UNIVERSITY OF CALIFORNIA, SAN DIEGO

Separation of a Known Speaker's Voice with a Convolutional Neural Network

A dissertation submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Electrical and Computer Engineering (with a specialization in Signal and Image Processing)

by

Michael Threet

Committee in charge:

Professor Truong Nguyen, Chair
Professor William Hodgkiss
Professor Bhaskar Rao

2018

The dissertation of Michael Threet is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT OF THE THESIS

Separation of a Known Speaker's Voice with a Convolutional Neural Network

by

Michael Threet

Master of Science in Electrical and Computer Engineering (with a specialization in Signal and Image Processing)

University of California, San Diego, 2018

Professor Truong Nguyen, Chair

Source Separation (SS) refers to a problem in signal processing where two or more mixed signal sources must be separated into their individual components. While SS is a challenging problem, it may be simplified by making assumptions on the signals that are present and the methods used to mix the signals. One example of this is to limit the range of signals to human voices and to limit the total number of speakers (through either estimation or always having a set number of speakers). This paper assumes that the speech from two speakers is mixed at one microphone, with the voice of one speaker (Speaker 1) being present in all recordings. Traditional approaches to the SS problem typically involve array processing and time-frequency methods to

perform the separation. Once such example is Non-Negative Matrix Factorization (NMF), which attempts to factor a spectrogram into frequency basis vectors and time weights for each speaker. This paper will explore the use of a Convolutional Neural Network (CNN) to learn effective separation of Speaker 1's voice from a variety of other speakers and background noises. The CNN will prove to be much more effective than NMF due to the ability of the CNN to learn a representative feature space of Speaker 1's speech.

1 Introduction

General Source Separation (SS) is difficult, but with assumptions made on the signals that are present and their mixing methods, it becomes a much more tractable problem. In this paper, assumptions were made that all signals were a mixture of two human voices at a single microphone, and that the voice of a certain speaker (Speaker 1) was present in every signal. The proposed solution is a Convolutional Neural Network (CNN) that is trained to separate Speaker 1's voice from the raw audio mixture recorded at the microphone. By keeping Speaker 1's voice constant through all training steps, the CNN will learn the underlying features of Speaker 1's voice and will be able to separate Speaker 1's voice from a variety of other voices. While many SS algorithms use time-frequency processing to acquire a better relationship of the mixed signals, this CNN method is more unique in its ability to take raw audio as an input and return the separated raw audio as an output. Non-Negative Matrix Factorization (NMF) for example, factors the spectrogram of the mixed signals into matrices whose columns and rows correspond to the time-frequency and activity components of each signal present.

2 Background

2.1 The Dataset and Evaluation

The dataset used in this paper was borrowed from the Language and Speech Lab's 1st Speech Separation Challenge [las06]. The dataset consists of a 34 individual speakers uttering short sentences, along with combinations of these speakers' voices in a variety of patterns. This paper focused solely on separating Speaker 1's voice from a number of different speakers. The dataset provided both a mixed version and a clean version of Speaker 1's voice for each example, which allowed for the quality of reconstruction to be measured.

The dataset provided the mixtures at varying Signal-to-Noise Ratios (SNRs), and this

paper used the whole range. All reconstructions of Speaker 1's voice were evaluated on the Root Mean Squared Error (RMSE) between the separated voice and the original voice.

2.2 Non-Negative Matrix Factorization

Non-negative Matrix Factorization attempts to factor an $n \times m$ matrix A with all $A_{ij} \geq 0$ into two matrices: W, an $n \times k$ matrix and H, a $k \times m$ matrix. For speech separation, A is the magnitude of the spectrogram of the mixed signals and k is the estimated number of speakers. For this paper, it is given that $k = 2$ in all cases, due to the nature of the dataset. Since A is a magnitude spectrogram, it satisfies the condition that $A_{ij} \geq 0$. For a given spectrogram magnitude A, n is the number of Discrete Fourier Transform (DFT) frequency bins, and m is the number of discrete time steps that the DFT was taken at.

