Speech Enhancement with Convolutional Autoencoder and LSTM Neural Networks

Richard Gilchrist - 201983648

Date: 21/08/2020

University of Strathclyde

Department of Electronic & Electrical Engineering

Supervisor: Professor John Soraghan

Declaration

I, Richard Gilchrist, hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Professor John Soraghan.

Signed: 

Date: 21/08/2020

Acknowledgements

I would like to thank Professor John Sorgahan for his help and advice over the course of this project. I would also like to thank John's PhD students, Qiming Zhu and Huiyi Wu, who helped me with some questions and problems I had along the way.

Finally, I would like to thank Kathryn, for listening to me talk about this project for the past 3 months.

Contents

List of Figures

List of Tables

Abstract

Speech enhancement is concerned with the removal of noise from speech signals, while maintaining the speech quality and intelligibility. There are numerous applications of speech enhancement, including hearing aids and communications, as well as human-machine interaction applications involving speech. Numerous conventional signal processing approaches have been developed, though these tend to perform poorly in non-stationary noise conditions. This work discusses some of the deep learning approaches that have been used for speech enhancement.

A testbed system was developed to evaluate the performance of a convolutional autoencoder (CAE) model and a long short-term memory (LSTM) model. Variants on each of these models were evaluated, making use of different training targets and utilising skip connections in the CAE. A variety of noise types and conditions were investigated, including the case of negative SNRs and the particularly challenging case of multi-talker babble noise. Model performance was evaluated in terms of short-time objective intelligibility (STOI) and segmental SNR (SSNR).

The results showed that the models trained with the ideal ratio mask (IRM) target consistently outperformed those trained with the ideal binary mask (IBM) target. The models were found to be unable to effectively generalise to new, unseen data. A number of explanations for this observation are proposed and recommendations for future work to improve the performance of the system are presented.

Chapter 1

Introduction

1.1 Introducing the Problem

Speech enhancement is concerned with the ability to remove noise present in a speech signal, thus improving the quality and/or intelligibility of the speech. A number of practical applications for speech enhancement exist, with one common example being hearing aids or cochlear implants (CIs) for hearing impaired individuals. Additionally, with the advent of so-called 'virtual assistants' such as Apple's *Siri* and Amazon's *Alexa*, the intended recipient of a speech signal or voice command is not necessarily a human. The realisation of effective speech enhancement for real-world applications is an ongoing research pursuit made challenging by the variability of the types of noise and speech encountered, and their respective properties. A number of conventional signal processing methods have been developed for application to the problem of speech enhancement. Advances in computational power and the emergence of specialised hardware have enabled the development and application of deep learning methods to the problem. This work focuses on the implementation of two deep learning architectures – a convolutional autoencoder network and a long short-term memory (LSTM) recurrent neural network – to investigate the suitability of these approaches for this application. This chapter will provide a brief overview of the work contained within this thesis, the objectives of the work and an outline of the structure of the remainder of the thesis.

1.2 Project Aims and Objectives

The overall aim of this project is to investigate speech enhancement and how deep learning methods may be used to effectively remove noise from recorded speech signals, while maintaining quality and intelligibility of the reconstructed speech. The work focuses on a range of noise conditions, including the particularly challenging cases of negative signal-to-noise ratios and speech-related noise types such as multi-talker babble and cafeteria noise.

In summary, the objectives of the project are to:

- Conduct an in-depth review of the relevant background theory and literature related to speech enhancement and deep learning methods of interest.
- Identify and obtain data appropriate for deep learning model implementation.
- Research selected neural network architectures, as well as associated advantages and limitations.
- Research deep learning models applied to speech enhancement in the literature.
- Design, implement and test a testbed system to develop deep learning models and evaluate their performance.
- Provide critical analysis of results and compare results with those reported in literature.

1.3 Contribution of Project

This project contributes to the ongoing research aim of investigating deep learning architectures and their application to speech enhancement in challenging noise conditions such as negative SNRs, babble and other speech-related noise. This was achieved as a result of the development of a testbed system that enabled the training and evaluation of a number of deep learning models.

1.4 Structure of Report

Chapter 1 provides an introduction to the project and its objectives. Chapter 2 provides the necessary background information to form a basis for subsequent discussion. This includes an overview of relevant deep learning methods, their advantages and disadvantages and their application to speech enhancement. A review of relevant literature is presented, along with commentary on the reported results. Chapter 3 presents the specific details of the methodology, as well as the structure and variants of the deep learning models implemented. Additionally, the experiment design is discussed and the hypotheses are formulated. Chapter 4 presents the experimental results obtained and includes discussion on how these results relate to the hypotheses presented in chapter 3. A critical evaluation of the results is presented, with discussion focused on explanations for the observations made. A number of challenges and alternative approaches are also presented and discussed in chapter 4. Finally, in chapter 5, the conclusions of the project are summarised and possible future work and improvements are discussed.

Chapter 2

Background and Related Work

This chapter will present the relevant background information and theory which will form the basis of later discussion. Firstly, in section 2.1, the problem of speech enhancement in general will be described and explored. Then in section 2.2, the necessary signal processing theory will be presented before reviewing conventional speech enhancement methods in section 2.3. Evaluation metrics used for measuring performance are discussed in section 2.4 and section 2.5 contains an overview of the training targets used in supervised learning methods. In section 2.6, an overview of deep learning will be presented, before specifically focusing on the architectures relevant to this work in sections 2.7 and 2.8. Finally, a review of the literature concerned with the application of these methods to the problem is presented.

2.1 Speech Enhancement and its Applications

Speech enhancement is concerned with the ability to remove noise present in a speech signal, thus improving the quality and/or intelligibility of the speech. A wide range of applications for speech enhancement are evident, including: hearing aids, communications and consumer technology such as mobile phones and cameras [1]. In addition to acting as a standalone component, speech enhancement may act as a pre-processing stage to reduce the error rate in applications involving human-machine interaction such as speech recognition for translation, command recognition or text-to-speech systems [2].

Noise from a variety of sources may cause a reduction in the quality and/or intelligibility of a speech signal. Noise may originate from a number of sources including traffic, household appliances, industrial environments or public spaces such as restaurants or airports. Babble is a type of noise defined as noise due to speakers in the vicinity. It is known to cause significant degradation of the intelligibility of speech [3]. Babble noise is considered the most challenging type of noise a speech enhancement system must face, due to its highly non-stationary nature and its similarity to the desired speech [4, 5]. In real-world situations, noise is rarely the result of the presence of a single noise source. Noises from many sources are generated in combination, resulting in a complex dynamic signal being perceived. The task of segregating a sound of interest, often a speech signal, from a complex auditory environment is commonly referred to as the 'cocktail party problem' [6]. Designing a speech enhancement system capable of generalising to new types of noise is challenging, given the wide range of possible sources and respective noise characteristics which may be present in a speech signal.

Additional challenges are present when the intended application imposes operational or physical constraints on the system, for example in hearing aids or CIs, where there is a requirement for low-latency operation. The processing delay must typically be below 10 - 20 ms to limit subjective discomfort and audio-visual asynchrony for the listener [7]. When considering the use of complex deep learning models with potentially millions of parameters, battery power, computational capability and physical size may become important not only for hearing aids but also for other types of embedded systems.

2.2 Speech Data Representations

Speech signals may be represented in the time domain or the frequency domain, with the discrete Fourier transform (DFT) being used to reveal information about the frequency content of a discrete signal, $x[n]$. For a signal of length N, the output of the DFT is an array of N complex values which represent the amplitude and phase of the signal's content at frequencies $\frac{2\pi k}{N}$, where $k = 0, 1, ..., N-1$, as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk} \tag{2.1}$$

The DFT assumes that the samples being analysed represent one period of a periodic signal. When the signal is not periodic, or if there is a non-integer number of periods within the analysis window, artificial discontinuities manifest themselves in the frequency domain as frequency components not present in the signal being analysed. This phenomenon is known as spectral leakage and its effects may be mitigated by using a technique called *windowing*. Windowing involves multiplying the samples in the analysis window by some function whose amplitude varies smoothly towards zero at the window edges, thus reducing the amplitude of the discontinuities. Commonly used window functions include Blackman, Hamming and Hanning, which are discussed further in [8].

The DFT provides information about the spectral components present in a signal over its entire length. As such, it offers no information about how the spectral components of non-stationary signals such as speech vary with time. The short-time Fourier transform (STFT) provides a representation of how the frequency content of the signal varies with time. It is computed by sliding an analysis window of length M samples over the signal and computing the DFT of the successively sampled sections using a fast Fourier transform (FFT) algorithm. For speech applications, a common approach is to use a window duration of 20-30 ms, with each windowed segment overlapping by 10-20 ms [9]. The STFT computation is defined as

$$\text{STFT}\left\{x\left[n\right]\right\} \equiv X\left(m,\omega\right) = \sum_{-\infty}^{\infty} x\left[n\right] w\left[n-m\right] e^{-j\omega n}, \tag{2.2}$$

where x is the time domain signal, w is the window function and m is the window hop size, in samples. The process of STFT computation is illustrated in figure 2.1.

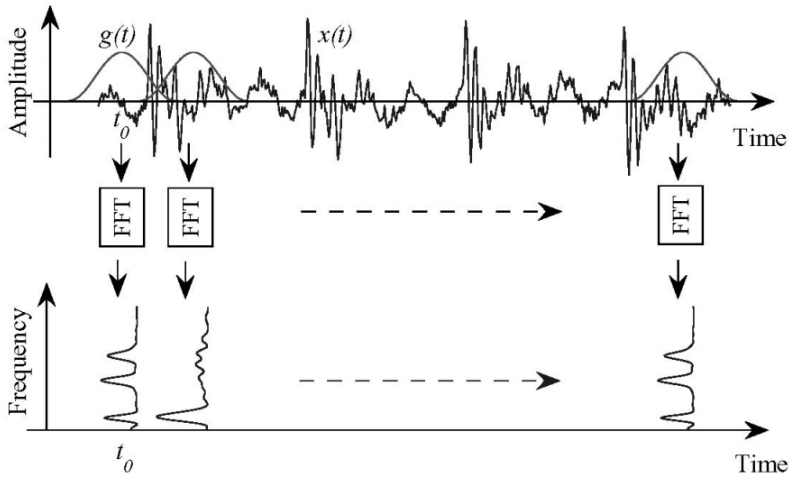


Figure 2.1: Illustration of the STFT computation [10]

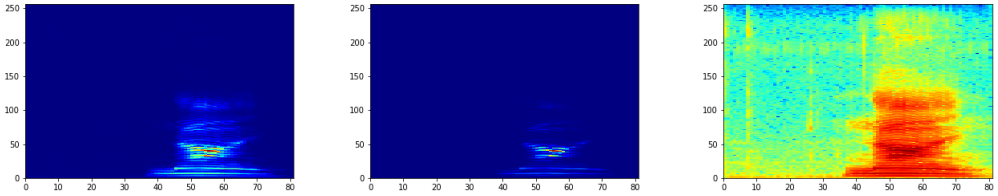
The output of the STFT is a matrix containing a complex number for each point in time and frequency. The magnitude and phase are obtained using the real and imaginary parts of the complex matrix, as follows,

$$|X| = \sqrt{Re(X)^2 + Im(X)^2} \tag{2.3}$$

$$\theta = atan\left(\frac{Im(X)}{Re(X)}\right) \tag{2.4}$$

Each of these matrices may be visualised as a time-frequency (T–F) representation such as a spectrogram. A trade-off exists between the frequency and time resolution, determined by the width of the analysis window. A narrower window gives a wideband spectrogram, with high time resolution but poor frequency resolution. A wider window gives a narrowband spectrum, with high frequency resolution but poor time resolution.

Speech enhancement algorithms proposed in the literature commonly make use of STFT magnitude, power or log-power spectrum (LPS) for the processing. Examples of these are illustrated in figure 2.2. The LPS is desirable, since it is believed to offer perceptually relevant parameters [11]. It is advantageous to minimise the distortion in a perceptually relevant domain if the objective is to improve the perceptual quality of the signal [12].



(a) Magnitude spectrum (b) Power spectrum (c) Log-power spectrum

Figure 2.2: STFT magnitude, power and log-power spectra

There are a number of other representations which are used in applications related to speech, including gammatone frequency cepstral coefficients (GFCC), relative spectral transform (RASTA) and mel-frequency cepstral coefficients (MFCC) [13]. MFCCs are commonly used for speech recognition and speaker identification applications and are discussed in more detail in [14] and [15].

2.3 Conventional Methods

Spectral subtraction is the most widely used conventional method to reduce additive noise for the purpose of speech enhancement [16]. For a speech signal contaminated with noise, an estimate of the noise spectrum may be obtained during periods of non-speech activity, which may be identified using voice activity detection (VAD) methods. A noisy signal consisting of clean speech and additive noise may be described as

$$y[n] = s[n] + d[n] \tag{2.5}$$

where $y[n]$, $s[n]$ and $d[n]$ represent the noisy, clean speech and noise signals, respectively. These signals are represented in the STFT domain as

$$Y(\omega, k) = S(\omega, k) + D(\omega, k) \tag{2.6}$$

where k represents a frame number, omitted for clarity from this point on. By assuming the speech and the interfering noise are uncorrelated, the cross terms are zero [17] and

the power spectrum is obtained by

$$|Y(\omega)|^2 = |S(\omega)|^2 + |D(\omega)|^2 \tag{2.7}$$

An estimate of the clean speech power spectrum $|\hat{S}(\omega)|^2$ is obtained by subtracting the power spectrum of the estimated noise from the power spectrum of the noisy signal, as

$$|\hat{S}(\omega)|^2 = |Y(\omega)|^2 - |\hat{D}(\omega)|^2 \tag{2.8}$$

where $|\hat{D}(\omega)|^2$ is an estimate of the noise spectrum, obtained by computing the average of recent frames without speech. In cases where the interfering noise is highly correlated with speech, such as multi-talker babble, the above assumption clearly does not hold. This results in a reduction of the performance of spectral subtraction algorithms under such conditions [18]. Further discussion on cross terms is presented in [19].

A trade-off exists between the removal of noise and the distortion of the speech signal. It is challenging to obtain a reliable estimate of the spectrum of non-stationary noise and speech degradation may result. It is possible that the resulting speech may sound distorted or have artefacts present, such as “musical noise”, which can result in significant deterioration of speech quality [20] and may adversely affect speaker recognition systems [21]. This musical noise occurs due to the subtraction of a smoothed estimate of the noise spectrum from the noisy signal, resulting in some sinusoidal energy emerging in the spectrum of the clean speech estimate [22].

In [23], a spectral subtraction approach was evaluated by examining sentence recognition with seven hearing impaired patients who had CIs fitted. The authors report significant performance improvements for all patients when considering speech-shaped noise (SSN), but state that the increase in recognition performance was modest when considering 6-talker babble noise. Though the enhancement algorithm led to good results under certain conditions, it should be noted that the number of subjects was very small and the signal-to-noise ratios (SNRs) examined ranged from 0 dB to 9 dB, thus excluding negative SNR conditions.

Wiener filtering is an alternative method that seeks to suppress the noise in the speech signal by minimising the mean squared error (MSE) between the magnitude spectrum of the noise-free signal $S(\omega)$ and the estimated signal $\hat{S}(\omega)$. The transfer function of the optimal Wiener filter may be computed as

$$H(\omega) = \frac{S_s(\omega)}{S_s(\omega) + S_n(\omega)} \tag{2.9}$$

where $S_s(\omega)$ and $S_n(\omega)$ represent the estimated power spectra of the clean signal and the noise, respectively [22]. The filtered speech signal is then computed as

$$\hat{S}(\omega) = X(\omega)H(\omega) \tag{2.10}$$

Methods based on signal processing, such as those discussed above, typically work reasonably well under relatively high SNR conditions. Under low SNR or non-stationary noise conditions however, they tend to become less effective in general [24]. Further discussion and comparison of these types of conventional algorithms for speech enhancement is presented in [17].

2.4 Evaluation Metrics

Subjective listening tests are the most accurate method for evaluation of speech quality [25]. However, such tests may be expensive and time consuming [26]. Objective evaluation measures do not always correlate well with subjective data, meaning that the results are not necessarily representative of human perception. Depending on the application, some evaluation metrics may be more appropriate than others. For applications in which the intended recipient of the enhanced signal is a human e.g. hearing aids or communications, a measure that correlates with subjective quality is desirable. This is not necessarily the case for applications such as automatic speech recognition (ASR).

The terms “quality” and “intelligibility” are not synonymous. The two are uncorrelated and independent of one another. A speech signal may have high intelligibility and low quality and vice versa [21]. Quality is highly subjective in nature and measures *how*

an utterance is produced by a speaker, with descriptions including “natural”, “hoarse” and “raspy”. Intelligibility assesses the *content* of the speech, in terms of the meaning of the words spoken [26]. A number of speech quality metrics are discussed in [25] and a review of intelligibility metrics is presented in [27].

The simplest method for evaluating noise reduction is the SNR, computed as

$$SNR = 10\ log_{10} \frac{\sum_{i=1}^{N} x^2[i]}{\sum_{i=1}^{N} (x[i] - y[i])^2}, \tag{2.11}$$

where $x[i]$ and $y[i]$ are the samples of the original and noisy (or processed) signals, indexed by i, and N is the total number of samples. As the correlation of SNR with subjective speech quality is poor, the time-domain segmental SNR (SSNR) is a common alternative [28]. The SSNR computes the mean of the SNR values within segments of the signal, as follows

$$SNR_{seg} = \frac{10}{M} \sum_{m=0}^{M-1} log_{10} \sum_{i=Nm}^{Nm+N-1} \left(\frac{\sum_{i=1}^{N} x^2[i]}{\sum_{i=1}^{N} (x[i] - y[i])^2} \right), \tag{2.12}$$

where N and M are the length of the segment and the total number of segments, respectively. Segments with SNRs above 35 dB do not reflect large perceptual differences and are replaced with a value of 35 dB. For segments that do not contain speech, where the SNR values may become very negative, a lower threshold of -10 dB is used to exclude those segments from the computation [28].

An objective measure referred to as the short-time objective intelligibility (STOI), proposed in [29] is appropriate for speech processed using a T–F weighting and shows high correlation (rho=0.95) with the intelligibility of noisy and processed noisy speech. The method, described in detail in the original paper, is designed for a sample rate of 10,000 Hz and uses a DFT-based method for time-frequency decomposition, with a Hanning window of approximately 400 ms. The measure represents an estimate of the linear correlation coefficient between the clean and processed T–F units.

The evaluation metrics used in this work were the STOI and SSNR. Other commonly used performance metrics include the signal-to-distortion ratio (SDR) and perceptual evaluation of speech quality (PESQ), described in [30]. These metrics work with time-domain representations of signals, so the processed T–F representations must be converted back to the time domain via the inverse STFT (ISTFT) by combining the phase information (either noisy or enhanced) with the enhanced magnitude.

2.5 Training Targets for Supervised Methods

There are a number of training targets defined in the literature for use with supervised learning methods for speech enhancement. One approach is to use the spectrogram of the clean speech signal as the training target, in order to directly predict it from the noisy spectrogram. Methods of this type are referred to as regression-based methods. In [11], Xu et. al used a deep neural network (DNN) to predict the LPS of clean speech from the LPS of noisy speech. They reported good performance on unseen noises and an absence of musical noise artefacts in the enhanced speech. In [31], Tan and Wang propose a convolutional recurrent neural network for real-time speech enhancement, using the magnitude spectrum of the clean speech as the training target. They report significant increases in performance, even for challenging noise types such as babble noise and cafeteria noise at negative SNRs.

Additionally, a number of mask-based methods have been proposed, including the ideal binary mask (IBM) [32], ideal ratio mask (IRM) [33] and complex ideal ratio mask (cIRM) [34]. In the case of mask-based methods, given the noisy spectrogram and the predicted mask, the estimate of the T–F representation of the target signal is obtained by multiplying the two in an element-wise fashion.

According to Eskimez et. al [35], mask-based methods have shown significant improvements over regression-based methods. In contrast to this, in [36], Fan et. al report significant performance benefit in terms of quality and intelligibility when using a regression approach to predict the LPS of target speech. It should be noted that the

dataset used in [36] was very small, consisting of only 500 utterances for training.

To introduce expressions for the ideal binary and ideal ratio masks, consider the mixture $y[n] = s[n] + d[n]$, as previously defined. Let the T–F representations of these signals be denoted $\mathbf{Y}, \mathbf{S}$ and $\mathbf{D}$, respectively. The IBM is defined as

$$\mathbf{M}_B(c,m) = \begin{cases} 1, & \text{if } |\mathbf{S}_{cm}|^2 > |\mathbf{D}_{cm}|^2 \,, \\ 0, & \text{otherwise}\,, \end{cases} \tag{2.13}$$

where $\mathbf{S}_{cm}$ and $\mathbf{D}_{cm}$ are the spectral values of $\mathbf{S}$ and $\mathbf{D}$ at a T–F unit u_{cm}, indexed by frequency c and time m, respectively. A T–F unit is assigned a value of 1 if the target signal energy exceeds the noise signal energy in that unit, so it is assumed the unit should be retained since it contains mostly the target signal. Otherwise, the unit should be removed since it contains mostly noise, and is assigned a value of 0. The prediction of the IBM frames the speech enhancement problem as a binary classification task on the level of individual T–F units. While the IBM has been shown to improve speech intelligibility, separation using the IBM typically results in the presence of musical noise in the processed signal [37].

The IRM is defined as

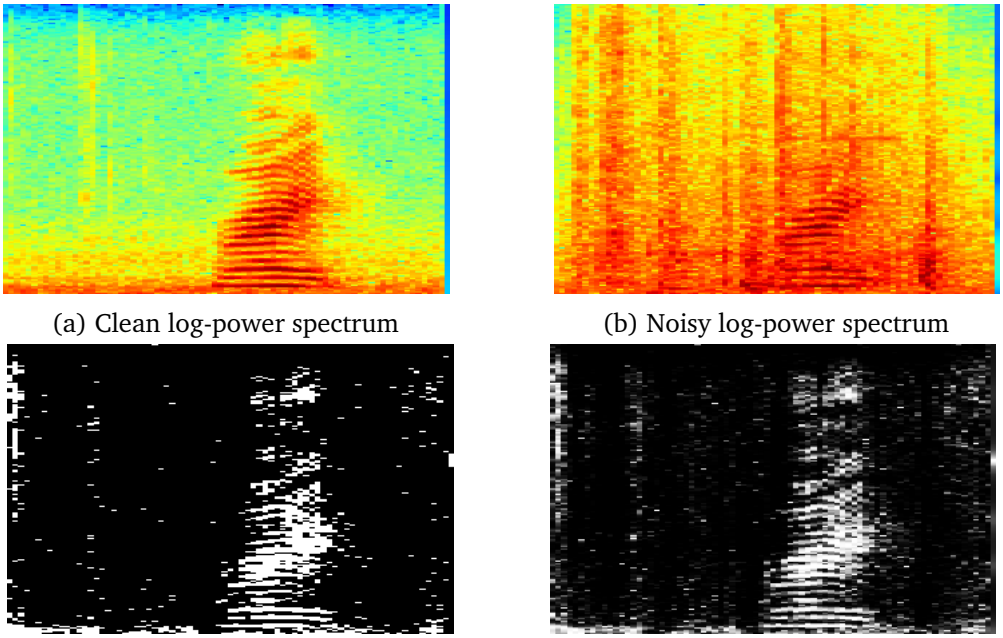
$$\mathbf{M}_R(c,m) = \frac{|\mathbf{S}_{cm}|^2}{|\mathbf{S}_{cm}|^2 + |\mathbf{D}_{cm}|^2}\,, \tag{2.14}$$

where it can be seen that the values of the IRM vary in the range $[0, 1]$, thus the IRM can be considered a smooth version of the IBM. Consider a region in the T–F representation, where the energy of the target signal and noise signal are very nearly equal i.e. the ratio computed by equation 2.14 is in the range $[0.49, 0.51]$. If the IBM was used, the T–F units corresponding to the ratio of 0.49 would be completely eliminated, leading to sharp discontinuities in the surface of the LPS. If the IRM is applied instead, the result would be a smoother surface with those T–F unit values suppressed according to the IRM.

As the objective is to predict the target using a deep learning model, consideration must be given to how well the target can be predicted by any given model. That is to say that

while the ‘ground truth’ target may result in good performance, if a model is unable to predict the target sufficiently well, errors in the prediction may result in performance degradation.

In figure 2.3a, the LPS of a clean speech utterance of the word “three” is illustrated. Photocopier noise added to the clean speech at 0 dB results in the LPS shown in figure 2.3b. Based on the clean speech signal and the isolated noise signal, the IBM and IRM may be computed according to equations 2.13 and 2.14, respectively. These two masks are shown in figures 2.3c and 2.3d.



(a) Clean log-power spectrum

(b) Noisy log-power spectrum

(c) Ideal binary mask

(d) Ideal ratio mask

Figure 2.3: A noisy-clean pair with associated IBM and IRM

Once an estimate of the target speech magnitude spectrum has been obtained by applying the mask to the noisy magnitude spectrum, this estimate must be combined with phase information in order to recover an audio signal via the ISTFT. The process is illustrated in figure 2.4.

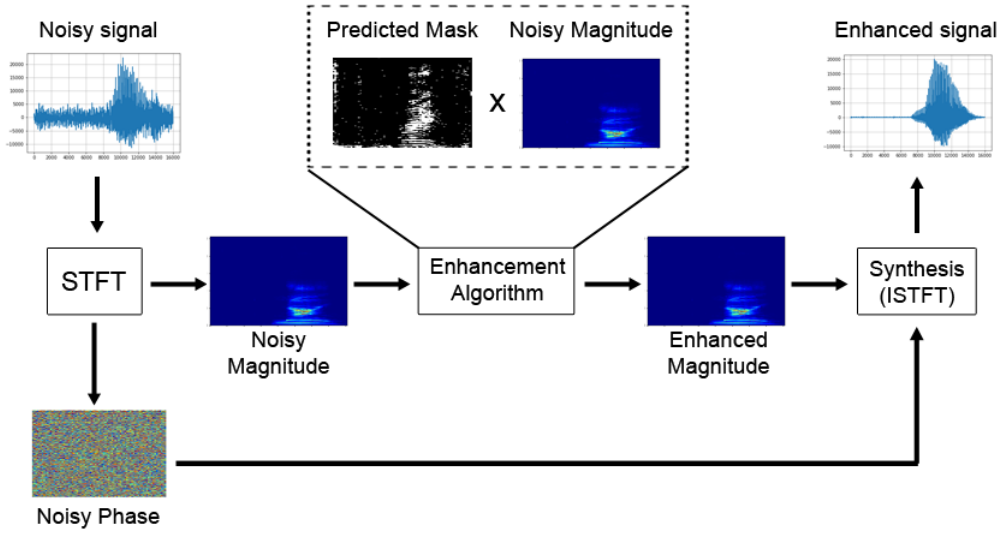


Figure 2.4: Flowchart of the speech enhancement process

Speech quality and intelligibility metrics used to assess the model performance are calculated using time domain signals, and the signals used to obtain these scores must be of equal length. In order for the output of the ISTFT to be the same length as the input (noisy) signal, the value k computed as

$$k = \frac{length(x) - N_{overlap}}{N_{window} - N_{overlap}}, \tag{2.15}$$

must be an integer value, where $length(x)$ is the length in samples of the input signal, $N_{overlap}$ is the window overlap size in samples and N_{window} is the window size in samples. The input signal x may be padded with zeros prior to the STFT operation to ensure the condition is met.

It is common for no processing to be applied to the phase information and for the noisy phase to be used to reconstruct the audio signal. In the literature there are conflicting opinions on whether or not enhancing the phase is important for achieving good results in speech enhancement. In [38], subjective tests were carried out involving nine listeners. It was reported when reconstructing a signal given an independently obtained estimate of the clean magnitude spectrum, efforts to enhance the phase from the noisy speech signal are unwarranted. Thus it was determined that the short-time

phase spectrum was not important for speech enhancement.

Conversely, in [39], the objective was to determine the effect of the phase information on speech quality. Clean speech signals were degraded with additive white Gaussian noise (AWGN) and the speech quality was evaluated considering the combination of the noisy spectrum and phase, then again considering the noisy spectrum and clean phase. The authors acknowledge that this scenario does not arise in practice, but it provides insight into the effect on the speech quality of enhancing the phase spectrum. They report that accurate phase spectrum information can improve speech quality.

While the IBM and IRM do not consider the phase information, in [34], the cIRM is proposed to perform monaural speech separation. The cIRM enhances both the magnitude and phase of a noisy speech signal, with a deep neural network being employed to predict the real and imaginary components of the cIRM. The output layer consists of two sub-layers: one for each of the real and imaginary components of the predicted cIRM. A training set of 500 utterances featuring one single speaker was used, where the noise types included babble and cafeteria noise at three SNRs. The experimental results obtained show that using the cIRM outperformed the IRM and that enhancing the magnitude and phase together was superior to doing so separately.

In [40], the authors state that using the noisy phase information for synthesis may be detrimental in low SNR conditions. Further, in [41], the authors propose that for negative SNR conditions, the use of the noisy phase as the enhanced phase cannot be justified, as the phase information of the noise is dominant in the phase of the noisy speech signal.

2.6 Deep Learning Methods

2.6.1 Deep Learning: An Overview

Deep learning is a sub-field of machine learning, concerned with artificial neural network (ANN) architectures. ANNs consist of a collection of connected units or 'nodes',

where each of these nodes computes a weighted sum of its inputs and a bias term before applying an 'activation function' to the result, giving a scalar value that forms the node's output [42]. In an ANN, these nodes may be arranged into layers. A multi-layer perceptron (MLP) is an example of an ANN that contains an input layer, an output layer and one or more intermediate layers, referred to as 'hidden' layers [43]. The term 'deep neural network' (DNN) refers to a network containing multiple hidden layers.

Deep learning has been applied to many problems, including speech recognition, object detection in images, and drug discovery [44]. One advantage deep learning methods have over conventional machine learning methods is their ability to use raw data as input and automatically extract features that are useful for the problem being solved. Conventional machine learning approaches require 'hand-crafted' features derived from the raw data, often requiring in-depth knowledge of the problem domain. Many variants of neural networks exist, including convolutional neural networks (CNN) [45], recurrent neural networks (RNN) [46] and generative adversarial networks (GAN) [47].

Figure 2.5 shows a simple feedforward neural network, where $\mathbf{x}$ is a vector of input data, $\mathbf{W}$ are weight matrices and $\mathbf{b}$ are bias vectors.

