Music Informatics Assignment 2: Song Identification

Jamie Pond, *MSc Sound And Music Computing*
(ec20768@qmul.ac.uk)

Queen Mary University of London, London, UK

In this assignment we will examine existing literature and develop a song identification algorithm based on methods from Meinard Müller's *Fundamentals of Music and Audio Processing* [1], and also the lecture content and slides of module ESC7006P (Music Informatics) at Queen Mary University of London[2]. Ideas and concepts from Avery Wang's *An Industrial Strength Audio Search Algorithm* [3] have also been implemented and referenced. Testing and development was completed on a subset of the GTZAN [4] data-set using 100 classical clips, and 100 pop clips. The best measured performance of the algorithm was 95.2% correct identification for pop and 75.0% for classical.

0 INTRODUCTION

There are many different use cases for audio identification algorithms, such as song identification for end-users e.g. Shazam [3] and in broadcast with copyright monitoring. This therefore motivates the development of audio identification algorithms, with respect to a commercial goal.

The following approach derives from Wang's concept of constellations, which is a form of binary matrix used to represent time-frequency information, as a method of encoding the essential song data, but then goes onto use the inverted list method of shifting and matching as a method of comparing a query constellation to any given database entry. A number of different parameters such as different STFT lengths, hop sizes and constellation 'neighborhood' sizes were auditioned to try and increase performance, but performance of overall 78.4%, 75.0% for classical and 95.4% for pop could not be bettered within the time-frame available for this assignment.

1 IMPLEMENTATION

The implementation of this algorithm is outlined below:

1) Generate audio fingerprints for all database items (stored as inverted lists).
2) Generate constellation of the query clip.
3) Compare each database item to the query clip using and record their 'similarity measure', as defined by the matching function $\Delta_F(m)$ [2]
4) Rank each of the entries based on their similarity measure.
5) Return the top 3 ranked results in an output text file.

There is also a richer, more insightful output that is generated as the database of queries is processed, that outputs the top 10 ranked results, along with their similarity measure, which was helpful in development in order to be able to see the confidence of the predictions. For example:

```
=========================================
Analyzing pop.00050-snippet-10-20.wav ...
The top 10 matches are:
[Rank | Clip Name | Similarity Measure]
** [1, 'pop.00050.wav', 77] **
   [2, 'pop.00065.wav', 6]
   [3, 'pop.00055.wav', 6]
   [4, 'pop.00069.wav', 6]
   [5, 'classical.00067.wav', 5]
   [6, 'classical.00056.wav', 5]
   [7, 'pop.00032.wav', 5]
   [8, 'pop.00030.wav', 5]
   [9, 'pop.00023.wav', 5]
   [10, 'pop.00098.wav', 5]
I predicted that the query is:
pop.00050.wav => CORRECT
Current performance overall: 75.5 %
```

```
Classical % Correct: 63.0 %
Pop % Correct: 96.9 %
Approx time remaining: 1 minutes.
```

Knowing an approximation of time remaining to process the database was helpful in maximising the number of variations that could be tested in the available time for this assignment. The time taken to process the whole query set was no more than 10 minutes, and as low as 2 minutes. This means that it only takes about half a second to process one query against the whole database.

1.1 Generating the database

The following operation was performed on each database audio file, in order to obtain each corresponding audio fingerprint. In this case this operation returns an inverted list, which is the smallest lossless representation of the constellation map which is compatible with our matching function.

Our database is defined as D, for number of documents N. Here we return constellation C. All audio files are normalized before this operation.

$$C_n = peak(log(|STFT(D_n)|)), \quad \forall\, n \in [1:N] \tag{1}$$

Where $STFT$ is the Short-Time Fourier Transform, as given by the `librosa` Python library, and the *peak* function the the `skimage` library `peak_local_max` peak picking function. The parameters given to the peak picking and STFT algorithms are discussed in later sections.

