ON THE USE OF SPARSE TIME-RELATIVE AUDITORY CODES FOR MUSIC

Pierre-Antoine Manzagol Thierry Bertin-Mahieux Douglas Eck
Université de Montréal
Department of Computer Science
Montreal, Canada
{manzagop,bertinmt,eckdoug}@iro.umontreal.ca

ABSTRACT

Many if not most audio features used in MIR research are inspired by work done in speech recognition and are variations on the spectrogram. Recently, much attention has been given to new representations of audio that are sparse and time-relative. These representations are efficient and able to avoid the time-frequency trade-off of a spectrogram. Yet little work with music streams has been conducted and these features remain mostly unused in the MIR community. In this paper we further explore the use of these features for musical signals. In particular, we investigate their use on realistic music examples (i.e. released commercial music) and their use as input features for supervised learning. Furthermore, we identify three specific issues related to these features which will need to be further addressed in order to obtain the full benefit for MIR applications.

1 INTRODUCTION

The majority of the features used in audio-related MIR research are based on Fourier analysis, which suffers from two weaknesses. The first is the trade-off in precision between time and frequency. The second, common to all block based representations, is a sensitivity to arbitrary alignment of the blocks with the musical events.

Sparse coding assumes a signal can be represented at a given point in time by a rather small number of basis functions taken from an overcomplete dictionary [9]. Recent work [2, 10, 13, 14] applies these ideas to audio streams. When set in the time domain, the result is a spikegram, an efficient representation of the signal that avoids both of the spectrogram's weaknesses. Figure 1 shows a given signal (an ascending and descending C-major scale played on a piano), its spectrogram and its spikegram. As can be seen, a spikegram is composed of a set of spikes, corresponding to the placement of a basis function (kernel) at a precise point in time and with a particular scaling coefficient. The spikegram encodes audio very efficiently and with arbitrary resolution along both axes.

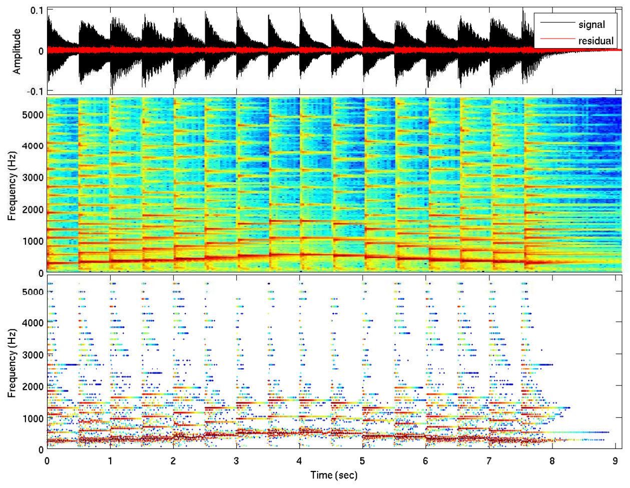

Figure 1. **Top:** original signal (an ascending and descending C-major scale played on the piano) and its residual (after encoding the signal to 20 dB SNR using the auditory codes). **Middle:** spectrogram. **Bottom:** spikegram. The horizontal axis represents time with the same resolution as that of the signal. The vertical axis corresponds to the frequencies of the n kernels used (Gammatones in this case). A dot represents a spike, a kernel placed at a specific point in time. The size and color of the dot are representative of the scaling coefficient of the kernel. This spikegram contains $13,000$ values, enough to encode the signal to 20dB SNR.

The organization of the paper is as follows. Section 2 reviews existing work on sparse coding for audio streams. Section 3 presents the sparse coding algorithm we use. Section 4 attempts to bring insight into the resulting codes. To this end, we encode the Tzanetakis genre dataset using predefined Gammatone kernels. In Section 5 the features are applied in a naive way to the task of genre recognition and are shown to work as well as other commonly used audio features. In section 6 we attempt to learn a set of kernels better suited to music than the general purpose Gammatones. The learned kernels are qualitatively different from those learned on other types of sounds. This suggests music poses specific coding challenges. Finally, in Section 7, we conclude and identify three specific issues with sparse coding for further research.

