SPARSE CODING OF AUDITORY FEATURES FOR MACHINE HEARING IN INTERFERENCE

Richard F. Lyon, Jay Ponte, and Gal Chechik

Google, Inc.

ABSTRACT

A key problem in using the output of an auditory model as the input to a machine-learning system in a machine-hearing application is to find a good feature-extraction layer. For systems such as PAMIR (passive–aggressive model for image retrieval) that work well with a large sparse feature vector, a conversion from auditory images to sparse features is needed. For audio-file ranking and retrieval from text queries, based on stabilized auditory images, we took a multi-scale approach, using vector quantization to choose one sparse feature in each of many overlapping regions of different scales, with the hope that in some regions the features for a sound would be stable even when other interfering sounds were present and affecting other regions. We recently extended our testing of this approach using sound mixtures, and found that the sparse-coded auditory-image features degrade less in interference than vector-quantized MFCC sparse features do. This initial success suggests that our hope of robustness in interference may indeed be realizable, via the general idea of sparse features that are localized in a domain where signal components tend to be localized or stable.

Index Terms— Auditory image, sparse code, PAMIR, sound retrieval and ranking

1. INTRODUCTION

We recently presented a detailed description of a sound ranking and retrieval system based on sparse coding of auditory images [1]. In this paper, we review the adopted strategy for sparse coding, and present the results on new tests design to determine whether the coding strategy has the intended feature of robustness to interference.

2. BACKGROUND

The layer of our system that generates sparse feature vectors was developed as a quick first try between a more principled auditory model based on a cochlea model with stabilized auditory image (SAI) generation, and a more principled machine-learning system based on training with a ranking objective. Based on successes in sparse coding of images and video, this sparse coding is the first attempt that we know of to interface the high-dimensionality SAI (tens of thousands of dimensions, or pixels, per frame) to a machine-learning system that can take advantage of the richness of the information. We were able to demonstrate good performance on the ranking/retrieval task, beating the best we could do with vector-quantized MFCCs, but we were left with little insight into whether the sparse coding was really capturing local and robust features of sound components, as we hoped. By developing a harder task with sound mixtures, and doing sound retrieval on that set, we sought to gain more insight into whether this approach to feature extraction was in fact getting a benefit from the structure of the SAI.

Sparse codes in the waveform domain have been explored by several groups [2,3]. The events or “spikes” in such codes represent sound “particles” that are usually fairly compact in the time–frequency space, and hence too primitive to effectively map higher-level structure of typical sound sources. Sparse coding has also been done in the space of spectral slices, or spectrogram time-frequency products [4,5]. These work fairly well for encoding musical notes in polyphonic

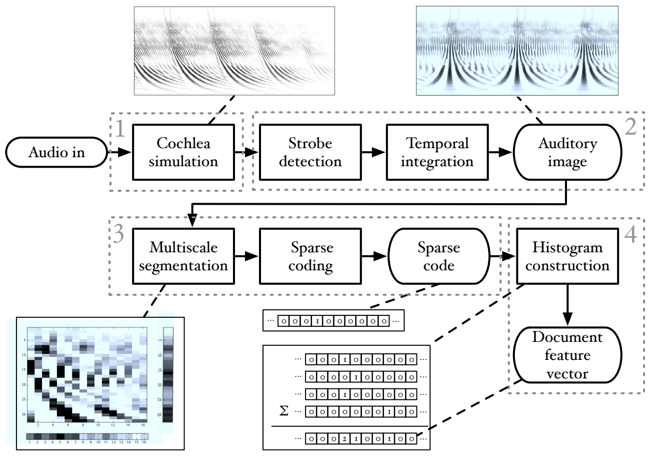

Fig. 1. Our process of generating sparse codes includes an auditory model that produces stabilized auditory images (SAIs) and a local box and vector quantization based way of generating sparse codes from the SAI frames. Codes are counted (histogrammed) to make a sparse representation for the whole audio file.

