The Egyptian e-Learning University

Faculty of Computers and Information Technology

**An Artificial Intelligence Approach to Speech Enhancement**

Graduation Project Documentation

Team members:

Abdelrahman Hussien Ahmed Elshouky 19-01194

Mazen Mostafa Ahmed Elhammadi 19-01197

Fouad Salah Sobhy Said 20-01073

Mahmoud Mazen Salman Mohamed 19-00727

Zeyad Mohamed Fouad Saleh 20-00581

Amr Mohamed Mahmoud Taha 18-00500

Zeyad Hany Mohamed Ali 20-01035

Supervised by:

Dr, Enas Elgeldawi

TA.Eng. Mohammed Saeed

**Catalog**

**[1 Introduction 2](#_Toc5000)**

[1.1 Problem Formulation 3](#_Toc4130)

[1.2 Related Work 4](#_Toc8833)

**[2 Deep Learning 8](#_Toc11468)**

[2.1 Artificial Neuros 9](#_Toc31719)

[2.2 Feed-Forward Network 10](#_Toc28178)

[2.3 Training neural network models 12](#_Toc15563)

[2. 4 Network Topologies 21](#_Toc21991)

**[3 Spectro-TemporaI Signal Representation 27](#_Toc23534)**

[3.1 Spectrogram 27](#_Toc29237)

[3.2 Mel-Spectrum 50](#_Toc8447)

**[4 Evaluation Criteria 51](#_Toc16307)**

[4.1 PESQ 52](#_Toc8848)

[4.2 STΟΙ 53](#_Toc15484)

[4.3 Speech Accuracy Score 54](#_Toc26843)

[4.4 Signal-to-Noise Ratio 55](#_Toc28745)

**[5 Time-Frequency Masks 56](#_Toc912)**

[5.1 Ideal Binary Mask 56](#_Toc8052)

[5.2 Ratio Mask 57](#_Toc4937)

[5.3 Phase Sensitive Filter 57](#_Toc18233)

[5.4 Comparison 57](#_Toc2895)

**[6 Experi menta I Setup 60](#_Toc13513)**

[6.1 Proposed Framework 60](#_Toc22231)

[6.2 Importing libraries 62](#_Toc22060)

[6.3 Data Set 63](#_Toc9443)

[6.3 Normalization and Visualization 64](#_Toc6922)

[6.4 Training and testing data 64](#_Toc30823)

[6.5 Evolution of the loss during the training 66](#_Toc11464)

**[7 Run and Resu Its 68](#_Toc18024)**

**[7.1.1 Comparison of Features 65](#_Toc14020)**

**[7.1.1 Comparison of Layer Sizes 67](#_Toc13213)**

[7.2 Bidirectional LSTMs 69](#_Toc9893)

**[7.3 Weighted Loss Function 71](#_Toc4681)**

**[7.4 Matched Training 73](#_Toc15579)**

[7.5 Convolutional Neural Networks 74](#_Toc408)

**[Conclusion and future work 76](#_Toc32564)**

**[future work 76](#_Toc5976)**

[Sound Enhancement: 76](#_Toc32433)

[Video Enhancement: 77](#_Toc20808)

**[Reference 78](#_Toc14288)**

1 Introduction

Speech enhancement has been an important research topic in the field of signal processing for the last few decades. It comprises a group of methods that aims to improve the quality and intelligibility of speech signals. While in the early days of digital speech communication research, the field of applications was dominated by military purposes, the emergence of mobile communication at the end of the last century introduced it into the every day life of many people. With the rise of mobile phones, telecommunication systems had to cope with a variety of noise scenarios since phones were now extensively used in situations such as automobile environments, public transportation, restaurants or on busy streets, as opposed to a landline were the interfering noise is typically less severe. While the suppression of unwanted background noise may be the most evident goal of speech enhancement, it also includes tasks such as dereverberation, bandwidth extension or packet loss concealment. However, this thesis is focused on the reduction of additive noise and more precisely the application in smart speakers, wireless loudspeakers with an integrated virtual assistant offering handsfree interaction via voice commands. Speech enhancement for the use in smart speakers poses a special challenge since the dialogue system is expected to function properly independent from the placement of the device in relation to the location of the user. Since conventional algorithms are reaching their limits when it comes to severe noise conditions and unstationary noise types such as babble noise, the goal of this thesis is to make use of the recent advantages in the field of machine learning and in particular deep learning and apply them to the problem of speech enhancement and noise reduction.

The thesis is structured as follows. Chapter 2 gives a brief introduction to the key concepts of artificial neural networks and deep learning. In chapter 3 basic signal processing techniques for the spectral representation of speech signals are explained followed by an overview of objective evaluation criteria used to assess the performance of speech enhancement algorithms in chapter 4. Chapter 5 describes different time-frequency masks as used in the field of computational auditory scene analysis (CASA) and their performance in terms of noise reduction is evaluated and compared. The proposed speech enhancement framework and the experimental setup is described in chapter 6 followed by a detailed description of the experimental results in chapter 7. Finally chapter 8 concludes the thesis by summarizing the results as well as prospects for future potential and limits of the proposed speech enhancement system.

1.1 Problem Formulation

The principal goal of speech enhancement or simply noise reduction is to improve perceived quality and intelligibility of noisy speech signals. If solely monaural signals are considered, the problem can be more precisely defined as Single-Channel Speech Enhancement. In this context, noisy means that the desired speech signal is degraded by additive noise and/or other effects such as nonlinear distortions or reverberation.

The focus of this work lies on additive noise sources which can be environmental sounds such as wind, traffic or additional speech, or noise induced by the transmission system

e.g. quantization noise. Therefore, we define the noisy signal x(t) as a sum of the clean speech signal s(t) and a the noise signal n(t).

The goal in speech enhancement is then to predict an estimate S(t) of the clean signal, given an observation of the noisy signal (see figure 1.1).

|  |  |
| --- | --- |
|  | Speech Enhancement Algorithm |
|  |

S(t)

Figure 1.1 — A speech enhancement algorithm outputs an estimate S(t) of the clean speech signal, given the noisy signal x(t) as an input

1.2 Related Work

At the present day most noise reduction and speech enhancement systems as used in mobile communications, hearing aids or speech recognition technologies, are based on signal processing techniques exploiting the statistical properties of the anticipated signals. Generally this means that simplified assumptions are made about the properties of the speech and even more of the noise sources.

Basic spectral subtraction algorithms obtain a noise estimate from segments of the signal where no speech is present. One critical assumption made here is that the properties of the interfering signal will not change as much from one frame to another, so that the knowledge obtained from the past few frames can be used to separate the speech and the noise signal [1], [2], [3], [4], [5]. However, if the signal does change its properties this leads to an perceptually unpleasant effect known as musical noise and of course if the changes are particularly drastic the suppression mechanism will lose its effect completely. While other adaptive methods such as wiener filtering [6], minimum mean-square error short-time spectral amplitude (MMSE-STSA) [7] or maximum-a-posteriori (MAP) estimation [8] are able to reduce the effect of musical noise by incorporating assumptions about the statistics of the speech signal, they still require complicated methods to estimate the power spectral density (PDS) of the noise signal [9], [10].

1.2 Related Work

So called model-based approaches incorporate a priori information about the spectrotemporal properties of speech and noise signals. While hidden Markov models (HMMs) [11], [121 can be used to include phonetic information to estimate the speech source, typical supervised speech enhancement systems such as methods based on nonnegative matrix factorization (NMF) [13], [14], [15], are trained on different types of noise and speech models.

In recent years there has been an increased interest in deep learning and several studies have been investigating the use of purely data-driven approaches to speech enhancement and source separation. In particular after the publications on Deep Belief Nets (DBN) and greedy layer wise pre-training by Geoffrey Hinton [16] and Yoshua Bengio [17] several authors have applied these concepts on speech enhancement and source separation.

Already in 1989 a neural network speech enhancement technique was proposed by Shin'ichi Tamura [18]. In particular the author used a neural network with 60 hidden units for each of the four layers, the input was a 60-point long noisy speech waveform sampled at 12 kHz and the output was the corresponding clean waveform of the same length. While the study focussed more on the analysis of the trained network and no actual numbers on the performance of the system were published, the authors reported that 'the noise reduction method using a neural network was comparable or better than the conventional spectral subtraction method

A few years later Fei Xie and Dirk van Compernolle [19] proposed a family of nonlinear spectral estimators implemented by a multilayer perceptron neural network. They used parameters describing the speech and noise distributions as extra inputs to the network and investigated the performance of a recognition system when using the nonlinear spectral estimator as its front end. In terms of the recognition rate an average improvement of around 10%, compared to the generalized spectral subtraction method, was reported

However, during that time neural networks were considered unpractical due to their computational complexity and advances were made in other fields of machine learning which led to a decline in popularity which lasted until 2007. At that time came a new wave of neural network research on grounds of the above mentioned publication by Geoffrey Hinton as well as the increase of computational power and dataset sizes [20, p. 17-19].

