Pune Vidyarthi Griha1 College of Engineering
and Technology Pune
&
G H Raisoni College of Engineering
Nagpur

DIGITAL SIGNAL PROCESSING

Application Assignment

Speech De-noising Using Deep Learning

Hybrid Group No.: 5
Conglomerate No.:
Group Member 1: Mohammad Shahbaz Alam
Group Member 2: Devidas Ingale
Group Member 3: Avdhoot Patil
Group Member 4: Damini
Group Member 5: Tina

June 20, 2021

Contents

1. Abstract

In this project, we aim to explore, implement and demonstrate a Speech denoising using CNN framework with multiple data Convolutional Neural Networks. This paper mainly focuses on the concepts of speech denoising system that has the job of removing the background noise in order to improve the speech signal. Besides many other applications, this application is especially important for audio conferences where noise can significantly decrease speech intelligibility, we tackle the problem of speech denoising using Convolutional Neural Networks (CNNs). When we give a noisy signal to the system, we aim to build a statistical model that can extract the clean signal (the source) and return it to the user.

Here, our prime focus is on source separation of regular speech signals from noise often found in an urban street environment. This paper presents denoising of audio using the convolutional neural network (CNN) model in deep learning. This analysis is done by adding 1% to 10% Environmental noise and granular noise to the input and then processing noisy audio files by short term fourier transform then further analyzing and processing by CNN model to denoise it. Further, qualitative and quantitative analysis of the denoised audio is performed. Under qualitative analysis comes the quality of audio where clarity factor, texture, uniform region and non-uniform region, smoothness is considered.

The results from the analysis and experiment show that the CNN model can efficiently remove a lot of Environmental noise and granular noise and restore the audio details and data than any other traditional/standard audio filtering techniques.

2. Description of Speech Denoising

The problem statement for the application is described as follows:
Usually, in communication systems, the received signal are polluted with noise and distortion, which are mainly caused due to channel behavior. Thus, at the receiving end, the signal may lose its information due to corruption. Thus, denoising of the received signal is essential for efficient communication in one-to-one as well as broadcast systems.

Speech signal denoising, or noise reduction, is the removal of noise and distortions present in the signal for recovering back the original signal, free from noise. The objective is to improve the Signal-to-Noise Ratio (SNR) of the incoming speech audio. This is an important application of Digital Signal Processing, which has various uses such as in cellular phones, hearing aids, teleconferencing (has become widespread after COVID-19 pandemic) etc.

In this project, we worked using principles of Digital Signal Processing, namely, Short-Time Fourier Transform; along with exploring deep learning convolutional neural network by 1D CNN & 2D CNN

We explored the principles and made comparisons to identify that 1D CNN outperforms 2D CNN.

3. Overview of our approach

We present a deep learning approach to denoising speech signals by processing the raw waveform directly. Given input audio containing speech corrupted by an additive background signal, then we use two long wav signals one clear version audio wav file and other same audio with background noise, the system aims to produce a processed signal that contains only the speech content. Recent approaches have shown promising results using various deep network architectures. In this paper, we propose to train a fully-convolutional context aggregation network using a deep feature loss. That loss is based on comparing the internal feature activations in a different network.

For our application, first, we pre-process the incoming audio samples. The pre-processing part is of major importance and utilizes digital signal processing concepts.

Here we used the dominant Short Term Fourier Transform-based analysis in our implementation. This method is unable to provide the information on variation of spectrum and it (in a constant time and frequency resolution) can represent a signal in both the domains: frequency as well as time, with the help of time-windowing function. Here, we used the Hann window.

After this we tried to improve upon our approach by using the deep learning approach using 1D CNN and 2D CNN. We use neural networks to find a transformation between the already pre-processed noisy clip and the clean clip. We used the same architecture of our neural network to train on transformed signal. Any nonlinear mapping between the input and the output frequency-time domains can be learnt using neural networks. This helps us in denoising speech.

4. Principles and Theory

We used the short term fourier transform, for pre-processing the audio samples, before giving them as inputs to neural network for training:

4.1 Short-Time Fourier Transform

Short-time Fourier transform (STFT) is a sequence of Fourier transforms of a windowed signal. STFT provides the time-localized frequency information for situations in which frequency components of a signal vary over time, whereas the standard Fourier transform provides the frequency information averaged over the entire signal time interval.

