BRAUDE
College of Engineering, karmiel

Software Engineering Department

Capstone Project Phase B –61999

Speech Enhancement Using U-Net with Compressed Sensing

24-1-R-11

Submitters:
Avihay Hadad – 209286665
Elad Fisher – 318882800

Supervisors:
Prof. Zeev Volkovich
Dr. Renata Avros

Table of Contents

ABSTRACT

This paper presents an approach to speech enhancement task based on deep neural networks, specifically focusing on a time-domain U-Net architecture. While traditional methods achieved good denoising performance, they often ignored the contextual information and detailed features present in input speech signals. To address this, our model includes a time-domain U-Net which combines lightweight Shuffle Attention mechanism for focusing on features of speech and suppressing irrelevant audio information, and a compressed sensing loss (CS loss) by using the measurements of clean speech and enhanced speech, that can further remove noise from noisy speech [9].

Our proposed approach offers a promising solution for speech enhancement task, by achieving high speech quality and intelligibility scores with low number of parameters. Furthermore, it demonstrates good performance in generalization, effectively handling scenarios where the input speech contains new and unfamiliar noise types.

Our project files and source code will be available at: GitHub

1. INTRODUCTION

Speech enhancement is very important subject in the field of speech processing. Its goal is to improve the quality and intelligibility of speech which disturbed by external noises. There are many traditional algorithms and cutting-edge deep learning models that were designed to solve that task. However, many of them were too complex and they had high number of parameters or their denoising performance were mediocre.

Inspired by the Attention Wave-U-Net for speech enhancement [7]; new research and deep learning models of Compress Sensing, we propose a time-domain U-Net model combining lightweight Shuffle Attention mechanism and CS loss [9]. The U-Net model pays attention to the speech contextual information and prevent the detailed features from being lost during down-sampling or up-sampling. The lightweight Shuffle Attention is present in the final output of the encoder for focusing on detailed features of speech and suppressing irrelevant information after down-sampling. The CS loss function is defined using the measurement of speech, which can further remove noise in noisy speech.

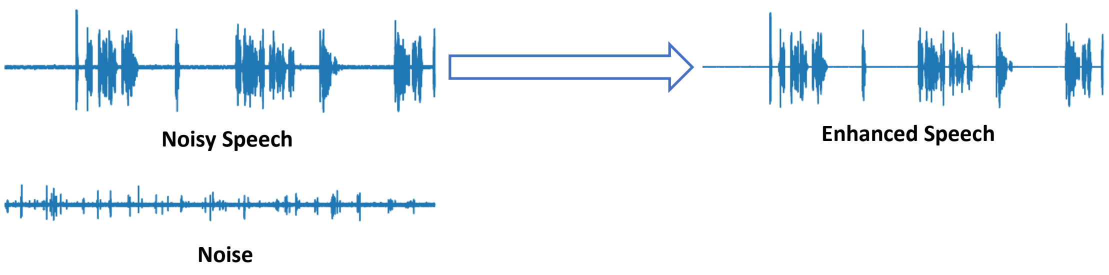


Figure 1. Speech enhancment example [13].

The proposed model can be utilized in various applications. It can enhance audio recordings in editing software, improve accuracy in speech recognition systems, and can even optimize voice calls in mobile communication devices. Furthermore, its potential to enhance speech intelligibility makes it valuable for individuals using hearing aids. Our goal in this paper is to research all the related articles, algorithms, models, and architectures utilized in the development and implementation of the proposed model.

2. BACKGROUND AND RELATED WORK

2.1 Denoising

In computer analyses, information (images, sounds etc.) can be made up by both useful data and noise, the last can reduce clarity for the information analyses [33]. The main goal of the denoising process is to reduce the noise data of the information and preserves more of the useful parts of the data. In audio aspects, denoising is an essential part of the processing, by removing noise and unwanted audio parts of the recordings. By 'filtering' the noise from a recording, we are improving the fidelity and audio quality of the content [16]. One of the classic techniques that are used for noise reduction and enhancement data in audio signals is Wiener Filtering. It works by applying a linear time-invariant filter to the noisy signal and by using the characteristics of the clean and noise speech in order to predict the clean signal in a more accurately way [16].

2.2 Sound Waves

A sound wave is a pattern of disturbances that are caused by the movement of sound, from its source through its entire travel: air, water, and all sorts of surface matter [28]. It is created by an object's vibrations that produce pressure waves – a short period of pressured fluctuation that makes a propagation of sound through the atmosphere [24]. The pressure wave that was created from the object disturbs the particles in the surrounding mediums, and they create a ripple effect of disturbance to the next particles, and so on, until they create a wave pattern. Sound waves have a few characteristics, some of them are amplitude, frequency, time, velocity and wave-length.

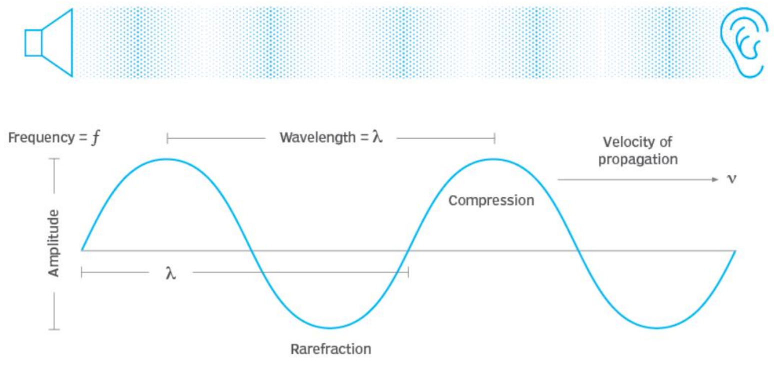


Figure 2. Characteristics of a sound wave [28].

2.3 Representations of audio

Time Domain

A time domain analysis is an analysis of signals; mathematical functions or environmental data, which are in reference to time and can show a graph of the signal changes in that reference [30].

In time domain, a signal will be defined as a function that changes over time (which is represented as a real number): $x(t), t \in \mathbb{R}$

With this definition, we can implement and represent the signal in discreate-time domain or continuous-time domain [27].

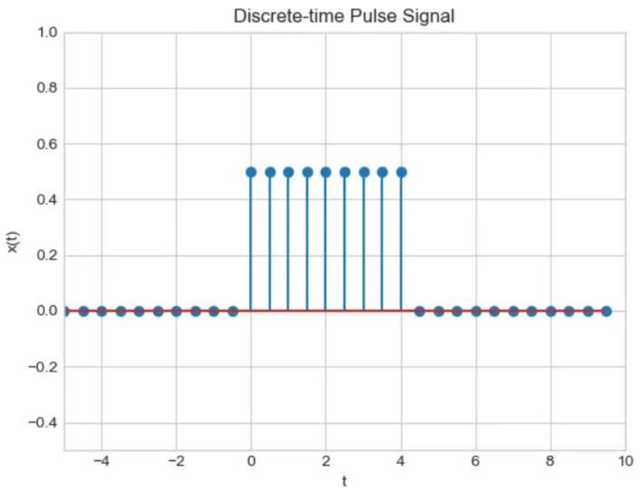


Figure 3. Discrete-time Pulse signal [27].

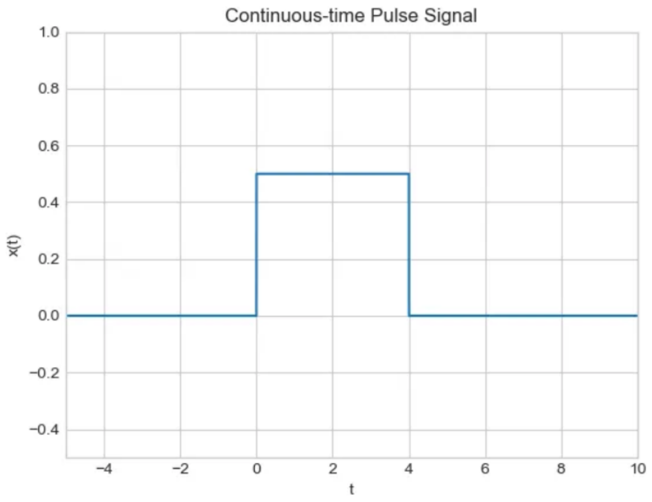


Figure 4. Continuous-time Pulse signal [27].

On the left, the signal will be classified as a discrete-time signal because it has a domain that is a small subset of lines from the real line, representing the signal at specific, isolated time points. On the right, the signal will be classified as a continuous-time signal because it contains at least one interval of the real line, representing the signal continuously over a time interval.

There are some cases where it is beneficial to turn our time-domain signals into time-frequency domain signals. In example, for tasks such as filtering and feature extractions [17].

Time-Frequency domain

The Time-Frequency domain analysis is an analysis of signals or mathematical functions in reference to both time and frequency. The representations of a signal in this domain are called a spectrogram:

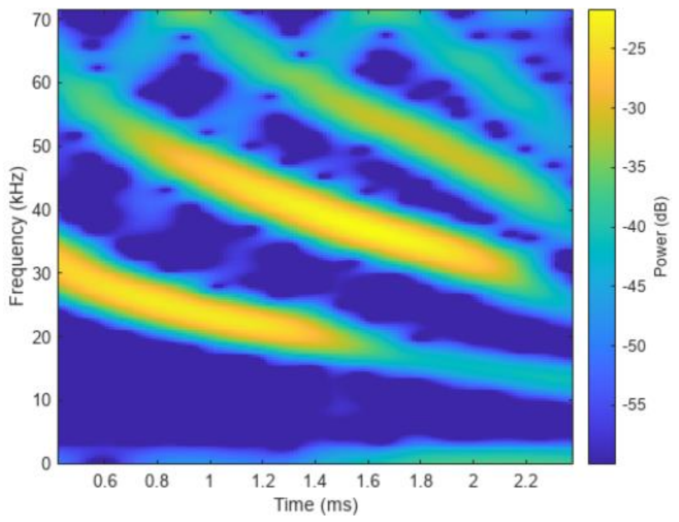


Figure 5. An example of a spectrogram [23].

The different colors representing the amplitude of each frequency at any given time. The advantages of a time-frequency domain representation over time domain lies in the ability to notice the sparsity and structure of different sounds in each timeslot.

Sparsity of different sounds:

We can infer by the

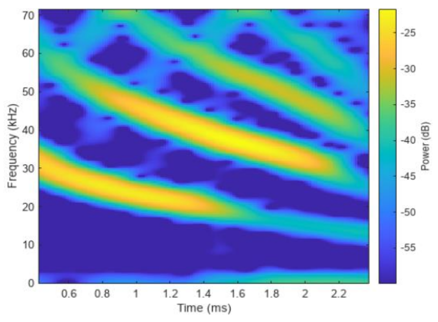


Figure 6. An example of a spectrogram [23].

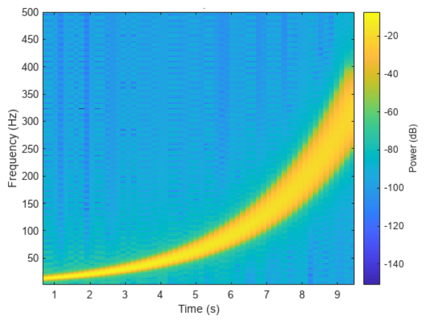


Figure 7. A spectrogram of a signal on a logarithmic scale [23].

yellow time frequency sections (that have a higher amplitude over the rest of the spectrogram), that it is only a small part of the entire spectrogram. with that in mind it is possible to see that many audio signals can have a sparse presence in time frequency domain, which can be helpful as the model will have fewer values/data that needs to be stored for further analysis. In time-frequency domain, the noise or any other disturbing signals, are typically not represented by the same coefficients as the source signal, like in the example, its visible to see that the spectrogram has four frequency ranges of sounds. For example, a compression between the signal's spectrograms:

The different sparsity in each spectrogram can lead to a conclusion that there's only a small probability that different sound sources are occupying different areas of the time-frequency in a spectrogram. With that, we can say that there is only one source with a significant amplitude in each time-frequency slot. This conclusion is called "W-disjoint orthogonality", which allows us to identify and separate each sound source from each other due to the fact they are mostly in separated areas of the spectrogram.

Different Structures of sounds:

Since naturally created sounds have a structure in both time and frequency, we can differentiate the source signal and noise signal with unique characteristics that make them. If we were to look at Figure 6 again, it is clear to see a structure for all four sound source's ranges. On the other hand, noise like sounds will lead to many coefficients that are distributed over the spectrogram without a clear structure at all.

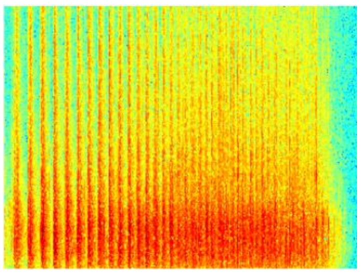
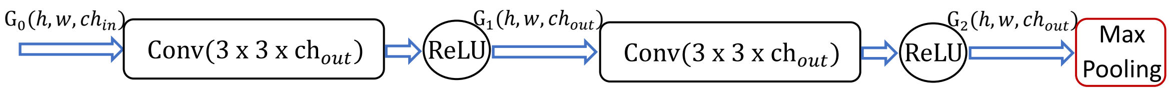


Figure 8. Sound spectrogram of applause [8].

With those characteristics, we can tell sources apart from each other, and can even improve the estimation of individual sources.

2.4 U-Net architecture

The U-Net architecture was proposed in 2015 [11] and was originally designed for segmentation of neuronal structures in electron microscopic stacks. The U-Net relies on the strong use of data augmentation to use the available annotated samples more efficiently, achieving precise segmentations even with minimal training data. The network consists of 2 symmetric paths, contraction and expansion, commonly known as the encoder and decoder, respectively, with a skip connection connecting them. Both paths contain equal number of steps, known as layers, where each layer perform the exact same operation as the other layers in its path. The contracting path performs the typical architecture of a regular convolutional network, downsampling the input to obtain higher level features. Each layer applies two 3x3 convolutions, each of them followed by a rectified linear unit (ReLU) and max pooling operation for downsampling. At each layer we double the depth of the feature maps. In other words:



The Expansive path aims to undo the shrinking caused by the contracting path, focusing on regaining spatial detail. Each layer involves upsampling of the feature map followed by a 2x2 transpose convolution that reduces the number of feature channels by half. The upsampled feature map is concatenated with its corresponding feature map from the contracting path. Then, two 3x3 convolutions are applied, each followed by a ReLU. The concatenation is done due to a loss of details and border pixels during the downsampling.

At the final layer a 1x1 convolution is used to map the output feature vector to the desired number of classes for the segmentation task.

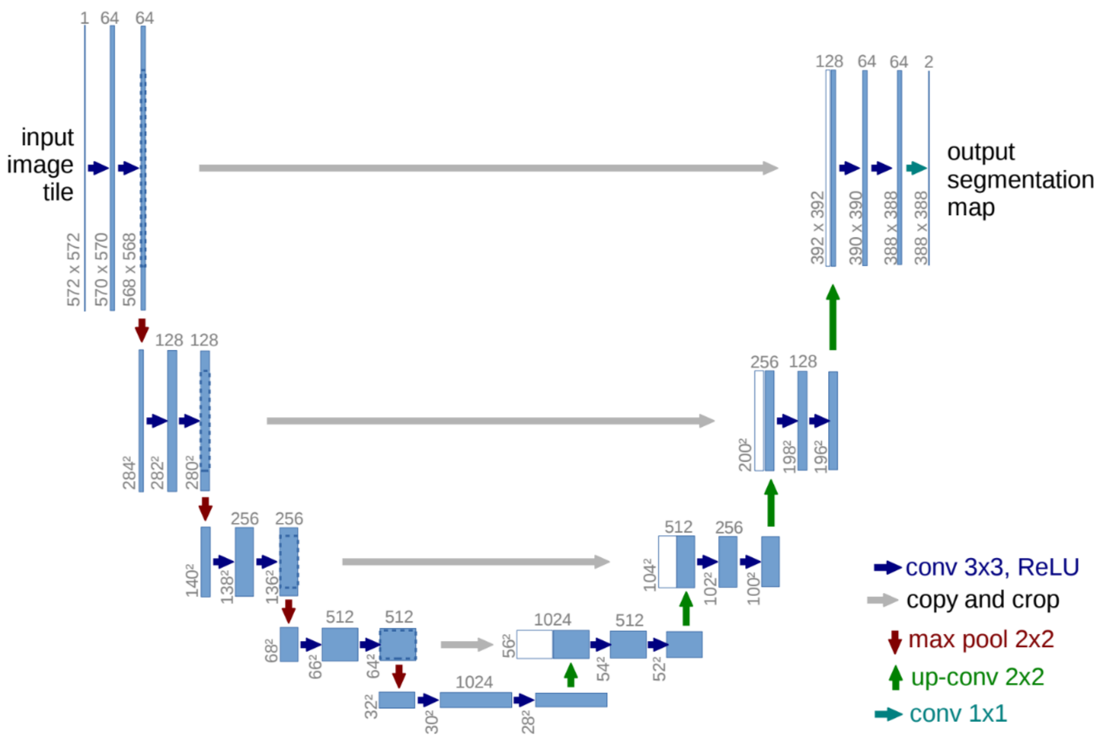


Figure 9. The original architecture of U-Net [11].

Variations of U-Net

Since the introduction of U-Net, many variations of the original architecture have been proposed. These variations mainly focus on modifying the operations within the encoding and decoding blocks; the total number of blocks and the bottleneck block itself, the block in the bottom center of the U-Net. The standard U-Net relies on standard convolutional blocks for feature extraction and upsampling in the encoding and decoding paths. The variations often use alternative operations such as dilated convolutions for capturing long-range dependencies or residual blocks for mitigating the vanishing gradient problem. while the original U-Net uses five blocks in each path, many variations increased or decreased this number depending on the complexity of the task and the available computational resources. The bottleneck block has been changed as well and many alternative methods have been explored for it. The original U-Net applies standard convolutional operations, while other methods used a Long Short-Term Memory (LSTM), attention mechanism, etc. which can further pay attention to the characteristic information output by the encoder.

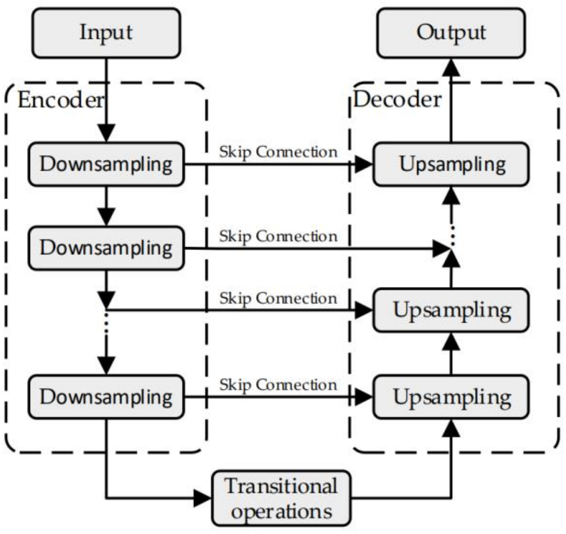


Figure 10. The general architecture of U-Net [9].