After deep learning was successfully used for noise robust speech recognition [21], [22], [23] as well as binary speech coding [24], several studies on neural network speech enhancement have followed

Based on [25], Lu et al. [26] built a deep denoising autoencoder (DAE) using greedy layerwise pretraining plus fine tuning for noise reduction and speech enhancement. Patches of mel frequency power spectra using 40 bands were used as features and the effects of the trainings data size, the number of hidden layers and the depth of the network were investigated. An inverse transform was performed to synthesize the restored speech and to compare the results with the clean reference signals and the results of a baseline algorithm, they were resynthesized from mel sprectra in the same way. While the depth of the DAE did not drastically effect the results, increasing the training data set size and hidden layer size showed general improvement in terms of noise reduction, distortion as well as perceptual evaluation of speech quality (PESQ) measure. Furthermore, an optimal patch size of 11 frames in terms of speech enhancement performance was reported. In comparison to an MMSE based algorithm (IMCRA) the P ESQ for the enhanced speech was up to 1.1 higher, but the effect on noise reduction in dB improved only for some of the test scenarios. Moreover, the evaluation only employed noise that was included in the training data and therefore the network's performance in terms of generalization was not determined

In [27] deep neural networks (DNN) were used as part of an algorithm to improve speech recognition in noise for hearing-impared listeners. From a 65-channel gammatone filterbank output features were extracted to train multiple subband DNNs. The features included amplitude modulation spectrogram, perceptual linear prediction and mel frequency cepstral coefficients (MFCC) as well as additional delta features. Ideal binary masks were used as training targets and the subband DNNs were used to classify the corresponding time-frequency points as either speech or non-speech. The authors reported increased intelligibility for speech-shaped noise and babble noise scenarios. The results were more distinct for hearing impared listeners, reaching up to 70% of improvement in intelligibility.

A different approach was used in [28]. Here a DAE is used to obtain the power spectrum estimate of clean speech and the a priori signal-to-noise (SNR) ratio is estimated using a posteriori SNR controlled recursive averaging (PCRA). Finally the enhanced speech is obtained by frequency domain wiener filtering. The autoencoder has one layer consisting of 300 hidden units and uses a frequency dependend linear weighting function to improve the perceptual quality. The training process was similar to [26], using unsupervised pre-training and fine-tuning. The method was compared to frequency domain wiener filter with Decision-Directed approach for SNR estimation and achieved similar results in terms of SNR and distortion and slightly better results in terms of P ESQ.

Huang et al. [29] studied deep learning for monaural speech separation and proposed joint optimization of the deep learning models with an extra masking layer to enforce a reconstruction constraint. They used DNN and recurrent neural network (RNN) models with 2 layers of 150 hidden units and evaluated conventional as well as log-mel spectra. The proposed models achieved about 3.8 to 4.9 dB Source to Interference Ratio (SIR) gain compared to NMF models and maintained better source-to-distortion (SDR) and source-to-artifact ratio (SAR)

Ding Liu, Paris Smaragdis and Minje Kim [30] presented various experiments using a deep learning model for speech denoising. In contrast to earlier studies, they did not apply any pretraining step and used backpropagation to estimate the model parameters. Compared to an NMF model trained on the same dataset as the neural network (NN) models, the proposed method achieved significant improvements in terms of SDR,

1.2 Related Work

SIR, SAR and short-time objective intelligibility measure (ST01). Furthermore, the experiments showed that this method is adequately robust to unseen mixing situations. Comparison of network topologies showed that the number of hidden layers and units was not crucial although the best results were achieved with a single layer of 2000 units. Additionally, it was shown that rectified linear units (ReLu) were superior to other common activation functions.

In [31] the authors evaluated different training targets for supervised speech enhancement. A DNN with three hidden layers of 1024 rectified linear units was trained using backpropagation without unsupervised pre-training. They compared different spectral masks, including ideal binary mask (IBM), target binary mask (TBM) and ideal ratio mask (IRM) as well as the short-time Fourier transform spectral magnitude, its corresponding mask (FFT-mask) and direct estimation of the gammatone frequency power spectrum. It was reported that when testing with different noise scenarios, softmasks generally produce the best results in terms of STOI and P ESQ

Xu et. al [32], [33] have been further investingating the use of restricted boltzmann machines (RBM) and DBNs in combination with large scale training to learn the mapping between noisy and clean log power spectral features. To build the model a stack of multiple RBMs was pre-trained layer-by-layer with noisy speech and then fine-tuned with noisy and clean speech pairs. Sigmoid activation was used for the hidden units and linear activation for the output layer. In addition global variance equalization was incorporated as a post-processing step. For the experiments in [33] a large training set was built, which consisted of about 2500 hours of training data including 4620 clean speech utterances and 104 different noise types. Different context sizes up to 13 frames were compared in terms of SNR and the training set size was evaluated in terms of P ESQ. Furthermore, the generalization capabilities of the trained network were investigated using different unseen noise types. Similar to the results in [26] no significant improvement for context sizes over 11 frames was reported and the best results were achieved using a training set size of 625h although the difference to training with 100h was only marginal It was stated that the combination of pre-training, variance normalization and dropout improved the performance for unseen noise types about 0.25 dB in average. Furthermore, in a subjective listening test 78% of the test candidates preferred the DNN-based enhanced speech to that obtained using a conventional approach (LogMMSE).

In a later study [341 the use of multi-objective learning and IBM-based post-processing was investigated. As a constraint in the objective function, MFCCs and IBM were used as additional training targets and in combination with the post-processing step about 0.2 P ESQ and 0.03 STOI improvements were obtained on average.

Several studies have investigated the use of deep learning in the context of speech separation, which we will mention here because of the similarities to mask-based methods for noise reduction.

In [35] the authors studied discriminatively trained recurrent neural networks for singlechannel speech separation. They used ideal ratio masks as training targets and compared DNNs and RNNs in particular long short-term memory (LSTM), using different network topologies. The best results were achieved using 2 layers of 256 LSTMs, which produced an average SDR of 12.2 dB in comparison to 10.46 dB when using a 3-layer DNN with 1024 hidden units. Furthermore, DFT spectra and mel spectra with both 100 and 40 bands were compared as input features. It was reported that in the DFT power spectrum domain, an 11.4 dB average SDR was obtained while in the mel magnitude domain with 100 bands 12.8 was achieved

In an additional paper [361, the use of bidirectional LSTMs was investigated and the framework was extended by an automatic speech recognition (ASR) system. It was shown that phase sensitive filters (PSF) generally produce better results than IBM and 'RM. Bidirectional LSTMs achieved an average SDR that was up to 0.5 dB higher compared to the results obtained with LSTMs and furthermore a phase sensitive cost function was used, which further improved the results by more than 1.5 dB.

Finally convolutional neural networks (CNNs), which are playing a big role in state of the art image classification results [37], have been used for speech enhancement as well In [38] the authors investigated the utilization of CNNs for the estimation of soft masks and log power spectra, furthermore the generalization performance for different languages was examined. They used a network consisting of two convolutional layers followed by to fully connected layers with 1024 hidden units. It was shown that when using log power spectra as targets, CNNs produced results that outperformed conventional DNNs by up to 0.14 P ESQ and 0.6 least significant difference (LSD). Additionally, it was reported that the network trained with a multilingual dataset was in average 3% better in terms of P ESQ than the ones trained with monolingual data.

In summary there has been a growing interest in deep learning and its application to speech enhancement in particular during the past few years. Different neural network architectures as well as signal analysis and synthesis methods have been studied which led to promising results. In the context of this work we will compare some of the mentioned ideas and built a framework which will combine the most promising approaches. The following will give a short introduction to deep learning and its basic concepts.

2 Deep Learning

The term deep learning describes a category of machine learning algorithms based on artificial neural networks (ANNs). In recent years, deep learning has been successfully applied to many tasks in the field of artificial intelligence, such as automatic speech recognition (ASR) [39] and dialogue systems [40], image classification [3T], handwriting recognition [41] or machine translation [42]. ANNs are computational models which apply an arbitrary function on a given input ᵡ to obtain a desired output y



(2.1)

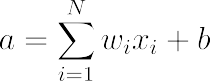
The model has a set of parameters (θ), which are gradually adjusted during the training phase so that the outputs y are as close as possible to the desired output values t of a given training set (ᵡ,t)

In the following, the basic concepts of deep learning will be coarsely described by first introducing the artificial neuron model as the basic building block of artificial neural networks in chapter 2.1. Then in chapter 2.2 the structure and the mathematical description of feed forward networks will be explained followed by an explanation of the basic training process in chapter 2.3. Finally, in 2.4 three basic network topologies for deep learning, namely autoencoders, convolutional networks and recurrent networks are described.

**2.1 Artificial Neuros**

ANNs are inspired by the biological nervous system which processes information through networks of neurons connected by synapses. Typically a neural cell consists of so called dendrites, which are able to receive electrochemical stimuli from other neurons, a cell body (soma) and an axon, which transfers a processed signal to the dendrites of other neurons in the network |43, p.35).

An artificial neuron, is a significantly simplified mathematical model of the way neurons receive, process and transmit information. In its general form, as depicted in figure 2.1, it is defined by a linear combination of N input values x, and a subsequent transformation of the resulting activation a



Where g is an arbitrary transfer function also referred to as the activation function the parameters wi and b are known as and weights and biases [43, p.72]

Using vector notation equations 2.2 and 2.3 can be compactly expressed as

y=g(wx+b)

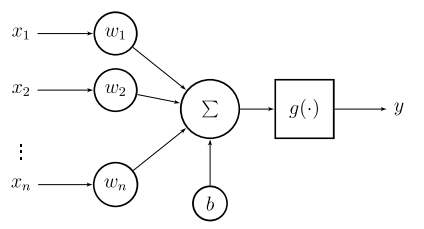


Figure 2.1 — Artificial neuron: The inputs are multiplied by weights w, and the linear combination is then transformed by an activation function y

Commonly used activation functions are shown in figure 2.2 and include logistic sigmoid, hyperbolic tangent (tanh), rectified linear unit ReLu or the softmax function, which is typically used for classification.

