Using Convolutional Neural Networks on Spectrograms to Perform Speech Enhancement

© 2020 Daniel Nissani (dn298@cornell.edu)

1 Abstract

A big problem in hearing aids, programs, such as zoom, and in telecommunications is noise. While speaking, noise can either muddle speech or make it impossible to hear the signal that one wants to hear. Inspired by the spectrogram project we did in class, I will build a convolutional neural network to process spectrograms to remove noise from speech signals.

2 Introduction

Speech denoising (also known as speech enhancement) is an extremely complex problem. Two methods, at present, create start-of-the-art results: processing speech signals in raw wavelet forms or processing speech signals as spectrograms. Although some success has come from processing speech signals in raw wavelet forms, primarily in the forms of model size and speed [1, 5], there are some advantages to using the spectrogram as data. Treating a spectrogram as an image, two dimensional convolutional neural networks can create features that researchers cannot create [2]. Moreover, taking advantage of new convolutional neural network architectures such as Redundant Convolutional Encoder-Decoder Networks (R-CED)[3], we address the concerns that spectrograms are not like typical images, since the pixels possess more meaning than in regular image tasks, such as temporal and relational information [7].

The goal of this project is to implement at least one convolutional neural network structure to remove noise from spectrograms. Like Daitan, I plan to use tools, such as PyTorch and LibRosa, to create my neural network architecure and digital signal processing. Specifically, I plan to replicate the STDFT implemented in one of our assignments to create spectrograms for our signal and then removing noise using a convolutional neural network. If time permits, I plan to implement R-CED and experiment with different windows (outside of the Hamm window). In terms of data collection, two avenues can be taken. Either look for a dataset or figure out how to create a dataset, much like what has been done in Daitan.

3 Dataset

The datasets used for this project were the Mozilla Common Voice dataset and the Urban8k dataset, which has 10 classes of background noise. The mozilla common voice dataset acted as the clean signal, and the Ubran8k dataset acted as the noise. To train the neural network, 600 points were sampled from the mozilla common voice dataset, and background noise called ”playground” was added to each datapoint from the mozilla common voice dataset. Moreover, I also held in memory each common voice data point as a clean sample for learning. 200 data points were sampled for validation in a similar way.

For data preparation, all data was put through an STDCT and each noisy input consisted of a noisy input window that matched the window for a clean input, concatenated with two previous noisy input windows. For the first two noisy input windows, they were contatenated with zeros.

4 Model

To describe the model, I first describe the STDCT with an MLT window implemented for this project. Unlike past research, I use the STDCT instead of the STDFT because it guarantees real value outputs. The MLT window matrix is defined as:

$$W[i,i] = sin(\pi * (\frac{i+0.5}{N}))$$

where W is an N x N matrix. In this case, $N = 128$ and we computed shifts of $N/2$ for the transform. I used the DCT II operation to before our DCT, which can be found in the scipy documentation. For visual purposes, I will call it D. Thus, our transform was the following:

$$DWs = \hat{s}$$

where s is a signal.

Three neural networks were trained for comparison. Each were trained for 200 epochs, used an Adam optimizer, and used an MSE loss function. All networks had a 384 dimension input and a 128 dimension output. The differences were the number of layers, the activation functions used on each layer, and the number of neurons for each layer. The two layer network had a tanh activation function on the first layer which had dimensions 384 by 256, and then the output came out of a linear layer which had dimensions 256 by 128. The three layer network had an selu activation function on the fist layer which had dimensions 384 by 320, a gelu activation function on the second layer whcih had dimensions 320 by 192, and a linear output layer with dimensions 192 x 128. The four layer network had an elu activation function on the first layer which had dimensions 384 by 320, a selu activation function on the second layer which had dimensions 320 x 256, a gelu activation function the third layer which had dimensions 256 x 192, and finally a linear output layer with dimensions 192 x 128. All layers were linear layers. These activation functions were chosen because the inputs to the neural network were quite small, and so I wanted something robust to small input sizes, which ReLUs are not.

Once the networks were trained, each data point was transformed by the network and then inverted back into the time domain via the inverse STDCT, which is defined as:

$$WD^{-1}\hat{s} = s$$

5 Results

Using the validation set to get results, we find three measurements to help us verify our results. First, we take the average MSE of each noisy input and each clean input. This works as our ground truth of how far away on average each noisy input is from a clean input. Then we find the average MSE of each noisy input and output of the neural network. This is primarily a comparative measure, that will tell us if we have gotten closer to the clean input or not. If the average MSE is smaller for the noisy input and the output than the clean signal and the output, then that is an indication that the model has done poorly. The last measurement is the average MSE of the clean signal and the output. Our results are shown in the table below:

	AMSE Noisy vs Clean	AMSE Noisy vs Output	AMSE Clean vs Output
Two Layer Network	00.00426	0.00239	0.00201
Three Layer Network	0.00426	0.00215	0.00173
Four Layer Network	0.00426	0.00225	0.00163

AMSE stands for the average MSE.

Note that the four layer network did best in terms of MSE. Although I am not able to provide the output of each validation point, a sample is provided. You'll note a ringing that is contained in the outputs of the network, with the two layer network performing worse than the three layer network and the four layer network. Worse here means that the background noise is not as muted. Although the background noise was not completely removed, the three layer network and four layer network did manage to dampen the background noise and amplify the clean voice signal.

6 Next Steps

Although many solutions presented involved using the spectrogram as the input to a neural network, inverting a spectrogram perfectly is not possible. It would be interesting to explore ways to get approximations of a spectrogram inversion as to try to use the spectrogram as an image input to a convolutional neural network. Moreover, the temporal structure of the data, particularly given the windows of the STDCT, lead me to believe an LSTM or an encoder-decoder would work quite well. Park Lee already implemented this idea with great results with the R-CED network. Now that I have a working version of a simple feed forward network, using the domain knowledge to build something more complex could yield much better results.

References

[1] Germain, François Chen, Qifeng Koltun, Vladlen. (2019). Speech Denoising with Deep Feature Losses. 2723-2727. 10.21437/Interspeech.2019-1924.

[2] Wyse, Lonce. (2017). Audio Spectrogram Representations for Processing with Convolutional Neural Networks.

[3] Park, Serim Lee, Jinwon. (2016). A Fully Convolutional Neural Network for Speech Enhancement.

[4] Doerfler, Monika Grill, Thomas. (2017). Inside the Spectrogram: Convolutional Neural Networks in Audio Processing.. 10.1109/SAMPTA.2017.8024472.

[5] Kiranyaz, Serkan Ince, Turker Abdeljaber, Osama Avci, Onur Gabbouj, Moncef. (2019). 1-D Convolutional Neural Networks for Signal Processing Applications. 8360-8364. 10.1109/ICASSP.2019.8682194.

[6] Daitan. (2019). How To Build a Deep Audio De-Noiser Using TensorFlow 2.0. https://medium.com/better-programming/how-to-build-a-deep-audio-de-noiser-using-tensorflow-2-0-79c1c1aea299

[7] Rothman, Daniel. (2018). What's wrong with CNNs and spectrograms for audio processing? https://towardsdatascience.com/whats-wrong-with-spectrograms-and-cnns-for-audio-processing-311377d7ccd