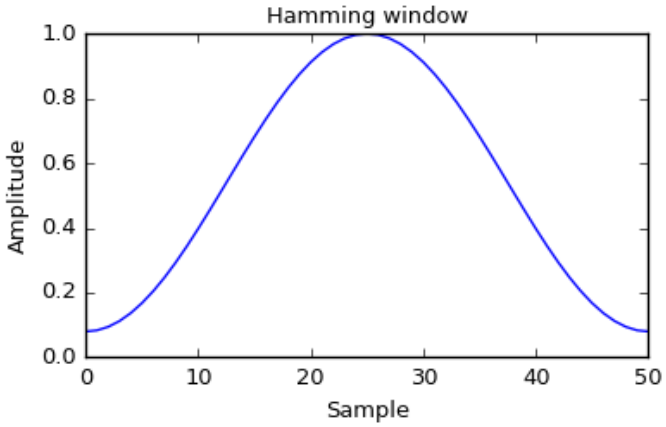


Figure 1: Hann Window

If we use window functions that are steep near the boundaries such as the rectangular window, this leads to a larger slope near that section's boundaries. This results in artifacts that are due to interferences that are spread across the entire frequency spectrum. These frequency components come from the window function instead of the original signal which is undesirable. Hence, we use the Hann window, which is popularly used in signal processing. It is a raised cosine window, which goes to 0 very smoothly at the boundaries. This softens the artifacts as discussed above in the frequency domain. The Hann window is given by the following formula:

$$w[n] = \begin{cases} 0.5 - 0.5\cos(\frac{2\pi n}{N}) & -\frac{N}{2} \leq n \leq \frac{N}{2} \\ 0 & \text{o.w} \end{cases}$$

4.1.1 Algorithm

By definition, the Short-Time Fourier transform pair is given as follows:

$$X_{STFT}[m,n] = \sum_{k=0}^{L-1} x[k]g[k-m]e^{-i2\pi nk/L}$$

$$x[k] = \sum_{m}\sum_{n} X_{STFT}[m,n]g[k-m]e^{i2\pi nk/L}$$

Where, $x[k]$ denotes a signal and $g[k]$ denotes an L-point window function. So, the computation is simply done by applying the window function in time domain and then taking the fast Fourier transform of the temporal divisions of the resultant sequence.

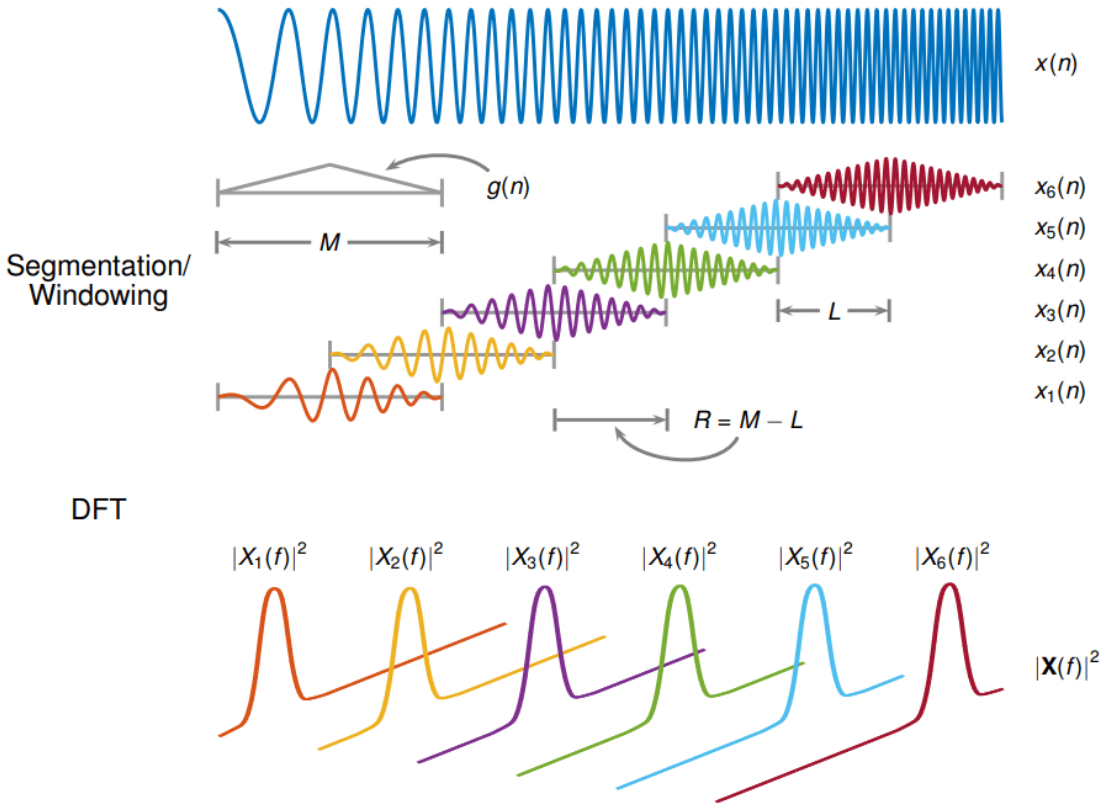


Figure 2: Short time fourier transform

4.1.2 Implementation

The input audio file was first converted into a NumPy array using the librosa.load() function from the Librosa library. The audio file was converted to a NumPy array. For implementation of STFT, we used the stft() function from the Librosa library we transformed this NumPy array (consisting of amplitudes in the time domain) to the frequency domain. We then use hard thresholding to try to filter out the high frequency noise components.

