Speech Enhancement

Link to project: https://github.com/SreemukhMantripragada/SpeechEnhancement

(All the code portions are clearly commented in the notebooks present in the repository)

Sreemukh Mantripragada

B.Tech ECE (2018-22)

National institute of Technology, Warangal

sm921848@student.nitw.ac.in

Overview

The goal of the project is to develop an approach to extract clean audio from sources that contain noises along with actual audio. In the following sections, an approach to carry out the process is mentioned.

There exist Digital signal processing filters to approach the problem such as a Wiener filter, but the proposal is to use a Deep learning model as the data is very rich, versatile and is available in a very large scale.

The model being trained is given inputs as (clean audio + noise) and we expect the model to return us the clean audio only, by removing the noise components.

As datasets related to some common noises and clean audio are present, we add noise intentionally to clean audio to generate the input for the model.

Goals

Given an input sound clip having noise along with clean audio, we need to output clean audio, but removing those components of the sound clip that correspond to noise.

Datasets being used

Mozilla common voice (MCV) Dataset - For all the recorded **clean audio**.

UrbanSound8K - Urban Sound Datasets - It consists of all the **common noises** that tend to get added to speech in daily lives.

Approach

- Sound clips from noises are added to the clean sounds and these are given as input to a keras based model and the expected output is corresponding to the clear sound.
- The model being trained is a deep convolutional neural network, which has an encoder - decoder architecture. The input and output images to the network are 2D representations of the audio present. 2D representations of audio being used are Spectral magnitude vectors(computation of Spectral magnitude vectors is explained below). Studies show that spectral magnitude vectors' representation gives better performance for this model architecture.

Understanding the Datasets

1. Mozilla Common Voice Dataset(English-50GB):

   This consists of 1,932 recorded hours of clean audio in the form of short mp3 files.

   This consists of a varied range of speakers and nearly 11,000 different voices which helps in generalisation of the model.

2. UrbanSound8K(6GB):

This consists of small snippets(<4 sec) of noise clips commonly observed in everyday phone calls apart from the clear person audio. There are more than 8000 labelled examples of common urban noises, The list as given in the website mentioned above is :

- Air conditioner
- Car horn
- Children Playing
- Dog barking
- Drilling
- Engine idling
- Gun shot
- Jack hammer
- Siren
- Street music

Data Preprocessing

Instead of feeding raw input as it is to the model, processing the data helps the model converge quickly and generalize better.

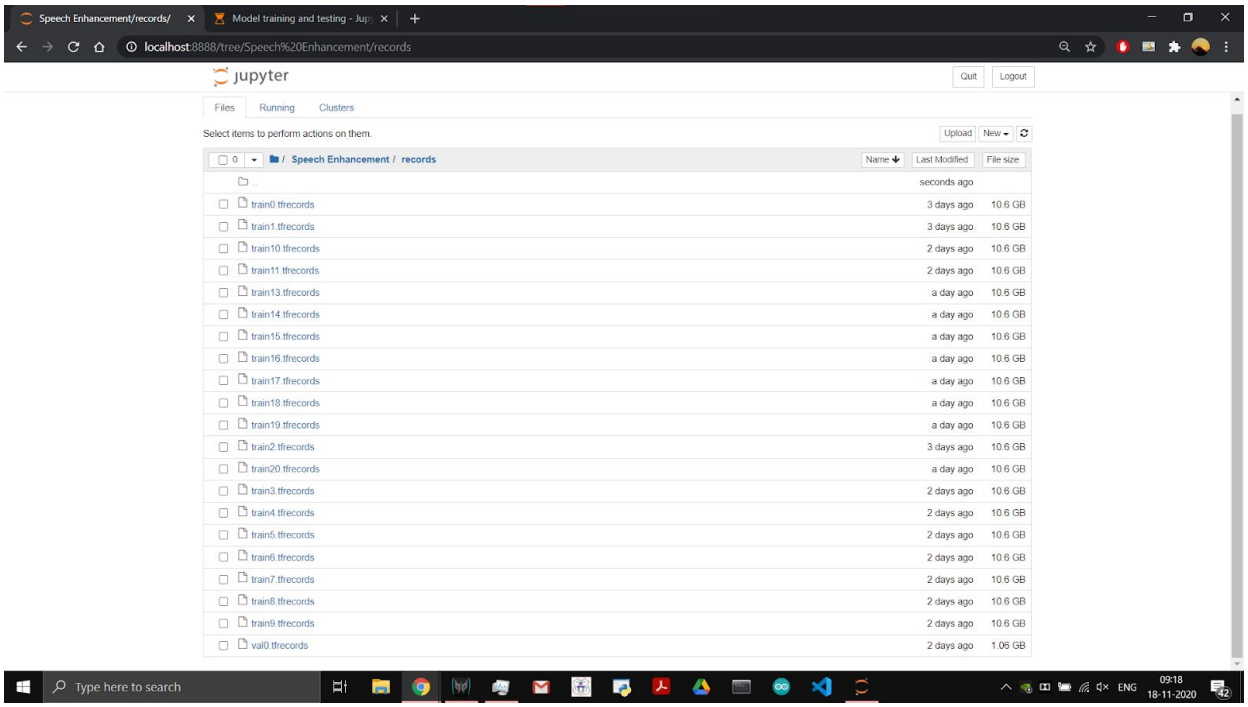
The steps followed for each example from the datasets:

- **Audio data is loaded** at 8khz rate into the workspace. In a mel spectrogram representation, if we sample the audio at 16kHz, we can represent the audio in the range of 0-8kHz. Hence, while using librosa.load, sample rate(fs) is set to 16kHz in the code.
- **Removing silent frames** from the examples, this helps shrink down the important information present in the audio clip, leading to lower computational needs as the datasets are quite large.
- As the approach involves using a Convolutional Network architecture, the audio cannot be fed directly as the data is audio which is a time series of scalar values. Hence, a time-frequency representation is used to represent the data in 2D. Instead of computing the fourier transform over the entire signal, we are computing

spectral magnitude vectors using a **Short Time Fourier Series**(STFT) that uses a window based approach for calculation.

- **Calculation of STFT**: It is defined with respect to a periodic hamming window with length 256 and hop size of 64. This ensures 75% overlap between the STFT vectors.
- We extract the magnitude vectors from the 256 point STFT and take 129 point portion, by removing the symmetric half.
- We **concatenate 8 consecutive STFT vectors** (noisy(clean+noise)) and use them as the input to the model. Hence, the model expects input of the shape(None, 129,8,1).

  All the data is being stored in the form of tensorflow tf_records, this helps for more efficient training in terms of computation.The tfrecords size is very large and cannot be uploaded to github. Each train tfrecord contains 10000 examples' data.



Model Architecture

The model being used is a **Cascaded Redundant Convolutional Encoder-Decoder** Network(CR-CED). It is based on symmetric encoder-decoder architectures. Both encoder and decoder components consist of repeated blocks of Convolution, Activation, Batch Normalisation.

There are also a few skip connections between some of the encoder and decoder blocks, similar to Residual networks(Res-Net) which help in avoiding vanishing gradients as the network is deep.

Important property of the CR-CED is that convolution is done in only one dimension, thus the frequency dimension stays intact, throughout the propagation.

As the model is very light, it can easily be implemented in edge cases, such as mobile phones, headphones etc.. It has 33k parameters.

Training

The produced output from the model is compared to the actual clean audio and is trained based on the **Mean squared loss** between Actual clean audio clip and the one produced. The loss function is chosen so as to converge the model to a mean position where it can handle all the source separation tasks overall the examples on an average.

Training metric chosen is **Root mean square error**.

As the training data is very large, data is fetched in **batch**es of size 512 randomly from the entire set of training tfrecords.

After the model is trained, random audio clips from clear sound and noise dataset are merged in the same fashion as done in preprocessing, and later fed as input to the model and the output is compared.

Output from the model is the magnitude of the stft features of the corresponding audio. Hence, to get back the audio, inverse STFT is done and it is scaled based on the input noisy audio.

Steps to use the model on a new audio clip are mentioned in the Github link.

Conclusion

The model performs fairly well on the examples, but it is still limited. Methods that can be deployed are using model ensembling, using a deeper network, sampling to get more data by reducing hop length. Experimenting with model architecture can also help us converge at a general model.