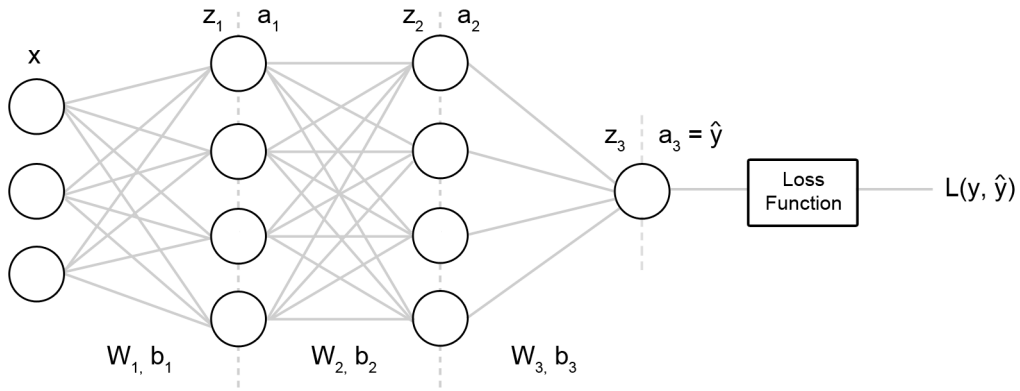


Figure 2.5: A simple feedforward neural network

In this notation, $\mathbf{z}$ and $\mathbf{a}$ are the vectors of outputs from the previous layer and the

current layer, respectively, computed as

$$\mathbf{z}_i = \mathbf{W}_i \mathbf{a}_{i-1} + \mathbf{b}_i \tag{2.16}$$

$$\mathbf{a}_i = g(\mathbf{z}_i) \tag{2.17}$$

where g is an activation function, used to introduce nonlinearity to the network. Examples of activation functions include the sigmoid, hyperbolic tangent (tanh) and rectified linear unit (ReLU), defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.18}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.19}$$

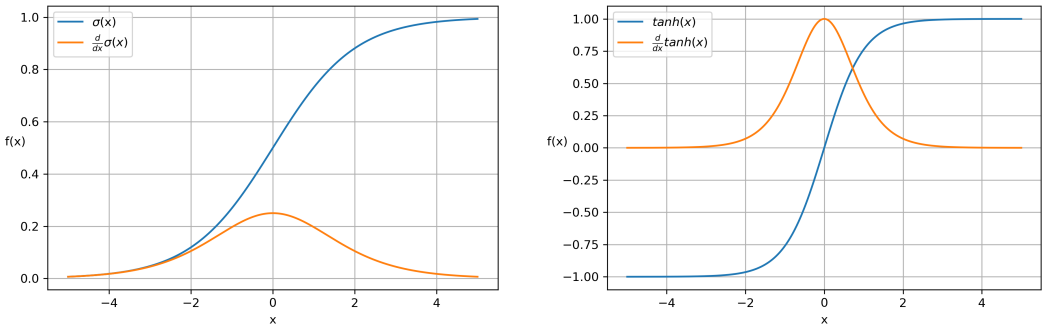
$$ReLU(x) = max(0, x) \tag{2.20}$$

In order to facilitate training in a neural network, the backpropagation algorithm is used to update the parameters of the network in order to minimise the error between the predictions of the network and the expected output resulting from an input. The training process via backpropagation may be summarised as follows:

- Provide the network with an input and propagate it forward through the network to obtain a prediction as output.
- Compare the prediction with the expected value corresponding to the given input, in order to compute a loss value using an appropriate loss function.
- Calculate the gradients i.e. the partial derivatives of the loss with respect to the network parameters.
- Adjust the network parameters using the computed gradients by taking a "step" in the negative gradient direction, where the step size is controlled by a "learning rate" hyperparameter denoted α.
- Repeat until a termination condition is satisfied.

Training using backpropagation involves the computation of the derivative of an activa-

tion function with respect to its input. For the sigmoid and tanh activation functions, figure 2.6 shows the values of these derivatives are in the range $(0, 0.25]$ and $(0, 1)$, respectively.



(a) Sigmoid function and its derivative

(b) Tanh function and its derivative

Figure 2.6: Sigmoid and tanh activation functions and their derivatives with respect to their inputs

Using the network in figure 2.5 as an example, the gradients of the weights are computed as follows, according to the backpropagation algorithm, with the bias terms omitted for brevity.

$$\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_3} \tag{2.21}$$

$$\frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial W_3} \tag{2.22}$$

$$\frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2} \tag{2.23}$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial W_2} \tag{2.24}$$

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1} \tag{2.25}$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial W_1} \tag{2.26}$$

As the gradients are propagated further back into the network, the number of multiplications by $\frac{\partial a}{\partial z}$ increases. Since $\frac{\partial a}{\partial z} < 1$ for sigmoid and tanh activation functions, for the weights near the input layer, the gradients $\frac{\partial L}{\partial W}$ may become close to zero i.e. they may

vanish. This means that the change in W due to the parameter update

$$W := W - \alpha \frac{\partial L}{\partial W} \tag{2.27}$$

may be negligible or zero, resulting in these layers not being trained. This problem is referred to as the vanishing gradient problem and may adversely affect very deep neural networks when using saturating activation functions like sigmoid and tanh. The use of the ReLU activation function helps mitigate the vanishing gradient problem, as discussed in [48]. Further discussion on the vanishing gradient problem may be found in [49].

The optimisation algorithm is responsible for determining how the parameters of a network should be updated such that the loss function is minimised. There are many optimisation algorithms available and the reader is directed to [50] for an overview.

For an excellent and accessible introduction to deep learning, neural networks, optimisation algorithms and backpropagation, the reader is directed to [44, 51, 52].

2.6.2 Deep Learning for Speech Enhancement

The objective when applying deep learning methods to the problem of speech enhancement is to train a model to be capable of generalising well enough to perform satisfactorily when exposed to previously unseen speech, noise and SNR conditions. This is achieved by training using sufficient volume and variety of speech and noise data. In the case of SNR generalisation, it is straightforward to add more SNRs to a training set, however in [40] it is reported that supervised algorithms for speech enhancement are not necessarily sensitive to the exact SNRs used in training. This is due to the fact that there may be wide range of local SNRs on the frame level, or T–F unit level, providing the appropriate conditions for generalisation.

There have been many deep learning models proposed in the literature for the application of speech enhancement, such as DNNs [53], RNNs/LSTMs [54, 55], GANs [56] and

CNNs [57, 58]. This work will focus on convolutional autoencoder and LSTM networks.

2.7 Convolutional Autoencoders

A simple autoencoder (AE) is a type of feedforward artificial neural network (ANN) that tries to learn a function in order to generate an output $\hat{x}$ that is a reconstruction of its input x. That is, the objective is to learn some function $f_{W,b}(x) \approx x$ [59]. While it may seem like a trivial task for the AE to reconstruct its inputs by learning an approximation to the identity function, constraints can be imposed to prevent the network from simply copying the inputs to the output. Applications of AEs and their variants include dimensionality reduction for data visualisation, anomaly detection and speech enhancement [60, 61].

An AE has a symmetric structure, consisting of an input layer and an output layer with an equal number of nodes, as well as one or more hidden layers, as illustrated in figure 2.7. Considering an AE with one hidden layer, a constraint may be imposed such that the number of units in the hidden layer is less than the number of units in the input layer. This means that the network is forced to learn a compressed representation of the input data (also referred to as a latent representation) from which the decoder must reconstruct the input. An example of this would be using a 20x20 image containing 400 pixel intensity values as input and having a hidden layer with 200 nodes. The decoder must then use the vector of activations from the hidden layer (the compressed representation of the input) to reconstruct the 400 pixel input [51]. An AE is trained by using backpropagation to minimise the reconstruction loss.

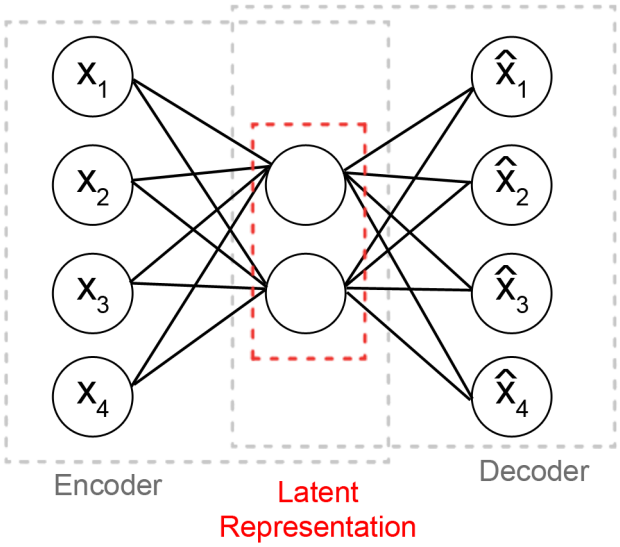


Figure 2.7: A simple autoencoder with one hidden layer

The latent representation is computed by the encoder as

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.28}$$

where f is an activation function, $\mathbf{W}$ is a weight matrix and $\mathbf{b}$ is a bias vector. The decoder uses $\mathbf{h}$ to compute the reconstruction as

$$\hat{\mathbf{x}} = g(\mathbf{W}'\mathbf{h} + \mathbf{b}') \tag{2.29}$$

where g, $\mathbf{W}'$ and $\mathbf{b}'$ are not necessarily related to their encoder counterparts. As a form of regularisation, to avoid overfitting, the weight matrices corresponding to the encoder and decoder may be 'tied', such that $\mathbf{W} = \mathbf{W}'^{\mathsf{T}}$ [61].

In [61], Lu et. al propose a deep denoising autoencoder (DAE) for speech enhancement, using the mel frequency power spectrum (MFPS) of noisy and clean speech as input and training target, respectively. A greedy layer-wise pre-training strategy was adopted to pre-train the individual AEs, which were then stacked to form the DAE. During this stage, the input to the next AE is the latent representation from the preceding hidden layer. The purpose of pre-training is to initialise the parameters to be close enough

to a good solution before the fine tuning stage which is the training of the DAE as a whole [62]. Car noise and factory noise were added to clean speech at 0 dB, 5 dB and 10 dB. The results showed that the model performed well in terms of noise reduction, speech distortion and PESQ; outperforming the minimum mean square error (MMSE) estimation method for most of the experiments conducted. The authors recognise that the use of only two noise types is a limitation in their work.

A convolutional neural network (CNN) is a particular type of deep neural network that is often applied to problems involving image data. Typical applications include image classification [45], document analysis [63] and medical applications including image segmentation [64, 65], as well as diagnosis support utilising speech data from patients [66, 67].

A convolutional layer in a CNN contains filters, which are applied to the layer's input in order to identify useful features. These filters, also referred to as *kernels*, consist of a grid of weight values, which are trainable parameters in a CNN. A convolutional layer may contain one or more filters, each used to identify different types of features. By considering an image classification application, it is apparent that the CNN should learn filter weights that allow it to identify useful features for classification and reduce the loss. Since the filter weights are learned during training, there is no requirement for hand-crafted filters designed to locate specific features in the inputs.

The input to a convolutional layer is filtered using a kernel in a convolution operation. Here, the kernel "slides" across the input and computes the weighted sum of the input pixel values to produce a scalar output at each position it slides over in its input. The output from a filter in a convolutional layer is referred to as an *activation map*. An activation map contains the response of a filter to its input, with a strong response observed when the filter finds the feature it was looking for. If a convolutional layer contains 10 filters, then the output from that layer will be 10 activation maps, which will then be passed to the next part of the network.

A convolutional layer has a number of properties and hyperparameters [68], including:

- Kernel size: Determines the size (in pixels) of the filters used in that particular convolutional layer.
- Filters: The number of filters a convolutional layer contains.
- Stride: Refers to the size of the steps taken as the filter is applied and moved over the image. For example, having a stride of (2,2) means that the filter moves two pixels to the right for every horizontal filter movement and two pixels down for every vertical filter movement. Using a stride value greater than one is a method to downsample the output activation map.
- Padding: Zeros may be added to the border of the input in order to determine how the edge pixels are filtered. The dimension of the output activation maps can also be modified using padding.

For the sake of brevity, further general discussion of CNNs is omitted, with the reader directed to [68, 69] for further reading. The following sections describe concepts which are utilised in CNNs and are directly relevant to the implementation presented in this work, specifically batch normalisation and skip connections.

2.7.1 Batch Normalisation

A phenomenon referred to as internal covariate shift describes the difficulty associated with training a deep neural network. It results from the fact that during training, the distribution of each layer's inputs changes as the parameters of the previous layers are updated. The problem is addressed in [70], by normalising layer inputs using a method called batch normalisation.

Training the network is complicated by the fact that for each layer, the parameters in all preceding layers affect the inputs to the current layer. This means that small changes in the parameters of shallower layers are amplified with increasing depth. A change in the distributions of the inputs to a layer causes the layer parameters to need to continuously adapt to the new input distribution. It is therefore advantageous for the distribution of

a layer's inputs to remain fixed over time.

For a given layer with a d-dimensional input $\mathbf{x} = (x^{(1)}, \ldots, x^{(d)})$, each dimension is normalised independently. Consider a mini batch of size m, and one particular activation $x^{(k)}$. Within the mini batch, there are m values $\{x_1, ..., x_m\}$ for this activation, where the k is omitted for clarity. These m values are normalised as

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.30}$$

where ϵ is a constant for numerical stability and μ_B and σ_B^2 are the mini batch mean and variance, respectively, calculated as

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{2.31}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{2.32}$$

Normalising layer inputs changes what the layer is capable of representing. Consider a layer with a sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$. Normalising the inputs to the sigmoid function restricts its outputs to the region where the function is approximately linear. Two additional trainable parameters, $\gamma^{(k)}$ and $\beta^{(k)}$, are introduced for each activation $x^{(k)}$, which are used to scale and shift the normalised value $\hat{x}^{(k)}$ as

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \tag{2.33}$$

The purpose of these two additional trainable parameters is to ensure that the identity transform may be represented. That is, the inclusion of these parameters allows batch normalisation to recover the original activations if that is the optimal thing to do.

As well as accelerating training, batch normalisation regularises the model, acting to improve the generalisation. The experiments conducted by Ioffe and Szegedy showed that though dropout may normally be used to reduce overfitting, for a network utilising batch normalisation, it may be removed or reduced in strength. For further information,

the reader is directed to [70] for an excellent review of the topic.

2.7.2 Skip Connections

A deep convolutional encoder-decoder network is proposed in [71], with the application of restoring images by removing noise. The network contains an encoder and a decoder, which consist of convolutional and deconvolutional layers, respectively. The network is trained to learn end-to-end mappings between images corrupted with noise and the original images.

The architecture also includes skip connections, which provide a direct connection between a convolutional layer in the encoder and the symmetric deconvolutional layer in the decoder. During the forward propagation step, a skip connection takes the activation maps produced by a convolutional layer and adds them in an element-wise fashion to the activation maps of the connected deconvolutional layer. A simplified example is illustrated in figure 2.8.

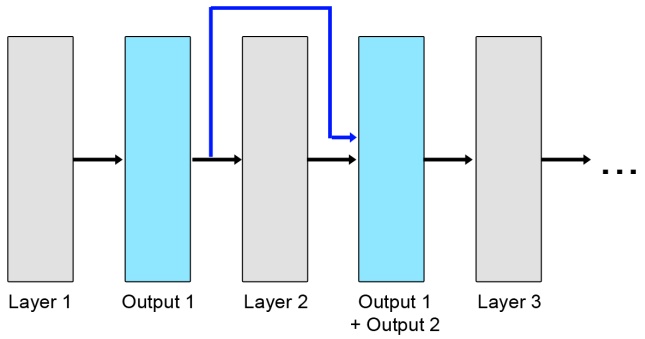


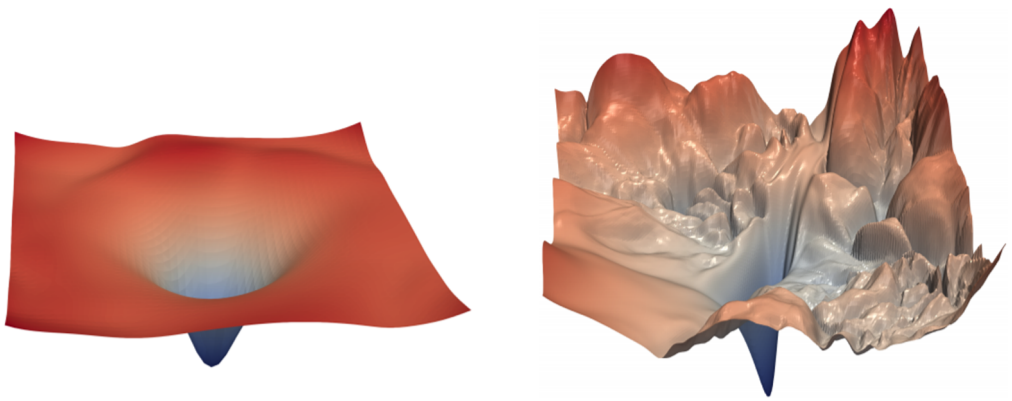
Figure 2.8: A simple illustration of a skip connection

The encoder acts as a feature extractor and encodes the relevant components of the image, while excluding the corruption from the latent representation. With increasing depth into the encoder layers, image details may be lost as a result of convolution and downsampling operations. The latent representation must include sufficient information to ensure the decoder is capable of effectively recovering these details. Skip connections allow information to be passed directly from convolutional layers to deconvolutional

layers, without passing through the intermediate layers. This is beneficial for the recovery of image details by the decoder.

The inclusion of skip connections makes deep networks easier to train, as a result of the higher quality local optima and faster convergence achieved during training. Skip connections allow gradients to bypass middle layers during backpropagation, so that information can flow directly from deeper layers to shallower layers, thus helping to address the vanishing gradient problem.

In [72], the effect of skip connections on the loss landscape for networks of varying depth is investigated. The loss landscape of a 56 layer ResNet with and without skip connections is illustrated in figure 2.9. It can be seen that without skip connections, there are many local minima in the surface. In this case, during training, the optimiser may get stuck in a local minimum, failing to find the global minimum and thus converging on a sub-optimal solution. The inclusion of skip connections results in a smoother loss surface and the absence of such local minima, as seen in figure 2.9a.



(a) With skip connections (b) Without skip connections

Figure 2.9: Loss surfaces of ResNet-56 visualised with and without skip connections [72]

The authors state that with increasing depth, the loss landscape transitions from being nearly convex to being highly chaotic, as shown in figure 2.10. This transition leads to a lack of trainability. It was observed that chaotic loss landscapes led to worse training

and testing errors than convex landscapes, with the most convex landscapes (the ones that show no noticeable chaotic behaviour) leading to the best generalisation. Skip connections prevent this transition from convex to chaotic loss landscapes, making them useful for training deep networks and improving generalisation. Figures 2.10a - c are contour plots which show the loss surfaces of three ResNets with 20, 56 and 110 layers, respectively, with skip connections included in the networks. Figures 2.10d - f show the loss surfaces visualised as contour plots for the same networks, but without skip connections.

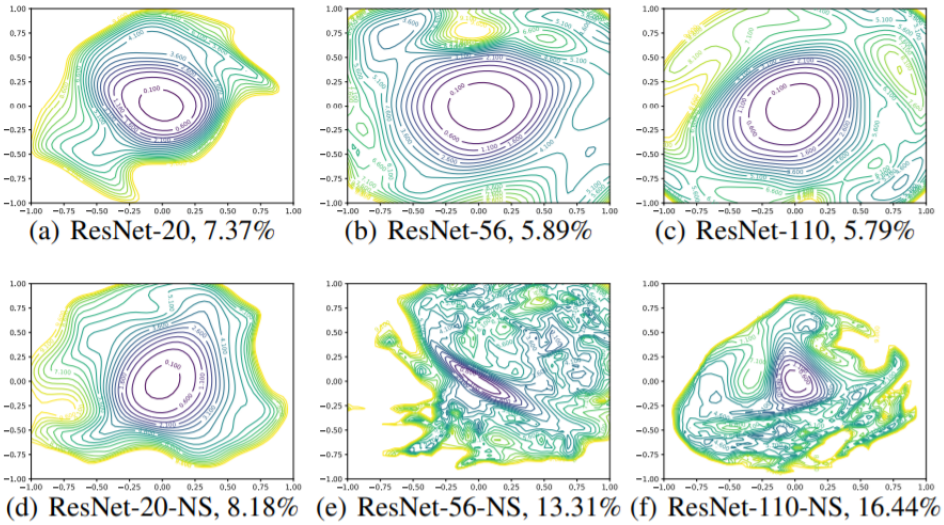


Figure 2.10: Visualisation of the loss surface of ResNets with varying numbers of layers [72]

In [3], a fully convolutional neural network was proposed for the application of speech enhancement to hearing aid devices. The noisy STFT magnitude spectrum was used as input and the network predicted the clean magnitude spectrum. The training data included babble and other noise types, mixed with clean speech at 0 dB. The application imposed the requirements that the system operate in real-time and that it must be suitable for use on an embedded device. The network uses a context window of 8 previous noisy frames to predict the current frame, thus the system is causal. The inclusion of skip connections in the network led to consistently improved results and the proposed model outperformed DNN and RNN models, while using far fewer parameters.

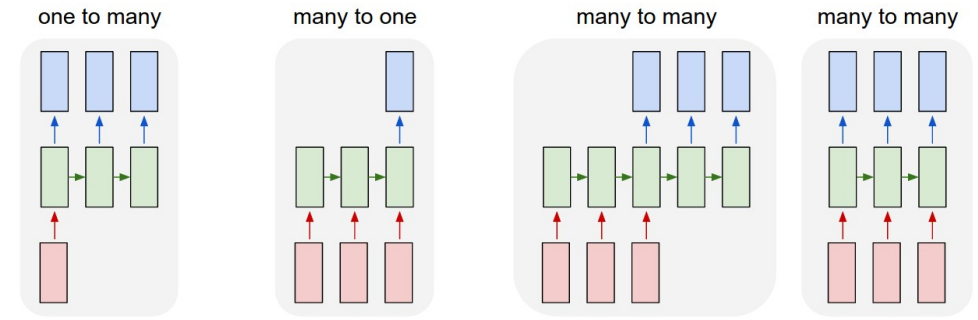
In [73], a comparison of the performance of CNN architectures with and without skip

connections was presented in the context of speech enhancement. The results showed that for both stationary and non-stationary noises, the inclusion of skip connections improved the denoising performance of the networks. This implied that the addition of skip connections may help the network to effectively learn the properties of complex noise types.

2.8 Recurrent Neural Networks

In contrast to 'regular' neural networks or convolutional neural networks, whose inputs and outputs are of a fixed size, recurrent neural networks (RNNs) are capable of operating using sequences of vectors as inputs, where these sequences can vary in length. RNNs are well suited to extracting and exploiting the long-term temporal dependencies within the input time series or sequence data.

Depending on the application, an RNN may be configured in a number of ways, as illustrated in figure 2.11 [74]. Each block in the diagram represents a vector, where red vectors are inputs, green vectors are the RNN state and blue vectors are outputs. The length of the input and output sequences are denoted T_x and T_y, respectively.



(a) One to many (b) Many to one (c) Many to many (d) Many to many (synchronised)

Figure 2.11: Examples of configurations of RNNs

Example applications for these configurations include (a) image captioning, where a variable length sequence of words is generated to describe the content of the fixed size input image, (b) sentiment analysis, where a variable-length input sequence of words

is analysed to predict a label, such as a positive or negative review, (c) translation, where both the input and output are variable length sequences and (d) video frame labelling/classification, where the input and output sequences are synchronised and $T_x = T_y$.

A 'vanilla' RNN structure is illustrated in figure 2.12, where it can be seen that at some time t, the RNN unit takes as input the vector $x^{<t>}$ and outputs $a^{<t>}$, referred to as the 'hidden state'. The RNN essentially consists of multiple copies of the same network, with each copy passing information to the copy one step forward in the sequence. As a result, the information received by a unit at any given time t consists of the input vector $x^{<t>}$ and the hidden state from the previous time step, $a^{<t-1>}$, while the output from a unit is the hidden state $a^{<t>}$. At time $t = 0$, the hidden state vector passed to the first RNN unit is initialised with either zeros or random values.

The hidden state acts as a form of memory, influenced by previous inputs, going back to the beginning of the sequence of input data. In this way, the hidden state provides some contextual information about preceding inputs in order to inform the computation at the current time step. By using an alternative visualisation to represent the loop shown in the network in figure 2.12, the application of this type of network to sequences becomes explicitly clear.

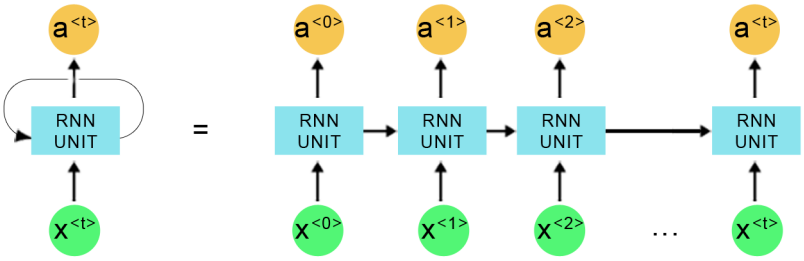


Figure 2.12: Unrolling a vanilla RNN structure

The parameters of an RNN are shared by all recurrent units in the network and they consist of 3 weight matrices, W_{aa}, W_{ax} and W_{ya}, as well as bias vectors b_a and b_y. In the weight matrix, the first subscript indicates the type of value being computed and the

second subscript indicates the type of value the weight matrix is multiplied with. For example, the weight matrix W_{ax} is multiplied by some x-like quantity (input vector) to compute an a-like quantity (hidden state).

The hidden state $a^{<t>}$ at time t is computed as per equation 2.34, where b_a is a bias term and f represents an activation function such as tanh. The resulting hidden state, $a^{<t>}$, is used to compute the predicted value $\hat{y}^{<t>}$, as

$$a^{<t>} = f\left(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a\right) \tag{2.34}$$

$$\hat{y}^{<t>} = g\left(W_{ya}a^{<t>} + b_y\right) \tag{2.35}$$

where g is an activation function.

Figure 2.13 illustrates the computations that are performed inside an individual RNN unit, where the block labelled 'A' represents some further processing to generate the output vector $\hat{y}^{<t>}$ from the hidden state.

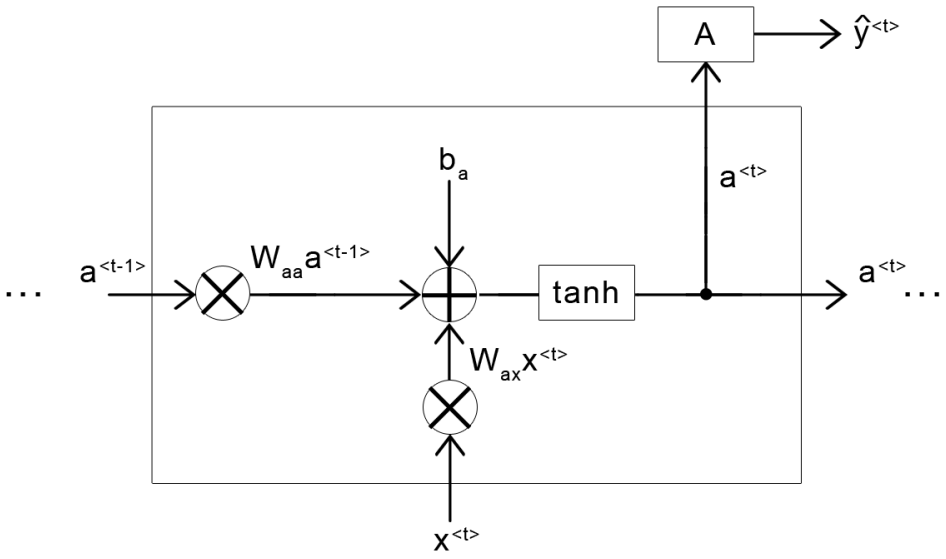


Figure 2.13: A vanilla RNN unit with computations shown

It is known that for feed-forward ANNs, increasing the depth of the model can increase the expressive ability of the model. RNNs are not limited to the single-layer architecture

illustrated in figure 2.12. Deep RNNs may be constructed by stacking RNN layers such that the hidden state from shallower layers is passed as input to the deeper layers. Figure 2.14 shows this with an illustration of a deep RNN with three recurrent layers.