Variations of U-Net for Speech Enhancement

Speech enhancement is a very important subject in the field of speech processing. Its goal is to improve the quality and intelligibility of speech which disturbed by external noises. Speech enhancement based on U-Net architecture has been proven effective in various configurations, including the Time-Frequency-Domain and Time-Domain. Time-Frequency Domain U-Net, uses U-Net architecture with speech data represented in the time-frequency domain, often called spectrograms [5, 6]. While achieving promising noise reduction results, the presence of the original noisy phase spectrum in spectrograms can impact the denoising effectiveness. Time-Domain U-Net, also known as Wave-U-Net [12], was initially introduced in the context of audio source separation as an adaptation of the standard U-Net to the one-dimensional time domain. Beyond audio source separation, research on speech enhancement has begun to explore the potential of using the Wave-U-Net architecture for their task [10]. Furthermore, time-domain U-Net addresses the limitations of spectrogram-based methods by directly processing the raw audio signal. This enables the model to leverage both the magnitude (strength) and phase (timing) information of the sound wave, potentially leading to superior noise reduction capabilities.

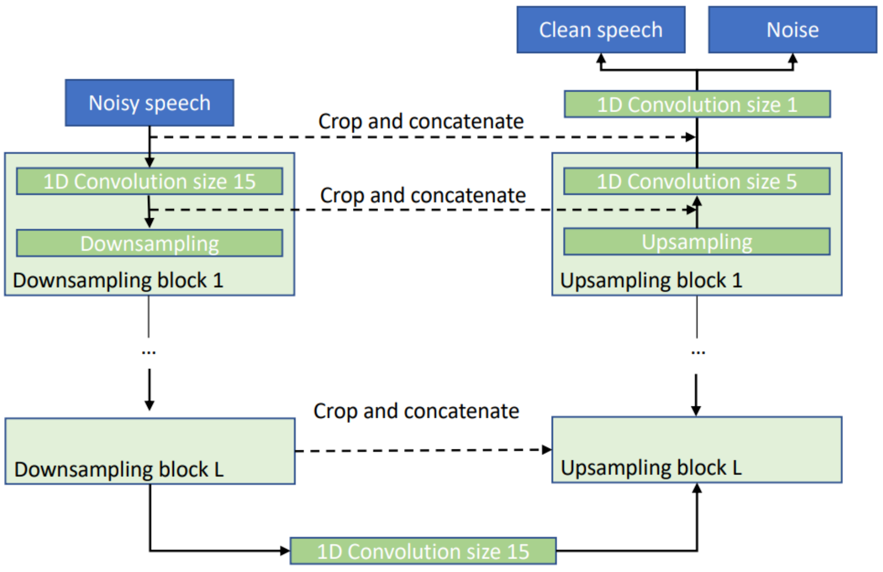


Figure 11. The Wave-U-Net architecture for speech enhancement [10].

2.5 Time-Domain U-Net

Time-Domain U-net differs from regular U-Net in the input and output representation of the model and in the convolution operations. In regular U-Net, the input typically consists of multi-channel images, such as RGB images. In contrast, in a Time-Domain U-Net, the input is usually a one-dimensional time series signal, such as audio waveform data. In the matter of the convolutions, the Time-Domain U-Net applies convolutions directly to the one-dimensional time domain data (1D-Conv).

2.6 Attention Mechanism

The first attention mechanism was first proposed by Dzmitry Bahdanau in 2015 and its purpose is to enhance deep learning models by selectively focusing on important input elements, depending on the contexts. With that, the attention mechanism improves the prediction's accuracy. The Attention mechanism enables the models to have an extra long-term memory and can make the model focus or have attention on all the previously tokens that were generated from the input. The model that was used before the attention was based on encoder-decoder of Recurrent Neural Networks (RNNs) or Lont-Short Term Memories (LSTMs). Even though RNN and LSTM are capable of looking at previous inputs as well, they have a shorter window of reference from any point. When the RNN/LSTMs were analyzing a much longer input, they couldn't access all the words that were generated at the start of the input, and then the translation task for example will turn out poorly [15]. That's why problem was called the "long-range dependency problem of RNN/LSTMs" [14]. In the model that Bahdanau proposed, he introduced the idea of "Attention", which means that when the model will generate a context vector, it will look for a set of positions in the encoder hidden states where the most relevant information is available.

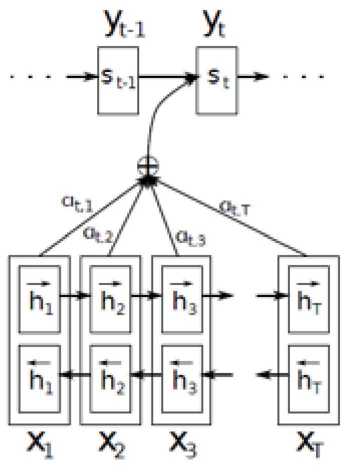


Figure 12. Diagram of the attention model from Bahdanau's paper [1].

As we can see in the diagram above, each block here is a bidirectional LSTM that is used to generate a sequence of annotations for each input sentence. All the vectors (h1, h2, h3 … hTx) are representations of Tx number of words in the original sentence (the input). In addition to that, Bahdanau put an extra emphasis on embeddings to all the words of the input – that are represented by the hidden states, in the process of creating the context vector. It was done by summarizing the weights of the hidden states. In short, for every query (a hidden state of the decoder), the attention will provide a "table" that shows how much attention that query have for each of the keys (encoder's hidden states), and with that can mark the level of focus that query will give for the rest of the input based on each decoded hidden state. With that in mind, we can use this model in order to improve and focus on relevant features in speech input rather than text input.

2.7 Shuffle Attention

As mentioned, the attention mechanism can focus on the important features of speech and lower the value of the unwanted data (noise) in the audio. One way to implement this is by using a shuffle attention mechanism, which will focus on the detailed features of the speech in that manner after the last down-sampling (the last output of the encoder). The mechanism works by dividing the speech feature map into groups along the channel dimension, and with that, the speech feature map of every group is being divided into two sub-feature maps (along the channel dimension as well). Each of the two sub-feature maps are now the input for one of the following: Channel Attention (C-Att) and Spatial Attention (S-Att) [9].

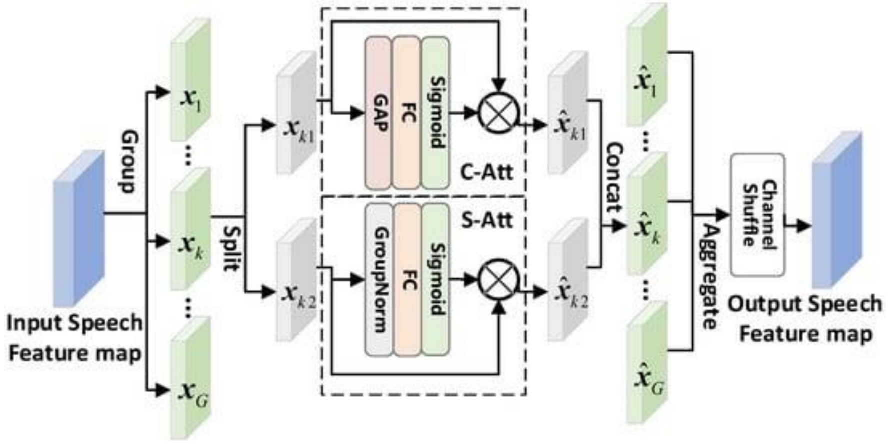


Figure 13. Shuffle attention mechanism with C-Att & S-Att [9].

2.7.1 Channel Attention (C-Att)

C-Att exploits the inner-channel relationships of the features, and with that the model builds a channel attention map [18]. C-Att focuses on the correlations between each of the channels of the speech feature map. From Figure 13, the C-Att uses Global Average Pooling (GAP), a pooling operation that generates one feature map for each corresponding audio characteristic. Instead of a fully connected layers, it averages each feature map.

2.7.2 Spatial Attention (S-Att)

S-Att uses the inter-spatial relationships of features to generate a spatial attention map [29]. Unlike C-Att, S-Att has more focus on the detailed features of the speech feature map.

By combining them together, S-Att and C-Att are complementing each other. In the Shuffle attention of Figure 13, we can see a specific speech feature x_k being split into x_{k1} and x_{k2} which are the speech sub-feature maps. x_{k1} and x_{k2} are then entered into C-Att and S-Att respectively, as can be seen from the following formulas:

$$\hat{x}_{k1} = \sigma(\omega_c f_{GAP}(x_{k1}) + b_c) \cdot x_{k1}$$

$$\hat{x}_{k2} = \sigma(\omega_s g(x_{k2}) + b_s) \cdot x_{k2}$$

$\hat{x}_{k1}$ and $\hat{x}_{k2}$ are the sub-feature maps output generated by C-Att and S-Att. ω_c, ω_s, b_c and b_s are the weights and biases of FC in C-Att and S-Att. $f_{GAP}()$ is the GAP operation, $g()$ is group norm, and $\sigma()$ is for the Sigmoid function. After calculation, the output sub-feature maps, they are concatenated to one another to create the output feature $\hat{x}_k$. After the process happens to all G speech feature group (from x_1 to x_G), the output features of attention from all the groups are aggregated. In addition to the aggregation, the final output is made with the channel shuffle operation – a similar operation to reshape operation. The operation allows information to fuse between different speech sub-features. After that process, the output feature map that was created has the same size as the original feature map input.

2.8 Convolution operation

The convolution operation is the base of all Convolutional Neural Networks (CNNs), and it combines two functions to generate a third function. By sliding a small filter, known as kernel, across the input image and computing the dot product between the filter weights and the corresponding pixel values in the input. This process is repeated across the entire image, generating feature maps that highlight important patterns and structures. The original input can be padded before the convolution begins with surrounding zeros. The convolution kernels are usually a small 2D matrix containing learnable weights. The size of the kernel determines the receptive field, which is the area of the input data considered at each position. The sliding size of the kernel is known as the stride.

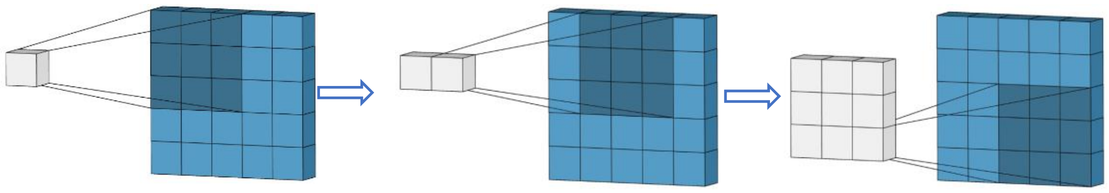


Figure 14. Convolution Operation with stride = 1 [20].

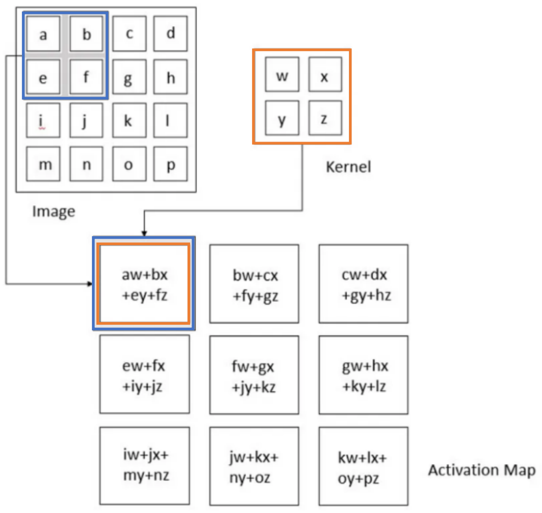


Figure 15. Convolution Operation [20].

However, the application of convolutional neural networks is not limited to two-dimensional data like images. For many tasks, such as time series analysis and audio processing, the data is often sequential. To effectively handle such data, one-dimensional convolution (1D-conv) is taking place instead of the regular convolution operation.

2.8.1 One Dimensional Convolution (1D-Conv)

1D convolution operates similarly to regular convolution but is specifically designed to process one-dimensional sequences. Instead of sliding a filter over a two-dimensional grid, 1D convolution involves moving a filter along a one-dimensional sequence, such as a time series signal or a sequence of words. The convolution formula is as follows:

$$y = x * w \ \rightarrow y[i] = \sum_{k=0}^{k=m-1} x[i-k] \cdot w[k]$$

Were x is the original input vector, filter w and m is the size of w. According to the formula, we can notice that the vector x is scrolled from right to left and w from left to right. In order to overcame the different directions, we can simply invert the filter vector w, and preform the vector product between x and a rotated w. Illustration of the process of 1D convolution:

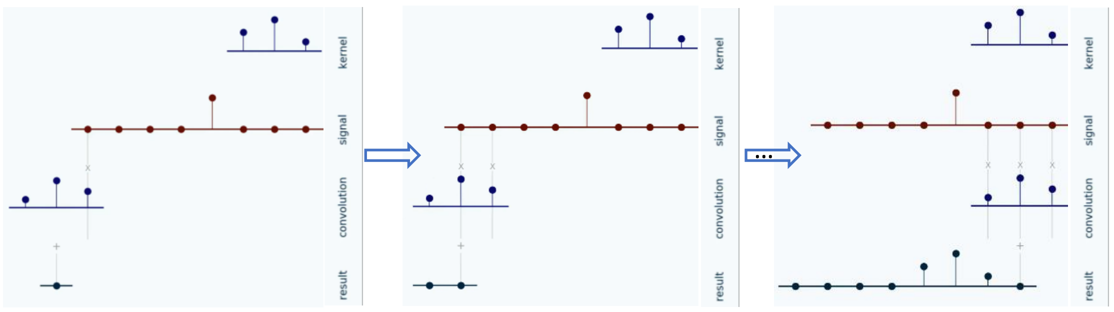


Figure 16. Flip kernel and sliding kernel along signal [19].

Initially, we begin by flipping the kernel. Next, we step the kernel along the signal according to the configured stride. At each position, we compute the dot product between the two by multiplying each pair of aligned values together and then summing up those products.

2.9 Dilated-Convolution

Dilated convolution is a variation of convolution where the kernel is expanded based on a dilation factor (l) before the convolution begins. Dilation involves increasing the size of the kernel by appending zeros to the right, bottom, and diagonals of each entry in the original kernel, except for the entries in the last row and column. The number of zeros added to the kernel is $l - 1$. For instance, a dilation size of 2 would result in one zero added along each direction.

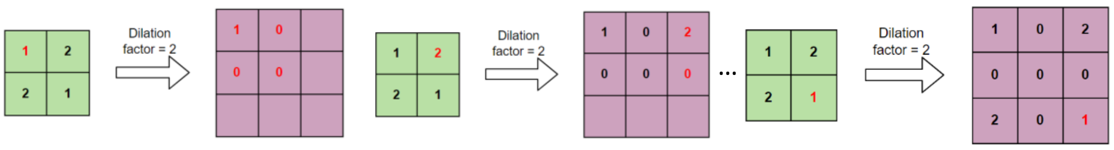
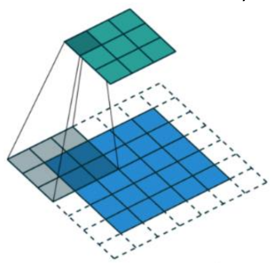


Figure 17. Dilation of a kernel with factor of 2 [34].

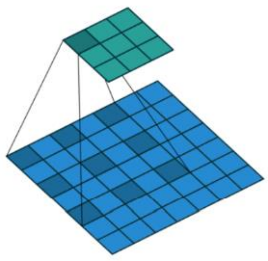
The dilated convolution can be explained as a convolution formula as follows:
Let F be the input, k is the filter, l the dilation factor.

$$\underbrace{(F * k)(p) = \sum_{s+t=p} F(s) \cdot k(t)}_{Standard\ Convolution} \qquad \underbrace{(F *_l k)(p) = \sum_{s+lt=p} F(s) \cdot k(t)}_{Dailated\ Convolution}$$

We can see that at the summation, it is $s + lt = p$ that we will skip some points during convolution. As well, when $l = 1$, it is a standard convolution.



Standard convolution



Dilated convolution (l=2)

Figure 18. Illustration of dilated convolution [25].

2.9.1 1D Dilated-Convolution

1D Dilation in convolution is like 2D operation we explained above, we increase the span of filters without increasing the number of weight parameters in them.

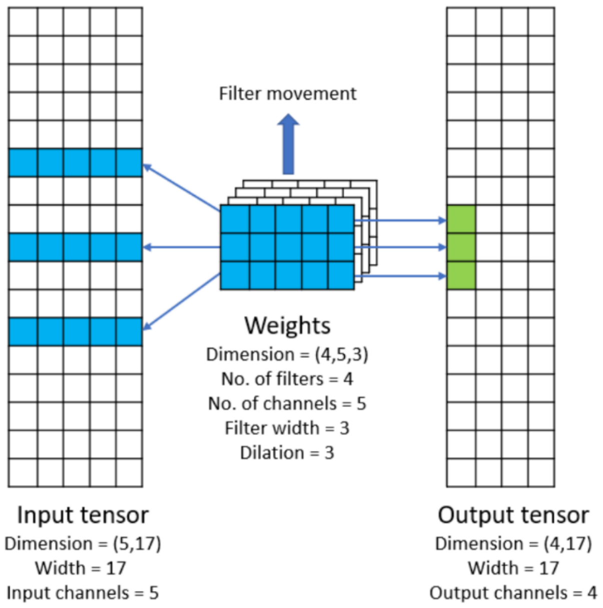


Figure 19. Illustration of 1D dilated convolution [2].

2.10 Transpose-Convolution

Transpose convolution (Trans-Conv) operation is an upsampling operation, generating an output feature map larger than the input feature map. The Trans-Conv operates similar to a regular convolutional operation but in reverse. Instead of sliding the kernel over the input, it moves the input over the kernel, conducting element-wise multiplication and summation. This process expands the output, and the size of the resulting feature map can be adjusted using the layer's stride and padding parameters. [35, 32]

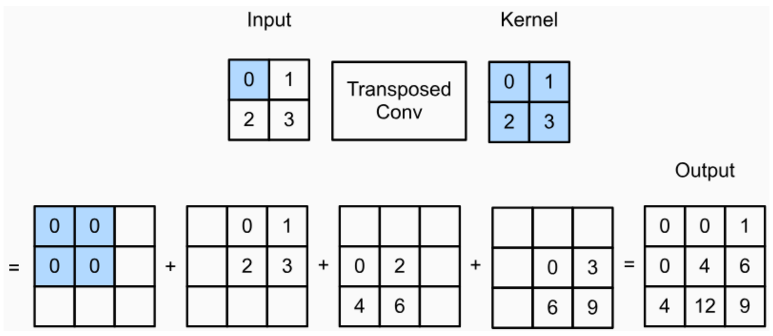


Figure 20. Illustration of Transpose-convolution with a 2x2 kernel [31].