4.1.3 Other applications

In our application, we used it to analyze speech signal for efficient denoising. It is highly popular in audio engineering, since audio signals, which are non-stationary can be effectively analyzed with the help of Short- Time Fourier transform.
Apart from denoising of speech signals, it is also used for analyzing music signals and filtering of them as well. This information can be used for tuning audio effects, which is a usual practice during music compositions. It can be utilized in almost every audio processing system.

4.1.4 Advantages and Disadvantages

Short-Time Fourier transform, being one of the earliest time dependent transforms, can be used in analysis of non-stationary signals as it is the one of the popular transforms in audio engineering, it’s usage in our application seemed essential and being highly documented helped a lot during its implementation.

But there is a trade-off involved between the between temporal and frequency dimensions, i.e., if you get a finer resolution in time domain, you lose resolution in the frequency domain (uncertainty principle in signal processing). This factor makes it less efficient for analyzing non-periodic signals with fast-transient features (i.e., high frequency content within shirt duration). Its usage has also been limited to analysis of speech and music signals.

4.2 Deep Learning

We now introduce deep learning into the picture. Neural networks have been widely used in the past to tackle a variety of problems in the fields of audio processing, image processing & video processing. They provide us a way to possible learn a nonlinear mapping relationship between the inputs and the outputs. The function or the model is dynamic and its constituents parameters or weights keep on changing over time or trained as it encounters new inputs and learns new dependencies. The model is trained using a loss function that measures some form of the error between the expected output and the output of the network. As we decrease this error, the model keeps on becoming more and more accurate. This training is done with the help of an optimizer and a loss function.

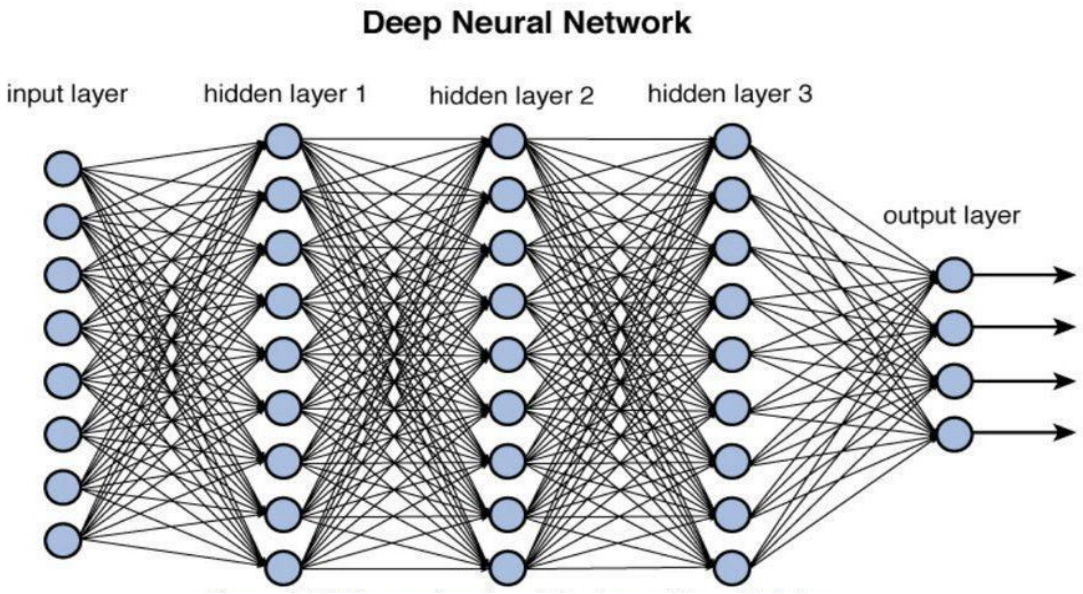
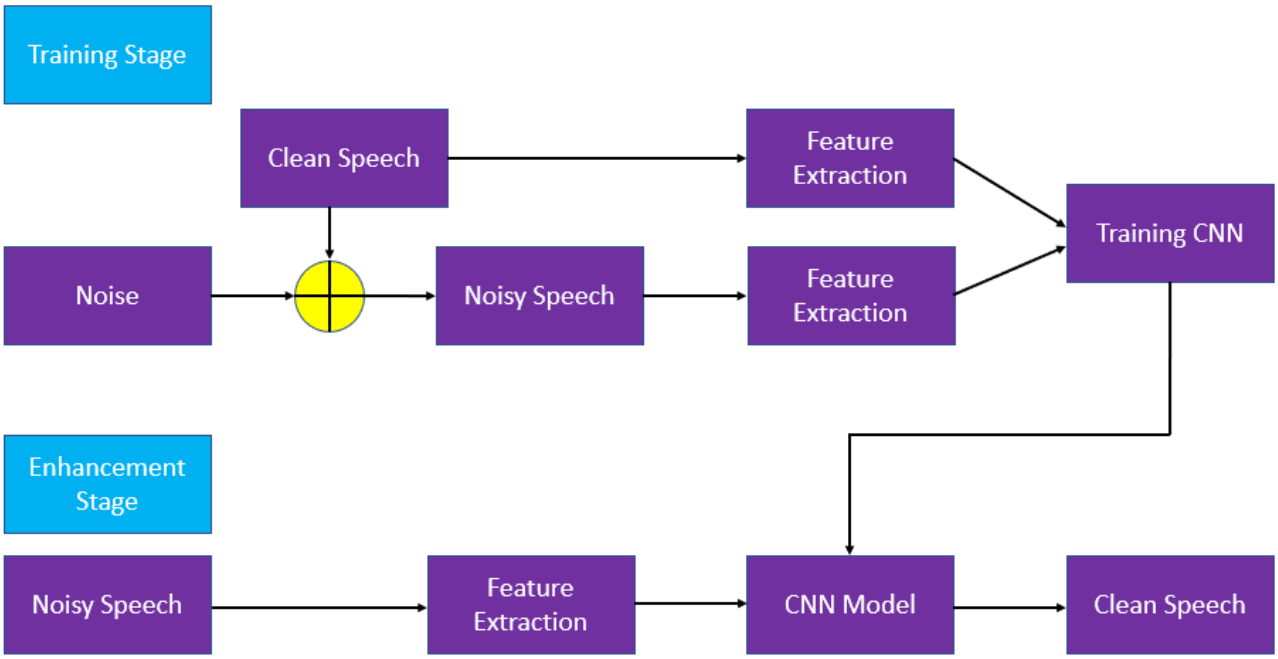


