Optimizing convolutional neural networks for singing voice detection

Jelmer T. Alphenaar
10655751

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
MSc Karen Ullrich

University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 24, 2016
1 Abstract

We propose an optimization of architectures of convolutional neural networks used in the task of detecting the vocal regions of music. The network is trained on a low-level feature set of MFCCs to discriminate vocals from pure background instrumental music. Additional optimization of classifier threshold and post-processing methods have been conducted. Results are compared on a publicly available data set which current state-of-the-art methods use. More testing of the network architecture is needed to achieve better results on this data set.

Index terms- Convolutional neural network, singing voice detection, music information retrieval, deep learning
2 Introduction

Singing voice detection (SVD) is used to determine the regions of a song where a singing voice is active. Most music contains leading vocals accompanied by instrumental background music. These vocals are often only present part of the time, while at other times there is a purely instrumental part. The detection is essentially a binary classification task where segments of music are either classified into singing or accompaniment. In the Music Information Retrieval literature, it is typically assumed that the singing voice dominates over the background music Salamon and Gómez (2012), whereas singing voices such as a choir or a cappella are usually not considered as the target for detection. Identifying the presence of human vocals in music is an important pre-processing step for a range of real-world applications where only the presence or absence of them is important. They include: segmentation of vocal and non-vocal segments, subtitling lyrics (for example for a karaoke application as seen in Shenoy et al. (2005)), query-by-humming and singer identification.

improving quality of audio conferencing software with methods such as echo cancellation, speaker recognition, bit rate reduction, and noise suppression while voice is absent. While this task does not seem very difficult for humans, regardless of the instrumental background, characteristics of the singer’s voice, language used, the automatic classification by an algorithm remains difficult. Several improvements have been made in this field that decrease computation time and improve accuracy of classification, which makes it seem real-time classification of singing voice activity is achievable.

While this problem may seem similar to that of voice activity detection, where the presence of speech is detected, there are in fact many differences to these two
types of problems. These tasks share some characteristics, but detecting vocals is a much more complex task for a number of reasons. For example, the singing voice covers a wider range of variation in terms of singing range and timbre compared to speech, as well as there often being some correlation between the singing and the background music which means there is an overlap on the frequency components. In addition, the great amount of variety of artists and instruments makes the characterization of music more complex. In contrast to the speech/music discrimination task done by Chou and Gu (2001), the assumption in SVD is made that the input signal consists only of music and the goal is to detect the regions with presence of a singing voice in it.

Like many other recent studies, a learning model is proposed for the detection of voice activity in music. We use a convolutional neural network (CNN) which is trained using only a spectrogram of mel-frequency cepstral coefficients (MFCC) as input, instead of a hard-coded model that works for all input data. The method is very similar to Schlüter and Grill, one of the state-of-the-art methods. They use CNNs to detect singing and focus on data augmentation for this problem, but we will instead focus on optimizing the network for this problem since they only used one network which is not optimized for this task. We will compare our results with them, as well as Ramona et al. (2008) who annotated the data set used in this thesis, Lehner et al. (2015) who use recurrent neural networks (RNN) with a number of high-level features, Leglaive et al. (2015) who use a bidirectional RNN.

In this thesis similar works on this subject along with their results are first compared, followed by a description showing how this was done with CNNs and in more detail how CNNs work. Some optimizations to the network architecture will be presented that gave better results on the test data and a summary is given showing room for improvement for future research.
Code is available online at github.com/jelmerta/karaokeai, but please keep in mind that it is not well-documented.

3 Background

First, information about the way music can be represented will be given and we will go into more detail with the chosen audio descriptor. After that, neural networks will be explained with a focus on convolutional neural networks. Parts of the architecture of the network, such as convolutional layers, fully-connected layers and activation function, as well as regularization methods, will be described. Lastly, in the literature review, an overview of existing methods is given, starting with older methods and ending with the current state-of-the-art. A number of publicly available data sets are introduced and then methods are identified which our results are compared to.

