Speech Enhancement Using a Deep Convolutional Network Trained by Ideal Binary Mask and Cross Entropy

Mhamdi Mariem **Asaadi Boubaker**

Professor Laurent Girin **Professor Pascal Perrier**

Abstract

Separating speech from Babble noise is one of the toughest problems in signal processing. Deep Learning Techniques have been used lately to tackle this issue, using supervised learning. One of these techniques is the Ideal Binary Mask (IBM) which consists of designing a binary mask which then will be applied on a signal's spectrogram in order to eliminate the regions that are noisy. In this work, we sought to separate noise from signal using an Encoder Decoder based approach, with a convolutional network to extract the main features form the signal's spectrogram and a Fully Connected Network in order to map the features back to an IBM.

This work is based on a paper released in 2018 which tackles the problem of Singing Voice Separation Using Deep CNNs. (Lin Kin Wah Edward & al, 2018)

1. Introduction

Bubble noise is a background noise that degrades the quality of speech in a signal. Removing this type of noise from a signal remains a challenging task. In the previous years, and with the rise of Deep Learning techniques, researchers started applying these techniques on this problem using Fully CNNs, RNNs or BLSTMs. In this work we opted for using an atypical architecture which uses both CNNs and Fully Connected Layers. This architecture, which combines the two approaches, has proven to be very effective when applied on image classification, and also on Singing Voice Separation (SVS) (Lin Kin Wah Edward & al, 2018).

Inspired by the work of Kin Wah Edward Lin et al. on Signing Voice Separation (SVS), where the authors designed a network to separate the voice from its music accompaniment and which gave good results, we used the same approach as the one used in their paper, although the architecture of their network is not exactly the same as ours.

Unlike the traditional models that target the denoised spectrogram directly, our model targets a binary mask. Our model takes the noisy signal's spectrogram as an input and outputs the Ideal Binary Mask (IBM) of this spectrogram, this IBM is then applied to the input spectrogram in order to obtain the denoised spectrogram which then is converted back to an audio signal. This approach of using IBM was shown to be very effective on several application among which is Singing Voice Separation.

In the next section, a detailed description of the steps followed in order to format the data set that was used in the training step and a formal definition of the Ideal Binary Mask (IBM), followed by a clear explanation of the cost function used in this model which is the Binary Cross Entropy function. We then explain the preprocessing of the data before feeding it to the network, then the network architecture followed by the postprocessing of the output data in order to obtain the denoised spectrogram. Finally, we describe the training methodology and the results of our model.

2. Preprocessing the dataset

We have a dataset of 2050 signal of approximately 3 seconds each, and a 5 minutes long Babble noise signal. First, we load the whole wave files into one folder, and then we noise every original signal in the dataset by adding excerpts from the Babble signal, of the same size as the original signal, to the original

signal. When adding the noise to the signal we multiply the noise by a coefficient α in order to control the overall signal to noise ratio (SNR). The noisy signal is expressed as,

$$Sn(t) = S(t) + \alpha * n(t)$$

Where Sn(t) is the noisy signal, S(t) the original signal and n(t) represents noise.

We want a noisy signal with an SNR of 0 dB, so,

$$10 * \log\left(\frac{Ps}{\alpha^2 * Pn}\right) = 0$$

Where Ps is the power of the original signal S(t), and $\alpha^2 * Pn$ is the power of the noise added to S(t).

So,

$$\alpha = \sqrt{\frac{Ps}{Pn}}$$

And for an SNR of 6 dB,

$$\alpha = \sqrt{\frac{Ps}{3.98 * Pn}}$$

We create two datasets, one with an SNR of 0 dB and the other with an SNR of 6 dB.

Then, we compute the spectrograms of the original signals, the noisy signals and the noise, using the Short-Time Fourier Transform. As the average length of a stationary sound is about 30ms, we chose a size of the window of 32ms which corresponds to W = 512 samples given that the sampling frequency is of 16 000 Hz. We also chose a hop size of 16ms which corresponds to H = 256 samples. And an FFT with a length of N = 512 samples. We then compute the IBM for every noisy signal, which represent the targeted output of our network.

