Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Using Deep Learning to Annotate Karaoke Songs

Semester Thesis

Juliette Faille

faillej@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Gino Brunner, Yuyi Wang
Prof. Dr. Roger Wattenhofer

January 7, 2018

Acknowledgements

I would like to thank Gino Brunner and Yuyi Wang for their support and helpful advice. I am very grateful for the many ideas and feedbacks they gave me during weekly meetings.

Abstract

Karaoke is a game in which players sing over pre-recorded instrumental backing tracks. To help the singer, lyrics are usually displayed on a video screen. The synchronization between the lyrics display and the song record, often done manually, is a tedious and time-consuming task. Automation of the annotation of karaoke songs can help save time and effort.

In this thesis we use the representation of songs as spectrograms to detect singing times. This timing information can be used later to align the lyrics display with a sound track. Convolutional neural networks are trained to detect at any moment in a song whether the artist is singing or not.

Contents

Chapter 1

Introduction

1.1 Motivation and Previous Master Thesis

Annotating karaoke songs means associating the lyrics of a song to the audio file with timing information. Collecting the lyrics of songs is quite easy, however gathering the timing annotation is much more tedious as it is often done manually.

In this semester project, I used the database resulting from the master thesis "Karaoke Song Generator" written by Vanessa Hunziker at the Distributed Computing Group of Computer Engineering and Networks Laboratory at ETH Zürich in 2016 [1]. This master thesis aimed at annotating song lyrics automatically by using two different techniques. The first was based on the alignment of two signals: the song itself and the text-to-speech signal created using the written lyrics. The second approach used crowdsourcing to annotate the data. To this end, a game for Android was developed and players had to select the lyrics they heard at the correct time.

The goal of this semester project is to investigate a new method for the annotation of karaoke songs.

We want to use a deep learning-based approach to annotate songs with their lyrics. The idea is to train a deep neural network on a annotated dataset. The model will then be able to predict the timing annotation for any other song.

1.2 The dataset

[1] resulted in the creation of a database containing for each song its mp3 file and its txt file with the lyrics.

The txt file is structured like lyrics files used in open source karaoke games like UltraStar [2]. The file starts with tags indicating for example the title, the artist, the language, the genre, the year of release... More relevant for the processing of the data in this project, the lyrics file also contains the gap, i.e. the

amount of time in milliseconds before the lyrics start and the bpm or beats per minute, i.e. the rate at which the text should display. After the tags, the lyrics and the notes are written. This information is divided into 5 columns separated with spaces. Each row of the file corresponds to a different syllable. The first column contains one of the following symbols: ':', '*', 'F', or '-'. '-' describes a line break and does not correspond to any lyrics. The other symbols indicate different scores for the player if he or she manages to find the correct timing for a particular syllable. This information is not used in this work. The second column indicates at what time the syllable is sung. That time is given in quarters of beats. The third column gives the number of beats during which the syllable lasts. The fourth column gives the number code of the syllable's pitch. The fifth column contains the syllable itself. An illustration is given in the figure 1.1.

1.3 Steps of the project

The project is divided into the 3 following steps.

- The mp3 and txt files from the dataset (see 1.2) will be preprocessed as explained in 4.
- Speech to Text Recognition is performed with IBM®Speech to Text API (see 5.1) to get the content of the lyrics.
- A neural network is trained to detect when the artist is singing. The detected voice times provide the timing information needed to align the lyrics found and the song record.

The chapter 2 defines the spectrograms and the ideal binary masks. The chapter 3 describes the two general approaches that have been studied to solve the problem of lyrics annotation. Chapter 4 explains the different preprocessing steps that were applied to the data. The chapter 5 analyses the results obtained with the IBM®Speech to Text API and with the neural network. Finally, chapter 6 concludes and gives ideas for improvement.

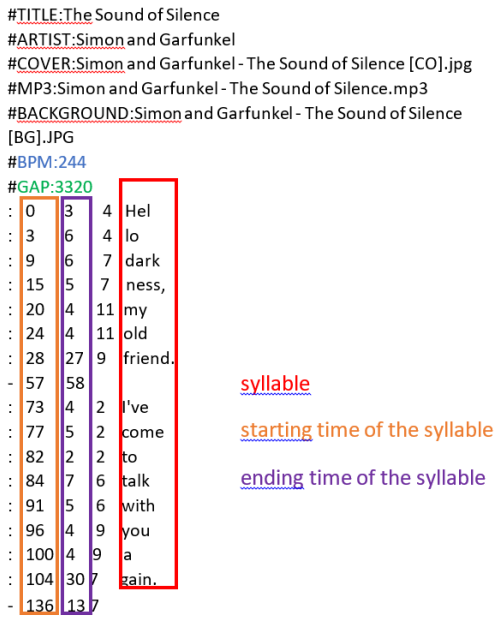


Figure 1.1: First lines of the file "Simon and Garfunkel - The Sound of Silence.txt"

Chapter 2

Spectrogram and Ideal Binary Mask

2.1 Spectrogram

2.1.1 Definition

A spectrogram is a representation of the spectrum of frequencies of the song record as they vary with time. The horizontal axis represents time and the vertical axis is frequency. The amplitude of a particular frequency at a particular time is represented by the colour of each point in the image.