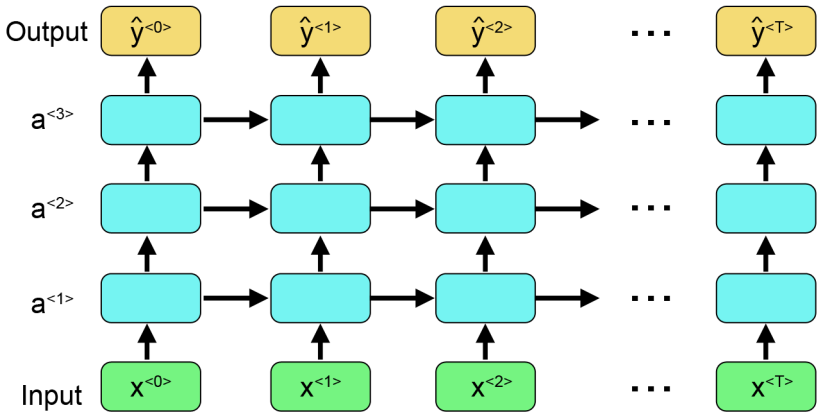


Figure 2.14: Illustration of a deep RNN

2.8.1 Backpropagation Through Time

In order to train an RNN, a variation of the backpropagation algorithm is used, called backpropagation through time (BPTT). By unrolling the structure of a synchronised many-to-many RNN, we see that each time step has one input $x^{<t>}$, one copy of the network parameters required to compute the hidden state, and one output $y^{<t>}$. It becomes conceptually similar to a deep feedforward neural network, where each time step is a layer. At each time step, the loss $L^{<t>}$ values are computed and these are summed to give the overall loss

$$L = \sum_{t=1}^{T} L^{<t>} \tag{2.36}$$

It can be seen in figure 2.15 that the prediction (and therefore the loss) at each time

step t depends on the hidden state at all previous time steps $1, ..., t$.

$$
\begin{aligned}
a^{<t>} &= f\left(W_{ax}x^{<t>} + W_{aa}a^{<t-1>} + b_a\right) \\
a^{<t-1>} &= f\left(W_{ax}x^{<t-1>} + W_{aa}a^{<t-2>} + b_a\right) \\
&\dots \\
a^{<2>} &= f\left(W_{ax}x^{<2>} + W_{aa}a^{<1>} + b_a\right) \\
a^{<1>} &= f\left(W_{ax}x^{<1>} + W_{aa}a^{<0>} + b_a\right)
\end{aligned}
$$

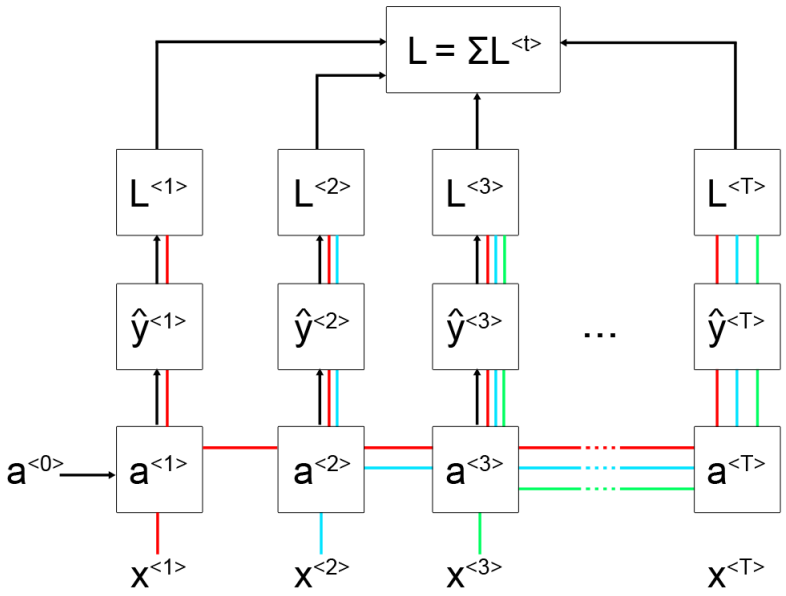


Figure 2.15: Dependence of loss $L^{<t>}$ on hidden states from all previous time steps

In order to compute the gradient at $t = 3$, the chain rule yields

$$
\frac{\partial L^{<3>}}{\partial W} = \frac{\partial L^{<3>}}{\partial \hat{y}^{<3>}} \frac{\partial \hat{y}^{<3>}}{\partial a^{<3>}} \frac{\partial a^{<3>}}{\partial W}, \tag{2.37}
$$

but the hidden state $a^{<3>}$ is a function of the previous hidden state $a^{<2>}$ and W, and so on. This results in the gradient at $t = T$ being computed as

$$
\frac{\partial L^{<T>}}{\partial W} = \sum_{k=1}^{T} \frac{\partial L^{<T>}}{\partial \hat{y}^{<T>}} \frac{\partial \hat{y}^{<T>}}{\partial a^{<T>}} \frac{\partial a^{<T>}}{\partial a^{<k>}} \frac{\partial a^{<k>}}{\partial W} \tag{2.38}
$$

where

$$\frac{\partial a^{<t>}}{\partial a^{<k>}} = \prod_{j=k+1}^{t} \frac{\partial a^{<j>}}{\partial a^{<j-1>}} \tag{2.39}$$

It can be seen that vanilla RNNs suffer from the vanishing gradient problem, as described in section 2.6. This means the vanilla RNN is capable of learning short-term dependencies, but not long-term dependencies [75].

2.8.2 Bidirectional Recurrent Neural Networks

Based on the discussion on RNNs presented so far, it can be seen that a prediction made at a given time step is informed only by past information. This means that there is no context about future events which may be relevant to inform the prediction made at the present time step.

This issue is effectively illustrated by considering applications related to language such as sentiment analysis, where the objective may be to classify a sentence as having a positive or negative sentiment. Consider the statement "He is my best friend, or so I thought." An examination of only the first half of this statement ("He is my best friend"), would lead to the conclusion that the sentence has a positive sentiment. With the inclusion of the second half, it is clear that the sentence in fact has a negative sentiment. However, this could not have been determined from the first half alone. The ability for the network to make use of all available information in a sequence to inform the prediction at a given time step, rather than only past information, is clearly desirable.

The bidirectional recurrent neural network (BRNN), proposed by Schuster and Paliwal in [76], overcomes the limitation of the regular RNN by using two separate networks, which are trained in parallel in both the positive and negative time directions. The structure of a BRNN is illustrated in figure 2.16, where it can be seen that there are two hidden states within each BRNN cell – one for the forward direction and one for the backward direction. The outputs from the forward hidden states are not connected to the input for the backward hidden state and vice versa. In [76], the authors used the TIMIT speech database for the classification of phonemes, observing the BRNN

achieved the best performance of the models evaluated, outperforming a number of MLP networks and RNN variants.

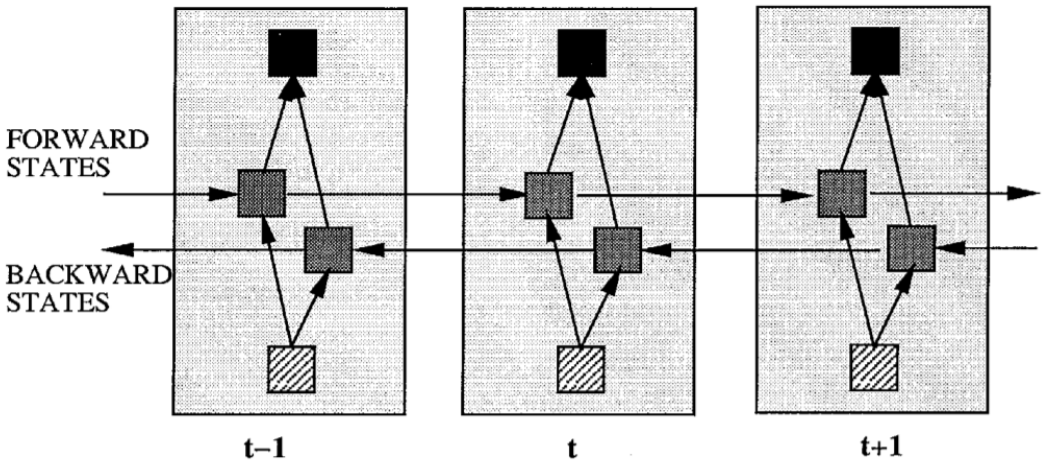


Figure 2.16: Structure of the BRNN, shown unfolded for three time steps [76]

One limitation of BRNNs is the requirement for the full sequence of data to be available before a prediction can be made. This clearly excludes such an architecture from being used for any application that requires real-time operation, such as speech enhancement in hearing aids.

2.8.3 Long Short-Term Memory Networks

The long short-term memory (LSTM) network was proposed as a solution to the vanishing gradient problem [77]. As with the vanilla RNN, each LSTM unit receives input $x^{<t>}$ and hidden state $a^{<t-1>}$, but the LSTM units also include a 'cell state', $C^{<t>}$. This cell state allows information to be retained for long periods of time, making LSTMs capable of learning longer term dependencies than vanilla RNNs.

The cell state contains an encoded representation of the relevant data from the inputs from all previous time steps. Information added or removed from the cell state $C^{<t>}$ is regulated by simple linear interactions with information provided by structures in the LSTM unit called gates; namely the forget, input and output gates. The LSTM resolves

the vanishing gradient problem by the inclusion of the cell state and the fact that $C^{<t>}$ is computed from $C^{<t-1>}$ with an addition operation rather than a matrix-vector product followed by an activation function [75]

A representation of the gates and connections inside an LSTM cell is shown in figure 2.17, where 'S' and 'T' represent the sigmoid and tanh activation functions, respectively. Each of the gates is actually a small fully connected network with its own weights, bias and activation function, where the input to each of the gates is the concatenation of $a^{<t-1>}$ and $x^{<t>}$. The number of hidden nodes in an LSTM cell is a hyperparameter that determines the shape of the hidden state and the cell state. Figure 2.17 aids with understanding the dimensions of the weight matrices and gate outputs by showing a simple example where the hidden state consists of 2 values and the input vector has 3 values.

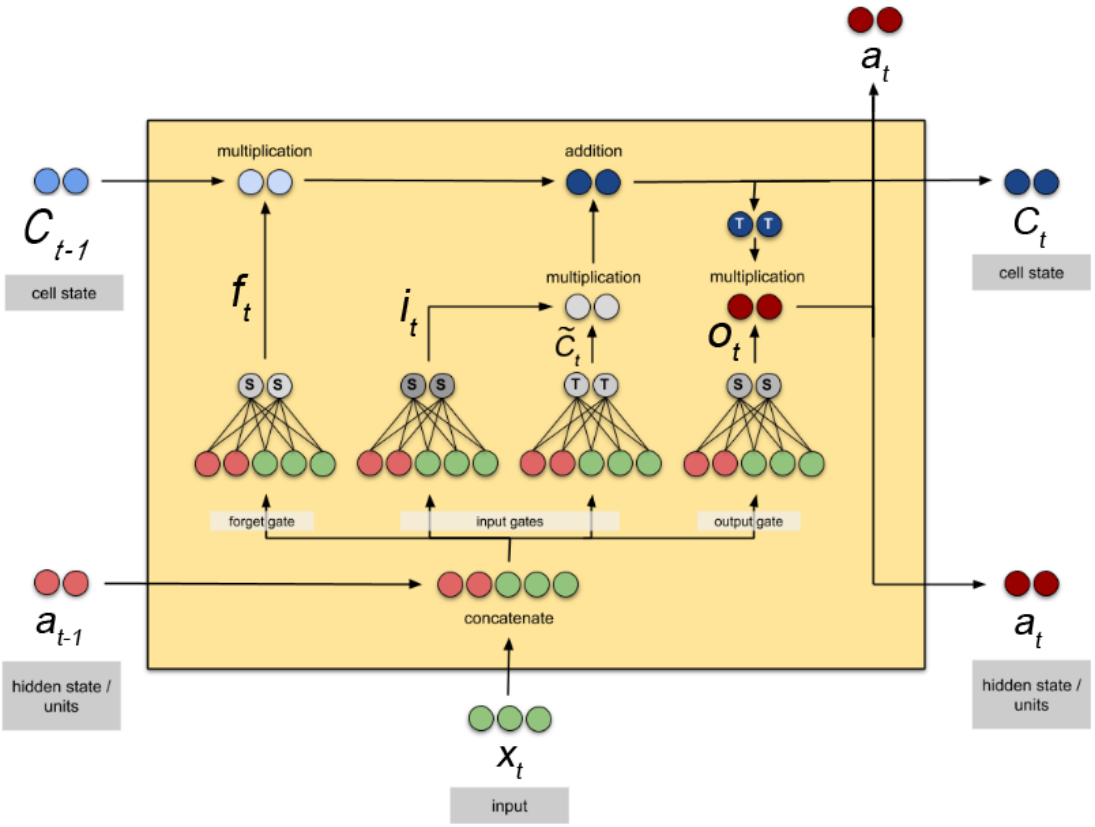


Figure 2.17: An LSTM cell with gates shown explicitly. Adapted from [78]

Equations 2.40 to 2.45 describe the computation of the outputs from each of the gates and how these are used to update the cell state and hidden state, which are passed on to the next LSTM unit

$$f^{<t>} = \sigma(W_f \cdot \left[a^{<t-1>}, x^{<t>}\right] + b_f) \tag{2.40}$$

$$i^{<t>} = \sigma\left(W_i \cdot \left[a^{<t-1>}, x^{<t>}\right] + b_i\right) \tag{2.41}$$

$$\tilde{C}^{<t>} = \tanh\left(W_C \cdot \left[a^{<t-1>}, x^{<t>}\right] + b_C\right) \tag{2.42}$$

$$C^{<t>} = \left(f^{<t>} * C^{<t-1>}\right) + \left(i^{<t>} * \tilde{C}^{<t>}\right) \tag{2.43}$$

$$o^{<t>} = \sigma\left(W_o \left[a^{<t-1>}, x^{<t>}\right] + b_o\right) \tag{2.44}$$

$$h^{<t>} = o^{<t>} * \tanh\left(C^{<t>}\right) \tag{2.45}$$

where $*$ represents element-wise multiplication.

The forget gate determines which information should be retained or discarded from the cell state as it is updated, prior to the next time step. The forget gate parameters (the weight matrix W_f, and bias b_f) are updated during training as the network learns which information is important and should be stored in the cell state. The forget gate uses a sigmoid activation to map its outputs to the range $(0, 1)$, before multiplying these outputs with $C^{<t-1>}$ to determine the extent to which existing information in the cell state is retained or forgotten.

The input gate determines which of the values within the cell state should be updated (i_t) and computes a vector of candidate values $\tilde{C}^{<t>}$, consisting of values which may be used to update the cell state. These vectors are combined by element-wise multiplication, and the result is added to the cell state, which is then passed on to the next LSTM unit.

The final step of computing the new hidden state is performed by the output gate. The vector o_t is used to determine which parts of the cell state should constitute the new hidden state, which is computed by applying a tanh activation to the cell state and multiplying with o_t. This new hidden state h_t is the output from the current LSTM unit

and it is also passed on to the next LSTM unit.

In [31], a convolutional recurrent neural network (CRNN) was proposed. The network consisted of an encoder-decoder structure with 5 convolutional and deconvolutional layers and the bottleneck consisted of 2 LSTM layers. This combination takes advantage of the feature extraction capabilities of CNNs and the temporal modelling capabilities of LSTMs. As with [3], real-time operation dictated the use of causal convolutions, where the output does not depend on future inputs. Babble and cafeteria noise were used in the training set, which consisted of 500 hours of audio and the magnitude spectrum was used as the training target. Two LSTM models were developed as baselines, each containing 4 LSTM layers with 1024 hidden nodes, with one model utilising a context window of 11 frames. The CRNN outperformed the LSTM models, using a significantly smaller number of parameters. All models tested provided significant improvement over the unprocessed baseline, however the use of the context window did not improve the performance of the LSTM models. In contrast to this, [79] reported increased performance with a DNN model when a larger context window was used.

Goehring et. al [7], motivated by having a real-time-feasible system with a low complexity model, developed an RNN for use with CI devices. Acoustic features to be used as input to the model were extracted using a 64-channel FFT-based gammatone filterbank and the training target was the IRM. Two training sets were used, containing babble noise and traffic noise, each with approximately 8000 instances. The model consisted of 2 LSTM layers, each with 128 hidden nodes, followed by a fully connected sigmoidal layer with 64 nodes. A context window of 5 frames was used to define the input feature vector. The results showed significant improvements in intelligibility of speech in babble noise, but not in traffic noise. In listening experiments, CI users reported significant improvement with both babble and traffic noise.

In [35], speech enhancement was investigated for the purpose of speaker verification. Two networks are proposed - one bidirectional LSTM (BLSTM) and one CED. For comparison, two baseline models were used. One was the network proposed in [3] and

the other shared the structure of the BLSTM but contained regular RNN layers. The log-amplitude spectrum was used as the input, with the amplitude soft mask being the training target. The BLSTM model contained 4 BLSTM layers each with 1024 hidden nodes and used a context window of $2c + 1$ frames either side of the current frame and predicted the mask at the current frame. Using future information in this way means this model is not suitable for real-time applications, but that is not a requirement for speaker verification. The CED network took 100 noisy frames as input at a time and predicted the corresponding 100 frames of the mask. The proposed networks provided benefit in terms of STOI, PESQ and verification error rate, with babble and cafeteria noise resulting in the worst performance.

Chapter 3

Design of Testbed System

In this chapter, the design of the testbed system will be discussed. Section 3.1 provides details of the data used in this work, outlining the types of noise data obtained, as well as the process of synthesising the required clean, noisy and noise audio clips. Section 3.2 outlines the data pre-processing steps that were required to generate appropriate model inputs and training targets from the raw audio data. Associated resource limitations and design decisions are also discussed in this section. The proposed convolutional autoencoder (CAE) model is discussed in section 3.3, with the LSTM model considered in section 3.4. Details of the hypotheses formulated are discussed in section 3.5, based on the specific details of the proposed models, as well as findings from the literature. The experiments carried out to test these hypotheses are discussed in section 3.6, along with some implementation details concerned with the evaluation of the performance of the proposed models. The reader is directed to appendix A for flowcharts that illustrate the training and testing procedures.

In this work, the programming language used for implementation was Python, with the deep learning models implemented using TensorFlow via the Keras API.

3.1 Datasets

This section will discuss the identification of appropriate data, the procedure for the creation of the datasets and the reasoning behind the choice of data sources used and

associated design decisions.

The Microsoft Scalable Noisy Speech Dataset (MS-SNSD) [80] contains a large number of clean speech files obtained from the Edinburgh 56 speaker dataset [81] and the Graz University's clean speech dataset [82] designated for training and testing, respectively. The MS-SNSD dataset also contains a number of noise clips, sourced from *Freesound.org* and the DEMAND database [83] designated for training and testing, respectively. The noise types include air conditioner, closing door, multi-talker babble, neighbour speaking, photocopier, squeaky chair, typing, vacuum cleaner and washing machine, among others. The noise clips provided for training and testing are different, but are of similar categories of noises. All of the clips are single channel (mono) wav files with a sample rate of 16 kHz.

Included with the MS-SNSD dataset is a collection of Python scripts which were used to synthesise the required audio clips. When directories containing clean speech files and noise files are provided, clean speech clips are randomly selected and concatenated to satisfy the user-defined minimum audio clip length. Noise clips are randomly selected and concatenated before being added to the clean speech, with SNR levels specified by the user. The resulting files are normalised to -25 dBFS. The final output consists of three folders containing clean speech, noisy speech and the corresponding noise clips. The process is illustrated in figure 3.1, for N clean speech clips and three SNR levels.

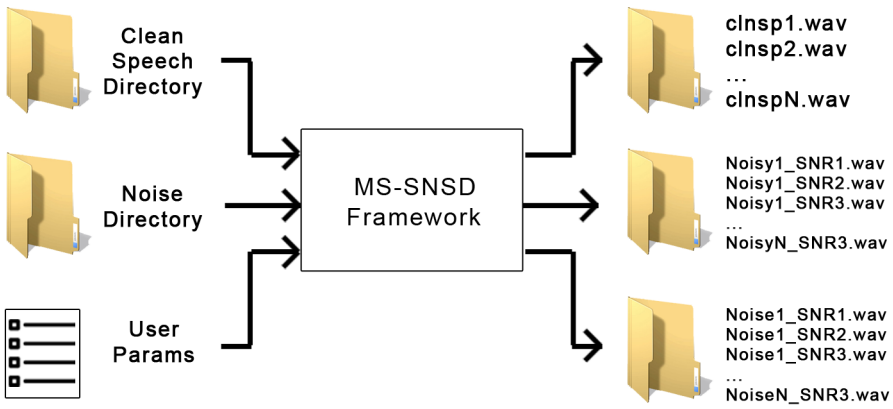


Figure 3.1: Illustration of the dataset synthesis procedure

The ability to configure these scripts makes this resource highly versatile and capable of generating arbitrarily large datasets. A number of parameters can be specified, including the minimum length of the synthesised clips, the total number of hours of audio to generate, as well as the SNR levels.

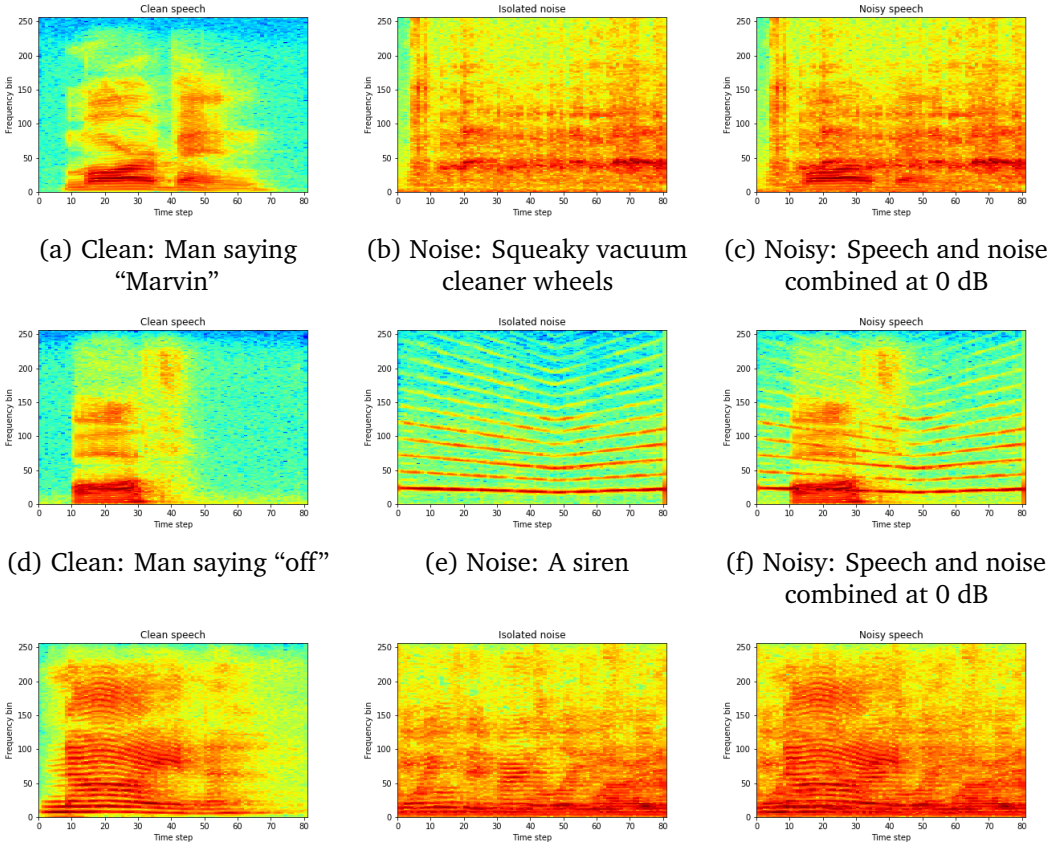
The duration of the clean utterances in the MS-SNSD dataset varied between 1 and 16 seconds. In this work, to remove the challenges associated with using such variably-sized data, clean speech data from an alternative source were used. The Speech Commands Dataset [84], originally designed for keyword spotting in speech recognition systems, contains 65,000 one-second long utterances of 30 different words, featuring thousands of speakers. The dataset also contains lists of file names specifying which utterances are to be used for training, validation and testing purposes, thus removing the requirement for the development of a strategy to split the dataset.

For the noise data used in this work, as well as the noise clips from the MS-SNSD dataset, additional clips were sourced from *Freesound.org*. This was done to increase the range of noises seen during training in order to try to aid model generalisation. Additionally, increasing the variety of noises allowed three levels of difficulty to be established. The first level consisted of relatively stationary and 'simple' noises from various electrical appliances. The second level consisted of non-stationary noises, typically containing a broader range of frequencies than the noises in the first level. The final level consisted of the most challenging noise types, with the clips containing speech from one or more other people, as well as babble noise. These noise types are summarised below in table 3.1.

Level	Noise Types
Easy	Air conditioner, copy machine, vacuum cleaner, washer/dryer
Intermediate	Applause, birds, car alarm, church bell, dogs barking, fireworks, munching (eating), shutting door, siren, squeaky chair, traffic, typing
Hard	Babble, cafeteria, neighbour speaking, party, restaurant

Table 3.1: Noise types associated with three levels of difficulty

Figure 3.2 illustrates example log-power spectra for clean speech, isolated noise and noisy speech. Each row corresponds to a noise from a different difficulty level, with the top row featuring vacuum cleaner noise, the middle row featuring siren noise and the bottom row featuring babble noise. In each case, the SNR of the noisy speech signal is 0 dB.



(a) Clean: Man saying "Marvin"

(b) Noise: Squeaky vacuum cleaner wheels

(c) Noisy: Speech and noise combined at 0 dB

(d) Clean: Man saying "off"

(e) Noise: A siren

(f) Noisy: Speech and noise combined at 0 dB

(g) Clean: Woman saying "Nine"

(h) Noise: Babble noise

(i) Noisy: Speech and noise combined at 0 dB

Figure 3.2: Examples of LPS featuring noises from different difficulty categories. Top row: easy, middle row: intermediate, bottom row: hard

For training, approximately 7,300 clean speech clips were randomly selected from the available 65,000 and mixed with noise clips at five SNRs (-6.0 dB, -3.0 dB, 0.0 dB, 3.0 dB and 6.0 dB). This range of SNR values was selected because it was observed that in the literature, some authors conducted experiments focusing only on SNRs of 0 dB or greater. In this work, the investigation of low and negative SNRs was of interest due to the challenges presented by those conditions. This configuration resulted in the training set containing approximately 36,500 noisy utterances, which was equivalent to around 10 hours of audio. The validation and testing sets contained approximately 2,200 clean utterances, mixed with noise at the same SNRs, providing approximately 11,000 noisy utterances in each case. The number of instances to be used in the training, validation and testing sets was determined using an iterative process of experimentation and assessing the trade-off between model performance and training time. The noise clips used to generate the test set were not seen during training or validation.

3.2 Data Pre-Processing

During development, it emerged that the process of managing the processed data and feeding it to the Tensorflow models was nontrivial, so a number of approaches were tested before arriving at a final method. Initially, the cloud-based development environment Google Colab was used, in order to take advantage of the powerful graphics processing units (GPUs) offered. The first approach involved pre-processing the data locally, then saving the Numpy arrays containing input spectrograms and target masks to file, to be uploaded to Google Drive for use in Colab. This approach had a number of disadvantages, including prohibitively long upload time and the fact that the datasets didn't necessarily fit into memory. The primary disadvantage was that if any changes to the pre-processing parameters were required, the whole process needed to be repeated.

A generator function is a particular type of function that returns a "lazy iterator", which does not store its contents in memory. A `DataGenerator` class was implemented based on [85] and modified for use in this work. Generator functions can be used directly with Tensorflow for training and evaluating models. During training, the generator code is

run on the CPU, preparing batches to be fed to the model. In parallel to this, the GPU is used to actually perform the training. Initially, the required wav files were uploaded to Google Drive for use by the generator in Colab. However it was determined that the I/O operations associated with reading many small files from Google Drive into Colab caused a bottleneck. Because of this, training a model took approximately 40 times longer than running the same code locally.

The final solution was to run everything locally, using an Intel Core i7-9750H CPU and an NVIDIA GeForce GTX 1050 3.0 GB GPU. The use of a generator solved the memory problem, since it was only ever necessary to load a small number of files at a time. It also proved to be a more flexible approach than saving the pre-processed data to file, because changes to the training target or the pre-processing parameters could easily be implemented and applied immediately, as processed data were generated on the fly. The I/O problem associated with Colab was solved by storing files and running the code locally.

Data pre-processing steps were implemented inside the generator to apply transformations to yield batches of data which were passed directly to the model for training, as illustrated in figure 3.3. During the processing of each batch, a number of wav files were loaded, according to the *batch_size* parameter. Since the file lengths varied slightly, the maximum file length in a given batch was identified and the other files were padded with zeros to match this length, as illustrated in figure 3.4. This was to fulfil the requirement that the instances contained within a batch all have the same shape.

The spectrograms were generated using the STFT, based on the supplied parameters. Using the clean and noisy spectrograms, *batch_size* masks were generated, to be used as training targets. Finally, the noisy spectrograms were standardised at the batch level by subtracting the mean and dividing by the standard deviation of the T–F unit values present in the batch.

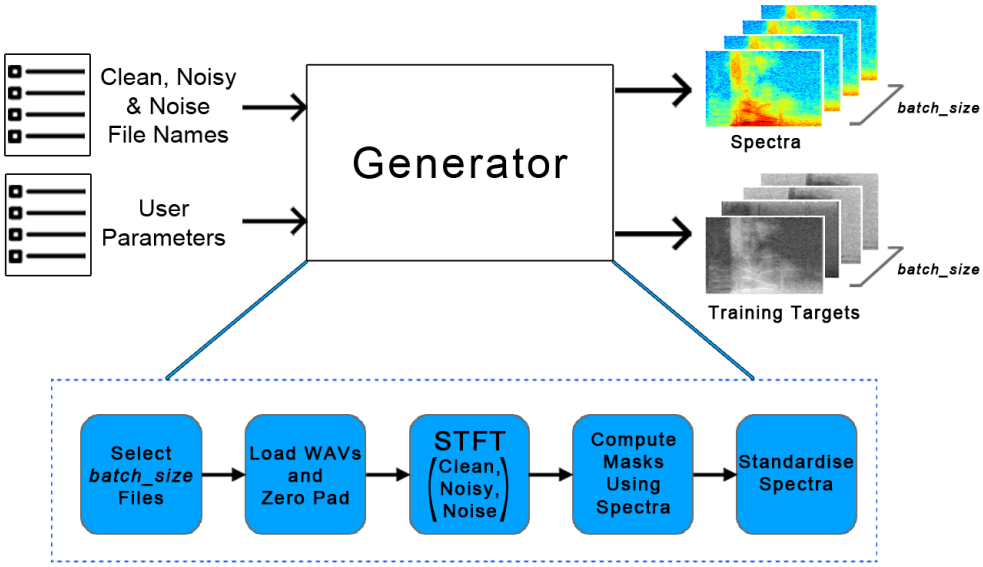


Figure 3.3: Illustration of the operation of the `DataGenerator`

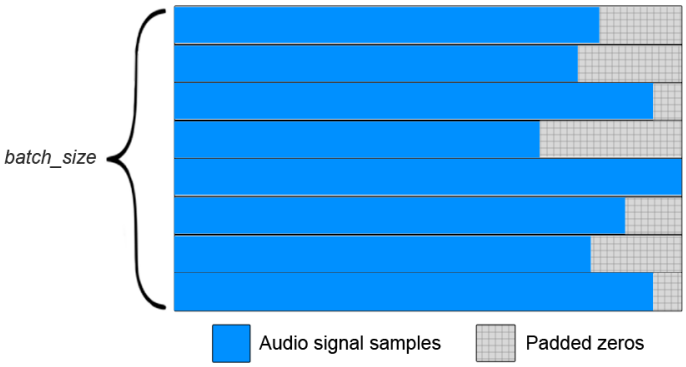


Figure 3.4: Audio clips padded with zeros to match longest clip in batch

For the STFT, a window size of 25 ms is commonly used for speech applications [86]. A Hanning window of size 400 samples (25 ms at 16,000 kHz) was selected in this work, with an overlap of 200 samples. The FFT length was set to 512 samples, since the FFT algorithm is optimised for powers of 2 [87]. As an experiment to investigate the effect on model performance of using a different window function, a model was trained with data pre-processed using this Hanning window, as well as a Hamming window of the same size. The Hamming window was selected for investigation because of the improved suppression of the first side lobe, compared to the Hanning window [88]. The differences in STOI and SSNR were found to be negligible upon comparing the results from a model trained using each type of window.

3.3 Convolutional Autoencoder

The proposed CAE network is illustrated in figure 3.5, inspired by [3, 31, 35]. In [3], a context window was used to define a range of input frames to be used to predict the target mask frame-by-frame. That meant 8 noisy STFT frames were used as input to predict one clean STFT frame. However, in this work, all noisy LPS frames are taken as input to predict the mask containing the same number of frames. Using this larger number of frames allows longer-term dependencies to be modelled. Additionally, taking this approach rather than predicting frame-by-frame was a simplification adopted for initial model implementation, requiring no modification to the generator described in section 3.2.