The constellations are in the form of a binary matrix, and can be translated in to the inverted lists as described by the ECS7006P module slides as follows. The inverted lists are a 'list of lists' where timestamps of peaks are stored for each frequency bin. For example, if there are 256 frequency bins, there will be 256 'lists of lists', where each sub-list will contain time stamps of corresponding peaks.

There are F number of inverted lists generated per database document which are generated as follows, where T is the total number of timestamps in the constellation, and F is the total number of 'frequency bins':

$$L_f := \; t \; if \; C_{f,t} = 1, \;\; \forall\, t \in [1:T], \;\; \forall\, f \in [1:F] \tag{2}$$

This step would be repeated for all input audio files' corresponding constellation C. This operation is more clearly interpretable from the provided source code in the `inverted_list.py` file. This inverted list is stored in a `database` entry D_n that is stored in the `database_file.pickle` in a list with the string of the original file name, as shown below:

$$D_n := \{\text{file name}_n,\, L_n\}, \;\; \forall\, n \in [1:N] \tag{3}$$

1.2 Query the database

The steps taken to query the database for a given query audio file Q are outlined as follows:

1) Convert the sampling rate of the query item to the sampling rate of the database items.
2) Normalize Q to bring it up to full range loudness (peak at 0dB).
3) Generate the constellation map of the query.
4) Generate a list of the coordinates of the peaks of the constellation map in the form $Q(n,h)$ where n is the timestamp index and h is the hash index.
5) Generate a corresponding match list with the formula $L(h) - n$.
6) Our best match timestamp between Q and any given database entry D_n is given by the most frequent entry in the whole of the new $L(h) - n$ structure. This maximum values is taken as the 'similarity measure' for this D_n.
7) After having processed all D with respect to Q, the D with the highest similarity measure is declared as the match prediction.

In the testing and evaluation of the algorithm, it is also essential to be able to assess if the prediction is correct, so a matching function is also required. This has been implemented narrowly for the given data-set, by simply comparing the first number of relevant characters of the query file name, to the known correct file name. This would need to be altered in order to evaluate at a larger scale.

2 DISCUSSION

A wide range of parameters were tested for this algorithm in order to attempt to optimize its performance. Here, we will try to demonstrate the iterative approach that was taken to achieve the results at hand.

Looking at Table 1, we can see the variability in the possible results, and it is likely that the parameters that have been selected for this algorithm are over-fitted to the given data, and this model may not perform as well on a different, but overall similar data-set.

The parameters that made the most difference were the STFT length and window size. The best parameter settings that were found were an STFT length of 2048, a window size of the same and a hop size of 128. This is the same for the database fingerprint encoding and query fingerprint encoding. It would have improved the work done in this assignment to have experimented with different settings for each.

In general the higher the STFT size and window size, the better the algorithm performed. Hop size seemed to also contribute, with lower hop sizes generally yielding better results.

In the following table, we see how the relevant corresponding STFT parameters affect the performance of the audio identification algorithm.

The every halving (lower power of 2) used as the hop size, the computational expense increases by a factor of 2. This does not appear to generate better results overall, but does seem to slightly change the inter-genre performance. However, this effect is small and may still be a function of the limited size of the data-set. This could therefore inform a future effort to improve this algorithm, which could perform a different STFT size on the query, depending on

Table 1. Demonstration of variation of % correct prediction. (% correct)

Pop	Classical	Overall
95.2	74.1	84.5
94.3	75.0	84.5
94.4	62.9	78.4
89.5	63.9	76.5

Table 2. Variation of STFT parameters, corresponding to Table 1 (samples)

FFT Size	Window Size	Hop Size
2048	2048	256
2048	2048	128
1024	1024	512*
512	512	128

* This is the FFT parameters for the STFT given in the ECS7006 lab material.

the detected genre, assuming a corresponding database fingerprint is also available. This would increase the cost of the implementation in terms of CPU cycles and in terms of raw storage required to store additional fingerprints per audio clip, so further study would be required to determine the value of this additional expense.

