A PROJECT REPORT ON

NOISE SUPPRESSION USING MACHINE LEARNING (U-Net)

Submitted in partial fulfillment of the requirement for the **VII & VIII** semester of

BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY

Submitted By:
ASHWINI JHA (17107215)
ANKUR TIWARY (17107214)
ABHISHEK KAMAL (17107211)

Under the supervision of
Prof. Anand Prakash Rawal & Prof. Suhel Ahamed
& Project Supervisor **Prof. Agnivesh Pandey**

TO

DEPARTMENT OF INFORMATION TECHNOLOGY,
SCHOOL OF STUDIES IN ENGINEERING & TECHNOLOGY,
GURU GHASIDAS VISHWAVIDYALAYA BILASPUR, INDIA
(CENTRAL UNIVERSITY)

SESSION 2020-2021

GURU GHASIDAS VISHWAVIDYALAYA BILASPUR , INDIA

CERTIFICATE

This is to certify that the project entitled "**NOISE SUPPRESSION USING MACHINE LEARNING**" carried out by "**Ashwini Jha**, **Ankur Tiwary** & **Abhishek Kamal** " under my supervision at department of Information Technology, School of Studies in Engineering & Technology, Guru Ghasidas Vishwavidyalaya, Bilaspur.

The work is original, as it has not been submitted earlier either in part or full for any purpose before.

Prof. Suhel Ahamed
(Assistant Professor
Department of IT)

Prof. Anand Prakash Rawal
(Assistant Professor
Department of IT)

DECLARATION

We, hereby declare that the work presented in this dissertation entitled "**NOISE SUPPRESSION USING MACHINE LEARNING**" has been done by us, and this dissertation embodies our own work.

Ashwini Jha
Ankur Tiwary
Abhishek Kamal

Approved By
Prof. Suhel Ahamed
Prof. Anand Prakash Rawal

Project Supervisor
(Prof. Agnivesh Pandey)

Head (IT)
(Dr. Rohit Raja)

ACKNOWLEDGEMENT

This project aims for **NOISE SUPPRESSION USING MACHINE LEARNING.** The project scope was very vast , but I've tried to do it to the best of my ability & knowledge. I would like to thank a few people who have helped me complete this project.

- My guide, *Prof. Suhel Ahamed & Prof. Anand Prakash Rawal* for directing me as to how to go about the project & helping me in figuring out different approaches for the problem at hand.

- *Dr. Rohit Raja*, Head of Information Technology Department, for giving us the opportunity to work on this project.

- All the esteemed faculty members, *Mr. Deepak Netam, Dr. Amit Khaskalam, Mr. Rajesh Mahule, Mr. Santosh Soni, Mr. Abhishek Jain, Mr. Agnivesh Pandey, Mr. Pankaj Chandra, Mr. Suhel Ahamed, Mr. Anand Rawal & Mrs. Akansha Gupta, Mrs. Aradhana Soni, Mr. Amit Kumar Dewangan* for their continued support & effort.

- The Dean of Institute of Technology, Guru Ghasidas Vishwavidyalaya, *Dr. T.V. Arjunan*, for providing all the necessary facilities.

- My *friends*, for all the fruitful discussions on this topic, which proved very helpful.

- Last but not the least my *parents*, for their constant support & encouragement.

I would like to thank all those who spared their valuable time to discuss & helped me present the project in its current shape & form.

Ashwini Jha
Ankur Tiwary
Abhishek Kamal

PREFACE

In every chain of reasoning, the evidence of the last conclusion can be no greater than that of the weakest link of the chain, whatever may be the strength of the rest.

-Thomas Reid

This project aims at building a noise suppression system using machine learning.

In recent years, single-channel speech enhancement has attracted a considerable amount of research attention because of the growing challenges in many important **real-world applications**, including **mobile speech communication**, **hearing aids design** and **robust speech recognition**. The goal of noise suppression is to improve the **intelligibility** and **quality** of a noisy speech signal degraded in adverse conditions.

This project presents how **machine learning (U-NET)** can be applied to **noise suppression**. The main idea is to combine classic signal processing with deep learning to create a **noise suppression algorithm** that's small and fast.

NOISE SUPPRESSION

Noise suppression is a pretty old topic in speech processing. But suppressing background noise using Machine Learning would be a challenging task. As the name implies, the idea is to take a noisy signal and remove as much noise as possible while causing minimum distortion to the speech of interest.

TWO FUNDAMENTAL NOISE TYPES are : Stationary & Non - Stationary

The project report goes in detail about implementation using one of the known methods which is U-NET. It covers the different models & functions used to suppress noise. Code snippets where possible have been inserted to help understand the entire implementation easily.

The project has been primarily done in **Python** Language. Scientific , **Machine learning libraries Tensorflow, librosa, matplotlib & keras** have been used to keep implementation accurate, concise & readable.

The project can be best described as a proof of the concept & a tool to validate between the true clean voice & the predicted denoised voice & calculate the loss & find the required spectrograms.