In the proposed model, the encoder consists of five convolutional layers and the decoder consists of five deconvolutional layers. Batch normalisation layers were added after each convolutional and deconvolutional layer, followed by a ReLU activation. In the final layer, the sigmoid activation function was used to ensure the T–F unit values in the predicted masks were in the range $(0, 1)$. For each T–F unit in the output, this value can be interpreted as the predicted probability that the unit belongs to the speech source i.e. the positive class. If the IBM is used as the training target, a threshold value of 0.5 is applied to the predicted mask T–F units, resulting in a binary mask.

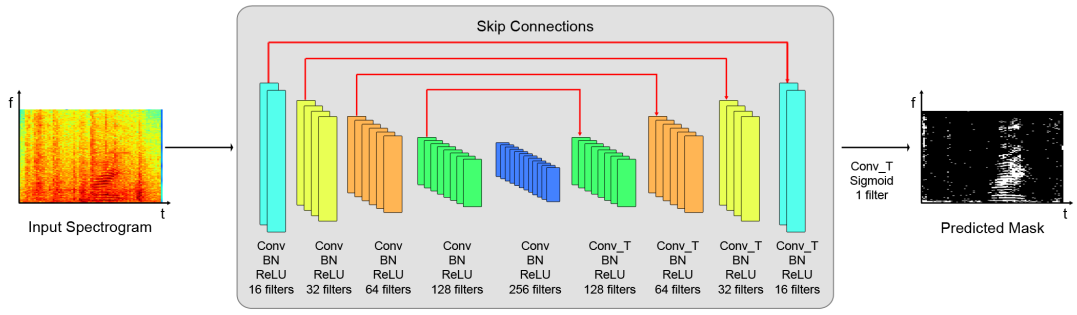


Figure 3.5: Proposed CAE network

The number of filters used in the convolutional and deconvolutional layers were M, 2M, 4M, 8M, 16M, 8M, 4M, 2M, M and 1, where $M = 16$. The final layer had one single filter, ensuring that the predicted mask had a size of 1 in the 'channels' dimension.

During the design and testing phase, the model was trained using $M = 32$. Despite the increased number of filters in the network, the observed difference in performance was negligible. To avoid unnecessarily increasing the training times, a value of $M = 16$ was deemed suitable.

The kernel size was set to (3, 2), stated in *(frequency_channel, time_step)* format, as in [35]. The activation maps are compressed in the encoder along the frequency dimension by utilising a stride value of 2 rather than a max-pooling operation. The reason behind this decision was due to the fact that an upsampling layer in the decoder would increase the dimensions of the activation map simply by duplicating the values. Alternatively, using a stride to perform downsampling allows the upsampling in the decoder to be performed by the deconvolutional layers that contain trainable parameters.

The symmetric number of filters in the convolutional layers, as well as appropriate padding ensured the activation maps shared the same shape at corresponding encoder and decoder layers. This symmetric architecture enabled skip connections to be incorporated, connecting each layer in the encoder to its corresponding layer in the decoder. The skip connections take the output from a layer in the encoder and sum this output element-wise with the input to the corresponding layer in the decoder. The inclusion of skip connections improves the flow of information and gradients through the network.

Two different training targets were considered in this work – the IBM and IRM. The prediction of the IBM is a pixel-wise binary classification task. As such, when the training target the IBM, the loss function used was binary cross-entropy loss, defined as

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\left(P(y_i)\right) + (1 - y_i) \log\left(1 - P(y_i)\right) \tag{3.1}$$

where y_i is the ground truth label for T–F unit i, and $P(y_i)$ is the value assigned to that unit in the predicted mask i.e. the probability that the T–F unit belongs to the speech source.

The values in the IRM are continuous in the range $(0, 1)$. When training using the IRM as the target, the mean squared error, defined by equation 3.2, was used as the loss function, as in [89]

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.2}$$

where $\hat{y}_i$ is the predicted value at the T–F unit indexed by i.

In all cases, the Adam optimiser [90] was used, with a learning rate of 0.005. The value for the learning rate was determined based on reviewing the literature, as well as the outcome of experimentation to observe the effects of varying this hyperparameter.

The configuration of layers and filters used led to the proposed network containing 525,665 parameters. This is significantly fewer than the networks proposed in [31] ($\sim$17.5 million) and [35] ($\sim$17.7 million). The R-CED network proposed in [3] has a greater depth than the network proposed in this work, but contained fewer filters per layer. It consisted of 15 layers, with the number of filters varying symmetrically as 10, 12, 14, 15, 19, 21, 23, 25, 23, 21, 19, 15, 14, 12, 10, 1. The CNN architectures proposed consisted of 33,000 and 100,000 parameters.

A summary of the network architecture is displayed in table 3.2. The input and output shapes are specified in (*frequency_channels, time_steps, channels*) format. The layer hyperparameters are specified in (*kernel_size, strides, filters*) format. In this case, the *time_steps* dimension has been specified as 81 for an illustrative example. In practice, it was specified as `None` to support variable input shapes.

Layer Name	Input Shape	Hyperparameters	Output Shape
conv2d_1	257, 81, 1	(3,2), (2,1), 16	128, 80, 16
conv2d_2	128, 80, 16	(3,2), (2,1), 32	63, 79, 32
conv2d_3	63, 79, 32	(3,2), (2,1), 64	31, 78, 64
conv2d_4	31, 78, 64	(3,2), (2,1), 128	15, 77, 128
conv2d_5	15, 77, 128	(3,2), (2,1), 256	7, 76, 256
Tconv2d_5	7, 76, 256	(3,2), (2,1), 128	15, 77, 128
Tconv2d_4	15, 77, 128	(3,2), (2,1), 64	31, 78, 64
Tconv2d_3	31, 78, 64	(3,2), (2,1), 32	63, 79, 32
Tconv2d_2	63, 79, 32	(3,2), (2,1), 16	128, 80, 16
Tconv2d_1	128, 80, 16	(3,2), (2,1), 1	257, 81, 1

Table 3.2: Summary of proposed CAE network architecture

3.4 LSTM

Rather than passing the noisy LPS as a two dimensional 'image' to the model, as was the case for the CAE model, the input to the LSTM model is inherently considered as a sequence. Each time step in this sequence consists of a 257-element vector of values – one value per frequency channel present in the noisy LPS. In this way, with the LSTM model, the temporal evolution of the frequency content of the noisy signal and the temporal dependencies in the data are explicitly modelled.

The proposed LSTM model, inspired by [7, 35], is illustrated in figure 3.6. It consisted of two BLSTM layers, each followed by a dropout layer and a final LSTM layer for the output layer. Each BLSTM layer contained 128 hidden nodes, and had its `return_sequences` parameter set to `True`. This meant the hidden state vector $a^{<t>}$ was returned at every time step, t, and passed as input to the next layer in the network. Having `return_sequences` set to `True` was necessary because the target to be predicted was a sequence with the same dimensions as the input. With `return_sequences` set to `False`, only the final hidden state vector $a^{<T>}$ would be returned from a BLSTM layer.

Dropout regularisation was implemented by including a `dropout` layer after each of the BLSTM layers (omitted in figure 3.6 for brevity), with the `rate` parameter set to 0.2, as in [35], in an attempt to reduce overfitting. In order to recover the mask prediction as

output from the final LSTM layer, the number of hidden nodes was set to 257 (equal to the number of frequency channels in the mask). Again, the sigmoid activation function was used in the final layer to ensure the output values were in the range $(0, 1)$.

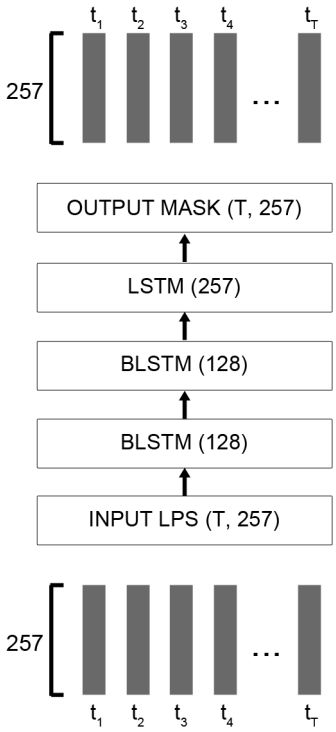


Figure 3.6: Proposed LSTM network

The dimensions of the input to the LSTM model were (*batch_size, time_steps, frequency_channels*), where the *batch_size* dimension is omitted in table 3.3 for clarity. As with the CAE network architecture summarised in table 3.2, the *time_steps* dimension has been specified as 81 as an example. In practice, this value was set to `None` to accommodate variable length input sequences.

Layer Name	Input Shape	Hyperparameters	Output Shape
bidirectional_1	81, 257	128	81, 256
dropout_1	81, 256	0.2	81, 256
bidirectional_2	81, 256	128	81, 256
dropout_2	81, 256	0.2	81, 256
LSTM_1	81, 256	257	81, 257

Table 3.3: Summary of proposed LSTM network architecture

Though this model was inspired by [35], there are some notable differences between the

two. Firstly, their network contained one more BLSTM layer, and each of their BLSTM layers contained 1024 hidden nodes rather than 128. This resulted in their network containing approximately 55 million parameters. The model proposed in this work contains approximately 1.3 million parameters. Secondly, in [35], the vector of mask values at each time step was predicted using the concatenation of neighbouring noisy frames as the input feature vector. However, in this work, only the noisy frame at time t was used as input to predict the mask at time t. This is equivalent to having a context window of size 1.

A generator function was written, with the intention of using a context window of 11 frames of the LPS spectrogram as input to the LSTM model. However it was not possible to adequately test this implementation for training models given the time available.

3.5 Hypotheses

The following hypotheses were formulated and tested in this work:

1. Using the IRM as the training target should provide better results than the IBM, both in terms of performance metrics and perceptual assessment. This is due to the IRM being a smooth version of the IBM and as such, there should be a reduction in perceptible artefacts and distortions resulting from sharp discontinuities in the enhanced T–F representation.
2. The inclusion of skip connections in the CAE network should improve the performance. This is due to the beneficial effect of skip connections on the convexity of the loss surface, leading to desirable training behaviour. Additionally, the ability to transfer information directly from the encoder to the decoder should help recover features and details that may be lost due to downsampling in the encoder.
3. The LSTM network should perform better than the CAE in highly non-stationary noise conditions. This is due to the ability of the LSTM to model long-term temporal dependencies and effectively exploit temporal context information.

3.6 Experiments

A number of experiments were conducted to test these hypotheses. Four variants of the CAE models were trained for each of the three noise categories investigated. The different training targets and presence or absence of skip connections were used to train the following model variants:

- IBM without skip connections
- IBM with skip connections
- IRM without skip connections
- IRM with skip connections.

In the case of the LSTM, the network structure remained the same for both of the model variants trained, with the training target varied to investigate the results for the IBM and IRM. In total, this resulted in 18 models being trained and evaluated.

Two Keras callbacks were defined, for use during training. The first monitored the validation loss at the end of each epoch and saved the model that had the lowest validation loss. The second was used to perform early stopping, where if the validation loss did not improve for more than 5 epochs, the training would be stopped and the best model would be restored.

The inclusion of the early stopping callback led to models typically training for 20 to 35 epochs. With the available computational resources, each epoch (36,500 instances) took approximately 3 to 4 minutes to complete. As a result, it was estimated that the total training time for the models was 30 hours.

For model evaluation, a function was defined that took lists of clean and noisy filenames as input, as well as the model to be evaluated. These lists of filenames were generated to be specific to the noise category (simple, intermediate or hard). The function returned a Python dictionary whose keys were the SNRs (-6 dB, -3 dB, 0 dB, 3 dB and

6 dB) and whose values were lists of strings containing filenames of noisy wav files at the corresponding SNRs. Using the dictionary structure provided a convenient way of passing SNR-specific filenames to subsequent evaluation functions to return the scores.

For each noise difficulty level, at each SNR examined, a baseline measure was established by computing the mean STOI and SSNR using the noisy (unprocessed) speech signal with the original clean speech as the reference signal. This was done in order to obtain a reference point for the performance scores corresponding to the signals processed by the models. This provided a determination of whether the model was beneficial or detrimental in terms of speech quality and intelligibility.

As an additional experiment, noise conditions consisting of combinations of noises were examined in an attempt to simulate conditions that are closer to those encountered in real life. Three noise types were mixed together using noise clips from the test set – specifically dogs barking, sirens and vacuum cleaner noise. This combined noise was then added to clean speech files using the same method described in section 3.1. Since the 'hard' category of noises consisted of complex, non-stationary noises consistent with real life situations, it was determined that the models trained using this category of noises would be suitable for evaluation with the combined noise types.

Chapter 4

Experimental Results and Discussion

In this chapter, the experimental results obtained will be presented and discussed. Initially, in section 4.1 the analysis will focus on the average performance of the models across all noise difficulty levels. This will provide a macro-level summary of model performance, examining how each of the models and model variants performed when considering a large variety of noise types. Then, the individual results for each of the three noise categories considered will be presented and discussed in sections 4.2 to 4.4. The final set of results, presented and discussed in section 4.5, will be the performance when the noise consists of a mixture of individual noises. In this case, the mixture is composed of dogs barking, sirens and vacuum cleaner noise, as described in section 3.6. The hypotheses stated in section 3.5 will be compared with the observations and the findings will be discussed in section 4.7.

For each of the cases described above, the log-power spectra of clean, noisy and enhanced speech will be presented for a number of interesting cases identified during the evaluation of the models. Section 4.8 presents an overview of a number of the problems and challenges faced. Finally, section 4.9 discusses a number of things that could have been done differently.

All models were evaluated on the test data and the validation data. In this chapter, for the sake of brevity, only the results from the test data evaluation are presented. The reader is directed to appendix B which contains the results from evaluation on both the test and validation sets in order to complement the commentary related to the ability of the models to generalise to unseen data. Additionally, LPS examples from the validation set are presented in appendix C.

In the following sections, for each noise category investigated, the results from the CAE and LSTM models will be compared. Further, the effect of the chosen training target, as well as the inclusion of skip connections for the CAE will be discussed. In each case a baseline score is provided, which was obtained using the unprocessed noisy utterances. For each set of results presented, the best scores are formatted in bold font.

4.1 Average of All Noise Categories

In order to provide a broad overview of model performance, the average of the results obtained for the easy, intermediate and hard noise categories was calculated. Table 4.1 shows the STOI scores obtained for all of the models and variants. As expected, the scores all increase with increasing SNR. It is seen that in terms of the STOI, the best score was obtained by doing no processing. Using the IRM as the training target rather than the IBM resulted in better model performance in all cases. When the IBM was used as the training target, the LSTM consistently scored lower than the CAEs. This observation was reversed when the IRM was used, though the improvement over the CAE was marginal.

	Metric	STOI				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.682**	**0.708**	**0.733**	**0.757**	**0.780**
	CAE (no skip connections)	0.621	0.651	0.680	0.707	0.732
	CAE (with skip connections)	0.620	0.650	0.679	0.707	0.733
	LSTM	0.596	0.620	0.645	0.668	0.688
IRM	Baseline (unprocessed)	0.682	0.708	0.733	0.757	0.780
	CAE (no skip connections)	0.663	0.690	0.716	0.740	0.763
	CAE (with skip connections)	0.662	0.690	0.716	0.741	0.764
	LSTM	0.670	0.695	0.719	0.743	0.765

Table 4.1: STOI test set results averaged for all noise categories

The results in table 4.1 are presented graphically in figure 4.1. It can be seen that the addition of skip connections to the CAEs had essentially no effect on the STOI score.

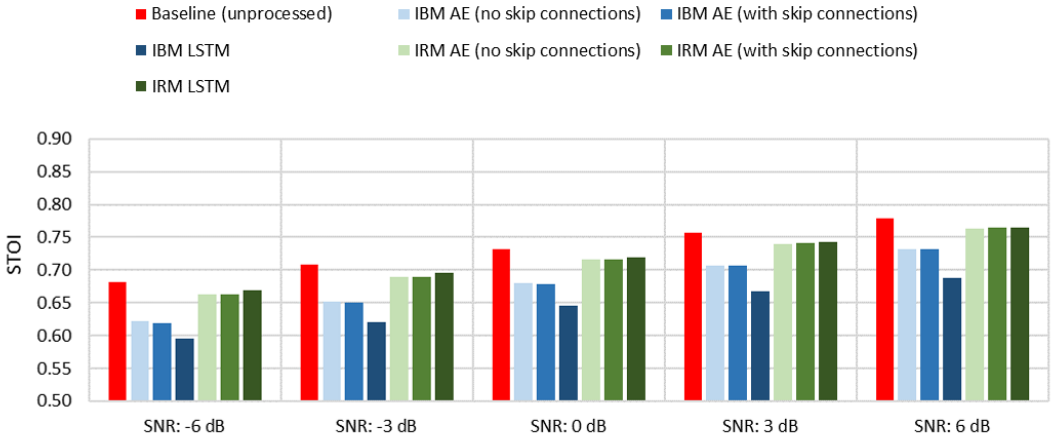


Figure 4.1: STOI test set results averaged over all noise categories

Table 4.2 shows that the results in terms of SSNR were more optimistic, with all models providing an improvement over the baseline. In this case, the best result was obtained by the CAE with skip connections, using the IRM as the training target. This offered an improvement of approximately 1.7 - 2 dB compared with the baseline.

	Metric	Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	-5.218	-4.588	-3.921	-3.218	-2.479
	CAE (no skip connections)	-3.831	-3.230	-2.606	-1.968	-1.313
	CAE (with skip connections)	-3.723	-3.128	-2.514	-1.879	-1.228
	LSTM	-4.197	-3.669	-3.134	-2.596	-2.070
IRM	Baseline (unprocessed)	-5.218	-4.588	-3.921	-3.218	-2.479
	CAE (no skip connections)	-3.362	-2.766	-2.155	-1.527	-0.888
	CAE (with skip connections)	**-3.272**	**-2.669**	**-2.050**	**-1.417**	**-0.771**
	LSTM	-4.012	-3.488	-2.966	-2.431	-1.904

Table 4.2: SSNR test set results averaged for all noise categories

The same general trends are observed for SSNR as in the STOI scores, such as the IRM offering better results than the IBM and the LSTM exhibiting consistently worse results than the CAEs. The inclusion of skip connections led to a modest improvement in both the IBM and IRM cases. It can be observed in figure 4.2 that the difference between LSTM and CAE performance becomes more pronounced at higher SNRs.

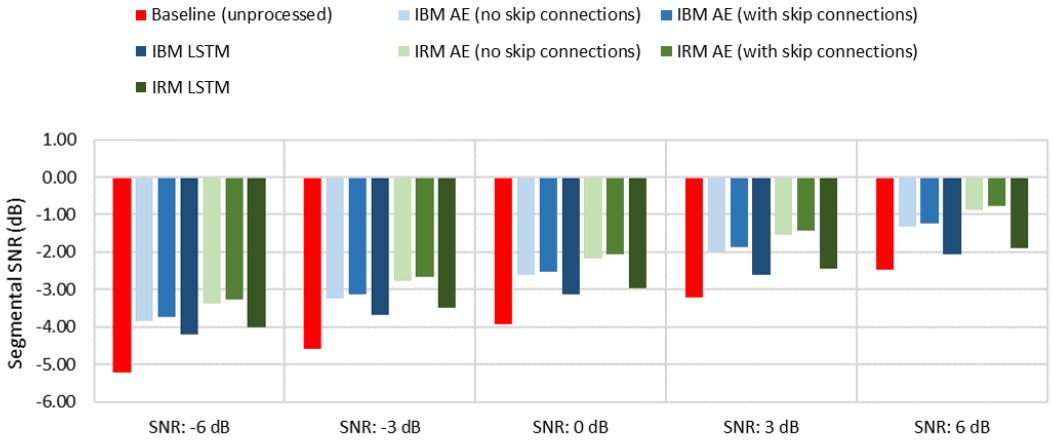


Figure 4.2: SSNR test set results averaged for all noise categories

By considering the STOI and SSNR results together, it can be concluded that on average, the models are effective at removing noise but this comes at the cost of a reduction in intelligibility. This could be due to speech components being affected or removed by the processing, resulting in distortions in the processed signal.

In the following sections, the results obtained for each noise category will be presented and discussed. Additionally, descriptions of the processed speech will be provided, based

on listening to a number of examples.

4.2 Simple Noise Category

The noises in this category included the sounds of vacuum cleaners, air conditioning units, photocopiers and other electrical appliances. Again, according to the STOI results presented in table 4.3, the best option was to do no processing, though the observed difference between the baseline and the LSTM with the IRM training target is very small. It is observed that the skip connections have no effect on the CAE trained using the IRM as the target, but when the IBM was the training target, the performance was actually degraded slightly by the inclusion of skip connections. The general trends observed for the averaged noises category are also observed here, though the LSTM outperforms the CAE by a slightly larger margin in the IRM case. The difference in performance between the LSTM models considering the two training targets is quite significant.

	Metric	STOI				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.675**	**0.702**	**0.728**	**0.753**	**0.777**
	CAE (no skip connections)	0.601	0.634	0.666	0.696	0.724
	CAE (with skip connections)	0.592	0.626	0.658	0.689	0.719
	LSTM	0.578	0.602	0.628	0.650	0.670
IRM	Baseline (unprocessed)	0.675	0.702	0.728	0.753	0.777
	CAE (no skip connections)	0.650	0.678	0.706	0.733	0.757
	CAE (with skip connections)	0.649	0.678	0.706	0.733	0.757
	LSTM	0.673	0.699	0.724	0.748	0.770

Table 4.3: STOI test set results for simple noise category

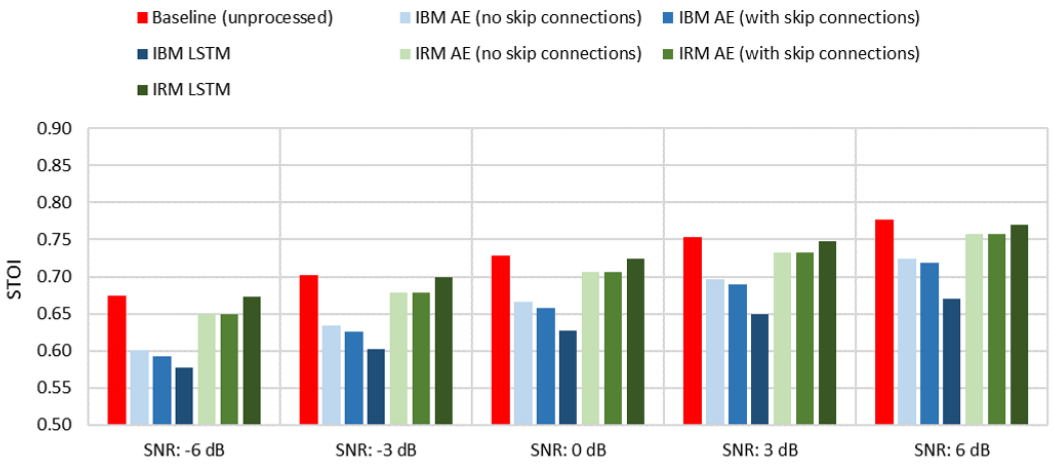


Figure 4.3: STOI test set results for simple noise category

Table 4.4 shows the SSNR results obtained for the simple noise category. Consistent with the average over all noises, the CAE trained using the IRM provided the best performance. The results from all models are contained within a small range of less than 1 dB. The effect of adding skip connections to the CAE appears to be almost negligible, but becomes slightly more pronounced with increasing SNR. The results are illustrated graphically in figure 4.4.

	Metric	Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	-6.496	-5.921	-5.307	-4.657	-3.970
	CAE (no skip connections)	-4.836	-4.260	-3.662	-3.047	-2.414
	CAE (with skip connections)	-4.837	-4.272	-3.690	-3.086	-2.465
	LSTM	-4.432	-3.918	-3.404	-2.895	-2.381
IRM	Baseline (unprocessed)	-6.496	-5.921	-5.307	-4.657	-3.970
	CAE (no skip connections)	-4.440	-3.858	-3.261	-2.647	-2.019
	CAE (with skip connections)	**-4.271**	**-3.688**	**-3.087**	**-2.468**	**-1.835**
	LSTM	-4.441	-3.905	-3.366	-2.815	-2.275

Table 4.4: SSNR test set results for simple noise category

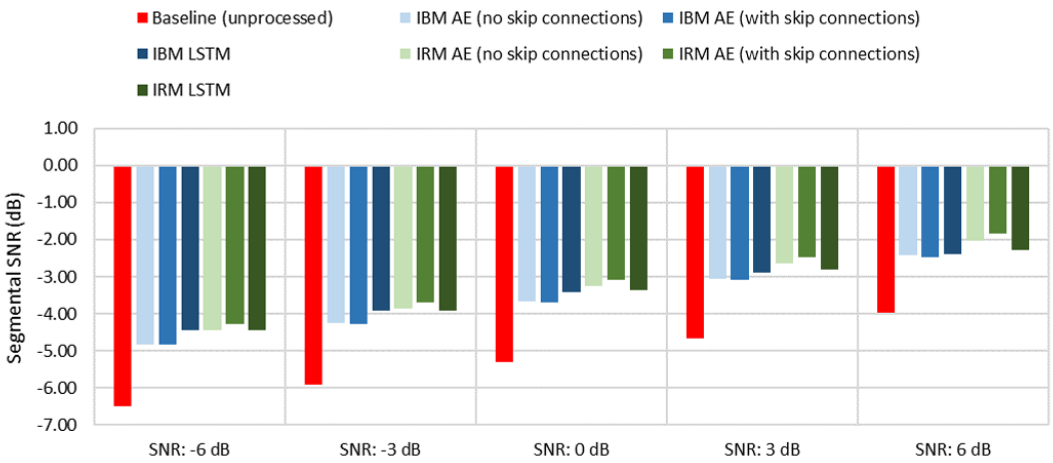
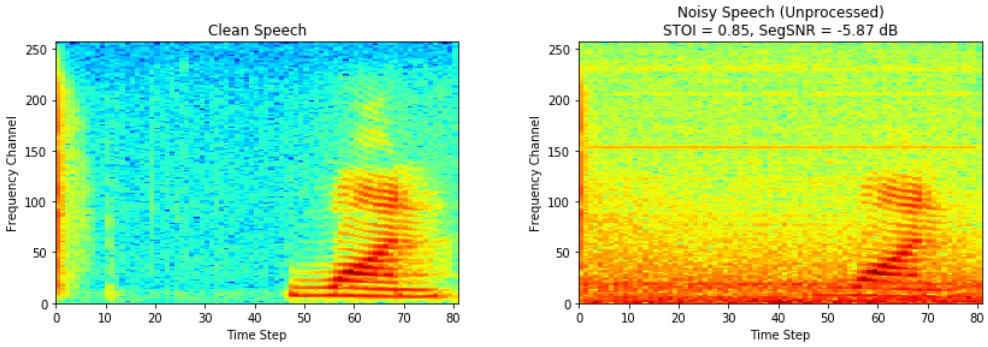


Figure 4.4: SSNR test set results for simple noise category

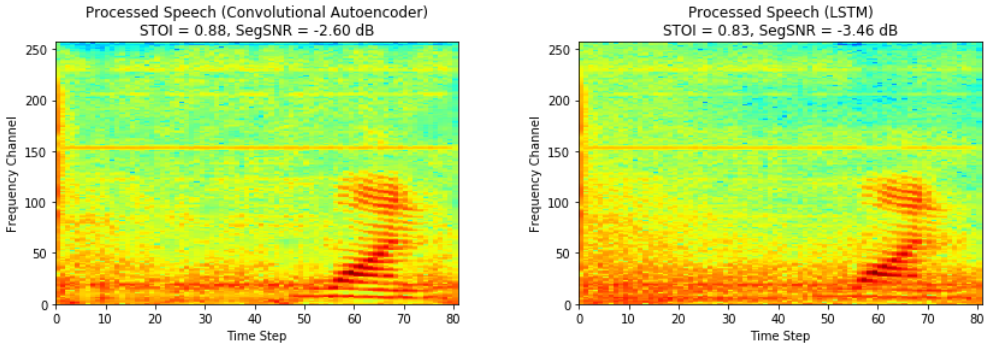
Figure 4.5 relates to the utterance of the word “one”, with air conditioning noise at 0 dB. Figure 4.5a shows the clean speech and figure 4.5b shows the noisy speech. The processed results are shown in figures 4.5c (CAE) and 4.5d (LSTM), where the IRM was used as the training target in both cases.

The reader should note that here, as well as in the remaining illustrative examples, the training target used was the IRM, unless otherwise stated. This decision was based on the IBM causing the processed LPS to have gaps in it, making it difficult to observe and comment on the effect the processing had on the speech-related structures in the data.



(a) Clean: Woman saying “one”

(b) Noisy: Speech with air conditioning at 0dB



(c) Processed Speech from CAE

(d) Processed Speech from LSTM

Figure 4.5: A comparison of clean, noisy and processed speech obtained from the CAE and LSTM models, using an example from the simple noise category

The CAE achieved a higher SSNR score than the LSTM, removing more noise, especially around the low frequency channels, where it is seen that the reconstruction of the structure of the speech is better. This could also explain the increase in STOI for the CAE model. Interestingly, the horizontal lines in the LPS corresponding to the ‘humming’ of the air conditioning unit were not removed by either the CAE or LSTM, despite the models being trained on noise types that shared this characteristic.

4.3 Intermediate Noise Category

The noises in this category included the sounds of sirens, dogs barking, the rainforest, typing and munching (eating crunchy food). The STOI results presented in table 4.5 show that the CAE model including skip connections, trained on the IRM, offers a slight improvement over the unprocessed noisy baseline. There is still a significant difference observed between the performance of the LSTM models, though it is smaller than the

difference observed in the simple noise case.

