CHORD DETECTION USING DEEP LEARNING

Xinquan Zhou
Center for Music Technology
Georgia Institute of Technology
royzxq@gmail.com

Alexander Lerch
Center for Music Technology
Georgia Institute of Technology
alexander.lerch@gatech.edu

ABSTRACT

In this paper, we utilize deep learning to learn high-level features for audio chord detection. The learned features, obtained by a deep network in bottleneck architecture, give promising results and outperform state-of-the-art systems. We present and evaluate the results for various methods and configurations, including input pre-processing, a bottleneck architecture, and SVMs vs. HMMs for chord classification.

1. INTRODUCTION

The goal of automatic chord detection is the automatic recognition of the chord progression in a music recording. It is an important task in the analysis of western music and music transcription in general, and it can contribute to applications such as key detection, structural segmentation, music similarity measures, and other semantic analysis tasks. Despite early successes in chord detection by using pitch chroma features [5] and Hidden Markov Models (HMMs) [14], recent attempts at further increasing the detection accuracy are only met with moderate success [3, 25].

In recent years, deep learning approaches have gained significant interest in the machine learning community as a way of building hierarchical representations from large amounts of data. Deep learning has been applied successfully in many fields. For instance, a system for speech recognition with deep learning was able to outperform state-of-the-art systems (not using deep learning) [9]. Several studies indicate that deep learning methods can be very successful when applied to Music Information Retrieval (MIR) tasks, especially when used for feature learning [1, 8, 12, 16]. Deep learning, with its potential to untangle complicated patterns in a large amount of data, should be well suited for the task of chord detection.

In this work, we investigate Deep Networks (DNs) for learning high-level and more representative features in the context of chord detection, effectively replacing the widely used pitch chroma intermediate representation. We present individual results for different pre-processing options such as time splicing and filtering (see Sect. 3.2), architectures (see Sect. 3.4), and output classifiers (see Sect. 4).

2. RELATED WORK

During the past decade, deep learning has been considered by the machine learning community to be one of the most interesting and intriguing research topics. Deep architectures promise to remove the necessity of custom-designed and manually selected features as neural networks should be more powerful in disentangling interacting factors and thus be able to create meaningful high-level representations of the input data. Generally speaking, deep learning combines deep neural networks with an unsupervised learning model. Two major learning models are widely used for unsupervised learning: Restricted Boltzmann Machines (RBMs) and Sparse Auto Encoders. A deep architecture comprises multiple stacked layers based on one of these two models. These layers can be trained one by one, a process that is referred to as "pre-training" the network. In this work, we employ RBMs to pre-train the deep architecture in an unsupervised fashion; this is called a Deep Belief Network (DBN). DBNs, composed of a stack of RBMs, essentially share the same topology with general neural networks: DBNs are generative probabilistic models with one visible layer and several hidden layers [10].

Since Hinton proposed a fast learning algorithm for DBNs [10], it has been widely used for initializing deep neural networks. In deep structures, each layer learns relationships between units in lower layers. With the number of RBM layers increasing the complexity of the system increases, making the structure, theoretically, more powerful. An extra softmax output layer can be added to the top of the network (see Eqn (6)); its output can be interpreted as the likelihood of each class.

LeCun formulated the idea of applying the Convolutional Neural Networks (CNNs) to images as well as speech and other time-series signals [15]. This approach, allows to deal with the variability in time and space to a certain degree; CNNs can be seen as a special kind of neural network in which the weights are shared across the input within a certain spatial or temporal area. Therefore, it reduces the overfitting problem due to the weight sharing. The weights thus act as a kernel filter applied to the input. CNNs have been particularly successful in image analysis. For example, Norouzi et al. used Convolutional RBMs to learn shift-invariant features [21].

