Internship Report on

## Real Time Implementation of Speech Enhancement using Deep Learning Technique

Prepared by

Swastik Srivastava

Persuing Btech. From:

Manipal Institute of Technology, Manipal

In:

Electronics and Communication

Under the Guidance of:

Prof. Arun Kumar

At:

Indian Institute of Technology, Delhi

Internship Duration :

2 months

Submitted on:

28th July 2023

# Speech Enhancement using Deep Learning

## Overview of speech enhancement

In the natural environment ,the existence of noise is inevitable. Speech, as it is required is mixed additively with the surrounding noise. This modifies the speech characteristics leading to its quality/intelligibility degradation. This is not only undesirable in human-to-human communication but may also lead to errors in human-to-machine interaction which the world is largely starting to depend on. Thus, in any speech-based application, it is important to first eliminate the background noise and obtain a clean speech from a recorded signal.

Speech enhancement aims to improve speech quality by using various algorithms. The aim of enhancement is an improvement in intelligibility and perceptual quality of degraded speech signal using audio signal processing techniques.

Intelligibility and clarity are difficult to measure by any numerical algorithm. Typically listening tests are used, but they are costly. The main methods for upgrading speech are the removal of background noise, echo suppression and the procedure of artificially bringing various frequencies into the speech signal. We shall concentrate on the removal of background noise after quickly talking about what different methods are about.

Each speech measurement performed in a natural environment contains some amount of echo. Echoless speech, estimated in a unique anechoic room, sounds dry and dull to human ear. Echo suppression is required in big halls to enhance the quality of the speech signal, particularly if the distance between the microphone and the speaker is large. In the current telephone networks speech is band-limited between 300–3400 Hz. At some point, the markets will be controlled by third-generation phones in which the frequency band of the speech is, for example, 50-7500 Hz. The enjoyment of this wideband speech will be restrained except if the entire conversation is travelling in a wideband network. Artificial bandwidth expansion can be used to restore the frequencies that vanishes on the route. Speech compression also used these methods. When the background noise is suppressed, it is crucial not to harm or distort the speech signal. Something else to recollect is that quite natural background noise sounds more comfortable than more quiet unnatural twisted noise. If the speech signal is not proposed to be listened to by humans but for instance by a speech recognizer, then the comfort level is not the issue. It is significant then to keep the background noise low. Background noise suppression has numerous applications. For instance, utilizing a phone in a noisy environment like in streets or a vehicle is an obvious application. Usually, the background noise has been suppressed in an airplane when sending speech from the cockpit to the ground or to the cabin.

Another example is with the age, the audibility shrinks. Not hearing a grasshopper is a little impairment contrasted to the situation where the audibility range gets narrower, means powerful sounds become irritating and quiet sounds disappear.

In these applications it is important to keep the delay small. More exotic applications are enhancement of speech in wiretapping in a criminal investigation or restoring of old historical recordings. It is also a smart thought to enhance speech for coding and recognition purposes. Speech codecs have been streamlined for speech and they usually make the background noise sound unusual. Also, enhanced speech can be suppressed in fewer bits than non-enhanced.

## Classification of Speech Enhancement Algorithms

Several algorithms have been proposed in the writing of speech enhancement with the essential objective of improving speech quality. These algorithms can be classified into filtering techniques and spectral restoration

• Filtering Techniques

1. Spectral Subtraction Method

2. Wiener Filtering

3. Signal subspace approach (SSA)

• Spectral Restoration

1. Minimum Mean-Square-Error Short-Time Spectral Amplitude Estimator

2. Speech-Model-Based

1. **Spectral subtractive algorithms**: These are the simplest enhancement algorithms to implement. They are formed on the basic principle that since the noise is additive in nature, one can estimate/update the noise spectrum when speech is absent and subtract it from the noisy signal. Spectral subtractive algorithms were initially proposed by Weiss et al, in the correlation domain and later by Boll in the Fourier transform domain.

2. **Wiener filtering and its variations**: Given a set of calculations, relating state to the Fourier transform coefficients of the noisy signal, we want to find a linear or nonlinear estimator of the parameter of interest, to be specific, the transform coefficients of the clean signal. Wiener algorithm was started in the speech enhancement field by Lim and Oppenheim.

