# Automatic Speech Recognition-Driven Speech Enhancement

Deep Neural Network

Master Thesis

Michail Kampitakis
Group 1074

Aalborg University
Electronics and IT

**Electronics and IT**
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

**Title:**
Automatic Speech Recognition-Driven Speech Enhancement Deep Neural Network

**Theme:**
Scientific Theme

**Project Period:**
Spring Semester 2023

**Project Group:**
1074

**Participant(s):**
Michail Kampitakis

**Supervisor(s):**
Iván López Espejo
Jesper Jensen
Zheng-Hua Tan

**Copies:** 1

**Page Numbers:** 51

**Date of Completion:**
June 1, 2023

**Abstract:**

Speech Enhancement systems improve the quality and intelligibility of noisy speech signals. It has been proved that conventional loss functions such as MSE do not correlate highly with how humans perceive speech, and they do not perform well in subjective listening tests. On the contrary, objective metrics used as loss functions show better performance on objective tests. However, there is not always a correlation between the subjective and objective intelligibility tests. In this Thesis, an Automatic Speech Recognition (ASR) model is employed as a loss function in the Speech Enhancement system, aiming at closing the gap in intelligibility performance between subjective and objective tests. The hypothesis is that minimizing the Word Error Rate of a noisy speech signal will improve intelligibility in both cases.

# Contents

# Preface

I want to thank my supervisors, Iván, Jesper and Zheng-Hua, for their excellent supervision and guidance during the writing and implementation processes and for providing useful tools and parts of the code.

Aalborg University, June 1, 2023

Michail Kampitakis
<mkampi21@student.aau.dk>

# Chapter 1

# Introduction

## 1.1 Speech Enhancement

Environments with ambient noise, reverberation, and multiple competing speakers distort the desired speech signal, making it harder for the listener to understand. This is even more intense in cases where processing is required. Such examples can be for people that are dependent on hearing aid devices, telecommunications including a wide range of applications, from everyday calls in mild noise environments (online meetings in big offices, making a call during public transportation, etc.) to communication for professionals working in dangerously loud conditions (airport workers, firefighters, etc.), or other technologies such as Automatic Speech Recognition (ASR) which is a speech to text algorithm. Hearing aid and other devices do not focus sufficiently on the target speaker while separating the noisy background conversations, widely known as the cocktail party problem. It remains a challenging problem to solve even with today's state-of-the-art methods [1].

Speech enhancement aims to solve this problem. It is a process where the perceptual quality and intelligibility of the observed noisy speech signal are improved by trying to retrieve the original clean signal. This process may find use in the cases mentioned, such as hearing aid devices, telecommunications, automatic speech recognition, etc. An example of this process can be seen in Figure 1.1

The most reliable way of measuring the performance of a speech enhancement algorithm is by performing subjective listening tests. However, this way is time-consuming and requires trained listeners to evaluate the processed signals. Therefore, researchers try to set objective metrics that can evaluate performance. The processing level of these metrics varies, as there are low-level processing metrics, psychoacoustics, and higher-level ones, such as linguistics, etc. [2]. Therefore, the performance of a Speech Enhancement System can be measured with quality and intelligibility metrics such as:

- Time-Domain Mean-Square Error (MSE) - Waveform-Match Metric

- Signal-to-Distortion Ratio (SDR) - Waveform-Match Metric
- Scale-Invariant Signal-to-Distortion Ratio (SI-SDR) - Waveform-Match Metric
- Perceptual Evaluation of Speech Quality (PESQ) - Quality Metric
- Short-Time Objective Intelligibility (STOI) - Intelligibility Metric
- Extended STOI (ESTOI) - Intelligibility Metric

Figure 1.1 illustrates the general use of a speech enhancement system using a Deep Neural Network (DNN). The speaker's speech is mixed with background noise, and the trained DNN system tries to retrieve the original speech signal. As machine learning and, specifically, Neural Networks show more and more potential, outperforming other methods, these objective metrics could be used for training a Neural Network (NN) architecture. Therefore, training a model to achieve higher scores in the quality metrics can also improve the overall intelligibility of the system in subjective tests. This, however, is not always the case in intelligibility tests [3]–[9].

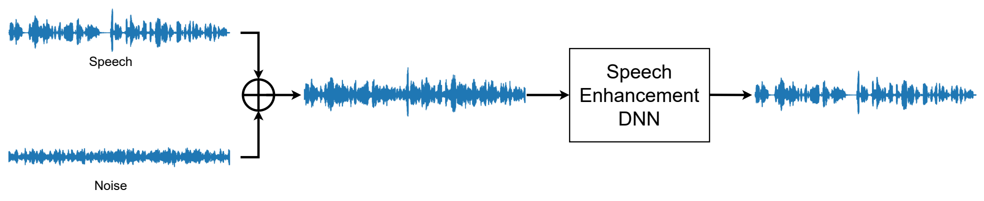


**Figure 1.1:** Signal model of a Speech Enhancement System.

## 1.2 State-of-the-Art

One of the first approaches for speech enhancement was simple solutions such as spectral subtractive algorithms, which learn the spectral properties of a noisy signal during the absence of speech signals and try to subtract. Other approaches include statistical information of the signal, such as Wiener filters and statistical models, such as Maximum Likelihood, MMSE, etc. Additionally, euclidean algebra algorithms were also widely used, as they assume that noise resides in a subspace of the signal and can be removed by using decomposition [2].

Machine Learning approaches started becoming popular over time as the technology progressed. The reason behind the transition to DNN models was to better analyse the noise, as the previously suggested methods suffer from either poor performance in unseen conditions or from artifacts after the signal is enhanced. The first suggested architectures used DNN models with features of spectral energy [10], [11].

With the DNN models becoming more popular in speech enhancement, DNN implementations using autoencoder structures have been proposed for signals in the time domain. These models accept time-domain signals and return outputs also in time-domain. The key have been found to be loss functions in the frequency domain, which have been proven to work well with speech enhancement. This method has the benefit of the model learning how changes in the frequency domain affect the time domain since the process is end-to-end [12], [13].

Regarding the loss function used in the training process, it was found that using conventional methods that perform well in other tasks, such as Mean Square Error (MSE), does not result in a good performance in speech enhancement, as the results do not perform well in subjective and objective metrics [3], [14]. On the other hand, using human perception-based criteria for optimizing the model shows better results with improved quality and intelligibility [3], [6], [7], [9], [14], [15].

## 1.3 Motivation and Problem Description

It is well known that end-to-end time-domain DNN models tend to perform significantly better than the ones that operate in the magnitude spectral domain [12], [13]. Additionally, as mentioned, subjective metrics can better evaluate the performance of a speech enhancement algorithm. Objective intelligibility and quality metrics can be sufficient; however, there is not always a correlation between objective intelligibility and subjective performance, meaning that even though the objective metrics might show an improvement in the intelligibility of the speech signal, subjective metrics could show otherwise [15].

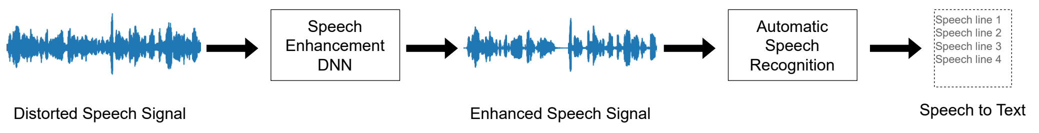


**Figure 1.2:** Signal flow graph of Speech Enhancement System explored in this Thesis.

The hypothesis of this Thesis lies in the fact that using an Automatic Speech Recognition (ASR) system as a loss function in a model architecture that has been proven to work well with speech enhancement could eventually close the gap between subjective and objective metrics. The system in theory could play the role of a trained listener based on which the speech enhancement system is optimized during the training process for maximizing overall perceptual intelligibility of speech signals. Additionally, it has been proven that the Word Error Rate of the ASR model has an inverse correlation to intelligibility metrics, such as STOI and quality metrics, such as PESQ [16]. In other words, a lower Word Error Rate in ASR ensures better scores in these metrics. Employing an ASR model as an objective to the Speech Enhancement model can be beneficial for two main reasons. Firstly, in contrast with the actual subjective evaluation test, the ASR method is neither time-consuming nor laborious. Secondly, similarly to objective tests, it allows the use of mathematical procedures to determine how well the system performs. Figure 1.2 illustrates the general idea of the process. First, the signal is enhanced by the speech enhancement model and then the enhanced speech signal is fed into the ASR system.

Training a Speech Enhancement model with ASR metrics as objectives, such as Character Error Rate (CER) and Word Error Rate (WER), could improve the intelligibility of the estimated signals. Hypothetically by doing so, the system would not only score high in objective intelligibility and quality metrics, but, more importantly, it would also score better in subjective intelligibility and quality metrics.

# Chapter 2

# Theoretical Background

This chapter discusses the theoretical background of Machine Learning that is required for explaining the methods that were used in this Thesis. Specifically, forward propagation in DNN models is explained and how the weights are updated using optimizers that they minimize a given loss function. Different types of models are further discussed, such as convolutional models and recurrent models, as well as the cases that perform well. Finally, a specific case of Automatic Speech Recognition (ASR) and the CTC process which is often used as a loss function in training ASR models are discussed.

## 2.1 Machine Learning and Neural Networks

Machine Learning is a data-driven process where data are used to train a model to perform certain tasks. In this Thesis, supervised learning is used, in which the models are trained using labelled data (target $t$), where the desired outcome is known.

### 2.1.1 Neural Networks

Neural Networks consist of $M$ linear combinations of fixed non-linear basis functions $\phi_j(x)$ as shown in the Equation 2.1. Usually, function $f(\cdot)$ of the equation is a non-linear activation function in the case of classification methods and an identity function in the case of regression. The function $\phi_j(x)$ could be either a linear or a non-linear function, and its parameters of the function are adjustable. This extends the model with weight coefficients $w_j$, which are initially randomized and adjusted during training. This is illustrated in Equation 2.2, where the superscripts $(1)$ and $(2)$ indicate which layer the weight coefficients belong to. The second layer of this example is the output layer, and the subscript variable $k$ belongs to the set

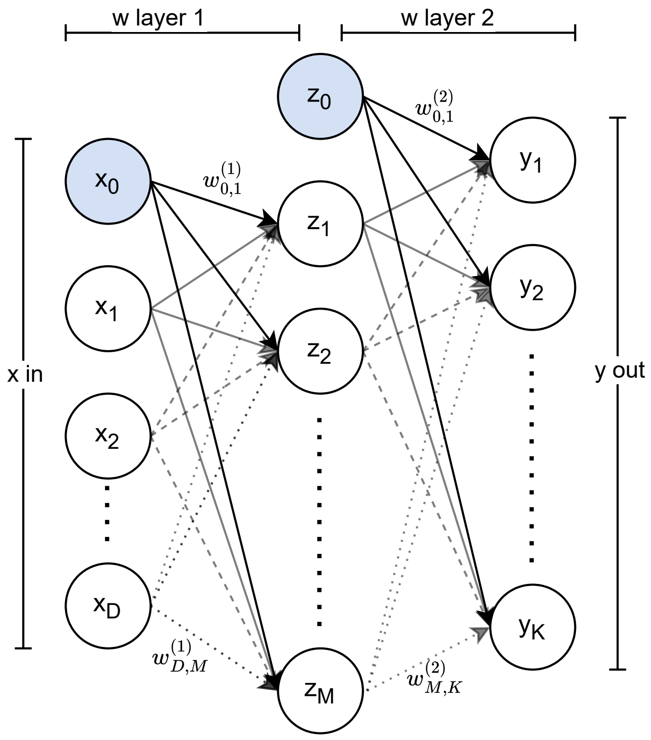


**Figure 2.1:** Neural Network as described in Equation 2.2, with the nodes $x_0$ and $z_0$ being the biases. Superscripts in weights illustrate the layer they belong to and subscripts the node connection [17].

$\{1, \cdots, K\}$ where $K$ is the total amount of outputs [17]. A diagram of the same model is also illustrated in Figure 2.1.

$$y(\mathbf{x}, \mathbf{w}) = f(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})) \tag{2.1}$$

$$y_k(\mathbf{x}, \mathbf{w}) = f(\sum_{j=0}^{M} w^{(2)}_{jk} h(\sum_{i=0}^{D} w^{(1)}_{ij}(x_i)) \tag{2.2}$$

The Equation 2.2 can be derived by describing the basic NN model into a series of functional transformations as shown in Equation 2.3, where $j = 1, \ldots, M$ and with parameter $w_{j0}$ being called bias and $x_0 = 1$. Integrating the bias parameter inside the sum, results in the Equation 2.4.

$$a_j = \sum_{i=1}^{D} w^{(1)}_{ij} x_i + w^{(1)}_{0j} x_0 \tag{2.3}$$

$$a_j = \sum_{i=0}^{D} w^{(1)}_{ij} x_i \tag{2.4}$$

### 2.1.2 Activation Functions

The quantities $a_j$ of linear combinations illustrated in the function in Equation 2.3 are called activations. Activations $a_j$ are transformed by using a differentiable, non-linear function $h(\cdot)$, known as activation function, as shown in Equation 2.5.

$$z_j = h(a_j) \tag{2.5}$$

These layers are known as *hidden layers* as their output is not directly observable in any part of the process [17], [18]. The activation functions that can be used in a Neural Network model vary depending on the target problem and data type. Such examples can be Sigmoidal functions such as *tanh* as shown in Equation 2.6, or logistic sigmoid as can be seen in Equation 2.7, Rectified Linear Unit (ReLU), which is derived as shown in the Equation 2.8 [18]. In recent research regarding deeper convolutional networks, Parametric Rectified Linear Unit (PReLU) is used, defined as shown in Equation 2.9. PReLU learns a factor $a$ that controls the slope and can improve the performance of a model with negligible extra computation cost. The parameter $a$ only affects the negative part of the function. Manually defining the slope as a very small value, such as $\alpha = 0.001$, the Equation 2.10 defines the Leaky ReLU, an activation function that holds the properties of ReLU but with only one zero value, at $x = 0$. which helps with gradient computation. In contrast with PReLU, the coefficient $\alpha$ of the Leaky ReLU is prefixed and not a trainable parameter [19].

$$f_{\text{tanh}}(x) = \frac{e^x - e^{-x}1}{e^x + e^{-x}} \tag{2.6}$$

$$f_{\text{sigmoid}}(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

$$f_{\text{ReLU}}(x) = x^+ = max(0, x) \tag{2.8}$$

$$f_{\text{PReLU}}(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \end{cases} \tag{2.9}$$

$$f_{\text{Leaky ReLU}}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{2.10}$$

### 2.1.3 Deep Neural Networks

Higher complexity tasks cannot be mapped by a single series of activations, as discussed earlier. In Neural Networks, this can be achieved by using more series, known as layers, and more components in each one of the layers. This kind of network is known as Deep Neural Network (DNN). Equation 2.1 for calculating the NN output can now be rewritten for $L$ layers as shown in Equation 2.11. For clarification, the second set of dots denotes the closing parenthesis.

$$y_k(\mathbf{x}, \mathbf{w}) = f(\sum_{j=0}^{M} w_{jk}^{(L)} \dots g(\sum_{u=0}^{U} w_{uv}^{(2)} h(\sum_{i=0}^{D} w_{iu}^{(1)}(x_i))) \dots ) \tag{2.11}$$

DNNs can often use fewer components on each layer while being capable of better generalization. The drawback of the increase in the number of layers is the ease of computing the gradient and minimizing the loss function. Knowing the exact number of layers and/or the components needed is impossible, as there is no mathematical way to do so. The process of finding a suitable architecture for the task is by extensive experimentation and monitoring of the loss function [18], [20].