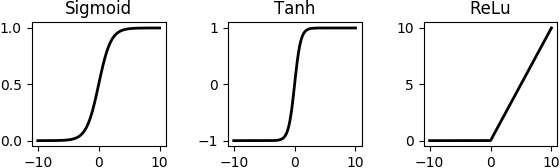


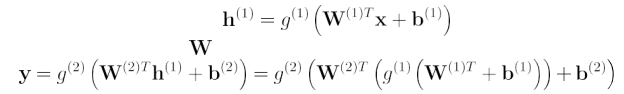
Figure 2.2 — Common activation functions. Sigmoid (left), hyperbolic tangent (middle) and rectified linear unit (fight).

In the following it will described how artificial neurons are used as building blocks for neural network models.

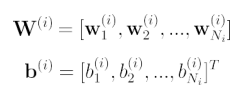
**2.2 Feed-Forward Network**

The most basic ANN structure is a feed forward network, sometimes also called multilayer perceptron (M LP), and has been a common choice for classification tasks. Multiple artificial neurons are combined to form a network organized in so called layers.

Figure 2.3 shows a simple feed forward neural network consisting of one hidden layer and an output layer. Each layer is defined by a weight matrix Wi and a bias vector bi combining the parameters for each of the neurons inside the layer



where the weight matrix is formed by the weight vectors corresponding to the neurons inside the associated layer and the bias vector h by the corresponding bias values accordingly.



In general, the outputs from the current layer are treated as the input for the subsequent layer

input layer x hidden layer h(1) output layer y

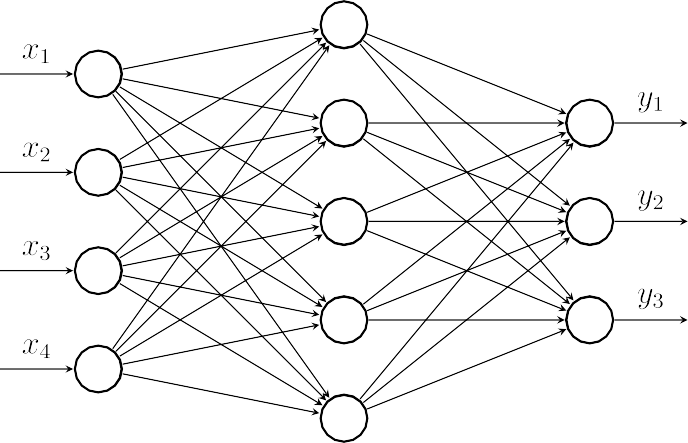


Figure 2.3 — Simple feed forward neural network with one hidden layer. Each of the connections represents an artificial neuron consisting of a weight vector w a bias value b and an activation function g.

The number of hidden layers connotes the depth of the corresponding network, whereby the term Deep Neural Network (DNN) implies an artificial neural network spanning multiple hidden layers. Figure 2.4 shows a deep neural network with three hidden layers, each consisting of n=5 neurons or hidden units

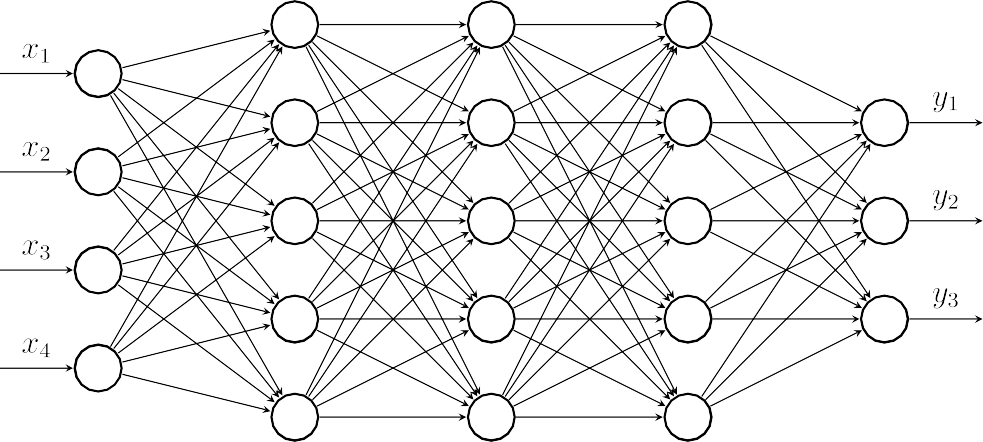


Figure 2.4 — Deep feed forward neural network with three hidden layers

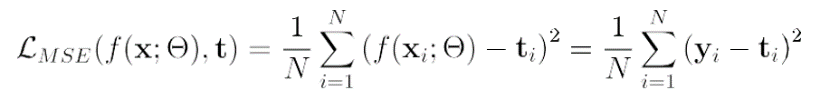
**2.3 Training neural network models**

In order to find the parameters of the model function fθ, the network is presented with data consisting of input and output pairs. This process is called learning or training. A cost function which measures the divergence between the true output and the network's prediction is used to evaluate the model. This will be briefly described in 2.3.1. Then the parameters are adjusted by a gradient based optimization algorithm as described in 2.3.2. In chapter 2.3.3 the backpropagation algorithm is introduced, which is needed to compute the gradient of the loss function with respect to the weights. Finally, regularization for deep neural networks will be briefly discussed in chapter 2.3.4.

**2.3.1 Cost Function**

During training, we try to minimize a loss function t) with respect to the parameters θ which is evaluated on a set of N training pairs xiti i= 1, .., N, consisting of feature and target vectors for a given observation *i* [44, 5.2] (2.10)

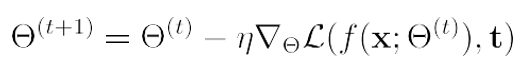
Note that the targets can be vectors of class labels or probabilities for classification as well as any kind of vectors, matrices or tensors for regression problems. A common loss function is the mean—squared error (MSE} [44,1 5 5), but depending on the context other objectives measures such as the **Kullback-Leibler divergence** or **cross-entropy losses** are popular as well. All loss functions measure the divergence of the predicted outputs from the true outputs t. In case of the MSE we measure the squared distances over all observations, and take the mean value:



**2.3.2 Parameter Optimization**

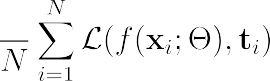
To find a solution to equation 2.10 i.e. to find the optimum parameters (or a given network an optimization algorithm is used. Typically optimization algorithms use information from the gradient of the cost function to guide the weights into the direction of a minimum. In the following two variants of the gradient descent algorithm are introduced the **Stochastic gradient descend** (SGD) and the *Adam* optimizer.

**Stochastic gradient Descent** A common optimization method is SGD [20 which is technically an approximation of the gradient descent optimizer. Gradient descent finds minima by taking steps in the direction of the negative gradient. The update rule is given by:



where η isa step size also known as learning rate and  is the loss function evaluated over all training samples

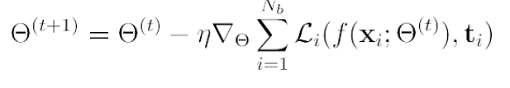




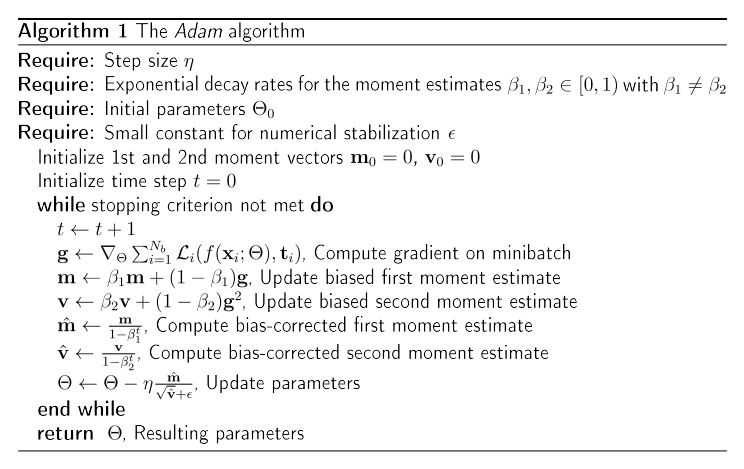
By contrast stochastic gradient descent performs the update rule only on one of the training samples at a time



(2.14] however a lot of times a modified version is used, which employs small subsets of the training data, known as mini-batches. With Nb,denoting the batch size, the update rule of SGD becomes



**Adam** Another variant of the gradient descent method is Adam, which we briefly describe here since it proved to be an efficient optimizer and was used for most of the experiments within the scope of this project. The name derives from adaptive moment estimation. In the so called momentum method the previous step is remembered and contributes to the current update. Usually it is multiplied by a forgetting factor and added to the current parameter. The ADAM method computes adaptive learning rates for the first and second moments. It is described in the following algorithm 1:



**2.3.3 Backpropagation**

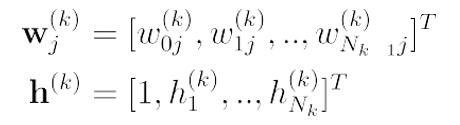
To calculate the gradient ∇Θl the so called backpropagation algorithm is used. In the first step, the forward pass, the error is computed by propagating the input through the network to calculate the output for the current parameters and a given training pair r minibatch. Then, by using the chain-rule, the error is propagated back through the network to obtain the contribution to the output value for each of the individual weights, this step is known as the backward pass46.

 In the following we will use the symbol to denote a single weight of the parameter set, where describes the index for the weight vector, j the hidden unit, and *k* the number of the corresponding layer.

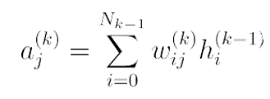
For simplification, the bias will be omitted, which could also be interpreted as adding a value

to the weight vector and having an additional output value

of value1 at the previous layer.