3**. Subspace algorithms**: The subspace algorithms are established fundamentally in linear algebra theory. These algorithms are based on the rule that the clean signal might be limited to a subspace of the noisy Euclidean space. The decay of the vector space of the noisy signal into “signal” and “noise” subspaces can be done using well known orthogonal matrix factorization techniques from linear algebra and the singular value decomposition (SVD) or the Eigen vector-Eigen value factorization. Work in this area was proposed by Dendrinos et al, who proposed the utilization of eigenvalue decomposition of the signal covariance matrix.

4. **Minimum mean square error (MMSE) algorithms**: Recently, deep learning techniques have gained popularity for speech enhancement results due to its capability to model the non-linearity in the data. Speech Enhancement is done in a supervised manner of deep learning, that is, a noisy speech is given at the input of neural network and the required clean speech at the output during its training phase.

Depending on how the output or the training target is learnt, this approach can further be divided as Mapping-based or Masking-based.

The mapping-based method aim to grasp a non-linear mapping function F from the observed noisy speech y(t) into the desired clean speech s(t).

𝑦(𝑡) 𝐹 → 𝑠(𝑡)

Owing to the fast-variation issues of raw speech signals and the high computational complexity they need. Such a learning strategy is often applied to the data in the spectral and cepstral domains rather than the temporal domain. To learn the function F, the neural networks are trained to reproduce the target features x that are extracted from the clean speech s(t) from the corresponding input features y that are extracted from the corrupted speech y(t). After the evaluated clean features, they will be then switched back to the time domain signals with the use of phase information from the original noisy speech.

Unlike the mapping-based method, masking-based method aim to learn a regression function from a noisy speech spectrum Y(n, f) to a Time-Frequency (T-F) mask M(n, f), i.e.

𝑌(𝑛, 𝑓) 𝐹 → 𝑀(𝑛, 𝑓)

Commonly binary based mask, that is, Ideal Binary Mask (IBM), refers to a T-F mask unit that is set to 1 if the local SNR is greater than a threshold R (signaling clean speech domination), or 0 otherwise (signaling noise domination). That is,

𝑀𝑏 (𝑛, 𝑓) = 1, 𝑖𝑓 𝑆𝑁𝑅(𝑛, 𝑓) > 𝑅

= 0, 𝑜𝑡ℎ𝑒𝑟𝑤𝑖𝑠𝑒

where SNR (n, f) represents the local SNR within the T-F unit at the frame index n and the frequency bin f. Hence, concluded that IBM is a binary matrix.

## Quality Metrics

To quantify the level of degradation in a speech signal, we use certain speech quality assessment methods. These are important to understand the initial degradation present in the signal and the improvement obtained after passing through a speech enhancement algorithm.

When a speech signal reaches human ear, a process of speech perception is initiated. The auditory event that is part of this process can only be measured by the description of the person listening to it. Thus, the listener can then judge a relationship between the received (degraded) and the expected (clean) speech. Hence, speech quality is as is perceived by the listener. This perception can be in terms of the intelligibility, loudness, naturalness and the listening effort required. Since the nature of the speech signal does not exist independent of the audience, it is an abstract measure. Although such a test would give genuine speech quality, the involvement of a human listener makes it is practically difficult, expensive and time-consuming. Thus, we use the objective measures which are calculated algorithms intended to assess the quality degradation.

In this work, to find the quality of the enhanced speech, we refer to the ITU-T P.862.2 recommendation for Perceptual Evaluation of Speech Quality (PESQ) of wideband speech. It is an intrusive method of assessment, that is, it takes as input the reference clean speech and the degraded speech signal as input. Based on this, it generates Mean Opinion Score (MOS) ranging from a minimum of -0.5 to a maximum of 4.5 which is comparable to the subjective score. The score is obtained for audio evaluated over a bandwidth of 50 – 7000Hz and taking into consideration the parameters like speech input levels, transmission channel errors, environmental noise, etc.

## Stages of the model

There are three stages of this on which this model works:

* **Training stage**: In the training stage, a regression DNN model is prepared from a collection of stereo data, including pairs of noisy and clean speech represented by the log-power spectra features.
* **Enhancement stage**: In the enhancement stage, the well-trained DNN model is fed with the features of noisy speech in order to generate the enhanced log-power spectra (LPS) features. The additional phase information is evaluated from the original noisy speech.
* **Overlap-add method**: In the end, an overlap-add method is incorporated to synthesize the waveform of the assessed clean speech.

## Feature Extraction

This is the most important part in the implementation of this algorithm. This process involves extracting log power spectra(LPS) as feature of the signal to be processed.Since the input signal is real, STFT is symmetric. For a frame size of 32ms (512 samples at 16 kHz) and overlap of 16 ms (256 samples at 16 kHz), Short Time Fourier Transform (STFT) is calculated.