Modifications of the network architecture result in different results. Grezl et al. used a so-called bottleneck architecture neural network to obtain features for speech recognition and showed that these features improve the accuracy of the

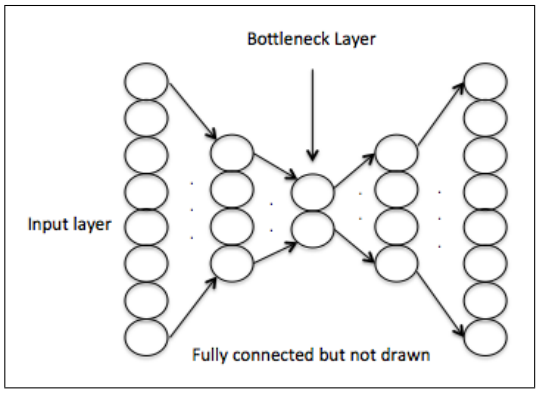

Figure 1. Bottleneck Neural Network Architecture

task [7]. The principle behind the bottleneck-shaped architecture is that the number of neurons in the middle layer is lower than in the other layers as shown in Fig. 1. A network with bottleneck can be structured in two halfs. The first layer to the bottleneck layer, with a gradual decrease of the number of neurons per layer, functions as an encoding or compression process which compacts relevant information and discards redundant information. The second half from the bottleneck layer to the last layer has a gradual increase in the number of neurons per layer. The function of this part can be interpreted as a decoding process. An additional benefit of bottleneck architectures is that they can reduce overfitting by decreasing the system complexity.

Recently, more researchers investigated deep learning in the context of MIR. Lee et al. pioneered the application of convolutional deep learning for audio feature learning [16]. Hamel et al. used the features learned from music with a DBN for both music genre classification and music auto-tagging [8]; their system was successful in MIREX 2011 with top-ranked results. Battenberg employed a conditional DBN to analyze drum patterns [1]. The use of deep architectures for chord detections, however, has not yet been explored, although modern neural networks have been employed in this field. For instance, Boulanger et al. investigated recurrent neural networks [2] and Humphrey has explored CNNs [11, 13]. While they also used the concept of pre-training, their architectures have only two or 3 layers and thus cannot be called "deep".

The basic buildings blocks of most modern approaches to chord detection can be traced back to two influential publications: Kawakami proposed to use HMMs for representing chords as hidden states and to model the transition probability of chords [14] and Fujishima introduced pitch chroma vectors extracted from the audio as input feature for chord detection [5]. Since then there have been a lot of studies using chroma features and HMMs for chord detection [4, 22]. Examples for recent systems are Ni, using genre-independent chord estimation method based on HMM and chroma features [20] and Cho used multi-band features and a multi-stream HMM for chord recognition [3]. Training HMMs with pitch chroma features arguably is the standard approach for this task and the progress is less marked by major innovations but by optimizing and tuning

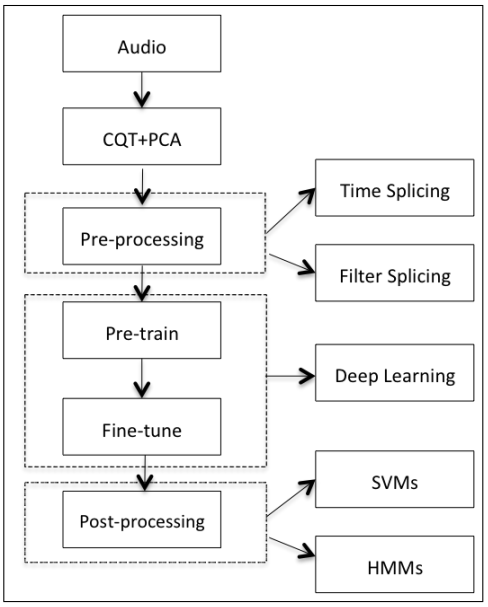

Figure 2. The overview of our system

specific components.

3. SYSTEM OVERVIEW

Figure 2 gives an overview of all components and processing steps of the presented system. The following section will discuss all of these steps in detail.

3.1 Input Representation

The input audio are converted to a sample rate of 11.025 kHz. Then, a Constant Q transform (CQT) is applied. The CQT is a perceptually inspired time-frequency transformation for audio. The resulting frequency bins are equally spaced on a logarithmic ("pitch") scale. It has the advantage of providing a more musically and perceptually meaningful spectral representation than the DFT. We used an implementation of the CQT as a filterbank of Gabor filters, spaced at 36 bins per octave, i.e. 3 bins per semitone, yielding 180 bins representing a frequency range spanning from 110 Hz to 3.520 kHz. Finally, we use Principal Component Analysis (PCA) for decorrelation, and apply Z-Score normalization [24].