Figure 3: A deep neural network

4.2.1 Block Diagram



For training we are using two audio files, one clean audio and other same audio with noise to extract the necessary information for our CNN model.

Then we have noisy test audio files to be denoised with the help of information from training datasets to obtain noise free audio.

4.2.2 Speech denoising using 1D CNN

In this problem, we have implemented a 1D CNN that does speech denoising in the STFT magnitude domain. 1D CNN here means a variant of CNN which does the convolution operation along only one of the axis. In this case it's the frequency axis. Taking transpose of this STFT matrix, so that each row of the matrix acts as a spectrum.

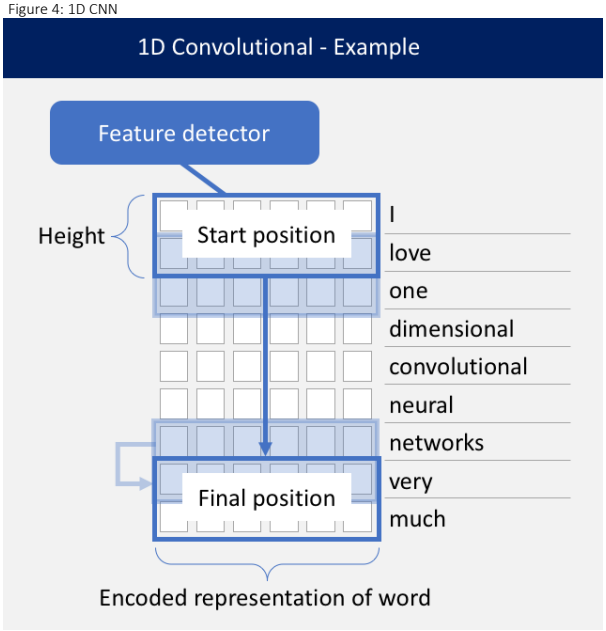
So, the 1D CNN takes one of these row vectors as an example coupled with the 'B' minibatch size. After deciding on the minibatch size, I defined 'K' kernels. Since the output matrix should be of size [B x 513], the dimensions of CNN was very high dimensional so I performed maxpooling and added a fully connected layer at the end to reduce the dimensionality and get the desired dimensions.