2.1.2 Creation

One spectrogram was created for each song of the dataset. They were created using the STFT (Short-Time Fourier Transform). To compute the STFT, the long time signal (the waveform of the song) is divided into short segments, and then the FFT (Fast Fourier Transform) is computed on each segment. The FFT is an algorithm allowing to compute a Discrete Fourier Transform. The STFT can be seen as the correlation of the signal with a function w depending on time such that the function w is real, symmetric and normalized.

The STFT of a function f is a function of 2 variables given by :

$$X(n,k) = \mathbf{STFT}\{x[n]\} = \sum_{m=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega m}$$

It can be interpreted as the multiplication of x by the window w which is centred on the time n, followed by a Fourier Transform.

For the application we used the Python function: scipy.signal.stft(signal, window='hann', nperseg=2048, noverlap=512). The computation was done with a Hann window w of size 2048 samples. The chunks resulting of successive

windowing of the signal overlap on 512 samples. These parameters were chosen according to [3]. The Hann window which was also chosen in [3] provides a good compromise between time and frequency resolution. The Hann window is defined as:

$$w(n) = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$$

with N=2048

Example:

The song *Dancing Queen* by ABBA has a record of length approximately 3 min and 52 sec (232 sec). The audio record has a sampling frequency of 44100 Hz. Thus, the entire song contains 10231200 samples (44100×232).

The STFT is computed with windows of size 2048 (in this song it represents about 46 ms) which overlap with 512 samples. Therefore, to compute the STFT of the whole song, $\frac{10231200}{(2048-512)} = 6660$ chunks are needed. Each chunk of the original signal is represented by a vertical line in the final spectrogram. Consequently, the spectrogram should be of size 6660 on the time axis. In fact the spectrogram has a width of 6665. This is due to the fact that the song does not last precisely 232 seconds but in fact its duration is closer to 232.09 seconds.

The audio signal is real, thus its Fourier transform is even. This explains why the resulting spectrograms are of height 1025 (instead of 2048) because in the signal, only half of the frequencies are needed to fully describe the Fourier transform. If the signal was complex, the height of the spectrogram would have been 2048. As the sampling frequency of the record is 44 100 Hz, the frequency resolution of the spectrogram is $\frac{44100}{2\times1025} = 21.5$ Hz. The maximum frequency represented is $\frac{44100}{2} = 22050$ Hz. Frequencies from 20 to 20 000 Hz are audible but human speech frequencies lie between 500 and 8000 Hz. Above 8000 Hz there are mostly ringing sounds, this explains why the amplitude values with frequency larger than 8000 Hz in the spectrogram are close to zero.

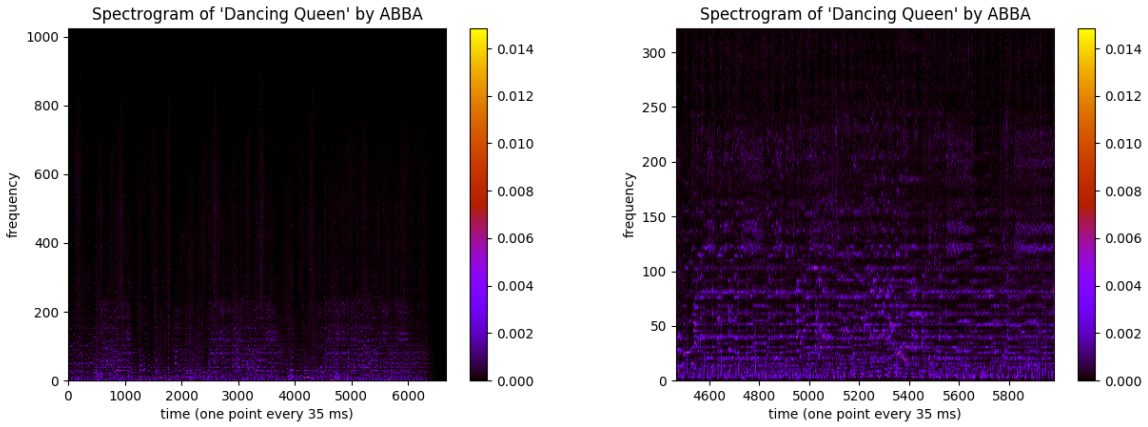
The figure 2.1 shows the entire spectrogram for the example as well as a zoomed-in picture.