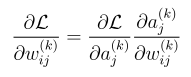


As already mentioned above, the activation of the j-th hidden unit inside the k-th layer is defined as

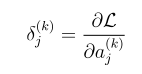


Where hi is the i-th output value after the nonlinear function g

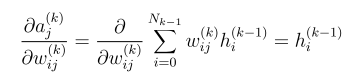
In general, the derivation of the loss function L w. r.t the weight wij depends on activation aj of the corresponding layer before it is passed to the activation function.



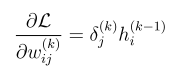
We donate the left term also referred to as errors by



Using equation 2.16 the right term can be written as



Substituting equations 2.19 and 2.20 into equation 2.18 results in:



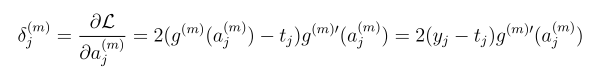
This means, that the partial derivative of a weight is simply the product of the error

in the current layer and the output

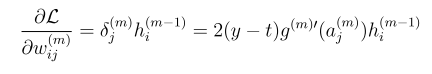
at node i in the previous layer. With this we can now derive the partial derivatives for the output layer and any given hidden layer. Assuming a single training sample the error function is



with m denoting the output layer . using equation 2.19

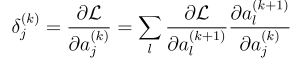


Which gives us the gradient of the error function for a weight in the output layer:

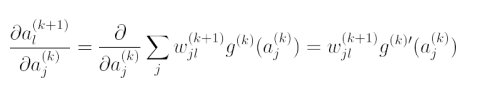


As for a arbitrary hidden layer, we use again the chain rule to evaluate the error term where the sum runs over all nodes in the subsequent layer k+ 1.

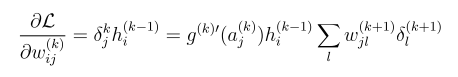
Using again equation 2 19, we can write



With equation 2.16 the right term becomes



Combining equations 2.26 and 2.27 leads to the backpropagation formula:



Which now can be used to compute the partial derivative of the loss function

For a given weight

The stated derivation of the backpropagation formulas are based on [44, 5.3] but as in 46 the superscript notation for indicating the layer index was added.

**2.3.4 Regularization**

In machine learning an essential goal is to develop a model that performs well on unknown test data and not only on the trailed data. This property is described by the term generalization.

If the training error is small but the model does not perform well on the test data this means that the underlying function is not well approximated. This case is called overfitting and is illustrated as plot c pf fig. 2.5. on the contrary, when the true function is not approximated well enough so that both training and test performance are low the term underfitting is used. This case is illustrated in plot a.

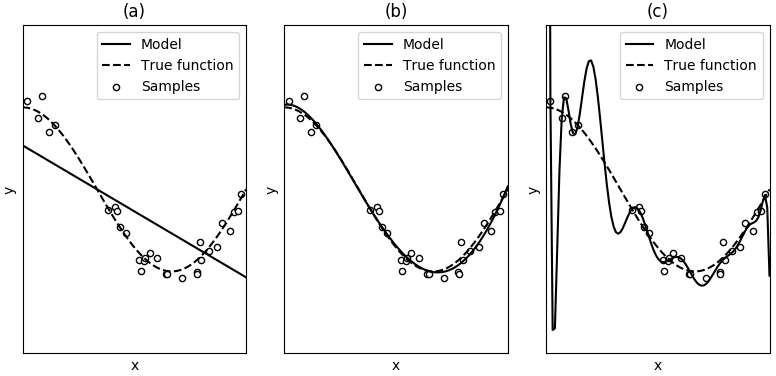


Figure 2.5 — Demonstration of underfitting and overfitting. In plot {a} the model is too simple and the true function is not approximated well enough, only a few samples fit to estimated function. In plot (C) the model is too complicated, most of the samples fit the function but the estimated curve is not a good approximation of the true function. Plot (b) shows an accurate model of the true function.

To improve generalization and avoid overfitting there exist many strategies namely early stopping and dropout .

**Early stopping** During training, the loss represents the performance in terms of generalization, the model is evaluated on a small validation set, usually after each epoch. Ideally training and validation loss decreasing steadily over time. However, in the case of overfitting, an increase of the validation loss may be observed while the training loss is further improving. One regularization strategy is to stop the training procedure if such a drift is observed and the validation loss starts to increase.

This Strategy is known as early stopping and is illustrated in figure 2.6. The model parameters are saved after each epoch and if the validation loss is not further improving the training stops and the parameters corresponding to the best validation performance are returned. Usually the validation state is a subset of the training data which is not fed to the model .

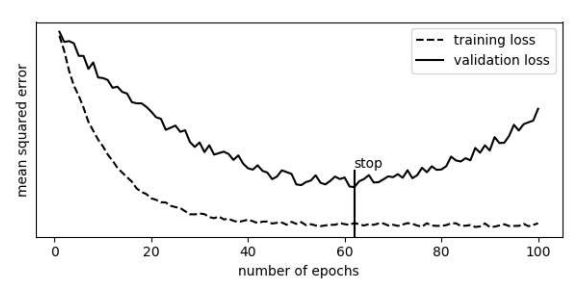


figure 2.6 illustration of early stopping the validation loss is monitored after each epoch and if no improvements can be observed the training procedure is stopped.

**Drop out** A different regularization approach is **dropout** training which was first published in 2014 by Srivastava et al.

The idea is to randomly set a certain percentage of the input and hit the units to 0 after each weight update. This can also be seen as randomly sampling a binary Mask which is applied to all the input until the units in the network. The probability of a mask value to be 0 is a parameter which has to be selected before training. There are different ways to implement Dropout regularization. The keras framework applies regular realization separately to each layer making it possible to use Dropout only for selected layers with different parameters

**2. 4 Network Topologies**

In addition to feed forward networks deep learning involves numerous network structures as well as different regularization and optimization procedures. In the following three common network topologies will be introduced namely autoencoders in convolutional neural networks and recurrent neural networks.

**2.4.1 Autoencoders**

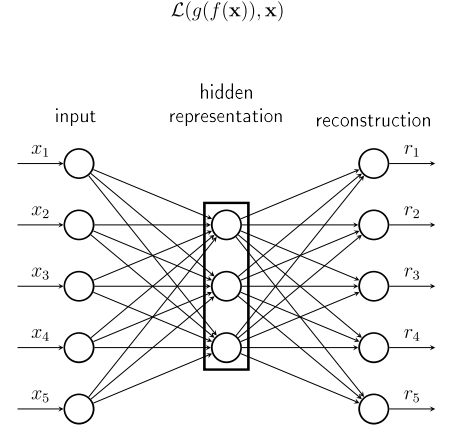
Autoencoders [20, ch.14/ area class of neural networks that attempt to output a copy of the input data. They consist of two parts: an encoder h= f (x) that produces a hidden representation of the input data and a decoder r =g(h) that outputs a reconstruction of the input from the hidden representation. The learning process can be described as minimizing a loss function over the input data and the estimated output

Figure 2.7 — An autoencoder consists of an encoder h= f'(x) that produces a hidden representation of the input data and a decoder r =g(h) that outputs a reconstructruction of the input from the hidden representation .

The model often learns useful properties of the input data which can be used for dimensionality reduction or feature learning. A variant is the so called **Denoising Autoencoder** (DAE), which tries to reconstruct the data x from the input which is corrupted by some form of noise



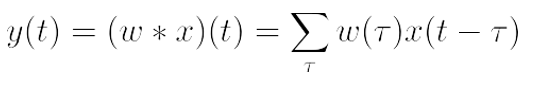
Autoencoders are often regularized by adding some kind of penalty term to the loss function, for example a sparsity constraint on the code layer or any kind of activity regularization [20, ch 14.2/.

Deep autoencoders can be formed the same way as simple deep feedforward networks by increasing the number of layers in the encoder and decoder.

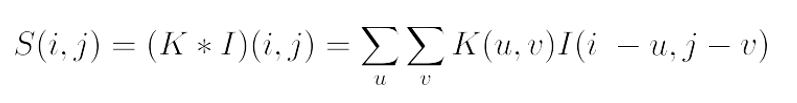
As described in (16], unsupervised pre-training in combination with supervised fine-tuning can be used to build a deep belief net (DBN). Originally restricted boltzmann machines (RBM) and contrastive divergence were used to build the DBNs, but for practical reasons the RBMs are sometimes substituted with autoencoder networks t25) t2d). Multiple single autoencoders are trained seperately to reconstruct the previous autoencoder's hidden layer. These mutiple autoencoders are then stacked on top of each other and can be trained the same way asa conventional DNN. The benefit of the pre-training stage is that the layer weights are already guided into a useful direction rather than starting with a random weight initialization for the fine-tuning. This can avoid the weights to get stuck at local minima and speeds up the training process.

2.4.2 Convolutional Neural Networks

A commonly used network topology, especially for problems involving image data are Convolutional Neural Networks (CNNs) [20, ch.6]. The principal difference to conventional neural networks is that they applya convolution operation to the input rather than a matrix multiplication.



Where w is referred to as the kernel and x as the referred to as the input. The output is sometimes feature map. Often convolutional layers operate on multidimensional input data so the convolution is carried out over multiple axis. In the case of 2-dimensional image data I and filter kernels K the convolution becomes



Note that there are different ways to handle the convolution near the edges of the input image. Sometimes indices for which the kernel would be convolved with values outside the range of the input image are excluded from the operation, which results in a smaller output image. A different option is to pad the input image outside the edges so that the output is of the same size as the input.

