Parallel Convolutional Neural Networks for Music Genre and Mood Classification

Thomas Lidy
Vienna University of Technology
Institute of Software Technology
lidy@ifs.tuwien.ac.at

Alexander Schindler
Austrian Institute of Technology
Digital Safety and Security Department
alexander.schindler@ait.ac.at

Abstract

Our approach to the MIREX 2016 Train/Test Classification Tasks for Genre, Mood and Composer detection is based on an approach combining Mel-spectrogram transformed audio and Convolutional Neural Networks (CNN). We utilize two different CNN architectures, a sequential one, and a parallel one, the latter aiming at capturing both temporal and timbral information in two different pipelines, which are merged on a later stage. In both cases, the crucial CNN parameters such as filter kernel sizes and pooling sizes were carefully chosen after a range of experiments.

1 Introduction

The use of Convolutional Neural Networks (CNN) have, after recent successes in image and text retrieval, also entered the audio domain. The power of Convolutional Neural Networks is in drastically reducing the weights that are needed to make the network learn, through the spatial weight sharing principle of the filter convolution approach.

A popular method in the audio domain is to use a spectrogram (derived from the Fast Fourier Transform and/or other transformations) as an input to a CNN and to apply convolving filter kernels that extract patterns in 2D. The presented approach advances an earlier publication related to our winning contribution to the MIREX 2015 music/speech classification and detection task, where we have shown the successful application of Mel-spectrogram based Convolutional Neural Networks on for the task of music/speech discrimination with 99.73 % accuracy [1]. Based on a number of alterations in the architecture of the Convolutional Neural Network we showed that a combination of a CNN that captures temporal information and another one that captures timbral relations in the frequency domain is a promising approach for music genre recognition [2].

2 Approach

Our approach trains Convolutional Neural Networks on Mel-scale spectrograms derived from the audio input. We describe these two parts in more detail.

2.1 Audio Preprocessing

Before being input to the neural network, a few preprocessing steps are carried out on the original audio which are depicted in Figure 1.

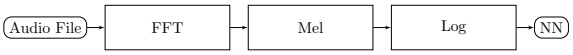

Figure 1: Audio preprocessing

First of all, a stereo audio signal is transformed to mono by averaging the two channels. Then, we apply a Fourier transform with a Hanning window of 3072 samples[1] length and 50 % overlap to compute a spectrogram. The choice for this window length

[1] All sample indications are given for 44 kHz and will be automatically adapted proportionally for 22 or 11 kHz input.

was motivated by the intention to triple the length of the 'default' window of 1024 samples to cover a larger time span, while keeping the number of frames equal (see below). Following the FFT, a Mel filterbank is applied to the spectrogram, deriving 40 Mel frequency bands. Subsequently, we perform a Log_{10} transform of all values.

This process is performed on chunks, or segments, of 124,416 samples length (2.82 seconds), resulting in 80 frames. The idea is to process a multitude of short-term segments from an audio example to be learned by the neural network. Our framework has the option to extract such segments consecutively from a file (which would result in $\tilde{1}0$ segments from a 30-second audio file). We, however, also implemented an option to extract any arbitrary number of such segments from random positions in the audio file. We set the number of excerpts to extract to 30 random segments. By that, a form of data augmentation is performed implicitly (without any audio transformations, however).

The extracted data matrices have a shape of 40 Mel bands$\times$80 frames.

2.2 Convolutional Neural Networks

2.2.1 Model 1

Our first model is a single layer CNN with a filter kernel size of 10 $\times$12 followed by a Max Pooling layer of size 1 $\times$20 (see Figure 2). The layer applies 30 such filters which are subsequently pooled drastically on the time axis, meaning that the result will mostly capture relations on the frequency axis, i.e. timbral characteristics.

2.2.2 Model 2

Following [2] we use a parallel CNN architecture for our second model, which comprises a CNN Layer

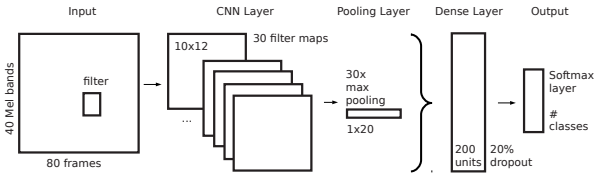

Figure 2: One layer CNN architecture (model 1)