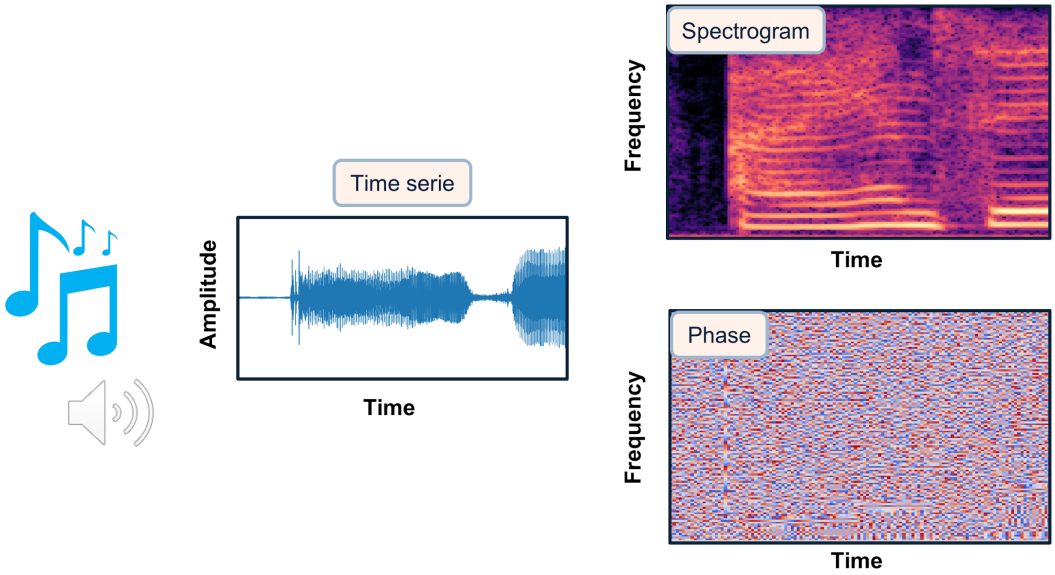
CONTENTS

Chapter 1

INTRODUCTION

Audios have many different ways to be represented, going from raw time series to time-frequency decompositions. The choice of the representation is crucial for the performance of your system. Among time-frequency decompositions, Spectrograms have been proved to be a useful representation for audio processing. They consist of 2D images representing sequences of Short Time Fourier Transform (STFT) with time and frequency as axes, and brightness representing the strength of a frequency component at each time frame. In such a way they appear a natural domain to apply the CNNS architectures for images directly to sound. Between **magnitude and phase spectrograms**, magnitude spectrograms contain most of the structure of the signal. Phase spectrograms appear to show only little temporal and spectral regularities.

In this project, magnitude spectrograms have been used as a representation of sound (image below) in order to predict the noise model to be subtracted to a noisy voice spectrogram.



1.1 Objectives

There are several objectives or benefits of using the above proposed system :

The **objective** of this project is to suppress the noise or background noise & to find the loss & to predict the noise model to be subtracted to a noisy voice spectrogram. Noise suppression using Machine Learning, could be implemented & be perfect for

Individuals

- **Professionals**
- **Online teachers**
- **Podcasters**

Business

- **Remote Teams**
- **Enterprises**
- **Call Centers**

1.2 Application of the project

The noise suppression technology could be applied to different devices be it the microphone, headset, speaker & also to different apps used for **Conferencing,**

Voice Messaging, Streaming & Recording. It could be useful for individuals & **businesses Professionals, Online Teachers, Podcasters, Remote Teams, Enterprises & Call Centers**.
In such **pandemic situations** this **Background Noise Suppression** could be quite helpful.

1.3 Important Terminologies

1.3.1 Representation of Audio

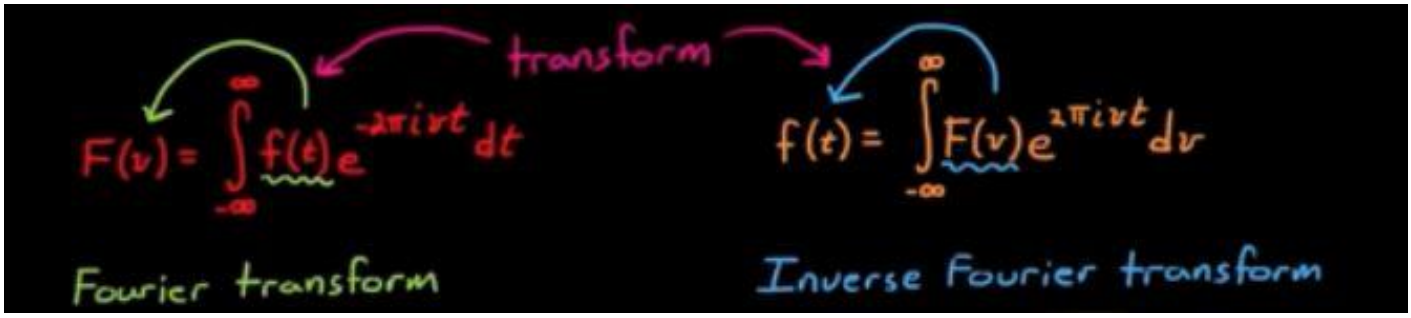
Audios have many different ways to be represented, going from **raw time series** to **time-frequency decompositions**. The choice of the representation is crucial for the performance of your system. Among time-frequency decompositions, **Spectrograms** have been proved to be a useful representation for audio processing.

1.3.2 Spectrograms

They consist of 2D images representing sequences of **Short Time Fourier Transform (STFT)** with time and frequency as axes, and **brightness** representing the strength of a frequency component at each time frame.

- **Magnitude spectrograms** contain most of the structure of the signal.
- **Phase spectrograms** appear to show only little temporal and spectral regularities.

1.3.3 Short Time Fourier Transform & Inverse Short Time Fourier Transform



The **Short-time Fourier transform** (**STFT**), is a **Fourier**-related **transform** used to determine the sinusoidal frequency and phase content of local sections of a

signal as it changes over **time**.
Fourier transform is a perfect representation in the frequency domain of the time domain signal & one of the implications of that lossless transformation is that one can use a similar operation to get back from the frequency domain into the time domain; that operation is called the **Inverse Fourier Transform**.

1.4 Preparing the Dataset

To create the datasets for training, english speech, clean voices and environmental noises have been gathered from different sources.

The clean voices were mainly gathered from **LibriSpeech**: an ASR corpus based on public domain audio books. The environmental noises were gathered from the **ESC-50** dataset .

