CS230

Creating a Better WFH Experience: Removing Noise from Audio

Schuyler Tilney-Volk[1], **Semir Shafi**[2], and **Marco Mora-Mendoza**[2]

[1]Department of Electrical Engineering, Stanford University
[2]Department of Computer Science, Stanford University

Abstract

As a result of the COVID-19 pandemic, non-essential workers have been working from home for the past couple months. Zoom, Microsoft Teams, and Webex have quickly replaced in person meetings. However, background noise including playing children, television, typing, coughing, breathing, and chewing can impact the quality of a video conference call with colleagues. Until recently, many models have used filters typically found in electrical and sound engineering to limit the range of transmitted frequencies. Our goal is to develop a tool capable of using deep learning to remove noise in real time. Even after COVID-19, this problem will most likely remain pertinent as companies continue to integrate work-from-home solutions into their work culture.

1 Task Description and Background

Noise filtering has typically been accomplished using a variety of tools and filters that stem from electrical engineering and signals processing. As applications in digital software systems have become more nuanced, typical filters are sometimes unable to meet the modern requirements for new applications of noise filtering, as they often operate on eliminating all sounds of a certain frequency. For example, this can prove to be problematic in contemporary voice filtering applications–if two humans are speaking at the same time and with voices of roughly the same frequency, traditional filtering techniques will eliminate neither or both voices. Deep learning is capable of recognizing more complex non-linear patterns such as undesired background noise, and is therefore a recent topic of study in voice filtering applications and background noise removal.

In 2012, Maas et al. introduced a RNN model to remove noise from the input to speech recognition software [5]. They conclude that both mutliple hidden layers and temporally recurrent connections are important to perform well on both the training data and unseen noise.

Kim and Smaragdis (2015) developed a fine-tune scheme to further improve the performance of an already-trained Denoising AutoEncoder (DAE) in the context of semi-supervised audio source separation [4].

Park and Lee (2017) attempt to remove babble noise to help increase the intelligibility of human speech for hearing aids [6]. They propose using a convolutional neural network, specifically the Redundant Convolutional Encoder Decoder (R-CED). This convolutional neural network can be 12 times smaller than a recurrent neural network and still achieve better performance indicating that usefulness in embedded systems.

Grais et al. (2017) used a similar approach of denoising autoencoders with convolutional neural networks [3]. However, their goal was to separate out all individual sources of noise in audio clips.

CS230: Deep Learning, Spring 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

They used three metrics to determine their success: SDR (signal distortion ratio), SIR (signal to interference), and SAR (signal to artifact).

Germain et al. (2018) tried a novel approach still using a convolutional neural network approach with deep feature losses [2]. This loss compares the internal feature activations with a different network trained for acoustic environment detection and domestic audio tagging.

Our project was inspired by the the 2017 paper by Park and Lee suggesting the use of a convolutional neural network to help solve the audio denoising problem. We use their preprocessing description on our datasets and their Cascaded R-CED Convolutional Neural Network.

2 Dataset

We use two sets to create data of speech with added noise: the Mozilla Common Voice (MCV) dataset, and the UrbanSound8K dataset [1]. The model adds urban sounds as noise signals to the speech examples from the MCV data, which is then used as input for our deep learning model.

2.1 The Mozilla Common Voice Dataset

The Mozilla Common Voice dataset consists of 3,401 validated hours of speech in 40 languages. Each entry in the dataset consists of a unique MP3 and corresponding text file. Many of the 4,257 recorded hours in the dataset also include demographic metadata like age, sex, and accent that can help train the accuracy of speech recognition engines. Overall, the dataset consists of 38GBs of data.

By accent, the dataset consists of 23% recordings in US English, 9% in England English, 4% in an Indian and South Asian accent, 3% Australian, 2% Scottish, 1% Irish, 1% Southern African, and 1% from New Zealand.

By age, 22% of the recordings come from individuals with ages 19-29, 15% from ages 30-39, 9% from ages 40-49, 5% from ages 50-59, 5% from ages <19, 4% from ages 60-69, and 1% from ages 70-79. By sex, 46% of the recordings are from males, 13% from females.

It should be noted that data preprocessing and creation for this dataset took a significant amount of time due to missing samples and empty files.

2.2 The UrbanSound8K Dataset

The UrbanSound8K dataset contains small snippets of sounds of less than 4 seconds each. There are 8732 labeled examples of ten different common urban sounds in WAV format. These include sounds of air conditioners, car horns, children playing, dogs barking, drilling, engines, gun shots, jackhammers, sirens and street music.

2.3 Preprocessing

We downsampled the original audio signal from both datasets to 8 kHz to reduce the inherent computation complexity and dataset size. This audio signal is a one dimensional series of analog data. Because we plan to use convolutional neural networks, which are traditionally used on images, we convert the audio signal into a two dimensional representation, time vs frequency. To achieve this, we leveraged a 256-point Short Time Fourier Transform which moves across the audio signal and computes a Discrete Fourier Transform. Following the steps mentioned in the original paper, we set the periodic Hamming Window to length 256 with a hop size of 64. This is process is illustrated in Figure 1.