### 2.1.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a special type of NN able to handle grid-like topology, including grids in 1 dimension, 2, or even more. Such examples can be sampled time-series data for 1D, images in 2D, etc. As the name suggests, CNN employs convolutional layers that operate the convolutional function instead of a general matrix multiplication operation.

Assuming as an example a time-series input $\mathbf{x}$ convolved with a weight function $\mathbf{w}$, the function can be defined as shown in Equation 2.12. Assuming also that the two series $\mathbf{x}$ and $\mathbf{w}$ are discrete (sampled data) and defining the time $t$ and an age factor $a$, the Equation 2.12 can be computed as shown in Equation 2.13. The factor of age $a$ of samples is flipped compared with the samples of time $t$. An illustration of this process for 1D convolution can be seen in Figure 2.2.

$$s(t) = (\mathbf{x} * \mathbf{w})(t) \tag{2.12}$$

$$s(t) = \sum_{a=-\infty}^{\infty} \mathbf{x}(a)\mathbf{w}(t-a) \tag{2.13}$$

Generalizing the terminology, the series $\mathbf{x}$ can be described as the *input* to the layer and the weight series $\mathbf{w}$ as the convolution *kernel*. The process's output $\mathbf{s}$ is widely referred to as *feature map*.

CNNs, therefore, are called networks that contain at least one convolutional layer. CNNs are known for their characteristics: sparse interaction, parameter sharing and equivariant representation. The first property of *sparse interaction* is achieved as the kernel is significantly smaller than the input, meaning the interaction occurs only between close samples. This implies that the kernel can be used for small feature extraction, which can be information such as edges in picture pixels, timbre or other events across the timeline in audio samples, etc. This also gives better statistical efficiency as fewer parameters are used in the model, which means less memory and operations are needed. From this property, the second property of *parameter sharing* can be derived, as the same parameters are used across all operations. In contrast, conventional NNs use different parameters for the needed components in the layer. Parameter sharing is illustrated in Figure 2.2, as the same parameters $\{w_1, w_2, w_3, w_4\}$ are repeated across the input, and they are not updated. Again, this property allows the final property of *equivariance*, that changes in the input will also imply the exact change in the output. In this property, $f(g(\mathbf{x})) = g(f(\mathbf{x}))$ holds between functions $f$ and $g$, and these two functions are called equivariant to each other. For example, applying transformation and then convolution to $\mathbf{x}$ will be exactly the same as first applying convolution and then transforming $\mathbf{x}$.

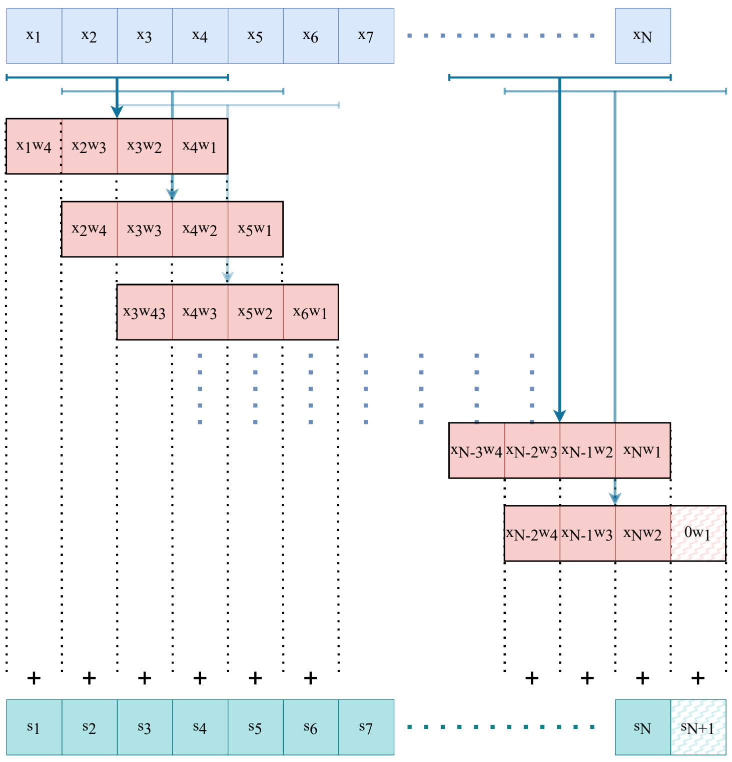


**Figure 2.2:** Convolution process with a 4-size kernel. Entry $s_{N+1}$ illustrates that the kernel may be extended over the edges. It always depends on the problem and the implementation of how the edges are treated.

Finally, it is worth mentioning that three stages take place in typical CNN. The first stage is applying the convolution to the input, which returns a set of feature maps. In the second stage, a non-linear function is applied to each of the feature maps, as discussed earlier in Subsection 2.1.2, which is also known as *detector stage*. Finally, in the third stage, a pooling function is used. In this stage, the output is replaced based on statistical criteria of nearby values. Examples include max pooling, which reports the maximum value within the range, average pooling, $L^2$ norm, etc. In this way, pooling changes the relation between input and output, as small input changes produce invariant output changes.

### 2.1.5 Recurrent Neural Networks

Recurrent Neural Networks (RNNs), as the name suggests, use memory to store older states. This property makes them ideal for use in time-series data, sequences, etc. Such examples are text prediction, handwritten recognition, or Automatic Speech Recognition (ASR).

#### Standard RNNs

There are various ways how an RNN can be implemented. Figure 2.3 illustrates two examples of recursive sections in RNN architecture. The left graph illustrates an example of recurrent connections between hidden units. As can be seen in the Figure, the recurrent connection is weighted with $\mathbf{W}$ matrix. Assuming an activation function of the node $h(\mathbf{a})$, the recurrent function $\mathbf{a}$ can be described as shown in the Equation 2.14, where $\mathbf{h}$ is the output of the $h(\mathbf{a})$ function, $\mathbf{x}$ is the input, $\mathbf{b}$ is the bias and with he superscript being the time step in the process.

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \tag{2.14}$$

Similarly, the right graph shows the recurrent connection between different nodes. Therefore, the output of another layer is used as the recurrent input to the recurrent layer, which mathematically can be described as shown in Equation 2.15.

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{k}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \tag{2.15}$$

#### Bidirectional RNNs

In cases such as ASR, samples from the past might not be sufficient to conclude information for the current sample. For example, when two or more phonemes are articulated together (coarticulation), the current phoneme might depend on the next few phonemes. In order to connect information from both the past and the future, two sub-RNNs are used, one for each direction. This kind of network is called Bidirectional RNN.

Figure 2.4 illustrates an example of a bidirectional RNN over time $(t-1)$, $(t)$ and $(t+1)$. As can be seen, layer $h$ propagates forward in time, illustrated with blue arrows, while layer $d$ does the opposite, illustrated with red arrows. In this

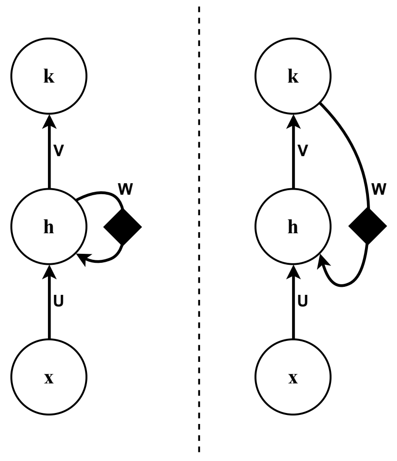


**Figure 2.3:** Two possible use cases of a Recurrent layer. The left graph shows a graph model of the recurrent layer $h$ taking previous values of itself. Matrix factors **U**, **W** and **V** are used to weight the outputs of the corresponding layers at time $(t)$, the output of layer $h$ at time $(t-1)$ and the output of layer $h$ at time $(t)$ respectively. The right graph shows a graph model when the output of another layer is used in the recurrent layer. This time, matrix factor **W** is used to weight the output of layer at time $(t-1)$.

way, the output unit $k$ can benefit from the summation of past and future information at each time step.

The information propagation concept can be extended to more dimensions as well. In the same way, as information is propagated forwards and backwards in 1d sequence, it propagated, for example, up, right, down and left in a 2d matrix, etc. In this way, the nodes capture mostly the local information from nearby nodes but still have a dependence on long-range information due to the overall propagation process. This computation is more expensive than convolution; however, it allows interaction between nodes that correlates features between them.

### LSTM and GRU layers

Optimizing an RNN network can be a difficult task due to the backpropagation process (discussed in Subsection 2.2.2). The gradients of the weights are calculated at each one of the steps. The problem of the gradients in RNN networks can be easily visualized by taking a look at Figures 2.3 and 2.4 in the opposite way of what the arrows indicate, which is how backpropagation takes place. This can possibly lead to one of the following cases;

- the Gradients shrink exponentially for small gradients of weights
- the Gradients grow exponentially for large gradients of weights

which is known as the vanishing/exploding gradient problem. A way to restrict

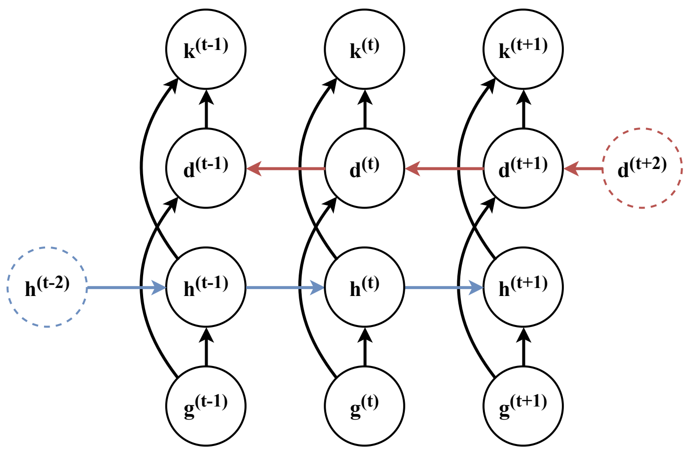


**Figure 2.4:** Illustration of bidirectional RNN over time samples $(t-1)$, $(t)$ and $(t+1)$. Blue arrows indicate forward-time propagation and red arrows indicate backward-time propagation to the past.

the problem is by applying some form of gating in the layers. Such examples are LSTM and GRU.

Long short-term memory (LSTM) utilizes gates (usually referred to as the input, output, and forget gates) to control the gradient propagation in the recurrent network's memory. These gates are considered neural network layers, and they learn when to forget, ignore, or keep the information.

Gated Recurrent Unit (GRU) holds similarities with LSTM in functionality. However, it has lower complexity and increased efficiency, which is achieved by deploying fewer parameters. For this reason, the choice of one over the other happens empirically or by conducting experiments to test their actual performance. It is implied that in case both have the same performance, GRU layers are deployed as they are more efficient.

In this Thesis, GRU layers are used in the RNN model. LSTM, however, is directly comparable to GRU and, for this reason, is also mentioned. The steps to compute GRU are given by Equation 2.16. Variable $\mathbf{x}$ is the input, $\mathbf{h}$ is the output, $\mathbf{z}$ and $\mathbf{r}$ are the update and reset parameters, and $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{b}$ the weights of the parameters and the bias. Symbol $\circ$ indicates the element-wise multiplication [20].

$$\begin{aligned}
\mathbf{z}_t &= f_{sigmoid}(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}) \\
\mathbf{r}_t &= f_{sigmoid}(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}) \\
\tilde{\mathbf{h}}_t &= f_{tanh}(\mathbf{W}_h\mathbf{x}_t + (\mathbf{U}_h\mathbf{h}_{t-1} \circ \mathbf{r}_t) + \mathbf{b}) \\
\mathbf{h}_t &= (1 - \mathbf{z}_t) \circ \tilde{\mathbf{h}}_t + \mathbf{z}_t \circ \mathbf{h}_{t-1}
\end{aligned} \tag{2.16}$$

## 2.2 Training

Model training is an iterative process in which the weights in the layers of the model are adapted accordingly. Weights are updated through the process of back-propagation, where, as the name suggests, the error flows backwards. The error is a scalar value and is calculated in each iteration. It is usually some form of mathematical distance between the target and the predicted values, and the way of computation depends on the defined loss function.

One problem with training DNN models is that they are prone to overfitting, as it cannot always be known in advance if an architecture can or cannot be employed for a task, and also usually a vast amount of parameters are used. Overfitting is referred to the relationship between training loss which is the prediction error of the data that are used in the training process, and validation loss which is the error of the data that are used for monitoring purposes. If there is no correlation between the training and validation losses is an indicator that the overfitting occurred. Ways of limiting this phenomenon include dropout layers which randomly drop a certain amount of weights by multiplying with either zero on one, early stopping with the training process stops according to certain criteria about the validation loss, etc [17], [18], [20].

Another way to increase the effectiveness of the training process is a variable learning rate. In this way, the learning rate of the gradient descent is decreased over time. This allows bigger changes in updating process of the weights at the early stages of training, where the parameters are far from local minimum points, and more precise updates when the weights are closer to local minimum points [18], [20].

### 2.2.1 Optimizers

Optimizer refers to the method that is used for minimization of the loss function with respect to the weight parameters. Supposing $E$ is the error defined by the loss function between target and predicted values. The weights in each iteration are updated from $\mathbf{w}$ to $\mathbf{w} + \delta\mathbf{w}$, based on the error which is a function of $\delta E \approx \delta\mathbf{w}^T \nabla E(\mathbf{w})$. Vector $\nabla E(\mathbf{w})$ shows the direction of the error increase, and therefore $-\nabla E(\mathbf{w})$ is the direction that will result in better performance given the weights. The general form of the weight update process can be seen in Equation 2.17. Neural Networks take advantage of some form of the gradient-based optimization process, such as the Gradient Descent, to choose the updated weights. Equation 2.17 can be therefore rewritten to point out to the negative gradient direction as illustrated in Equation 2.18.