2.2 Ideal Binary Mask

In order to analyse the songs, [3] uses Ideal Binary Masks (IBMs) of the songs. The IBMs are spectrograms, classically they are defined as:

$$IBM(t,f) = \begin{cases} 1 & \text{if } SNR(f) > \theta \\ 0 & \text{otherwise} \end{cases} \quad \text{typically, } \theta = 0dB$$

Each element of the mask is found by computing the Signal-to-Noise Ratio (SNR). If the ratio of the signal power to the noise power is bigger than a defined



(a) (b)

Figure 2.1: Entire Spectrogram (a) and Spectrogram between 2 min 37 sec and 3 min 29 sec and between frequencies 0 and 6884 (= $frac320 \times 441002 \times 1025$ Hz (b) of the song 'Dancing Queen' by ABBA

threshold, the element takes the value 1, otherwise it takes the value 0. In our case, the signal corresponds to the voice signal and the noise to the instrumental part of the music.

Once the IBM has been computed it can be applied to the spectrogram of the song (by multiplying the spectrogram and its corresponding IBM element by element). This representation has the advantage of reducing the problem of finding when the artist sings to a binary classification problem.

Chapter 3

Method

3.1 First approach: Voice Extraction

The first idea I considered is inspired by [3]. In order to annotate the songs, a solution could be to extract the singing voice (or voices) from the record, i.e. to remove the instrumental part. Then, speech recognition methods could be used to recognize the lyrics from the extracted vocal part. This approach would provide both the lyrics and their timing information. The spectrograms of the songs are computed. The IBM of each song of the training and test sets is also defined and computed such that:

$$IBM(t,f) = \begin{cases} 1 & \text{if } SNR(f) > \theta \\ 0 & \text{otherwise} \end{cases} \qquad \text{typically, } \theta = 0dB$$

Where the vocal part is considered to be the signal and the instrumental part is considered to be the noise.

A neural network will be used to predict the IBM for any song. To train the neural network, the set of the spectrograms of the songs is used as input data. The labels are the IBMs.

Once an IBM is predicted, it can be applied on the spectrogram of the song. The filtered spectrogram is then inverted to get the record of the voice alone, without the instruments.

By defining the mask in this way, some singing parts are possibly filtered if they are sung while the instruments are loud (so loud that the $SNR < \theta$). However, we assume that this case occurs rarely. Otherwise, it would be difficult to understand the lyrics, even for human ear.

The great disadvantage of this approach that the SNR in our case is not known. The dataset does not provide any information concerning the respective powers of the vocal part and of the instrumental part. Unlike in [3] the database I use contains only the records of the songs and not the records of the singing voice and records of instruments alone. Therefore, we should first solve a simpler

problem: "voice detection" instead of "voice extraction".

3.2 Second approach: Voice Detection

Instead of wanting to extract the singing voice, it could be enough just to detect when the singing occurs in the song. Each time the person starts singing, his or her voice is detected, which provides the timing information needed for the karaoke annotation.

Concerning the content of the lyrics, a possibility, as a first step, is to use an already existing API for speech recognition (IBM's API Watson for example) and apply the speech recognition directly on the songs. Then, thanks to the detected singing times, the lyrics found with the API can be align with the sound track.

Chapter 4

Preprocessing

4.1 MP3 files

For this part, the .mp3 files were converted into .wav files. I used the scipy.io.wavfile.read Python function to extract the waveform signal (figure 4.1) as well as the rate of the record. The songs were stereo records so I kept only one channel (the left one) and could then apply as STFT on this signal as explained in section 2.1.2. I then normalized the spectrograms with norm 2 with the Python function np.linalg.norm (figure 4.2).

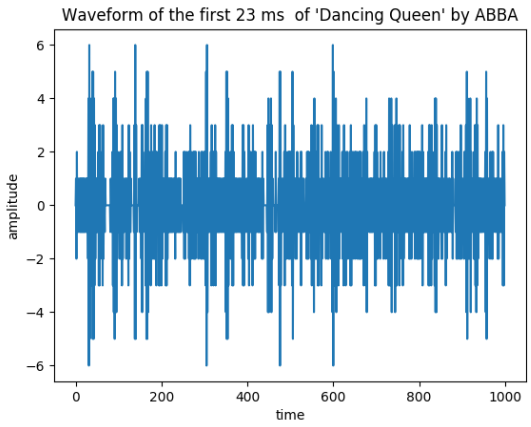
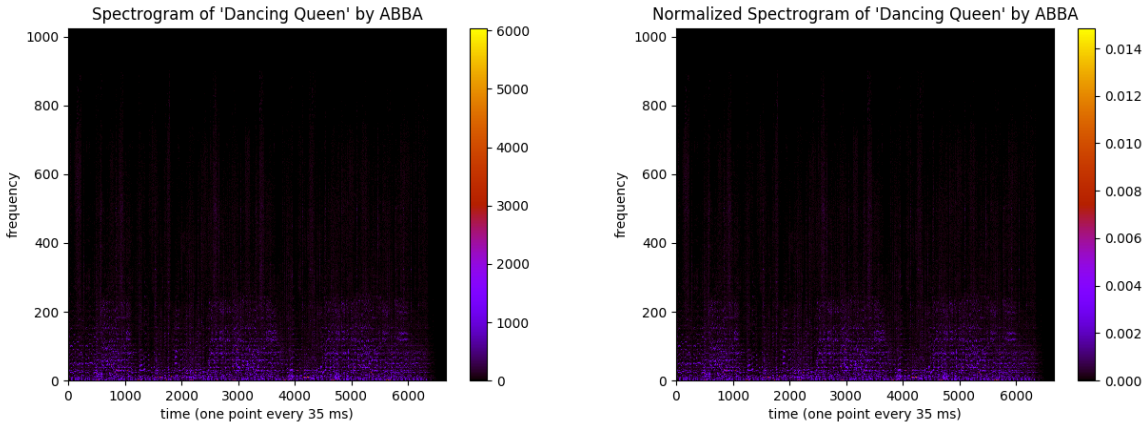