3.2 Pre-processing

Neighboring frames of the input representation can be expected to contain similar content, as chords will not change on a frame-by-frame basis. In order to take into account the relationship between the current frame and previous and future frames, we investigate the application of several pre-processing approaches.

3.2.1 Time Splicing

Time splicing is a simple way extend the current frame with the data of neighboring frames by concatenating the frames into one larger superframe. For example, assuming there are 7 input frames. In first order time splicing, we

concatenate the current frame, the previous frame, and the following frame. Thus, the 4th superframe is comprised of the 3rd, 4th, and 5th frame. Since the same operation will be applied for the 5th superframe etc., there will be overlap introduced between neighboring superframes.

3.2.2 Convolution

CNNs are extensively used in the tasks that want to take account the correlation between the inputs such as hand digits recognition. As stated before, it also works in the context of time series. For the reason that the deep learning toolbox does not support the convolution operation in time domain, we give up on the real CNNs, but we employ a pre-processing step inspired by CNNs. Essentially, CNNs have one or more convolutional layers between the input and lower layers of the neural network. The function of a convolutional layer can be interpreted as the application of a linear filter plus a non-linear transformation, sometimes also combined with pooling operation:

$$Y = \text{pool}(\text{sigm}(K * X + B)), \tag{1}$$

in which Y is the output of a convolutional layer, K is the linear kernel filter (impulse response), X is the input, B is the bias, $\text{sigm}()$ is a non-linear transform, and $\text{pool}()$ is a down-sampling operation. The uniqueness of convolutional networks stems from the convolution operation applied to the input X. We emulate this functionality by applying filters to the input signal. Instead of learning the filters, we manually design several filters.

A simple choice is low pass filter family. We employ a simple single pole low pass filter due to its easy parameter configuration. Eqn (2) shows the difference equation for an input x.

$$y_n = (1 - \alpha)y_{n-1} + \alpha x_n \tag{2}$$

Given that our system is not required to work in real-time, we can apply anti-causal filtering and filter the signal in both directions so that the resulting overall filter has a zero phase response.

We also apply two other low pass filters, both of which have exponential decay shaped impulse response. The difference equations are given in Eqn (3) and Eqn (4).

$$y_1(n) = \sum_{k=1}^{N} a^{-k+1} x(n - N + k) \tag{3}$$

$$y_2(n) = \sum_{k=1}^{N} a^{-k+1} x(n + N - k) \tag{4}$$

The filter length is N and a is the exponential base. These two filters are not centered around the current frame anymore, but shifted by N frames. Their impulse responses are symmetric to each other. One could interpret these filters as to focus on past frames and future frames, respectively. These filters will be referred to as "extension filters".

The ideas of splicing and convolution can be combined, as exemplified by Fig. 3. We refer to this kind of processing as "spliced filters".

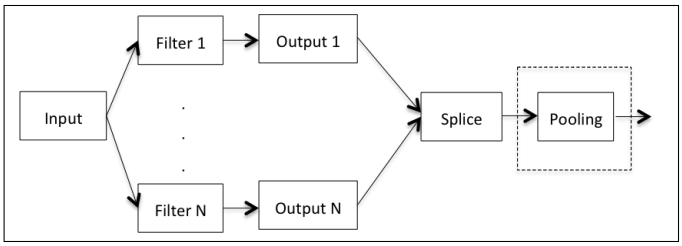

Figure 3. Splicing output of different filters

Furthermore, similar to the process in CNNs, a maximum pooling operation on the output of the spliced filters is optionally applied. The operation takes the maximum value among different filters per "bin".

3.3 Training

It is impractical to train DNNs directly with back propagation using gradient decent due to their deep structure and the limited amount of training samples. Therefore, the network is usually initialized by an unsupervised pre-training step. As our network consists of RBMs, Gibbs sampling can be used for training. The objective is to retain as much information as possible between input and output.

The computation between layers can be represented as:

$$Y_l = \text{sigm}(W_l X_l + B_l), \tag{5}$$

which is identical to many traditional neural networks. Therefore, a standard back propagation can be applied after pre-training to fine-tune the network in a supervised manner. The loss criterion we use in this work is cross-entropy.

