Deep Learning for Denoising of Audio Signals

A project report Submitted by

Kumar Alabhya	17EC222
Anirudh Kanfade	17EC255
Abhishek Choudhary	17EC103

Under the guidance of
Dr. Shyamlal
in partial fulfilment of the requirements for the award of the degree of
BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575025

Contents

List of Figures

List of Tables

1 Abstract

Existing deep learning-based speech denoising architectures require clean speech signals to be available for training. This paper aims to develop a deep learning-based approach to improve speech denoising in real-world audio surroundings by not requiring the availability of clean speech signals in a self-supervised manner. A fully convolutional neural network is trained by using two noisy realizations of the same speech signal, one used as the input and the other as the output of the network.Here we are trying to show the superiority of the developed deep speech denoising approach over the conventional supervised deep speech denoising approach based on commonly used performance metrics and also based on actual field-testing outcomes.

2 Introduction

Denoising speech signals has been a long standing problem. Decades of works showed feasible solutions which estimated the noise model and used it to recover noise-deducted speech. Given an input noisy signal, we want to filter out the unwanted noise without degrading the signal we want. As we see in our daily life or now a days in online classes someone talking in a video conference while a piece of music is playing or some kind of other noises in the background hindering the speech quality. In this situation, a speech denoising system can be very useful in removing the background noise in order to improve quality of the speech signal. Besides many other use cases, this application is especially important for video and audio conferences where noise can significantly decrease speech quality.

2.1 Motivation

In hearing aids, the presence of babble noise degrades hearing intelligibility of human speech greatly.

2.2 Problem Statement

To design and implement robust deep learning model for Denoising of audio Signals.

2.3 Objectives

1. Simulate the audio denoising using UNet encoder decoder deep learning architecture
2. Generate denoised audio out of predicted spectrogram and visualize the sound
3. Compare the signal to noise ratio for the noisy signal as well as predicted denoised signal

2.4 Organization of Report

In this report we have demonstrated denoising of speech using UNet encoder decoder deep learning architecture. The report is divided into sections or chapter. Chapter 3 describes the Methodology , which include details about dataset, preprocessing, deep learning architecture employed and training parameters. Chapter 4 is Results, which describes the audio set on which we tested our method and results we obtained

3 Methodology

3.1 Dataset

3.1.1 Dataset Overview

We have made use of the following dataset for our project

1. The Mozilla common voice dataset
2. The UrbanSound 8K dataset

The Mozilla common voice dataset is used as the clean data while the Mozilla common voice sunds are added to the UrbanSound8k dataset which act as the noisy data for our case

3.1.2 Mozilla common voice Data

Common voice dataset has more than 200K audio in their training set and more than 20K images in their test set. For the sake of simplicity and owing to the limited memory and training resource we have made use a part of the dataset. So 500 speech sounds are extracted from the training images to act as our training label. For validation purpose we have made use of 40 sounds from original test dataset and 7 sounds from the same dataset are used for testing purpose

3.1.3 UrbanSound dataset

UrbanSound dataset contain 10 folders with around 800 to 900 sounds in each of the folder. For training purpose we need to extract same number of samples as we extracted from mozilla dataset which is 500. So we have extracted first 50 sounds from all the folders. Similarily for validation we have extracted next 4 sound from each fo the folder. for test we have taken one sound each from the first 7 folder

3.1.4 Final samples

The clean mozilla sound act as label and the addition of mozilla and Urban Sound is the noisy dataset that we have. so we have the following number of samples

1. Training sample : 500
2. Validation sample : 40
3. Test sample : 7

3.2 Data preprocessing

We have made use of UNet encoder decoder architecture for which we required a fixed shape spectrogram to pass through the network without any issue. The size of the input to the architecture was set to 512x512, giving an output also of size 512x512. Therefore to get the spectrogram of shape 512x512 for both noisy signal (data) and clean signal (label) we followed the following preprocessing method