The objective of NMF is to minimize the Frobenius Norm between the original matrix A and the reconstructed matrix WH (i.e. $\|A - WH\|_2$). The Frobenius norm for any $n \times m$ matrix Q is defined as:

$$\|Q\|_2 = \sqrt{\sum_i \sum_j Q_{ij}^2} \tag{1}$$

Given the objective function, it is possible to solve for W and H iteratively with the following multiplicative update equations and random initializations for W and H [LS00]:

$$W \leftarrow W \circ \frac{AH'}{WHH'} \tag{2}$$

$$H \leftarrow H \circ \frac{W'A}{W'WH'} \tag{3}$$

where $A \circ B$ and $\frac{A}{B}$ represent elementwise multiplication and division respectively. With enough iterations, the Frobenius objective should converge to within a small threshold (with both the number of iterations and the threshold being tuneable hyperparameters).

NMF is useful due to its simple iterative implementation and because spectrogram magnitude matrices by definition satisfy the non-negative condition. Additionally, the matrices W and H have useful signal interpretations. The columns of W form k frequency basis vectors for the spectrogram A, while the rows of H act as time-varying activations for the frequency bases in W. The spectrograms of the k independent signals are then estimated as:

$$X_i = W_{:i}H_{i:}, i = 1, ..., k \tag{4}$$

where $W_{:i}$represents the the i^{th} column of W, and $H_{i:}$represents the the i^{th} row of H. In essence, each source is approximated as a basis spectrogram X_i for the total spectrogram A. The original time-series speech signals are then recovered by multiplying each of the individual magnitude spectrograms X_i with the phase of the spectrogram corresponding to A, and finding the inverse of the complex-valued spectrogram.

2.3 Convolutional Neural Networks

Convolutional Neural Networks generally use a combination of convolution, activation, and pooling layers to estimate a desired output signal given an input signal. CNNs will essentially learn a mapping from an input signal to an output signal, which for this paper were the mixed signal containing Speaker 1's voice and another voice, and Speaker 1's isolated voice, respectively. The benefit of a CNN is that the weights of all the convolutional filters do not need to be hand-designed. Instead, they are learned iteratively during a training process. Figure 2.1 shows the CNN used in this paper.

A convolutional layer consists of a collection of filter kernels, each with their own adjustable/learnable weights. For example, the first convolutional layer in Figure 2.1, contains 128 filter kernels, each of which is convolved with the input signal of length 2048. This in turn gives the first convolutional layer 128 outputs, each of length 2048.

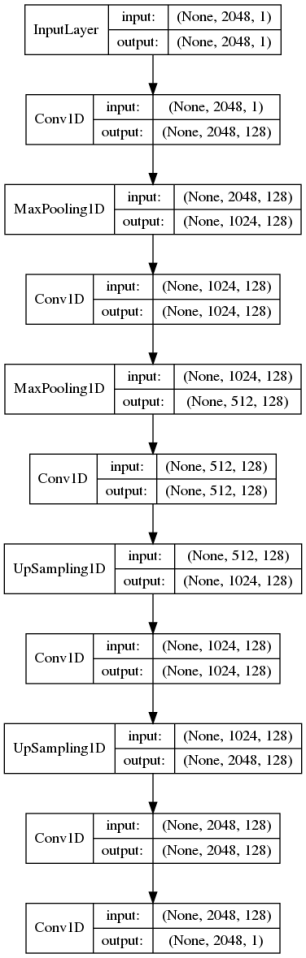


Figure 2.1: The CNN model with its layers. The graph shows the input and output size of each layer, along with the layer name/type.