We use the IBM as the targeted label in this work, instead of targeting the signal's denoised spectrogram directly. IBM can be defined as follows. Let FxT be the size of a spectrogram magnitude matrix X, whereby F denotes the numbers of frequency bins in the spectrogram, and T the number of time bins. X is the spectrogram of the noisy signal; it can be expressed as follows:

$$X(n,t) = S(n,t) + Sn(n,t)$$

S(n,t) : the magnitude spectrogram of the signal at the temporal bin t and frequency bin n.
Sn(n,t) : the magnitude spectrogram of noise at the temporal bin t and frequency bin n.

The IBM of the signal is calculated as,

$$B[n,t] = \begin{cases} 1 & if\ S(n,t) > Sn(n,t) \\ 0 & if\ S(n,t) \leq Sn(n,t) \end{cases}$$

Whereby, B is the IBM, $t \in [0,T]$ is the time index and $n \in [0,F]$ is the frequency index.

We then divide the dataset into three parts, one part for training one for validation and another for testing. The proportions are respectively 86% for training, 7% for validation and 7% for testing.

3. The cost function

We are targeting a binary output with 257 neurons, where the outcome is a value between 0 and 1, so the binary cross entropy cost function, L, is the best suited for our task.

$$L = -yi \,.\log(\hat{yi}) - (1 - yi) \,.\log(1 - \hat{yi})$$

Where yi is the true label and $\hat{yi}$ is the estimated label.

We can see that if the estimated label $\hat{yi} = 0$ while the true label $yi = 1$ the cost goes to infinity, and it is the same if the estimated label $\hat{yi} = 1$ while the true label $yi = 0$. So, the only way to minimize the cost function L is by getting the predicted label as close as possible to the true label.

The cost function is expressed using the IBM mask as,

$$L = \; B[n,t] \,.\log(\hat{yi}[n,t]) + (1 - B[n,t]) \,.\; \log\,(1 - \hat{yi}[n,t])$$

4. The network's design

4.1. Preprocessing

Our network takes as an input of 7 temporal bins from the input spectrogram and computes the estimated IBM mask of the middle bin as illustrated in the figure below.

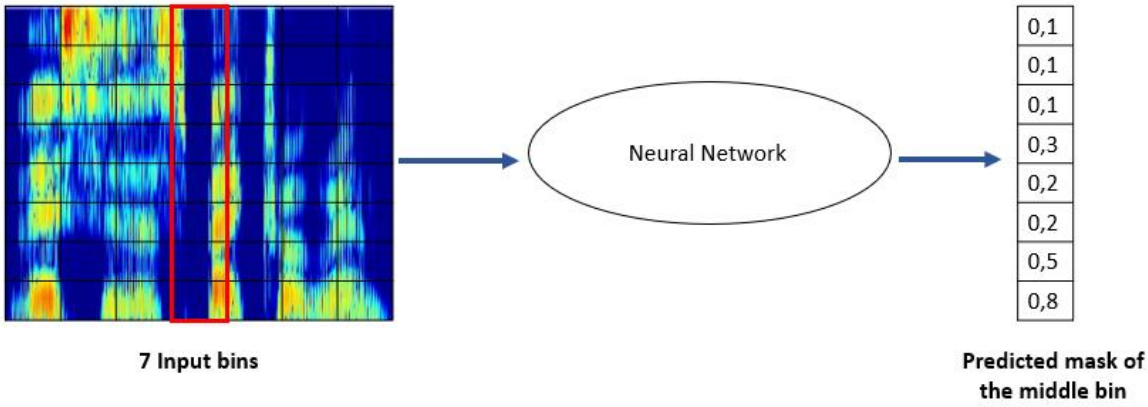


Figure 1: Illustration of the input and the output of the model.

The predicted mask contains values ranging from 0 to 1, these values represent the probability that a certain time-frequency bin contains more speech than noise.
In order to predict the mask of the whole spectrogram, firstly we take 7 temporal bins from the beginning of the spectrogram, we predict the mask of the middle bin, then we stride by 1 to the right and then we predict the next bin's IBM mask. We won't be able to predict the first and last three bin of the signal because we need 7 bins at each input to the network. So, the output signal won't have the same duration as the input signal. The difference between the duration of the two signal is 6 temporal bins.