1. Both clean file and noisy file were read using load method of librosa using the sample rate of 16000
2. First of all both the read files were normalized about zero mean and unit standard deviation
3. Any empty portion in the clean audio before the start of original speech were removed and the clean signal was truncated to a array size of 63875(which is around 4sec of audio).
4. We now took the noise sound that had read and truncated it to the same size as that of the clean sound
5. With both clean and noise ready , we added noise to the clean sound by using the following equation

$$noisy = clean + (\frac{Power_{clean}}{Power_{noise}}) * noise \tag{1}$$

6. Next we took the calculated the spectrogram for both noisy and clean sound as data and label for our network. We made use of librosa short time fourier transform library for calculating the spectrogram. Since we required the size of spectrogam to be 512x512 with given audio of size 63875, we made use of the following parameters for the stft function
   (a) windowLength = 500
   (b) overlap = 125
   (c) ffTLength = 500
   (d) numFeatures = ffTLength
   (e) window = hamming window
7. The spectrogram is then converted to decibel from amplitude and then normalized and the same procedure was followed for all sample in train dataset. The samples were then converted into tensor and passed through the neural network

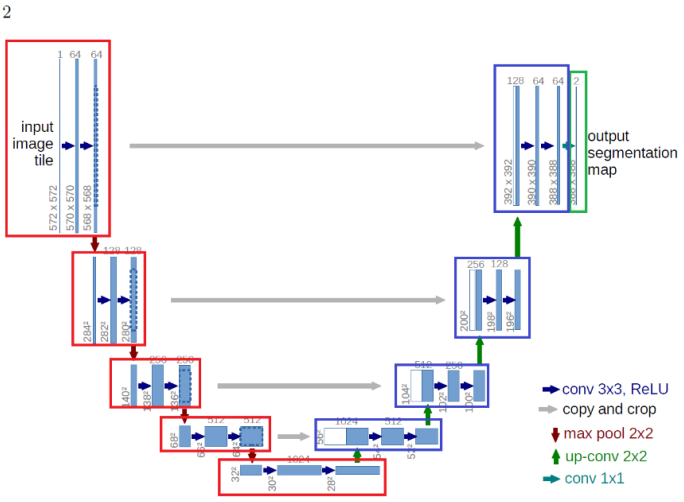

Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Figure 1: UNet Architecture

3.3 UNet Encoder Decoder Architecture

For the training we made use of UNet encoder decoder architecture .This architecture consists of the following blocks

1. **Encoder block** : The encoder block is used to encode the features in an image. The encoder block which UNet make use of consists of the 4 blocks each consisting of two convolution layer. Convolution Layers are followed by batch normalization and ReLU activation function. The blocks consists of the following number of layers:

   (a) block 1 : 64 filters

   (b) block 2 : 128 filters

   (c) block 3 : 256 filters

   (d) block 4 : 512 filters

   Each convolution block is followed by a Max Pooling layer with a pool size of 2x2 and stride of 2.

2. **bottleneck** : The bottleneck layer consists of 2 cascaded CNN block for filter size 1024, without any max pooling layer after that.

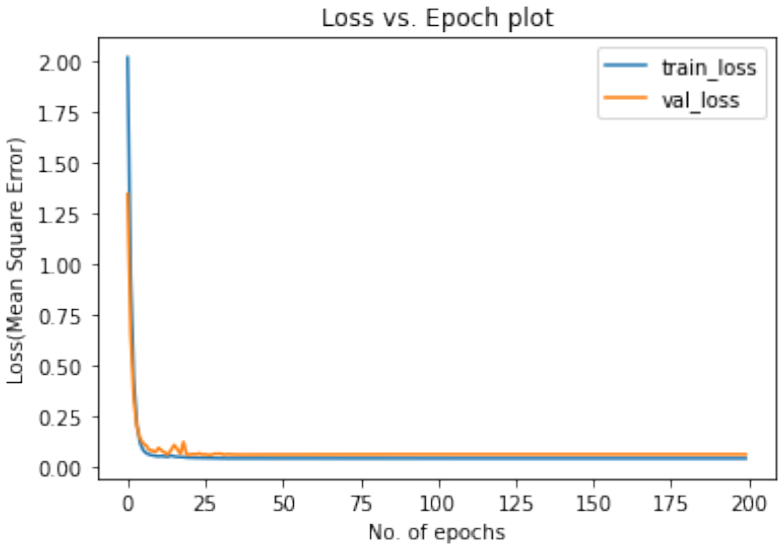

Figure 2: Loss curve

3. **Decoder Block** : Decoder block comes after bottleneck. This first use Transpose Convolution to Unpool the layer and then concatenates this layer with the layer of its corresponding encoder block through **skip connections**. This is then followed by two convolution block, with batch normalization and ReLu activations