3.4 Architecture

We investigate a deep network with 6 layers in two different architectures. The common architecture features the same amount of neurons in every layer, in our case 1024. The bottleneck architecture has a lower number of neurons in the middle layers, namely 256 neurons in the middle layer, 512 in its neighboring layers, and 1024 neurons in the remaining layers [7]. A softmax output layer is stacked on top of both architectures as described by Eqn (6).

$$\text{softmax}(Y_l) = \frac{\exp(Y_l)}{\sum_{k=1}^{N} \exp(Y_k)} \tag{6}$$

Our network is implemented using the Kaldi package developed by John Hopkins University [23].

4. CLASSIFICATION

The output of the softmax layer can be interpreted as the likelihood of each class; taking the maximum will provide a class decision (this method will be referred to as *Argmax*). Alternatively, the output can be treated as intermediate feature vector that can be used as an input to other classifiers for computing the final decision.

4.1 Support Vector Machine

Support Vector Machines (SVMs) are, as widely used classifiers with generally good performance, worthy trying. The SVM is trained using the output of the network as features, and the classification is carried out for every frame. The classification is followed by a simple prediction smoothing.

4.2 Hidden Markov Model

HMMs are, as pointed out above, a standard classifier for automatic chord detection because the characteristics of the task fit the HMM approach well: Chords are hidden states that can be estimated from observations (feature vectors extracted from the audio signal), and the likelihood of chord transitions can be modeled with transition probabilities. Modified HMMs such as ergodic HMMs and key-independent HMMs have been also explored for this task [17, 22]. In this work we are mostly interested in the performance comparison between high-level features, so a simple first-order HMM is used. Given the probabilistic property of softmax output layer, we can use it directly as emission probabilities in the context of HMMs. Therefore, there is no need to train the HMM using, e.g., the commonly used Baum-Welch algorithm. Instead, the histogram of each class in our training is used as initial probabilities, and the bigram of chord transitions is used to compute the transition probabilities. Finally, we employ the Viterbi decoding algorithm to find the globally optimal chord sequence.

5. EVALUATION PROCEDURE

5.1 Dataset

Our dataset is a combination of several different datasets, yielding a 317-piece collection. The data is composed of

- 180 songs from the *Beatles dataset* [18],
- 100 songs from the *RWC Pop dataset* [6],
- 18 songs from the *Zweieck dataset* [18], and
- 19 songs from *Queen dataset* [18].

The pre-processing as described in Sect. 3.2 ensures identical input audio formats.

5.2 Methodology

Because training deep networks is time-consuming especially without GPU. Thus we have to give up cross validation evaluation due to the limited computing resources. The dataset is divided randomly into two parts: 80% for the training set and 20% for the test set. On the training scale, we use a frame-based strategy, which means we divide each song into frames, and treat each frame as an independent training sample. On average each song is divided into about 1200 frames resulting in approximately 300k training samples. As for testing scale, we calculate the frame-based accuracy as well, resulting in approximately 76k test samples.

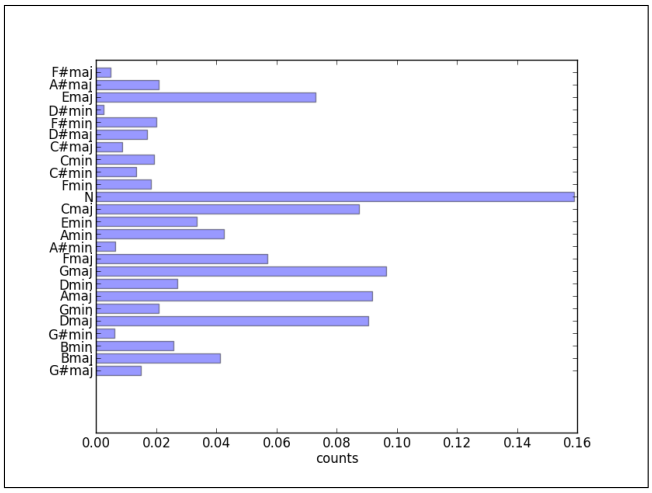

Figure 4. Chords histogram

Within the training set, 10% of the data is used as a validation set. For the post-processing, all data in the training set will be used to train the post-classifier.