	Metric	STOI				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	0.738	0.758	0.778	0.797	0.815
	CAE (no skip connections)	0.726	0.747	0.767	0.786	0.803
	CAE (with skip connections)	0.730	0.751	0.770	0.789	0.806
	LSTM	0.656	0.678	0.701	0.722	0.739
IRM	Baseline (unprocessed)	0.738	0.758	0.778	0.797	0.815
	CAE (no skip connections)	0.740	0.760	0.779	0.796	0.813
	CAE (with skip connections)	**0.742**	**0.761**	**0.780**	**0.798**	**0.815**
	LSTM	0.720	0.742	0.761	0.781	0.798

Table 4.5: STOI test set results for intermediate noise category

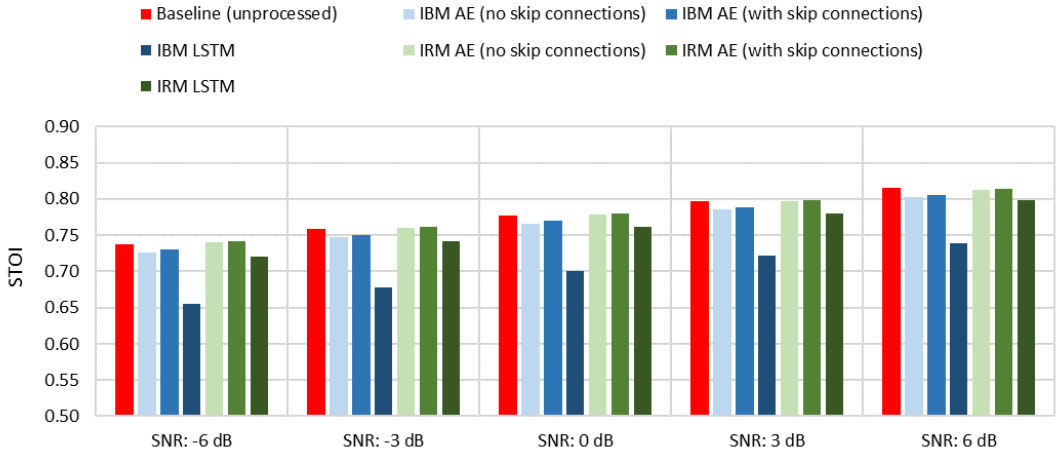


Figure 4.6: STOI test set results for intermediate noise category

Table 4.6 shows the SSNR results obtained for the intermediate noise category, with the results shown graphically in figure 4.7. The CAE with skip connections provided an improvement of almost 2 dB in the lowest SNR condition. The inclusion of skip connections offered a slight improvement for the CAE model trained using the IBM, but the difference for the IRM variant was negligible. Again, the LSTM performance was considerably worse than the CAE, with positive SNR conditions yielding results worse than the baseline.

	Metric	Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	-3.206	-2.490	-1.739	-0.954	-0.135
	CAE (no skip connections)	-1.765	-1.140	-0.501	0.145	0.805
	CAE (with skip connections)	-1.654	-1.028	-0.391	0.262	0.929
	LSTM	-2.857	-2.306	-1.752	-1.193	-0.681
IRM	Baseline (unprocessed)	-3.206	-2.490	-1.739	-0.954	-0.135
	CAE (no skip connections)	**-1.303**	**-0.691**	**-0.070**	**0.560**	1.195
	CAE (with skip connections)	-1.350	-0.724	-0.085	0.558	**1.208**
	LSTM	-2.773	-2.265	-1.761	-1.245	-0.741

Table 4.6: SSNR test set results for intermediate noise category

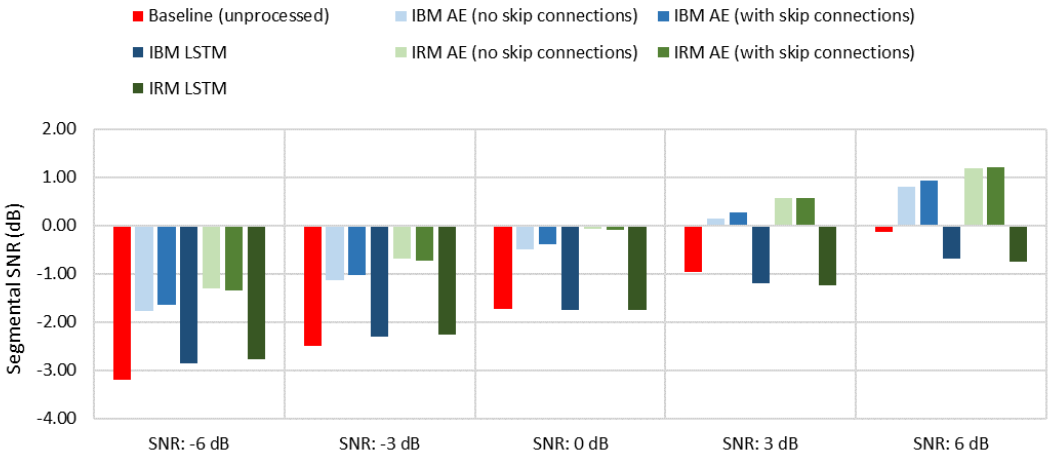


Figure 4.7: SSNR test set results for intermediate noise category

By listening to examples in this category, it was observed that the models were less effective at removing noises such as crunching and typing, compared to noises like sirens or birds. Figure 4.8 shows the LPS for munching noise and bird noise.

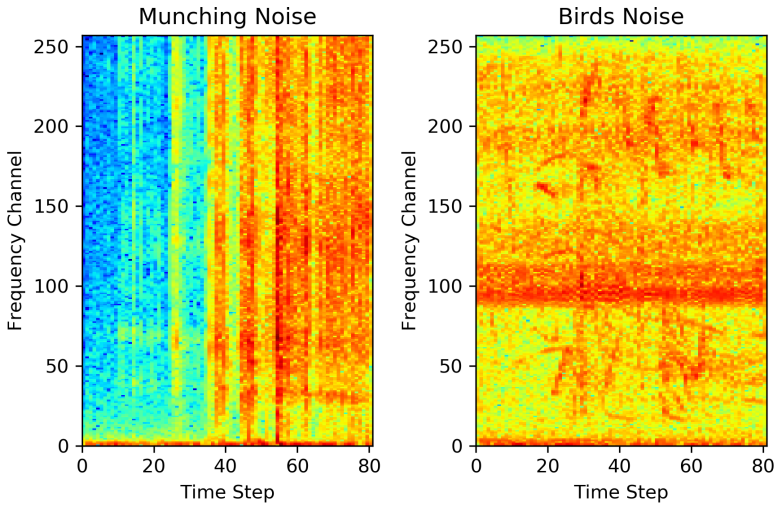
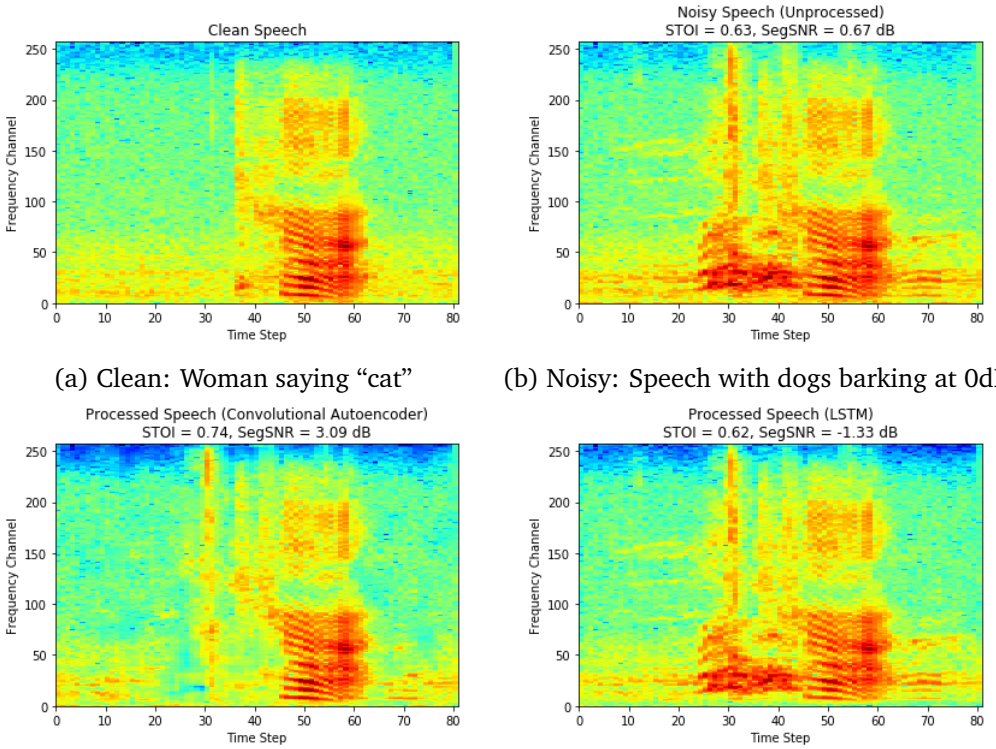


Figure 4.8: A comparison of the log-power spectra of munching noise and bird noise

Visually, there is little observable structure across the frequency channels for the munching noise. In contrast, there are many identifiable structures in the LPS of the bird noise, resulting from its tonal nature. It is supposed that this apparent lack of structure makes it challenging for the models to effectively characterise these types of 'staccato' or 'impact-like' noises.

Figure 4.9 relates to the utterance of the word "cat", with the noise of dogs barking at 0 dB.



(a) Clean: Woman saying “cat”

(b) Noisy: Speech with dogs barking at 0dB

(c) Processed Speech from CAE

(d) Processed Speech from LSTM

Figure 4.9: A comparison of clean, noisy and processed speech obtained from the CAE and LSTM models, using an example from the intermediate noise category

The result from the CAE is visually impressive. While a feature of the noise displayed as a vertical line remains around time step 30, much of the rest of the noise has been removed while the speech appears mostly intact. Upon listening to the reconstructed audio from the CAE, the barking was almost completely inaudible, apart from a little residual noise at the end of the speech segment. These observations are reflected in the scores, with a significant STOI increase of 0.11 and around 2.5 dB improvement in SSNR.

In contrast, the result from the LSTM model was poor, with a decrease in both STOI and SSNR. Upon listening to the reconstructed audio, there was little perceptible difference between the noisy audio and the reconstruction. It can be seen by comparing figures 4.9b and 4.9d that there was little effect on the barking noise.

4.4 Hard Noise Category

The noises in this category were all related to speech and included babble, cafeteria noise and airport announcements. The STOI results presented in table 4.7 show that the baseline score was the best. The LSTM model trained using the IRM consistently outperformed the equivalent CAE model, with the difference being more prominent at lower, more challenging SNRs. The effect of adding skip connections to the CAE models appears to be negligible in terms of STOI results.

	Metric	STOI				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.634**	**0.663**	**0.692**	**0.720**	**0.746**
	CAE (no skip connections)	0.537	0.573	0.607	0.640	0.669
	CAE (with skip connections)	0.537	0.573	0.608	0.642	0.673
	LSTM	0.554	0.581	0.608	0.633	0.656
IRM	Baseline (unprocessed)	0.634	0.663	0.692	0.720	0.746
	CAE (no skip connections)	0.599	0.631	0.662	0.692	0.719
	CAE (with skip connections)	0.596	0.630	0.662	0.692	0.720
	LSTM	0.616	0.645	0.673	0.700	0.726

Table 4.7: STOI test set results for hard noise category

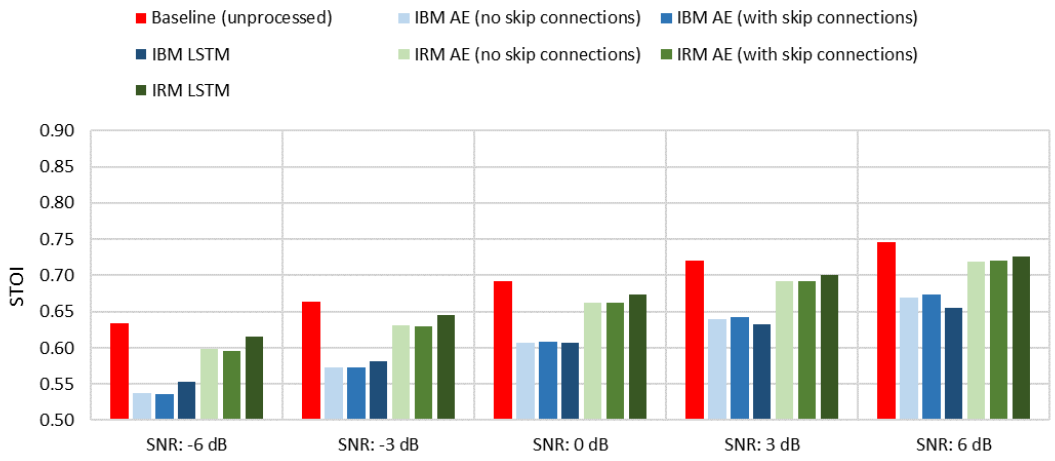


Figure 4.10: STOI test set results for hard noise category

Table 4.8 shows the SSNR results obtained for the hard noise category, with the results shown graphically in figure 4.11. The best performance was obtained by the CAE with skip connections, trained using the IRM. The improvement in dB offered by this model

over the baseline is similar to that observed for intermediate noises. The LSTM models again perform consistently worse than the CAEs in terms of SSNR.

	Metric	Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	-5.952	-5.353	-4.716	-4.043	-3.333
	CAE (no skip connections)	-4.890	-4.289	-3.656	-3.001	-2.330
	CAE (with skip connections)	-4.677	-4.084	-3.462	-2.813	-2.146
	LSTM	-5.301	-4.783	-4.247	-3.700	-3.149
IRM	Baseline (unprocessed)	-5.952	-5.353	-4.716	-4.043	-3.333
	CAE (no skip connections)	-4.343	-3.750	-3.134	-2.494	-1.839
	CAE (with skip connections)	**-4.196**	**-3.596**	**-2.978**	**-2.341**	**-1.687**
	LSTM	-4.820	-4.295	-3.770	-3.231	-2.696

Table 4.8: SSNR test set results for hard noise category

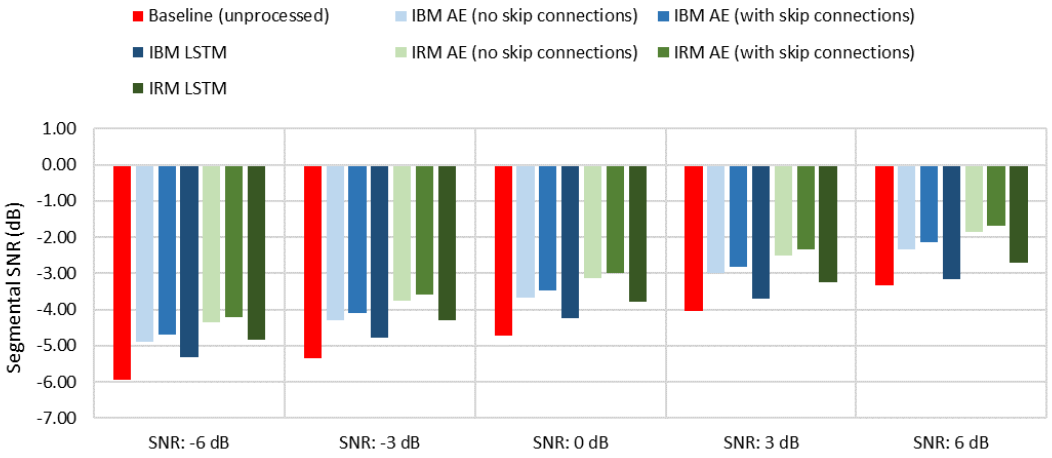
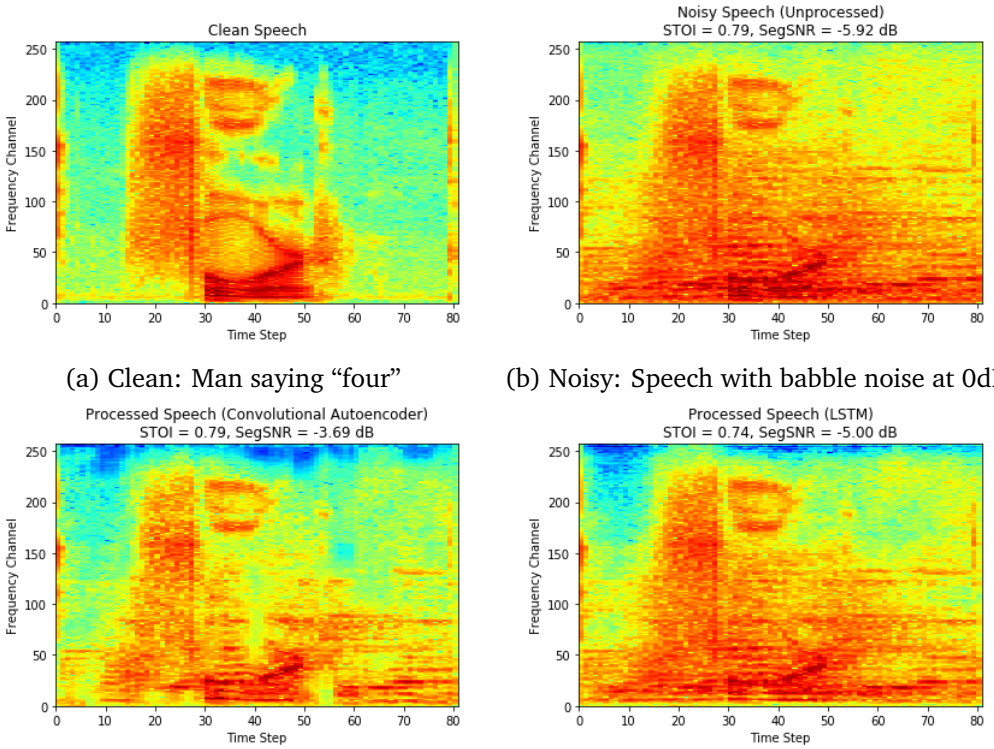


Figure 4.11: SSNR test set results for hard noise category

Figure 4.12 relates to the utterance of the word “four” with babble noise at 0 dB.



(a) Clean: Man saying “four”

(b) Noisy: Speech with babble noise at 0dB

(c) Processed Speech from CAE

(d) Processed Speech from LSTM

Figure 4.12: A comparison of clean, noisy and processed speech obtained from the CAE and LSTM models, using an example from the hard noise category

For the processed result from the CAE, the babble noise in the background was reduced, however the speech was also slightly distorted and sounded like it was in the background. For the LSTM result, the background noise was not reduced as much as with the CAE, and the speech sounded like the speaker had become more distant from the microphone.

In the processed LPS from the CAE, there are noticeable differences in the structure of the speech components. Between time steps 30 and 50 in the clean speech, there are clearly visible features present in the approximate range of frequency channels 40 to 120. The notable absence of these features after processing using the CAE may explain the lack of change in STOI, as well as the audible distortion in the reconstructed audio signal.

4.5 Combined Noises

In this category, vacuum cleaner noise, dogs barking and sirens were present simultaneously in the noise clips. The STOI results presented in table 4.9 are shown graphically in figure 4.13. The baseline once again showed the best performance, but the LSTM trained with the IRM was clearly the best model in terms of STOI. There is a clear difference in performance not only between the IBM and IRM, but also between the CAE and LSTM.

COMBINED	Metric	STOI				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.686**	**0.712**	**0.737**	**0.761**	**0.783**
	CAE (no skip connections)	0.530	0.558	0.588	0.621	0.651
	CAE (with skip connections)	0.553	0.584	0.614	0.644	0.671
	LSTM	0.539	0.569	0.600	0.623	0.651
IRM	Baseline (unprocessed)	0.686	0.712	0.737	0.761	0.783
	CAE (no skip connections)	0.604	0.631	0.658	0.685	0.710
	CAE (with skip connections)	0.617	0.642	0.666	0.690	0.714
	LSTM	0.681	0.704	0.726	0.749	0.767

Table 4.9: STOI test set results for combined noises

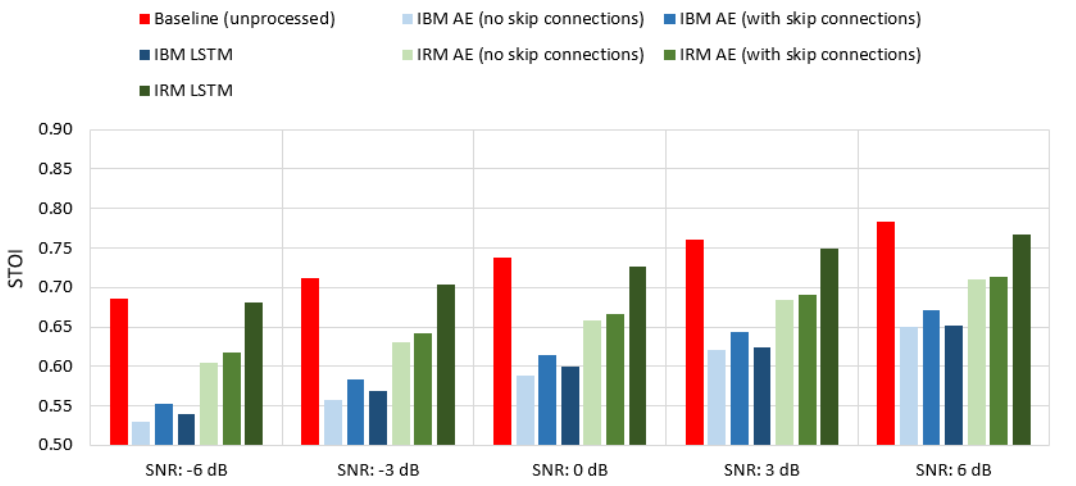


Figure 4.13: STOI results for combined noises (test set)

Table 4.10 shows the SSNR results obtained for the combined noises category, with the results shown graphically in figure 4.14. Though there was large variation in the STOI results, the SSNR results are more closely grouped across models. The CAE with skip connections trained on the IRM achieves the best results, with a marginal advantage

over the equivalent LSTM model.

COMBINED	Metric	Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6
IBM	Baseline (unprocessed)	-5.933	-5.300	-4.634	-3.935	-3.205
	CAE (no skip connections)	-5.523	-4.938	-4.320	-3.668	-3.004
	CAE (with skip connections)	-5.396	-4.793	-4.177	-3.560	-2.940
	LSTM	-5.728	-5.254	-4.739	-4.234	-3.702
IRM	Baseline (unprocessed)	-5.933	-5.300	-4.634	-3.935	-3.205
	CAE (no skip connections)	-5.064	-4.476	-3.871	-3.240	-2.610
	CAE (with skip connections)	**-4.957**	**-4.349**	**-3.729**	**-3.100**	**-2.472**
	LSTM	-4.992	-4.474	-3.944	-3.429	-2.937

Table 4.10: SSNR test set results for combined noises

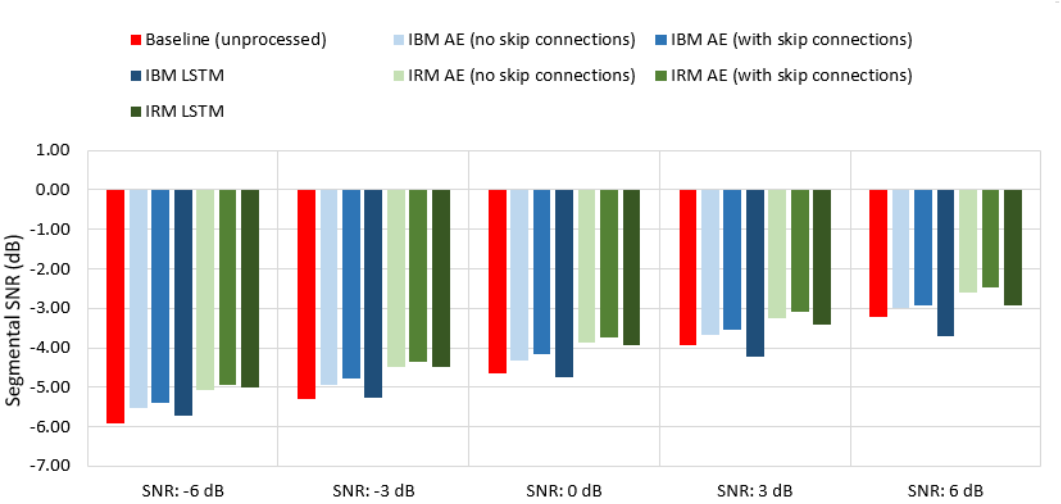


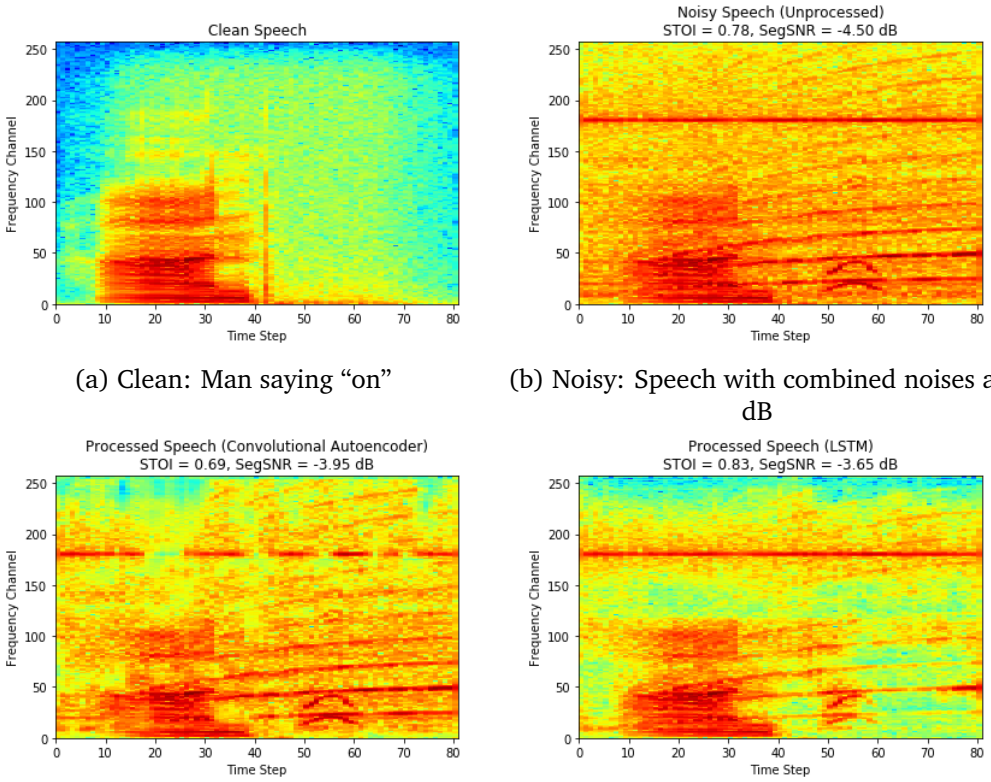
Figure 4.14: SSNR results for combined noises (test set)

Figure 4.15a shows the LPS for the clean utterance of the word “on”. Figure 4.15b shows the noisy speech, where the three noise types added to form the combined noise can be clearly distinguished. The vacuum cleaner noise can be identified as the horizontal line at around frequency channel 180, the siren is seen as the equidistant lines and the dog barking can be seen as a sharp arc between time steps 50 and 60.

In figure 4.15c, the processed result from the CAE shows that the vacuum cleaner noise was effectively removed from some frames, but not others. Audibly, there was a slight distortion to the speech and though the volume of the background noise was reduced,

no component in particular was noticeably attenuated.

In figure 4.15d, the processed result from the LSTM shows that the overall noise level was reduced. Some components of the siren were effectively removed, with this being confirmed audibly. The dog barking can still be seen, but this noise was suppressed very well upon listening. The structure of the speech components around frequency channels 50 - 80 appear to be retained better than in the CAE result. This may explain the fairly significant difference in STOI scores between the two.



(a) Clean: Man saying “on”

(b) Noisy: Speech with combined noises at dB

(c) Processed Speech from CAE using the IRM

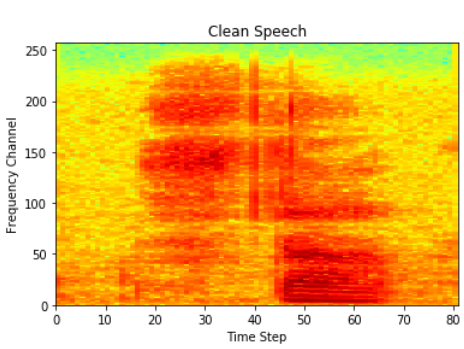
(d) Processed Speech from LSTM using the IRM

Figure 4.15: A comparison of clean, noisy and processed speech obtained from the CAE and LSTM models, using an example from the combined noise category

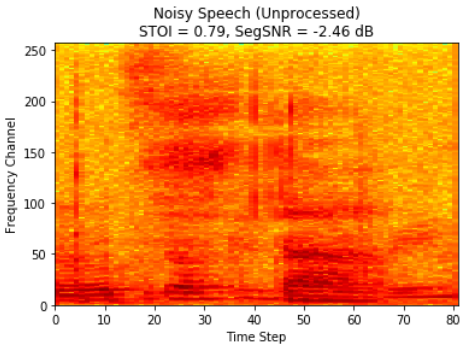
4.6 Additional Evaluations

For completeness, an example of the processed LPS resulting from the application of the IBM has been included. Figure 4.16 relates to the utterance of the word “stop”, with babble noise at 0 dB, when the IBM was used as the training target for the models.

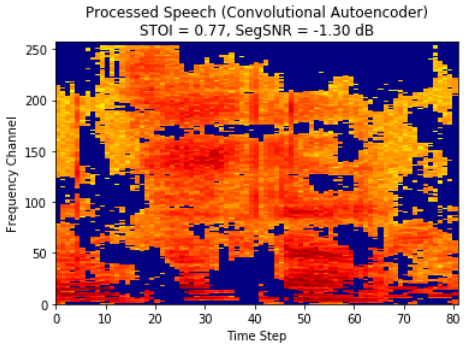
Figure 4.16a shows the clean speech and figure 4.16b shows the noisy speech. Figure 4.16c shows the processed LPS based on the predicted IBM from the CAE model. It can be seen that while a slight improvement in SSNR was attained, the overall structure of the clean speech was not reconstructed well. The observed gaps manifest themselves as audible musical noise in the reconstructed audio signal. In figure 4.16d, it can be seen that the LSTM model had little effect, though visually it appears to have retained the structure of the speech signal better than the CAE.



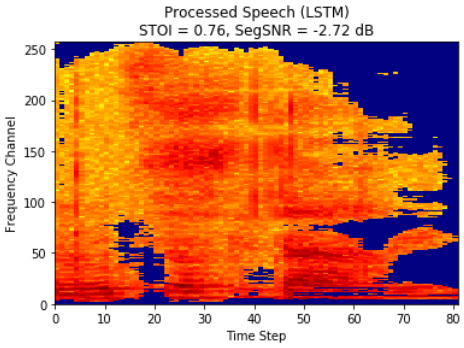
(a) Clean: Woman saying “stop”



(b) Noisy: Speech with babble noise at 0dB



(c) Processed Speech from CAE using the IBM



(d) Processed Speech from LSTM using the IBM

Figure 4.16: A comparison of clean, noisy and processed speech obtained from the CAE and LSTM models, using an example from the hard noise category, with models trained using the IBM

For many of the experiments conducted, the results obtained from the LSTM model were poorer than those from the CAE model. Steps were taken to investigate the possible reasons for this observation. The LSTM model contained three layers, two of which were BLSTM layers with 128 nodes and the final layer was an LSTM layer with 257 nodes to predict a mask with the correct shape. The number of layers and hidden nodes, and thus the complexity of the model, was lower than some of the similar models proposed in the literature, as discussed in 3.4.