The ESC-50 dataset is a labeled collection of 2000 environmental audio recordings suitable for benchmarking methods of environmental sound classification. The dataset consists of 5-second-long recordings organized into 50 semantical classes (with 40 examples per class) loosely arranged into 5 major categories:

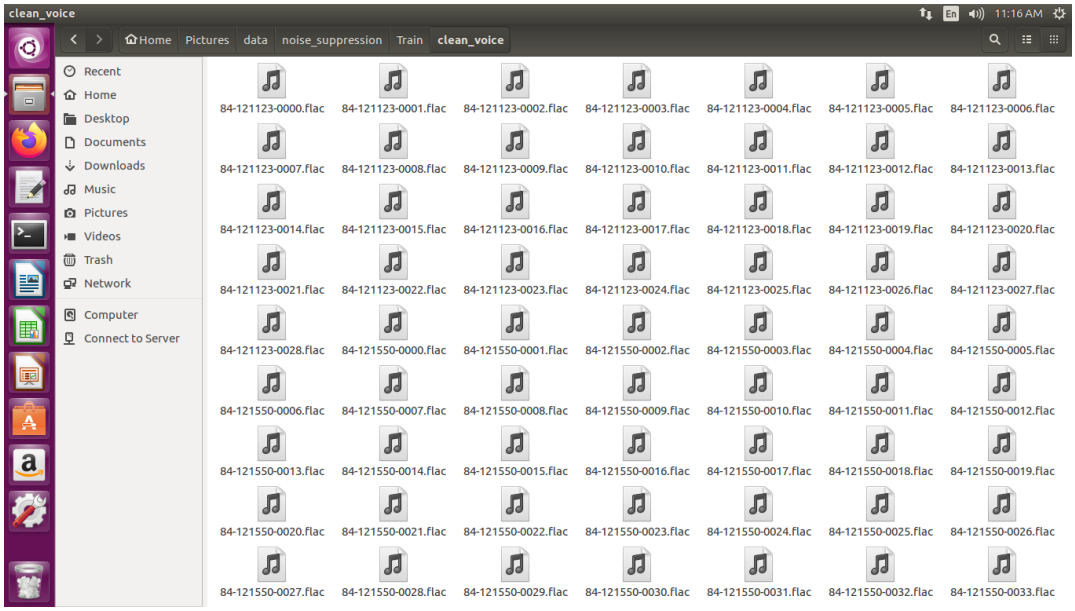
Animals	Natural soundscapes & water sounds	Human, non-speech sounds	Interior/domestic sounds	Exterior/urban noises
Dog	Rain	Crying baby	Door knock	Helicopter
Rooster	Sea waves	Sneezing	Mouse click	Chainsaw
Pig	Crackling fire	Clapping	Keyboard typing	Siren
Cow	Crickets	Breathing	Door, wood creaks	Car horn
Frog	Chirping birds	Coughing	Can opening	Engine
Cat	Water drops	Footsteps	Washing machine	Train
Hen	Wind	Laughing	Vacuum cleaner	Church bells
Insects (flying)	Pouring water	Brushing teeth	Clock alarm	Airplane
Sheep	Toilet flush	Snoring	Clock tick	Fireworks
Crow	Thunderstorm	Drinking, sipping	Glass breaking	Hand saw

For this project, **10 classes of environmental noise have been made: tic clock,**

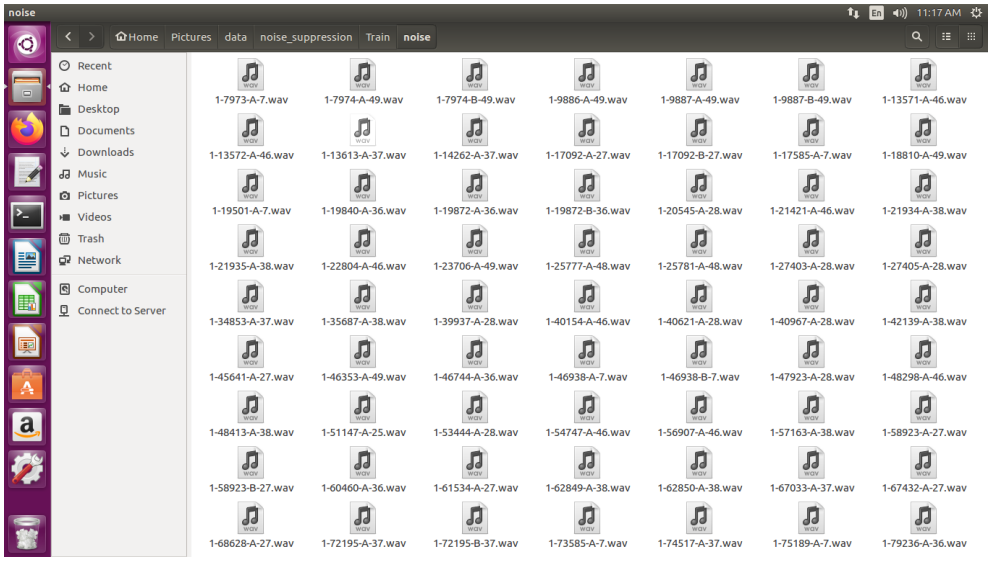
foot steps, bells, handsaw, alarm, fireworks, insects, brushing teeth, vacuum cleaner and snoring.

To create the datasets for training/testing, audios were sampled at 8kHz and windows slightly above 1 second were extracted. Noises have been blended to clean voices with a randomization of the noise level (between 20% and 80%), Some clean voices from clean_voice directory (538 .flac files) with some noises from noise directory (400 .wav files) are blended randomly and the spectrograms of noisy voices, noises and clean voices as well as complex phases, time series and sounds have been found in .npy format (numpy).Parameters used are: STFT, frame length, hop_length, number of frames for preparing the dataset.