After training the model, I tested the performance of the model by calculating the Signal-to-Noise Ratio (SNR) value and performed ISTFT to check how the audio sounded.

1D CNN design implemented:

- Two convolution layers with filters 16 and 32 respectively.
- Also kernel sizes of 16, 8 respectively.
- Same padding is used.
- ReLU activation function is used in all the convolution layers.
- Max pooling layers are implemented one each after the convolution layer.
- Flattening is implemented for the last max pooling layer.
- A dense layer of 513 units with a ReLU activation.
- Adam optimizer, mean squared error loss function are used.
- 1000 epochs are used for training.

Figure 4: 1D CNN



"In this example for natural language processing, a sentence is made up of 9 words. Each word is a vector that represents a word as a low dimensional representation. The feature detector will always cover the whole word. The height determines how many words are considered when training the feature detector. In our example, the height is two. In this example the feature detector will iterate through the data 8 times."

4.2.3 Speech denoising using 2D CNN

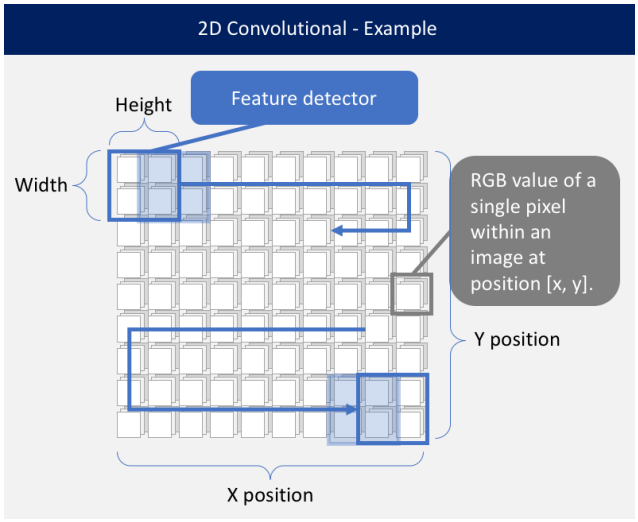
In this problem, since we are using a 2D CNN, we will consider both time and frequency of the audio. After applying STFT on the input audio files, I first take a matrix of [20 x 513] out of the entire STFT magnitude spectrogram (transposed). That's an input sample. Using this the 2D CNN estimates the cleaned-up spectrum that corresponds to the last (20th) input frame:

What it means is, the network takes all 20 current and previous input frames into account to predict the clean spectrum of the current frame, t + 19. Likewise, the next maxtrix will be another 20 frames shifted by one frame such that 19 frames overlap with the previous one. Since the original STFT spectrogram has 2,459 frames, we can create 2,440 such images as the training dataset. Since we are using a 2D CNN, the kernel size will also be in 2D such that both the width (frequencies) and the height axes (frames) be larger than 1. Once the model is built, we can test its performance on the test audio by calculating SNR and listening to check if the noise is removed or not.

2D CNN design implemented:

- Input layer with Tensor Shape as (-1,20,513,1).
- Two convolution layers used with filters of 16,32 are used respectively.
- Also kernel size of (4,4) for all of the above layers.
- ReLU activation is used in all the convolution layers.
- Max pooling layers are used after each convolution layer, with pool_size of (2,2).
- Final max pooling layer is flattened.
- A dense layer is used with 513 hidden units with ReLU activation function.
- Adam optimizer (0.0002 learning rate) and Mean squared error Loss function are used.
- Batch size of 64 and 500 epochs for training.

Figure 4: 2D CNN



"In this example for computer vision, each pixel within the Image is represented by its x· and y position as well as three values (RGB). The feature detector has a dimension of 2 x 2 in our example. The feature detector will now slide both horizontally and vertically across the image.

5. Observations and Investigations

5.1 Comparison

Here, we compare the performances of speech denoising first computing stft then applying deep learning model by 1D CNN & 2D CNN.

Method	Accuracy (SNR)
1D CNN	17. 641 DB
2D CNN	14. 632 DB

Table 1: Performance of stft + deep learning-based method

6. Conclusion

Hence it is evident from the above results that using (1D CNN + stft) gives better result compared to (2D CNN + stft) for speech denoising by removal of noise from noisy speech and speech enhancement.
By having SNR of 17.64 DB by 1D CNN compared to 14.632 DB by 2D CNN.

7. Deliverables

Removal of noise from audio recordings made on mobile phones.

- Noise removal from pre-recorded online lectures
- Filtering noise from audio recorded on mobile phones
- Audio enhancement of old recordings, movies, etc.

Restoration of lost audio signals from corrupted media, old audio recordings, etc. Singing and recording audios, can then be filtered for removal of background noise. Noise free pre-recorded presentations for professionals, working from home during pandemic.

