024	256	192	50.70	56.34	34.26	38.89	67.62	74.29
1024	512	128	59.62	63.85	44.44	45.37	75.24	82.86
1024	512	256	58.22	62.44	41.67	42.59	75.24	82.86
1024	512	384	55.87	64.32	38.89	44.44	73.33	84.76
1024	1024	256	66.67	71.83	52.78	56.48	80.95	87.62
1024	1024	512	63.85	67.61	47.22	49.07	80.95	86.67
1024	1024	768	58.22	67.61	38.89	45.37	78.10	90.48
2048	512	128	60.09	65.73	46.30	47.22	74.29	84.76
2048	512	256	59.62	64.79	46.30	47.22	73.33	82.86
2048	512	384	56.81	63.85	42.59	45.37	71.43	82.86
2048	1024	256	67.61	73.71	53.70	54.63	81.90	93.33
2048	1024	512	66.67	71.36	50.00	51.85	83.81	91.43
2048	1024	768	61.50	69.01	44.44	48.15	79.05	90.48
2048	2048	512	**72.77**	77.46	**58.33**	**60.19**	87.62	95.24
2048	2048	1024	66.67	71.83	52.78	52.78	80.95	91.43
2048	2048	1536	62.44	68.08	44.44	45.37	80.95	91.43
4096	1024	256	69.95	76.53	55.56	59.26	84.76	94.29
4096	1024	512	68.08	72.77	51.85	53.70	84.76	92.38
4096	1024	768	61.50	68.54	45.37	50.00	78.10	87.62
4096	2048	512	**72.77**	77.93	55.56	58.33	**90.48**	**98.10**
4096	2048	1024	68.54	74.18	54.63	54.63	82.86	94.29
4096	2048	1536	64.79	71.83	49.07	52.78	80.95	91.43
4096	4096	1024	72.30	**78.40**	56.48	59.26	88.57	**98.10**
4096	4096	2048	68.08	74.65	51.85	55.56	84.76	94.29
4096	4096	3072	56.81	65.26	41.67	46.30	72.38	84.76

each FFT. Increasing this value means time resolution is worsened but frequency resolution is improved.

- win_length: This defines the length of the windowing function applied to the signal before each FFT. Shorter windows can better capture sharp attacks, but with increased spectral leakage, whereas longer windows give smoother spectra and better frequency resolution but miss short events.
- hop_length: This defines how many samples you skip between consecutive windows, so a smaller value would give a smoother and denser spectrogram in the time dimension, with better overall time tracking but higher computational cost, while a larger value would reduce computational cost but might result in more events being missed or blurred.

TABLE 1 shows the results of these experiments. Combined Top-1 (%) indicates the percentage of queries which were correctly identified just from the first of the three returned match guesses, while Combined Top-3 (%) considers the results a success of any of the three returned guesses are a match. The best performance in terms of Combined Top-1 (%) is observed with nfft = 2048, window = 2048, and hop = 512, as well as with nfft = 4096, window = 2048, and hop = 512, both giving a value of 72.77%, while the choices nfft = 4096, window = 4096, and hop = 1024 gave a marginally better performance if Combined Top-3(%) is chosen as the most important metric. In many cases such as Shazam-like use of the system, Top-1 accuracy is more relevant as we only return one matching file from the database to the user, so this is the metric I will focus on further. Among the two sets of hyperparameter choices which give a joint-best performance of 72.77%, further tests (with the code being run to get results just for these choices as opposed to iterating over a grid of hyperparameter combinations) showed that the choices nfft = 2048, window = 2048, and hop = 512 led to faster result generation at 28 seconds, compared to 54 seconds with choices nfft = 4096, window = 2048, and hop = 512. Thus, I have deemed this as my preferred hyperparameter choice.

One clear pattern for all nfft and window length choices is that a hop size which is 25% the length of the window size always outperforms the other possible choices of 50% and 75%. This can be explained by the denser spectrogram and reduced blurring resulting from smaller hop sizes, so 25% should be chosen in general, unless an attempt to reduce computational cost is desired in which case a bigger hop size could be implemented.

The window size changes also show some clear effects. For nfft choices of 1024 as 2048, having the window size be equal to the nfft gives superior results to having it be 50% or 25%, showing that if each FFT is calculated with fewer time samples, trying to boost frequency resolution by increasing window size is beneficial. However, at nfft=4096, having the

TABLE II
EFFECT OF HASHING BIN SIZE ON ACCURACY AND RUNTIME

Bin Size	**Time**	**Combined Top-1 (%)**	**Combined Top-3 (%)**	**Classical Top-1 (%)**	**Classical Top-3 (%)**	**Pop Top-1 (%)**	**Pop Top-3 (%)**
4	**28s**	72.77	77.46	58.33	60.19	87.62	95.24
2	3m29s	76.06	80.75	62.96	65.74	**89.52**	**96.19**
1	4m50s	78.40	**82.16**	**67.59**	**68.52**	**89.52**	**96.19**

window size be 50% of the nfft value is the better choice, indicating that for larger nfft values which are looking at a wider time window, it is worth trying to somewhat emphasise capturing sharper attacks in the time dimension by lowering the window size.