Figure 2.8 depicts an example of a convolutional operation. The image on the left is convolved with the Kernel IN in the middle and results in the matrix shown on the right. In this example the input image is zero-padded to ensure an output image of the same dimensions.

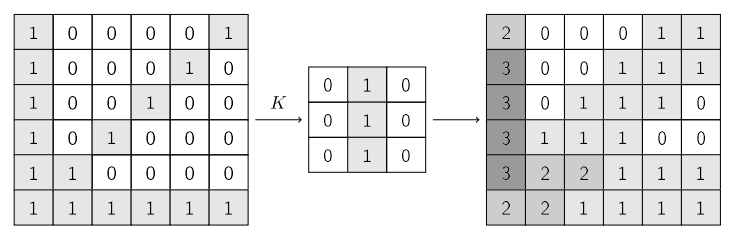
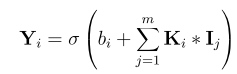
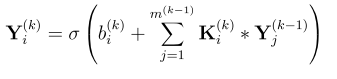
****

Figure 2.8 – Demonstration of the convolution operation. The image on the left side is convolved with the kernel K (middle), which results in the modified image on the right Since the kernel represents a vertical line, the resulting image show in –creasing values in areas where the input image are treated as zero.

In practice ,images often have multiple channels ,e.g. three channel j=1,2,3 for the colors red,blue and green (RGB). In this case the output of the input layer is summed over all channels



Here ᵢ is the index for the filter kernel in the current layer and b is a bias value. In general a layer will add up to m(ᵏ) feature maps



it should be noted that there are different ways to implement a convolutional layer. The above stated definitions are in accordance to the implentations used in the Keras APL. Typically a pooling operation takes place after the convolution and the subsequent non- linear transformation of the input.

A typical CNN for image classification is depicted in 2.10. It usually consists of multiple convolutional layers, each followed by a pooling layer. While this part can be interpreted as a feature extraction step, the following feed-forward network ,consisting of multiple fully – connected layers can be seen as the actual classifier. The values inside the feature maps previous to the full connected layer are stacked to one large vector.

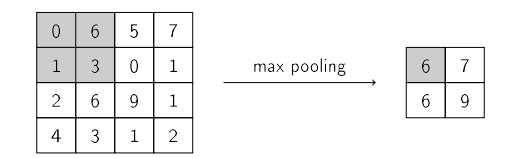


Figure 2.9 – Demonstration of the max pooling operation . The window size and stride in this example is (2,2) which results in reduced size of the image.

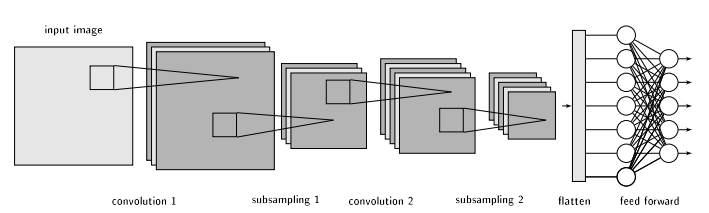
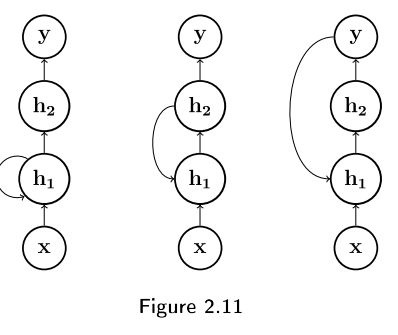


Figure 2.10 –Typical CNN consisting of 2 convolutional layers,2 pooling layers and 2 fully connected layers. After the second pooling operation all resulting pixels are stacked to one , vector, which is the input to the first fully connected layer.

**2.4.3 Recurrent Neural Networks and LSTMs**

**Recurrent Neural Networks** (RNN) are a class of neural networks which operate on sequential data and have been successfully applied to numerous tasks such as language processing, automatic speech recognition or handwriting recognition. The main difference to feed forward networks is that RNNs incorporate connections to values from previous time-steps. There exist numerous patterns for RNNs, which differ in the way the recurrent connections are placed inside the network.

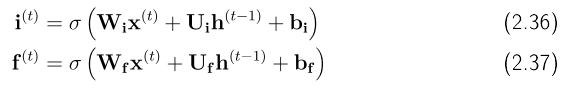
Figure 2.11 shows three simple possibilities for recurrent connections inside a RNN, in the left example the first hidden unit receives its own output from the preceding timestep in addition to the current input values xt. In the middle the recurrent connection is between the output of the second hidden layer h2(t-1) and the first hidden layer and on the right hand side the first hidden unit receives the net output from the preceding time-step y(t-1)



During training RNNs are unfolded to forward networks which can result in very deep and complicated structures. The computation of the gradient for unfolded networks may include many multiplications of a weight by itself which can cause the gradient to take very small values, aggravating the training process of RNNs. This so called vanishing gradient problem led to the design of long short-term memory cells or LSTMs.

LSTM recurrent networks are organized in cells that have an internal self-loop controlled by a system of gating units. Each gate behaves similar to a single conventional hidden bit, having a bias vector b,a weight matrix W as well as an additional weight matrix controlling the recurrent connection.

Such a LSTM structure is depicted in figure 2.12 and is described in the following. The current state s(t) is controlled by the input gate i(t) and the forget gate f(t), which adjust the contribution of the current and the previous state.



With t denoting the timestep and x(t) denoting the current input and the previous output of the cell. The sigmoid activation denoted by sets the weights to values between 0 and 1.

Introducing an intermediate state s(t)

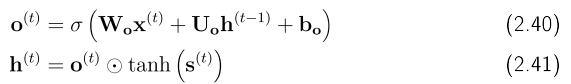


the current state is obtained as follows

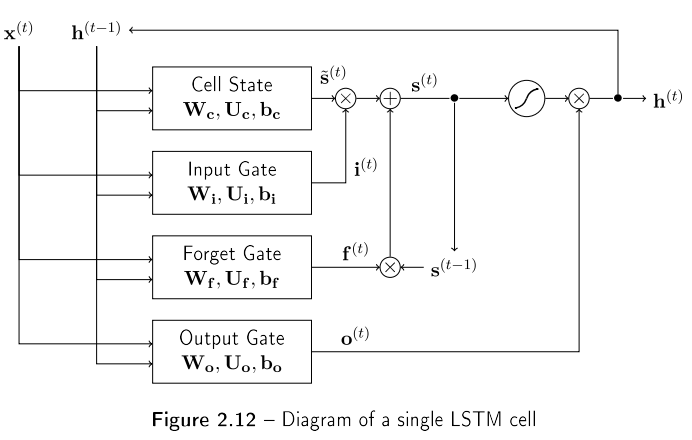


Where the operator denotes the Hadamard product.

Finally, the output gate o(t) controls the contribution of the current state to the output h(t)



There exist many variants of the internal structure of LSTMsaswell as alternatives such US gated recurrent units (GRUs). In GRUs thenumber ofparameters is reduced by simultaneously controlling the forgetting factor and the decision to update the weight.

The above stated definitions are in accordance to the implementations used in the Keras APL.

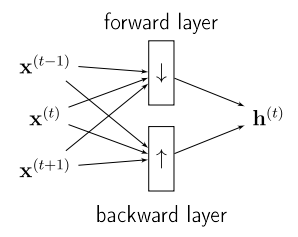
 To increase the amount of input information, recurrent neural networks are sometimes extended to bidirectional recurrent neural networks (BRNNs). For this the network is trained simultaneously in positive and negative time direction using separate forward and backwards layers, whose outputs are merged and bed to the next layer.

Figure 2.13- Bidirectional RNNs employ separate forward and backwards layers to increase the amount of input context.

Bidirectional RN Ns have been successfully used for optical character recognition, speech recognition as well as speech separation

# 3 Spectro-TemporaI Signal Representation

To exploit the time-frequency characteristics of the underlying signal components, many speech enhancement algorithms operate in the spectral or even the cepstral or modulation domain. Usually this is done by fourier transformation or filterbank analysis. Most algorithms can be divided into three coarse steps as depicted in figure 3.1. In the first stage the signal is analyzed by transformation into the frequency domain and in the second step the signal is modified by filtering out unwanted components. Finally the enhanced waveform is obtained by transforming the modified signal back to the time-domain [54, p.5].

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | Analysis | x | Modification |  | Synthesis | |  |  |  | |  |

Figure 3.1 — Analysis-modification-synthesis framework as used in many speech enhancement algorithms.

In the case of machine learning based systems the transformed signal X can also be seen as the feature-space for the learning algorithm, this means that the signal representation at this point should also meet specific requirements for efficient learning, such as independency, size and a certain value range.

Chapter 3.1 will give a short introduction to fourier analysis and the spectrogram representation followed by a brief outline of its inversion. In chapter ?? the mel-weighted spectrogram will be introduced.

## 3.1 Spectrogram

The Fourier Transform [T ODO cite speechcom] decomposes a time-domain signal

into its spectral components. It is defined by the following formula:

(3.1)

Where the t denotes the time index and w 27Tf the frequency.

The corresponding inverse transform is most commonly defined as

(3.2)

The above mentioned Fourier Transform pair is only defined for continuous signals both in time and frequency.

When working with time-discrete signals, as it is the case with most digital signals and systems the Discrete Fourier Transform (DFT) is used. A finite signal of length N, sampled at a rate of fs will result in a spectrum of frequency coefficients equally spaced by  .