2.11 Gated Linear Unit (GLU)

Gated Linear Unit (GLU) is an activation function for neural networks, aimed to address the issue of vanishing gradient and improving focus on relevant features. Unlike the standard linear activation ($f(x) = x * w + b$), GLU uses a gating mechanism that controls information flow by selectively passing or blocking information and operates as a logical gate. It can be explained by the formula from the paper [3]:

$$GLU(x) = \underbrace{(x * w_a + b)}_{A} \otimes \sigma(\underbrace{x * w_b + c}_{B})$$

Where x is the input vector, w_a, w_b, b, c are learned parameters (weights and biases, respectively), σ is the sigmoid function and $\otimes$ is the element-wise product between matrices. The idea lies in the multiplication between the information (A) and the gating signal $(\sigma(B))$, which is the output of the sigmoid function. The sigmoid function is critical because it outputs values between 0 and 1. As a result, when gating signal is closer to 1, more information from A will pass through. But, when gating signal is closer to 0, it restricts the flow of information. This enables GLUs to capture complex patterns and dependencies within the data [22]. Illustration of the GLU process:

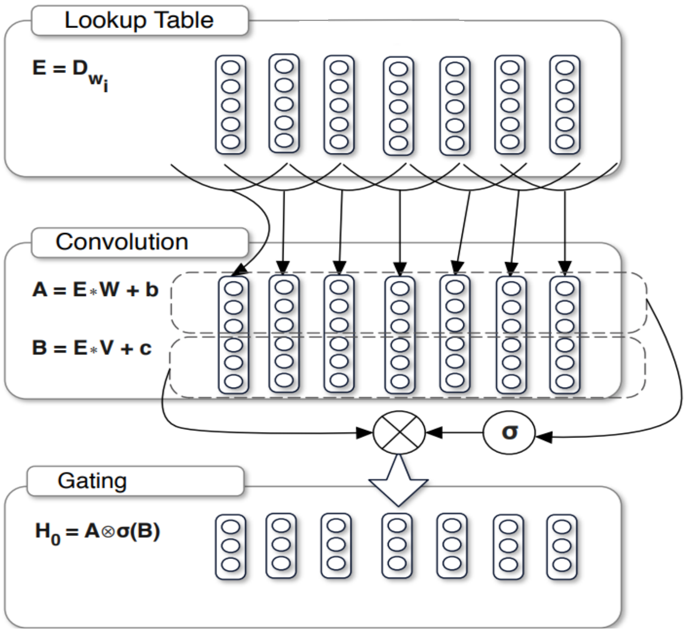


Figure 21. Illustration of operation of GLU [3].

2.12 Compressed Sensing

Image Compression

Regular image compression techniques take advantage of the fact that a lot of information in a picture is repeated. One common approach, as demonstrated below (Figure 22), uses the frequency domain by transforming the image using the Fast Fourier Transform (FFT). This transformation reveals the distribution of image information across different frequencies. In most pictures, the important details we see are usually stored in the lower-frequency components (long and smooth changes in color or brightness), traditional methods discard a portion of the high-frequency coefficients (rapid changes in color or brightness) while keeping the most significant ones, this reduces the file size of the image [21]. Illustration of the process:

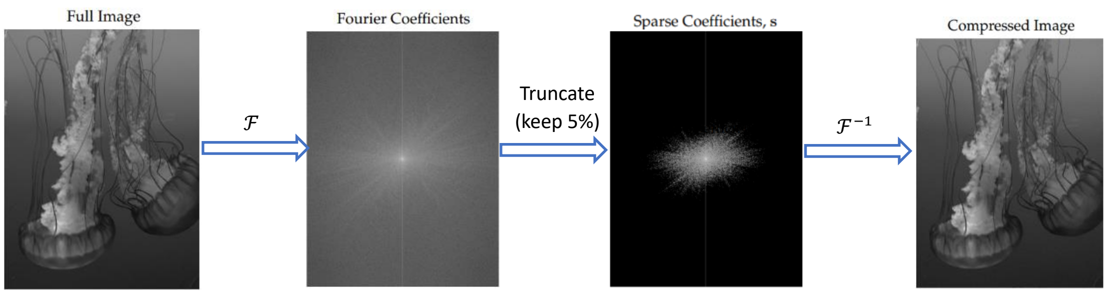


Figure 22. Illustration of compression with FFT $\mathcal{F}$ [21].

The compression follows the equation:

$$x = \Psi s$$

Where $x \in \mathbb{R}^N$ is the original image, $\Psi \in \mathbb{R}^{N\times N}$ is a Fourier transform basis and $s \in \mathbb{R}^N$ is a sparse vector.

Compressed Sensing

Compressed sensing (CS) is a method of compressed sampling [9]. CS requires the signal to be sparse, and the measurements of signal are obtained through sparse and dimensionality reduction of the original signal. Measurements can retain most of the information in the original signal with a very small amount of data. The following equation expresses the process of obtaining measurement signal $y \in \mathbb{R}^M$.

$$y = \Phi x, \ \ (x = \Psi s)$$

Where $x \in \mathbb{R}^N$ is the original signal and $\Phi \in \mathbb{R}^{M\times N}$ $(M \ll N)$ is the measurements matrix. The core problem of CS is to reconstruct x from measurement y, typically by applying constraints on x. A common constraint is that the original signal needs to be sparse. The original signal x can be recovered from measurements by solving an optimization problem for s [21]:

$$\hat{s} = \operatorname{argmin}\|s\|_1, \ \ subject\ to\ \|\Phi x - y\|_2 < \ \varepsilon$$

A schematic illustration of reconstructing x can be demonstrated as below:

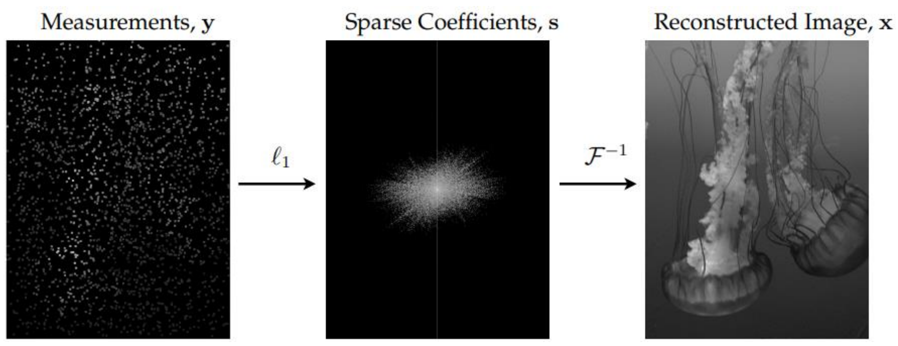


Figure 23. schematic illustration of reconstructing x by solving the optimization problem for s [21].

2.13 Compering Sounds

There are multiple ways to compere the amount of loss of the speech enhancement process. For the following methods we will use the same variables: s is the clean speech, n is the noise in the speech, and the combination of the two will represent the noisy speech signal $x = s + n$. The enhanced speech which is $\hat{x}$ and N is the total amount of samples in the speech signal. Most speech enhancement models that are based on U-Net use either Mean Square Error (MSE) or Mean Absolute Error (MAE) to compare between the enhanced and clean speech signals.

2.13.1 Mean Square Error (MSE)

In speech enhancement, MSE is a common metric used to evaluate the performance of the speech enhancement process. MSE measures the average of the squares of the differences between the enhanced speech and original speech signals.

$$L_{MSE}(s, \hat{x}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{x}_i - s_i)^2$$

Even though it is a common method to evaluate the speech enhancement, it has a significant fault: it is easily affected by unusual, extreme points in the speech signal. since MSE squares the difference between signals, it is possible that large errors can lead to a big impact to the result.

2.13.2 Mean Absolute Error (MAE)

Another common metric that is used to assess the performance of speech enhancement, Mean Absolute Error (MAE) measures the average of the absolute differences between the enhanced and original speech signal.

$$L_{MAE}(s, \hat{x}) = \frac{1}{N} \|\hat{x} - s\|_1 = \frac{1}{N} \sum_{i=1}^{N} |\hat{x}_i - s_i|$$

Since other researches had already proved that MAE is significantly improve the denoising performance [4], and since MAE doesn't square the differences, it can handle unusual points in a signal better than MSE. Because of that, MAE is considered better to use for this sort of evaluation rather than MSE.

2.13.3 Compressed Sensing (CS)

Using compressed sensing for calculating the amount of loss by using the measurements as the optimizable target. Measurements can retain the main information and even some unobvious features of original speech. Therefore, the enhancement performance of model can be improved by optimizing the MAE between measurements of clean speech and measurements of enhanced speech [9].

$$L_{CS}(s, \hat{x}) = \frac{1}{N} \|\Phi\hat{x} - \Phi s\|_1$$

2.13.4 Short-Time Fourier Transform (STFT)

Short-time Fourier transform (STFT) is a tool that is used to analyze speech signals and helps to understand how the frequency of a signal changes over time. The STFT works by initially splitting a signal into small, overlapping windows, focusing on a small portion of the signal at any given time during analysis. The way to calculate the STFT of a signal with frequency ω and time t is as the following:

$$X_{STFT}[t,\omega] = \sum_{k=0}^{L-1} x[k]g[k-t]e^{-\frac{j2\pi\omega k}{L}}$$

where $x[k]$ represents the k-th window of the signal, and $g[k]$ represents the analysis of the k-th window function that is centered at time t. j represents the imaginary unit ($\sqrt{-1}$). With this method, we transformed each window of the original signal from Time domain into Frequency domain, and by combining all of them in order (of time), the result will be a representation in Time-Frequency domain [26].

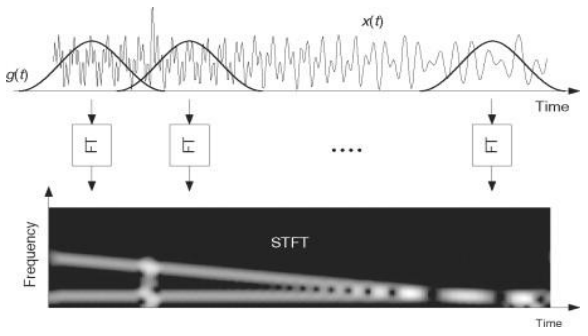


Figure 24. Illustration of computing STFT by taking Fourier transforms of a windowed signal [26].

After there is a way to represent the signals in time-frequency domain, we can optimize the clean and enhanced speech, to learn the time-frequency characteristics of them. With that, the loss function of a STFT can be defined as the following:

$$L_{STFT}(s,\hat{x}) = E_{\substack{s\sim p(s)\\ \hat{x}\sim p(\hat{x})}}\left[\frac{\||STFT(s)| - |STFT(\hat{x})|\|_F}{\||STFT(s)|\|_F} + \frac{1}{N}\|log|STFT(s)| - log|STFT(\hat{x})|\|_1\right]$$

Where E represents the expected value, by averaging the loss function of the probability distributions $p(s)$ and $p(\hat{x})$ of the clean speech s and enhanced speech $\hat{x}$. $|STFT()|$ stands for the absolute value of any given speech. $\|val\|_F$ represents the Frobenius norm – a matrix norm that is similar to the Euclidean norm, and $\|val\|_1$ is for the L_1 norm, that measures the absolute differences between the components inside it. The normalization ($\frac{1}{N}$) is used to scale the L_1 norm in the function so that the loss value is averaged over the length of the signal, N.

2.13.5 Perceptual Evaluation of Speech Quality (PESQ)

PESQ is an algorithm which is used to measure and assess the quality of speech in communication networks. The measurement works by comparing the original audio with a corrupted/noisy version of the audio, that has already passed through the system. PESQ is commonly used in the evaluation of performance of voices, echo cancellers, and other systems of communications that affect and change the speech clarity. The scoring system of PESQ ranges from -0.5 to 4.5, but in most cases the score will be around 1.0 – 4.5, where the higher the values, the better quality the audio has after analyzes. In recent years, PESQ has been a tool for evaluating Voice-over-IP (VoIP) systems, mobile networks, as well as other communication systems and platforms that speech quality is a crucial part of them [36].

2.13.6 Sound-To-Noise ratio (SNR)

Sound-To-Noise (SNR) is the ratio between the desired information/power of a signal against the undesired signal, which in turn is the background noises in the signal being more present. The unit of measurement is in decibels (dB), The higher SNR indicates a clearer, better signal, by having less noise or interferences relative to the strength of the signal. Opposite to that statement, the lower SNR can mean a distorted or a noisy output, which can make it difficult to tell the difference from the original signal. When looking at improving audio signals, we will look at a histogram of SNR improvement distribution of multipole signals. On average, improvements in the range of 0-10 dB, can take a signal from a very poor quality to a good or decent quality. Improvements beyond 10 dB, while moving to 20 dB, are indicative of a great quality signal, in which, the noise becomes practically unheard compared to it [37].

3. RESEARCH / ENGINEERING PROCESS

3.1 Research Process

In the second part of the project, we focused on implementing and optimizing a U-Net-based model for the task of speech enhancement. Unlike the SECS-U-Net model explored in Part A, our final implementation involved a different version of the U-Net architecture. This model was developed and trained using **TensorFlow** and **Keras** on Kaggle.

3.2 Product

Data Preparation

We utilized two datasets for the training and validation processes: **RAVDESS** (Ryerson Audio-Visual Database of Emotional Speech and Song) [38] for clean speech and **UrbanSound8K** [39] for noise. As TensorFlow could not natively read the raw audio files, the datasets were restructured locally before being uploaded back into Kaggle. Additionally, all audio was down-sampled to 8kHz to maintain most of the essential speech signal.

The combination of these datasets posed a challenge, as the dataset itself is relatively small for the speech enhancement task. Despite this, it offered a sufficient platform to test the model's generalization on unseen data.

3.2.1 Algorithm and Model Design

The core algorithm is based on a Short Time Fourier Transform (STFT), which transforms the audio data into a 2D time-frequency domain representation. The U-Net architecture was then applied to the amplitude of the STFT output, while the model disregarded the phase information. Additionally, a softmask component was introduced into the U-Net pipeline to further refine the noise reduction process.

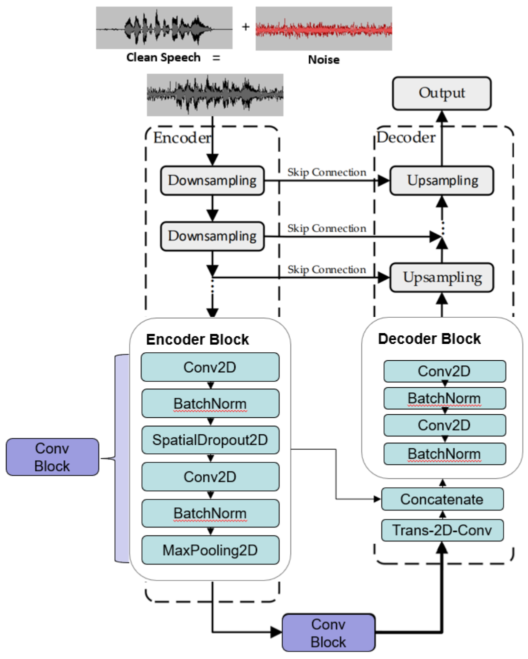


Figure 25. Workflow of the U-Net model. The input is a noisy speech that will pass through U-Net and output an enhanced speech. The model has custom configured encoding and decoding layers, and 1 Conv Block in the bottleneck of the U-Net.

This U-Net model is a modified version of a pre-existing architecture, where adjustments were made to incorporate the softmask, enhancing its speech-preserving ability. The design is inspired by the U-Net model developed in [9].

3.2.2 Custom Loss Function

To address limitations encountered with a simple Mean Squared Error (MSE) loss function, as explained in section 2.13.1, we developed a custom loss that prioritizes preserving speech over minimizing noise. This loss is composed of two main terms, the first is a regular MAE as shown in section 2.13.2. The second term focuses on preserving speech during noise reduction. It compares the squared true speech signal y^2 to the interaction between the predicted speech $\hat{y}$ and the true speech signal y, which can be written as:

$$Speech\ Preservation = 2\ \cdot |s^2 - \hat{x} \cdot s|$$

The term penalizes the loss of speech more heavily and ensure that while noise is being reduced, the essential speech content is retained as much as possible. The total loss function combines the two components (terms are as described in section 2.13):

$$Custom\ Loss = \frac{1}{N}\sum_{i=1}^{N}(|\hat{x}_i - \ s_i| + 2\ \cdot |y^2 - \hat{y} \cdot y|)$$

The combined loss balances and minimizing general errors and maintaining speech clarity. The weighting of the second term ensures that the model focuses more on preserving speech than just blindly reducing noise.

3.2.3 Model Training and Evaluation

The model was trained using Adam optimizer with varying hyperparameters such as batch size, number of filters, and epochs. The PESQ (Perceptual Evaluation of Speech Quality) metric was employed to automatically measure the speech quality improvements across different iterations. While PESQ has limitations, it provided a useful first-pass evaluation.

4. EXPERIMENTS

The model was checked against different experiments conducted to evaluate the performance of the U-Net model for speech enhancement. The experiments cover a range of audio inputs, testing the model's ability to generalize across various conditions.

4.1 The Developed Code

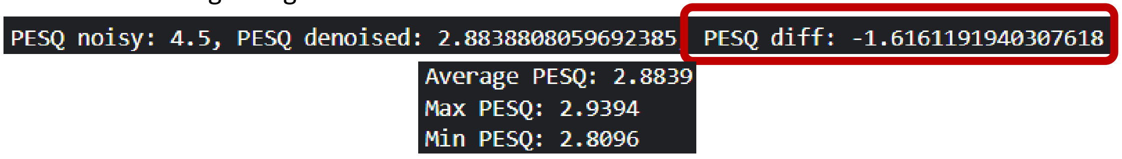
The code for this project was developed using Python and the Keras API within TensorFlow, implemented on the Kaggle platform. The datasets used for training and testing include RAVDESS_8K, which provides clean speech data, and UrbanSound8K, containing a variety of environmental noise samples, both dataset samples are in 8 kHz. Data preprocessing involved downsampling the audio files to 8 kHz, if they are different. As well as augmenting the clean speech data with different levels of noise, according to two types of factors, urban_noise_factor and white_noisy_factor, that determine the intensity of the noise that will be added to the clean speech from RAVDESS dataset.

4.2 Testing Additional Audio Sources

To evaluate the U-Net model's performance across different types of audio inputs, we conducted several experiments. Below are the results based on distinct categories of input sounds.