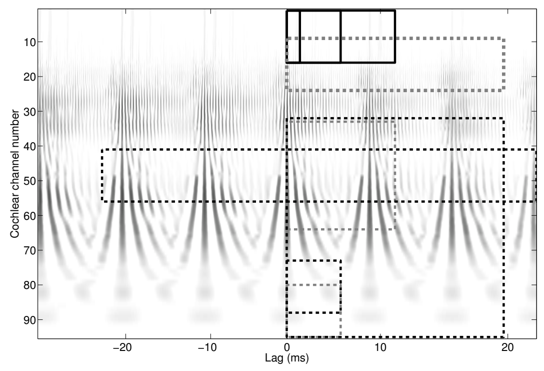

Fig. 2. The SAI is sparse-coded by many vector quantizers, each one looking at the contents of a local box, with different widths, heights and positions, some of which are illustrated here. The system reported used 49 different local boxes, giving a collection of sparse (1-of-M) sub-features at different scales of time and frequency resolution. The feature vector for an SAI frame is the concatenation of the sub-features from the boxes, which still has a sparsity of $1/M$, where M is the number of codewords per VQ codebook.

music, which is what they were designed for, but are not well suited for sounds of a less periodic or regular sort.

The PAMIR (passive–aggressive model for image retrieval) method [6] uses a simple linear mapping (a learned weight matrix) to produce scores by which documents (audio files) are ranked in response to a query; the k documents with the highest scores are said to be “retrieved”, and the goal is to optimize the precision (percentage of documents deemed relevant to the query) within that top k. Classification problems have a slightly different objective, but similarly can benefit from large sparse feature spaces in combination with simple linear mappings. Tom Cover pointed out in 1965 that “A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated” [7].

3. THE EXPERIMENT

3.1. The task

The experimental task that we focused on is retrieval of audio files in response to text queries. For example, if a user enters a query “fast car”, we want to see sound files retrieved that sound like they have something to do with “fast” and “car”. The system retrieves audio files by ranking them with a score function, and returning just the top-scoring documents. Since the number of documents and the number of queries are both potentially very large, index operations can be used to retrieve a list of candidate documents for each query term, to limit the number of documents for which the query needs to be scored. Still, we want a very fast and efficient scoring function. The PAMIR approach uses a fast linear sparse matrix multiplication, involving only as many matrix columns as there are query words in a query, to generate the score for a document. PAMIR also supplies an effective fast training algorithm to optimize the matrix.

The experiments reported are not such large scale that an indexing stage was needed.

3.2. The dataset

We collected a data set that consists of 7146 sound effects from multiple sources. Close to half of the sounds (3855) were collected from commercially available sound effect collections. Of those, 1455 are from the BBC sound effects library. The remaining 4783 sounds are taken from a variety of web sites; *www.findsounds.com*, *partners in rhyme*, *acoustica.com*, *ilovewavs.com*, *simplythebest.net*, *wav-sounds.com*, *wavsource.com*, and *wavlist.com*. Most of the sounds contain only a single “auditory object”, and contain the “prototypical” sample of an auditory category. Most sounds are a few seconds long but there are a few that extend to several minutes.

Most sound effects had associated labels, in the form of filenames, or tags. We manually labeled all of the sound effects by listening to them and typing in a handful of tags for each sound. This was used for adding tags to the existing tags (from *www.findsounds.com*) and to tag the non-labeled files from other sources. When labeling, the original file name was displayed, so the labeling decision was influenced by the description given by the original author of the sound effect. We restricted our tags to a somewhat limited set of terms. We also added high level tags to each file. For instance, files with tags such as ‘rain’, ‘thunder’ and ‘wind’ were also given the tags ‘ambient’ and ‘nature’. Files tagged ‘cat’, ‘dog’, and ‘monkey’ were augmented with tags of ‘mammal’ and ‘animal’. These higher level terms assist in retrieval by inducing structure over the label space. All terms are stemmed, using the Porter stemmer for English. After stemming, we are left with 3268 unique tags. The sound documents have an average of 3.2 tags each.

We created a set of sound mixtures by averaging pairs of sounds from this dataset. For each sound in the database, we randomly picked another sound from the data, truncated both sounds to the have the same duration, resampled to 22.5 kHz and computed the average of the two raw signals (in time domain). The new sound was labeled with the union of of the labels of the two original sounds. Since we had previously observed that most sounds were easily identifiable from the early part of the recording, we don’t think the truncation had much effect in making the task harder or easier, but we didn’t test that.