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \Delta\mathbf{w}^{\tau} \tag{2.17}$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta\nabla E(\mathbf{w}^{\tau}) \tag{2.18}$$

After the update, the weights are re-evaluated, and the process is repeated. Parameter $\eta > 0$ is called the *learning rate,* and scales the step to the next step of update. In other words, it determines how fast or slow the weights $\mathbf{w}$ change [17]. Optimally one should choose neither a large nor very small value. In the former case, the optimizer might skip the desired minimum, while in the latter case, the optimizer might never reach it.

### ADAM Optimizer

Adaptive Moment Estimation is a widely used optimizer as it is more efficient and less memory demanding. It computes individual adaptive learning rates from the estimates of the first and the second moments of the error function gradient. Additionally, it exponentially decays the average of the past of previous squared gradients while it also incorporates an average of past gradients $m_t$ [21].

The steps for computing the ADAM parameters are illustrated in Equation 2.19. First, values $\alpha$, $\beta_1$ and $\beta_2$ are required, which are the stepsize and the exponential decay rates, respectively. The exponential decay rates take values from the subset of $[0, 1)$. The stochastic objective function $f(\theta)$ of the parameter $\theta$ is also required. The 1st-moment vector $m_0$, the 2nd-moment vector $v_0$, and the timestep moment are all initialized to zero. The bias-corrected second moment $\hat{u}_t$ is then computed, and the parameters $\theta_t$ are updated. The whole process is iterative until $\theta_t$ converges, where the value of $\theta_t$ is returned by the algorithm [21].

$$
\begin{aligned}
t &:= t + 1 \\
g_t &:= \nabla_\theta f_t(\theta_{t-1}) \\
m_t &:= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
u_t &:= \beta_2 u_t + (1 - \beta_2) g_t^2 \\
\hat{m}_t &:= \frac{m_t}{1 - \beta_1^t} \\
\hat{u}_t &:= \frac{u_t}{1 - \beta_2^t} \\
\theta_t &:= \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{u}_t} + \epsilon}
\end{aligned}
\tag{2.19}
$$

## 2.2.2 Backpropagation

Backpropagation is used in the context of neural networks as it is an efficient way of numerically calculating the gradient of the loss function $E(w)$ with respect to weights. The computation takes place in two distinct parts. In the first part, the derivatives are calculated through the backpropagation process, and then in the second part, the derivatives are used in order to calculate the updated weights.

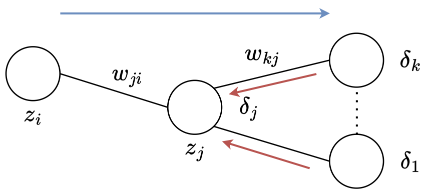


**Figure 2.5:** Information flow in Neural Networks. Forward propagation is illustrated with blue arrows, and backward propagation is illustrated with red.

Derivatives of the loss function are calculated as shown in 2.20. As shown earlier in Equation 2.3, each unit computes a weighted sum of its inputs. This is illustrated in a more general form in Equation 2.21 where $z_i$ is the non-linear transformation function of the unit $i$, as shown in Equation 2.5.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{2.20}$$

$$a_j = \sum_i w_{ij} z_i \tag{2.21}$$

Letting the partial derivatives of error $E$ with respect to the activation $a_j$, as shown in Equation 2.22, simplifies the notation, and therefore the relation shown in 2.20 can be rewritten as shown in Equation 2.23.

$$\delta_j := \frac{\partial E}{\partial a_j} \tag{2.22}$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j z_i \tag{2.23}$$

Additionally, it is possible to compute derivatives of functions by composing other functions whose derivatives are known. This is called the chain rule of calculus, and its form is defined as shown in Equation 2.24. As can be seen, the left part has already been defined as $\delta_j$ in Equation 2.22.

$$\frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{2.24}$$

Combining the definition of the chain rule with the definition of $\delta_j$ in Equation 2.22 and Equations 2.21, 2.5 and 2.23, it can be seen that the derivatives can be calculated as shown in the Equation 2.25.

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{2.25}$$

Function $h'(\cdot)$ is the derivative of the activation function. As an example, in the case of ReLU defined in Equation 2.8, the function $h'_{\text{ReLU}}(\cdot)$ is 1 for $x > 0$, 0 for $x < 0$ and undefined in $x = 0$.

Therefore, it can be concluded that the partial derivative $\delta$ values, which determine the step for updating the weights, can be derived by propagating the gradient higher in network layers, as shown in Figure 2.5. Values of $\delta_k$ (output) are easily derived as shown in Equation 2.26, where $y_k$ and $t_k$ are the predicted and target values, respectively. Taking advantage of the backpropagation properties allows calculating the rest of $\delta$ values in all hidden layers [17], [18].

$$\delta_k = y_k - t_k \tag{2.26}$$

## 2.3 Automatic Speech Recognition

As the name suggests, Automatic Speech Recognition (ASR) is a process where an application can recognise the speaker's speech, usually converting it to text. This process has been achieved with many implementations in the past, mainly by machine learning approaches such as Hidden Markov Models (HMM), which can model the state of transitions of speech representation, such as spectrum transitions. The first attempts to introduce neural networks to the ASR problem were by still relying on HMM models for sequence modelling with hybrid HMM/DNN models. Many modern end-to-end DNN systems have been proposed where the whole process takes place in a single Neural Network model [20].

### 2.3.1 Spectrogram Calculation

The spectrographic analysis $S$ of the audio signal is often used in Automatic Speech Recognition applications, as it offers an extra dimension of information compared to time-domain signals. A window analysis can be applied to the signal for acquiring quasi-stationary segments allowing the study of the temporal characteristics that do not drastically change over the steps. Depending on the application, one can favour different aspects of the spectrogram by adjusting the window length of the process. As an example, *Hamming* window is widely used in speech signal analysis and is given as shown in Equation 2.27. $L$ is the window size in time-domain samples [2].

$$w(n) = \begin{cases} 0.54 - 0.4\cos(\frac{2\pi n}{L-1}) & 0 \leq n \leq L-1 \\ 0 & \text{Otherwise} \end{cases} \tag{2.27}$$

Spectrograms are a visual way to illustrate the differences in the signal magnitude over the frequency bands of the spectrum. A widely used tool to obtain time-varying spectral and temporal characteristics is by applying Short-Time Fourier

Transform (STFT) to the signal. STFT is applied in a time analysis window $w$, usually 10 to 30 ms, in which the properties of the signal do not drastically change. STFT is given as shown in the Equation 2.28, with $x(m)$ being the input samples [2], [22].

$$X(n,\omega) = \sum_{m=-\infty}^{\infty} x(m)w(m-n)e^{-j\omega m} \tag{2.28}$$

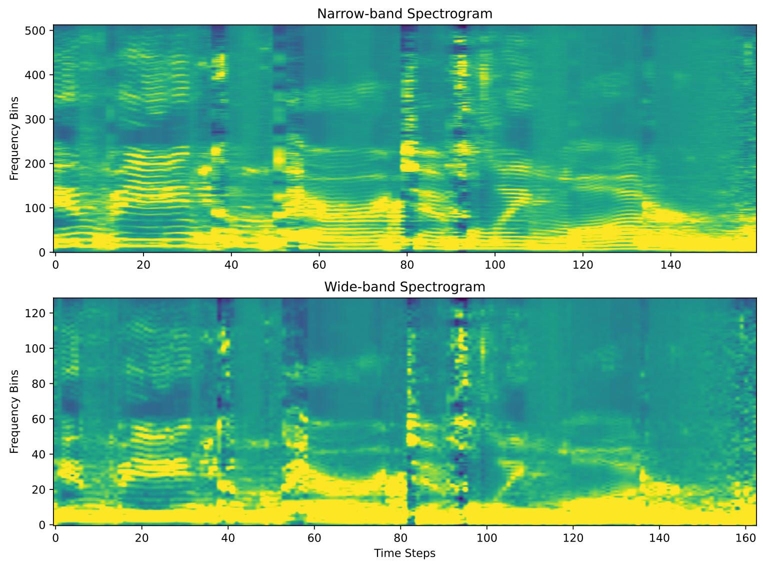


**Figure 2.6:** Comparison between Narrow and wide band spectrograms of the phrase "in being comparatively modern". Smearing can be noticed on the Y-axis in the narrow band and on the X-axis in the wide band. In this example, the FFT size is adjusted according to the window size, resulting in different frequency bins.

Applying a long-duration window to the signal produce a narrow-band spectrogram. Narrow-band spectrograms have pronounced detail in frequency resolution; they lack, however, time resolution. As a result, individual harmonics of the signal can be easily resolved, but short-duration segments are temporally smeared. On the other hand, a short-duration window produces a wide-band spectrogram which shows very good temporal resolution; however, it has a drawback in the frequency resolution representation. Smearing now happens in the frequency axis. However, the clear resolution in the time axis makes it suitable for applications such as language analysis. These differences between narrow-band and wide-band spectrograms can be seen in Figure 2.6. A method that is often used in DSP applications is to apply overlapping between the windows. Many of the window functions, such as Hanning, Hamming, etc., use fading in the edges to prevent *spectral leakage* as they spread across the whole frequency spectrum and not only in the center frequency. As a rule of thumb, 50% of overlapping and added-together windows are often used to overcome the fading. The size window of the window is equal to the wide-band seen in Figure 2.6. The resulting spectrogram can be seen in Figure 2.7 [2].

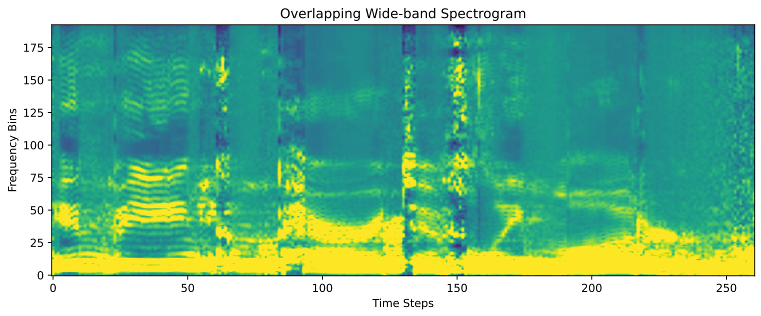


**Figure 2.7:** Spectrogram of the same audio sample as 2.6, however, overlapping is used between the sample windows, which also alters the number of time steps on the signal.

### 2.3.2 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) is an algorithm able to align sample inputs in the form of $\mathbf{x} = [x_1, x_2, \cdots, x_N]$ with outputs in the form of $\mathbf{y} = [y_1, y_2, \cdots, y_M]$, where the ratio $N/M$ is neither known nor constant [23]. For this reason, it is widely prevalent among ASR models, as the same text output can result from speakers with variations in speaking speed, pronunciation, etc.

The CTC algorithm in ASR models is used for decoding a string of $N$ samples into a string of $M$ characters. In the case of using spectrograms as inputs in the ASR model, $N$ depends on the frame hop size of the STFT. Probabilities are assigned for each possible character to each of the $N$ sampled windows by the ASR model. Either the most probable character or the most probable beam is selected during the CTC decoding. Finally, duplicate entries are discarded based on specific criteria, and the decoding process results in a string of $M$ characters, where $N > M$. The mentioned types of search and the criteria for discarding duplicate entries are further explained in Appendix A.

# Chapter 3

# Speech Enhancement with DNN

This section discusses in more detail the processes that are commonly used for designing and evaluating a DNN model for speech enhancement. First, the objective intelligibility and quality metrics are explained. Then, autoencoder models are explained for use in speech enhancement tasks. Finally, it is discussed why using metrics as loss functions improves the overall performance of the speech enhancement models.

## 3.1 Speech Intelligibility and Quality Metrics

Two kinds of test methods can be conducted to evaluate how well a speech enhancement system works. The subjective listening evaluation test is the more reliable one. This requires a group of listeners who assess the differences between the original (speech + noise) and the processed (enhanced) signals. The drawback of this method is how time-consuming it is. On the other hand, objective metrics mathematically evaluate the performance of the processed (enhanced) signal. For the objective metrics to be valid, there should be a high correlation between the objective and subjective tests [2]. In this Thesis, only selective objective methods will be discussed.

### 3.1.1 Perceptual Evaluation of Speech Quality (PESQ)

This metric was initially developed as an ITU-T standard in order to evaluate the speech signal in a wide range of applications, such as under network conditions, analogue connections, codecs, packet loss, variable delay, etc. The method requires the degraded or enhanced speech signal and the reference one which is a signal without distorting at all. First, these two signals are aligned to a standard listening level and are filtered, simulating a telephone transmission. In this stage, the Bark spectrum is estimated, which is based on an FFT analysis using 32 ms Hamming windows with 50% overlap and then summing up the band powers. The bands are spaced according to the Bark scale. The two signals are then aligned in time to correct delays and equalized. The latter is compensation for the filtering effect with a ratio of the average degraded Bark spectrum to the original Bark spectrum. Auditory transform is applied to obtain the loudness spectra. This is computed as shown in 3.1 where $b$ is the Bark spectrum, in a total of 42 Bark bands, $S_i$ is the loudness scaling factor, $P_0(b)$ is the absolute hearing threshold of the Bark spectrum, $B'_x(b)$ is the frequency compensated Bark Spectrum and $\gamma = 0.23$ for $b \geq 4$ or slightly higher otherwise [2], [24].

$$S(b) = S_i \left(\frac{P_0(b)}{0.5}\right)^{\gamma} \left[\left(0.5 + 0.5\frac{B'_x(b)}{P_0(b)}\right)^{\gamma} - 1\right] \tag{3.1}$$

The signed difference between the original loudness spectra $S(b)$ and the degraded $\bar{S}(b)$ which is also computed as shown in Equation 3.1, is denoted as $r_n(b) = S_n(b) - \bar{S}_n(b)$, where subscript $n$ indicates time, such as frames. A mask $m(b)_n = 0.25\text{min}\{S_n(b), \bar{S}_n(b)\}$ is defined, and based on the numerical comparison of $r_n(b)$ with $m_n(b)$, the disturbance density factor $D_n(b)$ takes the values $r_n(b) - m(b)_n$, 0 or $r_n(b) + m(b)_n$. The disturbance density is multiplied by an asymmetry factor to determine the Asymmetry Disturbance Density $DA_n(b)$.