4.2.1 Noiseless Audio

This experiment tested how the model performs when the input contains no noise. Since no noise was present, the model is expected to preserve the audio without introducing any artifacts. But, contrary to expectations, the results below show a downgrade in the quality of the speech, with the model introducing slight distortions even though no noise was present. This indicates that the U-Net model, in its current form, struggles to process completely clean audio without degrading it.



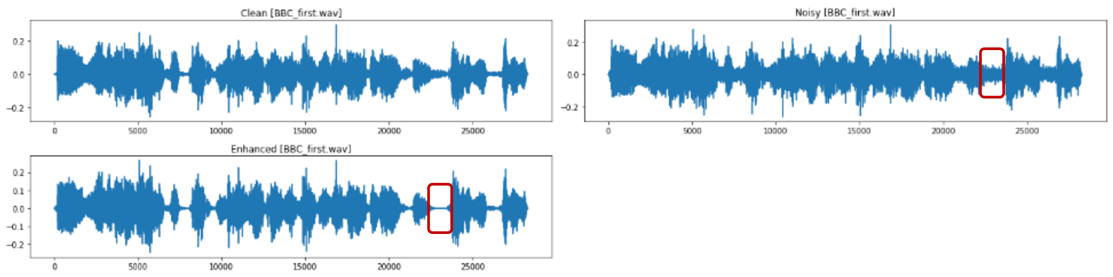
```
PESQ noisy: 4.5, PESQ denoised: 2.8838808059692385  PESQ diff: -1.6161191940307618
```

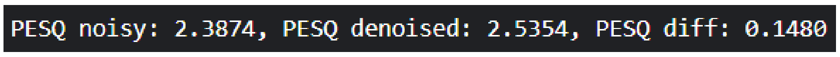
```
Average PESQ: 2.8839
Max PESQ: 2.9394
Min PESQ: 2.8096
```

Figure 26. The results of denoising a noisless audio file with the speech enhancment model. The red indicates the decrease in PESQ value of the audio after the process.

4.2.2 Multiple Speakers / Choir

We evaluated how the model behaves when dealing with multiple speakers or a choir-like audio input, combined with added noise. The U-Net was able to remove the noise sufficiently well, as shown below, demonstrating its capability to enhance speech in multi-speaker environments.





```
PESQ noisy: 2.3874, PESQ denoised: 2.5354, PESQ diff: 0.1480
```

Figure 27. Sound waveform representation of the denoising process of the audio file from the BBC, as well as PESQ values of the improved audio.
The area in red in both the noisy and enhanced waveform shows the denoising process at work.

4.2.3 Cascaded Model

In this experiment, we implemented a cascaded approach where the output of one U-Net model was fed into another kind of configured U-Net model for further enhancement. The goal was to iteratively improve the speech quality over multiple stages. However, this approach failed to refine the speech further and, in some cases, as described below, even reduced the overall quality. The results highlight the limitations of cascading models for this task.

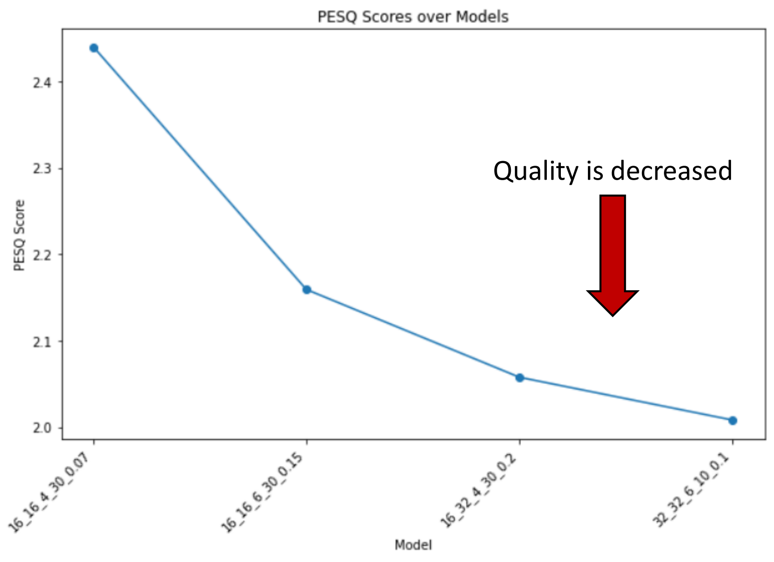


Figure 28. A graph showing the cascading process with 4 different models. After each model there is a decline in PESQ score, showing us this approach has failed to enhance the speech any further.

4.2.4 Iterative denoising

We performed iterative denoising by applying the U-Net model multiple times on the same input, aiming to incrementally reduce the noise. Similar to the cascaded approach, iterative denoising did not yield significant improvements and led to diminished speech quality after multiple iterations, as illustrated below. This suggests that a single pass of the U-Net model is optimal for the type of noise and speech signals used in this study.

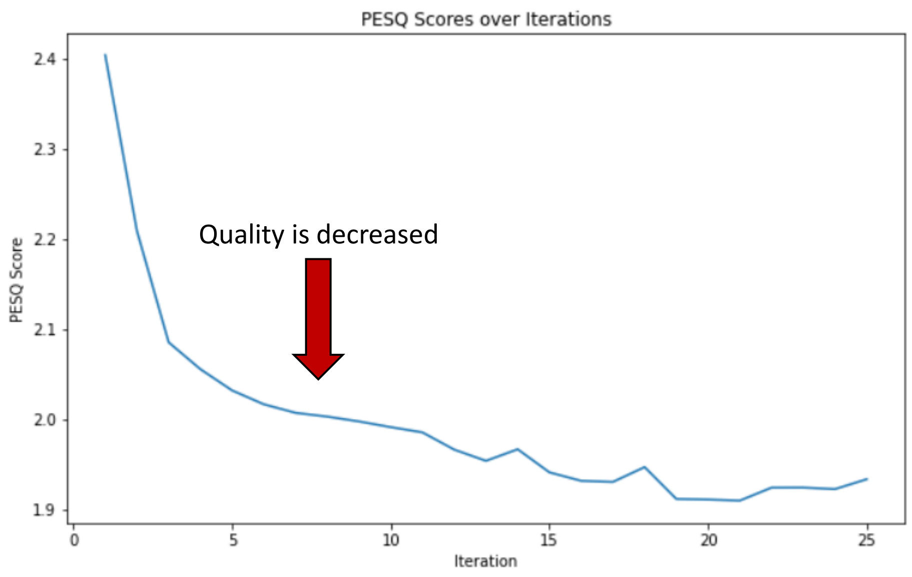
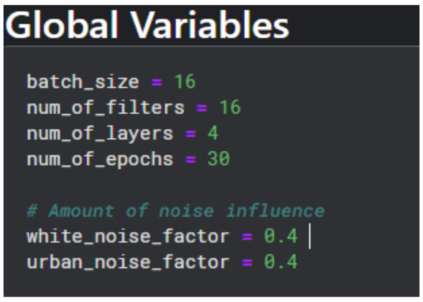


Figure 29. A grpah for the iterative denoising process, were it is visible to see the deacrease in PESQ value, which suggests that this approach cant improve the denoising process.

4.3 Numerical Study

The model is built with a configuration of global variables, we will go into detail over them.



Global Variables

```
batch_size = 16
num_of_filters = 16
num_of_layers = 4
num_of_epochs = 30

# Amount of noise influence
white_noise_factor = 0.4
urban_noise_factor = 0.4
```

Figure 30. Global variables that are changed to build different versions of the model.

4.3.1 Batch Size

The batch_size variable determines the number of training samples that the model process before updating its weights. The assumption is that increasing batch_size will lead to a more stable estimates, which will allow the model to converge more smoothly during its training, with the potential cost of more resources. On the other hand, a small batch_size could insert of a more noise into the training process of the model, making the training a bit harder, but will help it learn faster with frequent updates.

4.3.2 Filters

The num_of_filters variable controls the original number of convolutional filters in each convolutional layer of the model. As seen in [Figure 9], the number of filters is multiplied after entering the next layer (from 64 to 128 and so on). Our belief that decreasing num_of_filters will cause the model speed up its training which will make it simpler, by missing important details in the data during training. Conversely, we believe that increasing num_of_filters could actually help the model learn more complex patterns of signals, which will improve the speech enhancement process.

4.3.3 Number of Layers

The variable num_of_layers set the depth of the model, i.e., how many layers the model has. In the original model, they were 5 layers (for both encoder & decoder). We can see that reducing the depth of the model (num_of_layers) will make it less complex, since the model will have fewer transformations on the input data. As opposed to that, if we will increase the number of layers, we will let the model to extract more abstract features during each level, and with that, improving the model's performance.

4.3.4 Number of Epochs

The num_of_epochs variable defines how many passes through the training dataset the model will make during the training. From previews courses in our studies, we believe that increasing the number of epochs will allow the model to continue learning and gaining better weights, which will lead to better performance, but it can cause the model to overfit if the training is too long. With that, we will compare multiple models with different number of epochs and see when it causes to much training for the model.

4.3.5 Additional factors – White and Urban

The variables white_noise_factor and urban_noise_factor control the percentage of noises to add to the audio data during training to simulate various noisy cases for the model to learn from. The white noise factor adds synthetic, random noise across all frequencies of the audio, while the urban noise factor enters a real-world sounds to the audio, such as traffic, animals and so on. We believe that an average percentage of factors will improve the model, by exposing it to a more diverse noises during the training, making it more robust and ready to different types of noises to filter. On the other hand, since urban noises have a much stronger effect on the model – because it adds a random element to the audio, as opposed to white noise, if we were to increase the urban factor too much, we think it may cause the opposite response and even lower the performance of the model.

5. EVALUATING THE EXPEREMENTS

5.1 Loss-To-Epoch Comparisons

In this comparison, we selected the best model, model_weights_16_16_4_30_0.4_0.4, based on PESQ and SNR improvement scores based on the table in section 5.2.

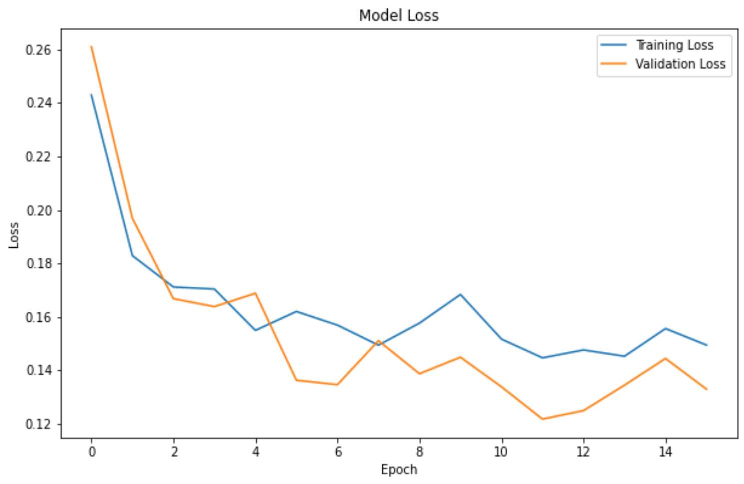


Figure 31. Loss vs Epoch graph of the model.
The most loss improvement happens during the first few epochs.

As seen in the graph, the most improvement of the model happens around the first 5 epochs, decreasing almost twice the amount of loss the model had at the beginning, and reaching the lowest loss value at the 11th epoch.

The graph shows us the "*callback_early_stop*" comes into account, setting at the value of 4 epochs, we can see the loss value not changing from the 11th epoch until the 15th, and with that the model stops the training. With that in mind we can assume that only if we were to set "*callback_early_stop*" value higher, during the final epochs the model maybe could reach an even lower loss value, but it will cost the training with more time and resources. In order to justify increasing the value of "*callback_early_stop*", we would need to set a goal for a more robust and detailed model.

5.2 PESQ & SNR Improvement between models

In order to compare the improvement of the models in both PESQ and SNR, we built a table that measures for each model the improvements.

model name	PESQ **Noisy - Enhacned (improvement)**	SNR improvement
model_weights_16_16_4_10_0.1_0.5	1.6965 -> 2.0989 (0.4024)	6.49 dB
model_weights_16_16_4_30_0.03_0.6	1.8632 -> 2.0656 (0.2024)	5.16 dB
model_weights_16_16_4_30_0.03_0.8	1.7306 -> 1.9439 (0.2133)	5.58 dB
model_weights_16_16_4_30_0.2_0.8	1.4402 -> 1.7717 (0.3315)	7.48 dB
model_weights_16_16_4_30_0.4_0.4	1.4139 -> 1.8463 (0.4324)	9.52 dB
model_weights_16_16_6_10_0.1_0.5	1.7081 -> 2.2393 (0.5311)	7.07 dB
model_weights_16_16_6_30_0.03_0.4	2.0798 -> 2.5547 (0.4749)	5.37 dB
model_weights_16_16_6_30_0.15_0.3	1.7191 -> 2.3458 (0.6267)	7.02 dB
model_weights_16_32_4_30_0.2_0.4	1.5899 -> 2.0987 (0.5088)	7.84 dB
model_weights_32_16_4_10_0.1_0.5	1.7278 -> 2.2193 (0.4916)	7.10 dB
model_weights_32_16_6_10_0.1_0.5	1.7358 -> 2.1830 (0.4472)	6.71 dB
model_weights_32_32_6_10_0.1_0.5	1.7303 -> 2.2815 (0.5512)	7.32 dB
model_weights_32_32_6_15_0.1_0.5	1.7424 -> 2.2634 (0.5213)	7.23 dB
model_weights_32_32_6_30_0.1_0.5	1.7476 -> 2.2658 (0.5182)	6.71 dB
model_weights_32_32_6_30_0.03_0.4	2.0658 -> 2.5433 (0.4775)	5.41 dB

The table shows the models average values of PESQ from the noisy to the enhanced, as well as the SNR improvements of each model. From the different models it's possible to assume that by only changing a variable or two, the models get very different results in terms of PESQ and SNR improvement. The models epoch amount ranges between 10,15 and 30, while filters have either 16 or 32. we will focus on the following models:

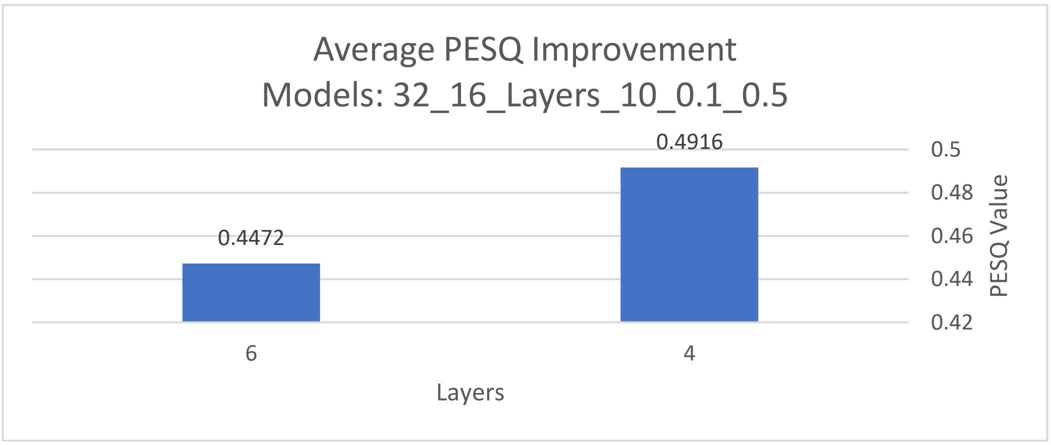


Figure 32. A graph measuring the PESQ improvement of two models where only the amount of layers has changed

We can see that having a low number of epochs (10) and filters (16), while having a higher number of layers in the model (6), will cause the PESQ improvement to decrease. With that in mind we can say that with a low number of epochs, the more layers the model has the worst the PESQ improvement will be. In order to verify that saying about the filters we ran another comparison.

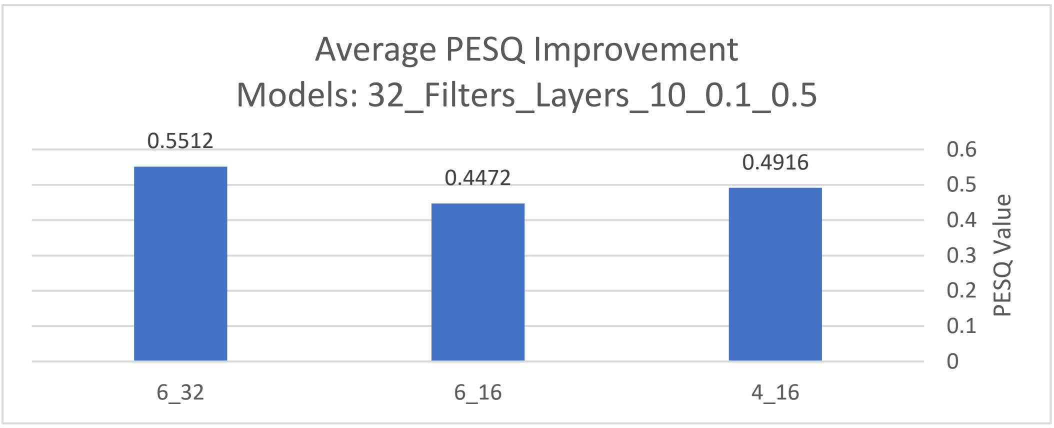


Figure 33. A comparison graph of the PESQ improvement of three models that have the same configuration beside Filters and Layers

We can see that by having a deeper model, with 6 layers, and more filters in each of them (32), the PESQ improvement has increased much more than increasing the number of filters or the depth of the model alone. Whan, we depend on the model without increasing the number of filters, we received a lower PESQ improvement than when we made 4 layers only. Now we can say that if the model has a high number of layers **and** filters, then the models PESQ value will improve more.

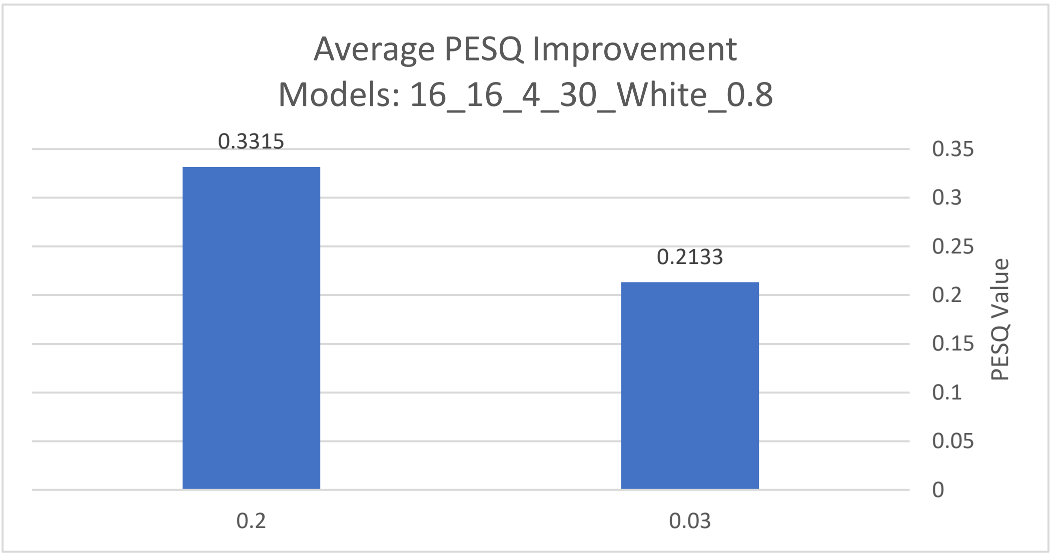


Figure 34. Comparing the PESQ improvement of 2 models with different precentage of white noise factor.

from the graph we can say that with an increased white noise factor, the model reaches a higher PESQ improvement value. In those two models we only looked at the white noise.