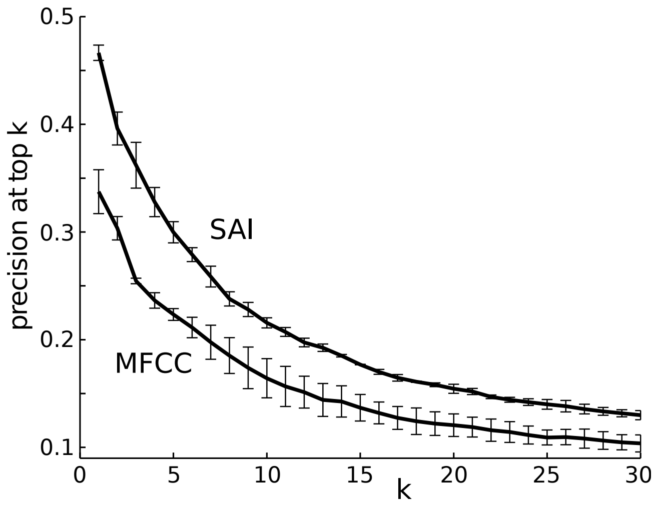

Fig. 3. Results of the experiment: precision at top k is significantly better for the SAI features than for the MFCC features, for all k values. Note that the y axis does not start at 0, so the effect is not as big as it looks at large k.

3.3. The training and testing

From this set of labeled sounds, three splits were made by randomly partitioning the data. Several constraints on the splits were tried, in an attempt to not test on queries or query terms that have no training data.

Though all the sounds are mixtures, we treat a sound mixture as "relevant" to a query if its combined tag set includes all the query terms.

For each of three splits, we hold out one third for testing and train a PAMIR scoring matrix on the other two thirds. As described before, queries and sound files were filtered such each query had at least 5 relevant sound files in each of the training and testing sets [1]. Using the trained matrix, we measure the number of relevant queries in the top k ranked results, and average over a frequency-weighted set of queries, for various k values, to get the precision at top k. Query frequencies were taken from the freesounds.org query logs.

We ran the same experiments with the SAI sparse code features and with sparse codes obtained by vector quantizing MFCCs (augmented with deltas and second deltas as in typical ASR systems).

4. RESULTS

As a comparison baseline, we first conducted a series of experiments representing sounds using vector-quantized MFCCs. We varied three parameters of this representation: The number of MFCC coefficients (we tested the standard 13, but also 25 and 40); the frame length (we tested 15, 25, 40 and 60 msec); and the number of kmeans centroids (1000, 2000 and 5000). The best precision was achieved using 40 coefficients, 40-msec frames and 5000 clusters. This result was consistent with previous experiments on pure sounds [1] (we previously had tried more than 5000 clusters, and found no improvement, but didn't repeat that on this test). The top-1 precision for MFCC was 0.337. The number of coefficients and frame length are large compared to typical practice in speech and music analysis, allowing representation of finer temporal structure.

For the vector-quantized SAI features, we tuned one parameter, the number of clusters used for quantization of each local box pattern (we tested 500, 750, 1000, 1500, and 2000 clusters per box). Performance varied smoothly and peaked at 1000 clusters per box ($M = 1000$ codewords per VQ codebook).

The top-1 precision for the SAI box features was 0.4663. This reflects a dramatic improvement of 39%, or equivalently a reduction of 20% in the top-1 error, relative to the best vector quantized MFCC features.

By comparison, the results previously reported for the baseline task of clean sounds were top-1 precisions of 73% for SAI and 67% for MFCC. It required 4000 clusters per box and many more total SAI features to achieve that difference. For 1000 clusters per box, the SAI top-1 results were very close to the best MFCC numbers (about 67%). That is, the best systems that we are comparing in interference started out with nearly equal precisions with clean sounds.

We further looked into the ranking precision of specific queries. For each query, we computed the average precision (a standard evaluation measure used in information retrieval), for both SAI and MFCC. Several queries yielded an average precision of 1.0 with SAI features (meaning that all of the N relevant audio files were ranked in the top N ranked results), and yielded average precision below 0.1 (meaning that probably many more than the top $10N$ were needed to find the N relevant audio files) with MFCC. These queries include "hit percuss", "cabasa", "bird chip", "woodblock", "dragster", "gargl", "heartbeat machin", "chip", "american psycho", "door knob", and "basketbal crowd", "nail wood", "park swing", and "comput mous" (note that many query terms have been truncated as part of the normalization). In the other direction, 1.0 for MFCC and less than 0.1 for SAI, there were fewer such queries: "nuthatch", "baboon", and "guitar loop". Looking at these and others, it appears that more impulsive sounds do well with SAI, while more tonal sounds do better with MFCC, though the pattern may be more complicated than that.