Here n denotes the time index and k the frequency index. When the DFT is carried out for consecutive frames the Short-Time Fourier Transform (ST FT) can be computed. The frames are weighted by a window function w(n) and shifted in time by a factor R, this yields the complex spectrum depending on the frame index I

A widely used signal representation is the spectrogram. It is obtained by considering only the squared magnitude of the complex spectrum. Usually it is expressed in dB, in which case representation is also referred referred to as log-power spectrum (LPS)

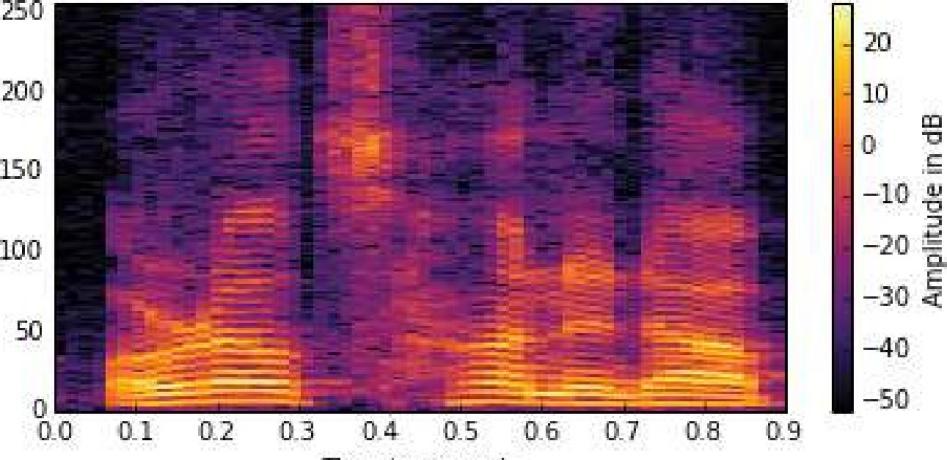
As shown in figure 3.2 the spectrogram is commonly presented as a two dimensional plot with time on the x-axis and frequency on the y-axis. The magnitude is shown in dB:

Spectrogram Inversion To obtain the signal waveform from the complex ST F T, each spectral frame is transformed back to the time domain via inverse DFT . The overlapping time signal frames are then weighted by a synthesis window s and summed together according to the frame shift. (3.7)

To reconstruct the time signal from a spectrogram representation it has to be combined with the corresponding phase

 (3.8)

8



Time

jn

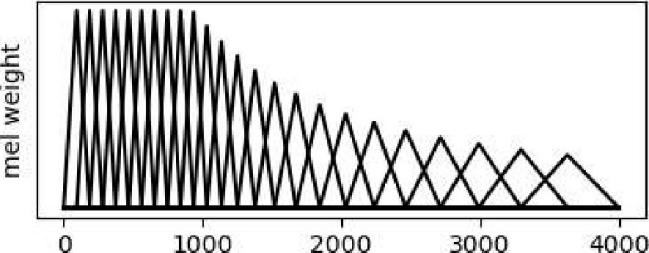
seconds

Figure 3.2 — Power spectrogram of a speech signal. The magnitude at a single time-frequency point is given as a color value.

## 3.2 Mel-Spectrum

The mel scale describes the perceptual pitch of a tone. It is based on listening experiments, which revealed that the intervals of equal pitch increments are increasing with frequency. For lower frequency bands, roughly below 500 Hz, the pitch intervalls are equally spaced while they increase gradually for higher frequencies. Since the mel scale is based on listening experiments, there exist mutiple versions of the conversion formula [55], [56].

Usually multiple frequency bins are combined by a wighting curve to form a single mel band [57]. Figure 3.3 shows the weighting curves for M 24 mel bands.



1000

2000

3000

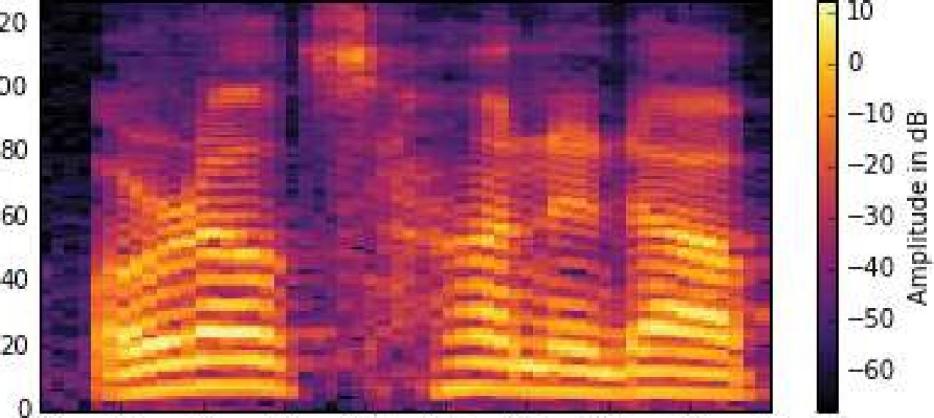
4000

frequency in Hz

Figure 3.3 — Mel filter as a function of frequency.

To transform a N-point spectrum into the mel domain it is multiplied by a weighting matrix T of size M x N, where M denotes the number of mel bands.

(3.9)



120

100

0.0 0.1 07 0.3 0.4 0.5 0.6 0.7 08 0.9

Time jn seconds

Figure 3.4 — Mel-spectrogram of a speech signal. The magnitude at a single time-frequency point is given as a color value.

**4 Evaluation Criteria**

To objectively evaluate the performance of speech enhancement systems, the following methods are used in the context of this work. The first two criteria, namely PESQ and STOI are full reference (FR) algorithms, meaning that they need to be provided with a reference signal for comparison. FR algorithms are considered to be more accurate than no reference (NR) systems, which only use the enhanced output signal for evaluation.

**4.1 PESQ**

PESQ stands for Perceptual Evaluation of Speech Quality and was originally de- veloped by the International Telecommunication Union (ITU) as an objective method for speech quality assessment of telephone networks and speech codecs. Since it includes measurements of environmental noise at the sending side when used as a full reference algorithm, it is also a widely used tool for the evaluation of speech enhancement algo- rithms. The PESQ compares a reference signal with a degraded signal, which in our case is the noisy signal processed by the speech enhancement algorithm. First a series of delays between the two signals are computed for time alignment. Then they are com- pared by using a perceptual model which takes account of the perceptual frequency and loudness in the human auditory system. The basic structure is depicted in figure 4.1. For our experiments, the implementation

Internal representation

Perceptual

model

Original input Of original signal

Difference in internal

quality

Cognitive

model

Time

alignment

Representation determines

The audible difference

Degraded

output

Perceptual

model

Internal representation of

Degraded signal

**Figure 4.1** - Overview of the basic philiosophy used in PESQ [58]

4 EVALUATION CRITERIA

as described in the ITU-T recommendation P.862 (02/2012) [58] is used. The algorithm reports the raw PESQ value ranging from -0.5 (worst) to 4.5 (best) as well as a mapping onto MOS-LQO (Mean Opinion Score - Listening Quality Objective) scale ranging from 1 to 5. For the results reported in the following chapters the latter will be used, unless otherwise stated.

**4.2 STΟΙ**

**Short-time objective intelligibility measure** as described in [59] is an algorithm to predict the intelligibility of noisy speech. It ranges from 0 (no intelligibilty) to 1 (perfect intelligibilty). In general it analyses short (approximately 400 ms) overlapping time segments and compares the clean and degraded speech signals based on a correlation coefficient between their temporal envelopes. In a first step the signals are resampled to 10 kHz and framed into 50% overlapping frames with a length of 256 samples. The frames are weighted by a Hann-window and are zero-padded up to 512 samples. To exclude non-speech frames, silent regions are removed by analyzing the energy content within the clean speech frames. After transformation into the frequency domain, the DFT-bins are grouped to 15 one-third octave bands ranging from 150 Hz to 4.3 kHz.

(4.1)

Where (k,m) denotes the k-th frequency bin of the m-th clean speech frame and j denotes the band index. The band edges are denoted by k₁ and k₂ . In the next step the single frames are grouped to short-time regions spanning N = 30 frames.

(4.2)

To componsate for global level differences, the degraded speech regions y(n) are nor- malized and clipped.

(4.3)

where β = -15 dB and || . || represents the norm. The intermediate intelligibility measure for a time-frequency unit is then calculated as the sample correlation coefficient between the clean and the noisy vector.

(4.4)

where refers to the mean value of the corresponding vector. Finally the result is given as the average over all bands and time-segments.

d= (4.5)

4.3 Speech Accuracy Score

Finally the score utility from Microsoft Speech Platform is used to assess the performance of the presented algorithms in terms of speech intelligibility. The resulting score reports the accuracy of Microsoft's speech recognition system and returns a percentage titled 'Speech Accuracy Score' and a text file containing pairs of recognitions and transcriptions. For this a noisy audio recording with a duration of 10 minutes, consisting of voice commands by different female, male and children speakers is provided. The raw audio file has a speech accuracy score of 41.9%. The tool is used at Harman for benchmarking in the context of projects in cooperation with Microsoft and further details on the technical background are kept confidential.

4.4 Signal-to-Noise Ratio

The signal-to-noise ratio (SNR) compares to power of the wanted signal, in our case the speech signal, to the power of the noise signal and is commonly expressed in decibels.

**SNR = (4.6)**

It can be calculated using the summed squared magnitudes

**SNR = (4.7)**

Note that for this metric, the noise signal has to be known. Considering an additive noise model (see eq. 1.1) the noise signal can be expressed by the difference between the the noisy signal x(t) and the clean speech signal.