Hence, the spectrum values at 257 frequency points are taken to obtain the input feature. To take into account the contextual information, the STFT from six adjacent frames (three on each side of current frame) are concatenated with current frame STFT to obtain the final input feature. Input Noisy Speech Feature Dimension: 257-D (STFT) x 7 (frames)

Output Clean Feature Dimension: 257-D (STFT) x 1 (frame)

## Result

The model was trained on different sizes of database and with varying number of noise types from Training database . Model with most number of noise types (115) and larger duration (~26 hours) showed the best results. Testing was done on WSJ Testing Database (Database B).

Along with DNN based approach, a pre- and post-processing method using LogMMSE was suggested for performance improvement. In pre-processing, the degraded signal is first passed through log MMSE based speech enhancement block then to the DNN during the enhancement stage. In post processing, the degraded signal is first passed through the DNN tool and thereafter the processed signal is passed through log MMSE block pre-processed signals give better PESQ score than normal processed and post processed signals at all SNRs except very low SNR of -10.

## Understanding of Codes

Training code operates by firstly mixing the noise speech to a clean speech to form a clean speech and then extracting LPS features of the mixed speech matching it with the features of clean speech and then finally saving the LPS features of clean speech.

Testing code just takes random noise and clean speech and runs the same code as in training code and gives the desired output as a clean speech.

This models code trains the model, monitors the validation loss, saves the best model weights, and plots the training and validation loss to visualize the model's performance during training.

The preparing data code aims at extracting features of various speeches and finding logarithmic transformations and changing them to suitable format for further analysis. Its also mixes various noises for specific SNR values and helps to find time taken by certain operations.

## Modifications in Testing Code for real time implementation

The testing code which has been prepared cannot be used to implement in real time. So some changes has to be done in the code to eliminate some that may arise in the course of implementing it in real time. The things that has to be done are:

1. A code to continuously read audio data from microphone or any input device on Raspberry Pi.
2. A code to continuously play the enhanced audio output device such as speaker.
3. For real time processing instead of reading wave files, frames of of the audio signals will be taken .
4. There must be changes in the code for buffer management in real time.

**Below is the modified code with all the above points taken into account.**

import numpy as np

import os

import pickle

import argparse

import time

import matplotlib.pyplot as plt

import prepare\_data as pp\_data

import config as cfg

from spectrogram\_to\_wave import recover\_wav

import sounddevice as sd

from tensorflow.keras.models import load\_model

# Set the workspace and other parameters

workspace = "C:/Users/Swastik/Desktop/Speech\_Enhancement\_LPS"

TE\_SPEECH\_DIR = "C:/Users/Swastik/Desktop/Speech\_Enhancement\_LPS/mini\_data \_small- Copy/test\_speech"

MODEL\_NAME = "results.h5"

ENHANCED\_FILES\_FOLDER = MODEL\_NAME.split('.h5')[0]

tr\_snr = 0

te\_snr = 5

N\_CONCAT1 = 7

n\_concat = N\_CONCAT1

n\_window = cfg.n\_window

n\_overlap = cfg.n\_overlap

fs = cfg.sample\_rate

scale = True

visualize = False

# Load the model

model\_path = os.path.join(workspace, "models\_multitask\_model2", "%ddb" % int(tr\_snr), MODEL\_NAME)

model = load\_model(model\_path)

# Load the scaler

scaler\_path = os.path.join(workspace, "packed\_features", "spectrogram", "train", "%ddb" % int(tr\_snr), "scaler.p")

scaler = pickle.load(open(scaler\_path, 'rb'))

# Initialize the audio buffer and frames list for real-time processing

audio\_buffer = np.array([])

frames = []

# Define the audio input and output callback functions

def audio\_input\_callback(indata, frames\_callback, time, status):

global audio\_buffer

# Append the incoming audio data to the buffer

audio\_buffer = np.concatenate((audio\_buffer, indata))

def audio\_output\_callback(outdata, frames\_callback, time, status):

global audio\_buffer

global frames

# Process complete frames in the buffer

while len(audio\_buffer) >= n\_window:

frame = audio\_buffer[:n\_window]

audio\_buffer = audio\_buffer[n\_overlap:]

# Apply frame processing (e.g., pre-processing, feature extraction, etc.)

frames.append(frame)

if len(frames) >= N\_CONCAT1:

# Concatenate N\_CONCAT1 frames and perform real-time inference

audio\_features = concatenate\_frames(frames, N\_HOP1)

enhanced\_audio\_frame = enhance\_speech(audio\_features)

# Output the enhanced audio frame

outdata[:len(enhanced\_audio\_frame)] = enhanced\_audio\_frame

# Remove frames that have been used

frames = frames[N\_HOP1:]

# Set up audio input and output streams

input\_stream = sd.InputStream(callback=audio\_input\_callback, channels=1, samplerate=fs)

output\_stream = sd.OutputStream(callback=audio\_output\_callback, channels=1, samplerate=fs)

try:

# Start audio input and output streams

input\_stream.start()

output\_stream.start()

# Run the program indefinitely, you can replace this with your main real-time loop

while True:

pass

except KeyboardInterrupt:

print("Interrupted by the user.")

finally:

# Stop and close audio streams

input\_stream.stop()

input\_stream.close()

output\_stream.stop()

output\_stream.close()

# Define the real-time processing function

def concatenate\_frames(frames\_callback, hop\_size):

# Assuming the frames\_callback list contains audio frames, you can concatenate them using np.concatenate

mixed\_x\_3d = pp\_data.mat\_2d\_to\_3d(mixed\_x, agg\_num=n\_concat, hop=1

def enhance\_speech(audio\_frame):

# Implement real-time model inference using the loaded model

# Convert the audio frame to a suitable format for inference (e.g., extract spectrogram features)

# Perform model prediction to get the enhanced audio frame (e.g., using model.predict)

# Post-process the enhanced audio frame if needed

# Return the enhanced audio frame

pred = model.predict(audio\_features)

pred\_sp = np.exp(pred)

enhanced\_audio\_frame = recover\_wav(pred\_sp, mixed\_cmplx\_x, n\_overlap, np.hamming)

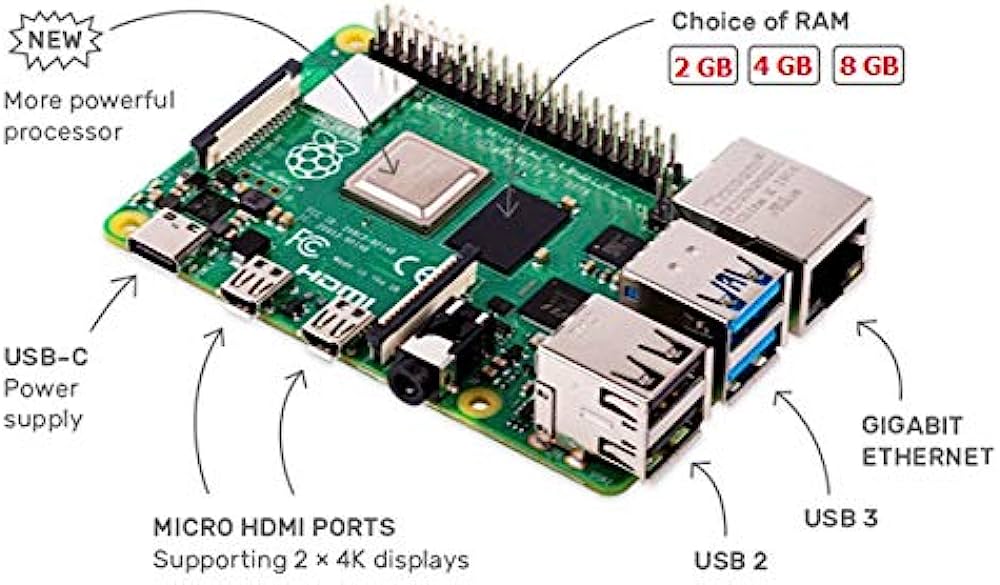
enhanced\_audio\_frame \*= np.sqrt((np.hamming(n\_window)\*\*2).sum()) # Scaler for compensate the amplitude change after spectrogram and IFFT.

return enhanced\_audio\_frame

## Implementing the code on Raspberry Pi

* Connect the raspberry pi-4 to a computer screen using a HDMI cable.
* Copy this modified testing code, trained code, and the testing dataset on the rpi-4 using a USB drive.
* Connect a microphone to the rpi either at jack input or using HDMI cable to generate sound from the computer screen.
* Run the code on the terminal of the rpi and test your model for the real time implementation.

