Assignment 2: Audio Identification

Matthew Rice
Queen Mary University of London
m.rice@se22.qmul.ac.uk

1. Introduction

The goal of this assignment was to create an audio querying system. This system would maintain a database of audio recordings of a specified length (hereafter called fingerprints), then have a method of querying the database with shorter segments of these recordings which also may be possibly corrupted by background noise or a different recording environment. A well-known commercial product of this system is Shazam[1], which is based on [6].

Recently, deep-learning-based methods have been proposed to handle such tasks. This is due to the need for high-quality, easily-searchable audio embeddings which deep learning models excel at creating, but only under certain conditions. In the implementations detailed below, I compare a simple spectral hashing approach based on [6] to a modified version of a recent deep learning implementation using the same GTZAN dataset. Because of the size of the dataset, the spectral hashing approach far outperforms the deep learning implementation. [2]

2. Fingerprinting Theory

The querying system consists of two components: a database, which computes and stores a fingerprint for an audio file, and a fingerprint search tool which computes the fingerprint and finds the most related fingerprint for a given input audio. In the real-world scenario, it is likely that we will have thousands if not millions of audio files in our database, and our inputs to the search tool likely will be shorter and corrupted versions of the files in our database. Because of these restrictions, there is a need for fingerprints that are quick to compute, compact, scalable, robust against noise and distortions, and unique. These fingerprints also need to be quick to search. However, improvements in one requirement usually cause a decrease in performance in others [1].

[1] https://www.shazam.com/

[2] Code for the spectral hashing method can be found at https://github.com/mhrice/Spectral-Hashing-Fingerprinting

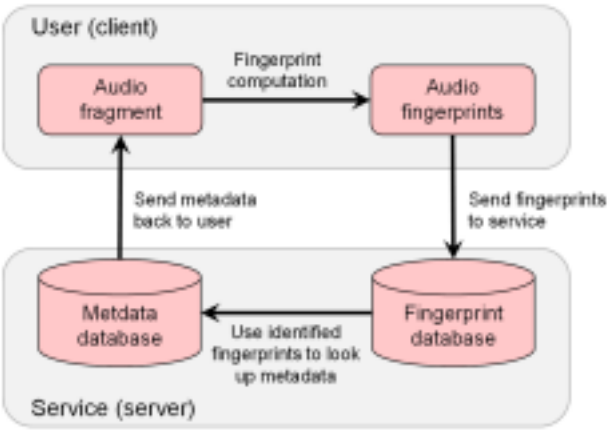

Figure 1: End-to-end audio identification system

3. Dataset

The dataset used for these experiments is the GTZAN dataset. This dataset consists of two subsets. Subset 1 is the database recording which consists of 200 audio files of recordings of classical and pop songs of 30 seconds in length. Subset 2 contains the query recordings, consisting of 213 audio files corresponding to one of the files from subset 1. These files are 10 seconds sections of the corresponding file, recorded on various smartphones in noisy environments.

4. Spectral Hashing Implementation

4.1. Theory

Luckily the short-time Fourier transform (stft) already contains many of these properties. It is somewhat compact, it is quick to compute, unique, and because many distortions can be modeled as a uniform distribution in the frequency spectrum, it can be made robust against noise. However, it is not easily searchable or scalable. To solve these issues (and fully solve the noise robustness), we can perform peak-picking on the stft and smartly save these points as a hash so that they can be easily searched. This peak-picking repre-

sentation is known as a constellation map. When querying the database, I follow the same process to create the constellation map, then for each database constellation map, I find the best location where the query constellation map lines up with the database constellation map, known as the match, and select the database constellation maps with the highest matches.

4.2. Creating the fingerprint database

Given a database audio file x:

1. Compute $X = stft(x)$, with hop size H and fft size N. X is now a 2d frequency/time representation of x.
2. Save the coordinates of the peaks of X defined by a specified threshold T and minimum distance M.
3. Compute inverted hash lists of peaks of X (lists of peak timesteps for each frequency bin).
4. Save the outputted hash lists to disk as binary data.

4.3. Querying the fingerprint database

Given a query audio file q and loaded database hash lists D:

1. Compute $Q = stft(q)$, with hop size H and fft size N. X is now a 2d frequency/time representation of q.
2. Save the coordinates of the peaks of Q defined by a specified threshold T and minimum distance M.
3. $\forall\, d \,\epsilon\, D$, create indicator functions using d(h) - n $\forall$peak coordinates $n, h(\,\epsilon$ Constallation. Then sum indicator functions and set the match score to the highest element. This represents the shift that best corresponds to the query.
4. Select the top k database files corresponding to the top k highest match scores.

4.4. Implementation Details

In my final version, I set $H = 512$, $N = 2048$, $T = 0.05$, $M = 10$, $k = 3$. All files are sampled at a sample rate of 44100.

5. Contrastive Learning

5.1. Theory

The deep learning implementation is a simplified version of the Neural Audio Fingerprint paper [2]. The approach used was a modified version of SimCLR[3], a famous contrastive learning method for self-supervised learning of images. The main idea is to train a convolutional neural network to map an audio file and its augmented pair to an embedding space. The model is trained such that the distance