Figure 4.1: Waveform of the first 23 ms of 'Dancing Queen' by ABBA

4.2 Text files

From the .txt files, I extracted the values of the BPM and of the GAP (defined in section 1.2) as well as an array of couples $(x_{syllable\ i}, y_{syllable\ i})$ where $x_{syllable\ i}$ is the starting time in milliseconds of the syllable i and $y_{syllable\ i}$, its ending time.



(a) (b)

Figure 4.2: Spectrogram (a) and Normalized Spectrogram (b) of 'Dancing Queen' by ABBA

That array thus contains the starting and ending times of all the syllables in the song.

$$x_{syllable\ i} = GAP + \frac{60000}{BPM} \times \frac{starting\ time\ of\ syllable\ i\ in\ quarter\ of\ beats}{4}$$

$$y_{syllable\ i} = x_{syllable\ i} + \frac{60000}{BPM} \times \frac{ending\ time\ of\ syllable\ i\ in\ quarter\ of\ beats}{4}$$

Note: It can be seen that the created array can be assimilated to an IBM as described in section 2.2.

$$IBM(t,f) = \begin{cases} 1 & \text{if } the\ artist\ is\ singing \\ 0 & \text{otherwise} \end{cases}$$

It is clear that this representation (example on figure 4.3) is redundant and only one row is enough to describe it. This mask can then be applied on the song spectrogram in order to remove from it all the parts of the song when only the instruments are playing.

An important observation to make is that the annotation of the starting and ending times in the dataset .txt files have been made during the master thesis [1] for a karaoke application and not a voice detection application. In fact, their accuracy is perfectly valid for lyrics display in a karaoke. However, they may not be accurate enough for a voice detection application. Indeed, some parts of the singing voice are deleted and some purely instrumental parts remain. The labels do not match exactly with the moments when the artist is singing, which can make it difficult for a neural network to detect these moments.

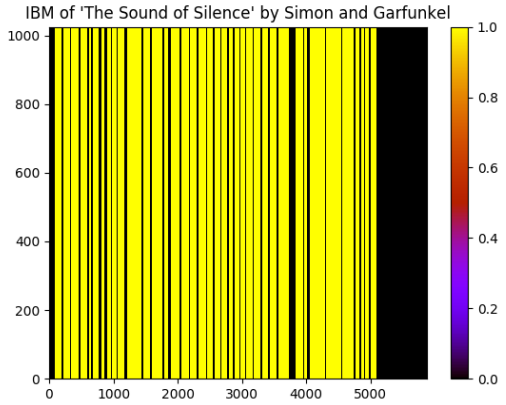


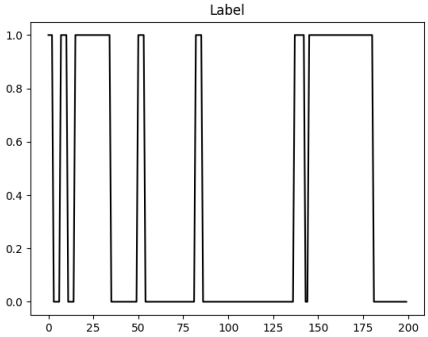
Figure 4.3: 'Ideal Binary Mask' for Voice Detection for the song 'Sound of Silence' by Simon and Garfunkel

4.3 Smoothing

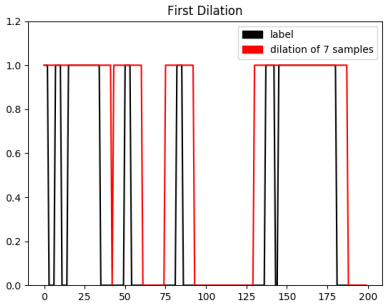
The realism of this representation can be improve by a smoother separation between the times when the artist is singing and when he or she is not singing. Indeed, the voice does not often stop suddenly but its volume decreases gradually. What is more, it usually does not make sense to detect a separation between syllables of the same words or even words of the same sentence. We can consider that blanks with duration of less than 50 milliseconds usually correspond to breathing and should not be detected as instrumental parts. Similarly, an isolated vocal part lasting less than a second does not in fact correspond to a word. Therefore, it seems meaningless to have "blocks" of zeros or ones with a size smaller than 30 samples (which correspond to approximately 1 second). The binary arrays created can be smoothed by an eroding and dilatation method (figure 4.4).