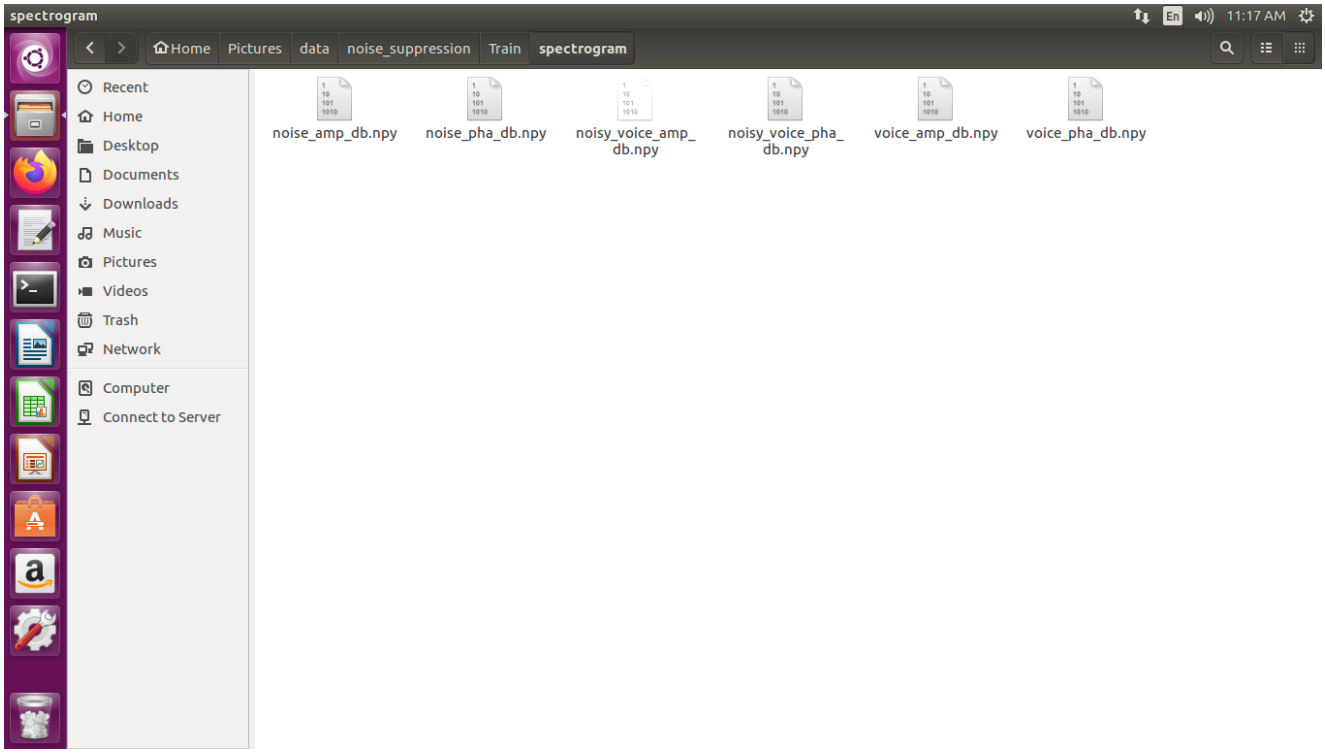
1.4.1 Clean_Voice directory (538 .flac files)