The chosen ground truth for classification are major and minor triads for every root note, resulting in a dictionary of $24 + 1$ chord labels. Ground truth time-aligned chord symbols are mapped to this major/minor dictionary:

$$Chord_{majmin} \subset \{N\} \cup \{S \times maj, min\} \qquad (7)$$

with S representing the 12 pitch classes (root notes) and N being the label for unknown chords. In the calculation of the detection accuracy, the following chord types are mapped to the corresponding major/minor in the dictionary: triad major/minor and seventh major/minor. Other chord types are treated as unknown chords. For instance, G:maj and G:maj7 are mapped to 'G:maj'; G:dim and G:6 are all mapped to 'N'. The histogram of chords in our dataset after such mapping is shown in Fig. 4.

5.3 Evaluation Metric

The used evaluation metric is the same as proposed in the audio chord detection task for MIREX 2013: the Weighted Chord Symbol Recall (WCSR) is used to estimate how well the predicted chords match the ground truth. WCSR, formulated in Eqn (8), is defined as the total duration of segments with correct prediction:

$$WCSR = \frac{1}{N} \sum_{k=1}^{n} C_k, \qquad (8)$$

in which n is the number of test samples (songs), N is the total number of frames in all test samples, and C_k is the number of frames that are correctly detected in the kth sample.

6. EXPERIMENTS

6.1 Post-classifiers

We use pre-training to initialize the networks followed by fine-tuning using back propagation (represented as $DN_{DBN-DNN}$).

Training Scenario	Classifier	WCSR
$DN_{\mathrm{DBN-DNN}}$	Argmax()	0.648
$DN_{\mathrm{DBN-DNN}}$	SVM	0.645
$DN_{\mathrm{DBN-DNN}}$	HMM	**0.755**

Table 1. Chord detection performance using different post-classifiers

In this experiment, no pre-processing is applied to the data; the input is simply the input representation (CQT followed by PCA) as described in Sect. 3.1. The chosen architecture for this experiment is the bottleneck architecture. Three different classifiers are compared: the maximum of the softmax output (Argmax), an SVM, and an HMM.

The results listed in Table 1 are unambiguous and unsurprising: the HMM with Viterbi decoding outperforms the SVM; using HMMs with a model for transition probabilities is an appropriate approach to chord detection as it models the dynamic properties of chord progressions, which cannot be done with non-dynamic classifiers such as SVMs. One noteworthy result is that the SVM does not improve the WCSR compared to the direct (Argmax) output of the network. Apparently, the SVM is not able to improve separability of the learned output features.

6.2 Pre-processing

As stated in Sect. 3.2, we investigate the application of different filters in the pre-processing stage. In a first experiment, a single pole filter (see Eqn (2)) is applied in an anti-causal way in order to assure zero phase. It has one parameter α for which the values 0.25, 0.5, and 0.75 are evaluated. In a second experiment, these filter outputs are spliced with the outputs of the extension filters as introduced in Sect. 3.2. These experiments are carried out with the $DN_{\mathrm{DBN-DNN}}$ training scenario, a bottleneck architecture, and an HMM classifier. Table 2 lists the results of with these different pre-processing variants. It can be observed that the network trained with filtered inputs slightly outperforms the network without pre-processing; splicing the filtered input with the extension filter outputs increases the results much.

α	Pre-processing	WCSR
0.25	Filtering	0.758
0.25	Spliced Filters	0.912
0.5	Filtering	0.787
0.5	Spliced Filters	0.857
0.75	Filtering	0.798
0.75	Spliced Filters	0.919

Table 2. Chord detection performance using different filter parameters

Architecture	Pre-processing	Training WCSR	WCSR
Common	None	0.843	0.703
Bottleneck	None	0.855	0.755
Common	Spliced Filters	**0.985**	0.876
Bottleneck	Spliced Filters	0.936	**0.919**
Common	Pooling	0.965	0.875
Bottleneck	Pooling	0.960	0.916

Table 3. Chord detection performance for different architectures and pre-processing steps

6.3 Architecture

6.3.1 Common vs. Bottleneck

The results of Grezl et al. indicate that a bottleneck architecture should be more suitable to learn high-level features than a common architecture and reduce overfitting [7]. In order to verify these characteristics for our task, the performance of both architectures is evaluated in comparison. The results are listed in Table 3 for the three pre-processing scenarios tested: no additional pre-processing, spliced filters and spliced filters followed by a max pooling. In order to allow conclusions about overfitting, the WCSR computed with the training set is reported as well. All results are computed for the $DN_{\mathrm{DBN-DNN}}$ training scenario with HMM classifiers.