2 RELATED WORK

In this section we present existing work on creating a sparse encoding of audio signals. First, we present work done in the frequency domain, and secondly, methods in the time domain.

2.1 Frequency Domain

Plumbley and Abdallah have many publications about sparse coding using a code book. See [10] for one of their recent articles. The main idea is to derive a dictionary of power spectra from a corpus of audio. One assumes that the spectra of the signal is a weighted sum of the dictionary elements. Weights can be derived in number of ways. For example in [10] a method employing non-negative matrix factorization and one employing variance estimation are used. In the same article, the authors apply their coding to the task of note detection. However, the most sparse representation they present is from a time domain method, discussed further in Section 2.2.

Another recent approach is introduced by Gardner and Magnesco [5]. The idea is to compute a transformation of the spectrogram that gives it higher precision. Thus it is possible to track a single signal efficiently.

2.2 Time Domain

The most used method in the time domain is based on a dictionary of kernels that are mapped onto a signal using a MAP estimate [2, 10, 13]. MAP gives the best reconstruction error, but is very computationally expensive. Thus, an approximation is developed in [2], and in [13] MAP is used as an optimization phase using only a subset of selected kernels.

Two other methods of encoding are described by Smith and Lewicki [13]. The first one uses a threshold on the correlation between the signal and the kernels; the second one uses matching pursuit. In experiments, matching pursuit comes out as the best compromise between computational complexity and Signal to Noise Ratio (SNR).

The kernels used can be either predefined or learned using a gradient ascent method. The first option is used by Smith and Lewicki in [13]. They use Gammatones, which are known to approximate the cochlear filters. The second option requires a corpus on which to learn the kernels. In [10], 57 kernels are learned on a recording of Beethoven's Bagatelle containing 57 played notes. Thus they expect each kernel to model a particular note. In [14], learning is done on environmental sounds (ambient and transient), mammalian vocalizations, speech and reversed speech.

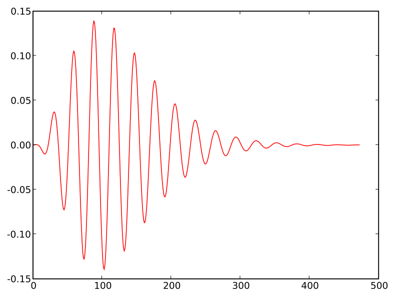

Figure 2. Example of a Gammatone kernel.

3 SPARSE TIME-RELATIVE CODING

In this section we describe the model we use for signal representation as well as the encoding algorithm. The model is the same as Smith and Lewicki's [13] and is used in conjunction with the matching pursuit encoding, as done in [13, 14].

3.1 Model for signal representation

We represent a signal using a sparse representation based on shiftable kernels [13]. This models the signal x as a linear superposition of kernels ϕ that are placed at precise time locations τ with a given scaling coefficient s. Formally:

$$x(t) = \sum_{m=1}^{M} \sum_{i=1}^{n_m} s_i^m \phi_m(t - \tau_i^m) + \epsilon(t),$$

where m runs over the different kernels, i over the different instances of a given kernel and ϵ represents additive noise. The length of the kernels is variable.

3.2 Choice of encoding algorithm

Encoding a signal amounts to determining the placement and scaling of kernels. This is a non-linear process and finding the optimal sparse representation using a generic dictionary of functions is NP-hard [3]. We choose to use matching pursuit [7], an iterative greedy algorithm, which is shown to offer a very good trade-off between computational resources and efficiency of representation [13]. The algorithm works iteratively in three steps. First, the signal is cross-correlated with the kernels. The best fitting projection is then selected and the corresponding kernel identity, placement and scaling are recorded. The projection is then subtracted from the signal and the procedure is repeated over the residual.