Next, we will observe the SNR improvements of models with the same variables but with different white and urban noise factor values.

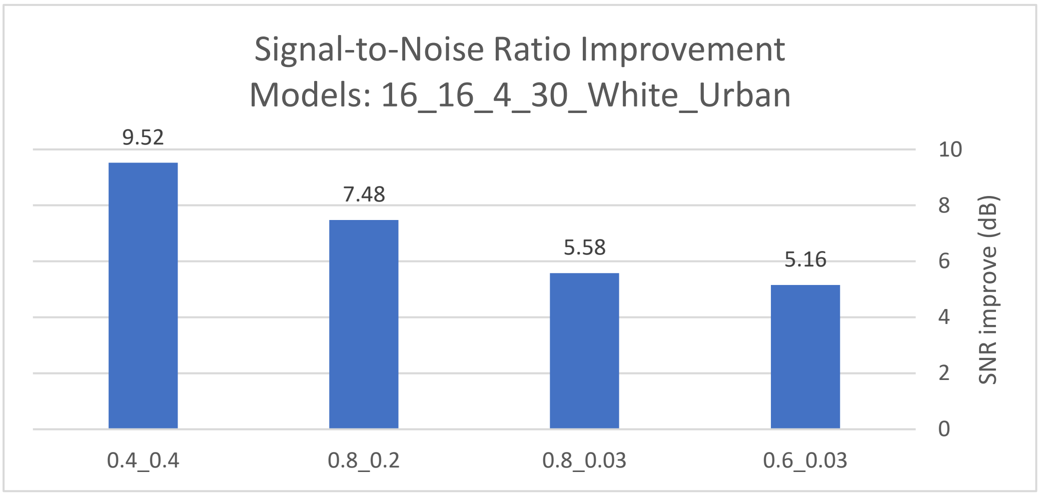


Figure 35. Comparing multiple models SNR's improvment with diffrent white and urban factors.

When reaching an equal or close values of white and urban noise factor, the SNR score of the model improves around 7 to 9 dB from the original audio files. With that we can say that the white noise factor softens the urban noise, that is very strong by itself.

5.3 Spectrogram Comparison

In this part, we present spectrograms from different speech enhancement models that have gone through training with different values to the global variables, as seen in Chapter [5.3]. These visualizations will demonstrate how each model processes and recovers speech from noisy environments, and with that shows the effects of noise reduction across multipole settings. By comparing spectrograms of both the best and worst cases of each model, we aim to determine how different training configurations are influencing the model's abilities to enhance the clarity of corrupted audio signals. A reminder for the order of variables in the model, look for figure 27. Each figure of spectrograms has 3 panels: `Ground Truth` – which represents the original clean audio without any noise, `Ground Truth + Noise` – the clean audio that has been mixed with noise, and `Corrected` – the output of the model after processing the noisy audio and attempting to reconstruct the original clean audio.

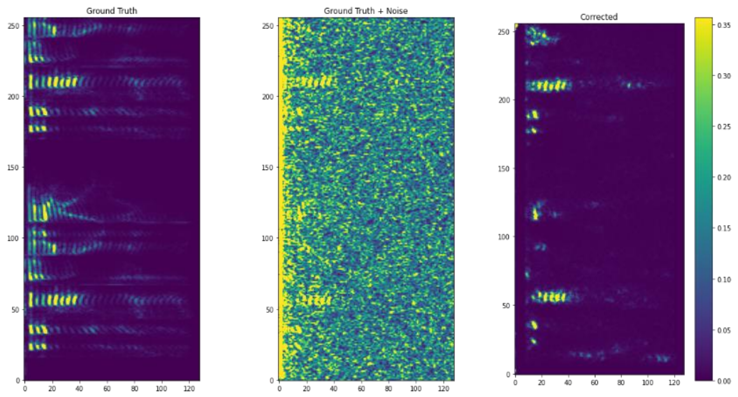


Figure 36. The best case of model_weights_16_16_4_10_0.1_0.5
The highest values of the amplitude are around 0.35 Hz.

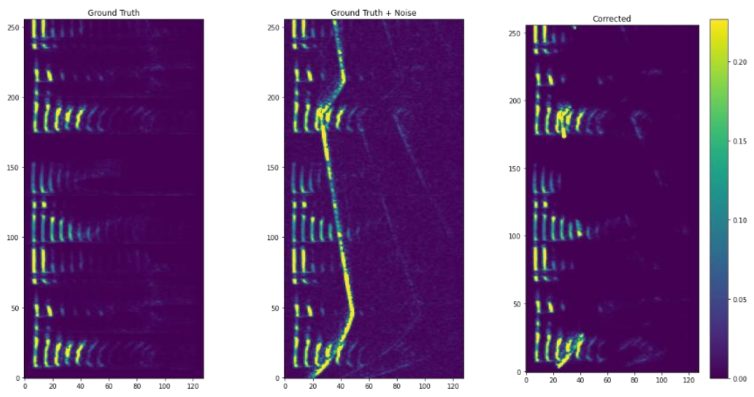


Figure 37. The best case of model_weights_16_16_4_30_0.03_0.6.
The highest values of the amplitude are around 0.25 Hz.

In this comparison, both models have the same number of batches, filters and layers. **Noise Reduction Efficiency**: Both models managed to reduce the noise effectively, as we can see from the `*Ground Truth + Noise*` to the `*Corrected*`. However, the second model (model_weights_16_16_4_30_0.03_0.6) seems to deal with a denser noise environment better, most likely due to the fact that he had more training (30 epochs compared to 10 for the first model), which allowed him to learn more complex noise patterns.

Residual Noise: The first model (model_weights_16_16_4_10_0.1_0.5) did displayed a more uniform noise reduction across the frequency bands. In contrast, the second model, while facing heavier noises, shows better clarity in higher frequencies, but still keeps more noise in the lower frequencies. That can be due to the different urban noise factor values and the amount of training epochs.

Model Training: if we look at the white noise factor of the models (0.1 for the first model and 0.03 for the second), we can say that this might be the reason that the second model could maintain a better fidelity to the clean audio in less random noise conditions, but it did had more urban noise to deal with, with 0.6 for urban factor (0.5 for the first model). This indicates that the second model was optimized for a different kind of noise environment, that also included a real-world sound instead of just synthetic white noise.

Overall, both models demonstrated commendable speech enhancement and noise-handling capabilities, yet there are some performance nuances. If we were looking for a model that can handle more real-world noises as supposed to synthetic white noise, it's better to take a model with a high urban noise factor, as well as a high amount of epochs, for more detailed noise reduction.

For the next comparison, we'll look at spectrograms of the worst case of 3 models with only the amount of epochs changing.

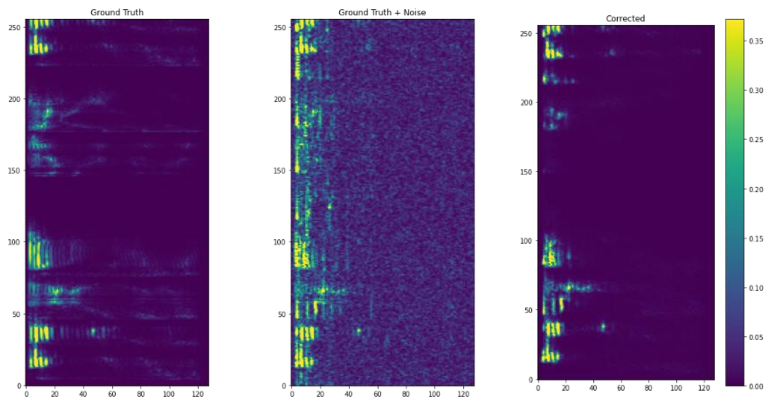


Figure 38. Worst case of model_weights_32_32_6_10_0.1_0.5
some structural details are restored in the corrected spectrogram, but some noise remain in the higher frequencies.

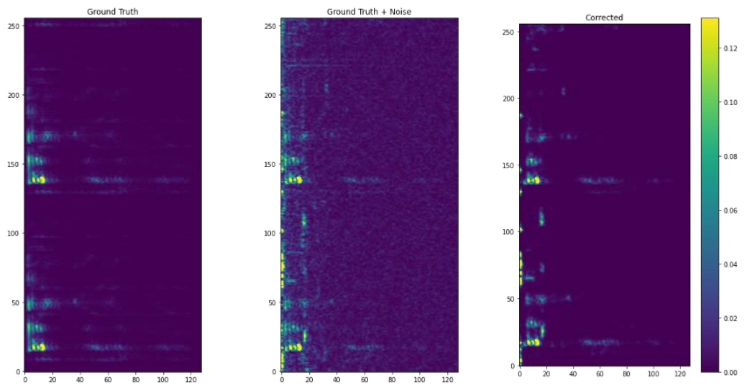


Figure 39. Worst case of model_weights_32_32_6_15_0.1_0.5
The corredted panel shows better noise reduction compared to the first model, with clearer restoration, aspecily in the lower frequencies.

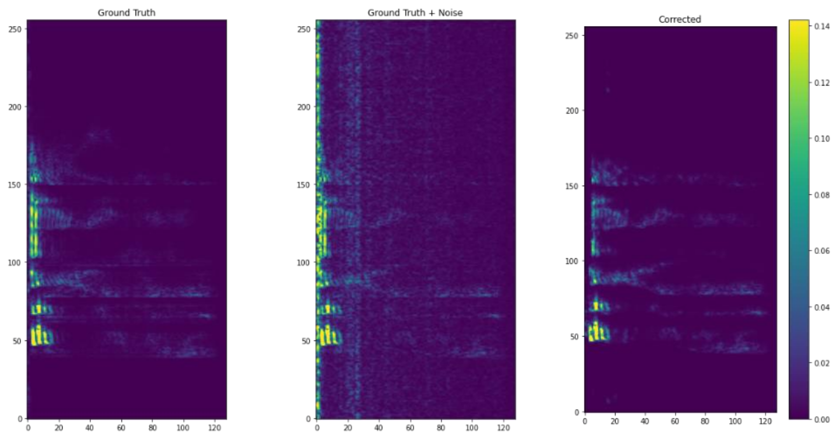


Figure 40. Worst case of model_weights_32_32_6_30_0.1_0.5
Shows improved noise reduction over the first model. The audio retains the original starcute after the enhancment, but some residual noise still present.

Noise Reduction Efficiency: The noise reduction efficiency varies distinctly across all the model, with the model with 30 epochs having the highest proficiency by lowering dramatically the noise patterns, and after him is the model with 15 epochs. This progression highlights the enhanced ability of models to reduce noise effectively, once they trained for longer periods. Even though the model with 30 epochs shows improved noise reduction, it is somewhat comparable to the 15 epochs model, which tells us that if we don’t need that much improvement, maybe an average amount of epochs will be enough. We can see that the max values of the amplitude in the models of 15 and 30 epochs are 0.14 and 0.12 Hz respectively, while the model with 10 epochs reaches around 0.35 Hz.

Residual Noise: The models with more training (15 and 30 epochs) are better at preserving details in the audio signal, especially in lower frequencies where noise typically dominates. Again, the model with the most epochs shows the lowest level of residual noise, it retains more of the original audio quality across a broader range.

Overall, even by looking at the worst case, we can see the models with higher amount of training are preforming and achieving better noise reduction and enhancing the audio, even though it is pretty comparable between the 30 and 15 epochs.

5.4 Time-Domain Waveform Comparison

From the measurements in the table in section 5.2, we selected one of the average models: *model_weights_32_32_6_10_0.1_0.5*, with a PESQ improvement of 0.5512 and SNR improvement of 7.32 dB. After analyzing the model, we built a waveform graph for the best, worst and average cases of audio file improvements. The comparison will be based on the enhanced waveform of each case of the model.

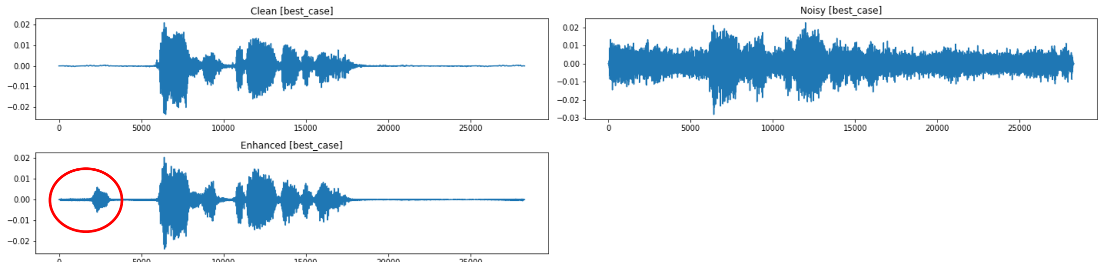


Figure 41. The best case waveform graphs of speech enhancements from the average model

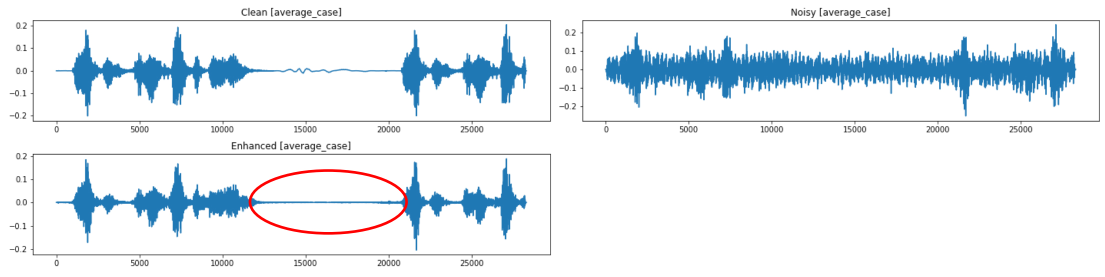


Figure 42. The average case waveform graphs of speech enhancments from the average model

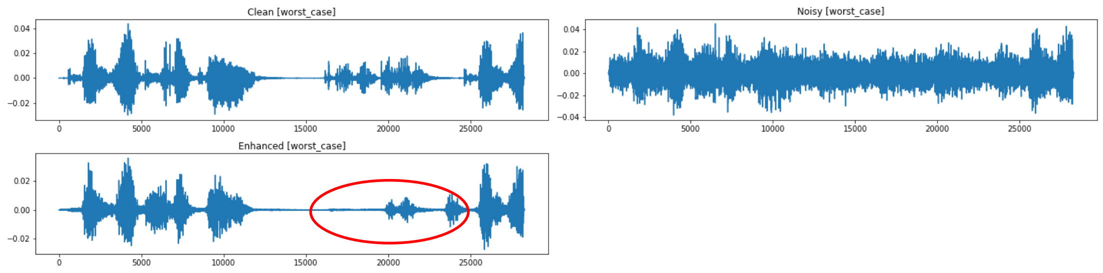


Figure 43. The worst case waveform graphs of speech enhancments from the average model

The best case of the model managed to reduce most of the noise audio, but still left a small portion at the start of the waveform (the red circle at figure 41). This could be an urban noise that the model decided to keep, but overall, the best case managed to maintain its resemblance to the clean original audio. The same can be said about the average case of the model. We can see that most of the noise was reduced, but during the process, it removed a small part of original audio (around the 15000 area of figure 42), probable because that part of audio was similar to the noise. While the best and average cases maintained a close similarity to the original waveform, the worst case removes almost completely the original audio that was mixed with the noise (in the red circle at figure 43). Overall, model_weights_32_32_6_10_0.1_0.5 shows variable effectiveness in enhancing the corrupted audio. Even in the average settings, this model displays a range of speech improvement capabilities, being able to reduce urban and white noise, after training extensively.

5.5 Testing the Factor Limits

To determine the most effective values of noise factor (white and urban), we selected the best model from the table in section 5.2, model_weights_16_16_4_30_0.4_0.4, that has an average improvement of PESQ (0.4324) and the highest SNR improvement (9.52 dB). For testing the limits of the noise factors, we ran this model, while changing the values of the factors.

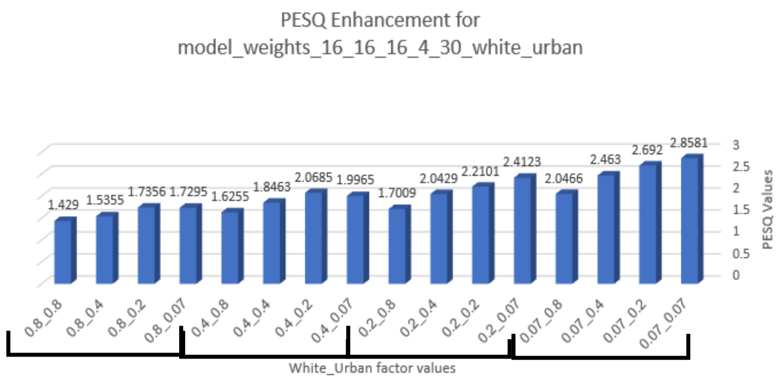


Figure 44. PESQ enhancement graph for model_weights_16_16_16_4_30_white_urban

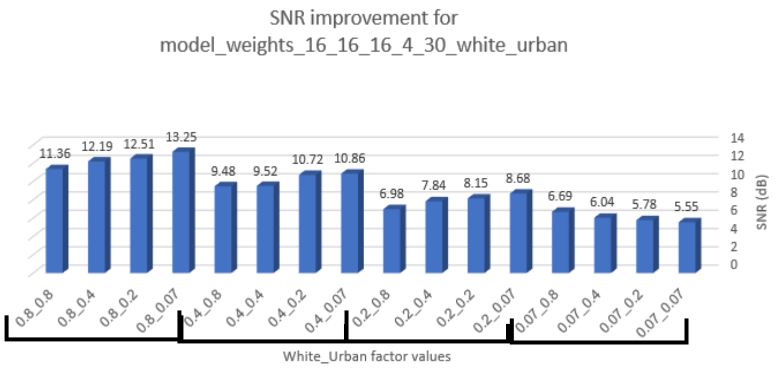


Figure 45. SNR improvement graph for model_weights_16_16_16_4_30_white_urban

As seen from the graphs, with a lower value of the white factor (0.07), the model has an above average PESQ values ranging from 2.0466 to 2.8581, but lacks in the SNR improvement, reaching at maximum 6.69 dB. When the white factor value is 0.4, and the urban factor is lower than 0.4, both the PESQ value and SNR improvements are **higher** than the value of the original best model (0.4_0.4), with the best model (0.4_0.2) reaching PESQ 2.0685 and SNR 10.72 dB. The more we increase the value of the white noise, even though the SNR improvements are much higher, the PESQ enhancement is very low, reaching at most to 1.735. For each group of 4 models (with the same white noise factor value), we can see that the higher the urban noise factor, the lower both the PESQ enhancement and the SNR improvement.
Overall, in order to achieve the best model from this setting and by changing only the factors, it is important to maintain the noise factor around the average, while increasing the factors above 0.7 will result in noises too hard to reduce.