The results show that the bottleneck architecture gives significantly better results ($p = 0.023$) on the test set (WSCR). Note that this is not true for the training set (Training WSCR), for which the common architecture achieves results in the same range or better than the bottleneck architecture. The difference between the results on the training set and the test set are thus much larger for the common architecture than for the bottleneck architecture. The bottleneck architecture is clearly advantageous to use: it reduces complexity and thus the training workload and increases the classification performance significantly. Furthermore, the comparison of classifier performance between training and test set in Table 3 clearly indicates that the common architecture tends to fit more to the training data rather, and is thus more prone to overfitting.

6.3.2 Single-Label vs. Multi-Label

As mentioned above, the pitch chroma is the standard feature representation for audio chord detection. Since we use the output of our deep network as feature, it seems an intuitive choice to learn pitch class information (and thus, a pitch chroma) instead of the chord classes. By doing so, the number of outputs is reduced by a factor of two, and there would also be a closer relation between the output and the input representation, the CQT. Therefore, the abstraction and complexity of the task might be decreased. It will, however, lead to another issue: the single-label output (one chord per output) will be changed into a multi-label output (multiple pitches per output). Therefore, the learning has to be modified to allow multiple simultaneous (pitch class)

Learning Targets	WCSR
Single-Label — 25 Chord Classes	**0.919**
Multi-Label — 12 Pitch Classes	0.78

Table 4. Chord detection performance for single-label vs. multi-label learning

labels. The experiment is carried out with both Splicing and Filtering in the pre-processing, the $DN_{\mathrm{DBN-DNN}}$ training scenario, and HMM classifiers. Table 4 lists the results.

Boulanger-Lewandowski et al. mention combining chroma features with chord labels for their recurrent neural network and report a slightly improved result [2]. They do not, however, provide a detailed description of this combination. As can be seen from the table, the performance for multi-label training is generally clearly lower than the result for single-label training. The results of the multi-label training of chroma vectors are disappointing; possible reasons for bad performance include (i) difficulties with multi-target learning, since it increases the difficulty to train; furthermore, our implementation of multi-label training might be sub-optimal as the same posterior is assigned to each target without any information on the pitch class energy, and (ii) the issue that not all pitches always sound simultaneously in a chord (or might be missing altogether) might have larger impact on the multi-label training than on the single-label training.

6.4 Results & Discussion

It it always challenging to compare previous results because of the varying evaluation methodologies, metrics, and datasets. It seems that the results of Cho and Bello [3], who reported a performance of about 76%, were computed with a comparable dataset. Referencing the recent MIREX results on Chord Detection, although the evaluation vocabulary is different, we can see a big improvement on the performance. In order to provide a baseline result to put results into perspective, we present the results of Chordino [19] with the default settings, computed on our dataset. It should be pointed out that this comparison is unfair as Chordino is able to detect as many as 120 chords, compared to our 24. The label mapping strategies are another significant issue for Chordino. Our label mapping results in nearly sixth of the total label are "N", which can mess the Chordino results up. The Chordino results are mapped to major/minor the same way as the ground truth annotations. The results are shown in Table 5. In the table, the best configuration is using Bottleneck architecture; spliced filters ($\alpha = 0.75$) as preprocessing; single label learning targets; Viterbi decoding as post classifier. The best configuration with max pooling is the same as best configuration except applying another max pooling layer after spliced filters. Although the performance is a little worse but the computation is much reduced. The presented results are clearly competitive with existing state-of-the-art systems.

Method	WCSR
Chordino	0.625
best configuration	0.919
best configuration with max pooling	0.916

Table 5. Comparison of the performance of the best configuration with Chordino

7. CONCLUSION & FUTURE WORK

In this work, we presented a system which applies deep learning to the MIR task of automatic chord detection. Our model is able to learn high-level probabilistic representations for chords across various configurations. We have shown that the use of a bottleneck architecture is advantageous as it reduces overfitting and increases classifier performance, and that the choice of appropriate input filtering and splicing can significantly increase classifier performance.