We chose an input size of 7 temporal bins in order to predict the middle bin's mask, this is because a bin is often dependent on its neighbors, that is why we added 3 bins on the right and on the left of the targeted bin. This improves the quality of the predicted mask, as it takes into account the neighboring bins of the bin whom we seek to predict the IBM mask.

By subdividing each input spectrograms into 7 temporal bins with a stride of one each, we get that the overall number of training data is 349,647 input excerpts of 7 time-bins, the overall number of validation data is 21,548, and the overall number of test data is 21,297.

4.2. The network's Architecture

The adopted architecture performs a series of convolutions of the input spectrogram bins in order to extract the main features, and then a fully connected layer maps these features to the targeted output, which is the middle bin's IBM.

Before feeding the spectrogram to the network, we normalize it to values between 0 and 1. This is a commonly used technique in order to speed the model's convergence.

Table 1: Network Architecture of the proposed model along with the number of parameters.

Layer	**Configuration**	**Number of trainable parameters**
Input	Input size of 7x257	N/A
Convolution	32 filter, size 3x3, Stride 1 Same zero padding, ReLu	(3x3) x 32+32 = 320
Convolution	64 filter, size 3x3, Stride 1 Same zero padding, ReLu	(3x3) x 32 x 64 + 64 = 18,490
Max Pooling	Non overlap (1x3) max pooling	N/A
Convolution	128 filter, size 3x3, Stride 1 Same zero padding, ReLu	(3x3) x 64x 128 +128 = 73,856
Max pooling	Non overlap (1x3) max pooling	N/A
Dropout	With probability 0.2	N/A
Flatten Layer	3712 neurons	N/A
Fully Connected	1024 neurons, ReLu	1024x3712+1024 = 3,802,112
Dropout	With probability 0.2	N/A
Fully Connected	2048 neurons, ReLu	2048x1024+2048 = 2,099,200
Output	257 neurons, Sigmoid function IBM labels as the target labels	257x2048+257 = 526,593
Objective function	Binary Cross Entropy	Total : 6,520,755

To improve the network's architecture, random initialization is commonly used. We opted for the Xavier uniform initializer which is named *glorot_uniform* in Keras library.

As we have only 7 temporal bins in the input, we did max pooling only on the frequency dimension and not the temporal dimension. We used a max pooling of 3 which is the value that seemed to give the best results. We also chose a kernel size of 3 which is the commonly used value.

The output of this network is a soft mask with values ranging from the minimum value 0 to the maximum value 1.

Although the number of parameters of the network might seem huge, but compared to the number of time-frequency bins in the training data, it is totally normal to obtain this number. The number of time-frequency bins in the training data is the number of temporal excerpts used in training multiplied by the size an excerpt, which is 7*257. So, the overall input training data is:

$$\textbf{number of input data bins} = 349{,}647 * 7 * 257 \approx 629 * 10^9$$

4.3. Postprocessing

The goal of signal denoising it to get an isolated signal containing speech. We therefore need, given the spectrogram of the noisy signal and the soft mask predicted by the model, we need to obtain an audio

signal. In order to do this, we multiply every predicted IBM mask by the corresponding input bin in the input spectrogram, and arrange the result bins in a matrix. This matrix represents the spectrogram of the denoised signal. We then convert this spectrogram back to an audio file using the Inverse Short-Time Fourier Transform by using the ISTFT function of python. The result spectrogram can be expressed formally as,

$$\hat{S}[n,t] = S[n,t] \,.\, \hat{y}[n,t]$$

Whereby, $\hat{S}[n,t]$ represents the denoised time-frequency bin, $S[n,t]$ the input time-frequency bin, and $\hat{y}[n,t]$ the predicted mask for the corresponding time-frequency bin.