In the end we make use of simple 1x1 CNN to get a shape of the output as 512x512. This do not use any activation here as we do not want to truncate the values between 0 and 1. The overview of the architecture is as shown in the figure 1

3.4 Training and implementation details

The following architectural component were used for training

1. Initializer : Xavier initializer for weight and zero initializer for bias
2. Optimizer : Adam Optimizer
3. Loss : Mean Square Error
4. learning rate : learning rate was set to $10^{-}3$, which was reduced by a factor of 5 on no improvement on validation for 5 epochs

The deep learning architecture was trained for 200 epochs. The graph for the loss function is as shown in figure 2

3.5 Data Post processing to get audio back

To get the audio back from the spectrogram we have to follow the following processes

1. denormalize the spectrogram by multiplying it with original standard deviation and adding the original mean to it
2. use decibel to amplitude converter to convert the signal from decibel back to amplitude
3. multiply with the exponential of the phase of original signal to get complex stft
4. apply inverse stft with same parameters as used in data pre processing to extract the audio

This would be predicted audio we would be getting

4 Simulation Results and Discussion

4.1 Test Data overview

We made use of 3 sample clean and noise signals to test our model. The noisy signal was created in the same way as we created the training noisy sample.

We kept our dataset small so that we can hear each and every generated audio and how our model is behaving on them . Though our test data set is small , it is very challenging with two out of three noise sample having negative SNR which means that noise is dominant over the speech signal

4.2 Comparing Predicted spectrogram with original ones

So here is the visualization of clean ,noisy and predicted denoised spectrogram for all the data in test data set

4.2.1 Test Sample 1

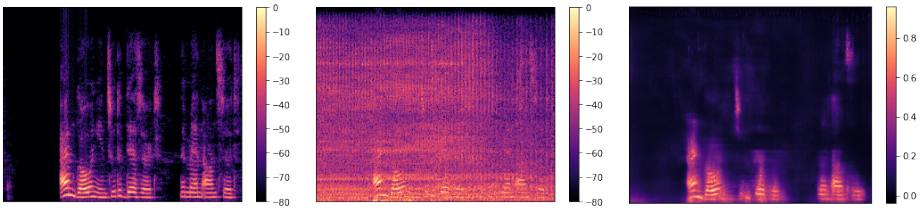

Figure 3: Spectrogram for clean signal (left), noisy signal (middle), predicted signal(right) for sample 1

4.2.2 Test Sample 2

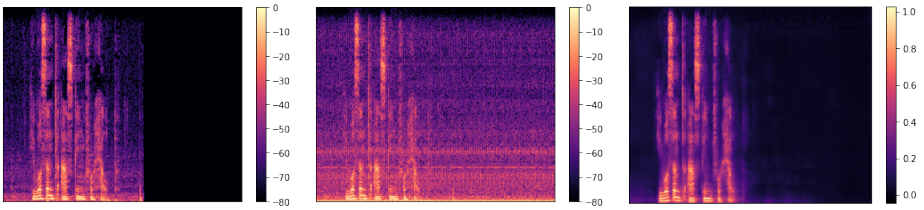

Figure 4: Spectrogram for clean signal (left), noisy signal (middle), predicted signal(right) for sample 2

4.2.3 Test Sample 3

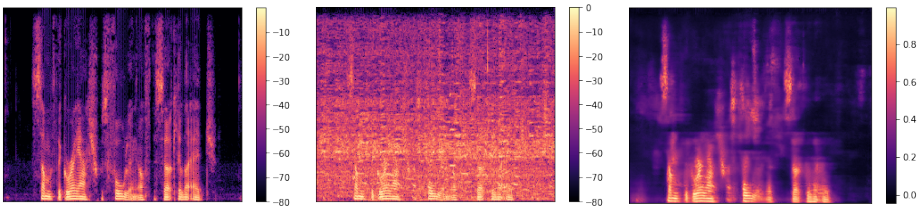

Figure 5: Spectrogram for clean signal (left), noisy signal (middle), predicted signal(right) for sample3