**SNR = (4.8)**

1. **Time-Frequency Masks**

To use deep learning for speech enhancement there are two commonly employed strate­ gies. The first approach is to train the network directly on pairs of noisy and clean amplitude spectra and to combine the estimated amplitude and the noisy phase in the enhancement stage [32] [33]. The other approach is to use time-frequency masks as learning targets, which are then used to segregate the speech and noise components in the noisy spectrum [31] [60].

Time-frequency masks are widely used in computational auditory scene analysis (CASA) [61]. CASA systems analyze the acoustic input such as a cochleagram or correlogram by trying to segregate the different sources. This is often done by computing a mask weight for each time-frequency point, which emphasizes regions dominated by the target source and supresses those dominated by other sources. In this case the estimated signal is obtained by element-wise multiplication of the source signal and the time-frequency mask.

*S(t, f)* = *M(t, f)* 0 *X(t, f)*

Where 0 defines the hadamard-product or element-wise multiplication.

In the following three different types of spectral masks, commonly used for source separa­ tion and speech enhancement will be presented. Namely Ideal Binary Mask (IBM), Ideal Ratio Mask (IRM) and Phase Sensitive Filter (PSF). Finally the different time-frequency masks will be evaluated and compared in terms of their speech enhancement capabilities.

* 1. **Ideal Binary Mask**

The **ideal binary mask** or IBM introduced by Wang et al. [62], [63], classifies a single time-frequency bin of the mixture spectrum *X* as being associated with either the speech or non-speech component. If the local SNR at a single point *(i,j)* is larger than a local criterion ; the corresponding mask-value is set to one otherwise it is considered as noise and will be zero to suppress the spectral magnitude at this specific point

The local criterion ( depends on the overall SNR of the mixture and a parameter *a.*

Where 5 dB is a commonly used value for *a* [31].

(5.3)

The IBM is considered globally optimal in terms of SNR if the window function is rectangular and the time-frequency decomposition is orthogonal [64], [65]. Plot *(c)* in figure 5.2 shows the ideal binary mask for a speech signal *(a)* masked by babble noise *(b).*

**5.2 Ratio Mask**

A commonly used spectral mask especially in the context of DNN based speech enhance­ ment is the **ideal ratio mask** (IRM) [66], [31], [37]. In contrast to the IBM, the IRM belongs to the family of softmasks, meaning that its range is not constrained to binary values. When the phase of the underlying signals *s* and *n* are equal, the IRM is consid­ ered to produce optimal results [36]. It is closely related to the wiener filter, which is the optimal filter in the minimum mean-square error sense and is defined as follows

If power spectral densities are used instead of spectral values and the speech and noise signals are uncorrelated, the above mentioned relation would match the wiener filter.

Several studies have found that the IRM generally performs better than the IBM in the context of source separation [31] [65] [36]. Plot *(d)* in figure 5.2 shows the ideal ratio mask for a speech signal *(a)* masked by babble noise *(b).*

* 1. **Phase Sensitive Filter**

A different type of soft mask is the **phase sensitive filter,** which was introduced by

Hakan Erdogan et al. and obtained promising results in the context of deep learning based speech separation [36]. The PSF is derived from the ideal complex mask M1CM = f by

considering only the real part of the filter.

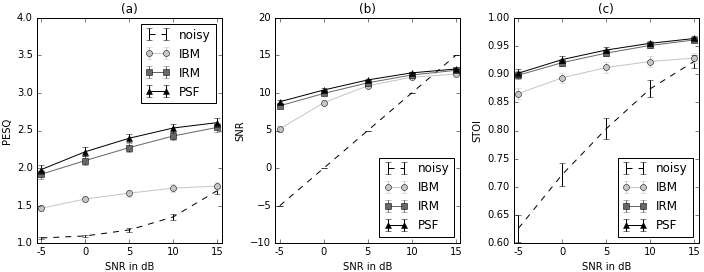
For the practical use as a learning target for neural networks the value range will be constrained by clipping the values to [O,1]. The resulting mask is also referred to as truncated PSF 1 and is illustrated in plot *(e)* of figure 5.2.

## Comparison

The described time-frequency masks are compared in an oracle scenario using a test set of 360 utterances and the above mentioned evaluation criteria. The test set includes four female and four male speakers and each of the utterances is superimposed with a noise signal resulting in 360 noisy utterances divided into five different signal to noise ratios, namely -5, 0, 5, 10 and 15 dB 2. Using the noisy signal and the corresponding clean speech signal the individual time-frequency masks are calculated. The resulting enhanced signals are constructed by multiplying the noisy spectra and the corresponding spectral masks and subsequently transforming the spectra back to the time domain by inverse STFT.

Figure 5.1 shows the PESQ *(a),* SNR *(b)* and STOI *(c)* for IBM, IRM and PSF in com­

parison to the noisy signals of the test data. The respective results are averaged for each of the four SNR levels in the test set.

Table 5.1 shows the delta values for the resulting PESQ, SNR and STOI for the indi-

**Figure 5.1** - Comparison of different spectral masks. For the metrics PESQ, SNR and STOI, the mean values as well as the 95% confidence intervals are indicated as measured for a test sest of 360 sentences with different SNRs

vidual mask approaches. For each of the resulting enhanced utterances the difference to the original noisy signal is calculated and the delta values are given as the mean value over all measurements.

It can be observed that in terms of speech quality and intelligibility, as measured by the PESQ and STOI metrics, the softmasks clearly outperform the binary mask. As for the SNR of the resulting signals the difference is only significant for severe noise conditions below O dB SNR. While the differences are clearly visible for the PESQ results, the perfor­ mance of the IBM in terms of the resulting SNR improves with better overall SNR of the original noisy utterances. Additionally, the IRM and PSF results are more independent from the SNR of the input signal than the IBM. When looking at the average results for the STOI measurements in figure 5.1 there is almost no visible difference between the

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mask | PESQ | SNR | STOI |  |
| IBM | 0.4 | 5.2 | 0.14 |  |
| IRM | 1.0 | 6.5 | 0.17 |  |
| PSF | **1.1** | **7.0** | **0.17** |  |

**Table 5.1** - PESQ results for different oracle masks at various SNR levels.

performance of the two softmasks and a difference of about 0.4 to that of the binary mask.

The delta values in table 5.1 show that the PSF performs best for all of the three evalu­ ation criteria closely followed by the IRM.

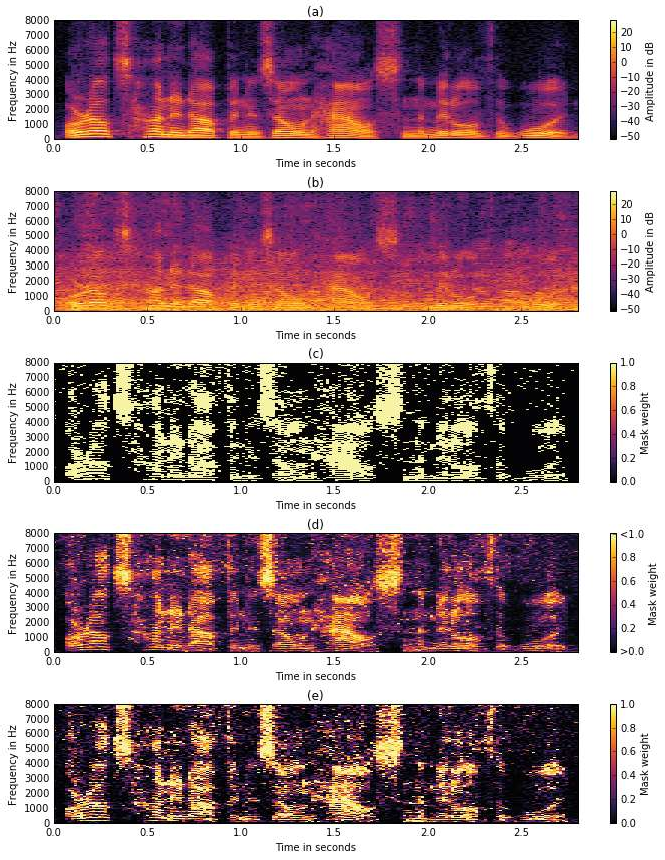
The mentioned results are consistent with the results presented in [31] and [36], which also shows that the phase sensitive filter may be superior to the ideal ratio mask not only in terms of speech separation, but also for speech enhancement tasks.

Figure 5.2 additionally shows a clean speech signal *(a)* and the same signal superim­

posed with babble noise at 5 dB SNR *(b)* as well as the corresponding IBM *(c),* IRM

*(d)* and PSF *(e).*

In the next chapter it will be presented how deep neural networks can be used for speech enhancement and the general framework for the final experiments will be introduced.



**Figure 5.2** - Illustration of different oracle masks. A clean speech signal *(a)* is superimposed with babble noise at 5 dB SNR, which results in the noisy signal in *(b).* Plots *(c), (d)* and *(e)* show the corresponding IBM, IRM and PSF respectively.

# Experi menta I Setup

This chapter will give an overview about the general framework for the conducted ex­ periments. First the basic speech enhancement approach is introduced followed by a description of the dataset and the generation of the training and test data. Furthermore, an explanation of the feature extraction and training procedures will be given along with a short description of the baseline algorithm used for comparison.