Although not listed in Figure 2.1, each convolutional layer is followed by an activation layer. These activations apply non-linear functions to the convolutional layer outputs, which allow for greater feature learning by the network. sIn an absence of non-linearities], the CNN could be viewed as a linear system, which would decrease the effectiveness of adding additional layers (i.e. the additional layers do not make the overall transfer function of the filter more complex). For this paper, hyperbolic tangent non-linearities were used after each convolutional layer, except for the last layer, which had a linear activation applied (so that the output signal can take on a more linear range of values and have a uniform probability of occupying the range $[-1, 1]$).

The pooling layers downsample the signals passing through the CNN. The model in

Figure 2.1 has max-pooling layers and a downsampling rate of 2, which translates to only the maximum value between pairs of consecutive samples remaining. Ideally, the convolutional layers will amplify the samples related to Speaker 1's voice, and max-pooling will retain only those samples.

The upsampling layers ensure that the output signal has the same length as the input signal. The upsampling layers learn upsampling filters that best reconstruct Speaker 1's voice given the previous layers' outputs.

The loss used in the CNN was a combination of the RMSE between the CNN output and Speaker 1's true speech signal, and the RMSE between the DFTs of the CNN output and Speaker 1's true speech signal. This allowed for both the time and frequency components of the reconstructed speech signal to be evaluated, while still making the network loss easy to compute.

The true power of a CNN is that, given an output error, the best updates to the filter weights in the network can be computed through backpropagation. Each layer in the CNN will is used to compute the output of the network, so the partial derivatives of each layer with respect to each other and the output can be computed. In this sense, the error at the output can be propagated backwards through the CNN to update the filter weights at each layer. This allows for the CNN to learn filters without any feedback from its designer. Instead of hand-tuning a variety of filters for each layer, these filters can be learned iteratively to minimize the reconstruction error.

3 Technical Approach and Results

All experimentation in this paper was done with Python 3.6. Python's librosa library was used for all audio processing, as it provides useful tools for loading, manipulating, and saving audio files.

3.1 NMF

The NMF algorithm used in this paper was always used with no initializations for W and H, a Frobenius norm objective, a maximum of 200 iterations, and with the number of components always set to two. The input audio signal was comprised of 32000 samples, and the spectrogram was computed using a variety of segment sizes and windows (see Results), a DFT length equal to the segment size, and an overlap one-fourth the length of the segment size.

The magnitude and phase of the resulting spectrogram were taken, and NMF was applied to the spectrogram magnitude with the parameters mentioned above. The resulting separated spectrogram magnitudes were then computed using the output of NMF, and the separated audio signals were obtained by taking the inverse spectrograms of the spectrogram magnitudes multiplied by the original spectrogram's phase.

The RMSE between the estimate of Speaker 1's separated speech and Speaker 1's actual speech was then taken and recorded as the metric for reconstruction.

3.2 CNN

The architecture of the CNN in this paper is shown in Figure 2.1. The filters in all convolutional layers had a kernel size of 16 and hyperbolic tangent activations applied, except for the final convolutional layer, which had a no activation (i.e. a linear activation) applied. Unlike the NMF processing, the input to the CNN was cut into segments of length 128, with various windows applied (See the Results section). The CNN was trained on 2,000 example files, each with 32,000 samples. After training, the CNN was evaluated on 500 separate example files, each with 32,000 samples. The evaluation files were chosen randomly from the whole dataset so that a representative portion was obtained.

3.3 Results

Tables 3.1-3.4 show a summary of the results for NMF and the CNN using a variety of window methods. Note that all signals for this case had a segment size of 256 and were normalized so that maximum value was one before the RMSE was taken.

Table 3.1: Summary of reconstruction results (given in RMSE) using no window at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.21	0.17	0.13	0.09	0.07
CNN	0.13	0.09	0.06	0.03	0.01

Table 3.2: Summary of reconstruction results (given in RMSE) using a Hamming window at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.19	0.16	0.11	0.07	0.05
CNN	0.12	0.09	0.05	0.03	0.01

Table 3.3: Summary of reconstruction results (given in RMSE) using a Blackman-Harris window at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.17	0.14	0.10	0.07	0.04
CNN	0.13	0.09	0.07	0.04	0.02

Table 3.4: Summary of reconstruction results (given in RMSE) using a Kaiser window ($\beta = 16$) at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.15	0.12	0.10	0.08	0.04
CNN	0.13	0.09	0.06	0.03	0.01

Tables 3.5-3.8 contain results for NMF and the CNN using a variety of segment lengths. Note that all signals were windowed with a Kaiser window ($\beta = 16$) prior to the algorithms being applied.