8. Contributions to Aatmanirbhar Bharat Initiative

We observed that most of the speech denoising software such as audacity (the US), wavepad (Australia), Sony Creative Noise Reduction (Japan), Solicall (Israel), etc. have been made and developed outside India. We did not come across any indigeneous (and cheap, open-source) speech noise removal software. Hence this will be our contribution.

9. References and Acknowledgements References

1. P. C. Loizou, Speech Enhancement: Theory and Practice, 2nd ed. CRC Press, 2013.
2. M. Bosi and R. E. Goldberg, Introduction to Digital Audio Coding and Standards. Springer, 2002.
3. F. G. Germain, Q. Chen, and V. Koltun. Speech Denoising with Deep Feature Losses. 2018.
4. A. Kumar and D. Florencio: Speech Enhancement in Multiple Noise Conditions using Deep Neural Networks. 2016.
5. Shi, Y., Rong, W., & Zheng, N. (2018). Speech Enhancement using Convolutional Neural Network with Skip Connections. 2018 11th International Symposium on Chinese Spoken Language Processing (ISCSLP). "DOI":10.1109/iscslp.2018.8706591
6. Ouyang, Z., Yu, H., Zhu, W.-P., & Champagne, B. (2019). A Fully Convolutional Neural Network for Complex Spectrogram Processing in Speech Enhancement. ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). doi:10.1109/icassp.2019.8683423
7. L1 loss for speech denoising: https://arxiv.org/abs/1806.10522
8. Adaptive Batch Normalisation: https://arxiv.org/abs/1603.04779
9. Short Time Fourier Transform: https://www.sciencedirect.com/topics/engineering/short-time-fourier-transform
10. Figure1: NumPy & SciPy documentation.
11. Figure2: MATLAB /Short Time Fourier Transform
12. Figure3,4: Blog/Good Audience
13. https://ieeexplore.ieee.org/document/9057895
14. https://www.deeplearningbook.org/contents/convnets.html
15. https://www.dataversity.net/brief-history-deep-learning/#
16. https://www.ijarcce.com/upload/2016/si/SITES-16/IJARCCE-SITES%206.pdf
17. https://github.com/shaharpit809/Speech-Denoising-using-DNN-CNN-and-RNN/blob/master/CNN/1D-CNN.ipynb
18. https://drive.google.com/drive/folders/1plJFLtRjicWFF-wzKs2PFmmXunpkxsC3?usp=sharing
19. https://sthalles.github.io/practical-deep-learning-audio-denoising/
20. https://towardsdatascience.com/speech-enhancement-with-deep-learning-36a1991d3d8d

10. Tools and Languages used

In the project, we mainly utilized python programming language worked in Jupyter Notebook IDE and build neural network by TensorFlow 2 along with its libraries: Librosa, NumPy, SciPy, Matplotlib, Soundfile etc. All these libraries are open source.

11. Appendix

The project will be made open-source as a contribution to the community of Digital Signal Processing. The codes utilized by us are given below:

Code for speech denoising

```
#Importing Libraries
import librosa
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
from scipy.io import wavfile as wav
import IPython.display as ipd
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

#Loading training and testing files & Computing STFT on all the files
s, sr = librosa.load('train_clean_male.wav', sr=None)
S = librosa.stft(s, n_fft=1024, hop_length=512)

sn, sr = librosa.load('train_dirty_male.wav', sr=None)
X = librosa.stft(sn, n_fft=1024, hop_length=512)

x_test, sr = librosa.load('test_x_01.wav', sr=None)
X_test = librosa.stft(x_test, n_fft=1024, hop_length=512)

x_test2, sr = librosa.load('test_x_02.wav', sr=None)
X_test2 = librosa.stft(x_test2, n_fft=1024, hop_length=512)

#Calculation of magnitude of all the input files
mag_S = np.abs(S)
mag_X = np.abs(X)
mag_X_test = np.abs(X_test)
mag_X_test2 = np.abs(X_test2)
```

```
#Defining model specifications
learning_rate = 0.0002
num_epochs = 1000

input = tf.placeholder(tf.float32, [None, 513])
labels = tf.placeholder(tf.float32, [None, 513])
```