4 ENCODING USING GAMMATONE KERNELS

In this section, we attempt to bring insight into the encoding. Basic characteristics and properties are illustrated through

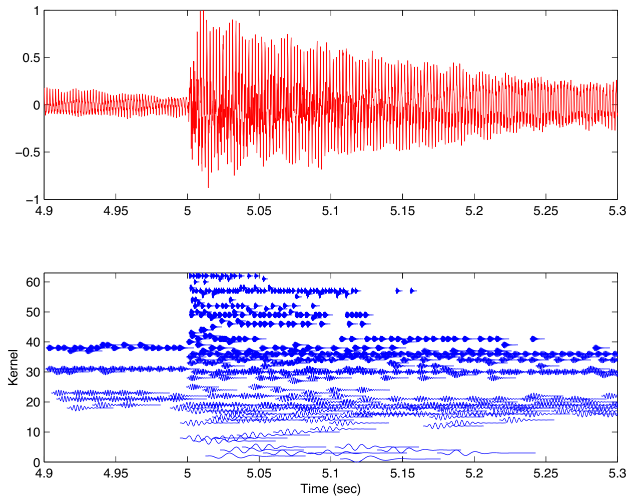

Figure 3. **Top:** original signal (close-up of a piano sounding an A4). **Bottom:** spikegram (using kernel shapes).

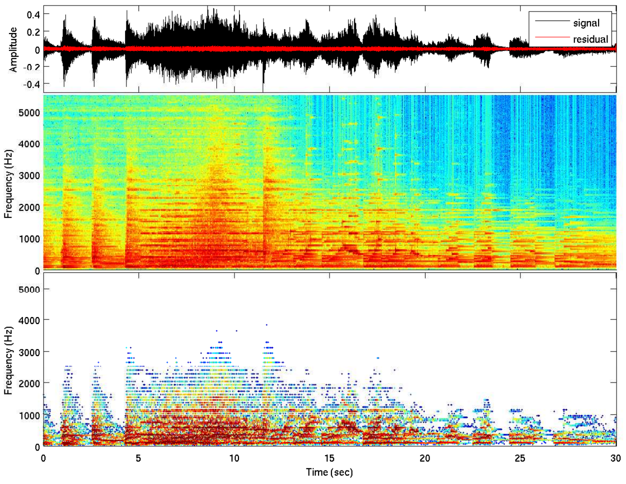

Figure 4. **Top:** original signal (30 seconds from a jazz song) and its residual (after encoding the signal to 20dB SNR using the auditory codes). **Middle:** spectrogram. **Bottom:** spikegram.

# of kernels	10^3	10^4	2.5×10^4	5×10^4
32	2.82 (0.97)	8.57 (2.17)	12.69 (2.87)	16.60 (2.88)
64	3.02 (1.03)	9.07 (2.29)	13.44 (2.98)	17.43 (2,70)
128	3.08 (1.04)	9.24 (2.31)	13.76 (2.99)	17.76 (2.52)

Table 1. Mean (standard deviation) of the SNR (dB) obtained over 100 songs (ten from each genre), as a function of the number of Gammatones in the codebook and of the maximum number of spikes allowed to reach a maximal value of 20 dB SNR.

the encoding of the Tzanetakis genre dataset. We use Gammatone kernels (see Figure 2), which are motivated as biological general purpose kernels used in the cochlea. Their characteristic sharp rise and slow decay also intuitively fit many real phenomenons. The Gammatones are set according to an equivalent rectangular band (ERB) filter bank cochlear model using Slaney's auditory toolbox for Matlab [12], as done in [13]. Unless otherwise stated, we use 64 variable length normalized Gammatone kernels, with frequencies ranging from 20Hz to the Nyquist frequency.