The original label (the binary array which indicates whether the artist is singing) is first dilated by 7 samples so that groups of zeros or ones with less than 15 samples are merged. Then, an erosion of 7 samples is applied to remove the ones or the zeros that have been added on edges. At this point, there are no more groups of zeros of size less than 30 samples in-between ones. This part allows to group the syllables of same words or phrases.

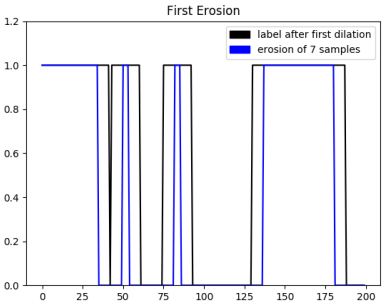
A second erosion of 15 samples is applied to eliminate groups of ones of size smaller than 30 samples. A second dilation is done to add the ones at the edges of groups of ones that have been deleted during the second erosion. This part helps eliminate very short and isolated vocal parts)



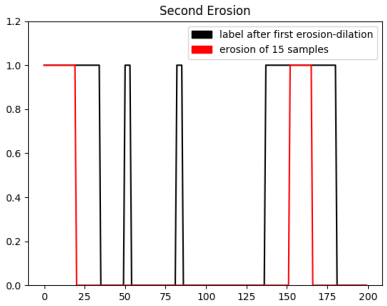
(a)



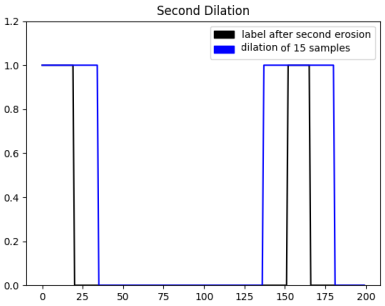
(b)



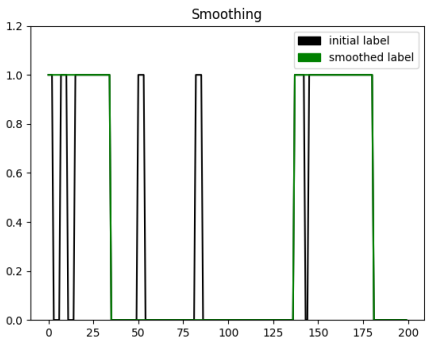
(c)



(d)



(e)



(f)

Figure 4.4: Initial label (a), label after first dilation (b), label after first erosion (c), label after second erosion (d), label after second dilation (e)

Chapter 5

Results

5.1 Speech to Text recognition

As explained in section 3.2, I used a Speech-to-Text API to retrieve the content of the lyrics. Surprisingly, the IBM®Speech to Text API [6] gives very poor results concerning the speech recognition on songs. I analysed the first 233 songs of the dataset with this tool and the mean number of words detected per song was 12.7 words, the median number words detected was 5. There are obviously not enough words to allow the use the IBM®Speech to Text API's song transcription for a Karaoke application.

Usually the best recognized songs have a very clear and close to speech singing part and a very quiet instrumental background. However even though some words or sentences are well detected, the resulting lyrics are not accurate enough to use them in Karaoke. An example of one of the better recognized songs is given in annexe (for the song "Don't Worry Be Happy" by Bobby McFerrin).

These tests using the IBM®Speech to Text API show that lyrics recognition in songs is a more difficult problem than speech recognition in spoken language. It would take more time to adapt the methods (for example neural networks) used in speech recognition to the lyrics recognition problem. Besides, collecting the lyrics of songs is not the most difficult task in annotating the songs for a karaoke application. Indeed, some databases like the Music Lyrics Database [4] contain lyrics of hundreds of thousands of songs and can easily be downloaded. Therefore, I chose to focus on the collection of timing information rather on the recognition of the lyrics' content.

5.2 Neural Network Training

5.2.1 Motivation

The goal of this semester project is to investigate how to use deep learning to annotate songs. The use of spectrograms allows to represent the song as an

image. As Convolutional Neural Networks (CNN) are particularly effective for image recognition [5] , the choice of this kind of network seemed natural.

5.2.2 Inputs

The song spectrograms are divided in spectrograms of size 200*1025. To make sure that the class 0 (no voice) is not over represented, the spectrograms containing only zeros (which often occurs at the beginning or at the end of the song) are not used. The size 200 corresponds to approximately 7s. 23835 inputs remain.