After these STFT hyperparameter choices were fixed, I explored the possibility of altering the freq_bin_size argument in the hashing function, which determines how many frequency bins from the original constellation map are grouped together to give one frequency bin in the hashed constellation map. The value that had been chosen in the previous experiments I detailed was 4. Changing this to 2, and subsequently to 1 (essentially eliminating any hashing) gave the results shown in TABLE 2.

Eliminating hashing altogether clearly led to improved performance, meaning that my concern about sensitivity to slight variations in frequency as highlighted in the method outline was not a real issue. This is not surprising in the context of this experiment as I didn't anticipate any pitch-shifting between the original database files and the noisy queries, so the elimination of hashing is a clear path for a more detailed frequency dimension. Nevertheless, the elimination of hashing comes with the notable trade-off of a significant increase in computational time. Hence, the decision of which approach to favour can be made based on context. For example, someone seeking a quick audio matching for some background music they hear in a public space might want to favour a freq_bin_size of 4 or even 8 for quicker results, while in the case where time is not so essential (perhaps in some musicological endeavour) the elimination of hashing will help ensure the correct matching happens a greater percent of the time.

V. DISCUSSION AND POTENTIAL IMPROVEMENTS

One consistent trend throughout all tests is that pop music is matched much more reliably than classical, as for all configurations, the top-1 and top-3 accuracies for pop are significantly higher (often 20–30 percentage points more) than for classical. One possible reason for this is that pop songs generally have sharper attacks (for example due to instruments like drums), producing clearer and more distinctive peaks, whereas classical music might exhibit longer legato gestures (such as with string instruments) which makes it harder to obtain distinctive constellation maps from them.

There are some potential ways in which future improvements could try and boost performance on classical music. Using Mel-spectrograms instead of linear STFT could improve frequency resolution in lower frequency bands which can often be very detailed in classical music, and perhaps chroma-based constellations could be attempted too as classical music often heavily emphasises harmonic content. Moreover, classical music's sudden dynamic changes and extended passages exclusively in very loud or very soft dynamics poses a challenge for peak-picking, which is usually not an issue of the same severity in pop music which has much more uniform dynamics. My proposed solution to this problem is to use a peak-picking method which will be able to identify prolonged loud and quiet periods through changes in energy, and raise or lower the peak-picking threshold accordingly so that an appropriate number of peaks is always being selected.

Another approach which would most likely result in enhanced performance and has already been implemented by various researchers is to follow a hashing approach that takes pairs of peaks [7], or even one that constructs triplets, with potential use of time and pitch ratios to ensure time and frequency invariance if desired [5].

Finally, I used the system with optimal hyperparameters and returned the file names for some of the queries for which the correct database file was not in the top three guesses. I then listened to some of these queries to see if any meaningful observations could be made, and whether these were queries for which I could reasonably expect to improve performance. Some query files, like classical.00003-snippet-10-10.wav and pop.00019-snippet-10-0.wav had a very significant noise level with very faint music that was just about audible. It is possible that pre-processing with appropriate denoising filters could provide a potential solution in this case. There were also some files, such as classical.00019-snippet-10-0.wav and pop.00079-snippet-10-0.wav, for which I could not hear any music at all when listening. It might thus be unreasonable to suggest that the model used ought to be able to return correct results for these files, and its poor performance here need not be held against it.

REFERENCES

[1] Bhattacharjee, Aditya, Shubhr Singh, and Emmanouil Benetos. "GraF-Print: A GNN-Based Approach for Audio Identification." ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2025.

[2] Dupraz, Elsa, and Gaël Richard. "Robust frequency-based audio fingerprinting." 2010 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2010.

[3] Haitsma, Jaap, and Ton Kalker. ”A highly robust audio fingerprinting system with an efficient search strategy.” Journal of New Music Research 32.2 (2003): 211-221.

[4] Chu, Renjie, et al. ”Peak-based Philips fingerprint robust to pitch-shift for massive audio retrieval.” 2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM). IEEE, 2019.

[5] Joren, Six, and Marc Leman. ”Panako-A scalable acoustic fingerprinting system handling time-scale and pitch modification.” Proc. ISMIR. 2014.

[6] Chang Sungkyun, et al. ”Neural audio fingerprint for high-specific audio retrieval based on contrastive learning.” ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021.

[7] Wang, Avery. ”An industrial strength audio search algorithm.” Ismir. Vol. 2003. 2003.