1.4.2 Noise directory (400 .flac files)



1.4.3 Spectrogram (.npy files) of noisy_voice, voice & noise



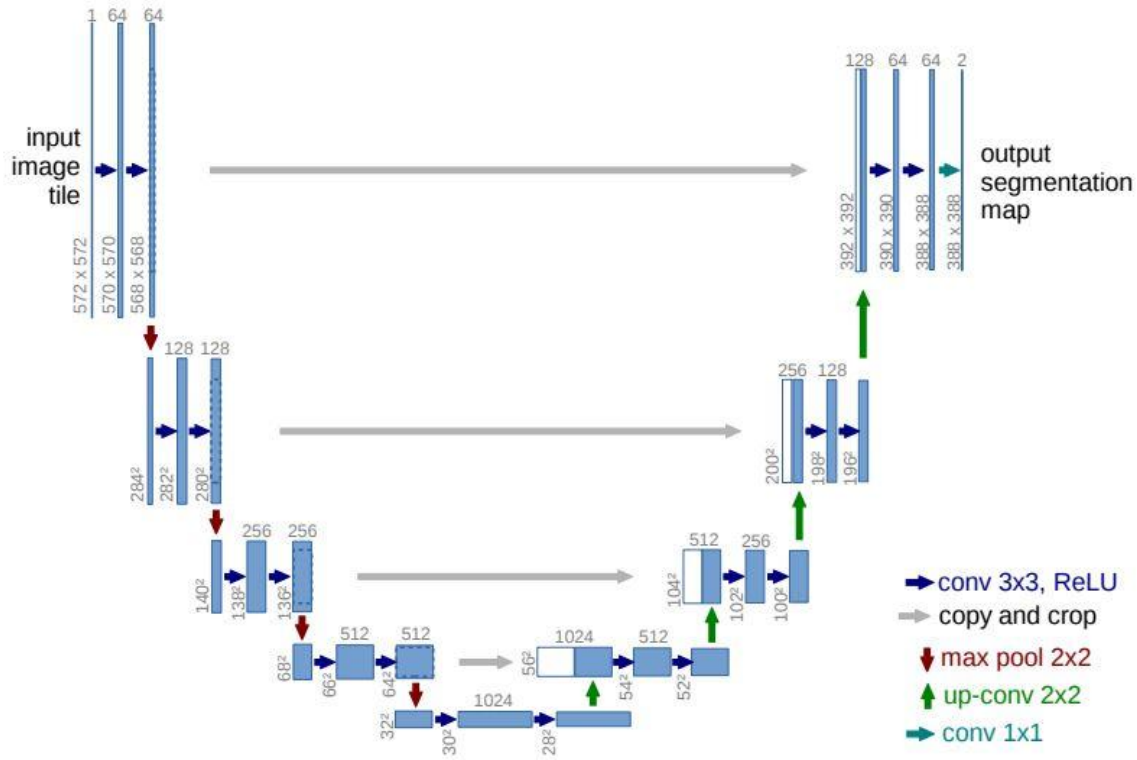
1.5 Proposed Solution

1.5.1 U-NET

U-Net, a Deep Convolutional Autoencoder with symmetric skip connection, was initially developed for Bio Medical Image Segmentation. Here the U-Net has been adapted to denoise spectrograms.

The encoder is made of **10 convolutional layers** (with LeakyReLU, max pooling and dropout). The decoder is a symmetric expanding path with skip connections. The last activation layer is a hyperbolic tangent (tanh) to have an output distribution between -1 and 1. For training from scratch the initial random weights were set with **He normal initializer**.

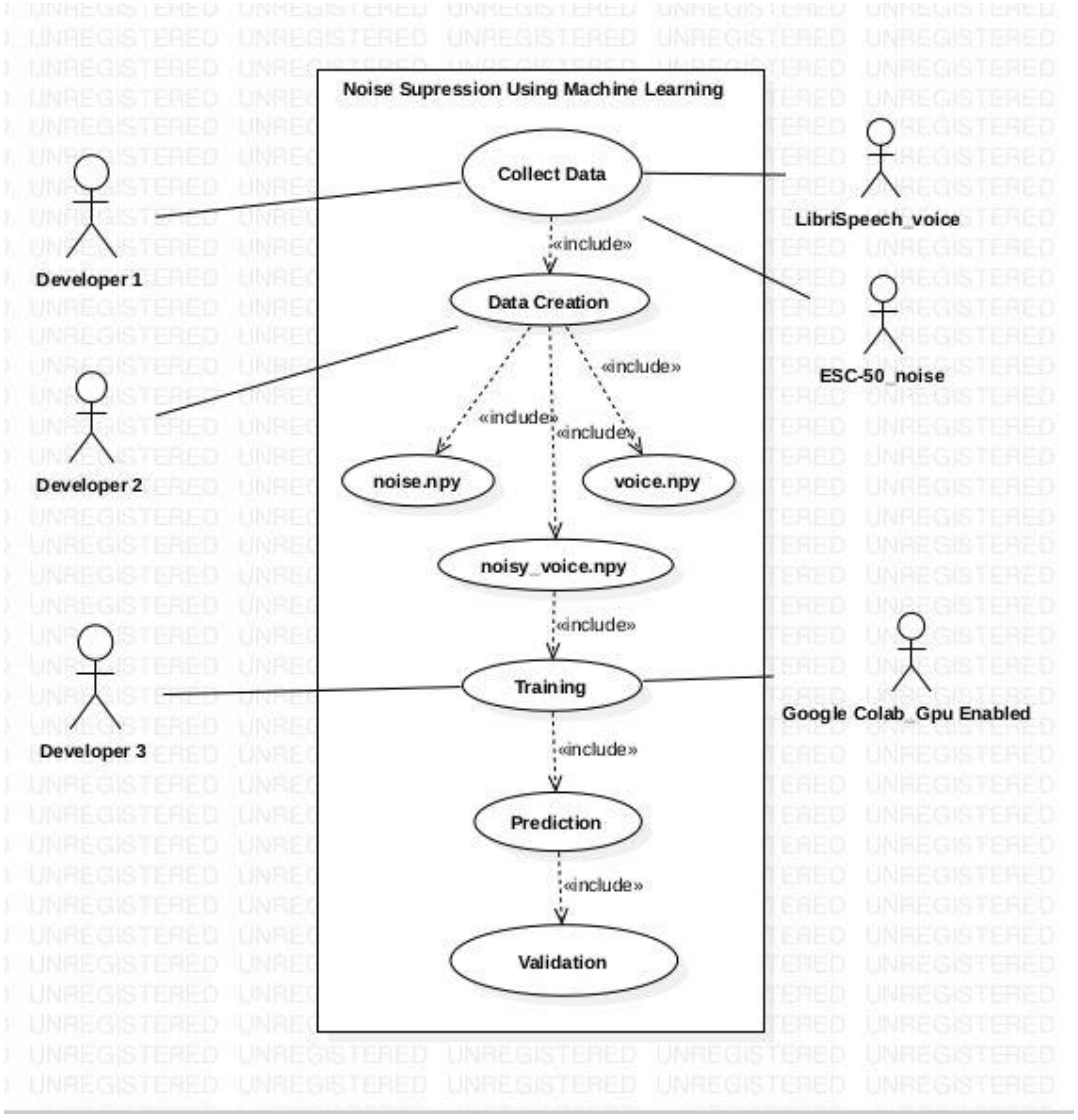
Model is compiled with an **Adam optimizer** and the loss function used is the Huber loss as a compromise between the L1 and L2 loss.



1.5.2 Network Architecture

The network consists of a **contracting path** and an **expansive path**, which gives it the u-shaped architecture. The contracting path is a typical convolutional network that consists of repeated application of convolutions, each followed by a rectified linear unit (ReLU) and a max pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive pathway combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path.