2.1 Normalization

In the development of this algorithm, peak normalization for both database and query clips before the STFT was implemented as a course of, what felt like common sense to the author, after having examined the waveforms of the clips in a digital audio workstation. All the waveforms were at radically different and varying peak levels.

The following operation was applied to all audio clips X, giving peak normalized output Y.

$$Y = \frac{X}{max(X)} \tag{4}$$

This appeared to have a positive effect on the performance of the algorithm, but only if the query set was normalized. If the query data-set was normalized this improved performance by around 5%, as shown in Table 3. However it did not seem to matter if the database samples were normalized. Because they are commercial recordings they are likely already have a peak normalization around 0dB.

This effect may be the case by implicitly setting a more optimal relative threshold for the peak picking algorithm. Therefore, with further experimentation, it may be possible to better optimize the performance of this algorithm, however, a larger data-set should be used to avoid over-fitting of these parameters.

Table 3. Effect of sample peak normalization on overall performance.

Normalization	Overall Performance (% correct)
Both Q and D normalized	84.5
Only D normalized	80.3
Only Q normalized	84.5
Neither D or Q normalized	80.3

2.2 Mel spectrograms

The equivalent Mel spectrogram constellations were generated with the expectation that they would out perform the standard STFT approach, however this was not the case. The author proposes that there was a limitation in the implementation of the code of this algorithm which lead this to being the case, and proposes a further investigation into the use of Mel spectrograms in this application.

2.3 Further evaluation

The ranking function in the code is not as robust as it could be. When there are several matches with the same similarity measure, depending on the chance order of those entries, the algorithm will be told it is correct, even though it did not make a firm decision. This is an edge case scenario, but it should be noted that the real performance of the algorithm may be marginally lower than reported.

3 CONCLUSION

In this assignment we have examined how different STFT parameters contribute to differing levels of performance in an audio identification task, with generally higher FFT sizes helping performance. The use of normalization of query clips it would seem also leads to increased performance on, at least, the given data-set. The author proposes that all audio should be normalized in order to standardize the process and algorithm.

One might propose a further examination of setting different peak picking thresholds for database and query samples which may lead to increased performance.

Though the Mel spectrogram approach was not beneficial to the performance of this implementation, one would imagine that there may still be benefit in exploring the use of representations that more closely match human hearing range patterns.

4 ACKNOWLEDGMENT

This assignment was completed as a part of the ECS700P Music Informatics module at Queen Mary University of London, taught by Professor Simon Dixon and Dr. Emmanouil Benetos. The author would like to thank both Professor Dixon and Dr. Benetos for their outstanding teaching and support throughout the module.

Their dedication to, and expertise of the field of Music Informatics has been instrumental in inspiring the author

to achieve their best, and to help develop their skills as a mathematician, engineer and programmer.

5 REFERENCES

[1] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications* (Springer), 1st ed.

[2] E. Benetos, "Week 9 - Audio Identification," ECS7006 Music Informatics.

[3] A. L.-C. Wang, "An Industrial-Strength Audio Search Algorithm," p. 7.

[4] G. Tzanetakis, G. Essl, P. Cook, "Automatic Musical Genre Classification Of Audio Signals," (2001).

THE AUTHOR

Jamie Pond

Jamie Pond is a software developer at PresentDayProduction and an MSc Sound and Music Computing at Queen Mary University of London, where his research interests include audio and music applications of signal and array processing, C++ programming and design and development of augmented instruments. From 2017 to 2020, Pond was an undergraduate student at the London College of Music, where he studied Music Technology with a focus on mixing and mastering. During this time, Pond has spent many hours working in commercial recording studios in and around London, including Cosmic Audio in Epping, and RYP Studios in Harrow. In the last two years, Pond has started developing audio plugins with PresentDayProduction, with an aim to create fun and engaging products that creatives resonate with. Pond was also the Vice-President of the Audio Programming Society at UWL, where he organised evening lectures with guest speakers such as Josh Hodge of The Audio Programmer and Professor Eric Tarr of Belmont University, on such subjects as developing audio compressor and reverb plugins in C++ with JUCE.