Table 3.5: Summary of reconstruction results (given in RMSE) using a segment size of 128 at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.20	0.18	0.15	0.11	0.09
CNN	0.14	0.11	0.08	0.05	0.03

Table 3.6: Summary of reconstruction results (given in RMSE) using a segment size of 256 at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.15	0.12	0.10	0.08	0.04
CNN	0.13	0.09	0.06	0.03	0.01

Table 3.7: Summary of reconstruction results (given in RMSE) using a segment size of 256 at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.14	0.13	0.08	0.06	0.03
CNN	0.11	0.08	0.06	0.03	0.01

Table 3.8: Summary of reconstruction results (given in RMSE) using a segment size of 256 at a variety of SNRs

Method	-6 dB	-3 dB	0 dB	3 dB	6dB
NMF	0.13	0.11	0.08	0.05	0.03
CNN	0.10	0.07	0.06	0.03	0.01

4 Conclusion

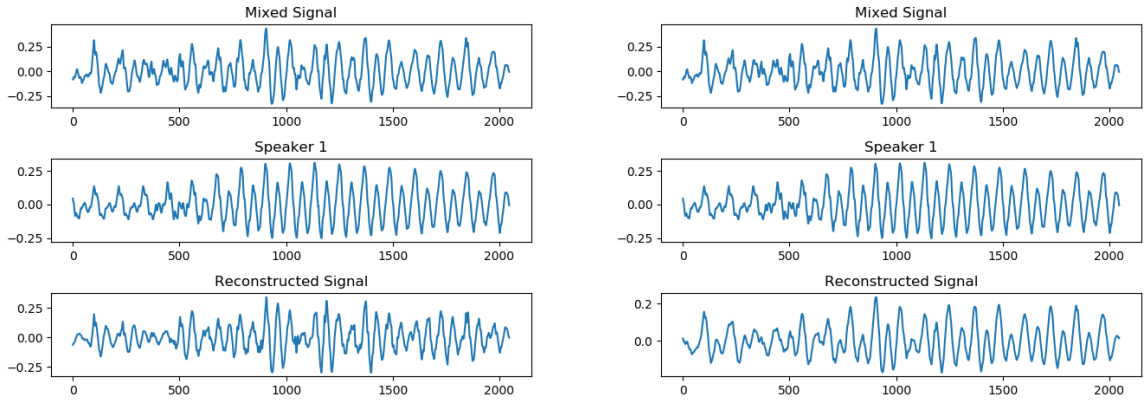
As seen in Tables 3.1-3.8, the CNN performs better in all scenarios at all SNRs. Both the NMF algorithm and the CNN perform better as the SNR increases, which makes sense due to both methods having to do less work to transform the input mixed signal into the separated signal. While NMF is greatly affected by window choice and segment size, the CNN results remained relatively consistent. The largest discrepancies in the RMSE results for the CNN were at the lower SNRs, with the higher SNR results not seeing much difference.

The CNN's resilience to window type is most likely because it operates on the raw audio input, and not the spectrogram of the input like NMF. Windowing is largely done to limit distortion in a Short-Time Fourier Transform due to the truncation of a signal, so its use on a raw audio processing algorithm is not as useful.

The CNN's resilience to segment size is due to its convolutional nature. While the input and output segment sizes change, the kernel sizes of the CNN's filters do not. Because the kernel sizes remain the same, the results will not differ drastically between inputs of different segment sizes. NMF, on the other hand, relies on the segment size to compute the spectrogram of the input signal. For NMF, changing the segment sizes changes how much "information" is in each time-indexed DFT, which leads to a larger change in reconstruction RMSE.