5.2.3 Labels

The binary arrays created as explained in section 4.2 are the labels. They are also divided in array of size 200 corresponding to the remaining spectrograms. The neural network has to predict a binary array corresponding to a spectrogram, i.e. predict for every time if there is a voice in the record or not. The problem of voice detection is a binary classification problem for each time step.

5.2.4 Training

The training set is composed of 19668 examples and the test set contains 4167 examples. I made sure that two examples build from the same song cannot be one in the training and the other in the test. Otherwise very similar patterns could be both in the training and test sets and create biased results.

5.2.5 Loss and Optimizer

We define the binary crossentropy loss as :

$$L(\mathbf{w}) \;=\; -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right], \tag{5.1}$$

where $\mathbf{w}$ is the vector of weights and N the number of samples. y_n is the ground truth for the sample n (the label) and $\hat{y}_n$ the CNN's prediction for the sample n.

The binary crossentropy is often used in machine learning for classification problems involving two possible classes. When the prediction $\hat{y}$ is very close to (resp. far from) the label y, the term $y \log \hat{y} + (1-y)\log(1-\hat{y})$ is very close to 0 (resp. very close to 1). In order to minimize the loss $L(\mathbf{w})$, the predictions have to get closer to the labels.

The optimizer I used is AdaGrad (adaptive gradient algorithm). This is a stochastic gradient descent with an adaptive learning rate which is often used

in image recognition. Changes in the learning rate (in a range of 0.0001 to 0.1) made no real differences on the results.

5.2.6 Size of the convolutional filters

The filters shape makes the training focus on particular features. As explained in [6], applying filters of size $m \times n$ with m and n bigger than 1 allows to learn time and frequency features at the same time. Filters with $m = 1$ helps to learn temporal features such as rythmic or tempo whereas filters with $n = 1$ are better to learn frequency features such as timbre. For the purpose of voice detection, as the main difference between singing voice and instruments are the frequency distributions and ranges. Therefore, it seems appropriate to choose a filter with large n. On the first layer, I chose to set the size of the filters such that n is equal to 1025 (the height of the spectrograms) and m takes much smaller values (between 3 and 19 samples).

5.2.7 Number of training samples

The first tests were made on smaller set of examples (with only 200 songs). The results got better when the training was done on the entire dataset.

5.2.8 Evaluation of the results

During the training the binary crossentropy losses on the training set and on the test set were saved after each epoch in order to plot the training and test loss curves. This curves help to choose after how many epochs the model is supposed to perform the best in the classification on the test set. As long as the test loss keeps decreasing, it makes sense to keep training the model. If the test loss increases on average for a certain number of epochs, the training has to be stopped.

The predicted array has elements which take real values between zeros and ones. To evaluate a prediction, its elements are rounded to 0 or 1 (if an element is smaller than 0.5 then it is set to 0, otherwise, it is set to 1).

The accuracy is defined as Accuracy $= \frac{tp+tn}{tp+tn+fp+fn}$.

- tp is the number of true positives, i.e. the number of elements in the predicted array that are equal to one with corresponding element in label array also equal to one.
- tn is the number of true negatives, i.e. the number of elements in the predicted array that are equal to zero with corresponding element in label array also equal to zero.

- fp is the number of false positives, i.e. the number of elements in the predicted array that are equal to one whereas corresponding element in label array is equal to zero.
- fn is the number of false negatives, i.e. the number of elements in the predicted array that are equal to zero whereas corresponding element in label array is equal to one.

5.2.9 Results

MLP

The first tests are carried with MLPs (Multi Layer Perceptron) with only fully connected layers. With one hidden layer, I obtained a clearly overfitting model with an accuracy of 99% on the training set and a maximum of 52% on the test set. When using dropout, I could reach an accuracy of 56% on the test set.

CNN

With the a convolutional architecture, the parameters that were tuned in the different simulations were:

- the number of layers from 3 to 6 layers
- the different sizes of the filters (1,7),(1,5), (1,3)
- different numbers of filters on each layer

The best accuraccy that could be obtained was of 60% in test (with 80% in training) with a 4 layers network with filter sizes (1025, 11) on the first layer and (1,5) on the next layers and 10 filters on each layer.

Adding a dropout layer did not change the results.

Smoothed Labels

These tests were all carried before smoothing the labels. Smoothing the label brought the biggest difference I could observe in all the different trainings. As shown in figure 5.2, the test loss decreases much more when the labels are smoothed. The confusion matrices 5.1 and 5.3 are drawn at the end of the training and the confusion matrices 5.2 and 5.4 after the epochs at which the test loss reaches its lowest point. Smoothing the labels helps the test accuracy to increase from 60% to 65%.

This shows that imroving the labels is a condition to have better results.