Disturbance Density and Asymmetry Disturbance Density are integrated using norms, as shown in Equation 3.2. $D_n$ and $DA_n$ are called frame disturbance, and $W_b$ are weights proportional to the width of the Bark bands.

$$\begin{aligned} D_n &= \left(\sum_{b=1}^{N_b} W_b\right)^{1/2} \left(\sum_{b=1}^{N_b} [|D_n(b)|W_b]^2\right)^{1/2} \\ DA_n &= \sum_{b=1}^{N_b} |DA_n(b)|W_b \end{aligned} \tag{3.2}$$

The frame disturbance is then checked to be between a certain threshold. If the frame is not below the threshold, it is considered a bad frame and is recomputed with different factors. The process is iterative until the frame fulfils the criterion. Finally, the PESQ value is calculated as shown in Equation 3.3. Values $d_{sym}$ and $d_{asym}$ are summations of frame disturbance and asymmetric frame disturbance over $n = \{1, 2 \ldots 20\}$ frames, which overall spans 320 ms. PESQ score is -0.5 to 4.5 [2].

$$\text{PESQ} = 4.5 - 0.1 d_{sym} - 0.039 d_{asym} \tag{3.3}$$

### 3.1.2 Signal-to-Distortion Ratio (SDR)

Source-to-Distortion Ratio (SDR) is a widely used metric for measuring the level ratios of speech signals with their distortion. The enhanced speech signal $\hat{s}(t)$ can be described as 3.4, where $s_{target}(t)$ is a deformation of the target source, $e_{inter}(t)$ is a deformation of source interfered by noise, $e_{noise}(t)$ is the noise that has been interfered to the source, $e_{artif}(t)$ artifacts that are produced by the algorithm. The SDR is given as illustrated in Equation 3.5, where $s_{dist}$ is the distorted component filtered from the enhanced signal [25].

$$\hat{s}(t) = s_{target}(t) + e_{inter}(t) + e_{noise}(t) + e_{artif}(t) \tag{3.4}$$

$$\text{SDR} = 10 \log_{10} \frac{\|\mathbf{s}_{dist}\|^2}{\|\mathbf{e}_{inter} + \mathbf{e}_{noise} + \mathbf{e}_{artif}\|^2} \tag{3.5}$$

### 3.1.3 Short-Time Objective Intelligibility (STOI)

Short-Time Objective Intelligibility (STOI) is an objective metric that shows a high correlation with listening tests while managing to be simple in structure and transparent. It is worth mentioning that this metric is designed for signals with a sample rate of 10kHz. For this subsection, the clean signal is denoted as $x$ and the processed one as $y$. First, time-frequency representation is obtained, as discussed in Subsection 2.3.1, for both target and processed signals, and 50% overlap is used. Each one of the $k$ DFT-bins for the $m$ frames is denoted as the norm of the $j^{th}$ one-third of the octave band, as illustrated in Equation 3.6, where $k_1$ and $k_2$ denote the edge of the one-third octave band. Signal $y$ gets a similar representation.

$$X_j(m) = \sqrt{\sum_{k=k_1(j)}^{k_2(j)-1} |x(k,m)|^2} \tag{3.6}$$

A region of consecutive $N$ values of TF-units $x_j(n)$ and $y_j(n)$ is used to calculate the intelligibility measure. Each region, denoted as $d_j(m)$, belongs to a set of $M = \{(m-N+1),(m-N+2),...,m-1,m\}$. The $y_j(m)$ units are scaled according to a normalization factor $a$ to equalize the energy compared to $x_j(m)$ inside the $m$ unit. The factor is calculated as shown in Equation 3.7. Equation 3.8 shows the normalized and clipped TF-units, where $\beta$ is the lower SDR bound.

$$a = \sqrt{\left(\frac{\sum_n X_j(n)^2}{\sum_n Y_j(n)^2}\right)} \tag{3.7}$$

$$\mathbf{y}' = \max\ \{\min\{a\mathbf{Y}, \mathbf{X} + 10^{-\beta/20}\mathbf{X}\}, \mathbf{X} - 10^{-\beta/20}\mathbf{X}\} \tag{3.8}$$

Finally, each region is calculated as shown in Equation 3.9, where $l = 1, \dots, M$, and then summed over all $j$ one-third octave bands as illustrated in Equation 3.10 to produce the final metric [26].

$$d_j(m) = \frac{\sum_n \left(X_j(n) - \frac{1}{N}\sum_l X_j(l)\right)\left(Y_j'(n) - \frac{1}{N}\sum_l Y_j'(l)\right)}{\sqrt{\sum_n \left(X_j(n) - \frac{1}{N}\sum_l X_j(l)\right)^2 \sum_n \left(Y_j'(n) - \frac{1}{N}\sum_l Y_j'(l)\right)^2}} \tag{3.9}$$

$$d = \frac{1}{JM}\sum_{j,m} d_j(m) \tag{3.10}$$

### 3.1.4 Extended STOI (ESTOI)

Extended Short-Time Objective Intelligibility (ESTOI) is a similar approach to STOI, with the difference that in the case of ESTOI, it is not assumed independence between frequency bands. ESTOI is designed to work well with highly modulated noise sources, while time-modulated noise maskers are better captured. Carrying again $x$ and $y$ as clean and processed signals, respectively, the signals are normalized as shown in Equation 3.11, where $x_{j,m} = [X_j(m-N+1), X_j(m-N+2), \dots, X_j(m-1), X_j(m)]$ and $X_j(m)$ are the STFT coefficient energies calculated as shown in STOI subsection, in Equation 3.6. The mean of $x_{j,m}$ is calculated as shown in Equation 3.12.

$$\overline{x}_{j,m} = \frac{1}{\|(x_{j,m} - \mu_{x_{j,m}}\underline{\mathbf{1}})\|}(x_{j,m} - \mu_{x_{j,m}}\underline{\mathbf{1}}) \tag{3.11}$$

$$\mu_{x_{j,m}} = \frac{1}{N}\sum_{m'=0}^{N-1} S_j(m-m') \tag{3.12}$$

Defining the row-normalized vector $\overline{X}_m = [\overline{S}_{1,m}^T, \dots, \overline{S}_{J,m}^T]$, the process of mean and variance normalization of matrix $\overline{X}_m$ is repeated, resulting to $\check{X}_m = [\check{S}_{1,m}^T, \dots, \check{S}_{J,m}^T]$. The intermediate intelligibility index is calculated as shown in Equation 3.13 depending on the time index $m$. Then, the temporal average of all indices is calculated, as illustrated in 3.14 [7].

$$d_m = \frac{1}{N}\sum_{n=1}^{N} \check{x}_{n,m}^T \check{y}_{n,m} \tag{3.13}$$

$$d = \frac{1}{M}\sum_{m=1}^{M} d_m \tag{3.14}$$

### 3.1.5 Scale-Invariant Signal-to-Distortion Ratio (SI-SDR)

Scale-Invariant Signal-to-Distortion Ratio (SI-SDR) is an objective metric in the time domain proposed as an improved variation of SDR discussed in Subsection 3.1.2. In this case, the mixture is considered as $\mathbf{x} = \mathbf{s} + \mathbf{n}$, where $\mathbf{s}$ is the target signal

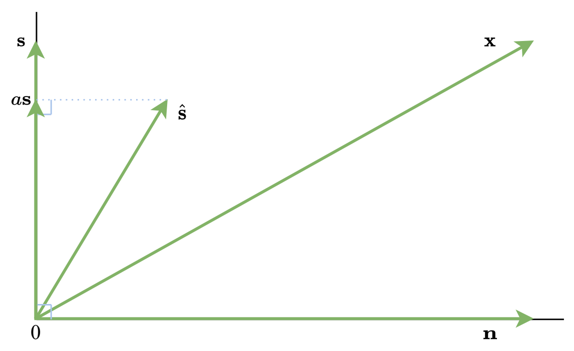


**Figure 3.1:** Signal model of SI-SDR.

and **n** the noise. This can be visually seen in Figure 3.1. The processed/enhanced signal is denoted as $\hat{\mathbf{s}}$. SI-SDR ensures that the residual $\mathbf{s} - \hat{\mathbf{s}}$ is orthogonal to the target $\mathbf{s}$. This is achieved by scaling the target signal by factor $a$, as seen in Figure 3.1, and allows better properties regarding SNR. The SI-SDR metric is calculated as illustrated in Equation 3.15 where the target signal is scaled with optimal $a = \hat{\mathbf{s}}^T\mathbf{s}/||\mathbf{s}||^2$. [8].

$$\text{SI-SDR} = 10 \log_{10} = \left( \frac{|| \frac{\hat{\mathbf{s}}^T\mathbf{s}}{||\mathbf{s}||^2}\mathbf{s}||^2}{|| \frac{\hat{\mathbf{s}}^T\mathbf{s}}{||\mathbf{s}||^2}\mathbf{s} - \hat{\mathbf{s}}||^2} \right) \tag{3.15}$$

## 3.2 DNN Architectures For Speech Enhancement

A DNN architecture that performs well in Speech Enhancement tasks and has been employed in many pieces of research in recent years is the Encoder-Decoder Fully-Convolutional Neural Network [3], [12], [13], [15]. Fully-Convolutional Neural Networks (FCNN), as the name suggests, are Neural Networks that consist only of convolutional or deconvolutional layers.

Autoencoder is a special type of DNN model trained to copy the input to the output. Autoencoders, however, are designed not perfectly to copy the input but rather to alternate it based on certain criteria. As the name suggests, autoencoders consist of two parts, the encoder part $\mathbf{h} = f(\mathbf{x})$ and the decoder part $\mathbf{r} = g(\mathbf{h})$ part. The model learns how to prioritize aspects of the input, which leads to learning the useful properties of the given signal. Allowing deeper information processing using an Encoder-Decoder scheme network results in deconstruction and then reconstruction of the data giving them specific properties. Autoencoders for enhancing/denoising minimize the loss function of $\mathcal{L}(\mathbf{x}, g(f(\tilde{\mathbf{x}})))$, where $\tilde{\mathbf{x}}$ is a degraded copy of $\mathbf{x}$ [18].

For conventional speech-enhancing operations, however, it is common for the implemented models to use Time-Frequency (TF) masking or spectral mapping as the input signal. The benefit of the Fully convolution architecture is that these models accept input signals in the time domain and return output also in the time domain. Studies showed that frequency-domain-based loss functions play a greater role in the performance of the model, and in this way, the model is able to learn what adjustments in the frequency domain of the loss affect the time domain signal. This integration is helpful for the core tasks of the speech enhancement process. In the autoencoders form, convolution layers are used in the encoder part, which decreases the size of the signal, and deconvolution layers are used in the decoder part, which increases the size of the signal [12], [13].

## 3.3 Loss Function

A common method that was initially applied to DNNs models for speech enhancement was MSE and MAE of the time-domain waveform, including during the development of the Encoder-Decoder FCNN mentioned in the previous section. These methods, however, show a low correlation with how the human ear perceives the quality and the intelligibility of speech [3], [12], [13], [15]. As the objective metrics are designed to provide mathematical information on whether or not the speech signal is pleasant or if its speech content is easily understandable, they can be adapted to be used to minimize an error function based on the metric and maximise the performance of the given model in this metric. In fact, it has been proven that using metrics as a loss function shows improvements in speech-enhancing performance, according to test results in the objective metrics [3], [15].

# Chapter 4

# Implementation

This chapter discusses in detail all the processes that took place during this Thesis project.

## 4.1 Dataset Preperation

As data are the driving factor of the DNN models, it is of the utmost importance the data be sufficient and well-prepared. This means that the dataset has to be properly cleaned and organized. Additionally, pre-processing of the dataset is required in some cases. As an example, for the Speech Enhancement experiment, the signals of speech passages are clean, and therefore, noise has to be applied using a specific range of SNR, as described in detail in the Noise Signal section of 4.1.2.

### 4.1.1 Datasets Characteristics

Multiple datasets have been used in this thesis project, which is due to the fact that different kinds of data are required for different tasks. LJSpeech contains clean speech signals with their corresponding prompt text and is used for training purposes. The initial plan was to use the WSJ0 dataset for training the Speech Enhancement Model to keep the methodology constant across similar research proposals [3], [15] and the LJSpeech dataset for only training the model of ASR which is used as a loss function later on. These two datasets however are sampled at different rates, and for simplicity reasons, only the LJSpeech has been used for both models. Dataset SLR83 from OpenSLR is used complementarily for testing the trained model alongside the test subset of the LJSpeech. Finally, complementary datasets are used containing the noise that is added to the speech datasets.

### Noise Signals

For this dataset, 7 noise types are used, synthetically generated stationary speech-shaped noise (SSN), synthetically generated non-stationary babble (NSB), café, street, pedestrian street, bus [15] and construction building [27]. The café, street, pedestrian street and bus noises come from the CHiME-3 dataset [28]. SSN is Gaussian white noise filtered by a twelfth-order all-pole filter. The filter coefficients have been obtained by linear predictive coding analysis of the concatenation of 100 utterances that have been randomly picked from the TIMIT dataset [29]. NSB is a mixture of utterances from 6 different speakers, both men and women. The speakers have been randomly chosen from the TIMIT dataset. Construction building noise has been added due to the fact that all the other provided noise types are in lower sample rate of 16kHz, that in theory, could affect the performance of the model as there is no frequency information above approximately 8kHz. Construction building was recorded in Aalborg University Hospital in 2021 during to its construction state at a higher sample rate, and the purpose of this type of noise is to show whether or not the absence of frequency information in the missing area has indeed an effect on the metrics.

Pedestrian, bus and construction building types of noise are not used in the training process, so some types of noise remain unseen in the test process. All the files sampled at 16kHz are upsampled to the desired 22,050 Hz that the model is trained with.

### LJSpeech

The LJSpeech dataset is part of the LibriVox project. It consists of speech audio clips of a female speaker reading text utterances. The average length of these passages is around 10 seconds, and the recordings are sampled with a sample rate of 22,050 samples/sec. [30]. The signals of this dataset are noise-free. This dataset is further split into Training, Validation and Test subsets. Training and Validation samples are segmented into 131,072 samples, which is approximately 6 seconds. This is enough time for full spoken sentences. In case the utterance is less than 6 seconds, the samples are zero padded.

### SLR83

This dataset contains transcribed audio of English sentences and has been recorded by volunteers from different parts of the UK [31]. The audio files have a sample rate of 44,100 samples/sec., which makes it ideal, as it allows downsampling to 22050Hz to be consistent with the operational sampling rate of the model.

From the SLR83 dataset, only a selected portion is used, as shown below;

- Midlands English female : 246 audio files

- Midlands English male : 450 audio files
- Northern English female : 750 audio files
- Northern English male : 2,097 audio files

This dataset is used as supplementary in order to further test the performance of the speech enhancement model in the objective metrics.