In figure 4.17, an example from the validation set was used to show a ground truth IBM compared with an IBM predicted by the LSTM model. It can be seen that the prediction contains many misclassified T–F units and it does not contain the finer resolution components of the ground truth IBM.

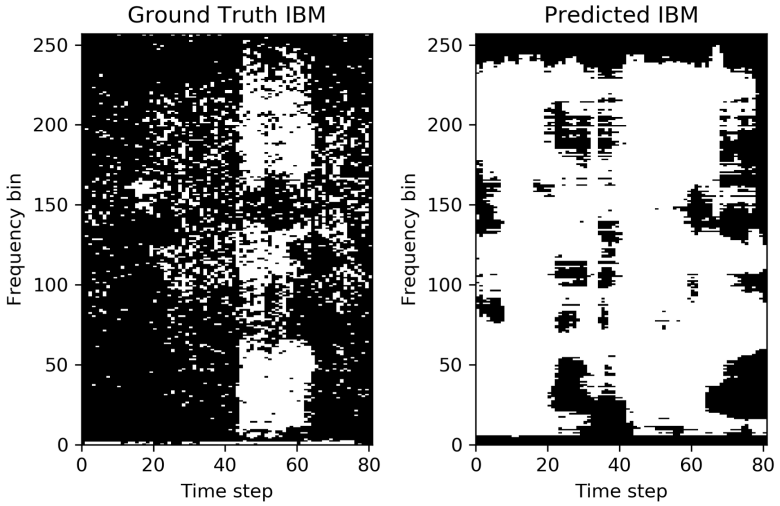


Figure 4.17: Ground truth IBM and predicted IBM from LSTM model

It was decided to investigate whether or not the model was complex enough to be capable of effectively learning a mapping from a noisy LPS input to a mask. In order to do this, single batch containing 32 examples from the training set was used to train the model for 1000 epochs. Due to the small number of examples, the model could quickly be trained for many more epochs than the number used for the full training set, with the training loss plotted in figure 4.18. The intention was to deliberately cause the model to overfit the training data to observe the prediction and compare it to the ground truth.

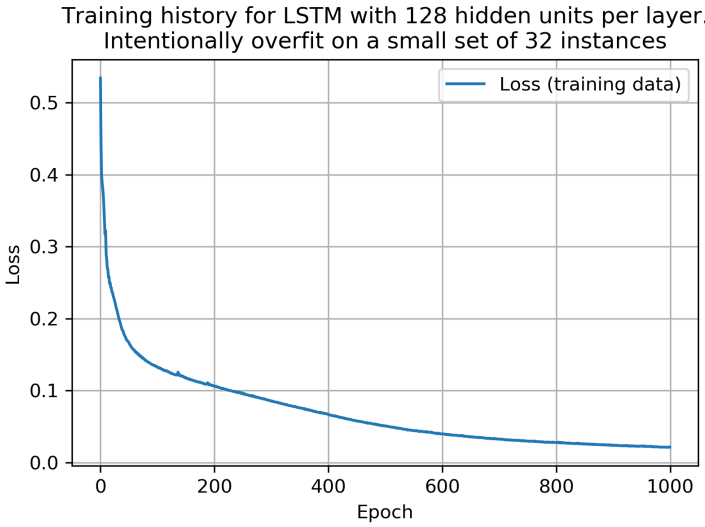


Figure 4.18: Training loss for model deliberately trained to overfit

Figure 4.19 shows the ground truth IBM beside the prediction from the LSTM model, where it can be seen that the two are almost indistinguishable.

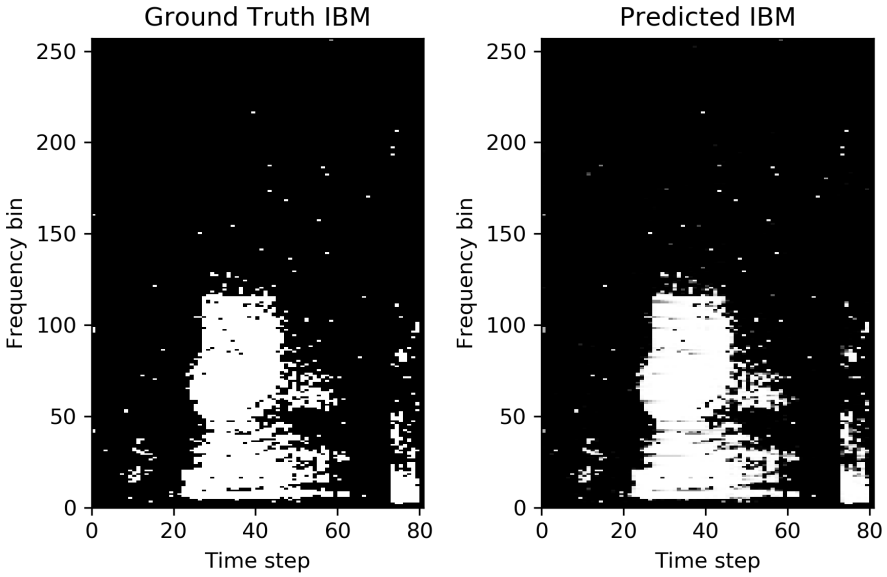


Figure 4.19: Ground truth IBM and predicted IBM from LSTM model trained on only 32 instances for 1000 epochs

It was concluded that when the LSTM model was trained on the whole training dataset of 36,500 instances, time constraints meant the number of epochs was insufficient for the model to learn effectively, despite it apparently being capable of doing so.

4.7 Evaluation of Hypotheses

In this section, each of the hypotheses presented in section 3.5 will be discussed with respect to whether the experimental results enable their verification or disproval.

1. **IRM should offer improvements over IBM**

   This was verified in the experimental results obtained for evaluation on the test data, in terms of both STOI and SSNR. It was observed that in the results obtained from the validation data, the above was also true for the STOI scores. However, when the SSNR performance metric was considered, the opposite was observed, apparently disproving the hypothesis. It is not clear why this was observed and it should be investigated further.

2. **Skip connections should improve performance**

   Though the results obtained show that the inclusion of skip connections resulted in improved performance in general, the observed difference in performance was was marginal. In order to more fully investigate the effect of the skip connections, the activation maps from the encoder layers could be examined visually and in terms of their statistical properties. This may determine whether or not the skip connections were adding useful information to the decoder inputs.

3. **LSTM should outperform CAE in highly non-stationary noise conditions**

   The most non-stationary noise was present in the combined case, where the noise consisted of a mixture of vacuum cleaner noise, a siren and dogs barking. In this case, the LSTM with the IRM as the training target outperformed all other models in terms of STOI, increasing the score by as much as 28%. Despite this, the results showed that the baseline of unprocessed noisy speech achieved the highest STOI score. The LSTM using the IRM was also seen to be competitive with the CAE in terms of SSNR.

4.8 Bugs and Other Challenges

It was determined that the Python script accompanying the MS-SNSD dataset described in section 3.1 was not working in the way it was initially expected. While the audio file containing the noise *was* randomly selected from the available files, the samples from within the selected audio file were *not* randomly selected. Instead, once a noise audio file been randomly selected, given a clean audio file of length N samples, the first N noisy samples were always selected.

As a result, irrespective of the length of a noise audio file, only the first one second of the file was ever used. This greatly reduced the variety of noises the synthesised noisy data actually contained. Testing confirmed that the number of unique noises in the synthesised dataset was determined by the number of noise files in the directory. Unfortunately this was discovered too late in the project to resolve the problem and re-train the models.

This discovery offers an explanation for the poor generalisation observed in the results. If the models had been exposed to thousands of different noise clips rather than only the small number they were presented with (i100), an improvement in the performance for the test dataset may have been observed.

Unfortunately, after the model training was complete, it was discovered that there was an error in the convolutional autoencoder model. The intention was to have repeated elements consisting of a convolutional layer, a batch normalisation layer and a ReLU activation layer. However, in figure 4.20, it can be seen that in the encoder part of the CAE, the batch normalisation and ReLU layers were bypassed.

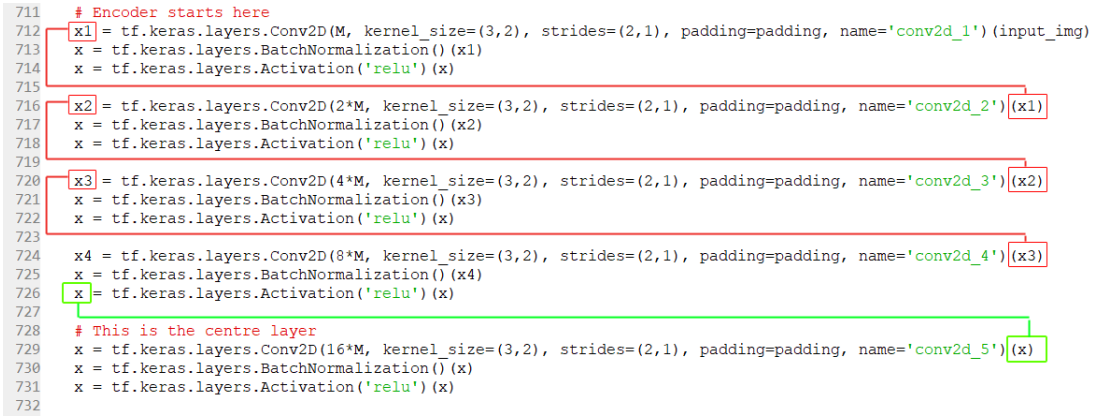


Figure 4.20: Error in CAE model construction

The output of the first `Conv2D` layer on line 712, `x1`, is passed to the `BatchNormalization` layer on line 713. The result is passed to the ReLU `Activation` layer on line 714. However, the `Conv2D` layer on line 716 takes `x1` directly, rather than `x`, which has passed through the `BatchNormalization` and ReLU `Activation` layers.

This pattern continues until the fourth `Conv2D` layer, on line 724, whose output is propagated through the remainder of the network as intended.

The error was made because in order to implement skip connections, the output from each {Conv2D, BN, ReLU} element in the encoder had to be given a unique name. This was so it could be passed to an `Add` layer to be summed with the input to the corresponding decoder element. This was overlooked because the output shapes from all of the layers were correct and the model trained without error.

Irrespective of the inclusion of skip connections, having the `BatchNormalization` layers properly configured in the encoder could have led to improved results, as it could have increased the rate of convergence during training.

4.9 Alternative Approaches

Upon reflection, in addition to fixing the bugs mentioned above, there are a number of things that could have been done differently during the course of this project. Some of

these are discussed below.

Using smaller training sets would have allowed more experiments to be conducted and would have enabled iterative refinements to the model, due to the decreased training time. The hyperparameters of the proposed models were determined based on values used in similar models proposed in the literature, or some basic experimentation. If smaller datasets were used, or increased computational resources were available, it may have been feasible to perform hyperparameter tuning. However, with the training datasets containing 36,500 instances, hyperparameter tuning was simply not feasible given the available time and resources.

Code efficiency may have been improved by utilising Tensorflow's vectorised implementations of the STFT and ISTFT, rather than the Scipy functions which were used.

It was observed while reviewing the literature that some authors trained their models for a fixed number of epochs, as opposed to implementing an early stopping condition as was done in this work. If there were no constraints on computational resources or available time, it may have been beneficial to train the models for longer. By implementing the model checkpoint callback, the best model would still have been saved irrespective of the number of training epochs completed.

Chapter 5

Conclusion and Future Work

In this work, speech enhancement was investigated, along with the application of deep learning methods to the problem. A testbed system was designed, implemented and tested, which allowed CAE and LSTM models to be implemented and evaluated. A number of factors were considered, such as using different training targets and including skip connections for the CAE. The developed deep learning models were evaluated in terms of the STOI and SSNR across a range of noise conditions.

It was hypothesised that the use of the IRM as the training target would lead to better results than using the IBM. The experimental results showed this to be the case consistently. Apart from this, there was a general lack of consistency observed in the trends of model performance across different noise categories. The inclusion of skip connections provides a slight improvement, but the experimental results do not allow conclusions to be drawn that hold over all conditions tested. This highlights the difficulty associated with developing a single model that exhibits consistent desirable performance across a range of noise types and conditions.

Problems identified in the code caused unexpected behaviour in the dataset synthesis routine, and incorrectly connected layers in the encoder of the CAE. The models were found to be unable to effectively generalise to unseen data. It is unclear whether this was due to the models and their hyperparameters, the bug in the dataset synthesis, or a combination of those factors.

There are a number of opportunities to further explore the problem and try to improve the results obtained. Some examples are listed below.

- As a priority, fix the bugs identified in the code. This would result in a properly constructed CAE model and the datasets would contain a much greater variety of noises.
- Improved training behaviour may be achieved by identifying favourable hyperparameter values using hyperparameter tuning.
- Investigate the effect of enhancing the phase information rather than using the noisy phase for audio signal reconstruction.
- Combine the two proposed models to form a hybrid CAE-LSTM network, similar to that proposed in [31].
- A context window containing neighbouring frames could be implemented to define the feature vectors used as input to the models.

Exploring these options may further the development of deep learning methods for speech enhancement.

Appendix A

Testbed System Design

Figures A.1 and A.2 illustrate the training and testing procedures, respectively.

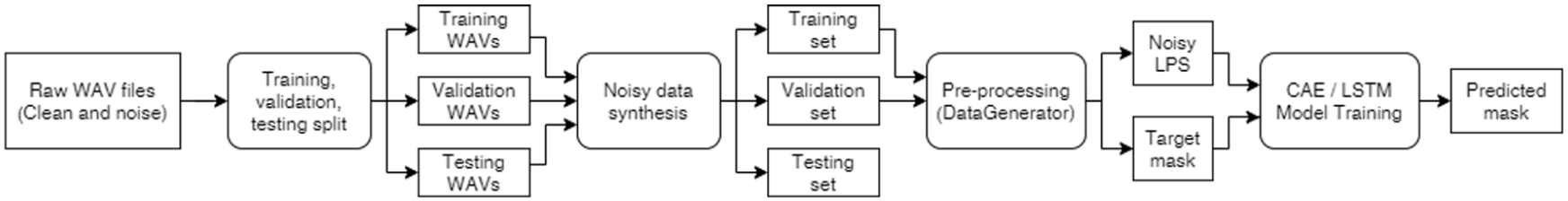


Figure A.1: Flowchart of training procedure

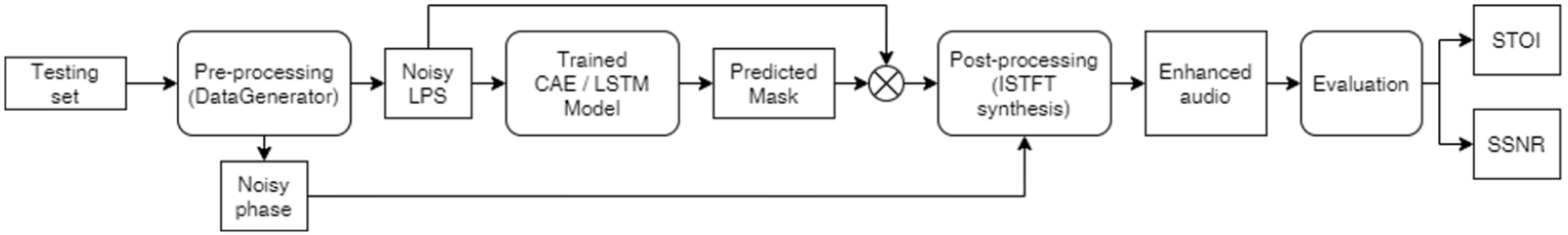


Figure A.2: Flowchart of testing procedure

Appendix B

Full Test and Validation Results

This appendix contains the full set of results obtained from the evaluation of the models on the test and validation sets. The results are presented in both tabular and graphical form. The purpose of including the validation set results is to permit analysis of the ability of the proposed models to generalise to novel conditions, in terms of both speakers and noise.

	Metric	STOI					Segmental SNR				
Mask Type	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.682**	**0.708**	**0.733**	**0.757**	**0.780**	-5.218	-4.588	-3.921	-3.218	-2.479
	CAE (no skip connections)	0.621	0.651	0.680	0.707	0.732	-3.831	-3.230	-2.606	-1.968	-1.313
	CAE (with skip connections)	0.620	0.650	0.679	0.707	0.733	-3.723	-3.128	-2.514	-1.879	-1.228
	LSTM	0.596	0.620	0.645	0.668	0.688	-4.197	-3.669	-3.134	-2.596	-2.070
IRM	Baseline (unprocessed)	0.682	0.708	0.733	0.757	0.780	-5.218	-4.588	-3.921	-3.218	-2.479
	CAE (no skip connections)	0.663	0.690	0.716	0.740	0.763	-3.362	-2.766	-2.155	-1.527	-0.888
	CAE (with skip connections)	0.662	0.690	0.716	0.741	0.764	**-3.272**	**-2.669**	**-2.050**	**-1.417**	**-0.771**
	LSTM	0.670	0.695	0.719	0.743	0.765	-4.012	-3.488	-2.966	-2.431	-1.904

Table B.1: Test set results averaged over all noise difficulty levels

	Metric	STOI					Segmental SNR				
Mask Type	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	0.676	0.701	0.726	0.750	0.772	-4.902	-4.289	-3.640	-2.957	-2.236
	CAE (no skip connections)	0.706	0.729	0.750	0.769	0.786	2.131	2.546	2.977	3.419	3.865
	CAE (with skip connections)	0.706	0.728	0.749	0.769	0.787	**2.351**	**2.754**	**3.172**	**3.602**	**4.037**
	LSTM	0.630	0.653	0.676	0.697	0.717	-2.136	-1.689	-1.246	-0.802	-0.354
IRM	Baseline (unprocessed)	0.676	0.701	0.726	0.750	0.772	-4.902	-4.289	-3.640	-2.957	-2.236
	CAE (no skip connections)	0.760	0.777	0.794	0.809	0.823	0.567	1.091	1.623	2.157	2.693
	CAE (with skip connections)	**0.762**	**0.779**	**0.795**	**0.810**	**0.824**	1.101	1.613	2.131	2.651	3.171
	LSTM	0.662	0.684	0.706	0.727	0.746	-3.326	-2.831	-2.338	-1.843	-1.349

Table B.2: Validation set results averaged over all noise difficulty levels

	Metric	STOI					Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.675**	**0.702**	**0.728**	**0.753**	**0.777**	-6.496	-5.921	-5.307	-4.657	-3.970
	Autoencoder (no skip connections)	0.601	0.634	0.666	0.696	0.724	-4.836	-4.260	-3.662	-3.047	-2.414
	Autoencoder (add skip connections)	0.592	0.626	0.658	0.689	0.719	-4.837	-4.272	-3.690	-3.086	-2.465
	LSTM	0.578	0.602	0.628	0.650	0.670	-4.432	-3.918	-3.404	-2.895	-2.381
IRM	Baseline (unprocessed)	0.675	0.702	0.728	0.753	0.777	-6.496	-5.921	-5.307	-4.657	-3.970
	Autoencoder (no skip connections)	0.650	0.678	0.706	0.733	0.757	-4.440	-3.858	-3.261	-2.647	-2.019
	Autoencoder (add skip connections)	0.649	0.678	0.706	0.733	0.757	**-4.271**	**-3.688**	**-3.087**	**-2.468**	**-1.835**
	LSTM	0.673	0.699	0.724	0.748	0.770	-4.441	-3.905	-3.366	-2.815	-2.275

Table B.3: Results for simple noises from the test set

	Metric	STOI					Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	0.690	0.716	0.740	0.764	0.787	-6.151	-5.572	-4.956	-4.301	-3.609
	Autoencoder (no skip connections)	0.723	0.745	0.765	0.784	0.800	**2.339**	**2.706**	**3.091**	**3.500**	**3.911**
	Autoencoder (add skip connections)	0.717	0.739	0.760	0.780	0.797	2.127	2.501	2.893	3.303	3.706
	LSTM	0.641	0.661	0.680	0.698	0.712	1.254	1.566	1.879	2.184	2.506
IRM	Baseline (unprocessed)	0.690	0.716	0.740	0.764	0.787	-6.151	-5.572	-4.956	-4.301	-3.609
	Autoencoder (no skip connections)	**0.785**	**0.801**	**0.815**	**0.829**	**0.842**	0.473	0.983	1.501	2.024	2.547
	Autoencoder (add skip connections)	0.783	0.799	0.814	0.827	0.840	0.866	1.364	1.866	2.369	2.870
	LSTM	0.742	0.761	0.779	0.797	0.813	-1.717	-1.230	-0.747	-0.273	0.195

Table B.4: Results for simple noises from the validation set

	Metric	STOI					Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	0.738	0.758	0.778	0.797	0.815	-3.206	-2.490	-1.739	-0.954	-0.135
	Autoencoder (no skip connections)	0.726	0.747	0.767	0.786	0.803	-1.765	-1.140	-0.501	0.145	0.805
	Autoencoder (add skip connections)	0.730	0.751	0.770	0.789	0.806	-1.654	-1.028	-0.391	0.262	0.929
	LSTM	0.656	0.678	0.701	0.722	0.739	-2.857	-2.306	-1.752	-1.193	-0.681
IRM	Baseline (unprocessed)	0.738	0.758	0.778	0.797	0.815	-3.206	-2.490	-1.739	-0.954	-0.135
	Autoencoder (no skip connections)	0.740	0.760	0.779	0.796	0.813	**-1.303**	**-0.691**	**-0.070**	**0.560**	1.195
	Autoencoder (add skip connections)	**0.742**	**0.761**	**0.780**	**0.798**	**0.815**	-1.350	-0.724	-0.085	0.558	**1.208**
	LSTM	0.720	0.742	0.761	0.781	0.798	-2.773	-2.265	-1.761	-1.245	-0.741

Table B.5: Results for intermediate noises from the test set

	Metric	STOI					Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	0.698	0.720	0.741	0.762	0.781	-3.122	-2.469	-1.780	-1.061	-0.310
	Autoencoder (no skip connections)	0.751	0.770	0.786	0.802	0.816	**3.324**	**3.761**	**4.204**	**4.652**	**5.102**
	Autoencoder (add skip connections)	0.750	0.768	0.786	0.802	0.817	3.279	3.718	4.169	4.625	5.088
	LSTM	0.628	0.649	0.671	0.691	0.709	-3.201	-2.682	-2.188	-1.678	-1.172
IRM	Baseline (unprocessed)	0.698	0.720	0.741	0.762	0.781	-3.122	-2.469	-1.780	-1.061	-0.310
	Autoencoder (no skip connections)	**0.779**	**0.795**	**0.809**	**0.822**	**0.835**	2.177	2.683	3.196	3.710	4.221
	Autoencoder (add skip connections)	0.778	0.794	0.809	0.822	0.835	2.220	2.743	3.269	3.797	4.326
	LSTM	0.688	0.709	0.729	0.748	0.767	-3.383	-2.898	-2.410	-1.930	-1.444

Table B.6: Results for intermediate noises from the validation set

	Metric	STOI					Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.634**	**0.663**	**0.692**	**0.720**	**0.746**	-5.952	-5.353	-4.716	-4.043	-3.333
	Autoencoder (no skip connections)	0.537	0.573	0.607	0.640	0.669	-4.890	-4.289	-3.656	-3.001	-2.330
	Autoencoder (add skip connections)	0.537	0.573	0.608	0.642	0.673	-4.677	-4.084	-3.462	-2.813	-2.146
	LSTM	0.554	0.581	0.608	0.633	0.656	-5.301	-4.783	-4.247	-3.700	-3.149
IRM	Baseline (unprocessed)	0.634	0.663	0.692	0.720	0.746	-5.952	-5.353	-4.716	-4.043	-3.333
	Autoencoder (no skip connections)	0.599	0.631	0.662	0.692	0.719	-4.343	-3.750	-3.134	-2.494	-1.839
	Autoencoder (add skip connections)	0.596	0.630	0.662	0.692	0.720	**-4.196**	**-3.596**	**-2.978**	**-2.341**	**-1.687**
	LSTM	0.616	0.645	0.673	0.700	0.726	-4.820	-4.295	-3.770	-3.231	-2.696

Table B.7: Results for hard noises from the test set

	Metric	STOI					Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	0.639	0.668	0.696	0.723	0.749	-5.433	-4.827	-4.186	-3.507	-2.789
	Autoencoder (no skip connections)	0.643	0.671	0.697	0.721	0.742	0.731	1.171	1.636	2.104	2.582
	Autoencoder (add skip connections)	0.651	0.677	0.702	0.726	0.748	**1.648**	**2.044**	**2.455**	**2.877**	**3.317**
	LSTM	0.621	0.649	0.677	0.703	0.729	-4.461	-3.950	-3.429	-2.913	-2.397
IRM	Baseline (unprocessed)	0.639	0.668	0.696	0.723	0.749	-5.433	-4.827	-4.186	-3.507	-2.789
	Autoencoder (no skip connections)	0.715	0.737	0.757	0.776	0.793	-0.948	-0.392	0.171	0.738	1.310
	Autoencoder (add skip connections)	**0.725**	**0.745**	**0.764**	**0.781**	**0.798**	0.217	0.733	1.258	1.786	2.316
	LSTM	0.557	0.583	0.611	0.635	0.659	-4.877	-4.364	-3.858	-3.326	-2.798

Table B.8: Results for hard noises from the validation set

COMBINED	Metric	STOI					Segmental SNR (dB)				
Mask	SNR (dB)	-6	-3	0	3	6	-6	-3	0	3	6
IBM	Baseline (unprocessed)	**0.686**	**0.712**	**0.737**	**0.761**	**0.783**	-5.933	-5.300	-4.634	-3.935	-3.205
	Autoencoder (no skip connections)	0.530	0.558	0.588	0.621	0.651	-5.523	-4.938	-4.320	-3.668	-3.004
	Autoencoder (add skip connections)	0.553	0.584	0.614	0.644	0.671	-5.396	-4.793	-4.177	-3.560	-2.940
	LSTM	0.539	0.569	0.600	0.623	0.651	-5.728	-5.254	-4.739	-4.234	-3.702
IRM	Baseline (unprocessed)	0.686	0.712	0.737	0.761	0.783	-5.933	-5.300	-4.634	-3.935	-3.205
	Autoencoder (no skip connections)	0.604	0.631	0.658	0.685	0.710	-5.064	-4.476	-3.871	-3.240	-2.610
	Autoencoder (add skip connections)	0.617	0.642	0.666	0.690	0.714	**-4.957**	**-4.349**	**-3.729**	**-3.100**	**-2.472**
	LSTM	0.681	0.704	0.726	0.749	0.767	-4.992	-4.474	-3.944	-3.429	-2.937

Table B.9: Results for combined noises from the test set

STOI results – validation set.

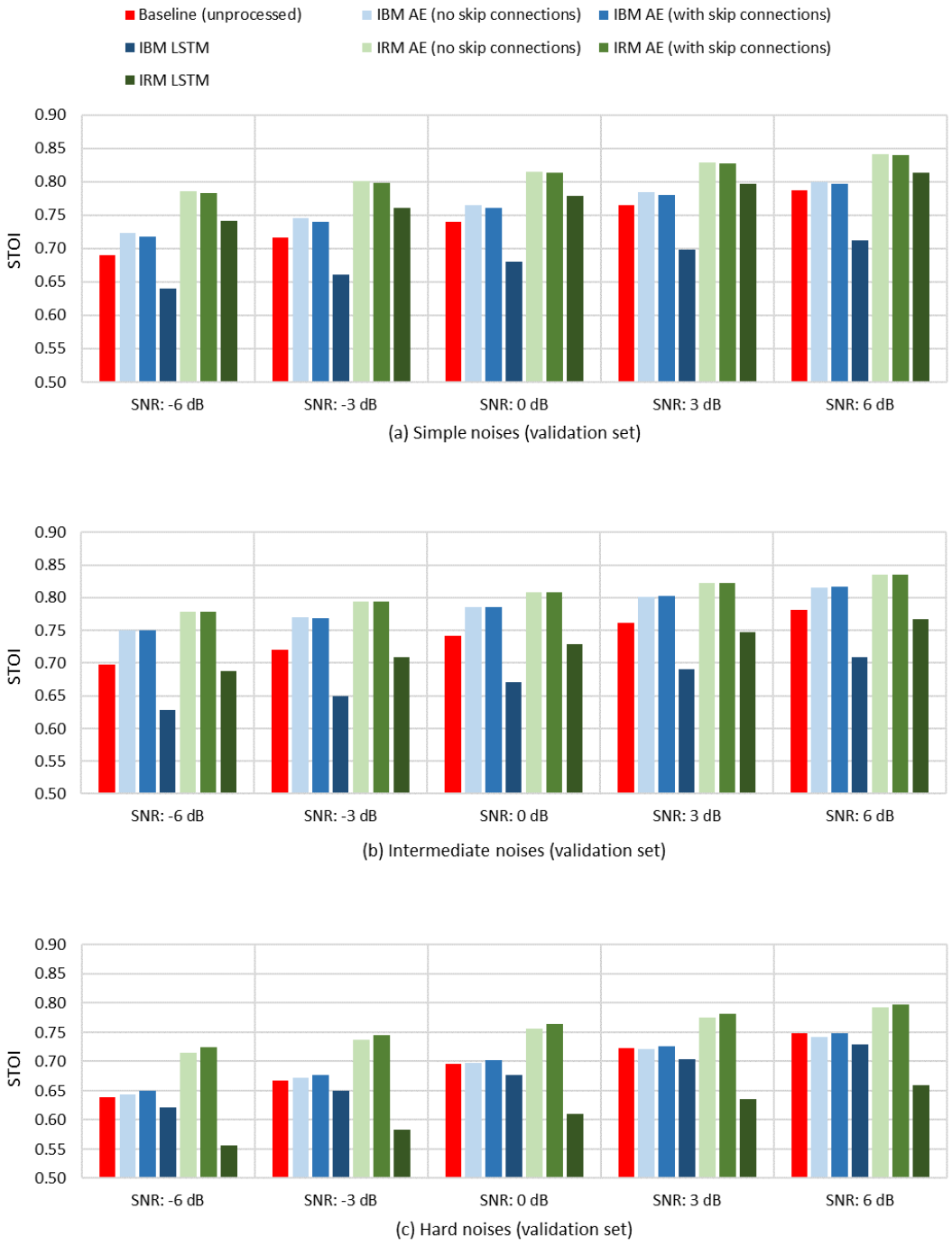


Figure B.1: STOI results for validation set

STOI results – test set.