Table 5.1: *Confusion matrix in training without smoothed labels*

Groundtruth / Prediction	0	1
0	38%	17%
1	14%	31%

Accuracy 69%

The CNN is represented in the figure 5.1.

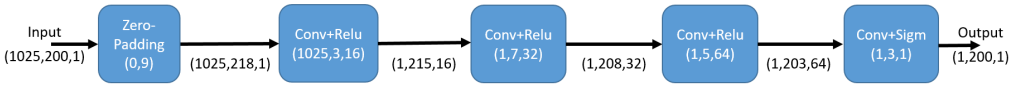
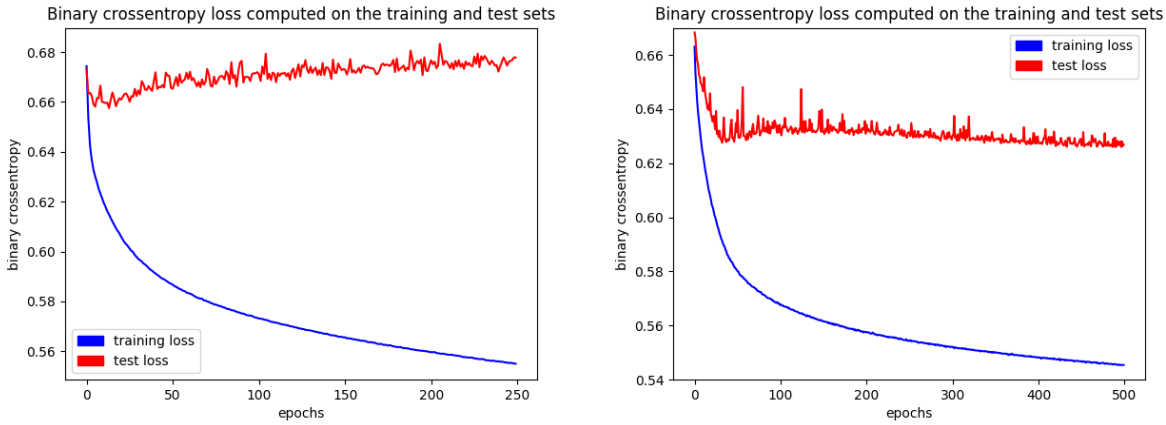


Figure 5.1: CNN



(a) (b)

Figure 5.2: Training and Test Losses with Non Smoothed (a) and Smoothed (b) Labels

An illustration of satisfying and not satisfying predictions is given in .

Table 5.2: *Confusion matrix in test without smoothed labels*

Prediction \ Groundtruth	0	1
0	35%	22%
1	18%	25%

Accuracy 60%

Table 5.3: *Confusion matrix in training with smoothed labels*

Prediction \ Groundtruth	0	1
0	22%	10%
1	19%	49%

Accuracy 71%

Table 5.4: *Confusion matrix in test with smoothed labels*

Prediction \ Groundtruth	0	1
0	18%	11%
1	23%	48%

Accuracy 66%

Chapter 6

Conclusion

The goal of this project was to annotate songs using deep learning for a karaoke application. The songs representation as spectrograms, their preprocessing as well as the preprocessing of the labels are important steps.

In order to detect the singing times, different types of neural networks were used. The results obtained with CNNs improve on the ones with MLPs. Smoothing the labels also refined the results.

However, the detected time information is not sufficiently accurate to properly synchronize the lyrics with the song record.

Other types of neural networks could be used such as LSTM (Long short-term memory) networks, which are recurrent networks that are used in many speech recognition problems.

The preprocessing of the songs can be improved by using the Constant-Q transform instead of Fourier transform (this kind of transform is preferred in some music applications) and by using the Mel-frequency cepstral coefficients (MFCCs).

One of the problems identified in this project was the lack of accuracy for voice detection of the timing information given in the dataset described in section 1.2 . The division of the lyrics in syllables is maybe not the most appropriate for voice detection. It would maybe make more sense to use a division of the lyrics in entire words or sentences.

In order to use the first approach described in section 3.1, another dataset including records of songs as well as separate records of the instruments and of the voice could be used (for example the MedleyDb [7]).

Bibliography

[1] Hunziker, V.: Karaoke song generator (2016)

[2] : UltraStar. http://www.ultraguide.net [Online; accessed 05-January-2018].

[3] Simpson, A.J.R., Roma, G., Plumbley, M.D.: Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network. CoRR **abs/1504.04658** (2015)

[4] : MLDb, The Music Lyrics Database. http://www.mldb.org/ [Online; accessed 05-January-2018].

[5] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) http://www.deeplearningbook.org.

[6] : CNN filter shapes discussion for music spectrograms. http://www.jordipons.me/cnn-filter-shapes-discussion/ [Online; accessed 05-January-2018].