3.1 Music representation

Analyzing music is a challenging task, since a lot of data is contained within which can be interpreted in many ways - people do not have the same perception of music. Features will have to be extracted from the input signal to reduce dimensionality of the data, which lowers computation time while keeping the essence of the original song intact. Several audio descriptors have been used for this task and comparisons have been made by Leung et al. (2004) and Rocamora and Herrera (2007). The former compares classification on feature sets based on perceptual linear prediction (PLP) features. A feature set of a supervised version of independent component analysis (ICA-FX) and generalized likelihood ratio (GLR) distance are created and classified with a support vector machine (SVM). ICA-FX performs better on the
test set in all of their experiments. The latter compares classification on a wide range of feature sets: mel-frequency cepstrum coefficients (MFCC), log frequency power coefficients (LFPC), harmonic coefficients (HC), PLP, as well as two general purpose feature sets used for classification of musical instruments. SVMs were used to classify their own manually labeled data set and the results show that MFCCs outperform the other descriptors by more than 6%. The results on these descriptors also show that spectral content of the audio signal seems to contain the most appropriate features for this problem (which are MFCC, LFPC, spectral coefficients and PLPC). Although different data sets are being used by both studies, MFCCs clearly outperform PLP-based features. This seems in line with many more recent studies using only MFCCs as their features as in Lehner et al. (2013) and Schlüter and Grill, or MFCCs along with other audio descriptors as in Lehner et al. (2015), which is why we will also use MFCCs as our feature set. We hope the neural network can find additional higher-level features which represent the music and are fitted for this task, such as first-order derivatives of the MFCC. Adding additional features that are not captured by the filters in our network learned on the MFCCs would of course still be beneficial.

3.1.1 MFCC

A mel-frequency cepstrum is used to represent a short-term power spectrum of a sound. It consists of mel-frequency cepstral coefficients which are low-level features derived from a cepstral representation of the audio. A difference with normal spectrograms is that an MFC has its frequency bands equally spaced on the mel scale, which instead of using linearly-spaced frequency bands, approximates the response of the human auditory system better. What this scale means is that pitches perceived by listeners to be equal in distance from another are evenly spaced. Especially in higher frequency ranges (above 500 Hz), the actual pitch increase is a
lot higher than the perceived increase, which can allow for a better representation of sound.

To create MFCCs, a short-time fourier transform (STFT) with Hann windows is first taken on a single channel of the original audio data, followed by applying a mel filterbank with triangular filters to map the power of the spectrum onto the mel scale. The magnitudes are then logarithmized. Lastly, the discrete cosine transform of the list of log powers is taken. The MFCCs are then the amplitudes of the resulting spectrum. There are many variations to this process with addition of dynamic features or differences in the spacing or shape of the windows used to map the scale.

3.2 Neural networks

Because the stated problem is not a very simple task, proposed hard-coded models often do not have a very good performance. Most if not all of the more recent studies done on this topic have been using a learning model such as support vector machines, random forest or neural networks to predict classifications on the feature set. These models have dynamic parameters which change during training phase where data is fed into the model to optimize them. In this section, we explain how convolutional neural networks work and how they can be used to be more efficient with visual data, since the representation of features we use in this thesis is a 2-dimensional image.

3.2.1 Convolutional neural networks

CNNs are very similar to ordinary neural networks. They both have layers of neurons with weights and biases which are trained to map some input value to an output value acquired by performing the dot product of the parameters and the input, often followed by some non-linearity. The output of the network is calculated
in the same way and neuron still have a loss functions in the last layers of the CNN. A big difference is that CNNs make the assumption that inputs to the network are images. Regular network have hidden layers where each neuron is fully-connected to all neurons in the previous layers and where neurons each function on their own and have no connectivity to other neurons in the same layer. These same layers are still used in convolutional networks, but are only used at the end of the network to predict class scores if the network is used for classification problems. Ordinary neural networks do not scale well to full images with their many features, whereas CNNs use convolutions to decrease the amount of neurons needed, which reduces the amount of parameters in the network, which in turn decreases the amount of processing required.

While convolutional neural networks have only been used efficiently since the 2000s, they have been around since the 80’s. The reason that only more recently they have become more popular is is mainly because of improvements to the processing power of GPUs. CNNs often run on GPUs instead of CPUs because they outperform CPUs when it comes to computations such as matrix-matrix multiplications, which is what these machine learning operations are usually reduced to, and software is well-optimized for these operations.