### 4.1.2 Data Processing

Since the aim of this Thesis is speech enhancement, noise should be introduced to the speech passages. Assuming $\mathbf{x}_{clean} \in \mathbb{R}^L$ is the clean speech signal vector with time-domain samples, a segment of scaled noise signal $\mathbf{n} \in \mathbb{R}^L$ with the same dimension is selected randomly from a noise sample and added to the clean signal, resulting in a noisy speech signal, as shown in the Equation 4.1.

$$\mathbf{x}_{noisy} = \mathbf{x}_{clean} + \mathbf{n} \tag{4.1}$$

The Signal-to-Noise Ratio (SNR) is drawn from a random uniform distribution, as $w \sim \mathcal{U}(-10, 10)$ dB. The noise is added according to ITU-T standards regarding the proper filtering before measuring the power of the signal, and therefore properly defining the SNR by scaling the noise [25], [32]. The SNR value for signal $\mathbf{x}$ and scaled noise $\mathbf{n}$ is defined as shown in 4.2. Therefore, for each one of the SNR values of $\mathbf{w}$, the noise is scaled accordingly.

$$SNR = 10 \log_{10} \left( \frac{\|\mathbf{x}\|^2}{\|\mathbf{n}\|^2} \right) \tag{4.2}$$

## 4.2 ASR model with CTC loss function

The development of a proper Automatic Speech Recognition (ASR) model is essential for the later design of the loss function. A suggested architecture by the *TensorFlow* presented in one of its tutorials is used due to the time limitation of the Thesis project [33]. The training of the model takes place using the LJSpeech Dataset and the dictionary is formed by the letters and special characters (including space characters) available in the prompt texts as shown below.

abcdefghijklmnopqrstuvwxyz'?!*space*

### 4.2.1 Spectrogram Calculation

The ASR model uses images of spectrograms to extract the features. The process of spectrogram calculation is shown in the flow graph in Figure 4.1. As can be seen, the spectrum **S** is first calculated using a short-time Fourier transform (STFT). The absolute value and the square root $\hat{\mathbf{S}}$ in the frequency spectrum are then calculated. Finally, the spectrogram $\hat{\hat{\mathbf{S}}}$ is batch normalized. Assuming $L$ length size of a time-domain signal $\mathbf{x}$, the mean value $\mu_{\hat{\mathbf{S}}}$ and the standard deviation value $\sigma_{\hat{\mathbf{S}}}$ are first calculated, and then the normalized spectrogram is calculated as shown in the Equation 4.3. The exponent value in the denominator stands for avoiding divisions by zero.

$$\hat{\hat{\mathbf{S}}} = \frac{\hat{\mathbf{S}} - \mu_{\hat{\mathbf{S}}}}{\sigma_{\hat{\mathbf{S}}} + 10^{-10}} \tag{4.3}$$

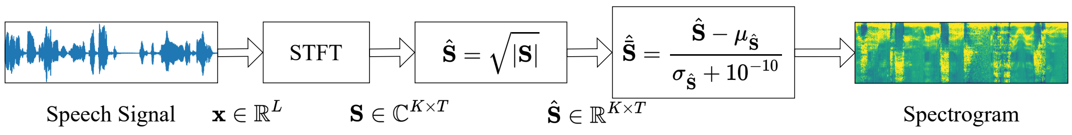

STFT
$\hat{\mathbf{S}} = \sqrt{|\mathbf{S}|}$
$\hat{\hat{\mathbf{S}}} = \frac{\hat{\mathbf{S}} - \mu_{\hat{\mathbf{S}}}}{\sigma_{\hat{\mathbf{S}}} + 10^{-10}}$
Speech Signal
$\mathbf{x} \in \mathbb{R}^L$
$\mathbf{S} \in \mathbb{C}^{K \times T}$
$\hat{\mathbf{S}} \in \mathbb{R}^{K \times T}$
Spectrogram

**Figure 4.1:** Flow graph of spectrogram calculation of a time-domain signal $\mathbf{X}$

### 4.2.2 Label Encoding and Decoding processes

Since the Neural Networks are not good at working with letters, two processes take place before and after the estimation process of the model. First, during the Encoding, the letters are converted into lowercase ones. A lookup function occurs afterwards, where the letters are mapped into integers.

The model outputs a tensor of probabilities in the shape of *(samples, time_steps, num_categories)* which is used as input to the CTC loss function. In this case, the number of categories is the number of characters in the dictionary. The time step depends on the length of the audio file in samples and on the hop size of the STFT. The output tensor is then decoded. In the decoding process, greedy search is used, which means that simply the most prominent integer entry for each step is selected and then decoded. The decoding of the string is described in more detail in Appendix A. Finally, the selected path is mapped back to the characters.

### 4.2.3 ASR Model Structure

The architecture of the ASR model is proposed by *TensorFlow* and is based on the DeepSpeech2 implementation [33], [34]. The model is implemented as shown in the Figure 4.2 and consists of 20 layers (not counting the reshape). The first 2 are 2D convolutional layers with 32 channels, [11, 41] and [11, 21] kernel size and with stride being [2, 2] and [1, 2] respectively. Each one of the 5 GRU layers in

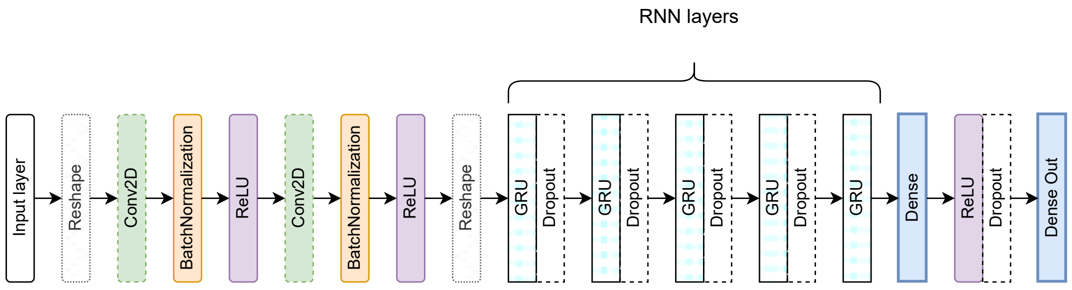


**Figure 4.2:** Diagram of the ASR model employed in this work.

this implementation has a size of 512 components, as suggested by TensorFlow. A fully-connected (dense) layer follows which has double the amount of components compared with the GRU layers. Finally, a projection layer is used (dense out), which has the size of the dictionary.

### 4.2.4 Data preparation

The preparation of the dataset follows the instructions of the Keras model example on their website [33]. The LJSpeech dataset contains the audio snippet as well as the script transcriptions and normalized transcription, where numbers and symbols have been expanded into full words (UTF-8) [30]. Normalized transcriptions are used in the training process. As discussed earlier, the spectrograms are extracted from the audio snippets and the transcripts are mapped from characters to numbers.

Further on, the dataset is split into two parts, namely, the training and validation. In order to favour the performance of the whole implementation, TensorFlow dataset objects are used for both of these datasets. A function that loads and processes the data is mapped on each dataset object, and the tuples of the spectrograms and their corresponding transcripts are yielded back in order to be used in the training process.

### 4.2.5 Training

The ASR model is fitted with the aforementioned training and validation sets. The processing takes place in batches of 32 samples. As a rule of thumb, the higher the number of samples per batch, the better, as it more accurately approaches the mean of the data.

After the forward propagation of the data through the model layers, a posterior probability matrix is formed which contains the probabilities of each one of the characters for all time steps. As a reminder, and since the model is optimized for batches and not single samples, the output of the model has the form of *(samples, time_steps, num_categories)*. The CTC loss is calculated based on this matrix for each one of the samples of the batch. The loss is finally backpropagated for the next mini-batch.

The loss function is minimized using the ADAM optimizer with a learning rate of 0.0001. Additionally, early stopping with patience 5 and learning rate reduction with a factor of 0.5 and patience 2 are deployed during the training process. The patience factor denotes how many epochs the optimizer can wait without seeing a decrease in the validation loss.

Finally, an additional callback function takes place at the end of each epoch, where the validation data are used to display the model performance, as well as to calculate the Word Error Rate (WER) and Character Error Rate (CER) performance of the model, by comparing the predicted and true transcript strings of the given sample. The displayed samples and their corresponding predictions are selected randomly. In order to acquire the transcript strings from posterior probability matrices, a CTC decoder is used. The decoder uses greedy search, similar to the training process.

## 4.3 Speech Enhancement with ASR Loss

This section describes the process that took place for the development of the Speech Enhancement model using the ASR model as a loss function. The model is identical to the ones deployed in the research papers that this Thesis took inspiration from [3], [15], since it is proven that it performs well in speech enhancement. Additionally, adopting the same model allows direct comparison between previously studied methods of loss functions with the methods studied in this Thesis.

### 4.3.1 Implementation Description

The general idea of the Speech Enhancement approach can be seen in Figure 4.3. Two different approaches have been considered for implementing CTC loss function. The first is minimizing the Kullback–Leibler distance (KLD) between the clean and estimated posterior probability matrices, and the second is by learning to decode the posterior probability matrix of the noisy one using CTC based on ASR estimations of the clean signals.

The dataset contains two variations of the same speech signal, a clean signal and a mixture of clean signal with noise, which is added as explained in the Subsection 4.1.2. The clean signal is encoded (spectrograms) and then is used as input in the ASR model. At this stage, the ASR model is pre-trained, and its parameters are frozen, meaning that no further optimization or weight changes take place during the training of the Speech Enhancement model.

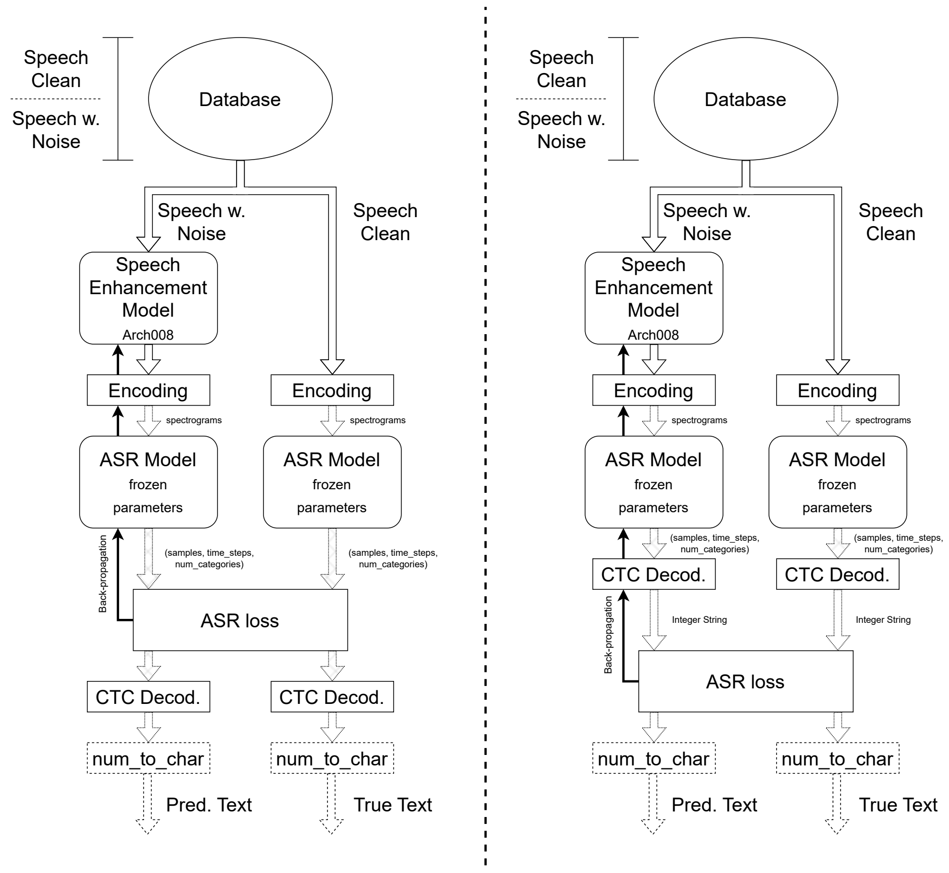


**Figure 4.3:** Flow Graph of the implementation. (Left) Comparison of posterior probability matrices using KLD. (Right) comparison of the decoded string of integers using the CTC loss function.

The noisy signal is first propagated forward through the layers of the Speech Enhancement model. Reconstruction of the waveform takes place in this process, aiming at removing the added noise or, as the name suggests, enhancing the speech. The reconstructed signal is then encoded (in the same way as the clean signal) and then is also fed in the ASR model.

In the KLD approach, two posterior probability matrices are compared to minimize the distance between them. In this case, the matrices of the clean and reconstructed (enhanced) signals are acquired from the ASR model output, and the Kullback–Leibler divergence is calculated. The loss is then backpropagated to adjust the weights of the Speech Enhancement model accordingly.

In the CTC approach, the output of the Speech Enhancement model is decoded using the CTC decoder. To do so, first, the posterior probability matrix of the clean signal is decoded to a string of integers again using the same CTC decoder. This is used as a target, and the loss is formed based on the difference between these two decoded strings.

### 4.3.2 Model Architecture

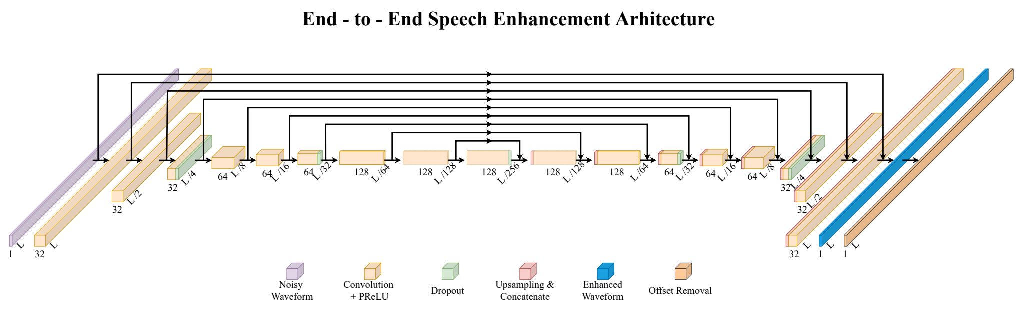


**Figure 4.4:** Architecture of the Speech Enhancement Model. The dropout of some layers is up to 20%.

The Speech Enhancement model architecture deployed in this Thesis is an encoder-decoder type fully convolutional neural network, which its benefits are described in 3.2. It was initially proposed for Speech Enhancement of signals in the time domain, using loss functions also in the time domain, such as Mean Squared Error (MSE) or Mean Absolute Error (MAE); however, it has been proved that these loss functions do not describe how humans perceive the speech, and proposed a frequency domain loss instead [13]. Other research based on this architecture showed that the model performs better in objective intelligibility tests when some form of objective intelligibility metric is deployed as a loss function, including both time and frequency domains, making this architecture a good choice for this implementation. [3], [6], [7], [9], [15].

The model consists of a total of 31 layers, where 17 of which are convolutional layers with a filter size of 11 samples, as can be seen in Figure 4.4. The numbers in the layers illustrated in the figure present the number of the feature maps of the convolutional layers, and as can be seen, the layers deployed in the encoder and decoder parts are mirrored.

The first section of the model is the encoder part, where, as can be seen in Figure 4.4, convolution is used in the input signal $\mathbf{x} \in \mathbb{R}^{L}$ in the first convolutional layer transforming it into $\hat{\mathbf{x}} \in \mathbb{R}^{L \times M}$, where $M$ is the number of feature maps, and t size of the input is decreased with a factor of 2, up to $L/256$. For the decoder section, the opposite process takes place as the size of the signal is also increased by a factor of 2. First, the information from the previous layer is upsampled and then concatenated with the output for the corresponding encoder layer, as the figure shows.