Speech Denoising by 1D CNN

```
def getModel(x):
# Input Layer
input_layer = tf.reshape(x, [-1, 513, 1])

# Convolutional Layer #1
conv1 = tf.layers.conv1d(
inputs=input_layer,
filters=16,
kernel_size=16,
padding="same",
activation=tf.nn.relu)

# Pooling Layer #1
pool1 = tf.layers.max_pooling1d(inputs=conv1, pool_size=2, strides=2)

# Convolutional Layer #2 and Pooling Layer #2
conv2 = tf.layers.conv1d(
inputs=pool1,
filters=32,
kernel_size=8,
padding="same",
activation=tf.nn.relu)
 pool2 = tf.layers.max_pooling1d(inputs=conv2, pool_size=2, strides=2)

# Dense Layer
pool2_flat = tf.layers.flatten(pool2)
logits = tf.layers.dense(inputs=pool2_flat, units=513, activation=tf.nn.relu)
return logits
output = getModel(input)
#Defining the loss function along with its optimizer
loss = tf.reduce_mean(tf.square(output - labels))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

```
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
count = 0
batch_size = 100
flag = True
while flag:
size = 0
#Mini batching with the given batch size
for i in range(0 , 2459, batch_size):
size += batch_size
if size <= 2459:
batch_x = mag_X[:,i : size]
batch_y = mag_S[:,i : size]
else:
batch_x = mag_X[:,i : 2459]
batch_y = mag_S[:,i : 2459]
feed_dict = {input: batch_x.T, labels: batch_y.T}
train_step.run(feed_dict=feed_dict)
if count%2 == 0:
loss_calc = loss.eval(feed_dict=feed_dict)
print("Epoch %d, loss %g"%(count, loss_calc))

#Once all the epochs are completed, training is stopped
if count >= num_epochs:
flag = False
count+=1

#Calculating the output from the given input, trained model and layer number
def feedforward(input_data, dnn_output):
output = dnn_output.eval(feed_dict = {input : input_data})

return output

#Recovering the complex values of the file from the output of the model
def recover_sound(X , mag_X , mag_output):
temp = X / mag_X
s_hat = temp * mag_output
return s_hat
```

```
#Computing the output from the model for both the test files
s_hat_test1 = feedforward(mag_X_test.T , output)
s_hat_test2 = feedforward(mag_X_test2.T , output)

#Recovering the complex values of both the test files
s_hat1 = recover_sound(X_test , mag_X_test , s_hat_test1.T)
s_hat2 = recover_sound(X_test2 , mag_X_test2 , s_hat_test2.T)

#Reconstructing the test files after removing noise
recon_sound = librosa.istft(s_hat1 , hop_length=512 , win_length=1024)
sf.write('test_s_01_recons_q1.wav',  recon_sound, sr)

# data, samplerate = sf.read('existing_file.wav')
recon_sound2 = librosa.istft(s_hat2 , hop_length=512 , win_length=1024)
sf.write('test_s_02_recons_q1.wav',recon_sound2, sr)

#For testing purpose, feeding the model with train_dirty_male file
#From the output generated, reconstructing the audio file
s_hat_test3 = feedforward(mag_X.T , output)
s_hat3 = recover_sound(X, mag_X , s_hat_test3.T)
recon_sound3 = librosa.istft(s_hat3 , hop_length=512 , win_length=1024)
size_recon_sound3 = np.shape(recon_sound3)[0]

#Once the audio file is generated, calculating the SNR value
s = s[: size_recon_sound3]
num = np.dot(s.T , s)
den = np.dot((s - recon_sound3).T,(s - recon_sound3))
SNR = 10 * np.log10(num/den)
print('Value of SNR : ' + str(SNR))
```

Speech Denoising by 2D CNN

```
def getModel(x):
# Input Layer
Input_layer = tf.reshape(x, [-1, 20, 513, 1])

# Convolutional Layer #1
conv1 = tf.layers.conv2d(
inputs=input_layer,
```

```
filters=16,
kernel_size=[4,4],
padding="same",
activation=tf.nn.relu)

# Pooling Layer #1
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2,2], strides=[2,2])

# Convolutional Layer #2 and Pooling Layer #2
conv2 = tf.layers.conv2d(
inputs=pool1,
filters=32,
kernel_size=[2,2],
padding="same",
activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2,2], strides=[2,2])

# Dense Layer
pool2_flat = tf.layers.flatten(pool2)
logits = tf.layers.dense(inputs=pool2_flat, units=513, activation=tf.nn.relu)
return logits
def transform_data(x , size , window_size):
temp = x[0 : 0 + window_size,:]
for i in range(1 , size - window_size + 1):
temp_mini = x[i : i + window_size,:]
temp = np.vstack((temp , temp_mini))
return temp

#Transforming the data in such a way that it takes 20 current and previous input frames
transformed_x = transform_data(mag_X.T , np.shape(mag_X.T)[0] , window_size)

#Keeping a copy of transformed x because we will require it later on to calculate the SNR
transformed_x1 = copy.deepcopy(transformed_x)

#Transforming the input data into 2D format
transformed_x = np.reshape(transformed_x , (2440 , 20 , 513))
```