**Convolutional layer** The convolutional layer is the core of CNNs and the reason it works so well for images. Its weights consist of small filters which can learn some structure in the given input volume, as well as the usual bias value. Both the input volume to the layer and the filters are 3-dimensional. During forward propagation, the filter is slid across the image and convolved with the matching input volume. This means that a neuron is only connected to a smaller regions of neurons
in the previous layer. The result in the next layer is a new 2-dimensional activation map which contains the associated value of the filter at every spatial position in the image. After having successfully trained the network, the filters in the first layer might now activate when some type of visual feature such as a blob, color, or some form of edge are detected. In the next layer, more abstract features may be learned from the inputs of first layer such as a cross or green circles. Eventually, even activation filters with features such as chairs or types of animals could be created. Each one of these filters will create this activation map and these are stacked together to form a 3-dimensional output volume.

Three hyperparameters control how the filter are used on the input: the stride, depth and zero-padding, which in turn determines the dimensions of the output volume. The stride defines how we slide the filter over the input. A stride of 1 means the filter will move 1 pixel at a time. A higher stride will produce a lower dimensional output volume. The depth relates to the amount of filters used in the layer. Lastly, zero-padding is a useful method to directly control the spatial size of the output volume. It does this by padding the input volume with a certain amount of zeros around its border. These hyperparameters are often set so the output volume has the same width and height as the input volume. A pooling layer is then used to reduce width and height of the output volume.

As an example, when a color image is used with RGB values, Filters in the first layer often have a size of 3x3x3 or 5x5x3, where the first two dimensions refer to the width and height of the convolution and the third dimension refers to the depth of the input volume, which is 3 for RGB images since each pixel contains 3 data points.
Just to be clear: All neurons in the same depth share the same parameters (weights and biases). The reasoning is: If a feature learned by a filter is relevant at a certain location \((x_1, y_1)\) in the input volume, it should also be useful at another location \((x_2, y_2)\). This is the reason why only a fraction of the parameters are required compared to ordinary networks - locations of features are fairly irrelevant in most situations. During backpropagation, gradients are still computed for each neuron, but these are added up across each of the depths and will only update the set of weights once. One drawback of this method of sharing the parameters is that it does not always make sense. For example, when a data set is used where all images are similarly centered and different features are expected in different parts of the image, such as pictures taken from passports. In these, you would expect a face that is centered in the frame to have eyes, hair and other details of the face in fairly specific locations. Features should therefore probably also be computed in these areas of the image instead of the whole image to increase classification accuracy. A solution for this is to use locally-connected layers which we will not further discuss here.

**Pooling layer**  To reduce spatial size of the volume while it is progressing through the convolution layers, a pooling layer is commonly inserted periodically in-between successive convolution layers. This in turn reduces the number of parameters required to fit the new representation and computation in the network, which also helps avoid overfitting. Most commonly, a small filter of size 2x2 is used with a stride of 2 and then a function is used to determine a value for the 4 values found to reduce the width and height of the volume by 2, which results in a volume only 25% the size of the original. A popular function is to take the highest value in the filter (using the MAX operation). This filter operates on every depth layer inde-
pendently and the dimension of the depth does not change. Pooling layers are very slowly losing popularity in favor of using larger strides to reduce the spatial size which in some cases result in better models.

Architectures Now that we have a full understanding of the different types of layers in convolutional networks (convolutional, pooling and fully-connected), we can describe the most common type of architectures used in them. First, the input volume goes through a convolution layer after which some element-wise non-linear activation such as RELU is applied. Then, either another one of these layers is applied, or pooling is used to reduce dimensionality. In some models, it might be beneficial to do these steps several times, such as when very abstract concepts such as cars or cats have to be detected. After this, one or more fully-connected layers are used to know the activated filters in the previous layer which results in classification scores if a classification setting is used in the network.

3.2.2 Regularization

While using neural networks many problems may arise. One such problem is overfitting, where the network did a very good job fitting its model to the training data, but failed to generalize to data points outside of the training set. There are several reasons this might happen, perhaps the model is too complex (it has too many parameters) or too little data is available. Several regularization methods, such as dropconnect, dropout, L1 and L2 regularization and early stopping exist, which reduce overfitting.