Finally, Parameterized Rectified Linear Unit (PReLU), discussed in Subsection 2.1.2, is used as an activation function for the convolutional layers, and dropout layers are used with a dropout of 20%.

### 4.3.3 Data Preparation

The first step to training a model to enhance speech signals would be adding noise to the clean signals. Subsection 4.1.2 describes how the noise $\mathbf{n}$ is added to the clean speech signal $\mathbf{x}_{clean}$. The noisy samples $\mathbf{x}_{noisy}$ are stored alongside their corresponding clean ones for comparison during the training and evaluation processes. In this way, the samples from the noisy signals are used as the input to the Speech Enhancement model while the samples from the clean signals (in any form, as will be described in the processes for calculating the loss functions in Subsections 4.3.4 and 4.3.5) are used as the Ground Truth $\mathbf{y}_{true} := \mathbf{x}_{clean}$. The output of the model with the enhanced signal is defined as $\mathbf{y}_{pred}$. In Neural Network training, in many cases there are benefits if the data are processed in batches, such as better memory allocation and more accurate statistics of the data. Therefore, in the context of this chapter $\mathbf{x}_{noisy}, \mathbf{y}_{true}, \mathbf{y}_{pred} \in \mathbb{R}^{B \times L}$, where $B$ is the size of the batch.

For this process, the audio files should be normalized in duration. The audio recordings are either split or zero-padded to fit the desired time. Due to the nature of the model architecture, the number of samples of the audio recordings should be an integer divisible by 256. Additionally, the length should be sufficient to fit the average length of speech passages. $2^{17}$ is chosen as the number of samples since, with a sampling rate of 22,050kHz, the length is approximately 6 seconds.

### 4.3.4 KLD in ASR Loss Function

In this approach of loss function using the ASR model, the loss is calculated based on the Kullback Leibler Divergence (KLD) of the two posterior probability matrices of the ASR model. This method relies on the hypothesis that by minimizing the probability distance between these two matrices, the decoding process would result in an identical output string, and, in addition, it is also more efficient than decoding the matrices.

A more detailed flow graph of the ASR loss function using KLD can be seen in Figure 4.5. The diagram shows that the two snippets of audio are processed in parallel. Although it is feasible, the clean signal is handed as constant for performance reasons and is preprocessed before the fitting of the Speech Enhancement model. Therefore, the dataset contains only the batch posterior probability matrix of the clean speech signal. In this way, the ASR model predicts the clean signals only once during the training process instead of having to predict the same information again and again in each epoch.

The loss is finally calculated by the KDL of the two posterior probability matrices. This metric is in the form of the distance between two hypotheses. Let $\mathbf{y}_{true}^{ppm}, \mathbf{y}_{pred}^{ppm} \in I^{B \times s \times V}$ the posterior probability matrices from clean and the enhanced signal, respectively, where $I$ represent the Unit Interval [0, 1] set of prob-

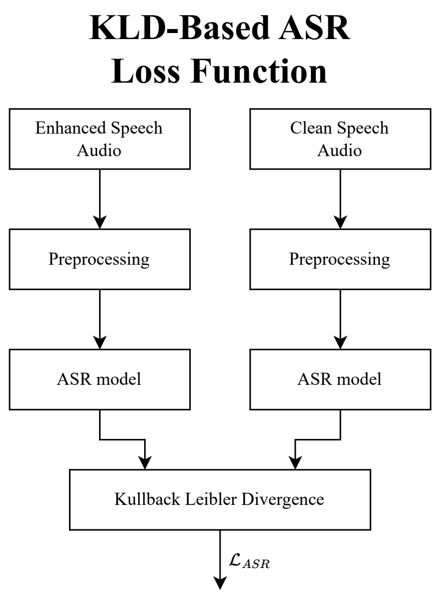


**Figure 4.5:** Flow graph of the ASR loss function, where the Kullback Leibler Divergence between the two posterior probability matrices is calculated. The two chained processes do not occur in parallel, as the clean signal can be treated as constant and calculated once.

abilities, $B$ the batch size, $s$ the number of time steps in the signal and $\mathcal{V}$ corresponds to the cardinality of the set of characters. The KLD loss function is defined as illustrated in Equation 4.4, where the symbol $\odot$ denotes the element-wise multiplication [35].

$$\mathcal{L}_{\mathbf{KLD}} = \mathbf{y}_{true}^{ppm} \odot log\left(\frac{\mathbf{y}_{true}^{ppm}}{\mathbf{y}_{pred}^{ppm}}\right) \tag{4.4}$$

### 4.3.5 CTC in ASR Loss Function

In this approach, the same process as with the Kullback Leibler Divergence is followed, with the difference that the CTC decoding process follows after the retrieval of the ASR posterior probabilities of the clean signal. The CTC decoding process searches for the most probable sequence, which is then aligned and returned as a string of integers. The decoded string of the clean signal is considered the Ground Truth. This method has the advantage of not considering the total entries of the posterior probabilities during the minimization process as KLD does. Applying CTC might be more expensive as it requires decoding the posterior probability matrix. However, in theory, it also focuses directly on the output string.

Again, the dataset is preprocessed beforehand for performance reasons to avoid

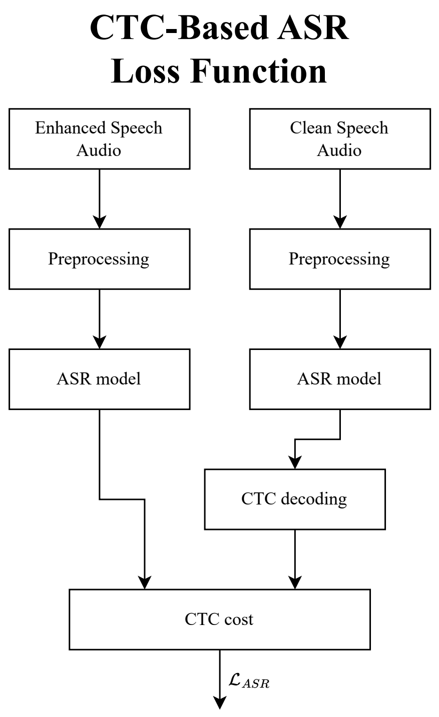


**Figure 4.6:** Flow graph of the ASR loss function, where the CTC cost between the posterior probability matrix of the ASR output with the enhanced signal and the decoded string of integers of the clean signal is calculated. The two chained processes do not occur in parallel, as the clean signal can be treated as constant and calculated once. The figure shows the posterior probability matrices propagating directly to the loss, as the CTC decoding is part of the CTC loss function in the implementation.

recalculating the spectrograms and the most probable string. Figure 4.6 illustrates a more detailed diagram of the loss function.

Appendix A describes how the CTC algorithm aligns non-even inputs and outputs. The loss function $\mathcal{L}_{\mathrm{CTC}}$, therefore, can be derived as illustrated in the Equation 4.5, with $\mathcal{D}$ symbolizing the vocabulary set and $\mathbf{y}_{pred}^{dec}, \mathbf{y}_{true}^{dec} \in \mathcal{D}^D$ the decoded strings, where $D$ a fixed integer defining the number of entries in the string. Zero-padding is applied if necessary. The fixed integer ensures that the same amount of entries is compared each time. Although this is not important for the CTC algorithm, software libraries such as *TensorFlow* that were used for the implementation can really benefit from constant dimensions of data.

$$\mathcal{L}_{CTC} = \sum_{(y_{pred}^{dec}, y_{true}^{dec}) \in \mathcal{D}} -\log\, p(y_{true}^{dec} | y_{pred}^{dec}) \tag{4.5}$$

The loss $\mathcal{L}_{\mathrm{CTC}}$ stands for one entry. Loss $\mathcal{L}_{\mathrm{CTC}}^{D}$ in vector form contains individual entries of $\mathcal{L}_{\mathrm{CTC}}$. $D$ Depends on the number of windows applied in the signal during the STFT (steps), as each one of the steps defines an entry of probabilities over the whole set of recognizable characters. Integer $D$ is set as 200, which works well with 6 seconds of audio used in the training process. As a reminder, multiple continuous entries can contain the same element, which is discarded later in the decoding process, as discussed in Appendix A.

### 4.3.6 Training and Validation

For the training process, the data of the LJSpeech dataset is split into Training Validation and Test subsets by 80%, 10% and 10%, respectively. Each sample in the Test subset is further segmented into two 2-second samples. For Test samples that have very short utterances that last less than 2 seconds, the second segment is tossed. In this way, more than 2,000 Training samples that are needed for Testing purposes are available.

At the end of each epoch, the performance of the Speech Enhancement model is tested on the validation dataset by how accurately the ASR model performs on the enhanced outputs. This is just for visualization purposes to illustrate whether the performance of the speech enhancement model is improving or not over the epochs. At this stage, WER and CER are also calculated; however, these metrics do not play another role in the training process other than indicating whether the training process was successful or not.

The learning rate of the optimizer is 0.0005, as it has been proven to work well and is an essential factor in the training process [3], [15]. Similarly to the training process of the ASR model discussed in Subsection 4.2.5, early stopping and learning rate scheduling is deployed. These methods have a patience parameter that determines how many epochs of no improvements in the validation loss the method waits until it takes place.

#### Training with Single Loss Function

Training with a single loss function means only the ASR loss is used. ASR loss includes either the method using KLD or the one using CTC. During the training, the loss is calculated in mini-batches and is updated as a running average over each mini-batch. When all mini-batches are propagated forward (epoch), the average loss of the mini-batches is defined as the loss of the epoch. The validation dataset is then used to calculate the validation loss of the model.

#### Training with Composite Loss Function

In this type of training, a composite loss is used, which combines the ASR loss, as described in Subsection 4.3.6, with a loss function that has been proven to work well. In this case, SI-SDR is used due to its good performance in speech enhancement tasks [3]. SI-SDR objective metric is defined in Equation 3.15. Due to the fact, however, that higher SI-SDR values mean better performance, maximizing the performance would be equivalent to minimizing the negative of SI-SDR. This is defined as the loss function [3], [15].

The composite loss is defined by weighting individual losses and adding them together. The function is defined in Equation 4.6, where $\alpha, \beta$ are the weights where the individual loss functions are weighted with. The weights are normalized for simplicity, meaning that $\alpha + \beta = 1$. The $\mathcal{L}_{\text{ASR}}$ defines the ASR loss function, either using CTC or KLD methods.

$$\mathcal{L}_{\text{composite}} = \alpha \mathcal{L}_{\text{ASR}} + \beta \mathcal{L}_{\text{SI-SDR}} \tag{4.6}$$

## 4.4 Metrics Evaluation

This section discusses the process used to test the performance of the speech enhancement model. Additionally, it is discussed what the baseline of comparison is to determine how this implementation proposed in the Thesis stands against previously proposed methods.

### 4.4.1 Baseline

As mentioned, this project relies on previous experiments. It would therefore be advantageous to recreate the experiment of interest and reproduce the results with the LJSpeech dataset used in this Thesis. Due to time limitations, only one method is used for comparison, precisely the SI-SDR loss function [3]. The model and the training process are identical to the ones mentioned for training using ASR loss, with the difference that the SI-SDR loss function is used.

### 4.4.2 Test Process

The intelligibility, quality and waveform-level metrics used for evaluation purposes were discussed in Section 3.1. For the training process, the 7 noise types discussed in Subsubsection 4.1.1 are used and evaluated over the set of SNR values { -10, -5, 0, 5, 10, 15, 20}. For the process, 2000 files are randomly selected from the dataset and chopped into a length of 44,288 samples, which is about 2 seconds of audio, and then equal length noise is added to the speech signals. Separate experiments are conducted with all the different types of noises. The noisy signals are then enhanced. The resulting estimated speech signals are downsampled into 8k, as the intelligibility metrics considered require it. The noise is added in the same way as the training process, as mentioned in Subsection 4.1.2, again by using the ITU-T

P.56 recommendation for scaling the noise signals. Although many metrics require signals without silence, the silent parts are not removed before enhancement. This approach is since, in real-work cases, the absence of speech in a received signal is unknown [15]. Finally, the noisy signals are enhanced using the speech enhancement model, and the enhanced signals are compared with the original clean ones to calculate the metrics.

# Chapter 5

# Results

This chapter presents the quality and intelligibility metrics resulting from testing the performance of the Speech Enhancement model.

## 5.1 Results with Test Speech Data

**Table 5.1:** Average Results

| SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ | SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | STOI | 0.53591 | 0.64452 | 0.45895 | 0.65264 | | STOI | 0.64121 | 0.76171 | 0.5127 | 0.77078 |
| | ESTOI | 0.21518 | 0.4285 | 0.23001 | 0.40555 | | ESTOI | 0.33586 | 0.59091 | 0.32833 | 0.57642 |
| -10 | PESQ | 1.2295 | 1.5526 | 1.1988 | 1.5607 | -5 | PESQ | 1.3299 | 1.8621 | 1.2519 | 1.8697 |
| | SDR | -9.4803 | 3.0189 | -11.2334 | 2.6899 | | SDR | -4.7512 | 6.7852 | -8.9696 | 6.4735 |
| | SI-SDR | -9.9344 | -0.1433 | -17.8826 | -0.84904 | | SI-SDR | -4.9295 | 4.455 | -14.5126 | 3.6516 |
| | STOI | 0.74969 | 0.83442 | 0.56638 | 0.84782 | | STOI | 0.84198 | 0.88813 | 0.60367 | 0.89501 |
| | ESTOI | 0.47392 | 0.70646 | 0.41203 | 0.69904 | | ESTOI | 0.61307 | 0.79293 | 0.46845 | 0.79022 |
| 0 | PESQ | 1.509 | 2.1752 | 1.3153 | 2.1938 | 5 | PESQ | 1.7707 | 2.4764 | 1.3731 | 2.4947 |
| | SDR | 0.15901 | 10.2739 | -7.3054 | 10.0429 | | SDR | 5.1303 | 13.5717 | -6.241 | 13.5115 |
| | SI-SDR | 0.072299 | 8.4344 | -12.29 | 7.9008 | | SI-SDR | 5.073 | 12.249 | -10.752 | 12.0435 |
| | STOI | 0.90846 | 0.93802 | 0.63025 | 0.94119 | | STOI | 0.95026 | 0.96668 | 0.65579 | 0.96648 |
| | ESTOI | 0.73826 | 0.86458 | 0.51379 | 0.86412 | | ESTOI | 0.83773 | 0.90984 | 0.55853 | 0.90772 |
| 10 | PESQ | 2.1201 | 2.8188 | 1.4178 | 2.8242 | 15 | PESQ | 2.5542 | 3.1718 | 1.4533 | 3.1551 |
| | SDR | 10.1212 | 16.7217 | -5.2926 | 16.7109 | | SDR | 15.1184 | 19.6239 | -4.4494 | 19.5289 |
| | SI-SDR | 10.0733 | 15.7264 | -9.4357 | 15.4003 | | SI-SDR | 15.0734 | 18.6507 | -8.4657 | 18.0039 |
| | STOI | 0.97402 | 0.9775 | 0.66904 | 0.97586 | | | | | | |
| | ESTOI | 0.90667 | 0.93334 | 0.58435 | 0.92923 | | | | | | |
| 20 | PESQ | 3.0552 | 3.4533 | 1.4817 | 3.4274 | | | | | | |
| | SDR | 20.1175 | 21.8724 | -4.0102 | 21.6212 | | | | | | |
| | SI-SDR | 20.0735 | 20.7345 | -8.0335 | 19.7123 | | | | | | |

In this section, the results from testing the model's performance in the test subset of the LJSpeech dataset, in which the Speech Enhancement and ASR models were trained on. The process is explained in Subsection 4.4.2. All the methods that have been compared use the same DNN model as described in 4.3.2. The loss functions tested are ASR with KLD, ASR with CTC, composite function with KLD and SI-SDR, composite function with CTC and SI-SDR, and the SI-SDR for reference. The two composite methods are formed as shown in Equation 4.6 with importance factors $\alpha = \beta = 0.5$. Both single CTC and composite CTC did not converge sufficiently during the training, and the models using these two functions produced noisy and distorted outputs, even during the silent parts of the signal. For this reason, their results are not presented.