An important characteristic of the coding is that kernels are placed at precise time locations. This characteristic allows for precise localisation of events. This is illustrated in Figure 3, which shows the encoding of a note played on a piano. The onset clearly stands out.

Another important property of the coding is the fact that spikes are only used on a per-need basis. In Figure 4 we see that most of the spikes are placed in the middle of the song where the energy is. This contrasts with the uniform distribution of encoding resources in the spectrogram and illustrates a characteristic of sparse coding: it is adaptive in the number of spikes for a given encoding ratio. Figure 5 shows the encoding of a metal song. Though the code is still sparse, more spikes are needed than for the jazz example, perhaps due to the constant presence of high-frequency guitar with heavy distortion.

We further investigate the previous result, i.e. that jazz seems easier to encode than metal. For this purpose we used Tzanetakis' genre dataset later presented in Section 5. In top of Figure 6 we show the average number of spikes per song needed to encode songs from different musical genres to a given SNR. Some genres seem to require more spikes, metal being the most needy. We placed an upper limit on the number of spikes computed (10^5), which is why the metal box is compressed. To make sure the difficulty lies in the high frequencies, we divide the 64 kernels in eight bins and count them separately. The results are at the bottom of Figure 6. The highest frequencies are on the leftt, the lowest on the right. As expected, many high frequency Gammatones were used for metal songs, whereas classical or jazz songs required few.

It is also interesting to determine the effect on the signal to noise ratio of using different numbers of kernels in the ERB and different numbers of spikes in the spikegram. The results are presented in Table 1. Surprisingly, the number of kernels has relatively little impact for the tested range of values.

5 GENRE RECOGNITION

In order to explore the discriminative ability of the spikegram representation, we apply it to the classification task of genre recognition. As classifier, we use the AdaBoost meta-learning algorithm. This choice is motivated by the fact that source code [8] is available as well as published results for a set of genre recognition experiments [1]. Those experiments

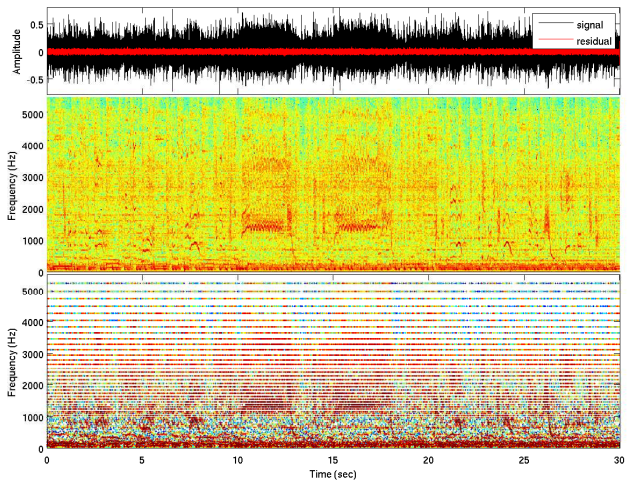

Figure 5. **Top:** original signal (30 seconds from a metal song) and its residual (after encoding the signal to 20dB SNR using the auditory codes). **Middle:** spectrogram. **Bottom:** spikegram.

were performed on a freely available dataset from Tzanetakis. (Similar datasets were used for previous MIREX contests. However those datasets are not available for download.)

The Tzanetakis genre dataset contains one thousand 30-second songs evenly distributed among 10 genres and is used in several papers [1, 16, 15, 6]. The dataset is very small and is annotated with only a single winner-take-all genre label. Though we believe the dataset is suitable for exploring the predictive power of the spikegram, it is clearly too small to reflect the challenges faced in finding structure in large datasets of audio.

