Removing Noise from Speech with Deep Learning

First Milestone

1 Task summary

Our goal is to reduce (or in the best case, entirely remove) the noise from audio recordings containing noisy speech. The idea is to use the WaveNet[1][2] architecture to generate clean audio from the noisy one.

2 Data acquisition and exploration

For our training and testing data, we used a dataset called Noisy speech database for training speech enhancement algorithms and TTS models[3] by the University of Edinburgh. It consists of ∼23000 clean-noisy pairs2 from 56 different speakers. The samples are stored in separate .wav files of varying length.1

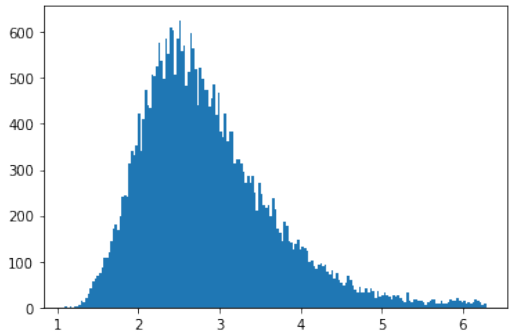
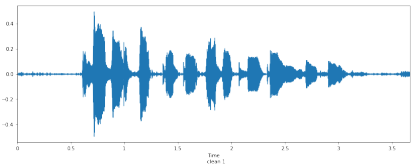
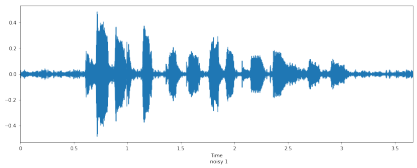


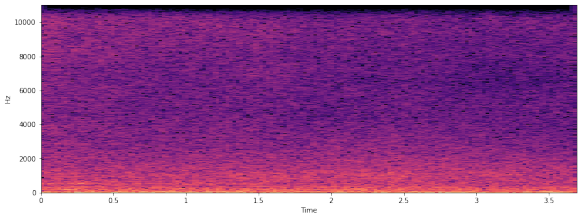
Figure 1: Duration distribution histogram of a subset of speeches



(a) clean



(b) noisy



(c) noise spectogram

Figure 2: Noisy and clean samples and the spectogram of the noise from the audio in figures 2a and 2b

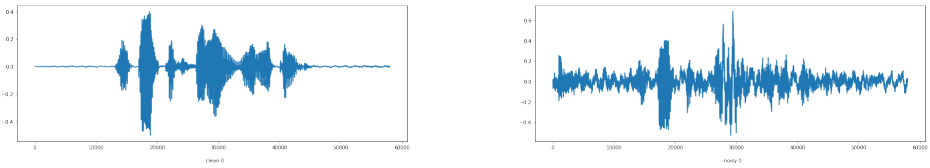
3 Data preprocessing

3.1 Data selection by duration

We intend to zero pad the data in order to make the samples the same length. To reduce the number of zeroes, we only use the n samples that are closest to each other in duration. This way, we decrease the number of training samples, but hopefully, the efficiency of our network will be much better.

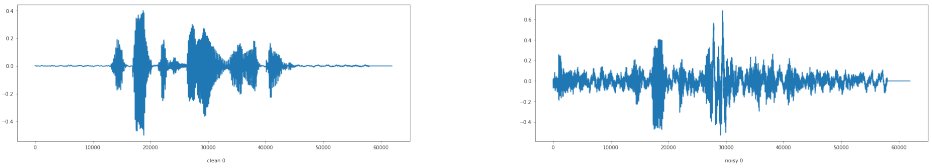
3.2 Zero padding

Our network architecture only supports samples of the same length. To fulfill this constraint, we add zeroes to the end of each recording. The number of necessary zeroes is calculated from the difference between the longest and the current sample.