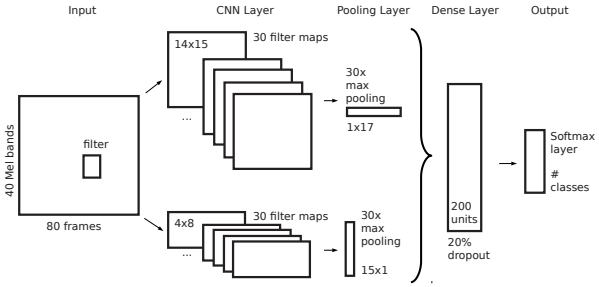

Figure 3: Parallel time/frequency CNN (model 2)

optimized for capturing relations in frequency domain, and a parallel one which is aimed at capturing temporal relations (see Figure 3). Both parts of the CNN architecture use the same input, i.e. the 40 bands$\times$80 frames Mel spectrogram matrix, as described in Subsection 2.1. In each epoch of the training, multiple training examples, sampled from the segment-wise Mel extraction of all files in the training set, are presented to both pipelines of the neural network. In each of the two CNN pipelines we use 32 filter kernels, followed by a Max Pooling stage. However, the two pipelines differ in shapes of both the filters and the pooling sizes. Setting the filter and pooling parameters is less straight-forward than in image retrieval where typically quadratic shapes are used. In audio analysis, however, the different semantics of the two axes need to be taken into account. The parameters we describe were found after a larger set of experiments.

In our architecture, the upper pipeline is aimed at

modeling frequency relations. Its filter kernel sizes are set to 14×15 and the Max Pooling size to 1×17. This means that the output of the filtering step is 32 matrices of shape 67×68, which are then "pooled" to 32 matrices of shape 67×4, preserving more information on the frequency axis than in time. On the contrary, the lower pipeline uses filter sizes of 4×8 and pooling of 15×1, aggregating on the frequency axis and therefore retaining more information on the time axis: Its output shape is 5×75 (32 times).

In the next step, the parallel architecture is merged into a single pipeline, by flattening all the matrices from both previous pipelines, concatenating them and feeding them into a dense (fully connected) layer with 200 units. The last layer is a so-called Softmax layer: It connects the 200 units of the preceding layer with as many units as the number of output classes and applies the Softmax function to guarantee that the output activations to always sum up to 1 [3]. The output from the Softmax layer can be thought of as a probability distribution and is typically used for single-label classification problems.

2.2.3 Model 3

Our third model is an experimentation, where two pipelines with different filter shapes are created to learn different frequency characteristics. One pipeline processes a filter shape of 10×12 while the other uses a much larger filter of 10×34. As with model 2, both pipelines learn 30 filters of these shapes, respectively. In both pipelines, we use the same max pooling strategy in time: 1×20. All the other parameters remain the same as described in model 2 before.

2.3 Activation and Initialization

All layers are initialized with the Glorot uniform initialization [4]. In all architectures, we apply Leaky

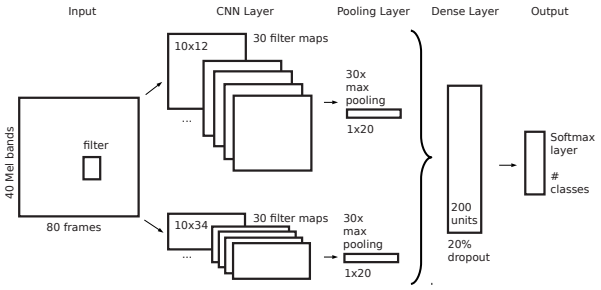

Figure 4: Parallel frequency CNN (model 3)

ReLU activation with $\alpha = 0.3$ in all Convolutional layers, and Sigmoid activation in the dense layers. The *Leaky ReLU* [5] is an extension to the *Rectified Linear Unit* [6] that does not completely cut off activation for negative values, but allows for negative values close to zero to pass through. It is defined by adding a coefficient α in $f(x) = \alpha x$, for $x < 0$, while keeping $f(x) = x$, for $x \geq 0$ as for the *ReLU*.

We apply a *Dropout* value of 0.2 to the dense layer. Dropout is aimed at reducing overfitting by dropping a percentage of random units at each weight update [7, 8].

2.4 Learning Parameters

Both CNN architectures are trained over 150 epochs with a constant learning rate of 0.02. The model is adapted in each epoch using Stochastic Gradient Descent (SGD) and a mini-batch-size of 40 instances.

2.5 Implementation

The system is implemented in Python and using *librosa* for the Mel transform and *Theano*-based library *Keras* for Deep Learning. It supports training of the Neural Network on a GPU.

References

[1] T. Lidy, "Spectral convolutional neural network for music classification," in *Music Information Retrieval Evaluation eXchange (MIREX)*, Malaga, Spain, October 2015.

[2] J. Pons, T. Lidy, and X. Serra, "Experimenting with musically motivated convolutional neural networks," in *Proceedings of the 14th International Workshop on Content-based Multimedia Indexing (CBMI 2016)*, Bucharest, Romania, June 2016.

[3] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com

[4] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 9, 2010, pp. 249–256.

[5] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *ICML 2013*, vol. 28, 2013.

[6] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," *Proceedings of the 27th International Conference on Machine Learning*, no. 3, pp. 807–814, 2010.

[7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv: 1207.0580*, pp. 1–18, 2012.

[8] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, pp. 1929–1958, 2014.