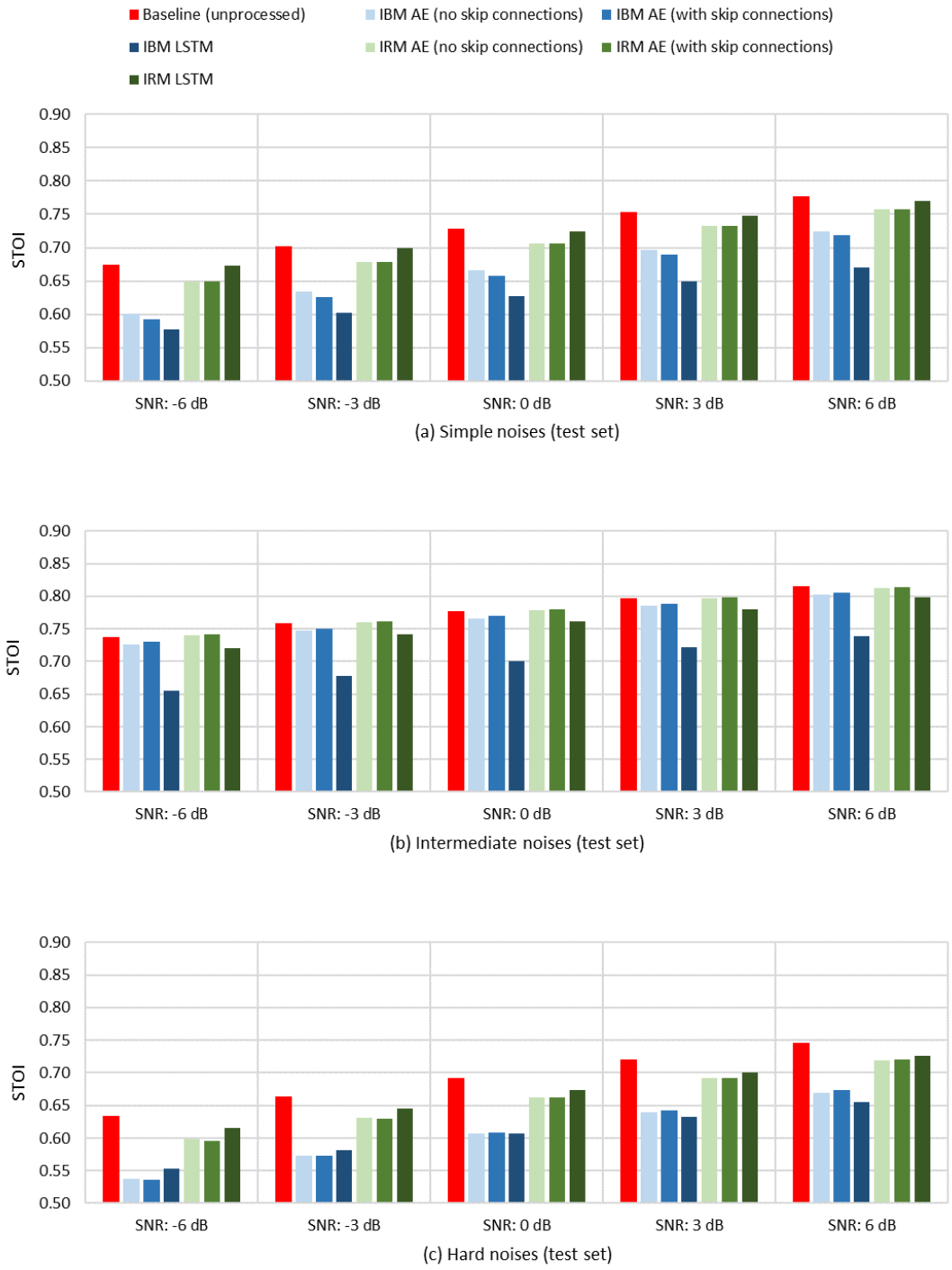


Figure B.2: STOI results for test set

SSNR results – validation set.

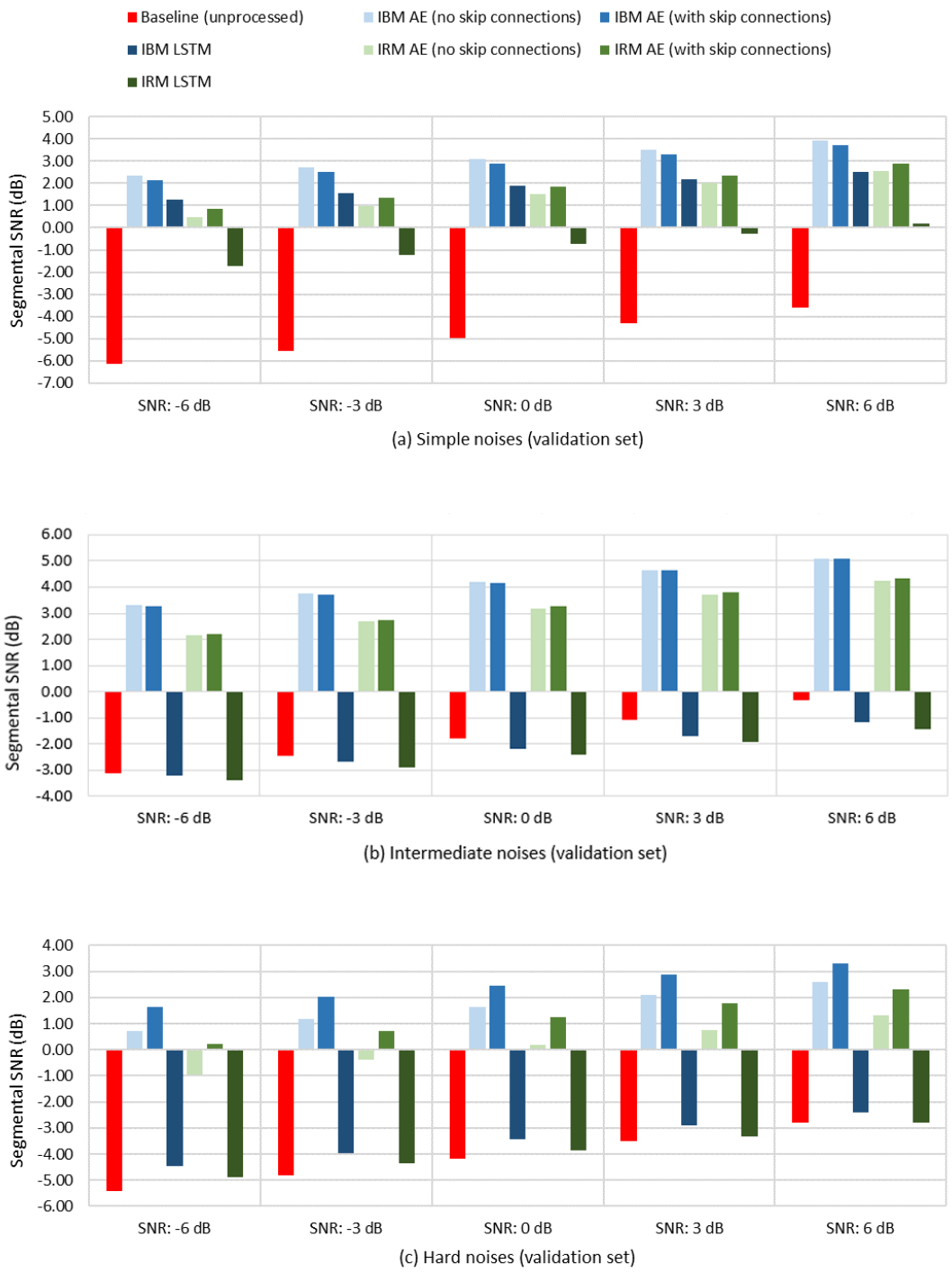


Figure B.3: SSNR results for validation set

SSNR results – test set.

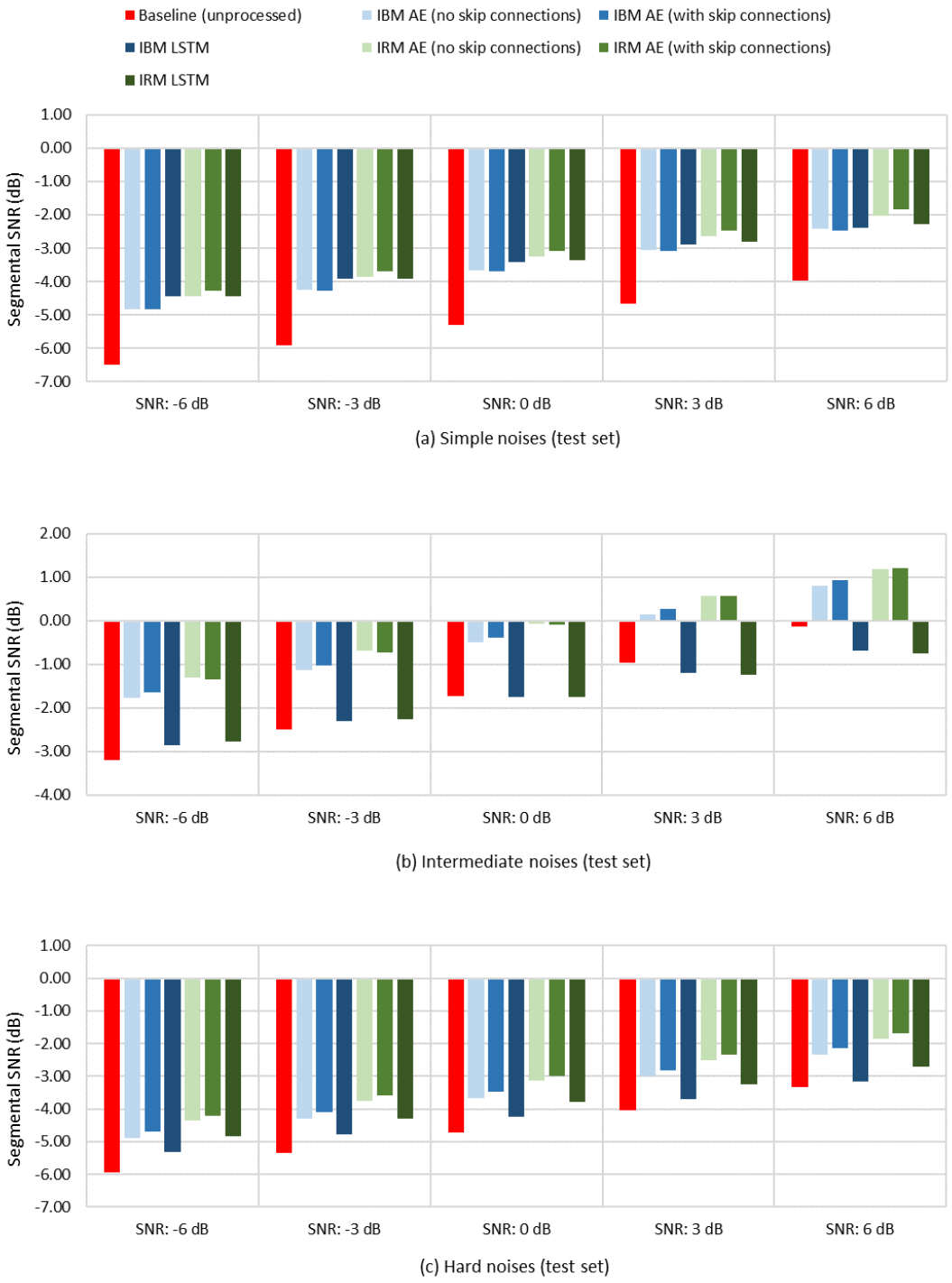


Figure B.4: SSNR results for test set

Average STOI and SSNR – validation set.

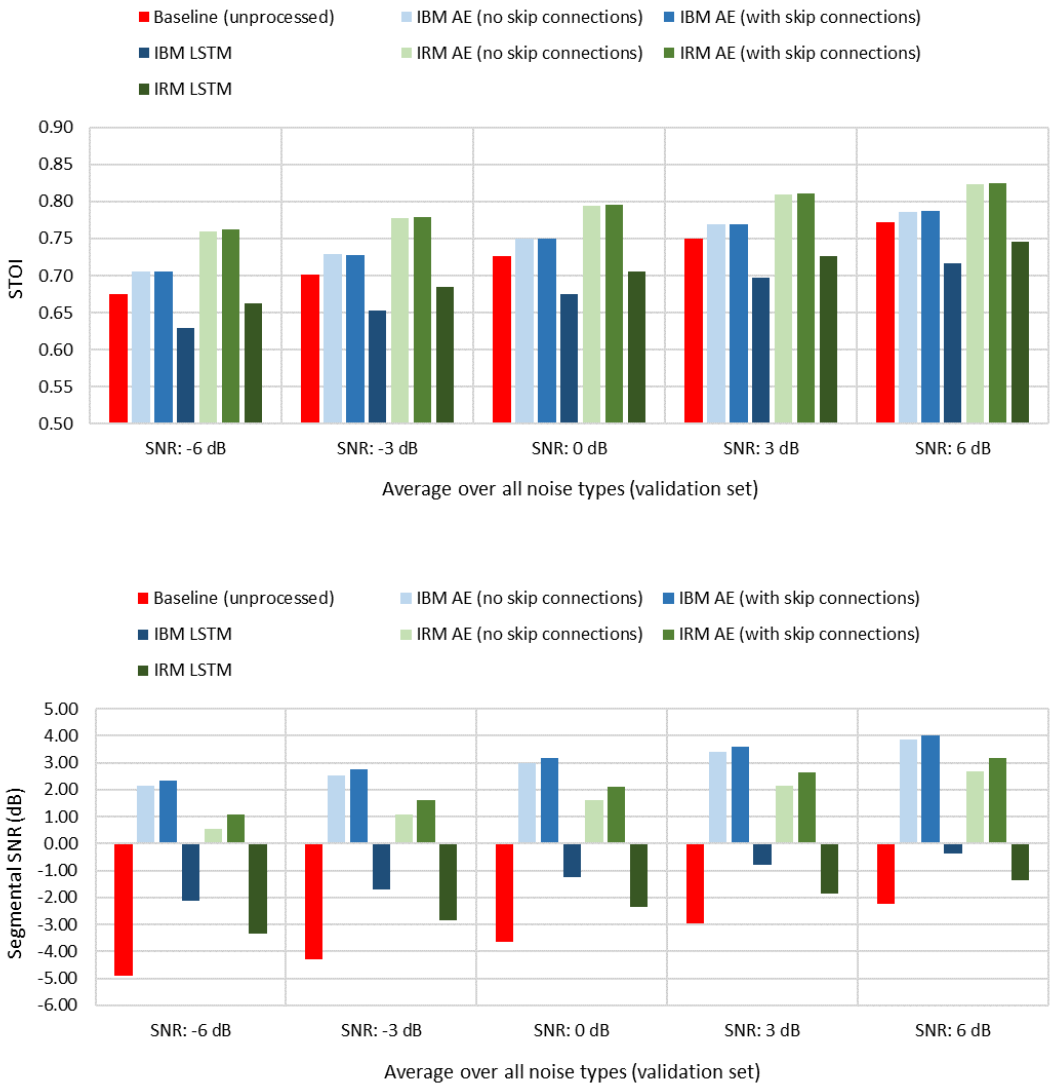


Figure B.5: STOI and SSNR results averaged over all noises for validation set

Average STOI and SSNR – test set.

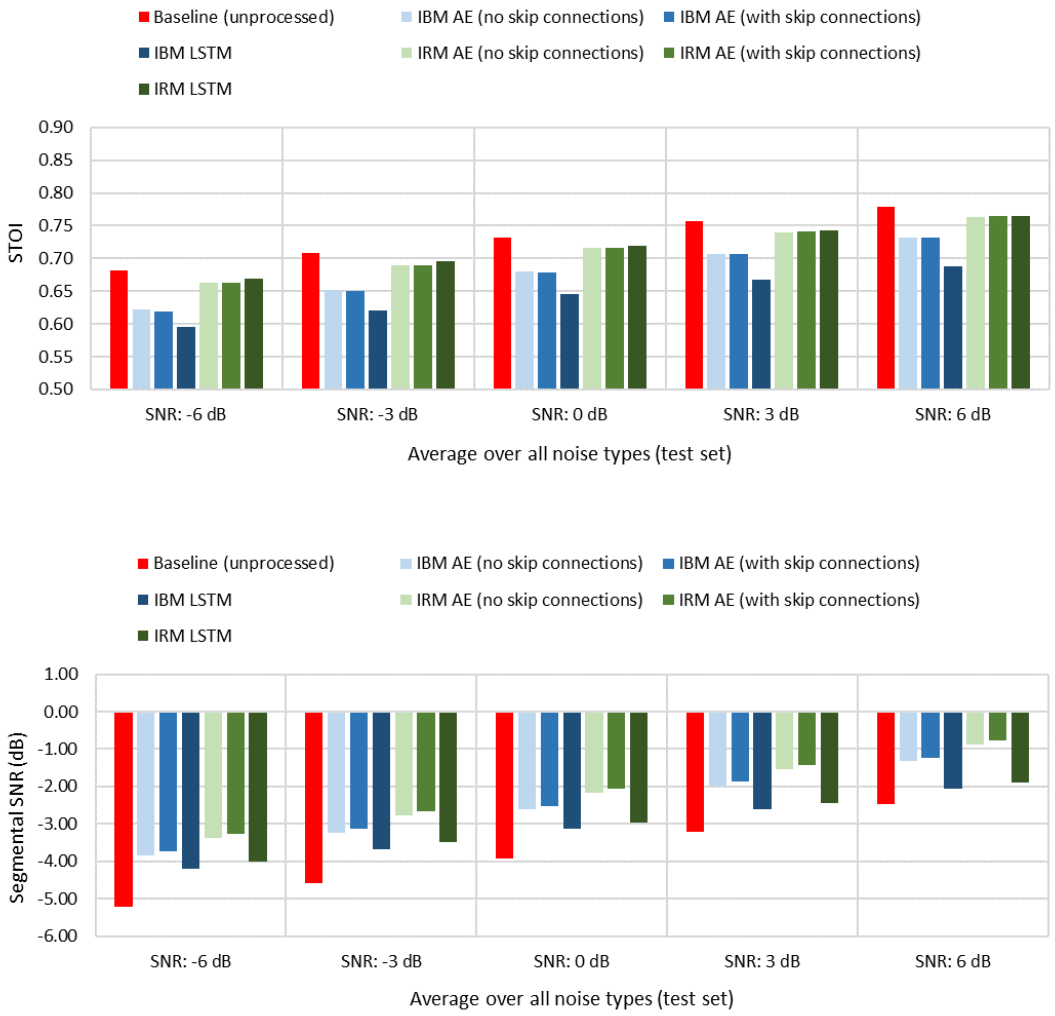


Figure B.6: STOI and SSNR results averaged over all noises for test set

STOI results – validation set.

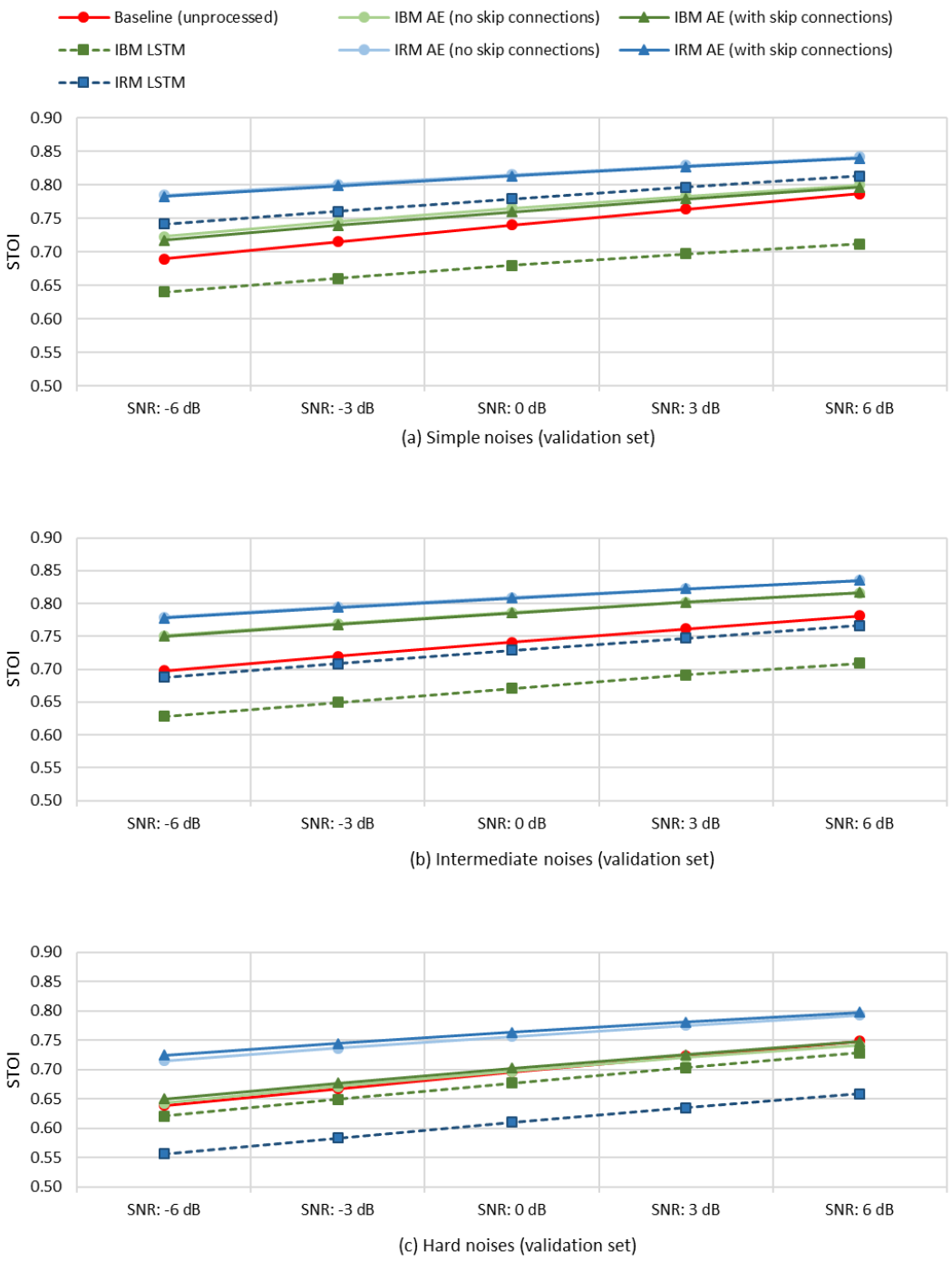


Figure B.7: STOI results for validation set

STOI results – test set.

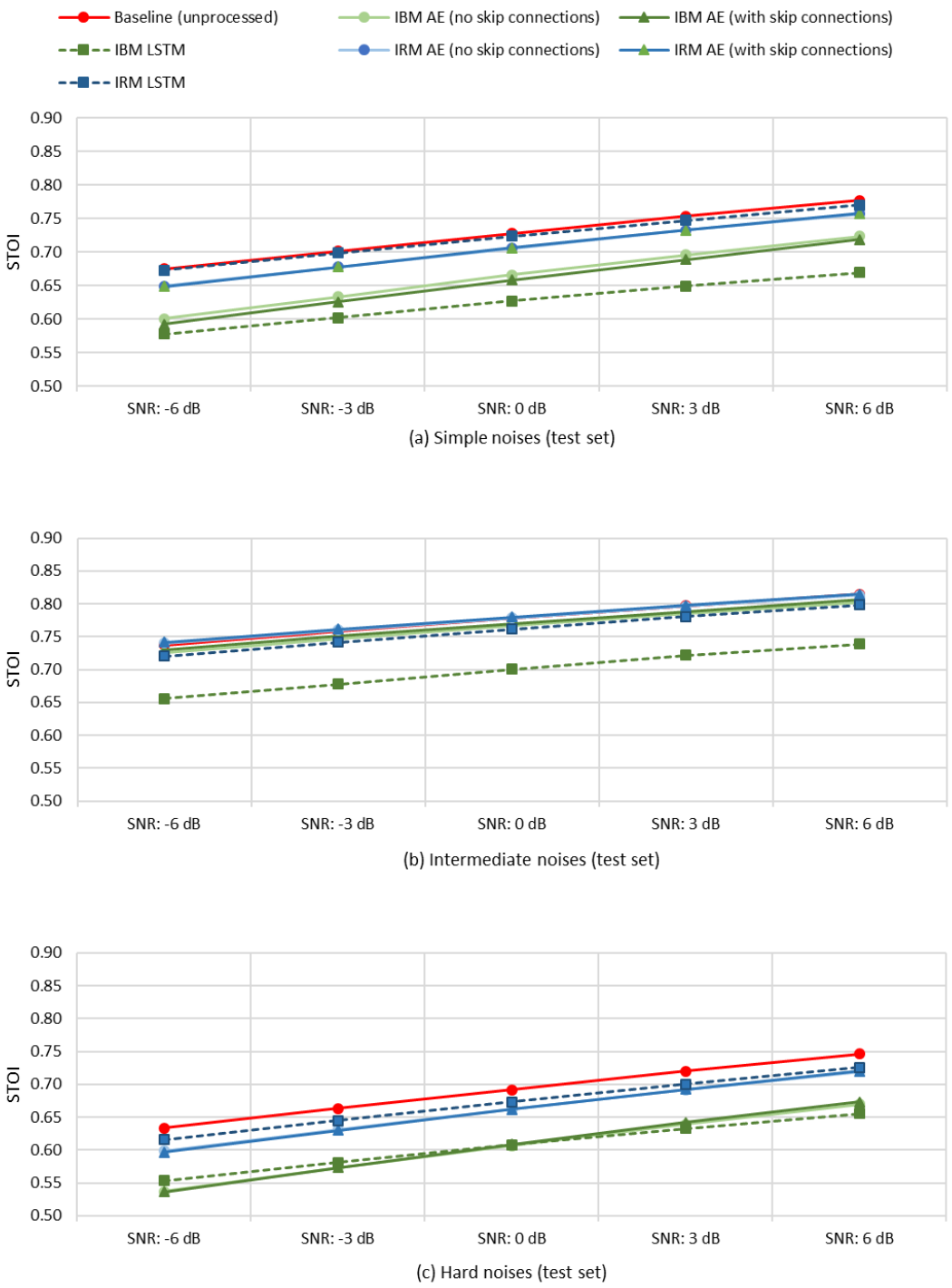


Figure B.8: STOI results for test set

SSNR results – validation set.

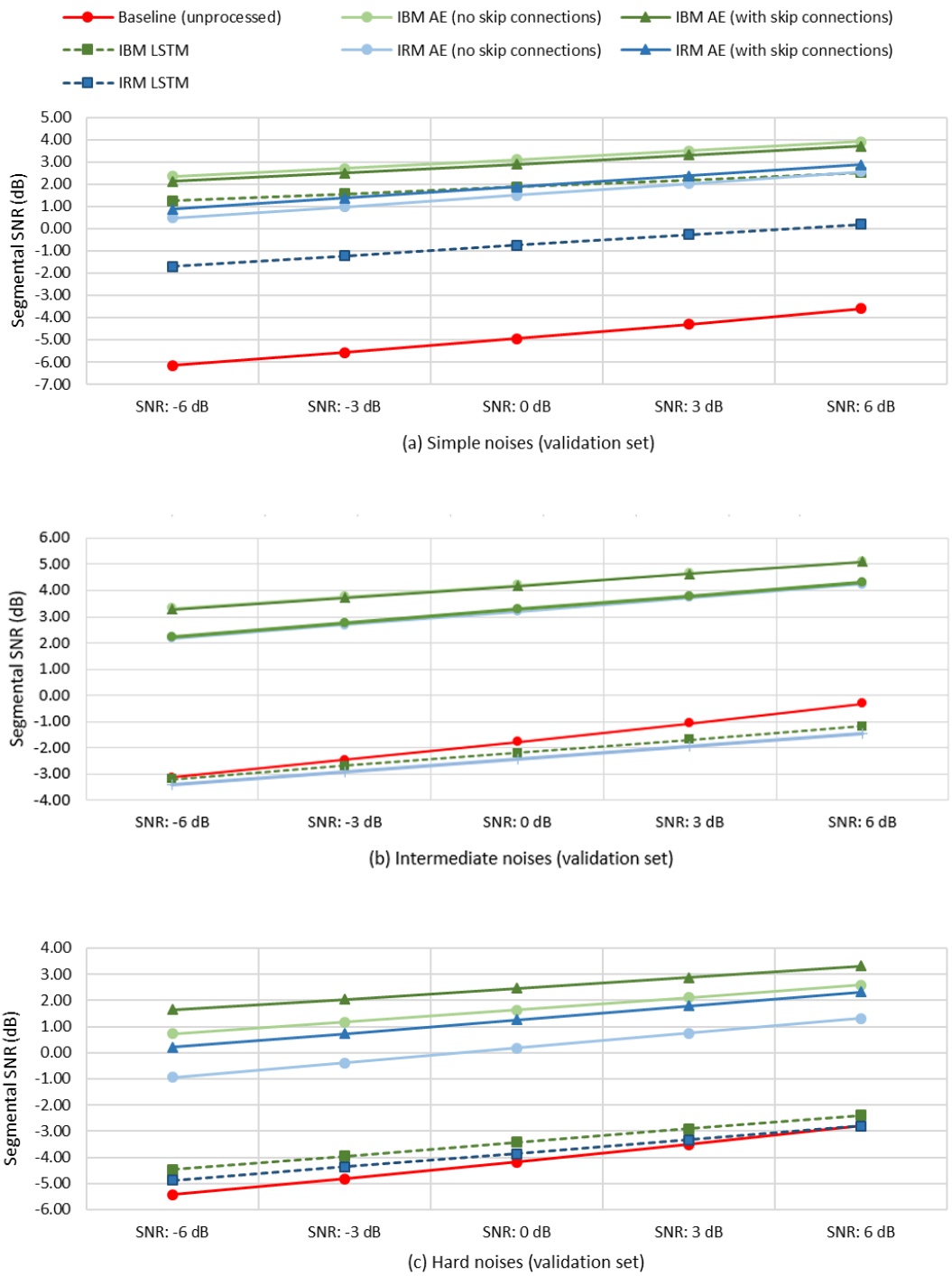


Figure B.9: SSNR results for validation set

SSNR results – test set.