5.6 Testing Process

Test ID	Module	Description	Expected Result
1	Global Variables	Insert value for the variable batch_size	Batch size should be a positive integer (max 256)
2	Global Variables	Insert value for the variable num_of_filters	Filter amount should be a positive integer (max 512)
3	Global Variables	Insert value for the variable num_of_layers	Number of layers should be a positive integer (max 100)
4	Global Variables	Insert value for the variable num_of_epochs	Number of epochs should be a positive integer (max 1000)
5	Global Variables	Insert value for the variable white_noise_factor	White noise factor value should be a positive float (range 0.0 to 1.0)
6	Global Variables	Insert value for the variable urban_noise_factor	Urban noise factor value should be a positive float (range 0.0 to 1.0)
7	Validating Params	Runing validation with defined global variables	If values are in range, print a message and continue creating the model.
8	Model Creation -> conv2d_block	Insert value for the variable dropout_type	Dropout_type must be one of: ['spatial', 'standard']
9	Model Creation -> custom_unet	Create the UNet model using the defined global variables	The model was created with the predefined global variables.
10	Model Creation -> callback_early_stop	Insert value for the variable patience	Patience should be a positive integer (max 1000)
11	Loading saved weights	Running the code with defined global variables	If model is saved in dataset, load the model.
12	Run Model – Training	Running the code with the defined global variables	Strat training the model based on variables.
13	Cascading	Choose 3 different models, that are saved in the dataset for the cascading process	Chosen models for cascading denoise should be different from each other.

14	Cascading	Choose models that aren’t saved in the dataset	Notifying the parameters aren’t in range and wont cascade.
15	Iterative Denoising	Insert value for the variable num_iterations	Num_iterations should be a positive integer (max 100).
16	Run Experiments -> displayed Audio	Run All cells of available experiments	The audio of clean, noisy, enhanced for the best/worst/average cases should be playable by order.
17	Run Experiments	Run All cells of available experiments	Experiment results for analysis should be saved in a desired local output directory in the current session of Kaggle.

6. DOCUMENTATION

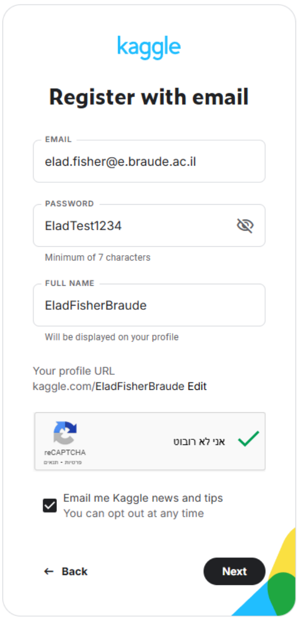
6.1 General Description

The model's notebook is intended to provide a detailed look at the code of our U-Net model for speech enhancement that this paper proposes. The notebook will present cells with documentation and titles to understand the outputs that the user sees. The notebook was written in Kaggle to take advantage of the GPU as well as CPU of the user's computer. Once the model trained with the configuration the user entered, he will be able to see the outputs of audio from the speech enhancement process, as well as the iterative denoising and the other tests that were shown in the paper.

6.2 User Guide

6.2.1 Setting an account at Kaggle Instructions

At the website Kaggle.com, the user will need to register via email or with a google account.



kaggle

Register with email

EMAIL: elad.fisher@e.braude.ac.il

PASSWORD: EladTest1234

Minimum of 7 characters

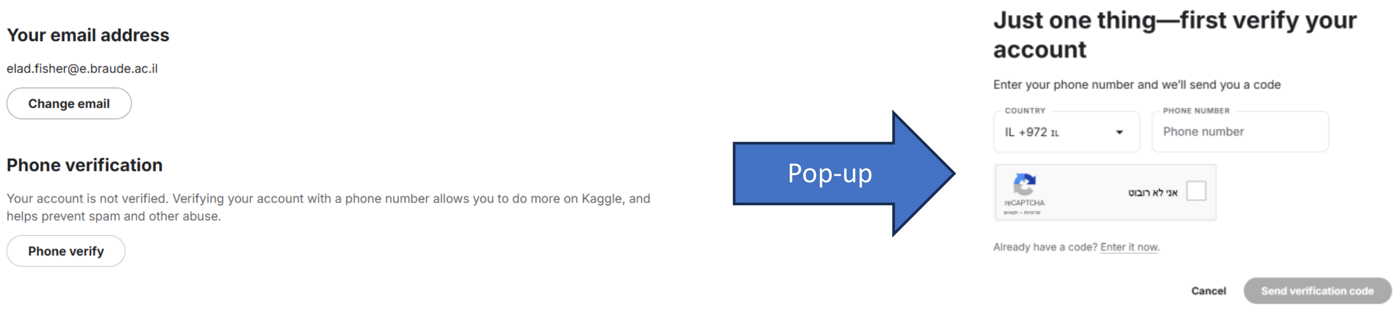
FULL NAME: EladFisherBraude

Will be displayed on your profile

Your profile URL
kaggle.com/EladFisherBraude Edit

Figure 46. Register page at Kaggle

After the email has been verified, the user will need to go to his settings page and verify his phone to have full control of internet connection to run the notebook.



Your email address

elad.fisher@e.braude.ac.il

Phone verification

Your account is not verified. Verifying your account with a phone number allows you to do more on Kaggle, and helps prevent spam and other abuse.

Just one thing—first verify your account

Enter your phone number and we'll send you a code

COUNTRY: IL +972 IL

PHONE NUMBER: Phone number

Already have a code? Enter it now.

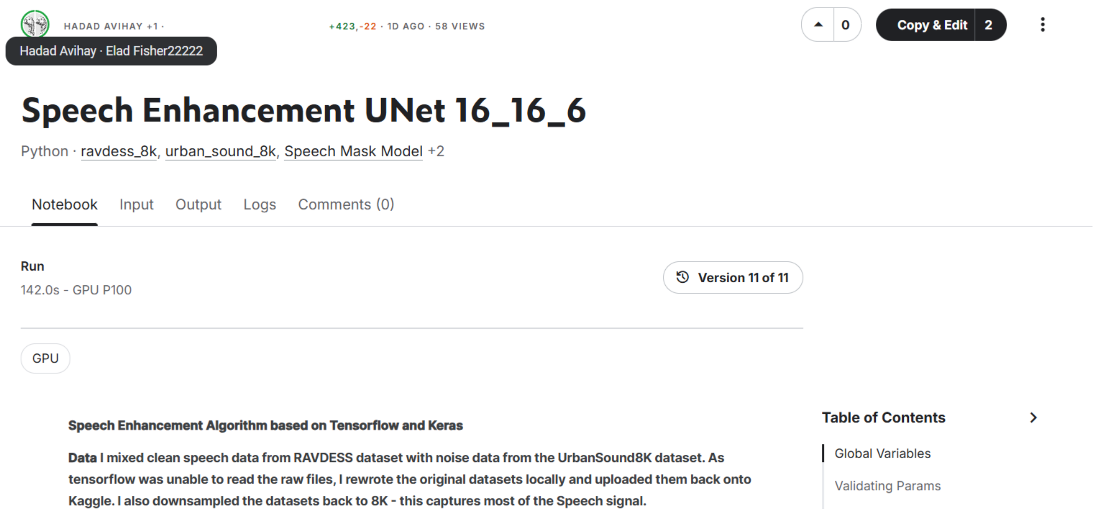
Figure 47. Phone verification process

After that, the user will be able to go to the notebook.

6.2.2 The Model's Notebook Instructions

Kaggle Instructions

open the following link: Speech Enhancement UNet (kaggle.com), and the main page of the notebook will open.



HADAD AVIHAY +1 · +423,-22 · 1D AGO · 58 VIEWS

Hadad Avihay · Elad Fisher22222

Speech Enhancement UNet 16_16_6

Python · ravdess_8k, urban_sound_8k, Speech Mask Model +2

Run
142.0s - GPU P100

Version 11 of 11

GPU

Speech Enhancement Algorithm based on Tensorflow and Keras

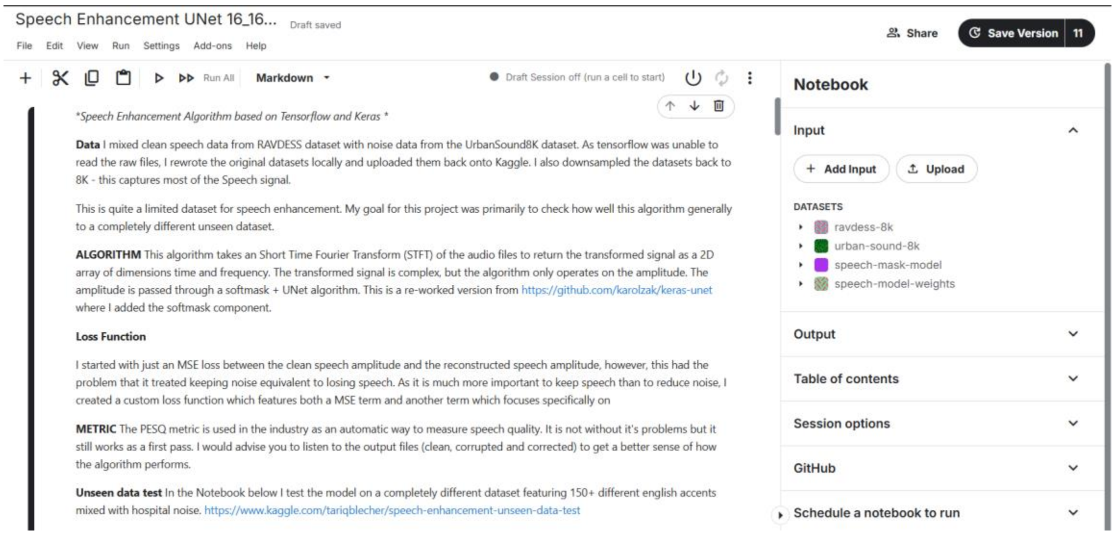
Data I mixed clean speech data from RAVDESS dataset with noise data from the UrbanSound8K dataset. As tensorflow was unable to read the raw files, I rewrote the original datasets locally and uploaded them back onto Kaggle. I also downsampled the datasets back to 8K - this captures most of the Speech signal.

Figure 48. The main page of the notebook of the paper

Once there, you'll need to click on the "Copy & Edit" button to go to a runnable version of the code on your computer.

Notebook Instructions

once you entered the notebook, on the right side of the screen there will be a window titled "Notebook".



Speech Enhancement UNet 16_16... Draft saved

Draft Session off (run a cell to start)

*Speech Enhancement Algorithm based on Tensorflow and Keras *

Data I mixed clean speech data from RAVDESS dataset with noise data from the UrbanSound8K dataset. As tensorflow was unable to read the raw files, I rewrote the original datasets locally and uploaded them back onto Kaggle. I also downsampled the datasets back to 8K - this captures most of the Speech signal.

This is quite a limited dataset for speech enhancement. My goal for this project was primarily to check how well this algorithm generally to a completely different unseen dataset.

ALGORITHM This algorithm takes an Short Time Fourier Transform (STFT) of the audio files to return the transformed signal as a 2D array of dimensions time and frequency. The transformed signal is complex, but the algorithm only operates on the amplitude. The amplitude is passed through a softmask + UNet algorithm. This is a re-worked version from https://github.com/karolzak/keras-unet where I added the softmask component.

Loss Function

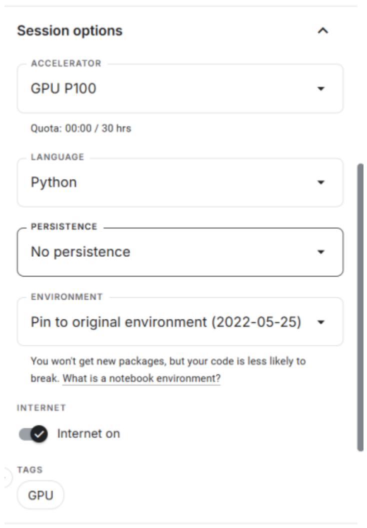
I started with just an MSE loss between the clean speech amplitude and the reconstructed speech amplitude, however, this had the problem that it treated keeping noise equivalent to losing speech. As it is much more important to keep speech than to reduce noise, I created a custom loss function which features both a MSE term and another term which focuses specifically on

METRIC The PESQ metric is used in the industry as an automatic way to measure speech quality. It is not without it's problems but it still works as a first pass. I would advise you to listen to the output files (clean, corrupted and corrected) to get a better sense of how the algorithm performs.

Unseen data test In the Notebook below I test the model on a completely different dataset featuring 150+ different english accents mixed with hospital noise. https://www.kaggle.com/tariqblecher/speech-enhancement-unseen-data-test

Figure 49. The notebook, on the main window is a description of the project and code, on the right is the notebook setting and data window

Open the "Session options" tab, and make sure the toggle "Internet On" is turned on (can be done only after verifying the user's phone, from 6.2.1). You can also select an accelerator on the first dropdown of this tab.



Session options

ACCELERATOR: GPU P100

Quota: 00:00 / 30 hrs

LANGUAGE: Python

PERSISTENCE: No persistence

ENVIRONMENT: Pin to original environment (2022-05-25)

You won't get new packages, but your code is less likely to break. What is a notebook environment?

INTERNET

Internet on

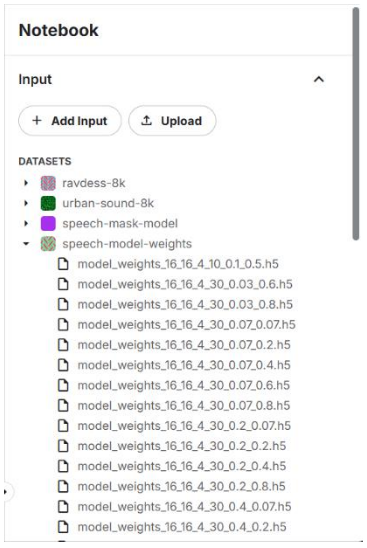
TAGS

GPU

Figure 50. The Session options tab on the right side, on the lower part is the toggle to turn on internet.

Dataset

In the tab "Input", under the "DATASETS", open "speech-model-weights" and you'll be able to see all the existing models that we created during our experiments.



Notebook

Input

+ Add Input | Upload

DATASETS

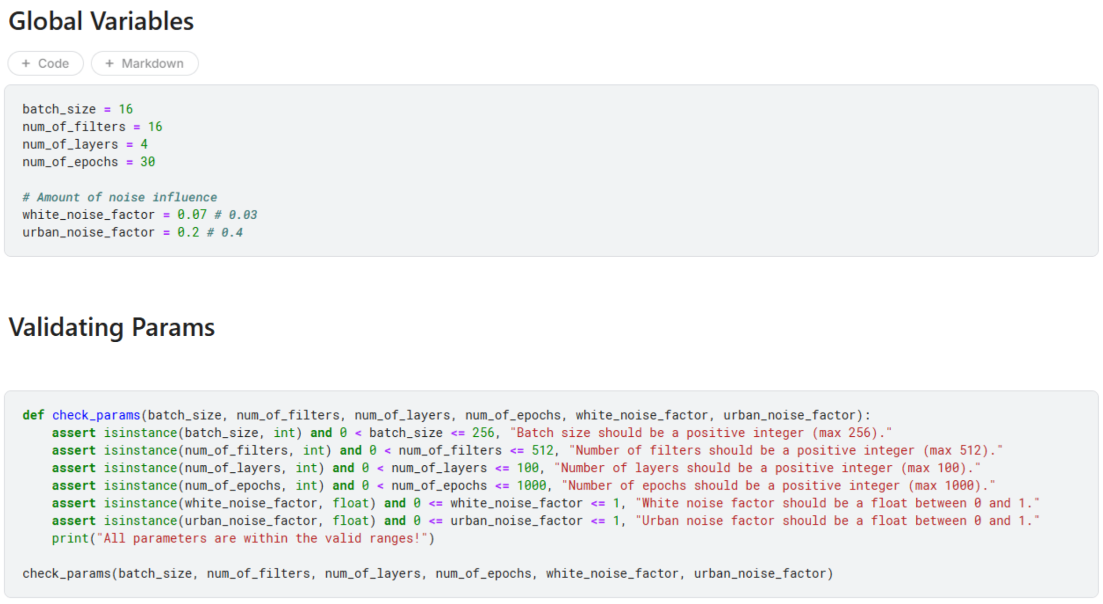
- ravdess-8k
- urban-sound-8k
- speech-mask-model
- speech-model-weights
  - model_weights_16_16_4_10_0.1_0.5.h5
  - model_weights_16_16_4_30_0.03_0.6.h5
  - model_weights_16_16_4_30_0.03_0.8.h5
  - model_weights_16_16_4_30_0.07_0.07.h5
  - model_weights_16_16_4_30_0.07_0.2.h5
  - model_weights_16_16_4_30_0.07_0.4.h5
  - model_weights_16_16_4_30_0.07_0.6.h5
  - model_weights_16_16_4_30_0.07_0.8.h5
  - model_weights_16_16_4_30_0.2_0.07.h5
  - model_weights_16_16_4_30_0.2_0.2.h5
  - model_weights_16_16_4_30_0.2_0.4.h5
  - model_weights_16_16_4_30_0.2_0.8.h5
  - model_weights_16_16_4_30_0.4_0.07.h5
  - model_weights_16_16_4_30_0.4_0.2.h5

Figure 51. The existing configurations of the model.

6.2.3 Code Instruction

Setting variables

The first cell that the user can and should interact with is the "Global Variables" cell, as mentioned in 4.3 (Numerical Study), they are the variables that are setting each configuration of the model. All the variables go through validation before training the model.