Dropout A method to reduce overfitting used by many state-of-the-art algorithms that incorporate neural networks is dropoutSrivastava et al. (2014). Dropout refers to randomly ”dropping out” units along with their connections in layers of the network when forward propagating and creating several thinned networks with dif-
ferent inputs active. Usually, this method is used on fully-connected layers where about 50% of the input units are dropped. When on some occasions it is used in a different layer such as the input layer, it should probably drop less units for better results. When forward propagation on the networks is finished, the model back-propagates to update parameters like normal. Usually, model combination would be a very expensive method to increase performance, but with dropout, this is now feasible. All the separate networks are trained on and then combined into a new network with scaled-down weights. This means that during testing it is not necessary to feed the input to multiple networks, since this approximate averaging method is used to get a prediction for the output value. Since the networks likely differ a lot in structure - different units are used - The units will not be co-adapting too much, meaning their values change independent of each other. This is a very useful method, since it helps prevent the weights of units from converging to identical positions, which prevents the output to get stuck in a local optimum. It is important that dropout is only active during training and not when testing, because all parameters in the model are now trained and will give better results on the set than just part of the network.

3.3 Related work

In this section, a number of comparable methods will be described that attempt to solve this problem. We will see that studies have shifted from older methods, which often use high-level features as the input and are very hand-crafted i.e. they threshold audio descriptor values to classify to newer methods mostly using some type of learning, often with lower level features. Current state-of-the-art methods build on CNNs or recurrent neural networks. The methods currently performing the best are Lehner et al. (2015), Leglaive et al. (2015) and Schlüter and Grill (results of their proposed methods can be seen in 1 in the evaluation.)
One of the methods that does not use learning is Regnier and Peeters (2009), but instead relies on the extraction of the characteristic specific to singing and use a simple threshold method to classify. They report an F-measure of 0.768, but they also get rid of non-vocal segments shorter than one second which may increase performance.

Ramona et al. (2008) first introduced the data set which is still used by current state-of-the-art methods and proposed a support vector model, based on a large feature set. Temporal smoothing of posterior probabilities with a hidden Markov model is used to improve results.

Lehner et al. (2015) trained an RNN on a set of five high-level features, some of which where especially designed with this task in mind. They studied these earlier in Lehner et al. (2014) where they introduce temporal characteristics of the signal to reduce the amount of false positives in their earlier results. They achieve the second best scores on the Jamendo data set.

Leglaive et al. (2015) trained a bidirectional RNN on mel spectra that have been pre-processed with a highly-tuned stage where the harmonic and percussives have been separated. They used a ”shotgun approach” where they train 20 variants and then picked the one that performed the best on the test set.

Schlütter and Grill focus on augmenting data for this task and use CNNs to classify mel spectograms. They basically introduce additional data to a given data set by augmenting it with noise, pitch shifting, time stretching, random frequency filtering and other methods and report very good improvements in results, with as low as 7.3% classification error.
4 Method

In this section is described how a convolutional neural network was designed and optimized for use on MFCC features of audio files to detect activity of vocals.

4.1 Data

Originally, the idea was to use a large data set of karaoke music with labels extracted from the visual component to train the model on, but the extracted labels did not match the length of the audio track so a different data set had to be used. As mentioned in the literature review, several data sets of audio files are publicly available specifically for this problem. Some of them include: Jamendo, RWC and MIR-1K. In this thesis, we will use the Jamendo collection, which was made by Ramona et al. (2008) who collected and annotated the music with singing labels. It contains 93 copyright-free songs, all made by different artists from a variety of genres, from the Jamendo music sharing website. The songs are sampled at 44.1 kHz and range from about 2 to 6 minutes in length. An example song is shown with its labels in figure 3b. The data is classified as ”singing” approximately 55% of the time, which can be used as a base comparison. This data set has been used in most studies in this field and thus makes comparison of the different proposed methods possible. For comparison with other research, we will use the the original split with 61 files for training, and 16 for both validation and testing.