## Raspberry Pi-4



Raspberry pi a microprocessor which is generally used to implement a python code in real time. It is the best known user friendly microprocessor. So we are using this for our real time implementation experiment. Some properties of raspberry pi-4:

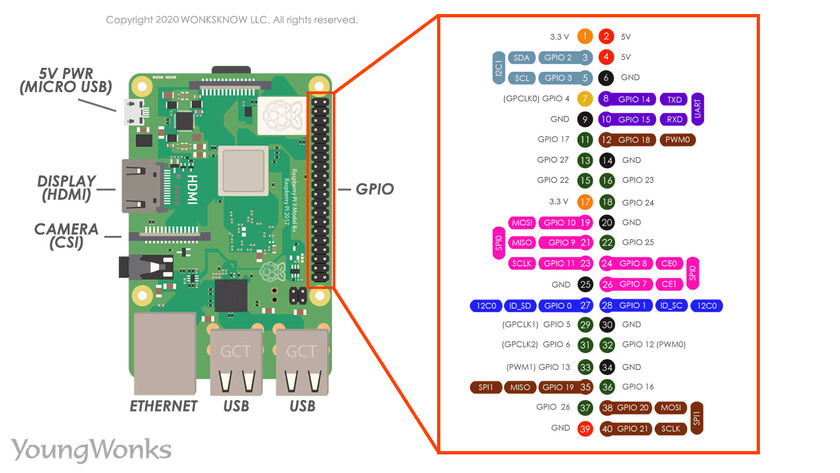
* **Performance:** The Raspberry Pi 4 has less computational power as compared to desktop computers so running deep learning models in real time can be computationally intensive
* **Dependencies:** Install all required libraries on Raspberry Pi4.
* **Latency:** Real time processing of audio might introduce some delay due to hardware limitations. So code must optimized accordingly.
* **Power Supply:** Adequate power supply to the processor must be ensured otherwise it may lead to unstable results.
* **Audio Hardware compatibility:** Audio device must compatible with the processor which we are using.

Now coming to implementing it to this project only Raspberry Pi4 can be used because it is the only Pi which is compatible to Tensorflow which is a very important library to run a deep learning model.

## How to setup raspberry pi on a computer

1. Connect the raspberry pi to the computer screen using a micro HDMI cable.
2. You will have to setup a username and password for the board.
3. Now connect a local mobile hotspot to the Pi.
4. Now your raspberry pi is connected to a computer system.

## GPIO Pins chart



It’s a tiny computer board that comes with CPU, USB ports, GPU, Wi-Fi, I/O pins, USB, Bluetooth and network boot and can do perform functions like a regular computer.

An amazing element of the Raspberry Pi is the row of GPIO (universally input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi sheets (uninhabited on Pi Zero and Pi Zero W). Preceding the Pi 1 Model B+ (2014), boards included a shorter 26-pin header.

Voltages - Two 5V pins and two 3V3 pins are available on the board, just as various ground pins (0V), which are not configurable. The rest of the pins are all general-purpose 3V3 pins, which means outputs are set to 3V3 and inputs are 3V3-tolerant.

Outputs - A GPIO pin assigned as an output pin can be set to high (3V3) or low (0V). low (0V).

Inputs - A GPIO pin assigned as an input pin can be perused high (3V3) or low (0V). This is made simpler with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for many other pins, this can be arranged in software.

PWM (pulse-width modulation)

- Software PWM available on all pins

- Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19

SPI(serial peripheral interface)

- SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)

- SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)

I2C

- Data: (GPIO2); Clock (GPIO3)

- EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)

Serial

- TX (GPIO14); RX (GPIO15)

## How to Remote Desktop to your Raspberry Pi with VNC Viewer

VNC Viewer is a way for you to access and control your Raspberry Pi desktop from another computer such as a Mac or Windows system.

Here are the steps to do this setup:

1. Connect the raspberry pi to a monitor system with the keyboard and mouse attached to the board.
2. Open the terminal of the microprocessor and type the command **iconfig** which will open the configuration screen of the raspberry pi and hence will get its IP address.
3. Now download the VNC viewer on your remote desktop.
4. Once downloaded, enter the IP address noted in the software.
5. Now, a dialog box will appear seeking for username and password and enter both the details same as entered at the time of setting up the raspberry pi on the main system.
6. Now you are good to go on your remote desktop.

## Conclusion

I was able to completely understand the algorithm used in this project. I was also able to identify the hardware that should be used to implement this algorithm in real time and got a fare idea of how to operate with a raspberry pi. Identification of certain changes in code to make it function in real time has been done.