To further improve the quality of separating noise from speech, we set a threshold θ in order to obtain a binary mask instead of a soft one. This is done as follows,

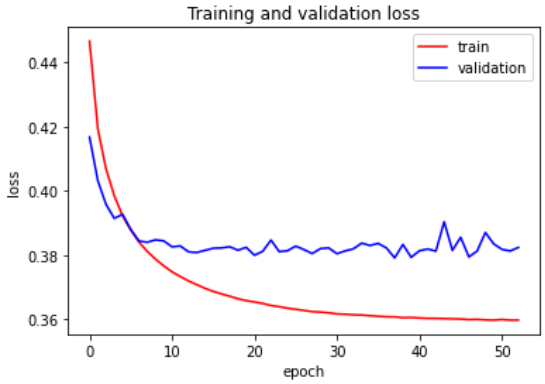
$$\begin{cases} \hat{y}[n,t] = 1 & if\ \hat{y}[n,t] > \theta \\ \hat{y}[n,t] = 0 & if\ \hat{y}[n,t] \leq \theta \end{cases}$$

After several experiments with several signals we found that θ = 0.3 seems to perform good denoising, but we can always tune the value of θ for every signal if this value doesn't give good results.

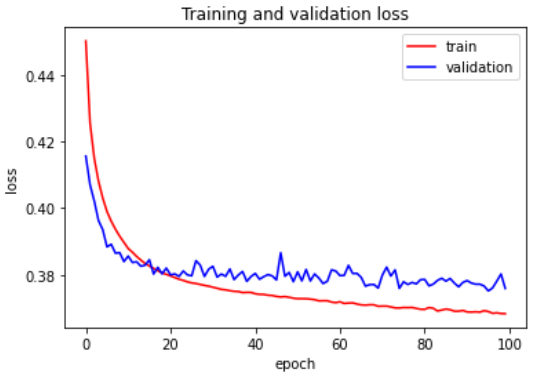
5. Training

The training data was created by dividing every spectrogram into several excerpts of 7 temporal bins each, with a hop of 1 temporal bin and an overlap of 6 temporal bins between two neighboring excerpts. Then, the data is shuffled randomly.

We trained the model using the two datasets created in **2**, the dataset with 0 dB SNR and 6 dB SNR.



(a) The cost function of the model trained with the 0 dB SNR dataset.



(b) The cost function of the model trained with the 6 dB SNR dataset

Figure 2 : Evolution of the cross-entropy cost function for the 0 dB SNR dataset and the 6 dB SNR dataset.

We trained the two models while specifying to the algorithm to end training if the validation binary accuracy doesn't surpass its best score for the next 15 coming epochs. We reached 53 epochs with the 0dB SNR dataset and 100 epochs with the 6dB SNR dataset.

Tested on the test data, the model that was trained on the 0dB SNR dataset reached a value of 0.391 for its binary cross-entropy, and the model trained on 6 dB reached a binary cross-entropy of 0.384.

6. Results

Our model gives good results when applied to new audio signals. The hearing experience is much better after denoising using our model.

We see, in the figure below, the spectrograms of the three signals, the original signal, the noisy signal and the denoised signal by the model. We can observe that the noisy signal contains more energy in low frequencies, more than the original signal, especially below 4 KHz. This low frequency energy corresponds to the energy of Babble noise. After denoising the noisy signal using our model, we see clearly that the denoised spectrogram is much cleaner and is much closer to the spectrogram of the original signal.

As mentioned above, the denoised spectrogram doesn't have the same time length as the original one, this because we strip 3 temporal bins in the beginning the end of every spectrogram and the reason is clearly explained in **4.1**.