[7] Bittner, R.M., Salamon, J., Tierney, M., Mauch, M., Cannam, C., Bello, J.P.: Medleydb: A multitrack dataset for annotation-intensive mir research. In: ISMIR. Volume 14. (2014) 155–160

Appendix A

Appendix Chapter

A.1 Example of an IBM®Speech to Text API's test

The following lyrics give the example of the song "Don't Worry Be Happy" by Bobby McFerrin.

In bold type are the lyrics that have been recognized.

Lyrics from [4]

Here's a little song I wrote
You might want to sing it note for note,
Don't worry, be happy
In every **life we have some trouble**,
When you worry you make it double
Don't worry, be happy

Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh
(Don't worry, be happy)
Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh
(Don't worry, be happy)

Ain't got no place to lay your **head**,
Somebody came and took your **bed**,
Don't worry, be happy

The land lord say your rent is **late**,
He **may have** to litigate
Don't worry, be happy
(Look at me I'm happy)

Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be Happy)
Ooh oo-ooh oo-ooh
Here I give you my phone number
When you worry call me, I make you happy
Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh

Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be Happy)
Ooh oo-ooh oo-ooh
Here **I give you my phone number**,
When you worry call me, I make you happy
Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh

Ain't got no cash, ain't got no style
Ain't got no gal to make you smile
But don't worry, be happy
'Cause when you worry your **face**will frown
And that **will bring everybody down**,
So don't worry, be happy
Don't worry, be happy now

Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh

Don't worry, be happy
Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh
Don't worry, be happy

Now there, is **this song I wrote**,
I hope you learned it note for note
Like good little **children**,
Don't worry, be happy
Listen to what I say
In your **life** expect some trouble
When you worry you make it double
Don't worry, be happy, be happy now

Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh
Don't worry, be happy
Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Be happy)
Ooh oo-ooh oo-ooh
Don't worry, be happy

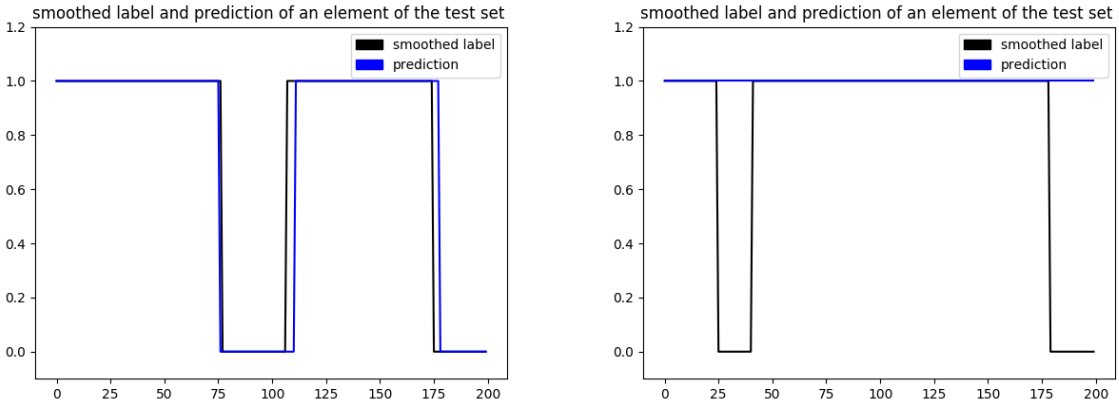
Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(Don't worry, don't worry, don't do it, be happy)
Ooh oo-ooh oo-ooh
(Put a smile on your face, **don't bring everybody** down)
Ooh, ooh ooh ooh oo-ooh ooh oo-ooh ooh ooh oo-ooh
(Don't worry)
Ooh oo-ooh ooh ooh oo-ooh
(It will soon pass, whatever it is)
Ooh oo-ooh oo-ooh
Don't worry, be happy
Ooh oo-ooh oo-ooh
I'm not worried, I'm happy

Result from the Speech to Text API

here's a little song I
you might want to sing it note for note gone
we have
do you have a **life** and we have some trouble **when you worry you** may get dial tone
may have been all the way we have
where
it happened
where have
we had
don't
and god will bless
they are **ahead**
somebody came in your **bed** gong
we had
they ran **late** and he **may have**
let
don
and then we have a
you have
I **give you my phone number when you worry** about
where
and Capital cascade god knows that
and god will gather
and
because when
faced with from
that **will bring everybody down**
so it's all
we have a dog where they have been that
so where
happy
and the
where
Dorgan well
this song I wrote
I hope you learned it
children don't worry
the a happening
they let it all out

in your **life** back from
but when you add a
top of
dawn
I have
go all the way
the happy
don't worry be
wearing
elaborate may have a
we have
don't bring everybody
going to wear
the sun passed with the
they have

A.2 Example of 2 predictions and labels in the test set



(a) (b)

Figure A.1: Example of a satisfying (a) and not satisfying (b) predictions