4.2 Feature Extraction

Significant features are extracted from the songs to reduce dimensionality of the audio data and to represent them in a way that is better to process for convolutional neural networks. To do this, a Mel-frequency cepstrum will be used. The audio files are downsampled to 24KHz and then for each audio file, a spectogram of MFCCs
is created using the YAAFE library, with a block size of 50 ms, a step size (or hop size) of 16.6 ms and 40 cepstral coefficients, including the first coefficient. 80 triangular filters are used from 27 Hz to 8 kHz. Most of these specifications for the MFCC are taken from Schlüter and Grill. At first, we wanted to use frequencies in the range of about 50 - 1500 Hz, since this approximates the range of human vocals and it is known that for humans lower frequencies are more important for recognition, but it might also be a good idea to look outside of this range so that the instrumental music during inactivity of singing is also taken into account. A step size smaller than the block size means that there will be some overlap in the acquired blocks, which should result in a smoother transition between blocks. After all the MFCCs have been created, we normalized resulting coefficients over the whole set by calculating the z-scores of them, but actually got worse results even after optimizing the learning rate, so no normalization is done. Similar to other studies, longer frames of 200 ms will be classified, but the total window over which this frame is classified can be bigger, because context will be taken into account. A window of 1.4 second, which means the frame has 600 ms of context from both sides, is used for classification. So for every frame of 200 ms that are to be classified, 84 (12 * 7) blocks of 16.6 ms will be used, resulting in an image of 84 by 40 which is used as the input to the network for classification.

Figure 1: Raw audio file, normal spectrum and MFCC spectrogram(Spectograms, 2007)
4.3 Neural network architecture

We will describe the architecture we will use to compare other architectures. We will use this architecture and alter small components, such as the activation function or convolution layer and pooling, run these networks several times and then compare their performance on the test set. The convolutional network that was originally used has 2 convolutional layers. The size of the first convolution filter is 5x5 and outputs 32 features for each patch. The second layer has a filter of 3x3 and outputs 64 features for each patch. After each convolution layer, max pooling is applied on every 2x2 patch, which scales the image down to half its size. Then, a fully connected layer with 1024 outputs is applied, which combines each of the features of the previous layer. Dropout is applied to reduce overfitting of the parameters, before entering another fully connected layer, which takes the 1024 features as input and outputs the 2 resulting features which will be used to classify them into the two classes with the softmax function and then taking the argmax of these values to compare them to the ground truth.

We will change the activation filter in the last layer from softmax to sigmoid and compare these. Sadly, due to time constraints, no runs on other architectures have been conducted.

4.4 Training

As gradient descent optimization algorithm, we will be using the ADAM optimizer, which adds bias-correction and momentum to RMSprop. Ruder (2016) The bias-correction is reported to help outperform RMSprop slightly towards the end of optimization as gradients become sparser. Other optimizers have been tested as well, but the rate of change in loss did not seem to change much. A learning rate of $1^{-4}$ was found to be close to optimal.
The networks are trained with TensorFlow on a GTX TITAN X GPU and looping over the whole training set once takes approximately 3 minutes.

4.5 Hyperparameter optimization

4.5.1 Finding optimal classifier threshold

Normally, the threshold for the vocal class is 0.5, but to test if this value generalizes well to the test set, predictions are made on the validation set using varying thresholds for values between 0.4 and 0.6 with an interval of 0.01.

4.6 Post-processing

A number of methods are often used to improve results on the acquired predictions given by the network. We use smoothing to find a better threshold for classification which changed the error on the test set slightly.

4.6.1 Smoothing

To improve accuracy on the test set, two types of smoothing were applied to the activations found by the neural network: a gaussian filter and a median filter. The reason this may work is that because the data is only generally classified, average predictions around the specified value may sometimes be more representative of the classification - it takes some temporal information into account.

A gaussian filter was first applied with a standard deviation of 1 is taken with a limit of 4 windows around the window to be classified. Next, a median filter was used, which took a few windows around every window in the activation list, and the classification of the median (the middle one) is reassigned to that window. For optimization, several ranges were tested and a range of 9 seems to give the best
5 Results/Evaluation

Due to time constraints, this section is sadly still missing some key research.

We tried to put more filters in the convolutional layers but actually got worse results. This would likely be a case of overfitting.