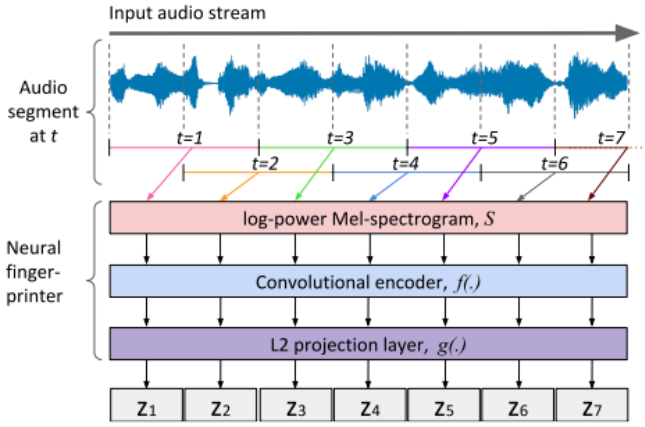

Figure 2: Overview of audio embedding procedure as outlined in

(inner product) between the audio pair embeddings are minimized while the distance between the audio pair embeddings and the other audio embeddings are maximized. The augmented pair is created with various corruption augmentations to mimic a real-world scenario. For searching, the paper proposes a more optimal searching method a, however, I simply used the model to extract the query embedding and performed cosine similarity, choosing the top results. One notable difference is that instead of using stft to represent the audio, they used mel spectrogram as previous approaches had found it performed better than stft.

5.2. Data Augmentation

In the original paper, several data augmentation methods were used to create the audio augmentation. These were time offset modulation, background mixing, IR filters, cutout, and spectral augment. I implemented all of these methods with some simplifications. Instead of using audio set for background mixing, I resorted to ESC dataset[4] due to its simplicity in downloading and using the files. For similar reasons, I used the MIT IR dataset [5] instead of the Xaudia and AIR datasets for the IR filters. Finally, I created my own probability-based cutout and spectral augment methods which are likely different than the paper's implementation. See the paper for the descriptions of these augmentations.

5.3. Creating the fingerprint database

Given a batch of database audio files x:

1. Split x into 1-second samples xn with a hop of 0.5 seconds
2. Perform data augmentation to get xn_aug
3. Compute $X = mel(xn)$, $X_aug = mel(xn)$.

Method	Accuracy		Runtime	
	Top-1 (%)	Top-3 (%)	CPU	Search Time (s)
Spectral Hashing	83.1	86.9	Apple M1 Max	2.96
Contrastive Learning	4.1	6.9	Apple M1 Max	5.07
Chang et al. [2]	95.8	(unreported)	Intel i9-10980XE	1.5

Table 1: Performance of the audio identification systems

4. $F = CNN(X), F_aug = CNN(X_aug)$ to encode X, X_aug into h-dimension embedding.
5. $G = L2(F), G_aug = L2(F_aug)$ to project F, F_aug into d-dimension embedding.
6. Calculate contrastive loss L between G and G_aug using NTxent and update CNN and L2 to minimize L.
7. When converged, perform inference on the model, saving a mapping of the output embeddings (G) with the file name. Also save the model checkpoints.

5.4. Querying the fingerprint database

Given a query audio file q, embedding map e and converged models CNN and L2:

1. Compute $Q = mel(q)$.
2. $F = CNN(Q)$ to encode X into h-dimension embedding.
3. $G = L2(F)$ to project F into d-dimension embedding.
4. $\forall\ e\ \epsilon\ E$, calculate cosine similarity between G and e.
5. Select the top k database files corresponding to the k highest cosine similarities.

5.5. Implementation Details

In my final version, I set $h = 1024,\ d = 64,\ k = 3$. All files are sampled at a sample rate of 44100. I also used a slightly smaller batch size of 128 due to GPU memory limitations. Please see the paper for a more complete implementation as well as the dataset details. All parameters used in this approach are the same as in Table 2 of the paper.

6. Results and Discussion

I evaluated the two implementations above as well as the reported results of x using top 1 accuracy and top 3 accuracy. The audio identification search time for a 10s query file is shown as well. Table 1 shows these results. While the baseline spectral hashing system performs moderately well, it is clear that the implementation of the contrastive learning approach performs very poorly. However, it did not completely fail as it does performer much better than top-1 probability of choosing at chance (0.47%). This is likely due to the lack of data used to train the models. As shown in [7], the quality of the representations is significantly reduced with sparse numbers of samples. In the original paper, their training dataset was more than 50 times larger than GTZAN. Also, their searching method was much improved than mine, and likely explains the runtime difference. Overall, it seems like for small datasets, a version of the spectral hashing system would work well. Perhaps with modifications such as grouping frequency bins together, using pairs of hashes with an anchor hash, and multi-resolution stft approaches could improve the accuracy and runtime.

7. Conclusion

Overall, these results show that it is possible to achieve quick, compact, scalable, robust, and unique fingerprints. Although neither of the proposed systems are able to maximize all the metrics to the fullest, they do attempt to have the best compromise between them all. Also, if one wants to build a successful implementation of contrastive learning techniques for audio fingerprinting and identification, more data is needed. Moreover, the paper present impressive results not only on finding the correct file, but also on the location of the fingerprint in the file. Finally, as part of the submission, I am providing the code for the spectral hashing implementation, due to performance. My implementation of the contrastive learning method can be found here: https://github.com/mhrice/AudioFingerprint.

References

[1] E. Benetos. Week 9 - audio identification.

[2] S. Chang, D. Lee, J. Park, H. Lim, K. Lee, K. Ko, and Y. Han. Neural audio fingerprint for high-specific audio retrieval based on contrastive learning. version: 4.

[3] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations.

[4] K. J. Piczak. ESC: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, MM '15, pages 1015–1018. Association for Computing Machinery.

[5] J. Traer and J. H. McDermott. Statistics of natural reverberation enable perceptual separation of sound and space. 113(48):E7856–E7865. Publisher: Proceedings of the National Academy of Sciences.

[6] A. L.-C. Wang. An industrial-strength audio search algorithm.
[7] Y. Wu, Z. Wang, D. Zeng, Y. Shi, and J. Hu. Data-efficient contrastive learning by differentiable hard sample and hard positive pair generation.