Learning a pitch class vector instead of chord likelihood by incorporating multi-label learning proved to be less successful. The idea has, however, a certain appeal and would allow the number of output nodes be independent of the number of chords to be detected. It is also conceivable to investigate a different option for the network output: instead of training chords or pitch classes we could — under the assumption that we are only after chords comprised of stacked third intervals — train the output with octave-independent third intervals in a multi-label scenario with 24 output nodes.

8. REFERENCES

[1] Eric Battenberg and David Wessel. Analyzing drum patterns using conditional deep belief networks. In *Proc. ISMIR*, pages 37–42. Porto, Portugal, 2012.

[2] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Audio chord recognition with recurrent neural networks. In *ISMIR*, pages 335–340, 2013.

[3] Taemin Cho and Juan P Bello. Mirex 2013: Large vocabulary chord recognition system using multi-band features and a multi-stream hmm. *Music Information Retrieval Evaluation eXchange (MIREX)*, 2013.

[4] Taemin Cho, Ron J Weiss, and Juan Pablo Bello. Exploring common variations in state of the art chord recognition systems. In *Proceedings of the Sound and Music Computing Conference (SMC)*, pages 1–8. Barcelona, Spain, 2010.

[5] Takuya Fujishima. Realtime chord recognition of musical sound: A system using common lisp music. In *Proc. ICMC*, volume 1999, pages 464–467, 1999.

[6] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. Rwc music database: Popular, classical and jazz music databases. In *ISMIR*, volume 2, pages 287–288, 2002.

[7] Frantisek Grezl, Martin Karafiát, Stanislav Kontár, and J Cernocky. Probabilistic and bottle-neck features for lvcsr of meetings. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–757. IEEE, 2007.

[8] Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In *ISMIR*, pages 339–344. Utrecht, The Netherlands, 2010.

[9] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[10] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[11] Eric J Humphrey and Juan Pablo Bello. Rethinking automatic chord recognition with convolutional neural networks. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 2, pages 357–362. IEEE, 2012.

[12] Eric J Humphrey, Juan Pablo Bello, and Yann LeCun. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In *Proc. ISMIR*, pages 403–408. Porto, Portugal, 2012.

[13] Eric J Humphrey, Taemin Cho, and Juan Pablo Bello. Learning a robust tonnetz-space transform for automatic chord recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 453–456. IEEE, 2012.

[14] Takashi Kawakami, Mitsuru Nakai, Hiroshi Shimodaira, and Shigeki Sagayama. Hidden markov model applied to automatic harmonization of given melodies. *Information Processing Society of Japan,(in Japanese), SIG Notes*, pages 59–66, 2000.

[15] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361:310, 1995.

[16] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, 2009.

[17] Kyogu Lee and Malcolm Slaney. Acoustic chord transcription and key extraction from audio using key-dependent hmms trained on synthesized audio. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):291–301, 2008.

[18] Matthias Mauch, Chris Cannam, Matthew Davies, Simon Dixon, Christopher Harte, Sefki Kolozali, Dan Tidhar, and Mark Sandler. Omras2 metadata project 2009. In *Proc. of 10th International Conference on Music Information Retrieval*, 2009.

[19] Matthias Mauch and Simon Dixon. Approximate note transcription for the improved identification of difficult chords. In *ISMIR*, pages 135–140, 2010.

[20] Yizhao Ni, Matt McVicar, Raul Santos-Rodriguez, and Tijl De Bie. Using hyper-genre training to explore genre information for automatic chord estimation. In *ISMIR*, pages 109–114, 2012.

[21] Mohammad Norouzi, Mani Ranjbar, and Greg Mori. Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2735–2742. IEEE, 2009.

[22] Hélene Papadopoulos and Geoffroy Peeters. Large-scale study of chord estimation algorithms based on chroma representation and hmm. In *Content-Based Multimedia Indexing, 2007. CBMI'07. International Workshop on*, pages 53–60. IEEE, 2007.

[23] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[24] J Sola and J Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *Nuclear Science, IEEE Transactions on*, 44(3):1464–1468, 1997.

[25] Yushi Ueda, Yuuki Uchiyama, Takuya Nishimoto, Nobutaka Ono, and Shigeki Sagayama. Hmm-based approach for automatic chord detection using refined acoustic features. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 5518–5521. IEEE, 2010.