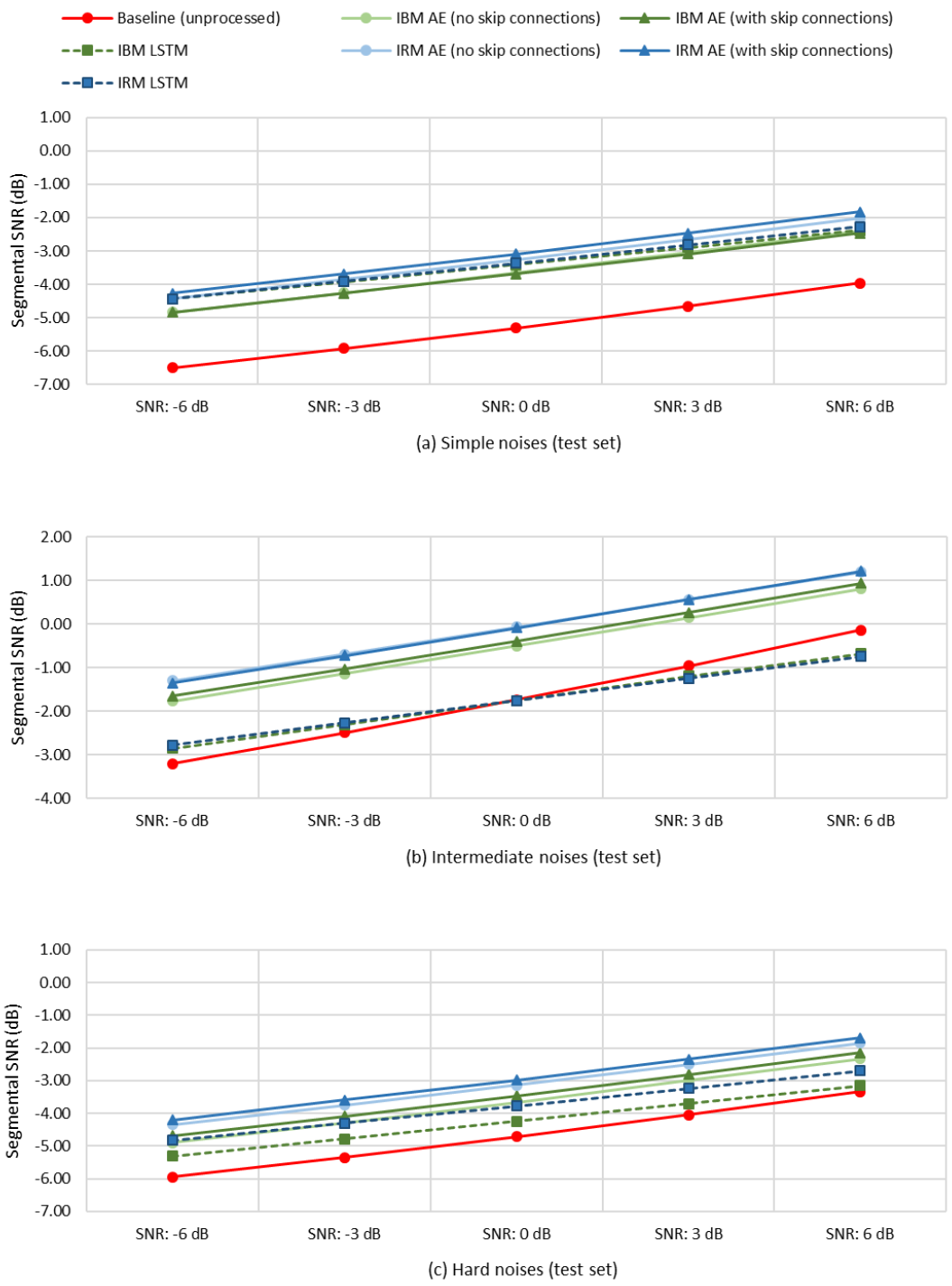


Figure B.10: SSNR results for test set

Average STOI and SSNR - validation set.

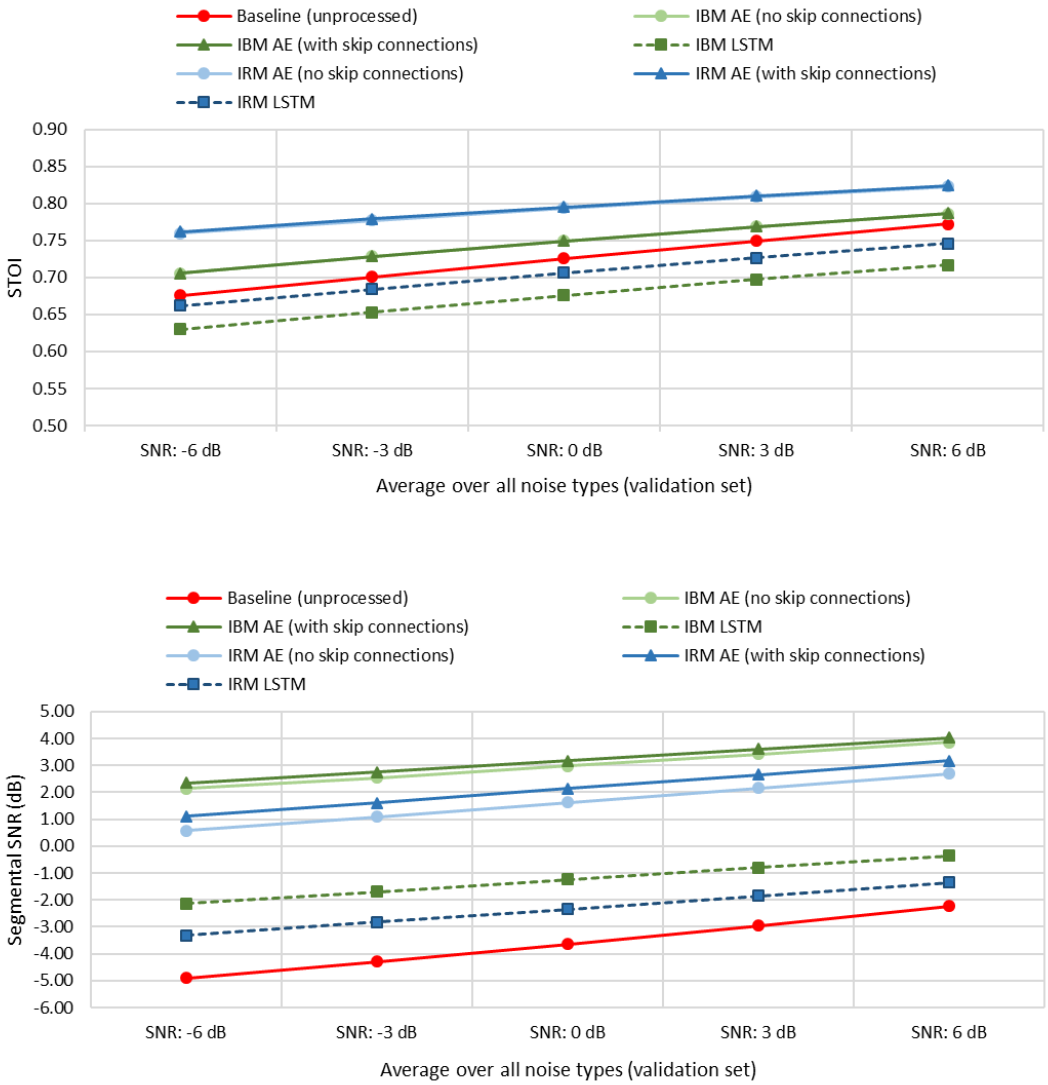


Figure B.11: STOI and SSNR results averaged over all noises for validation set

Average STOI and SSNR – test set.

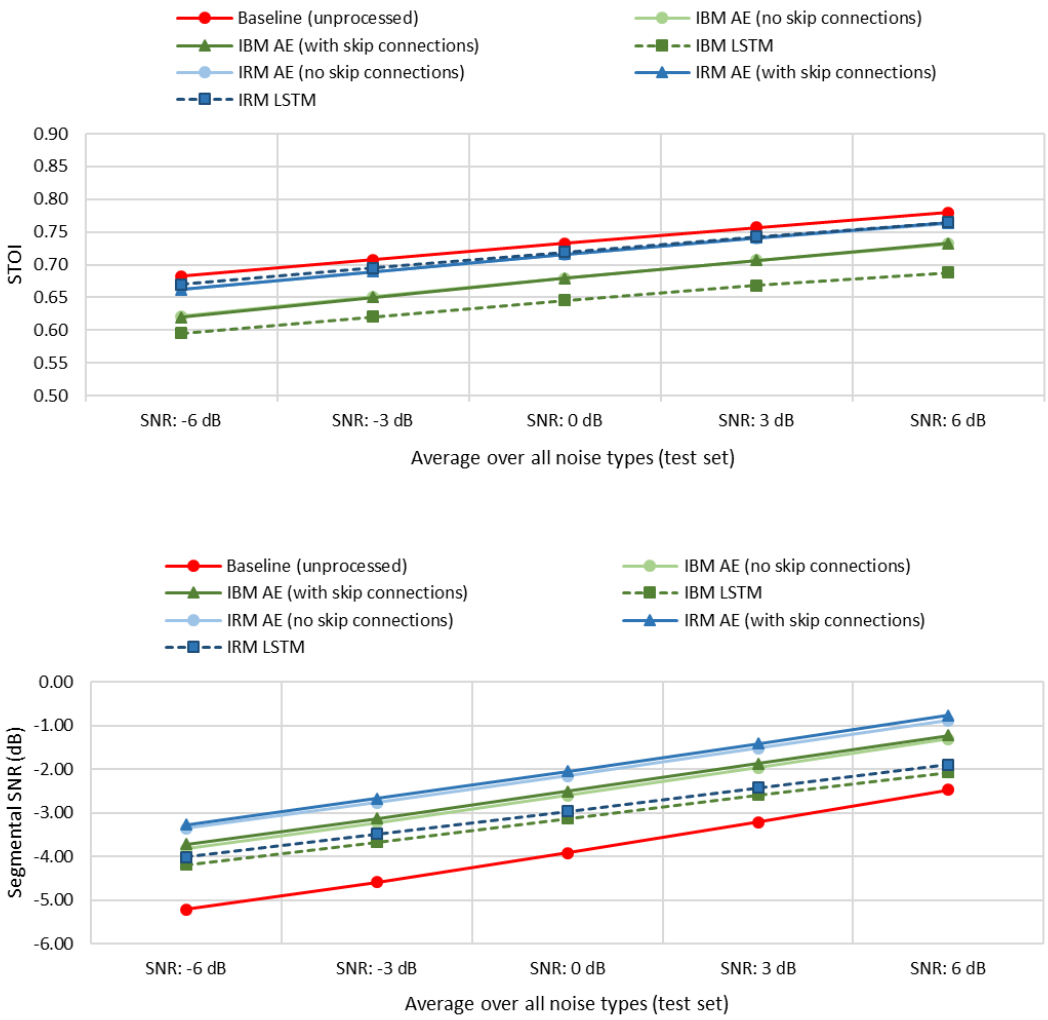


Figure B.12: STOI and SSNR results averaged over all noises for test set

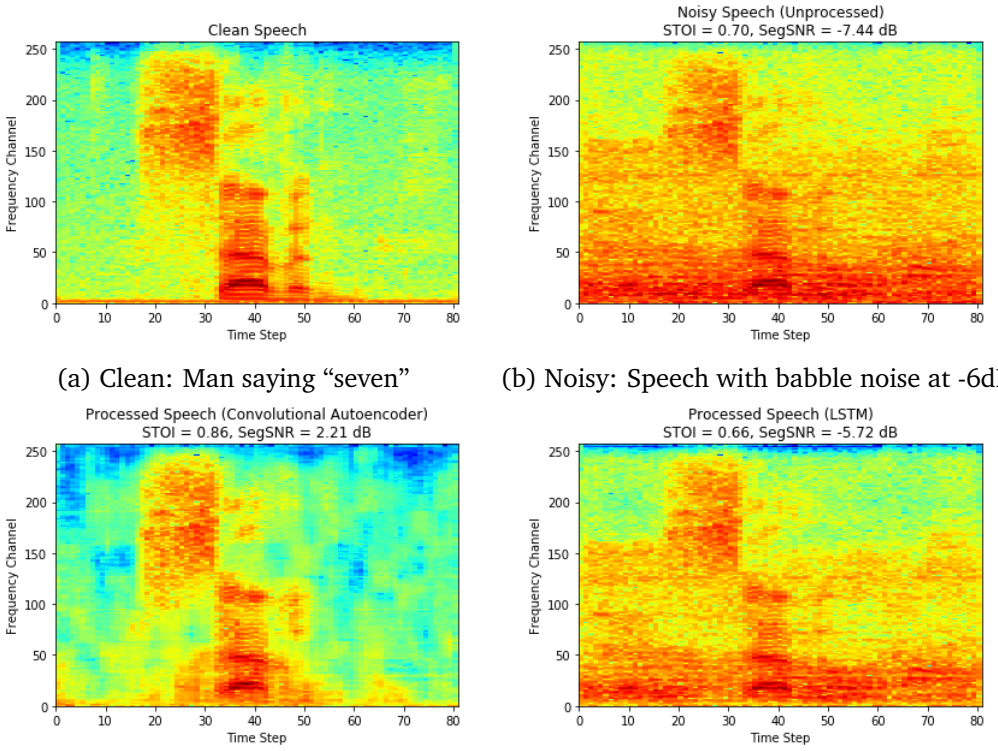
Appendix C

Validation Set Processed Examples

This appendix presents some examples of the clean, noisy and processed log-power spectra for results from the validation set to act as a comparison to the examples from the test set presented in section 4.

The first example, shown in figure C.1 is a man saying the word “seven” in the presence of babble noise at 0 dB. Figure C.1a shows the LPS of the clean speech and figure C.1b shows the LPS for the noisy, unprocessed speech, along with the baseline performance scores. Figures C.1c and C.1d show the LPS processed by the CAE and LSTM model, respectively, where the training target was the IRM and the CAE model included skip connections.

The CAE is very good at removing the babble noise, increasing the SSNR by over 9 dB. However, there are audible distortions in the resulting speech. These may be visible in figure C.1c, which shows that some T–F units were attenuated compared to the clean speech. The LSTM did not do a good job of removing the babble noise. The STOI decreased and the improvement in SSNR was modest compared to that achieved by the CAE.



(a) Clean: Man saying “seven”

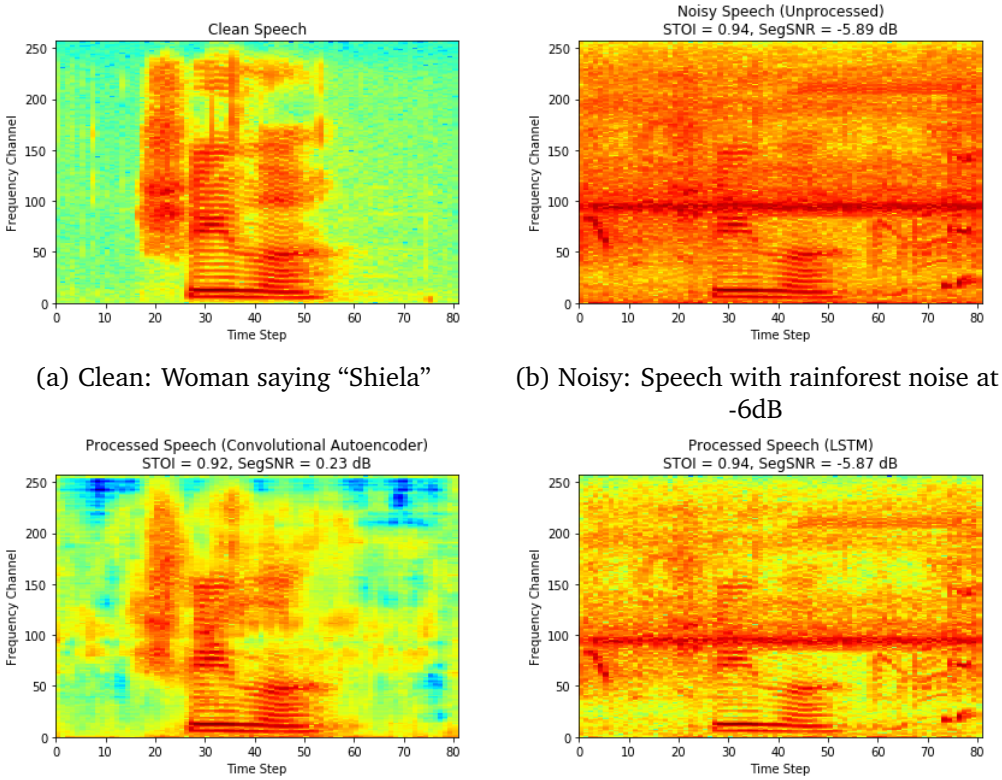
(b) Noisy: Speech with babble noise at -6dB

(c) Processed Speech from CAE

(d) Processed Speech from LSTM

Figure C.1: A comparison of clean, noisy and processed speech obtained from the CAE and LSTM models, using an example from the validation set

A further example taken from the validation set is a woman saying “Shiela” at -6 dB with rainforest noise consisting of birds and insects. The CAE does a very good job of removing the noise, almost completely eliminating it audibly. There is a visible remnant of the strong noise component at frequency channel 100 remaining in the processed LPS. The LSTM appears to do very little, offering no improvement in STOI and only 0.02 dB improvement in SSNR. Figure C.2d shows the the structure of the speech signal is mostly still obscured by the noise after processing. This observation is in agreement with the overall results, where for intermediate noises in the validation set, the LSTM is significantly worse than the CAE in terms of SSNR with the IRM as the training target.



(a) Clean: Woman saying “Shiela”

(b) Noisy: Speech with rainforest noise at -6dB

(c) Processed Speech from CAE

(d) Processed Speech from LSTM

Figure C.2: A comparison of clean, noisy and processed speech obtained from the CAE and LSTM models, using an example from the validation set

Bibliography

[1] P. Loizou, *Speech Enhancement: Theory and Practice, Second Edition*. Taylor & Francis, 2013.

[2] M. Woelfel and J. McDonough, *Distant Speech Recognition*. Wiley, 2009.

[3] S. R. Park and J. Lee, "A fully convolutional neural network for speech enhancement," in *INTERSPEECH*, 2017.

[4] N. Krishnamurthy and J. H. Hansen, "Babble noise: modeling, analysis, and applications," *IEEE transactions on audio, speech, and language processing*, vol. 17, no. 7, pp. 1394–1407, 2009.

[5] N. Saleem, M. Irfan, X. Chen, and M. Ali, "Deep neural network based supervised speech enhancement in speech-babble noise," in *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, 2018, pp. 871–874.

[6] J. H. McDermott, "The cocktail party problem," *Current Biology*, vol. 19, no. 22, pp. R1024–R1027, 2009.

[7] T. Goehring, M. Keshavarzi, R. P. Carlyon, and B. C. Moore, "Using recurrent neural networks to improve the perception of speech in non-stationary noise by people with cochlear implants," *The Journal of the Acoustical Society of America*, vol. 146, no. 1, pp. 705–718, 2019.

[8] P. Podder, T. Z. Khan, M. H. Khan, and M. M. Rahman, "Comparative performance analysis of hamming, hanning and blackman window," *International Journal of Computer Applications*, vol. 96, no. 18, 2014.

[9] F. Owens, *Signal Processing of Speech*, ser. New Electronics. Macmillan Education, Limited, 1993.

[10] R. Gao and R. Yan, "Non-stationary signal processing for bearing health monitor-

ing," *IJMR*, vol. 1, pp. 18–40, 01 2006.

[11] Y. Xu, J. Du, L. Dai, and C. Lee, "A regression approach to speech enhancement based on deep neural networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 1, pp. 7–19, 2015.

[12] E. A. Wan, A. T. Nelson, S. Katagiri *et al.*, "Networks for speech enhancement," *Handbook of neural networks for speech processing. Artech House, Boston, USA*, vol. 139, no. 1, p. 7, 1999.

[13] Y. Wang, K. Han, and D. Wang, "Exploring monaural features for classification-based speech segregation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 2, pp. 270–279, 2013.

[14] V. Mitra, H. Franco, R. M. Stern, J. Van Hout, L. Ferrer, M. Graciarena, W. Wang, D. Vergyri, A. Alwan, and J. H. Hansen, "Robust features in deep-learning-based speech recognition," in *New Era for Robust Speech Recognition*. Springer, 2017, pp. 187–217.

[15] M. R. Hasan, M. Jamil, M. Rahman *et al.*, "Speaker identification using mel frequency cepstral coefficients," *variations*, vol. 1, no. 4, 2004.

[16] S. Boll, "Suppression of acoustic noise in speech using spectral subtraction," *IEEE Transactions on acoustics, speech, and signal processing*, vol. 27, no. 2, pp. 113–120, 1979.

[17] N. Upadhyay and A. Karmakar, "Speech enhancement using spectral subtraction-type algorithms: A comparison and simulation study," *Procedia Computer Science*, vol. 54, pp. 574 – 584, 2015.

[18] Y. Hu, M. Bhatnagar, and P. Loizou, "A cross-correlation technique for enhancing speech corrupted with correlated noise," *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, 03 2001.

[19] Y. Lu and P. C. Loizou, "A geometric approach to spectral subtraction," *Speech communication*, vol. 50, no. 6, pp. 453–466, 2008.

[20] Jong-Won Seok and Keun-Sung Bae, "Reduction of musical noise in spectral subtraction method using subframe phase randomisation," *Electronics Letters*, vol. 35, no. 2, pp. 123–125, 1999.

[21] N. Upadhyay and R. K. Jaiswal, "Single channel speech enhancement: Using wiener

filtering with recursive noise estimation," *Procedia Computer Science*, vol. 84, pp. 22 – 30, 2016, proceeding of the Seventh International Conference on Intelligent Human Computer Interaction (IHCI 2015).

[22] M. A. A. El-Fattah, M. I. Dessouky, A. M. Abbas, S. M. Diab, E.-S. M. El-Rabaie, W. Al-Nuaimy, S. A. Alshebeili, and F. E. A. El-Samie, "Speech enhancement with an adaptive wiener filter," *International Journal of Speech Technology*, vol. 17, no. 1, pp. 53–64, 2014.

[23] L.-P. Yang and Q.-J. Fu, "Spectral subtraction-based speech enhancement for cochlear implant patients in background noise," *The Journal of the Acoustical Society of America*, vol. 117, no. 3, pp. 1001–1004, 2005.

[24] P. C. Loizou, *Speech enhancement: theory and practice*. CRC press, 2013.

[25] Y. Hu and P. Loizou, "Evaluation of objective quality measures for speech enhancement," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, pp. 229 – 238, 02 2008.

[26] P. C. Loizou, *Speech Quality Assessment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 623–654.

[27] Y. Tang, M. Cooke, and C. Valentini-Botinhao, "Evaluating the predictions of objective intelligibility metrics for modified and synthetic speech," *Computer Speech & Language*, vol. 35, pp. 73 – 92, 2016.

[28] J. H. Hansen and B. L. Pellom, "An effective quality evaluation protocol for speech enhancement algorithms," in *Fifth international conference on spoken language processing*, 1998.

[29] C. Taal, R. Hendriks, R. Heusdens, and J. Jensen, "A short-time objective intelligibility measure for time-frequency weighted noisy speech," 04 2010, pp. 4214 – 4217.

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

[31] K. Tan and D. Wang, "A convolutional recurrent neural network for real-time

speech enhancement." in *Interspeech*, 2018, pp. 3229–3233.

[32] Y. Li and D. Wang, "On the optimality of ideal binary time–frequency masks," *Speech Communication*, vol. 51, no. 3, pp. 230 – 239, 2009.

[33] A. Narayanan and D. Wang, "Ideal ratio mask estimation using deep neural networks for robust speech recognition," 10 2013, pp. 7092–7096.

[34] D. Williamson, Y. Wang, and D. Wang, "Complex ratio masking for joint enhancement of magnitude and phase," 03 2016, pp. 5220–5224.

[35] S. E. Eskimez, P. Soufleris, Z. Duan, and W. Heinzelman, "Front-end speech enhancement for commercial speaker verification systems," *Speech Communication*, vol. 99, pp. 101 – 113, 2018.

[36] N. Fan, J. Du, and L.-R. Dai, "A regression approach to binaural speech segregation via deep neural network," 10 2016, pp. 1–5.

[37] Y. Wang, A. Narayanan, and D. Wang, "On training targets for supervised speech separation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 12, pp. 1849–1858, 2014.

[38] D. Wang and Jae Lim, "The unimportance of phase in speech enhancement," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 30, no. 4, pp. 679–681, 1982.

[39] K. Paliwal, K. Wójcicki, and B. Shannon, "The importance of phase in speech enhancement," *speech communication*, vol. 53, no. 4, pp. 465–494, 2011.

[40] D. Wang and J. Chen, "Supervised speech separation based on deep learning: An overview," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 10, pp. 1702–1726, 2018.

[41] A. Sugiyama and R. Miyahara, "Phase randomization: A new paradigm for single-channel signal enhancement," 10 2013, pp. 7487–7491.

[42] W. S. Sarle, "Neural networks and statistical models," 1994.

[43] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Let a biogeography-based optimizer train your multi-layer perceptron," *Information Sciences*, vol. 269, pp. 188 – 209, 2014.

[44] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep

convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[46] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[47] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5907–5915.

[48] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," *arXiv preprint arXiv:1903.06733*, 2019.

[49] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998.

[50] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[51] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[52] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[53] T. Gao, J. Du, L.-R. Dai, and C.-H. Lee, "Snr-based progressive learning of deep neural network for speech enhancement." in *INTERSPEECH*, 2016, pp. 3713–3717.

[54] F. Weninger, H. Erdogan, S. Watanabe, E. Vincent, J. Le Roux, J. R. Hershey, and B. Schuller, "Speech enhancement with lstm recurrent neural networks and its application to noise-robust asr," in *International Conference on Latent Variable Analysis and Signal Separation*. Springer, 2015, pp. 91–99.

[55] L. Sun, J. Du, L.-R. Dai, and C.-H. Lee, "Multiple-target deep learning for lstm-rnn based speech enhancement," in *2017 Hands-free Speech Communications and Microphone Arrays (HSCMA)*. IEEE, 2017, pp. 136–140.

[56] S. Pascual, A. Bonafonte, and J. Serra, "Segan: Speech enhancement generative adversarial network," *arXiv preprint arXiv:1703.09452*, 2017.

[57] T. Kounovsky and J. Málek, "Single channel speech enhancement using convolutional neural network," 05 2017, pp. 1–5.

[58] S.-W. Fu, Y. Tsao, and X. Lu, "Snr-aware convolutional neural network modeling for speech enhancement." in *Interspeech*, 2016, pp. 3768–3772.

[59] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

[60] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, ser. MLSDA'14. New York, NY, USA: Association for Computing Machinery, 2014, p. 4–11.

[61] X. Lu, Y. Tsao, S. Matsuda, and C. Hori, "Speech enhancement based on deep denoising autoencoder." in *Interspeech*, vol. 2013, 2013, pp. 436–440.

[62] C. S. N. Pathirage, J. Li, L. Li, H. Hao, W. Liu, and P. Ni, "Structural damage identification based on autoencoder neural networks and deep learning," *Engineering Structures*, vol. 172, pp. 13 – 28, 2018.

[63] P. Y. Simard, D. Steinkraus, J. C. Platt *et al.*, "Best practices for convolutional neural networks applied to visual document analysis." in *Icdar*, vol. 3, no. 2003, 2003.

[64] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *2016 fourth international conference on 3D vision (3DV)*. IEEE, 2016, pp. 565–571.

[65] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[66] K. An, M. J. Kim, K. Teplansky, J. R. Green, T. F. Campbell, Y. Yunusova, D. Heitzman, and J. Wang, "Automatic early detection of amyotrophic lateral sclerosis from intelligible speech using convolutional neural networks." in *INTERSPEECH*, 2018, pp. 1913–1917.

[67] M. Wodzinski, A. Skalski, D. Hemmerling, J. Orozco-Arroyave, and E. Noeth, "Deep

learning approach to parkinson's disease detection using voice recordings and convolutional neural network dedicated to image classification," vol. 2019, 07 2019, pp. 717–720.

[68] I. Zafar, G. Tzanidou, R. Burton, N. Patel, and L. Araujo, *Hands-On Convolutional Neural Networks with TensorFlow: Solve computer vision problems with modeling in TensorFlow and Python*. Packt Publishing, 2018.

[69] S. Khan, H. Rahmani, S. Shah, M. Bennamoun, G. Medioni, and S. Dickinson, *A Guide to Convolutional Neural Networks for Computer Vision*, ser. Synthesis Lectures on Computer Vision. Morgan & Claypool Publishers, 2018.

[70] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[71] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in neural information processing systems*, 2016, pp. 2802–2810.

[72] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems*, 2018, pp. 6389–6399.

[73] N. Zheng, Y. Shi, W. Rong, and Y. Kang, "Effects of skip connections in cnn-based architectures for speech enhancement," *Journal of Signal Processing Systems*, vol. 92, no. 8, pp. 875–884, Aug 2020.

[74] A. Karpathy. The unreasonable effectiveness of recurrent neural networks. "Accessed on 28/07/2020". [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[75] T. Dutoit, C. Martín-Vide, and G. Pironkov, *Statistical Language and Speech Processing: 6th International Conference, SLSP 2018, Mons, Belgium, October 15–16, 2018, Proceedings*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2018.

[76] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[77] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[78] R. Karim. Animated rnn, lstm and gru. “Accessed on 13/08/2020”. [Online]. Available: https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

[79] Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, “An experimental study on speech enhancement based on deep neural networks,” *IEEE Signal processing letters*, vol. 21, no. 1, pp. 65–68, 2013.

[80] C. K. Reddy, E. Beyrami, J. Pool, R. Cutler, S. Srinivasan, and J. Gehrke, “A scalable noisy speech dataset and online subjective test framework,” *Proc. Interspeech 2019*, pp. 1816–1820, 2019.

[81] C. Valentini Botinhao, X. Wang, S. Takaki, and J. Yamagishi, “Speech enhancement for a noise-robust text-to-speech synthesis system using deep recurrent neural networks,” in *Proceedings of Interspeech 2016*, ser. Interspeech. International Speech Communication Association, Sep. 2016, pp. 352–356.

[82] G. Pirker, M. Wohlmayr, S. Petrik, and F. Pernkopf, “A pitch tracking corpus with evaluation on multipitch tracking scenario.” 01 2011, pp. 1509–1512.

[83] J. Thiemann, N. Ito, and E. Vincent, “DEMAND: a collection of multi-channel recordings of acoustic noise in diverse environments,” Jun. 2013, Supported by Inria under the Associate Team Program VERSAMUS.

[84] P. Warden, “Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition,” *ArXiv e-prints*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[85] A. Amidi and S. Amidi, “A detailed example of how to use data generators with keras,” 2018. [Online]. Available: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

[86] S.-H. Kim and Y.-H. Park, “Interaural coherence estimation for speech processing in reverberant environment,” *Applied Sciences*, vol. 10, no. 3, p. 769, 2020.

[87] R. L. Brennan, “Low power algorithms for hearing aid and embedded applications,” *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1097–1100, 2014.

[88] L. Tan and J. Jiang, *Digital Signal Processing: Fundamentals and Applications*. Elsevier Science, 2013.

[89] S. Xia, H. Li, and X. Zhang, “Using optimal ratio mask as training target for

supervised speech separation," 09 2017.

[90] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.