4.3 Comparing Predicted regenerated audio with original ones

So here is the visualization of clean ,noisy and predicted denoised audio plots for all the data in test data set

4.3.1 Test Sample 1

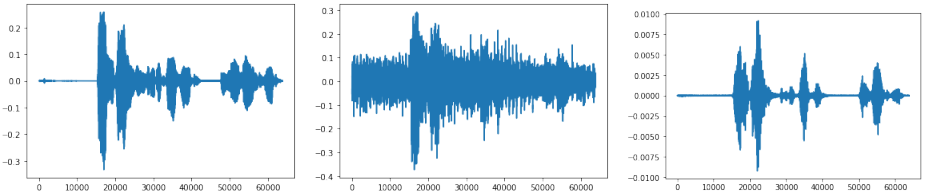

Figure 6: Plot for clean signal (left), noisy signal (middle), predicted signal(right) for sample 1

4.3.2 Test Sample 2

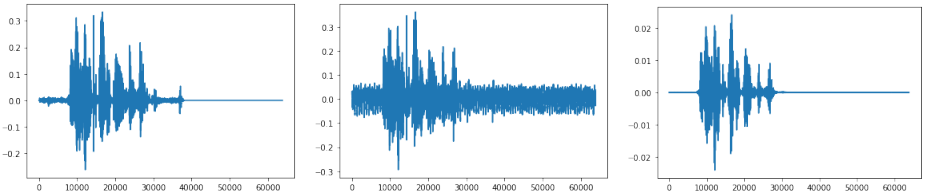

Figure 7: Plot for clean signal (left), noisy signal (middle), predicted signal(right) for sample 2

4.3.3 Test Sample 3

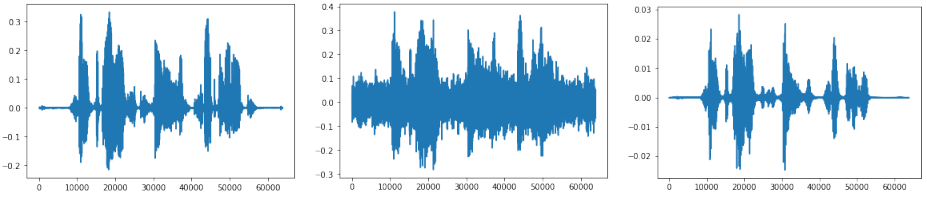

Figure 8: Plot for clean signal (left), noisy signal (middle), predicted signal(right) for sample3

4.4 Metric calculation (SNR)

To evaluate our model on the signal denoising task we made use of signal to noise ratio metric or the SNR. The equation for calculating SNR is as shown below

$$SNR = 10 * log10\left(\frac{P_{sig}}{P_{noise}}\right) \tag{2}$$

where P_{sig} corresponds to the power of pure signal and P_{noise} corresponds to power of pure noise

we calculated the SNR with respect to the noisy signal and clean signal and then with predicted signal and clean signal. Both the SNR for all the three cases are as shown in the table

Table 1: SNR before and afer denoising on our test cases

Test Samples	SNR (between noisy and clean signal	SNR (between predicted clean and clean signal
Test case 1	-3.882 e-06	0.137
Test case 2	5.177 e-07	0.451
Test case 3	-6.730 e-06	0.499

5 Conclusion

We saw from the table 1 we see that the signal to noise ratio increased by a large amount. Even our algorithm worked great for the case where there was negative SNR which means that noise was highly dominant over speech signal.

References

1. Se Rim Park, Jinwon Lee, A Fully Convolutional Neural Network for Speech Enhancement, Machine Learning, 2016, arXiv:1609.07132

2. Olaf Ronneberger, Philipp Fischer, Thomas Brox, UNet Comvolutional network for biomedical image segmentation, Computer Vision and Pattern Recognition, 2015, arXiv:1505.04597

3. Nasser Kehtarnavaz, Short-Time Fourier Transform, in Digital Signal Processing System Design (Second Edition), 2008,