Afterwards, we concatenate eight of these consecutive noisy STFT vectors into an input vector with shape (129, 8). We get the dimension 129 since the STFT vector is symmetric and thus, we only need 129 of the 256 datapoints. Our target vector has shape (129, 1) sampled from the most recent timestep from the clean audio. Thus, the neural network looks at 7 previous noisy inputs timesteps to help output the most recent value.

In Figure 2 we see a sample clean and noisy audio sample respectively from the same datapoint.

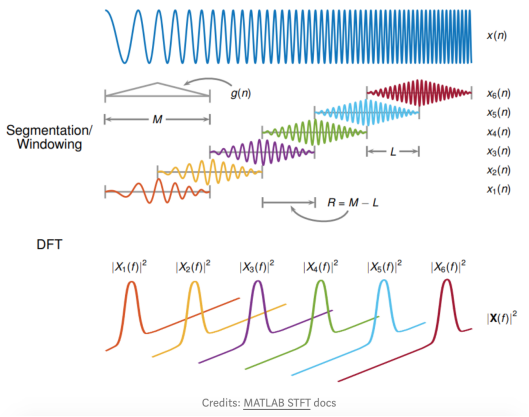
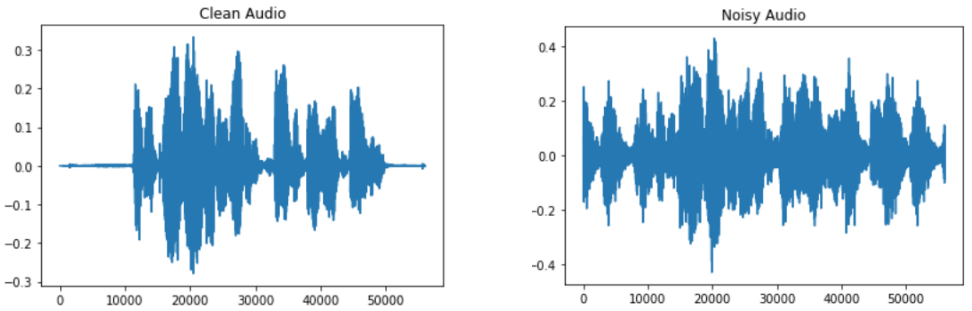


Figure 1: MATLAB STFT Documentation



(a) Clean audio

(b) Noisy Audio Created by Overlaying Clip from UrbanSound8k on Clip from MCV Dataset

Figure 2: Sample Audio Clips

3 Architectures

In order to improve performance from the baseline, we examined several architectural changes and their filtering effects.

3.1 Modifying the Autoregressive System with an LSTM layer

The preprocessing steps described combined the previous 7 STFT noisy vectors with the current timestep to help the neural network determine the current signal based on past observations. We added an LSTM layer at the beginning and end of the original architecture, as well as in a skip layer from the beginning to end.

To create a system that has the potential to become productionized we need to make all of the processes as memory-efficient and quick as possible including the preprocessing steps. We explore appending a different number of previous noisy vectors to see the outcome on the results on the SDR of our test set.

The observations in the table below ran on the same CNN for 200 epochs.

3.2 Architecture Stacking

While the original architecture worked well, we found it wasn't removing as much noise as desired. This possibly indicated that the complexity of our model wasn't high enough–it lacked enough non-linearities to reduce all of the noise patterns. As such, we doubled the model size by attaching the output of the original architecture to the input of a copy of the original architecture. This created

a bottleneck-like model that we hypothesized might act similarly to an encoder-decoder and aid in noise removal.

The observations are shown below for 200 epochs.

3.3 Convolutional Layer Filter Width

Our final architectural modification was editing the filter width. We noticed that the model found sudden impulses of noise, like a dog barking, more difficult to remove that gradual build-ups or constant background noise. Therefore, we increased the filter width from 9 to 13 and 17, as this would give the model a larger visual field.

4 Results

For our baseline, we ran the model described in the Medium article on a subset of their dataset. Our baseline model was trained on 10 training and 5 validation packages consisting of a total of 45,000 examples.

The network used contains 16 of blocks of convolution, ReLU, and batch normalization, adding to 33K parameters. Once the network produces an output estimate, the model aims to minimize the mean squared difference (MSE) between the output and the target signals.

After 200 epochs, we obtained a root mean square error of .4426 on the training set. After the first epoch, it was 1.3541.

We computed a noise to distortion ratio comparing the original clean audio and the output audio from the neural network using this formula:

$$\text{SDR} = 10 \log_{10} \frac{||y||^2}{||f(x) - y||^2}$$

After 200 epochs on our baseline model, we got an average SDR of -6.456 on 877 test observations.