5.1 Algorithm

AdaBoost [4] is a *meta-learning* method that constructs a *strong classifier* from a set of simpler classifiers, called *weak learners* in an iterative way. Originally intended for binary classification, there exist several ways to extend it to multiclass classification. We use AdaBoost.MH [11] which treats multiclass classification as a set of one-versus-all binary classification problems. In each iteration t, the algorithm selects the best classifier, called $h^{(t)}$ from a pool of *weak learners*, based on its performance on the training set, and assigns it a coefficient $\alpha^{(t)}$. The input of the *weak learner* is a d-dimensional observation vector $x \in \Re^d$ containing audio features for one segment of data (5 seconds in our experiments). The output of $h^{(t)}$ is a binary vector $y \in \{-1, 1\}^k$ over the k classes. $h_l^{(t)} = 1$ means a vote for class l by a *weak learner* while $h^{(t)}$, -1 is a vote against. After T iterations, the algorithm output is a vector-valued

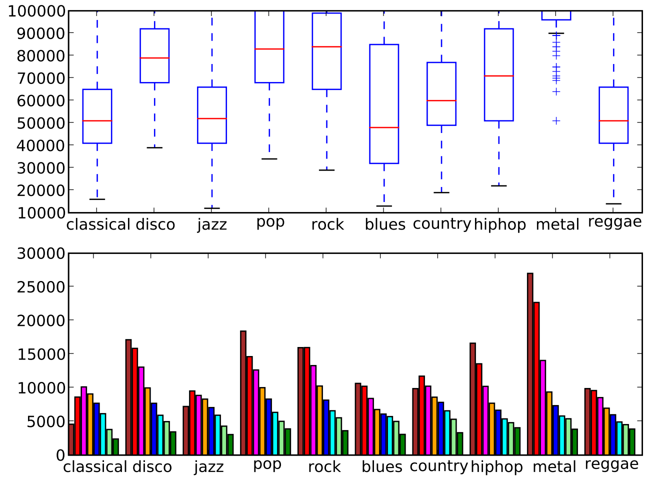

Figure 6. **Top:** box-and-whisker plot of the number of spikes (out of a maximum of 10^5) required to encode songs from various genres up to a SNR of 20dB. The red line represents the median and the box extends from the lower quartile to the upper one. **Bottom:** Bar plot of the mean number of kernels used per song depending on the genre. The 64 kernels are split into eight bins going from the highest frequency Gammatones (left) to the lowest ones (right).

discriminant function:

$$g(x) = \sum_{t=1}^{T} \alpha^{(t)} h^{(y)}(x) \qquad (1)$$

We obtain a single label by taking the class with the "most votes" i.e $f(x) = \arg\max_l g_l(x)$. As *weak learner* we use *single stumps*, a simple threshold on one element of the feature vector.

We use an AdaBoost method for genre recognition based on a method detailed in Bergstra et al. [1].

5.2 Features from the spikegram

The input of the classifier needs to be a feature vector of fixed size for every segment of a song (5 seconds segments here). Of course, a sparse code is designed to adapt to specific audio signals. The number of spikes per segment changes, and the spikes can be positioned at any given music frame. Thus we have encode the information about the spikes differently in order to use it as input to a typical classifier.

Here we try two types of features based on the spikes. First, we simply count the number of times each spike is used in every segment, and we also sum their scaling coefficients s. We also give the same information, but normalized by the number of spikes in the segment. Finally, we give the classifier the total number of spikes in this segment. For 64 kernels, this yields an input vector of size 257 per segment.

feat. SC	feat. SC + MFCC	MFCC (B.)
63.0%	68.2%	63%

Table 2. Genre recognition results on Tzanetakis dataset with 5-fold cross-validation using the sparse code features (SC), the sparse code features and our implementation of MFCC (SC + MFCC) and result using MFCC from Bergstra et al. [1] (B.). Result from [1] is approximate (taken from a graph).

5.3 Results

Results are reported using a 5-fold cross-validation. For more details about reported results on this dataset in the litterature, see Bergstra et al. [1].