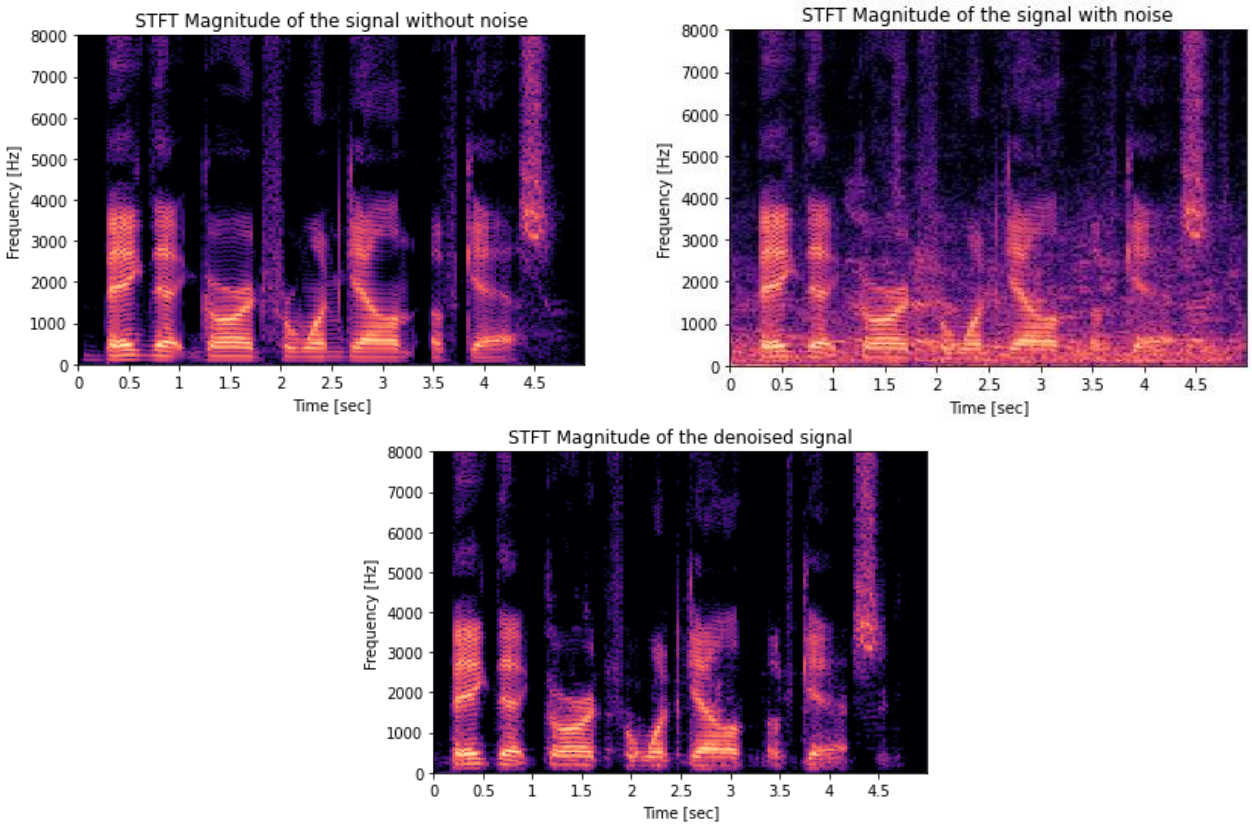


Figure 3: Magnitude spectrograms of the original signal, the noisy signal and the denoised signal.

In order to measure the similarity between the original signal and the denoised signal, we can use another neural network to measure the similarity between the two audio signals, or use a Siamese neural network applied on the spectrograms of the two audio signals, in order to measure their similarity.

Unfortunately, we haven't measured the quality of our denoised signal using a more objective metric due to the lack of time.

7. Conclusion

In this work we have been able to design and train a model that has good performance in eliminating Babble noise form an audio signal. Our model has clearly a large number of trainable parameters, about 6.5 million, but due to complexity of the task of denoising signals, it is totally normal to require this number of parameters, given also that we used a fully connected layers as a decoder.

In future work, it would be interesting to measure the performance of our network using an objective metric of similarity of the original and the denoised signal. It would also be promising to try to reduce the number of parameters, should that be possible, without hurting the performance. Also, comparing the performance of our model as opposed to other models that use FCNNs (Park & Lee, 2017).

8. References

Lin Kin Wah Edward, & al. (2018). *Singing Voice Separation Using a Deep Convolutional Neural Network Trained by Ideal Binary Mask and Cross Entropy.* arXiv.

Park, S. R., & Lee, J. (2017). *A Fully Convolutional Neural Network for Speech Enhancement.* arXiv.