**Table 5.2:** SSN (seen noise)

| SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ | SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -10 | STOI | 0.47666 | 0.58812 | 0.43765 | 0.59265 | -5 | STOI | 0.58508 | 0.78648 | 0.48752 | 0.78007 |
| | ESTOI | 0.13378 | 0.36182 | 0.16634 | 0.30946 | | ESTOI | 0.26351 | 0.59102 | 0.29261 | 0.56228 |
| | PESQ | 1.2153 | 1.3843 | 1.1495 | 1.3725 | | PESQ | 1.2695 | 1.7631 | 1.1933 | 1.7382 |
| | SDR | -9.4027 | 2.2578 | -12.0388 | 1.6703 | | SDR | -4.722 | 7.1736 | -9.0043 | 6.7966 |
| | SI-SDR | -9.9356 | 0.25323 | -16.8712 | -0.23425 | | SI-SDR | -4.9312 | 5.9312 | -12.7616 | 5.5256 |
| 0 | STOI | 0.71441 | 0.87864 | 0.53923 | 0.87679 | 5 | STOI | 0.82699 | 0.92371 | 0.58691 | 0.92315 |
| | ESTOI | 0.42619 | 0.73402 | 0.41405 | 0.72748 | | ESTOI | 0.58986 | 0.81741 | 0.49785 | 0.81659 |
| | PESQ | 1.3807 | 2.107 | 1.2705 | 2.1338 | | PESQ | 1.5529 | 2.433 | 1.3538 | 2.4721 |
| | SDR | 0.17147 | 10.7831 | -6.7953 | 10.4802 | | SDR | 5.137 | 13.6466 | -5.4257 | 13.3654 |
| | SI-SDR | 0.069708 | 9.8032 | -10.4313 | 9.4495 | | SI-SDR | 5.0698 | 12.8792 | -9.2098 | 12.4769 |
| 10 | STOI | 0.90408 | 0.94925 | 0.62443 | 0.9483 | 15 | STOI | 0.94991 | 0.96486 | 0.64876 | 0.96329 |
| | ESTOI | 0.73034 | 0.86871 | 0.54749 | 0.86636 | | ESTOI | 0.83709 | 0.90206 | 0.57436 | 0.89853 |
| | PESQ | 1.8159 | 2.7583 | 1.419 | 2.7535 | | PESQ | 2.1787 | 3.0213 | 1.4622 | 3.0027 |
| | SDR | 10.1259 | 16.195 | -4.6195 | 15.9536 | | SDR | 15.1223 | 18.6092 | 15.0695 | 18.4463 |
| | SI-SDR | 10.0696 | 15.5351 | -8.5307 | 15.0685 | | SI-SDR | 15.0695 | 17.9127 | -4.1726 | 17.3464 |
| 20 | STOI | 0.97535 | 0.97448 | 0.66271 | 0.97252 | | | | | | |
| | ESTOI | 0.91006 | 0.92475 | 0.58813 | 0.92039 | | | | | | |
| | PESQ | 2.6267 | 3.2647 | 1.4864 | 3.26 | | | | | | |
| | SDR | 20.1211 | 20.8755 | -3.9288 | 20.7011 | | | | | | |
| | SI-SDR | 20.0694 | 19.9844 | -7.9455 | 19.1663 | | | | | | |

Regarding the metrics presented in this Chapter, some return invalid ("None" type) results due to lack of signal energy, divisions by 0, etc. These values are excluded from the averaging performance of the metrics. The average results include all 7 noises, 4 of which have been used in the training process, and 3 are not. The average results across all 7 noise types can be seen in Table 5.1, where the noisy column stands for the measurements of the metrics in the unprocessed mixture of noise and speech signal samples. The following three columns stand for SI-SDR loss function, ASR with KLD loss function and composite SI-SDR and ASR with KLD. The best-performing model in each metric and for each SNR is highlighted in green colour.

To further show how the models perform in seen and unseen data, two more Tables are presented. Table 5.2 shows the average performance of the models in speech-shaped noise (SSN), which is used in the training process. Similarly, Table 5.3 shows the results across construction building noise to which the models have not been exposed to. In both tables, Green colour highlights the best-performing model, while red also highlights the best-performing model, but the model performance is degraded compared to the unprocessed noisy sample.

Regarding the LJSpeech dataset, it is clear that the ASR loss function is not performing well, both in the case of the CTC method which heavily distorts the

**Table 5.3:** Inside Construction Site (unseen noise)

| SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ | SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -10 | STOI | 0.58469 | 0.35331 | 0.37836 | 0.37635 | -5 | STOI | 0.6952 | 0.35967 | 0.41424 | 0.43693 |
| | ESTOI | 0.25056 | 0.10702 | 0.028634 | 0.091839 | | ESTOI | 0.38339 | 0.19044 | 0.056871 | 0.16761 |
| | PESQ | 1.2129 | 1.1311 | 1.1702 | 1.169 | | PESQ | 1.3406 | 1.1536 | 1.1797 | 1.195 |
| | SDR | -9.5462 | -14.3154 | -18.0037 | -14.5018 | | SDR | -4.7779 | -10.1859 | -16.8373 | -10.6875 |
| | SI-SDR | -9.929 | -24.0784 | -38.2298 | -27.2134 | | SI-SDR | -4.9275 | -17.7481 | -32.7652 | -21.3995 |
| 0 | STOI | 0.79743 | 0.43992 | 0.47151 | 0.53871 | 5 | STOI | 0.87712 | 0.59455 | 0.49612 | 0.64675 |
| | ESTOI | 0.52582 | 0.31068 | 0.084493 | 0.2792 | | ESTOI | 0.66311 | 0.49813 | 0.10878 | 0.48919 |
| | PESQ | 1.555 | 1.2132 | 1.1674 | 1.2328 | | PESQ | 1.8632 | 1.3668 | 1.1654 | 1.4164 |
| | SDR | 0.14547 | -4.9798 | -15.5785 | -5.2284 | | SDR | 5.1209 | 1.4017 | -14.3601 | 2.1687 |
| | SI-SDR | 0.072818 | -10.8925 | -27.9127 | -12.404 | | SI-SDR | 5.0729 | -2.1472 | -22.7767 | -0.78117 |
| 10 | STOI | 0.93092 | 0.81831 | 0.53289 | 0.84671 | 15 | STOI | 0.9629 | 0.943 | 0.625 | 0.95084 |
| | ESTOI | 0.78147 | 0.73845 | 0.21362 | 0.75039 | | ESTOI | 0.86968 | 0.88173 | 0.41363 | 0.88775 |
| | PESQ | 2.2561 | 1.9803 | 1.2004 | 2.0748 | | PESQ | 2.7271 | 2.8794 | 1.2857 | 2.8828 |
| | SDR | 10.113 | 8.3316 | -11.1605 | 9.2369 | | SDR | 15.1105 | 15.0854 | -7.1155 | 15.4251 |
| | SI-SDR | 10.0729 | 6.6298 | -16.472 | 7.9258 | | SI-SDR | 15.0728 | 14.0409 | -11.2254 | 14.3174 |
| 20 | STOI | 0.98042 | 0.97477 | 0.67009 | 0.97469 | | | | | | |
| | ESTOI | 0.92622 | 0.93178 | 0.53858 | 0.93014 | | | | | | |
| | PESQ | 3.2696 | 3.4199 | 1.4001 | 3.3669 | | | | | | |
| | SDR | 20.1097 | 19.939 | -5.0742 | 19.678 | | | | | | |
| | SI-SDR | 20.0728 | 18.8609 | -9.061 | 18.1601 | | | | | | |

signal and in the case of KLD which as can be seen in all tables, degrades the signal instead of enhancing it. The composite loss function using SI-SDR and ASR with KLD shows slightly lower performance in average results in the intelligibility metrics compared to SI-SDR. Interestingly, For SNRs less or equal to 10, the PESQ, which is a quality metric, favoured the composite function.

Taking a look at the individual noises results, one can see that the results between the average performance and the SSN are pretty similar, with the SI-SDR loss performing better. It has already been proven, however, that SI-SDR performs well in stationary noise [3], [15]. On the other hand, on the unseen noise type of construction site, the composite loss seems to perform better than SI-SDR in many cases. According to the metrics, however, this performance is not sufficient enough, as in most cases, the enhanced signal is worse than the unprocessed noisy signal.

## 5.2 Results with Unseen Speech Data

As both models have been trained using the LJSpeech dataset, which contains a single female speaker, a supplementary dataset is adopted to test the models with speech signals from unseen speakers. The dataset has been described in Subsection 4.1.1. The performance of the models is illustrated in Table 5.4. As can be seen, the results are similar to the unknown noise type. The composite loss shows better performance in quality metrics in general. Although it seems to be improvements in enhancement for both SI-SDR and composite loss function in SNRs of -10 and -5, in anything above that, the results are getting worse. Especially for higher SNRs where the clean signal is more pronounced in the noisy mixture, the results

**Table 5.4:** SLR83 (unseen speakers)

| SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ | SNR | Metric | Noisy | $\mathcal{L}_{SI-SDR}$ | $\mathcal{L}_{KLD}$ | $\mathcal{L}_{SISDR+KLD}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -10 | STOI | 0.5556 | 0.48297 | 0.41708 | 0.49981 | -5 | STOI | 0.66624 | 0.5827 | 0.44307 | 0.59052 |
| | ESTOI | 0.23836 | 0.28835 | 0.14088 | 0.24657 | | ESTOI | 0.37466 | 0.40268 | 0.19864 | 0.36611 |
| | PESQ | 1.2739 | 1.3368 | 1.2296 | 1.3487 | | PESQ | 1.35 | 1.4701 | 1.2166 | 1.4712 |
| | SDR | -10.6368 | -3.2273 | -14.5561 | -3.5838 | | SDR | -6.026 | -0.16839 | -12.7566 | -0.41525 |
| | SI-SDR | -11.2783 | -10.3118 | -22.5875 | -11.2015 | | SI-SDR | -6.2719 | -6.0333 | -18.9408 | -7.0299 |
| 0 | STOI | 0.77889 | 0.66043 | 0.47451 | 0.67352 | 5 | STOI | 0.87139 | 0.71333 | 0.49997 | 0.72402 |
| | ESTOI | 0.53005 | 0.50327 | 0.25782 | 0.47551 | | ESTOI | 0.68241 | 0.58396 | 0.3078 | 0.56606 |
| | PESQ | 1.5086 | 1.6292 | 1.2193 | 1.6306 | | PESQ | 1.7464 | 1.7997 | 1.2395 | 1.8069 |
| | SDR | -1.1589 | 2.3143 | -11.1302 | 2.1189 | | SDR | 3.7978 | 4.4168 | -9.9221 | 4.3777 |
| | SI-SDR | -1.2701 | -3.2398 | -16.2466 | -3.9302 | | SI-SDR | 3.7304 | -1.0848 | -14.3674 | -1.2115 |
| 10 | STOI | 0.93396 | 0.74614 | 0.51793 | 0.76199 | 15 | STOI | 0.96978 | 0.7679 | 0.5348 | 0.78879 |
| | ESTOI | 0.80996 | 0.6464 | 0.35255 | 0.64169 | | ESTOI | 0.89981 | 0.69285 | 0.39108 | 0.69388 |
| | PESQ | 2.0589 | 1.9608 | 1.2605 | 1.9924 | | PESQ | 2.4426 | 2.1099 | 1.2754 | 2.1771 |
| | SDR | 8.7839 | 6.0859 | -8.9812 | 6.2507 | | SDR | 13.7795 | 7.2031 | -8.1202 | 7.4408 |
| | SI-SDR | 8.7305 | 0.62368 | -12.9913 | 0.91502 | | SI-SDR | 13.7305 | 1.786 | -11.7821 | 2.1209 |
| 20 | STOI | 0.98752 | 0.78141 | 0.54643 | 0.80424 | | | | | | |
| | ESTOI | 0.95278 | 0.72323 | 0.41703 | 0.72513 | | | | | | |
| | PESQ | 2.8739 | 2.2542 | 1.2883 | 2.3408 | | | | | | |
| | SDR | 18.7781 | 7.8375 | -7.4793 | 8.0743 | | | | | | |
| | SI-SDR | 18.7305 | 2.4241 | -10.9806 | 2.6817 | | | | | | |

are disappointing as all methods heavily degrade it instead of enhancing it. This shows the importance of the data in data-driven methods such as DNNs. Finally, the KLD loss function failed in to improve the signals in all test cases.

Chapter 6

# Discussion and Conclusion

This chapter discusses the results presented in the previous chapter and the conclusion of the Thesis.

## 6.1 Discussion

The challenge in ASR-based Speech Enhancement is that the performance of the Speech Enhancement model depends on the performance of the ASR model. Although there are many available well-organized datasets containing speech utterances, the amount of these that also contain utterances in text form is significantly lower. The Wall Street Jurnal (WSJ) dataset, which is widely used in speech enhancement, was a choice; however, the sampling rate that was recorded is 16kHz and also many subsets of the datasets do not include the text. LJspeech is a better option for training the ASR model, however, due to the fact that it is recorded at 22,050Hz and the model draws its features from the frequency spectrum, it means that the trained ASR model would be incompatible with the lower sampling rate of the WSJ. In other words, training the ASR model with LJSpeech and the Speech Enhancement with WSJ was not an option, at least without downsampling LJSpeech. Due to the limited time of this project, the LJSpeech dataset was selected for training both ASR and Speech Enhancement models, even though it has the drawback of only one speaker reading all the utterances. The effect that the lack of variety in speakers has on the enhancement process is clear in the results of the model in the SLR83 dataset presented in 5.4, which are not in line with previous findings, specifically with SI-SDR loss function [3], [15]. The results based on the LJSpeech dataset presented in Chapter 5 are sufficient enough to make a conclusion; however, the limitations of the dataset in the performance of the models should be taken into account.