While the tables in the Results section show that the two methods are not too far off in their reconstruction errors, the graphical results reveal a different result. Figure 4.1 shows a reconstruction example where both NMF and the CNN perform decently well at reconstruction. While neither method is perfect, and both methods tend to look more like the mixed signal in areas of uncertainty, the output is generally smooth and natural.

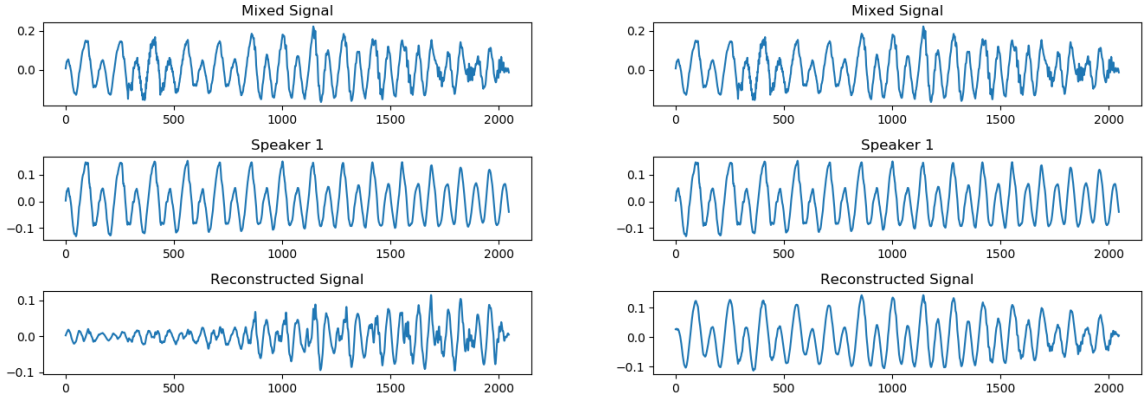
A more typical reconstruction example is shown in Figure 4.2. In most cases, NMF was not able to effectively reconstruct Speaker 1's voice in a realistic manner, even when the mixed signal was comprised mainly of only Speaker 1's voice. The CNN, on the other hand, was actually



(a) (b)

Figure 4.1: a) The results using NMF b) The results using the CNN

able to perform a small amount of denoising at the end of mixed signal.



(a) (b)

Figure 4.2: a) The results using NMF b) The results using the CNN

The better performance of the CNN in the more realistic examples indicate the the CNN performs better in total reconstruction and . The CNN has three advantages over NMF: it can use nonlinear methods, it can use a variety of loss functions, and it can use a large amount of training examples to learn more robust reconstructions.

The CNN applies nonlinear activations after each convolutional layer, which allows for nonlinear approximations to be made. Conversely, NMF relies on a linear matrix factorization, which places limitations on the reconstruction quality of Speaker 1's voice.

The CNN loss function for this experiment was a combination of the difference in the reconstruction and Speaker 1's original speech in both the time and frequency domains. This allows for better phase reconstruction in the reconstructed signal. NMF, meanwhile, is only able to use the spectrograms of the original and reconstructed speech signals, which ignores any phase information, and does not consider time-domain accuracy.

Finally, the CNN is able to use the thousands of past examples it has seen to influence the reconstruction of future signals. The CNN can learn common patterns or features in Speaker 1's voice and reconstruct them more easily as training time increases. NMF, on the other hand, does not keep track of any past examples, so only the current input is weighted for consideration. In other words, NMF cannot learn any patterns or features of Speaker 1's voice.

Bibliography

[las06] Ikerbasque and university of the basque country language and speech lab's 1st speech separation challenge. http://laslab.org/SpeechSeparationChallenge/, 2006.

[LS00] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *In NIPS*, pages 556–562. MIT Press, 2000.