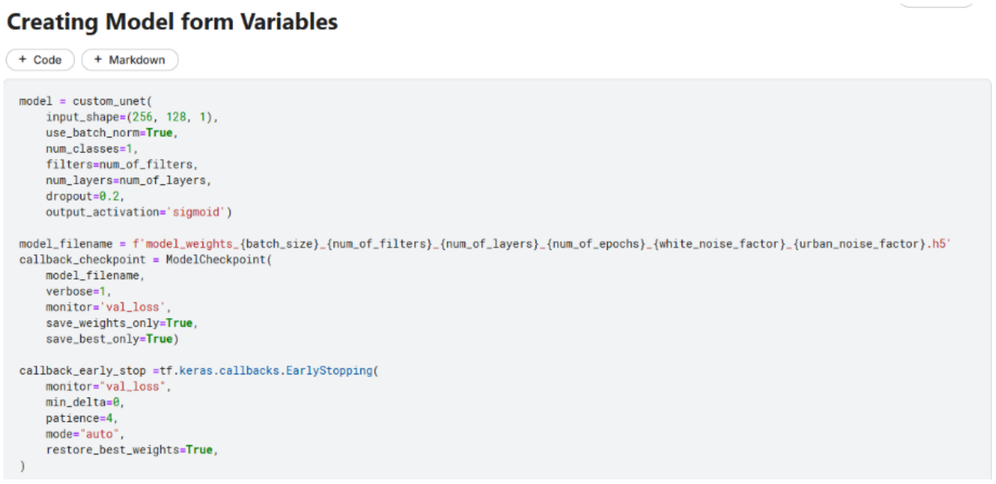
```
batch_size = 16
num_of_filters = 16
num_of_layers = 4
num_of_epochs = 30

# Amount of noise influence
white_noise_factor = 0.07 # 0.03
urban_noise_factor = 0.2 # 0.4
```

```
def check_params(batch_size, num_of_filters, num_of_layers, num_of_epochs, white_noise_factor, urban_noise_factor):
    assert isinstance(batch_size, int) and 0 < batch_size <= 256, "Batch size should be a positive integer (max 256)."
    assert isinstance(num_of_filters, int) and 0 < num_of_filters <= 512, "Number of filters should be a positive integer (max 512)."
    assert isinstance(num_of_layers, int) and 0 < num_of_layers <= 100, "Number of layers should be a positive integer (max 100)."
    assert isinstance(num_of_epochs, int) and 0 < num_of_epochs <= 1000, "Number of epochs should be a positive integer (max 1000)."
    assert isinstance(white_noise_factor, float) and 0 <= white_noise_factor <= 1, "White noise factor should be a float between 0 and 1."
    assert isinstance(urban_noise_factor, float) and 0 <= urban_noise_factor <= 1, "Urban noise factor should be a float between 0 and 1."
    print("All parameters are within the valid ranges!")

check_params(batch_size, num_of_filters, num_of_layers, num_of_epochs, white_noise_factor, urban_noise_factor)
```

Figure 52. Global Variables and Validation cells. Setting the models current configuration and validating the variables.

You can enter values based on the existing configurates of the dataset if you wish to load an existing model instead of training a new one.

If you are planning on training a new model, in the cell titled "Creating Model from Variables" you can set the value of the variable "callback_early_stop", which is in charge of the number of epochs without an improvement to stop the training and continue the code.

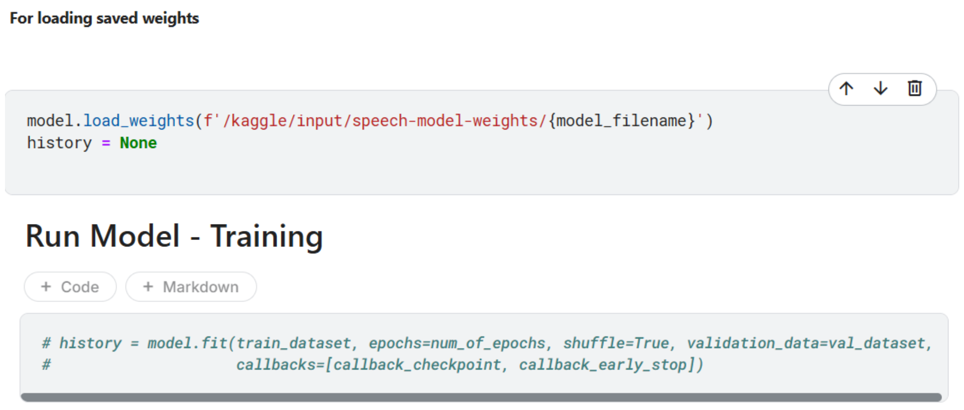


```
model = custom_unet(
    input_shape=(256, 128, 1),
    use_batch_norm=True,
    num_classes=1,
    filters=num_of_filters,
    num_layers=num_of_layers,
    dropout=0.2,
    output_activation='sigmoid')

model_filename = f'model_weights_{batch_size}_{num_of_filters}_{num_of_layers}_{num_of_epochs}_{white_noise_factor}_{urban_noise_factor}.h5'
callback_checkpoint = ModelCheckpoint(
    model_filename,
    verbose=1,
    monitor='val_loss',
    save_weights_only=True,
    save_best_only=True)

callback_early_stop =tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=4,
    mode="auto",
    restore_best_weights=True,
)
```

Figure 53. The cell where you can change the number of epochs without improvement. In "callback_early_stop", change the value of "patience" to the number of epochs you want.

Loading existing models

After entering the values based on the model from the dataset you want to load, go to the cell with the caption "For loading saved weights" and make sure the cell is not a comment. Right next to that cell, there is a cell titled "Run Model – Training", make sure this cell is commented.


For loading saved weights

```
model.load_weights(f'/kaggle/input/speech-model-weights/{model_filename}')
history = None
```

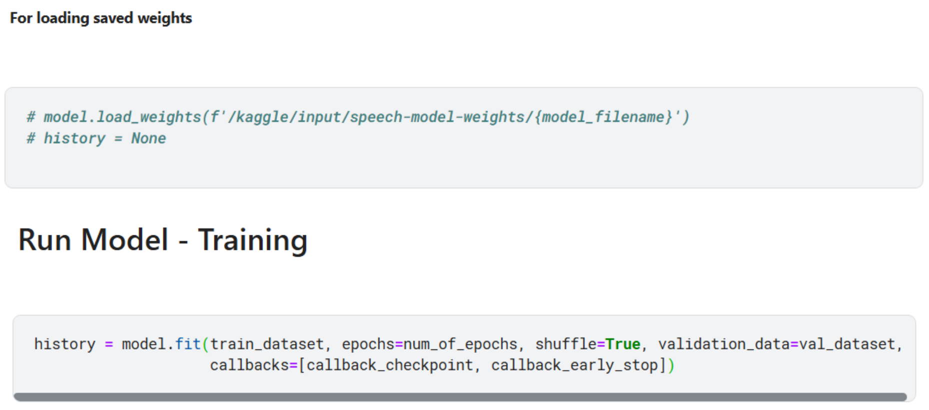
Run Model - Training

```
# history = model.fit(train_dataset, epochs=num_of_epochs, shuffle=True, validation_data=val_dataset,
#                     callbacks=[callback_checkpoint, callback_early_stop])
```

Figure 54. The cells of loading model from the dataset, and training a new model commented, so that once the code runs, it will load the model instead of training.

Creating a new model

Much like loading an existing model, here make sure the "For loading saved weights" is commented and the "Run Model – Training" is not in a comment.


For loading saved weights

```
# model.load_weights(f'/kaggle/input/speech-model-weights/{model_filename}')
# history = None
```

Run Model - Training

```
history = model.fit(train_dataset, epochs=num_of_epochs, shuffle=True, validation_data=val_dataset,
                    callbacks=[callback_checkpoint, callback_early_stop])
```

Figure 55. The cells of loading model from the dataset commented, and training a new model, so that once the code runs, it will train a new model.

Starting the Session

Once you finished setting the variables and the cells to train/load the model, you'll need to turn on the session, this can be done by clicking the "power" icon on the tool bar.

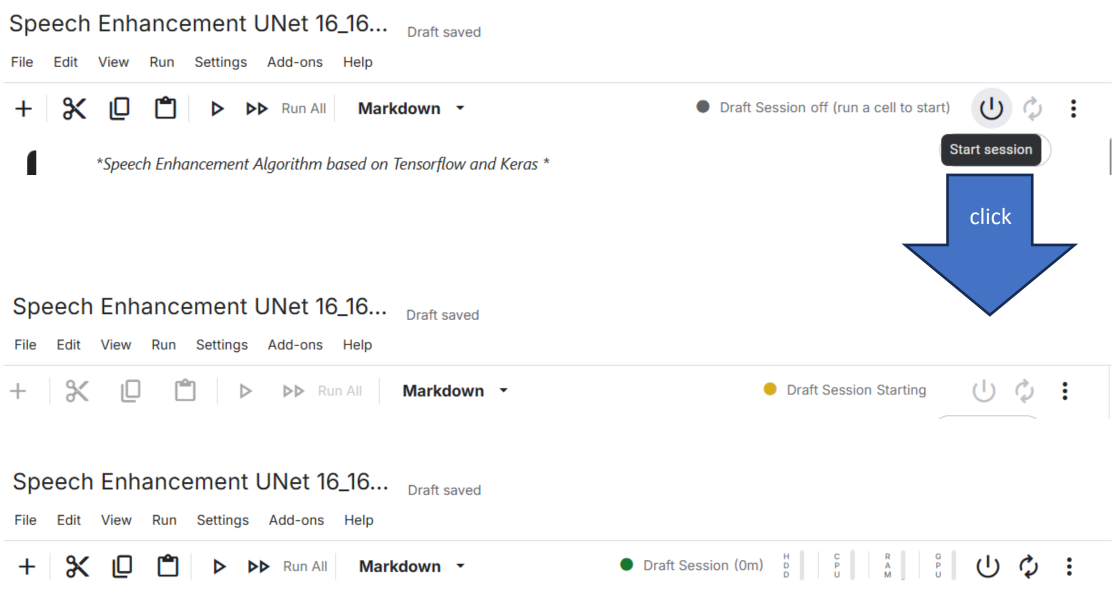

Speech Enhancement UNet 16_16...
Draft saved
File Edit View Run Settings Add-ons Help
Run All
Markdown
Draft Session off (run a cell to start)
Start session
*Speech Enhancement Algorithm based on Tensorflow and Keras *
click
Speech Enhancement UNet 16_16...
Draft saved
File Edit View Run Settings Add-ons Help
Run All
Markdown
Draft Session Starting
Speech Enhancement UNet 16_16...
Draft saved
File Edit View Run Settings Add-ons Help
Run All
Markdown
Draft Session (0m)
HDD
CPU
RAM
GPU

Figure 56. the process to turn on the session.

Running the Code

Once the session has turned on, you can click "Run All" to start the process of training/loading the model and seeing its results.

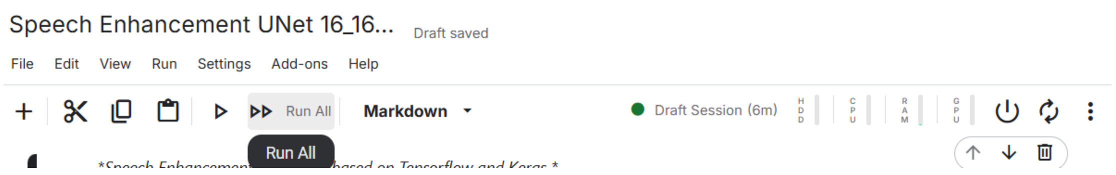
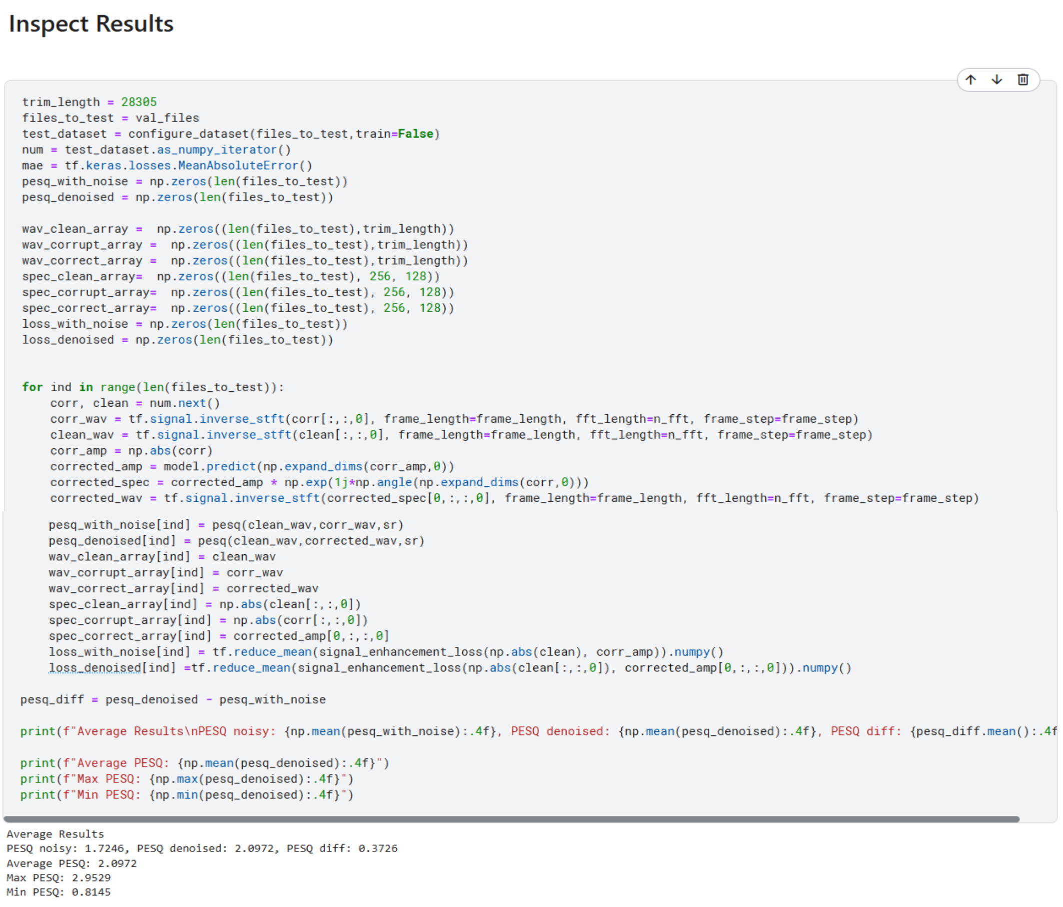

Speech Enhancement UNet 16_16...
Draft saved
File Edit View Run Settings Add-ons Help
Run All
Markdown
Draft Session (6m)
HDD
CPU
RAM
GPU
Run All

Figure 57. Running all the cells of the notebook.

Inspecting Results

After the model was loaded/finished training, you can see the cell "Inspect Results" to see the PESQ improvements of the model in the best and worst cases.



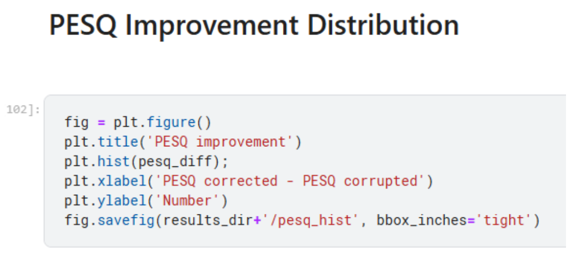
Inspect Results

```
trim_length = 28305
files_to_test = val_files
test_dataset = configure_dataset(files_to_test,train=False)
num = test_dataset.as_numpy_iterator()
mae = tf.keras.losses.MeanAbsoluteError()
pesq_with_noise = np.zeros(len(files_to_test))
pesq_denoised = np.zeros(len(files_to_test))

wav_clean_array =  np.zeros((len(files_to_test),trim_length))
wav_corrupt_array =  np.zeros((len(files_to_test),trim_length))
wav_correct_array =  np.zeros((len(files_to_test),trim_length))
spec_clean_array=  np.zeros((len(files_to_test), 256, 128))
spec_corrupt_array=  np.zeros((len(files_to_test), 256, 128))
spec_correct_array=  np.zeros((len(files_to_test), 256, 128))
loss_with_noise = np.zeros(len(files_to_test))
loss_denoised = np.zeros(len(files_to_test))


for ind in range(len(files_to_test)):
    corr, clean = num.next()
    corr_wav = tf.signal.inverse_stft(corr[:,:,0], frame_length=frame_length, fft_length=n_fft, frame_step=frame_step)
    clean_wav = tf.signal.inverse_stft(clean[:,:,0], frame_length=frame_length, fft_length=n_fft, frame_step=frame_step)
    corr_amp = np.abs(corr)
    corrected_amp = model.predict(np.expand_dims(corr_amp,0))
    corrected_spec = corrected_amp * np.exp(1j*np.angle(np.expand_dims(corr,0)))
    corrected_wav = tf.signal.inverse_stft(corrected_spec[0,:,:,0], frame_length=frame_length, fft_length=n_fft, frame_step=frame_step)

    pesq_with_noise[ind] = pesq(clean_wav,corr_wav,sr)
    pesq_denoised[ind] = pesq(clean_wav,corrected_wav,sr)
    wav_clean_array[ind] = clean_wav
    wav_corrupt_array[ind] = corr_wav
    wav_correct_array[ind] = corrected_wav
    spec_clean_array[ind] = np.abs(clean[:,:,0])
    spec_corrupt_array[ind] = np.abs(corr[:,:,0])
    spec_correct_array[ind] = corrected_amp[0,:,:,0]
    loss_with_noise[ind] = tf.reduce_mean(signal_enhancement_loss(np.abs(clean), corr_amp)).numpy()
    loss_denoised[ind] =tf.reduce_mean(signal_enhancement_loss(np.abs(clean[:,:,0]), corrected_amp[0,:,:,0])).numpy()

pesq_diff = pesq_denoised - pesq_with_noise

print(f"Average Results\nPESQ noisy: {np.mean(pesq_with_noise):.4f}, PESQ denoised: {np.mean(pesq_denoised):.4f}, PESQ diff: {pesq_diff.mean():.4f

print(f"Average PESQ: {np.mean(pesq_denoised):.4f}")
print(f"Max PESQ: {np.max(pesq_denoised):.4f}")
print(f"Min PESQ: {np.min(pesq_denoised):.4f}")
```

```
Average Results
PESQ noisy: 1.7246, PESQ denoised: 2.0972, PESQ diff: 0.3726
Average PESQ: 2.0972
Max PESQ: 2.9529
Min PESQ: 0.8145
```

Figure 58. The result inspection cell, where we can see the average results of PESQ improvement in the model.

After that, every cell shows a specific part of the results that we can learn and examine.