Using AdaBoost.MH with single stumps as weak learners, and 5-second segments, Bergstra et al. [1] report an error per song of about 63% using MFCCs. Our results using the features derived from the spikegrams are presented in Table 2. We see that even with a naive use of the spikegram, results are comparable with commonly used audio features. The spikegrams also add information to the MFCC as can be seen in Column 2 of Table 2.

6 LEARNING KERNELS

The spikegrams used in the previous sections were based on the use of kernels of a specific predefined form (Gammatones). It is also possible to learn the kernels as done in [10, 14] over different audio corpora. Here we investigate the learning of kernels for western commercial music. There are at least three motivations for this. First, we hope to better encode music with learned kernels than with Gammatones. Secondly, these kernels should be more efficient as input for MIR tasks. Thirdly, perhaps these learned kernels can form meaningful blocks representing concepts like compression, note onsets or timber. This was the case in Plumbley et al. [10] where the authors could relate the learned kernels to piano notes. The difference here is that we train on more complex music and we put less constraint on the kernel lengths.

6.1 Algorithm

The learning is done by gradient ascent. The approximate gradient of a signal x, given a kernel ϕ, is (from [14]):

$$\frac{\partial}{\partial \phi} p(x|\phi) = \frac{1}{\sigma_\epsilon} \sum_i \hat{s}_i [x - \hat{x}]_{\tau_i} \tag{2}$$

We can then perform gradient ascent on every dimension of the kernels using the residual signal at corresponding positions. The σ_ϵ is constant and can be ignored as we set the learning rate empirically.

In our experiments, 32 kernels are initialized using random values drawn from a Gaussian distribution. Throughout the learning, the kernels are kept normalized. We use a database containing tens of thousands of MP3s. Learning proceeds by iteratively randomly choosing a song from the database, encoding it using 1000 spikes, computing the residual signal and performing gradient ascent on the kernels. The learning rate was set to 0.01 and we did approximately 2000 iterations. Figure 7 shows samples from the 32 kernels we learned. The learned kernels are qualitatively different from Gammatones. They closely resemble those learned by [10]. However, our kernels are of variable length which may be better suited to encoding music with varying note durations.

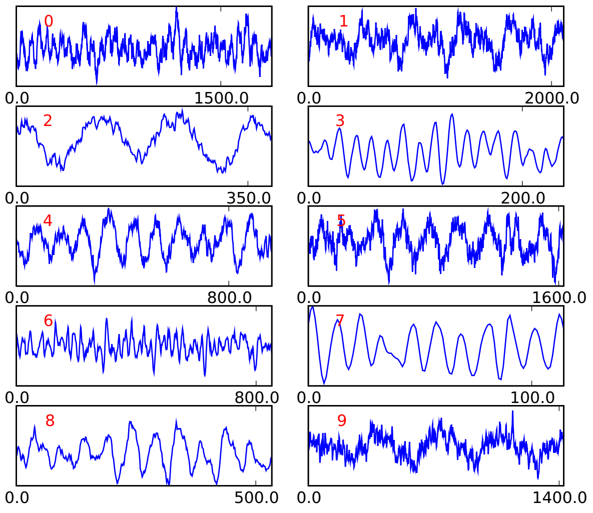

Figure 7. Examples of learned kernels

kernel	10^3	10^4	2.5×10^4	5×10^4
gammatones	2.82 (0.97)	8.57 (2.17)	12.69 (2.87)	16.60 (2.88)
learned	1.86 (0.57)	6.06 (1.49)	8.86 (2.00)	11.53 (2,37)

Table 3. Mean (standard deviation) of the SNR (dB) obtained over 100 songs (ten from each genre) with either 32 Gammatones or 32 learned kernels in the codebook.

6.2 Learned Kernels versus Gammatones

As a first experiment, we evaluate how well the learned kernels encode songs compared to the Gammatones. Results are shown in Table 3. Unfortunately, Gammatones perform significantly better at this task. This is a surprising result, as the learned kernels are trained to optimize this specific cost, suggesting that learning kernels for complex music poses specific challenges.