(a) clean (b) noisy

Figure 3: An example clean-noisy pair

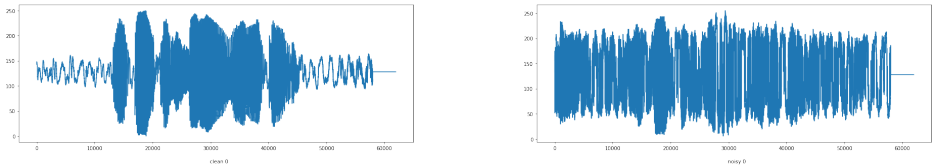


(a) clean (b) noisy

Figure 4: An example clean-noisy pair after zero padding

3.3 μ-law transformation

To make future computations faster we, reduce the samples to 8 bit with μ-law transformation.



(a) clean (b) noisy

Figure 5: An example clean-noisy pair after padding and μ-law transformation

3.4 Other preprocessing approaches

The techniques mentioned above are the preprocessing steps we are using right now. Because we don't have a model yet, we can't be sure if these are the right steps, so we came up with other approaches if we find out that the original data manipulation is not suitable for our network. These steps are shown in the figure$_6$ below.

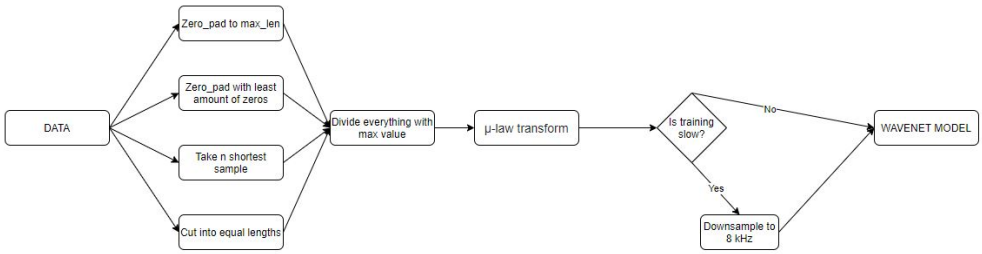


Figure 6: Preprocessing flowchart

3.4.1 Making the samples the same length

- **Zero padding to maximum length**

  This approach is the easiest to implement. We find the longest sample and zero pad the others to match its length (so we can use all the audio from the training set). The problem with this approach is that the number of zeroes added to some of the shorter samples is a lot compared to their original size, and we don't know how well our model can handle it. The increase in size means that we will need a vast amount of memory to store it. The other disadvantage is that if the model can handle the changes, the learning will probably be much slower than before because of the unnecessary zeroes.

- **Zero padding with the least amount of zeroes**

  This step is what we described in sections 3.1 and 3.2.

- **Take the n shortest samples**

  In this approach, we select an arbitrary number of samples from the beginning of our dataset (the data is arranged from short to long). This way, the number of zeroes in the padding is small, and the memory requirement is reduced drastically. Because of the reduction in the number of samples, the learning will be faster, but finding enough test data with the necessary length could be challenging.

- **Cut the samples into equal lengths**

For this technique, we have to find the optimal sample length (which could be hard) and discard every data point shorter than that. After we figured out the optimal length, we cut the end of all longer samples to match that length. This process could provide us optimal sized data, but cutting can break the cohesion of the audio.

3.4.2 Divide everything with max value

The μ-law transform we described in section 3.3 needs inputs between -1.0 and 1.0. Dividing every value with the maximum solves this problem.

3.4.3 Downsampling to 8kHz

If we figured out the right combination of preprocessing steps, and the only problem is the speed of training, we will downsample the audio to 8kHz. This step can make training much faster. On the other hand, the quality of the audio will be worse.

References

[1] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD].

[2] Dario Rethage, Jordi Pons, and Xavier Serra. *A Wavenet for Speech Denoising*. 2018. arXiv: 1706.07162 [cs.SD].

[3] Cassia Valentini-Botinhao. *Noisy speech database for training speech enhancement algorithms and TTS models*. 2017. DOI: 10.7488/DS/2117.