PESQ Improvement Distribution

```
[102]:
fig = plt.figure()
plt.title('PESQ improvement')
plt.hist(pesq_diff);
plt.xlabel('PESQ corrected - PESQ corrupted')
plt.ylabel('Number')
fig.savefig(results_dir+'/pesq_hist', bbox_inches='tight')
```

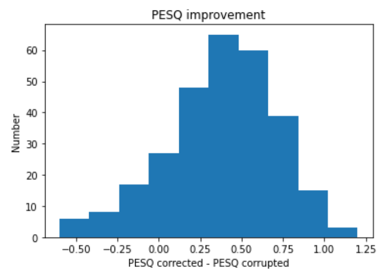
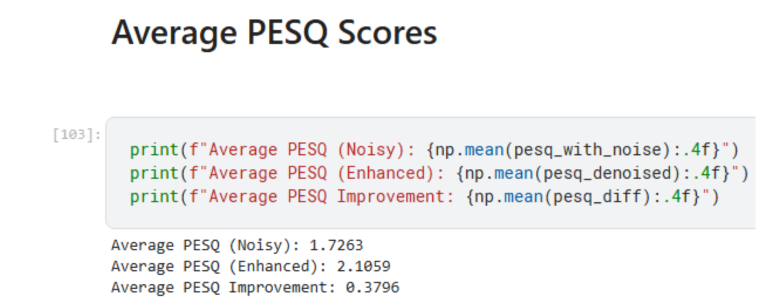


Figure 59. PESQ Improvement cell, shows us a graph of the number of results from the model that achieved a PESQ improvement value.



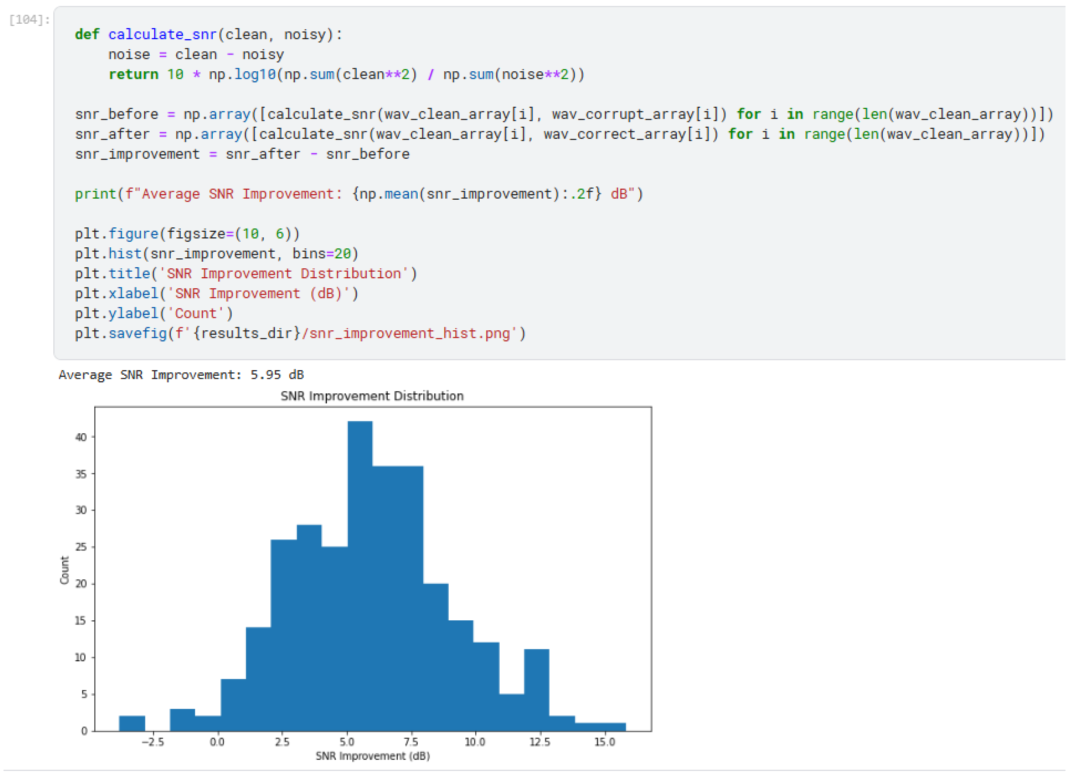
Average PESQ Scores

```
[103]:
print(f"Average PESQ (Noisy): {np.mean(pesq_with_noise):.4f}")
print(f"Average PESQ (Enhanced): {np.mean(pesq_denoised):.4f}")
print(f"Average PESQ Improvement: {np.mean(pesq_diff):.4f}")
```

```
Average PESQ (Noisy): 1.7263
Average PESQ (Enhanced): 2.1059
Average PESQ Improvement: 0.3796
```

Figure 60. The average PESQ scores of the model before and after the enhancement.

Signal-to-Noise Ratio (SNR) Improvement

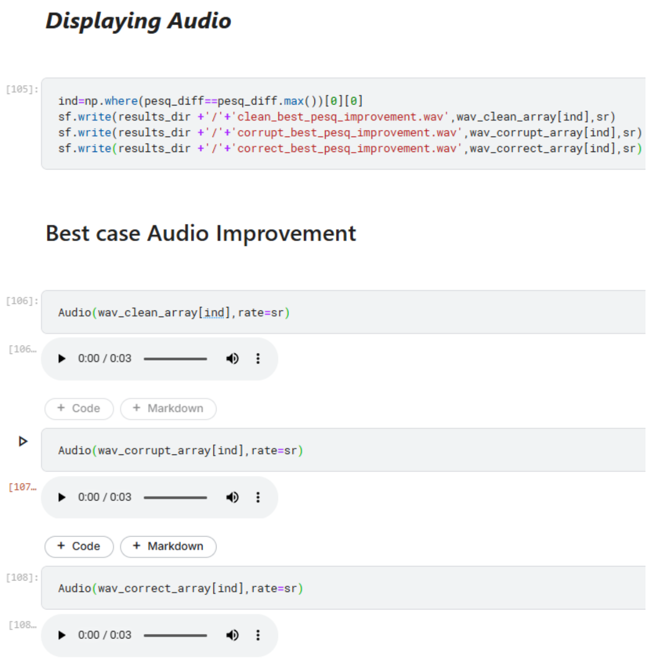


```
[104]:
def calculate_snr(clean, noisy):
    noise = clean - noisy
    return 10 * np.log10(np.sum(clean**2) / np.sum(noise**2))

snr_before = np.array([calculate_snr(wav_clean_array[i], wav_corrupt_array[i]) for i in range(len(wav_clean_array))])
snr_after = np.array([calculate_snr(wav_clean_array[i], wav_correct_array[i]) for i in range(len(wav_clean_array))])
snr_improvement = snr_after - snr_before

print(f"Average SNR Improvement: {np.mean(snr_improvement):.2f} dB")

plt.figure(figsize=(10, 6))
plt.hist(snr_improvement, bins=20)
plt.title('SNR Improvement Distribution')
plt.xlabel('SNR Improvement (dB)')
plt.ylabel('Count')
plt.savefig(f'{results_dir}/snr_improvement_hist.png')
```

```
Average SNR Improvement: 5.95 dB
```

Figure 61. A cell showing the SNN improvement of every result in the model in dB.

After examining the model, the following cells are displaying a test audio from the dataset before the corruption of noise, after the corruption of noise to the audio and after the model corrects the audio.



Displaying Audio

```
[105]:
ind=np.where(pesq_diff==pesq_diff.max())[0][0]
sf.write(results_dir +'/'+'clean_best_pesq_improvement.wav',wav_clean_array[ind],sr)
sf.write(results_dir +'/'+'corrupt_best_pesq_improvement.wav',wav_corrupt_array[ind],sr)
sf.write(results_dir +'/'+'correct_best_pesq_improvement.wav',wav_correct_array[ind],sr)
```

Best case Audio Improvement

```
[106]:
Audio(wav_clean_array[ind],rate=sr)
```

```
[106…
0:00 / 0:03
```

```
Audio(wav_corrupt_array[ind],rate=sr)
```

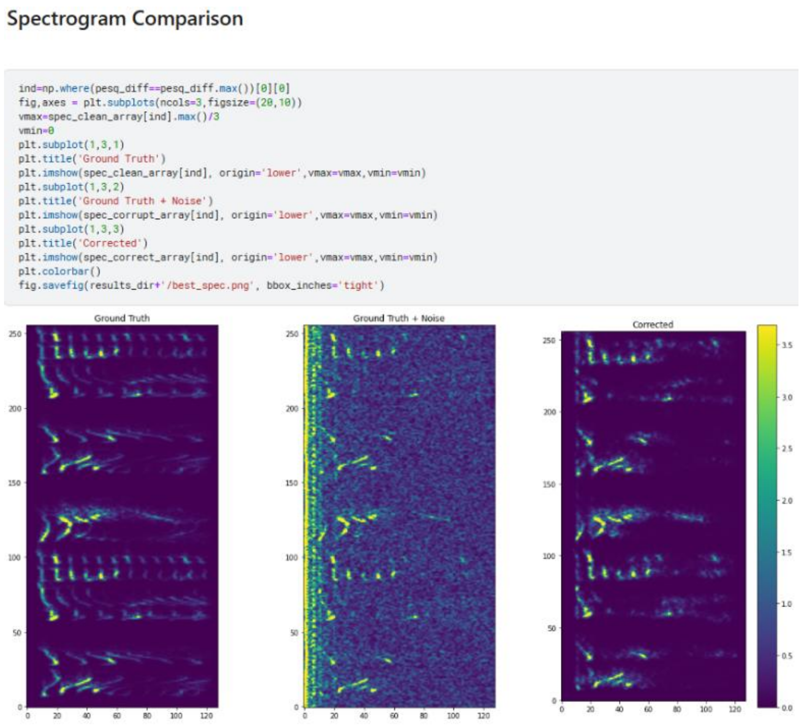
```
[107…
0:00 / 0:03
```

```
[108]:
Audio(wav_correct_array[ind],rate=sr)
```

```
[108…
0:00 / 0:03
```

Figure 62. The cells in charge of displaying the audio of the best case of the model of clean, noisy and enhanced audio. There are other cells with the average case and worst cases.

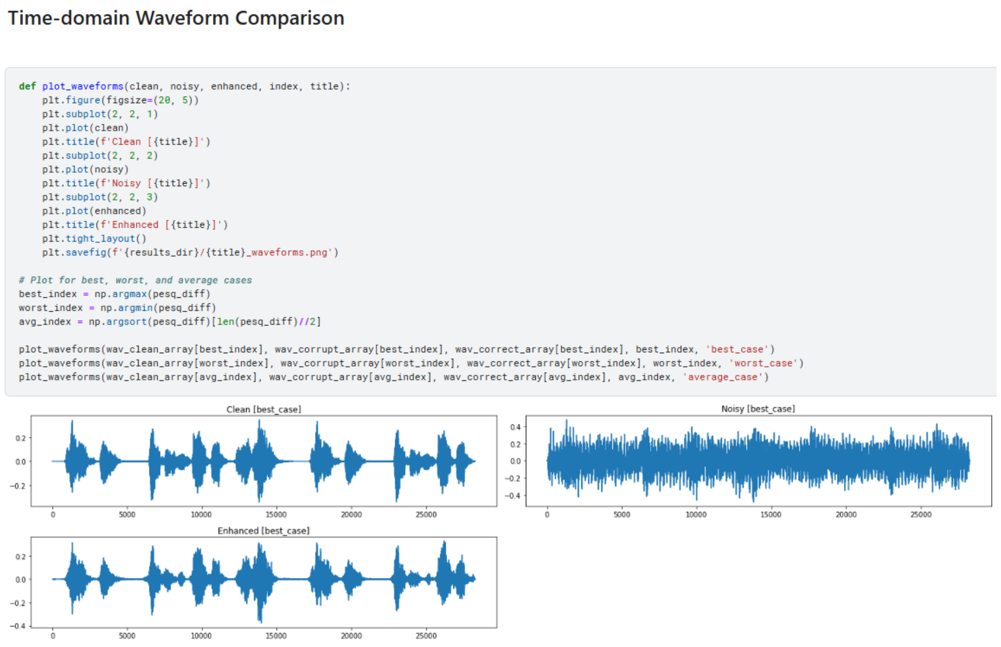
By the end, there are cells that shows the user spectrograms of the audio, clean, noisy and enhanced, for both the best and worst case. Those cells are under the title "Spectrogram Comparison". After that there is a cell called "Time-domain Waveform Comparison" that shows the waveforms of the models best, average and worst cases of the audio: clean, noisy and enhanced.



Spectrogram Comparison

```
ind=np.where(pesq_diff==pesq_diff.max())[0][0]
fig,axes = plt.subplots(ncols=3,figsize=(20,10))
vmax=spec_clean_array[ind].max()/3
vmin=0
plt.subplot(1,3,1)
plt.title('Ground Truth')
plt.imshow(spec_clean_array[ind], origin='lower',vmax=vmax,vmin=vmin)
plt.subplot(1,3,2)
plt.title('Ground Truth + Noise')
plt.imshow(spec_corrupt_array[ind], origin='lower',vmax=vmax,vmin=vmin)
plt.subplot(1,3,3)
plt.title('Corrected')
plt.imshow(spec_correct_array[ind], origin='lower',vmax=vmax,vmin=vmin)
plt.colorbar()
fig.savefig(results_dir+'/best_spec.png', bbox_inches='tight')
```

Figure 63. The cell that compares spectrograms of the audio enhancement process.



Time-domain Waveform Comparison

```
def plot_waveforms(clean, noisy, enhanced, index, title):
    plt.figure(figsize=(20, 5))
    plt.subplot(2, 2, 1)
    plt.plot(clean)
    plt.title(f'Clean [{title}]')
    plt.subplot(2, 2, 2)
    plt.plot(noisy)
    plt.title(f'Noisy [{title}]')
    plt.subplot(2, 2, 3)
    plt.plot(enhanced)
    plt.title(f'Enhanced [{title}]')
    plt.tight_layout()
    plt.savefig(f'{results_dir}/{title}_waveforms.png')

# Plot for best, worst, and average cases
best_index = np.argmax(pesq_diff)
worst_index = np.argmin(pesq_diff)
avg_index = np.argsort(pesq_diff)[len(pesq_diff)//2]

plot_waveforms(wav_clean_array[best_index], wav_corrupt_array[best_index], wav_correct_array[best_index], best_index, 'best_case')
plot_waveforms(wav_clean_array[worst_index], wav_corrupt_array[worst_index], wav_correct_array[worst_index], worst_index, 'worst_case')
plot_waveforms(wav_clean_array[avg_index], wav_corrupt_array[avg_index], wav_correct_array[avg_index], avg_index, 'average_case')
```

Figure 64. The cell that compares the waveforms of the audio enhancement process.

6.3 Maintenance Guide

After registering to Kaggle and setting your account (explanation is on section 6.2.1), the user can edit each cell in the project's notebook. Once the edits are done, the user can run the code (as shown in 6.2.3) and observe the changes in the behaver of the code. In case of edits causing the code to crash, we suggest copying and opening a new copy of our notebook of the project and work on that version.

7. REFERENCEES

[1] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[2] Chaudhary, N., Misra, S., Kalamkar, D., Heinecke, A., Georganas, E., Ziv, B., ... & Kaul, B. (2021). Efficient and generic 1d dilated convolution layer for deep learning. arXiv preprint arXiv:2104.08002.

[3] Dauphin, Y. N., Fan, A., Auli, M., & Grangier, D. (2017, July). Language modeling with gated convolutional networks. In International conference on machine learning (pp. 933-941). PMLR.

[4] Defossez, A., Synnaeve, G., & Adi, Y. (2020). Real time speech enhancement in the waveform domain. arXiv preprint arXiv:2006.12847.

[5] Deng, F., Jiang, T., Wang, X., Zhang, C., & Li, Y. (2020, October). NAAGN: Noise-Aware Attention-Gated Network for Speech Enhancement. In Interspeech (pp. 2457-2461).

[6] Geng, C., & Wang, L. (2020, June). End-to-end speech enhancement based on discrete cosine transform. In 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA) (pp. 379-383). IEEE.

[7] Giri, R., Isik, U., & Krishnaswamy, A. (2019, October). Attention wave-u-net for speech enhancement. In 2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA) (pp. 249-253). IEEE.

[8] Gu, J., Lu, L., Cai, R., Zhang, H. J., & Yang, J. (2005). Dominant feature vectors based audio similarity measure. In Advances in Multimedia Information Processing-PCM 2004: 5th Pacific Rim Conference on Multimedia, Tokyo, Japan, November 30-December 3, 2004. Proceedings, Part II 5 (pp. 890-897). Springer Berlin Heidelberg.

[9] Kang, Z., Huang, Z., & Lu, C. (2022). Speech enhancement using u-net with compressed sensing. Applied Sciences, 12(9), 4161.

[10] Macartney, C., & Weyde, T. (2018). Improved speech enhancement with the wave-u-net. arXiv preprint arXiv:1811.11307.

[11] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18 (pp. 234-241). Springer International Publishing.

[12] Stoller, D., Ewert, S., & Dixon, S. (2018). Wave-u-net: A multi-scale neural network for end-to-end audio source separation. arXiv preprint arXiv:1806.03185.

[13] Audio Denoiser: A Speech Enhancement Deep Learning Model (analyticsvidhya.com)

[14] Attention Mechanism In Deep Learning (analyticsvidhya.com)

[15] Attention Networks: A simple way to understand Self Attention (medium.com)

[16] Audio Enhancement and Denoising Methods (medium.com)

[17] Audio journey part 5 : Audio, time or frequency domain (medium.com)

[18] Channel Attention Module (paperswithcode.com)

[19] Convolution in one dimension for neural networks (e2eml.school)

[20] Convolutional Neural Networks, Explained (towardsdatascience.com)

[21] Data Driven Science & Engineering Machine Learning, Dynamical Systems, and Control - University of Washington (databookuw.com)

[22] Gated Linear Unit — Enabling stacked convolutions to out-perform RNNs (medium.com)

[23] Practical Introduction to Time-Frequency Analysis (mathworks.com)

[24] Pressure Wave (ametsoc.org)

[25] Review: DilatedNet — Dilated Convolution (towardsdatascience.com)

[26] Short-Time Fourier Transform (sciencedirect.com)

[27] Signal Representation — Time Domain versus Frequency Domain (medium.com)

[28] Sound Wave (techtarget.com)

[29] Spatial Attention Module (paperswithcode.com)

[30] Time Domain Analysis vs Frequency Domain Analysis (cadence.com)

[31] Transposed Convolution (d2l.ai)

[32] Understand Transposed Convolutions (towardsdatascience.com)

[33] What Is Denoising? (blogs.nvidia.com)

[34] what-is-dilated-convolution (educative.io)

[35] What is Transposed Convolutional Layer? (geeksforgeeks.org)

[36] Perceptual evaluation of speech quality (PESQ) (itu.int)

[37] What is Signal to Noise Ratio and How to calculate it? (cadence.com)

[38] RAVDESS Emotional speech audio (kaggle.com)

[39] urbansound8k dataset (urbansounddataset.weebly.com)