## 6.2 Conclusion

It is clear that ASR as a loss function cannot be used alone for Speech Enhancement. Especially using the CTC method that performs well in ASR tasks shows terrible results in Speech Enhancement. Additionally, although the KLD method manages to make the speech pronounced clearer in the noisy samples compared to CTC, listening to the estimated samples is unpleasant as it distorts both speech present and speech absent parts.

On the other hand, the composite loss function using the ASR loss in combination with some other loss function, the SI-SDR in this case, showed more promising results. Not only in many cases, the resulting metrics are slightly behind the SI-SDR, but also, the composite loss showed better performance in the quality (PESQ) metric compared to using SI-SDR alone.

As a conclusion for this Thesis, one can argue whether the methods using ASR as a loss function discussed in this report can stand compared to state-of-the-art methods used in Speech Enhancement. On the one hand, it is shown that combining ASR with other loss functions can improve certain areas of the signal, such as the quality in the case of ASR and SI-SDR. On the other hand, the deviation of the composite loss from the SI-SDR loss function in the intelligibility and waveform-based metrics did not show sufficient improvement to justify the extra effort required for proper training of an ASR model and the extra computation power that is required during the training compared to solely using SI-SDR as a loss functions. And even for the gains in the quality metrics, the difference is so small in most cases that the actual subjective difference is negligible.

# Bibliography

[1] S. Choi, H. Hong, H. Glotin, and F. Berthommier, "Multichannel signal separation for cocktail party speech recognition: A dynamic recurrent network," eng, *Neurocomputing (Amsterdam)*, vol. 49, no. 1, pp. 299–314, 2002, ISSN: 0925-2312.

[2] P. C. Loizou, *Speech enhancement : Theory and practice, Second Edition* (Signal processing and communications.), eng. London: Taylor & Francis ltd, 2017, ISBN: 9781138075573.

[3] M. Kolbæk, Z.-H. Tan, S. H. Jensen, and J. Jensen, "On loss functions for supervised monaural time-domain speech enhancement," eng, 2020, ISSN: 23299290.

[4] P. Hoang, Z. H. Tan, J. M. D. Haan, and J. Jensen, "The minimum overlap-gap algorithm for speech enhancement," eng, 2022, ISSN: 21693536.

[5] Y. Ephraim and D. Malah, "Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator," eng, *IEEE transactions on acoustics, speech, and signal processing*, vol. 32, no. 6, pp. 1109–1121, 1984, ISSN: 0096-3518.

[6] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "An algorithm for intelligibility prediction of time-frequency weighted noisy speech," eng, *IEEE transactions on audio, speech, and language processing*, vol. 19, no. 7, pp. 2125–2136, 2011, ISSN: 1558-7916.

[7] J. Jensen and C. Taal, "An algorithm for predicting the intelligibility of speech masked by modulated noise maskers," eng, *IEEE/ACM transactions on audio, speech, and language processing*, vol. 24, no. 11, pp. 2009–2022, 2016, ISSN: 2329-9290.

[8] J. L. Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "Sdr - half-baked or well done?" eng, 2018.

[9] J. M. Martin-Doñas, A. M. Gomez, J. A. Gonzalez, and A. M. Peinado, "A deep learning loss function based on the perceptual evaluation of the speech quality," eng, *IEEE signal processing letters*, vol. 25, no. 11, pp. 1680–1684, 2018, ISSN: 1070-9908.

[10] Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, "A regression approach to speech enhancement based on deep neural networks," eng, *IEEE/ACM transactions on audio, speech, and language processing*, vol. 23, no. 1, pp. 7–19, 2015, ISSN: 2329-9290.

[11] R. Rehr and T. Gerkmann, "Normalized features for improving the generalization of dnn based speech enhancement," eng, 2017.

[12] A. Pandey and D. Wang, "A New Framework for Supervised Speech Enhancement in the Time Domain," in *Proc. Interspeech 2018*, 2018, pp. 1136–1140. DOI: `10.21437/Interspeech.2018-1223`.

[13] A. Pandey and D. Wang, "Tcnn: Temporal convolutional neural network for real-time speech enhancement in the time domain," eng, in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 6875–6879, ISBN: 9781479981311.

[14] S.-W. Fu, W. Tao-Wei, Y. Tsao, X. Lu, and H. Kawai, "End-to-end waveform utterance enhancement for direct evaluation metrics optimization by fully convolutional neural networks," eng, *arXiv.org*, 2018, ISSN: 2331-8422.

[15] I. López-Espejo, A. Edraki, W.-Y. Chan, Z.-H. Tan, and J. Jensen, "On the deficiency of intelligibility metrics as proxies for subjective intelligibility," eng, *Speech communication*, vol. 150, pp. 9–22, 2023, ISSN: 0167-6393.

[16] A. S. Subramanian, X. Wang, M. K. Baskar, *et al.*, "Speech enhancement using end-to-end speech recognition objectives," in *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2019, pp. 234–238. DOI: `10.1109/WASPAA.2019.8937250`.

[17] C. Bishop, *Pattern recognition and machine learning.* (Information science and statistics.), eng. Berlin: Springer, 2006, ISBN: 9780387310732.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," eng, in *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, 2015, pp. 1026–1034, ISBN: 1467383910.

[20] U. Kamath, *Deep Learning for NLP and Speech Recognition*, eng, 1st ed. 2019. Cham: Springer International Publishing, 2019, ISBN: 3-030-14596-4.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," eng, 2014.

[22] T. F. Quatieri, *Discrete-time speech signal processing : principles and practice* (Prentice-Hall signal processing series.), eng. Upper Saddle River, NJ: Prentice Hall, 2001, ISBN: 013242942X.

[23] *Sequence modeling with ctc*, `https://distill.pub/2017/ctc/`, Accessed: 08-02-2023.

[24] A. Rix, J. Beerends, M. Hollier, and A. Hekstra, "Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs," eng, in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, vol. 2, IEEE, 2001, 749–752 vol.2, ISBN: 0780370414.

[25] C. Févotte, R. Gribonval, and E. Vincent, *BSS_EVAL Toolbox User Guide – Revision 2.0*, eng. 2005, pp. 19–.

[26] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, "A short-time objective intelligibility measure for time-frequency weighted noisy speech," eng, in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, IEEE, 2010, pp. 4214–4217, ISBN: 9781424442959.

[27] B. B. Madsen, G. Kiss, M. Borresen, and M. Kampitakis, "Robustness of Acoustic Scene Classification Using a CNN in a real-world scenario," Edited Worksheet for 7nth Semester Project, 2021.

[28] J. Barker, R. Marxer, E. Vincent, and S. Watanabe, "The third 'chime' speech separation and recognition challenge: Dataset, task and baselines," eng, 2015.

[29] J. S., "TIMIT acoustic-phonetic continuous speech corpus," Linguistic Data Consortium, 1993, ISBN: 1-58563-019-5. DOI: `https://doi.org/10.35111/17gk-bn40`. [Online]. Available: `https://catalog.ldc.upenn.edu/LDC93s1`.

[30] K. Ito and L. Johnson, *The lj speech dataset*, `https://keithito.com/LJ-Speech-Dataset/`, 2017.

[31] I. Demirsahin, O. Kjartansson, A. Gutkin, and C. Rivera, "Open-source Multi-speaker Corpora of the English Accents in the British Isles," in *Proceedings of The 12th Language Resources and Evaluation Conference (LREC)*, Marseille, France: European Language Resources Association (ELRA), May 2020, pp. 6532–6541, ISBN: 979-10-95546-34-4. [Online]. Available: `https://www.aclweb.org/anthology/2020.lrec-1.804`.

[32] *ITU-T p.56: Objective measurement of active speech level*, `https://www.itu.int/rec/T-REC-P.56`, 2011.

[33] *Automatic Speech Recognition using ctc*, `https://keras.io/examples/audio/ctc_asr/`, Accessed: 02-03-2023.

[34] *DeepSpeech2 model (nvidia)*, `https://nvidia.github.io/OpenSeq2Seq/html/speech-recognition/deepspeech2.html`, Accessed: 02-03-2023.

[35] *TensorFlow kullback–leibler divergence*, `https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence`, Accessed: 12-04-2023.

# Appendix A

# CTC Decoding

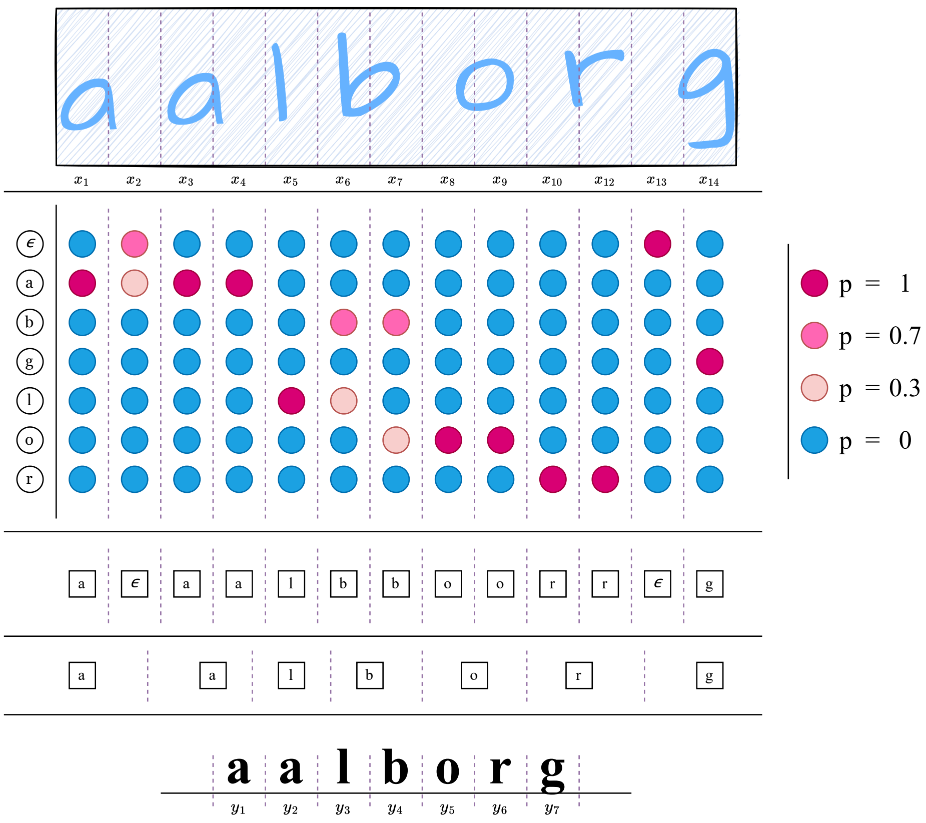
This chapter discusses the algorithm of Connectionist Temporal Classification, which is a method for aligning uneven sequences between input and output. First, the algorithm is described, and then the use case as a loss function is explained.

## A.1 Algorithm Description

Connectionist Temporal Classification (CTC) is an algorithm widely used in Machine Learning and specifically in Deep Neural Networks for tasks that the input does not perfectly align with the output. Examples of this kind of problem could be automatic speech recognition, automatic handwriting recognition or other types of sequence problems. For simplicity, an example with set $\mathcal{D}$ of characters is illustrated; however, this method can be used in any use mentioned above cases [23].

### A.1.1 Decoding Process

Assuming the input as a length of samples $\mathbf{X} = [x_1, x_2, x_3, \ldots, x_T]$ and the ground truth as $\mathbf{Y} = [y_1, y_2, y_3, \ldots, y_U]$, the CTC algorithm addresses three problems. The lengths of $\mathbf{X}$ and $\mathbf{Y}$ are unknown and can vary. The ratio between these two can also vary; finally, no accurate alignment is known beforehand. Figure A.1 illustrates a naive example of this process with 14 input samples and seven output samples. Samples have been retrieved from an image shown in the first stage of the figure, and as can be seen in the example, the length of input samples and ground truth samples are different. Supposing one has access to the posterior probabilities matrix of $\mathcal{D}$, as shown in the second stage of the figure, two methods can be used to decode the string and get the results illustrated in the third stage. The first method is greedy search, which calculates the optimal output $\mathbf{Y}^*$ as shown in the Equation A.1 [23]



$x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ $x_9$ $x_{10}$ $x_{12}$ $x_{13}$ $x_{14}$
$\epsilon$
a
b
g
l
o
r
p = 1
p = 0.7
p = 0.3
p = 0
a $\epsilon$ a a l b b o o r r $\epsilon$ g
a a l b o r g
a a l b o r g
$y_1$ $y_2$ $y_3$ $y_4$ $y_5$ $y_6$ $y_7$

**Figure A.1:** An exaggerated example of the CTC process for a handwritten-to-text classification. All the letters are lowercase to simplify the example. The same example could have been illustrated with spectrograms-to-text to showcase the actual use of the CTC in this Thesis. The example of handwritten characters, however, makes clearer pointing out the process.

$$\mathbf{Y}^* = \underset{\mathbf{Y}}{\arg\max}\, p(\mathbf{Y}|\mathbf{X}) \tag{A.1}$$

The second method is called beam search, and as the name suggests, beams are used to estimate the total probabilities for possible paths, and the most probable path is selected. Although not necessarily the most probable character is used, the beam search can give an asymptotically better solution. Beam search calculates the optimal beam $A^*$ by searching over several possible beams $A \in \mathbf{A}_{X,Y}$ as shown in the Equation A.2 [23].

$$A^* = \underset{A}{\arg\max} \prod_{t=1}^{T} p_t(a_t|\mathbf{X}) \tag{A.2}$$

### A.1.2 Aligning process

CTC function uses a special token $\epsilon$ which denotes the characters that should not be merged. This can be seen in the third and fourth stage of Figure A.1 between the letters "a". Letters "b", "o", and "r", on the other hand, do not have special tokens between them and therefore are merged. In this way, the final output $\mathbf{Y} = [y_1, y_2, \ldots, y_7]$ shown in the last stage is retrieved.

## A.2 CTC as loss function

CTC loss function provides a natural way of transitioning from each sample's individual probabilities to the output sequence's overall likelihood. The type of search can be tuned depending on the problem type, considering that the beam search is more expensive than the greedy search.

The CTC as a loss function can be used to maximize the probability of selecting the correct entries of $\mathcal{D}$. Maximizing the likelihood is equivalent of minimizing the negative log, and the loss function is given for any set space $\mathcal{D}$ as shown in the Equation A.3 [23].

$$\mathcal{L}_{CTC} = \sum_{(X,Y)\in\mathcal{D}} -\log p(Y|X) \tag{A.3}$$