Model	SDR
LSTM	-6.732
Stacked	-5.25
Conv13	-4.98
Conv17	-4.90

# Vec	MSE	SDR
1	1.143	INF
3	.7823	-735.424
5	.5998	-42.135
7	.4426	-6.456
9	.4318	-5.184
11	.4825	-5.312

4.1 Analysis

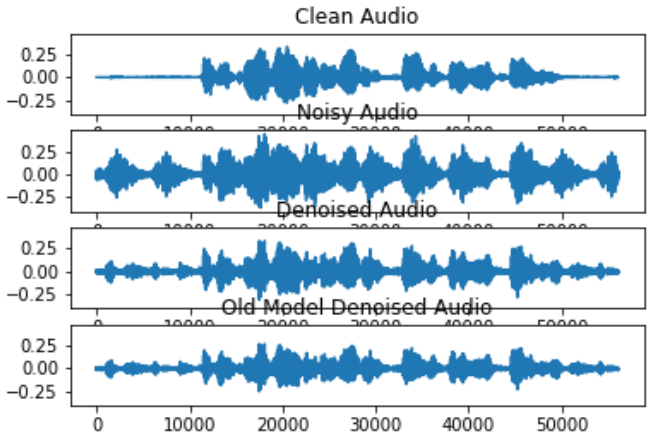
Our results are fairly explainable logically. The smaller the number of previous vectors the neural network is allowed to observe, the larger the magnitude of the SDR. By restricting the visual field of the model with using smaller filters, the model was given less analytical capability.

In addition, the addition of the LSTM layer actually slightly hurt performance. The LSTM forces the model to analyze in time rather than in frequency or spatially by looking for feed-forward correlations in sequential samples. The addition of such a capability, and the negative effect it had, showed that the model was more capable when drawing conclusions mostly from the frequency domain, and in this way was acting more similarly to a typical hardware filter.

The additional complexity introduced by the stacked model also yielded few benefits. The resulting samples from this model seemed no better to human ear than the samples from the original. This new model also contained twice as many parameters, which makes the trade-off between accuracy and speed significantly less attractive than the original model.

Finally, the only addition that did seem to help was the width of the convolutional layers within the blocks, but only after significant increase. The SDR presented with the Conv13 and Conv17 layers

was better than the original, and the amplitude of the new model's output was closer to the amplitude of the noisy signal than the original model's output. This was beneficial in that the clean signal (voice) could be more clearly heard. In addition, the conservation of amplitude also showed that the model wasn't just reducing the amplitude of all sounds in order to reduce the noise, which was one of our primary concerns.



However, there was a downside to this model in that the increase visual field ended up creating almost a slightly slurred version of the voice. The difference is barely noticeable to the human ear in the actual sound samples, but may be more obvious in real-time filtering implementations.

5 Discussion/Conclusion

Overall, the original architecture proved to be the best architecture, only improved upon with minor changes such as convolutional filter width. This was somewhat unexpected, as we thought that the addition of an LSTM would make the model far more robust to a larger variety of background noises.

In the future, we believe a better model can be achieved via different data processing experiments. Because the focus of this project was strictly on noise removal, we did not address how the model acts on samples with no noise injected. Modifying the training set is a rather painstaking process that fell outside the scope of the project, but we recommend this for future study as it may lead to a more applicable model.

6 Contributions

- Semir wrote milestone 2, fixed implementations of original model, and ran experiments with smaller conv filters.
- Schuyler fixed the dataset generation code, modified the original model architecture, and collected results.
- Marco did most of the writing: he wrote the proposal, milestone 1, and the majority of final paper.

7 Acknowledgements

We would like to acknowledge our assigned CS230 Course Assistant, Jo Chuang, for guiding us through our project.

References

[1] Daitan Innovation (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

[2] Germain, Francois G., Qifeng Chen, and Vladlen Koltun (2018). *Speech denoising with deep feature losses*. https://arxiv.org/abs/1806.10522

[3] Grais, Emad M., and Mark D. Plumbley (2017). *Single channel audio source separation using convolutional denoising autoencoders.* 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP), https://ieeexplore.ieee.org/document/8309164.

[4] Kim, Minje, and Paris Smaragdis (2015). *Adaptive denoising autoencoders: A fine-tuning scheme to learn from test mixtures.* International Conference on Latent Variable Analysis and Signal Separation, https://link.springer.com/chapter/10.1007%2F978-3-319-22482-4_12.

[5] Maas, A., Le, Q. V., O'neil, T. M., Vinyals, O., Nguyen, P., & Ng, A. Y. (2012). *Recurrent neural networks for noise reduction in robust ASR.* https://research.google/pubs/pub45168/

[6] Park, Se Rim and Jinwon Lee (2017). *A Fully Convolutional Neural Network for Speech Enhancement.* INTERSPEECH, https://arxiv.org/abs/1609.07132.

[7] Foster et. al. (2015) *CHIME-HOME: A DATASET FOR SOUND SOURCE RECOGNITION IN A DOMESTIC ENVIRONMENT* https://core.ac.uk/download/pdf/30342817.pdf

[8] J. Chien and A. Misbullah, (2016) "Deep long short-term memory networks for speech recognition," 2016 10th International Symposium on Chinese Spoken Language Processing (ISCSLP), Tianjin, pp. 1-5, doi: 10.1109/ISCSLP.2016.7918375.