1.6 Methodology (USE CASE DIAGRAM)



Chapter 2

TRAINING

2.1 Dataset used for Training

Datasets to be used for training will be magnitude spectrograms of noisy voices and magnitude spectrograms of clean voices. Each window is converted into a spectrogram matrix of size 128 x 128 before training, also the number of frames is 100.
As input to the network, the magnitude spectrograms of the noisy voices. And as output the Noise to model (noisy voice magnitude spectrogram - clean voice magnitude spectrogram). Both input and output matrices are scaled with a global scaling to be mapped into a distribution between -1 and 1.

2.2 Training on Google Colab

Colaboratory, or 'Colab' for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

2.2.1 Using GPU

For this project a free GPU available at Google colab is used for training. To enable GPU one has to Change Run time type or Notebook Setting to **GPU** which is the Hardware Accelerator.
A modern GPU takes a couple of hours. For training model_unet.h5 is used with epochs = 9 & batch size = 80. The best model is saved & weights are saved regularly & reloaded for next training.

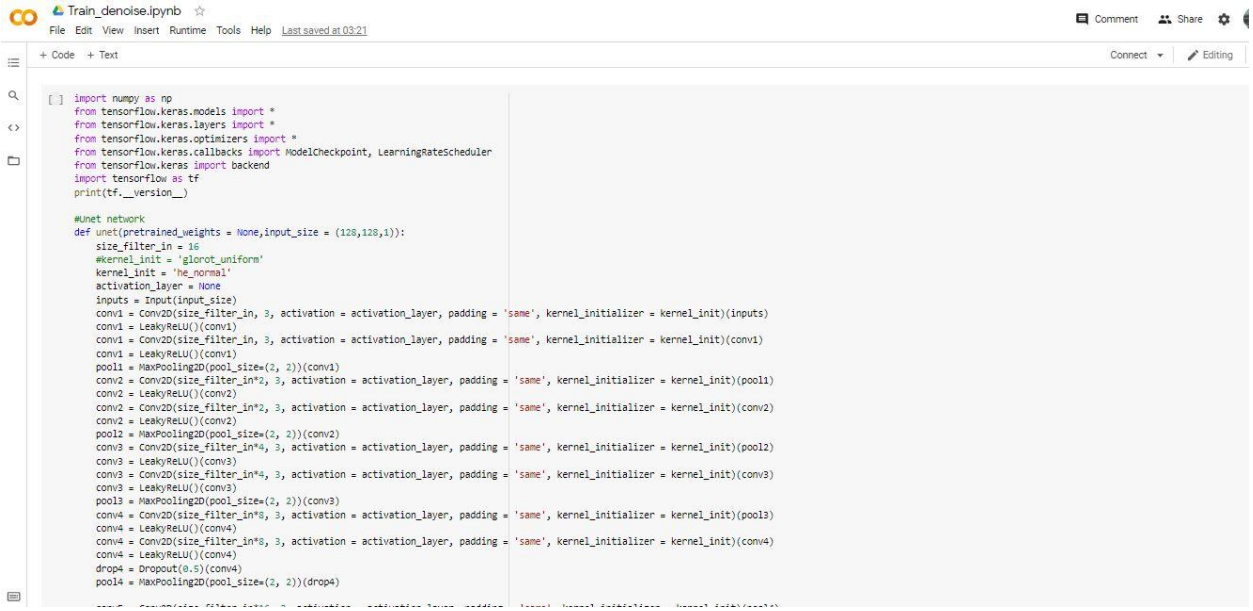
2.3 He Normal

For training from scratch the initial random weights were set with He normal

initializer. It draws samples from a truncated normal distribution centered on 0 with stddev = sqrt(2 / fan_in) where fan_in is the number of input units in the weight tensor.

2.4 Code Snippets:

U-Net Model




```
conv5 = Conv2D(size_filter_in*16, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(pool4)
conv5 = LeakyReLU()(conv5)
conv5 = Conv2D(size_filter_in*16, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(conv5)
conv5 = LeakyReLU()(conv5)
drop5 = Dropout(0.5)(conv5)

up6 = Conv2D(size_filter_in*8, 2, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(UpSampling2D(size = (2,2))(drop5))
up6 = LeakyReLU()(up6)
merge6 = concatenate([drop4,up6], axis = 3)
conv6 = Conv2D(size_filter_in*8, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(merge6)
conv6 = LeakyReLU()(conv6)
conv6 = Conv2D(size_filter_in*8, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(conv6)
conv6 = LeakyReLU()(conv6)
up7 = Conv2D(size_filter_in*4, 2, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(UpSampling2D(size = (2,2))(conv6))
up7 = LeakyReLU()(up7)
merge7 = concatenate([conv3,up7], axis = 3)
conv7 = Conv2D(size_filter_in*4, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(merge7)
conv7 = LeakyReLU()(conv7)
conv7 = Conv2D(size_filter_in*4, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(conv7)
conv7 = LeakyReLU()(conv7)
up8 = Conv2D(size_filter_in*2, 2, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(UpSampling2D(size = (2,2))(conv7))
up8 = LeakyReLU()(up8)
merge8 = concatenate([conv2,up8], axis = 3)
conv8 = Conv2D(size_filter_in*2, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(merge8)
conv8 = LeakyReLU()(conv8)
conv8 = Conv2D(size_filter_in*2, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(conv8)
conv8 = LeakyReLU()(conv8)

up9 = Conv2D(size_filter_in, 2, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(UpSampling2D(size = (2,2))(conv8))
up9 = LeakyReLU()(up9)
merge9 = concatenate([conv1,up9], axis = 3)
conv9 = Conv2D(size_filter_in, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(merge9)
conv9 = LeakyReLU()(conv9)
conv9 = Conv2D(size_filter_in, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(conv9)
conv9 = LeakyReLU()(conv9)
conv9 = Conv2D(2, 3, activation = activation_layer, padding = 'same', kernel_initializer = kernel_init)(conv9)
conv9 = LeakyReLU()(conv9)
conv10 = Conv2D(1, 1, activation = 'tanh')(conv9)
```

```
    model = Model(inputs,conv10)

    model.compile(optimizer = 'adam', loss = tf.keras.losses.Huber(), metrics = ['mae'])

    #model.summary()

    if(pretrained_weights):
      model.load_weights(pretrained_weights)

    return model
```