Finding the optimal threshold on the validation set led to a threshold of 0.48 performing slightly better, but predictions on the test set did however not improve.

The smoothing was tested in an earlier stage where we had an average accuracy after 5 tests of 72.0% without smoothing. Smoothing helped out slightly and brought this up to 72.4% after smoothing with the median filter.

When using the sigmoid activation function instead of softmax in the classifying step, we were surprised to see it was very often only learning to always output 1, which gave us the accuracy of 55%, just like the ratio of ones in the test set. Perhaps a different approach is necessary to make sure a sigmoid activation is able to classify better. Schlüter and Grill report setting the target value to 0.02 and 0.98 instead of 0 and 1 to avoid driving the output layer weights to larger magnitudes while the network attempts to optimize them when it finds training examples it already got, which makes them reach the asymptotes of the sigmoid function. Due to time constraints, we did not attempt to put their target values do attempt to solve this problem.
The following graphs show our predictions and the ground truth on one of the songs from the test set.

Results of the network before smoothing can be seen in 2a

Results of the network after smoothing can be seen in 3a

Results after thresholding the predictions with a value of 0.48 can be seen in 2b

The ground truth of the song can be seen in 3b

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision (%)</th>
<th>Acc. (%)</th>
<th>Recall (%)</th>
<th>Spec. (%)</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed method</td>
<td>78.5</td>
<td>76.7</td>
<td>78.5</td>
<td>82.8</td>
<td>.764</td>
</tr>
<tr>
<td>Ramona et al.</td>
<td>-</td>
<td>82.2</td>
<td>-</td>
<td>84.3</td>
<td>-</td>
</tr>
<tr>
<td>Lehner et al.</td>
<td>89.8</td>
<td>89.42</td>
<td>90.6</td>
<td>-</td>
<td>.902</td>
</tr>
<tr>
<td>Leglaive et al.</td>
<td>89.5</td>
<td>91.5</td>
<td>92.6</td>
<td>-</td>
<td>.910</td>
</tr>
<tr>
<td>Jan Schlüter et al.</td>
<td>-</td>
<td>-</td>
<td>90.3</td>
<td>94.1</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Our results compared to current state-of-the-art

6 Summary

It is clear from other research that CNNs can be used in the task of singing voice detection to receive very good scores. We have shown that activation using the softmax filter outperforms sigmoid in this task. Sadly, there was no time to check
(a) Predictions of the network, smoothed with median filter
(b) The ground truth labels given for the same music file in the Jamendo data set

for additional network architectures for better performance.

There are still several improvements other than architecture to be made. Of course, additional features such as fluctograms, spectral flatness, spectral contractions as in Schlüter and Grill could be used, as well as other audio descriptors that have been mentioned before like PLP and LFPC features. Another feature we have not seen in other similar studies is phonemes, which is the way words are pronounced. This is a very high level feature which might be too hard to associate with the music, but if it does it would be very easy to predict singing if a phoneme is found. This would however take a lot of time to annotate the data with phonemes. Lastly, data augmentation seems to have been quite successful in Schlüter and Grill and could be used to improve results.

Batch normalization Ioffe and Szegedy (2015) could be added to the neural network, which is a method to reduce the effect of internal covariate shift caused by saturating nonlinearities in the volume. This method allows higher learning rates to be used by normalizing inputs to each layer which would make training faster and potentially make better predictions. Another improvement to the model is mentioned by Schlüter and Grill, which mentions adding recurrent connections to the hidden layers might help to take more context into account without using
too much processing power. Two of the models discussed use them and do indeed have very good scores.

Another thing to note is that the data is not very precisely annotated. It only gives a general classification in a larger time frame where small bits of quietness are not annotated as 0. It is also quite small in size, 93 songs is not very representative of all music. A better data set would improve scores, but would of course also make comparison with other methods impossible.
7 References


8 Appendix

8.1 CD+G to labels

A lot of time was first spent to create labels from graphical karaoke data, which did eventually succeed on a smaller set of the data, but was not useful after we found out the audio and visuals often did not start or end at the same time and the length of the visuals and music often did not match up in the complete data set.

Implementation of this is available at github.com/jelmerta/karaokeai in the file cdg2voicetiming.py