```
#Dropping first 19 frames from y(clean wave) signal
transformed_y = (mag_S.T)[window_size - 1 : , :]

learning_rate = 0.0002
num_epochs = 500
batch_size = 64
window_size = 20

output = getModel(input)
#Defining the loss function along with its optimizer
loss = tf.reduce_mean(tf.square(output - labels))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(loss)

sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())

count = 0
flag = True

for count in tqdm(range(num_epochs)):
    size = 0
 #Mini batching with the given batch size
 for i in tqdm(range(0 , 2440, batch_size)):
 size += batch_size
  if size <= 2440:
  batch_x = transformed_x[i : size, :]
  batch_y = transformed_y[i : size, :]
  else:
  batch_x = transformed_x[i : 2440, :]
  batch_y = transformed_y[i : 2440, :]
   batch_x = batch_x.reshape((np.shape(batch_x)[0] * np.shape(batch_x)[1] ,
   np.shape(batch_x)[2]))
  feed_dict = {input: batch_x, labels: batch_y}
  train_step.run(feed_dict=feed_dict)
  if count%1 == 0:
  loss_calc = loss.eval(feed_dict=feed_dict)
  print("Epoch %d, loss %g"%(count+1, loss_calc))
```

```
#Calculating the output from the given input, trained model and layer number
def feedforward(input_data, dnn_output):
output = dnn_output.eval(feed_dict = {input : input_data})
return output

#Recovering the complex values of the file from the output of the model
def recover_sound(X , mag_X , mag_output):
temp = X / mag_X
s_hat = temp * mag_output
return s_hat

#Recovering the lost frames
def recover_data(x , size , value):
tmp = np.full(size , value)
output = np.vstack((temp , x))
return output

#Transforming the data in such a way that it be given to the model for testing
transformed_x_test = transform_data(mag_X_test.T , np.shape(mag_X_test.T)[0] ,
window_size)
transformed_x_test2 = transform_data(mag_X_test2.T , np.shape(mag_X_test2.T)[0] ,
window_size)


#Computing the output from the model for both the test files
s_hat_test1 = feedforward(transformed_x_test , output)
s_hat_test2 = feedforward(transformed_x_test2 , output)

#Recovering the first 19 frames that were lost
recovered_x_test1 = recover_data(s_hat_test1 , (window_size - 1 ,
np.shape(s_hat_test1)[1]) , 1e-15)
recovered_x_test2 = recover_data(s_hat_test2 , (window_size - 1 ,
np.shape(s_hat_test2)[1]) , 1e-15)

#Recovering the complex values of both the test files
s_hat1 = recover_sound(X_test , mag_X_test , recovered_x_test1.T)
s_hat2 = recover_sound(X_test2 , mag_X_test2 , recovered_x_test2.T)

#Reconstructing the test files after removing noise
recon_sound = librosa.istft(s_hat1 , hop_length=512 , win_length=1024)
sf.write('test_s_01_recons_q1.wav',  recon_sound, sr)
```

```
# data, samplerate = sf.read('existing_file.wav')
recon_sound2 = librosa.istft(s_hat2 , hop_length=512 , win_length=1024)
sf.write('test_s_02_recons_q1.wav',recon_sound2, sr)

#For testing purpose, feeding the model with train_dirty_male file
#From the output generated, reconstructing the audio file

s_hat_test3 = feedforward(transformed_x1 , output)
recovered_x1 = recover_data(s_hat_test3 , (window_size - 1 , np.shape(s_hat_test3)[1]) ,
1e-15)
s_hat3 = recover_sound(X, mag_X , recovered_x1.T)

recon_sound3 = librosa.istft(s_hat3 , hop_length=512 , win_length=1024)
size_recon_sound3 = np.shape(recon_sound3)[0]

#Once the audio file is gener#For testing purpose, feeding the model with
train_dirty_male file
#From the output generated, reconstructing the audio file

s_hat_test3 = feedforward(transformed_x1 , output)
recovered_x1 = recover_data(s_hat_test3 , (window_size - 1 , np.shape(s_hat_test3)[1]) ,
1e-15)
s_hat3 = recover_sound(X, mag_X , recovered_x1.T)

recon_sound3 = librosa.istft(s_hat3 , hop_length=512 , win_length=1024)
size_recon_sound3 = np.shape(recon_sound3)[0]ated, calculating the SNR value


s = s[: size_recon_sound3]
num = np.dot(s.T , s)
den = np.dot((s - recon_sound3).T,(s - recon_sound3))
SNR = 10 * np.log10(num/den)
print('Value of SNR : ' + str(SNR))
```

Code links: https://github.com/mohdshahbaz123/Speech-Denoising-using-Deep-Learning