2.4.1

2.4.2 Output after training (Training loss & Validation Loss)

```
Epoch 00008: val_loss improved from 0.00851 to 0.00849, saving model to /content/gdrive/My Drive/Noise_Suppression/model-008-0.002680-0.008490.h5
Epoch 9/9
2/2 [==============================] - 0s 83ms/step - loss: 0.0026 - mae: 0.0484 - val_loss: 0.0078 - val_mae: 0.0899

Epoch 00009: val_loss improved from 0.00849 to 0.00780, saving model to /content/gdrive/My Drive/Noise_Suppression/model-009-0.002585-0.007797.h5
```

2.4.3 Plotting a training & validation loss graph made in one of the training using matplotlib

```
[ ]  #Plot training and validation loss
     from matplotlib import pyplot as plt
     loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs = range(1, len(loss) + 1)

     plt.plot(epochs, loss, label='Training loss')
     plt.plot(epochs, val_loss, label='Validation loss')
     plt.yscale('log')
     plt.title('Training and validation loss')
     plt.legend()
     plt.show()
```

Chapter 3

Evaluation Methodology

3.1 Calculation of loss

3.1.1 Huber Loss

The Huber loss is a loss function used in robust regression that is less sensitive to outliers in data than the squared error loss. A variant for classification is also sometimes used.
The Huber loss function describes the penalty incurred by an estimation procedure *f*. Huber (1964) defines the loss function piecewise by

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

This function is quadratic for small values of *a*, and linear for large values, with equal values and slopes of the different sections at the two points where |*a*|= *delta*
The variable *a* often refers to the residuals, that is to the difference between the observed and predicted values *a* = y- *f(x),* so the former can be expanded to

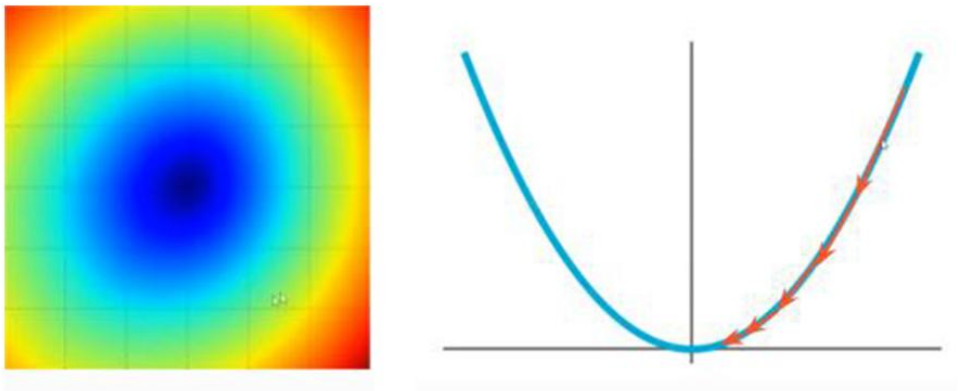
$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

3.1.2 Weight Updation

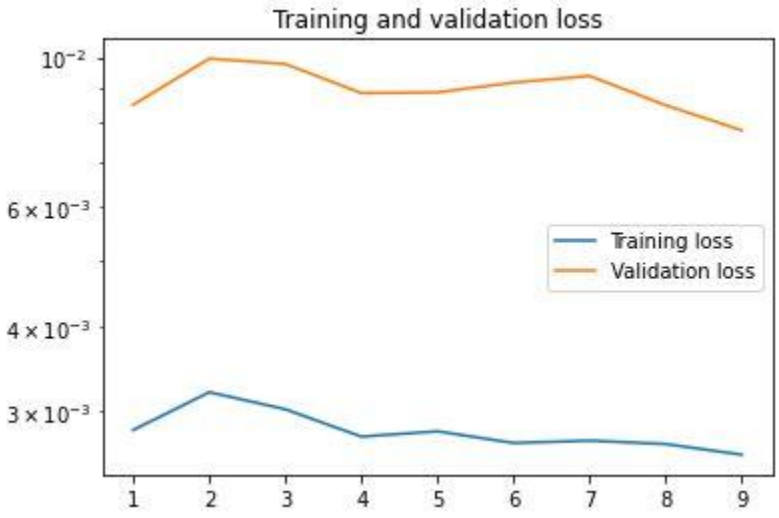
Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. The algorithm is called Adam.

3.1.2.1 Adam Optimizer

Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.