5. OTHER APPLICATIONS

We, and others, have used the same SAI box-and-VQ local sparse coding method to represent sound files for other applications. For example, Tzanetakis and Ness made an open-source version of the process in their Marsyas system [8] and used it among their entries in the MIREX (music information retrieval exchange) 2010 evaluation, where preliminary results show it worked about as well as their other Marsyas-based entries. We consider this a good first showing, since the sparse coding was not at all tuned for music tasks. At Google, we have added both MFCC and the SAI sparse code features to a large collection of content-based audio and video features that various classifiers can draw on. On a number of tasks, such as language classification, the SAI sparse-code features have proved to be more useful than the MFCC. The SAI features are more useful than the video features in several audio/video tasks. We hope to be able to turn some of these tasks into reportable research results.

6. OTHER METHODS

There are many ways besides vector quantization to convert the local dense patterns to sparse codes. For example, some of the methods developed for locality-sensitive hashing, such as SPEC-hashing, may be good alternatives [9]. These are also much faster than vector quantization, which is one reason we are looking into them. Techniques like matching pursuit, on the other hand, tend to be slower, though they may produce better results in applications where reconstruction error is a relevant metric. Methods such as deep belief networks also look like promising ways to extract stable local structural features.

7. CONCLUSION

The retrieval and ranking of mixed-sound (SNR distributed about 0 dB, but highly variable) audio documents from text queries using SAI features would put a “relevant” result in first place 39% more often with SAI features than with the best MFCC features that we tried. In the top 10, the MFCC would show an average of about 1.6 relevant hits, while the SAI would show about 2.2. For sounds without interference, the feature performance is not so disparate. These results suggest that the SAI features degrade much less in interference, compared to MFCC features.

This tolerance of other interfering sounds is exactly what we had hoped to achieve by making the sparse feature encoding more local and multi-scale (the boxes) in a space (the SAI) where sounds would show some separation based on either spectral or temporal pattern differences—as opposed to global vector quantization of a single feature vector such as a spectrum or cepstrum. The results suggest that this multi-scale feature extraction from auditory images is a good approach.

These results also suggest several more experiments to try: perhaps it would help to vector quantize local portions of the MFCC or the underlying log-spectral vector, to take advantage of sounds that separate out along the spectral dimension; and perhaps we can find better, possibly more localized, ways to cut up the SAI into sparse features that work better. We haven’t tested human performance on this mixture dataset; it’s hard to do a big ranking test with humans, but we could do a small re-ranking experiment on top-10 results, or pairwise judgements on top 2, and see if subjects can do much better at finding the top 1 from such small sets.

8. REFERENCES

[1] R. F. Lyon, M. Rehn, S. Bengio, T. C. Walters, and G. Chechik, “Sound retrieval and ranking using sparse auditory representations,” *Neural computation*, vol. 22, no. 9, pp. 2390–2416, 2010.

[2] E. C. Smith and M. S. Lewicki, “Efficient auditory coding,” *Nature*, vol. 439, no. 7079, pp. 978–982, 2006.

[3] P-A. Manzagol, T. Bertin-Mahieux, and D. Eck, “On the use of sparse time-relative auditory codes for music,” in *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, 2008.

[4] P. Smaragdis and J. C. Brown, “Non-negative matrix factorization for polyphonic music transcription,” in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2003, pp. 177–180.

[5] M. D. Plumbley, S. A. Abdallah, T. Blumensath, and M. E. Davies, “Sparse representations of polyphonic music,” *Signal Processing*, vol. 86, no. 3, pp. 417–431, 2006.

[6] D. Grangier and S. Bengio, “A neural network to retrieve images from text queries,” in *Artificial Neural Networks – ICANN 2006*. 2006, pp. 24–34, Springer.

[7] T. M. Cover, “Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition,” *IEEE transactions on electronic computers*, pp. 326–334, 1965.

[8] G. Tzanetakis, “Marsyas-0.2: a case study in implementing music information retrieval systems,” in *Intelligent Music Information Systems: Tools and Methodologies*, J. Shen et al., Ed., pp. 31–49. Information Science Reference, 2008.

[9] R. S. Lin, D. A. Ross, and J. Yagnik, “SPEC hashing: Similarity preserving algorithm for entropy-based coding,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 848–854.