As a second experiment, we use the 32 learned kernels as input for genre recognition, and we compare the results with those obtained using 32 Gammatones. Results are shown in Table 4. We compare the results when encoding using 10^3

gamma (1K)	learn (1K)	gamma(10K)	learn (10K)
47.5 %	46.3 %	52.5 %	54.3 %

Table 4. Genre recognition results on Tzanetakis dataset with 5-fold cross-validation. Number in parentheses is the maximum number of spikes used to encode each song.

and 10^4 spikes for computational reasons. Learned kernels seem to perform similarly as Gammatones. This shows that the encoding capacity of kernels is different from their predictive property for a particular task.

7 DISCUSSION AND FUTURE WORK

Our goal with this work is to draw further attention to time-relative sparse codings. These codes manage to avoid the weaknesses of the spectrogram and encode an audio signal very efficiently. Yet there are three issues related to their use that will need to be addressed. The first is how to properly use them as input to MIR tasks. Indeed, in our genre recognition experiments, we summarize spike counts and scaling coefficients over a block. This destroys the useful time-relative configurations of the spikes and reintroduces a sensitivity to arbitrary block alignment. The fact that we are still able to get results comparable to MFCCs shows the potential of the features. The second issue relates to learning in the case of complex music. Our work seems to reveal specific difficulties. In particular, kernels trained over music tended to continually grow, and thus we had to enforce some length constraints. The disappointing results are also indicative that the learning problem becomes difficult with complex music. Solutions might lie in using weight decays, and starting from kernels trained on less complex signals. Finally, there is an issue with the computational complexity of the technique. Even with the use of matching pursuit, the computational requirements of encoding are above real time, which may not suit all applications. Answers may rely in performing approximate matching pursuit, or learning a function to do an approximate encoding using machine learning.

8 ACKNOWLEDGEMENTS

We would like to thank James Bergstra for helpful discussions. Pierre-Antoine Manzagol is supported by an FQRNT scholarship. Thierry-Bertin Mahieux and Douglas Eck are supported by an NSERC discovery grant.

9 REFERENCES

[1] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kégl. Aggregate features and AdaBoost for music classification. *Machine Learning*, 65(2-3):473–484, 2006.

[2] T. Blumensath and M. Davies. Sparse and shift-invariant representations of music. *IEEE Transactions on Speech and Audio Processing*, 2006.

[3] G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive approximation*, 13(1):57–98, 2004.

[4] Y. Freund and R.E. Shapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[5] T. J. Gardner and M. O. Magnasco. Sparse time-frequency representations. *Proceedings of the National Academy of Science*, 103:6094–6099, April 2006.

[6] Tao Li and George Tzanetakis. Factors in automatic musical genre classification. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2003.

[7] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):3397–3415, December 1993.

[8] Multiboost, a pure C++ implementation of AdaBoost.MH by N. Casagrande available at http://www.iro.umontreal.ca/˜casagran/.

[9] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, June 1996.

[10] M. Plumbley, S. Abdallah, T. Blumensath, and M. Davies. Sparse representations of polyphonic music. *Signal Processing*, 86(3):417–431, March 2006.

[11] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[12] M. Slaney. Auditory toolbox, 1998. (Tech. Rep. No. 1998-010). Palo Alto, CA:Interval Research Corporation.

[13] E. Smith and M. S. Lewicki. Efficient coding of time-relative structure using spikes. *Neural Computation*, 17(1):19–45, 2005.

[14] E. Smith and M. S. Lewicki. Efficient auditory coding. *Nature*, 439:978–982, February 2006.

[15] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, Jul 2002.

[16] George Tzanetakis, Andrey Ermolinskyi, and Perry Cook. Pitch histograms in audio and symbolic music information retrieval. In Michael Fingerhut, editor, *Proceedings of the Third International Conference on Music Information Retrieval: ISMIR 2002*, pages 31–38, Oct 2002.