At the end, I obtained a training loss of 0.0026 and a validation loss of 0.0078. **Below a loss graph made in one of the training.**



Chapter 4

Prediction & Validation

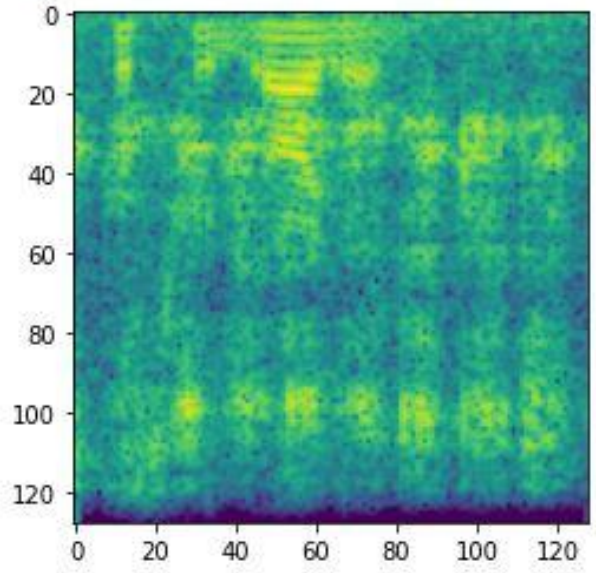
For prediction, some of the noisy voice audios are converted into numpy time series of windows slightly above 1 second. Each time series is converted into a magnitude spectrogram and a phase spectrogram via STFT transforms. Noisy voice

spectrograms are passed into the U-Net network that will predict the noise model for each window.
Then the model is subtracted from the noisy voice spectrogram. The "denoised" magnitude spectrogram is combined with the initial phase as input for the inverse Short Time Fourier Transform (ISTFT).

```
[ ]  #Predict and QC the output
     X_pred_test = generator_nn.predict(X_test)
```

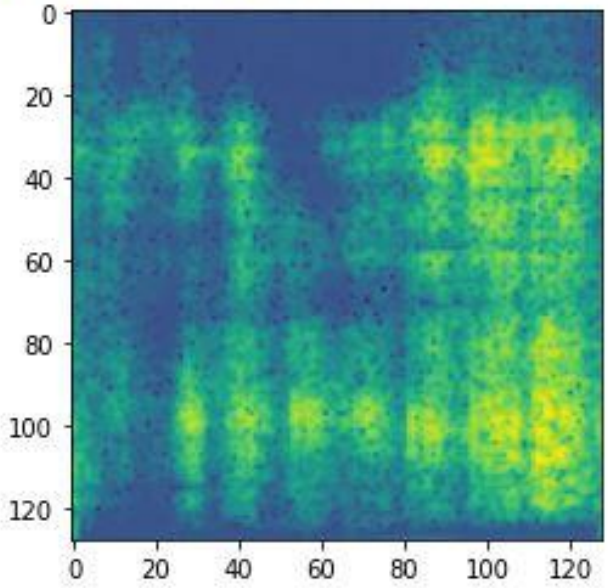
Spectrogram voice + noise

```
<matplotlib.image.AxesImage at 0x7fec24e47cf8>
```



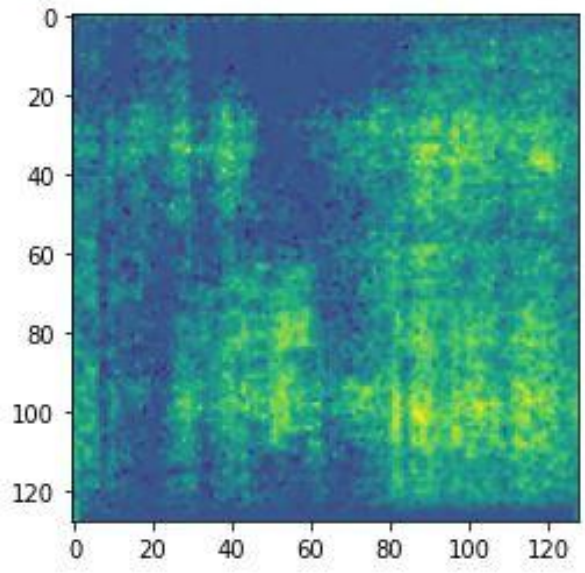
Spectrogram predicted noise

```
<matplotlib.image.AxesImage at 0x7fec24c48908>
```



Spectrogram true voice

```
<matplotlib.image.AxesImage at 0x7fec24aa0898>
```



Chapter 5

5.1 LIMITATIONS

1. A very large amount of dataset is required to train the model.
2. More data & training can lead to a lower value of training loss & validation loss.
3. Real Time Noise Suppression does not take place.

5.2 APPROACHES

Some possible approaches that can be used for the suppression of noise are:

- RNN (Recurrent Neural Network)
- A Hybrid DSP

5.3 CONCLUSION

This project is proposed to build a noise suppression technology using the U-Net model, to suppress background noise. The experiments were conducted using the U-NET model on one benchmark datasets including LibriSpeech for clean voices & ESC-50 dataset for environmental noise, and performance was evaluated using different performance metrics. The performance of the proposed method was evaluated via sample rate, window size, number of frames, epochs, batch size, he normal, huber loss, adam optimizer . A different criteria such as scaling factor between 1 & -1 is also considered, resized sample inputs & outputs (128, 128). In addition, this method shows 0.0026 of training loss & 0.0078 of validation loss after training on the blended dataset in numpy format, which is significantly better than that of any other ML based approach performance respectively. Evaluation of the performance of the proposed method with U-NET. Thus, the experimental results show state-of-the-art suppressing noise using machine learning.

5.4 FUTURE WORK

- Real Time Noise Suppression.
- Using GPUs for good results like the single Nvidia 1080 ti could scale up to 1000 streams without any optimizations.
- Using outbound & inbound noise suppression, imagine that you want to suppress both your mic signal (*outbound noise*) and the signal coming to your speakers (*inbound noise*) from all participants; this concept is applicable.
- Noise suppression on the edge device, which means noise suppression is bound to the microphone. You get the signal from mic(s), suppress the noise, and send the signal upstream.
- Play with more machine learning methods (RNN or Hybrid DSP) to get lower training & validation loss.

5.5 REFERENCES

- https://arxiv.org/pdf/1505.04597.pdf
- https://arxiv.org/pdf/1709.08243.pdf
- https://ejhumphrey.com/assets/pdf/jansson2017singing.pdf
- https://dl.acm.org/doi/10.1145/2733373.2806390