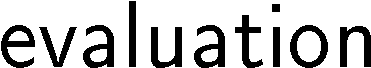
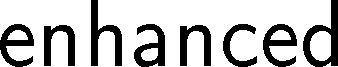
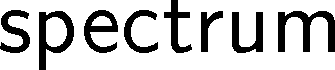
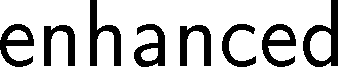
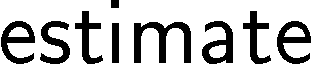
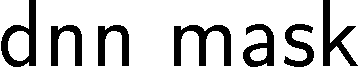
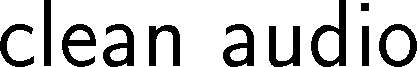
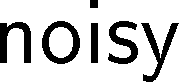
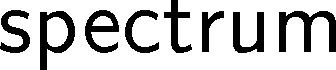
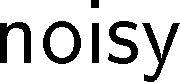
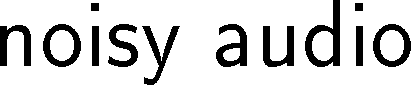
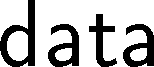
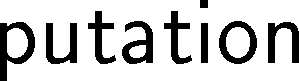
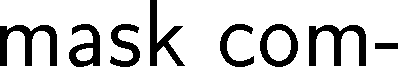
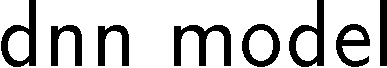
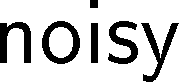
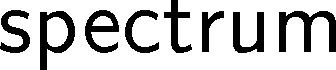
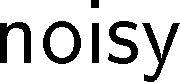
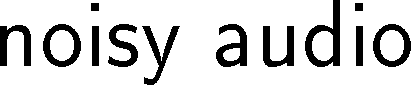
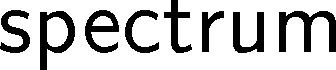
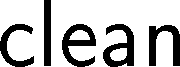
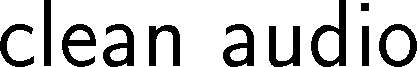
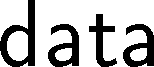
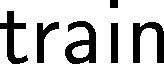
## Proposed Framework

For the experiments conducted in the context of this work, the following framework is used. The neural network models are trained to predict spectral masks from noisy speech input. For that the model is fed with features extracted from noisy speech i.e. lps or mel spectra. After training, the neural network model is used to predict the time-frequency mask for the noisy input features frame by frame. Figure 6.1 shows the proposed speech enhancement method. The upper half of the diagram depicts the training process of the neural network model and the lower half shows the enhancement and the testing framework respectively.

From the noisy time signal the spectrum is obtained by Fourier analysis. Then the input features for the neural network are extracted and standardized using the global mean and variance of the training data.

A detailed explanation of the feature extraction process as well as the estimation of the global mean and variance of the training data is given in chapter 6.3.

The enhanced complex spectrum is obtained by element-wise multiplication of the es­ timated spectral mask with the complex spectrum of the noisy signal. Finally, the enhanced speech spectrum is obtained by inverse fourier transform.



**Figure 6.1**

Proposed speech enhancement framework. Features are extracted from the noisy spectogram and then fed into the neural network model subsequntly the pre-dicted spectral mask is multiplied by thr noisy spectrogram to attain an estimate of the clean speec spectrogram. Finally the enhanced speech signal is obtained by inverse FFT and overlap and add

To provide the model with information about the temporal context, the input features will span multiple time frames corresponding to exactly one framr of the respective spec-tral mask. More specifically, a context size of 11 frames was chosen for the experiments and depending on the particular ANN model, the output will either correspond to the 6th (center) or 11th (end) input framr. It should be noted that the target mask is estimated framr by framr, which means that there is no temporal dependency between successive target predictions

* 1. **Importing libraries**

In []: import gc

import os

import re

import traceback

from functools import wraps

from random import randint

from time import perf\_counter

import IPython

import matplotlib.pyplot as plt

import numpy as np

import psutil

import torch

import torch.nn as nn

import torchaudio

from IPython.core.display import display

from ipywidgets import interact

from matplotlib.lines import Line2D

The code you provided is a Python script that imports several libraries and modules for various purposes. Let me break down the functionality of each import:

import gc: This imports the gc (Garbage Collector) module, which provides an interface to the optional garbage collector. This can be used to control the garbage collection process.

import os: This imports the os module, which provides a way to interact with the operating system. It can be used for tasks such as creating, deleting, or renaming files and directories.

import re: This imports the re module, which provides support for regular expressions. Regular expressions are a powerful way to perform text matching and manipulation.

import traceback: This imports the traceback module, which provides functions to extract, format, and print stack traces of Python programs.

from functools import wraps: This imports the wraps function from the functools module, which is used for creating decorator functions.

from random import randint: This imports the randint function from the random module, which generates a random integer within a specified range.

from time import perf\_counter: This imports the perf\_counter function from the time module, which provides a high-resolution time function for performance measurements.

import IPython: This imports the IPython module, which provides an enhanced interactive Python shell.

import matplotlib.pyplot as plt: This imports the pyplot module from the matplotlib library, which is a popular data visualization library for Python.

import numpy as np: This imports the numpy library, which is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices.

import psutil: This imports the psutil (Python System and Process Utilities) library, which provides a cross-platform interface to retrieve information about running processes and system utilization.

import torch: This imports the torch library, which is a popular machine learning and deep learning library for Python.

import torch.nn as nn: This imports the nn module from the torch library, which provides functionality for building neural networks.

import torchaudio: This imports the torchaudio library, which provides functionality for audio processing and manipulation.

from IPython.core.display import display: This imports the display function from the IPython.core.display module, which is used for displaying output in the Jupyter Notebook environment.

from ipywidgets import interact: This imports the interact function from the ipywidgets library, which is used for creating interactive widgets in Jupyter Notebooks.

from matplotlib.lines import Line2D: This imports the Line2D class from the matplotlib.lines module, which is used for creating line plots in Matplotlib.

* 1. **Data Set**

In []: def load\_folder(folder\_path):

if not folder\_path.endswith("/"):

folder\_path += "/"

tensor\_array, sampling\_rate = [], None

for file\_name in os.listdir(folder\_path):

if file\_name.endswith(".wav"):

tensor\_data, sr = torchaudio.load(folder\_path + file\_name)

tensor\_array.append(tensor\_data)

sampling\_rate = sampling\_rate or sr

assert sampling\_rate == sr, "Sampling rate not uniform on all the data"

print(f"Sampling rate: {sampling\_rate}")

return torch.stack(tensor\_array)

In []: noise\_train = load\_folder("noise/train/")

voice\_train = load\_folder("voice/train/")

noise\_test = load\_folder("noise/test/")

voice\_test = load\_folder("voice/test/")

Sampling rate: 8000

Sampling rate: 8000

Sampling rate: 8000

Sampling rate: 8000

In this part we expect to find in the current directory two folders: noise and voice, each of them containing two subfolders train and test. We will use combinations of the noise and voice samples in the train subfolders to perform the training part, and combinations taken from the test subfolders for the evaluation part.

**6.3** **Normalization and Visualization**

In []: normalize = True

if normalize:

noise\_train = nn.functional.normalize(noise\_train, dim=2)

voice\_train = nn.functional.normalize(voice\_train, dim=2)

noise\_test = nn.functional.normalize(noise\_test, dim=2)

voice\_test = nn.functional.normalize(voice\_test, dim=2)

In []: @interact

def plot\_voice(i = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) -> None:

display(IPython.display.Audio((voice\_train[i]), rate=8000))

interactive(children=(Dropdown(description='i', options=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10), value=1), Output()), …

In []: @interact

def plot\_noise(i = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) -> None:

display(IPython.display.Audio((noise\_train[i]), rate=8000))

interactive(children=(Dropdown(description='i', options=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10), value=1), Output()), …

6.4 Training and testing data

**Train and test pipelines**

In []: def train(dataloader, model, loss\_fn, optimizer, print\_loss=True):

size = len(dataloader.dataset)

model.train(True)

for batch, (X, y) in enumerate(dataloader):

X, y = X.to(device), y.to(device)

optimizer.zero\_grad()

*# Compute prediction error*

pred = model(X)

*# pred = torch.argmax(pred, dim=1)*

loss = loss\_fn(pred.squeeze(), y.squeeze())

*# Backpropagation*

loss.backward()

optimizer.step()

if batch % 10 == 0:

loss, current = loss.item(), batch \* len(X)

if print\_loss:

print(f"loss: {loss:>7f} [{current:>5d}/{size:>5d}]")

return loss.item()

In []: def test(dataloader, model, loss\_fn, pesq\_fn):

num\_batches = len(dataloader)

model.eval()

test\_loss, pesq\_err = 0, 0

with torch.no\_grad():

for X, y in dataloader:

X, y = X.to(device), y.to(device)

pred = model(X)

test\_loss += loss\_fn(pred, y.squeeze()).item()

pesq\_err += pesq\_fn(pred, y.squeeze()).item()

test\_loss /= num\_batches

pesq\_err /= num\_batches

print(f"Test Error: \n Avg loss: {test\_loss:>8f} \n Avg pesq: {pesq\_err:>8f}")

return test\_loss, pesq\_err

defines two functions, `train` and `test`, for training and testing a machine learning model, respectively.

The `train` function takes a dataloader, a model, a loss function, an optimizer, and an optional boolean flag for printing the loss. It iterates through the batches of data in the dataloader, computes the prediction, calculates the loss, performs backpropagation to update the model parameters, and optionally prints the current loss. The function returns the final loss value.

The `test` function takes a dataloader, a model, a loss function, and a PESQ (Perceptual Evaluation of Speech Quality) function. It evaluates the model on the test data, computes the average test loss and PESQ error, and prints the results. The function returns the test loss and PESQ error.

These functions can be used to train and evaluate a machine learning model, such as a neural network, on a dataset. The `train` function is responsible for the model training process, while the `test` function evaluates the model's performance on a separate test set.

**6.5** **Evolution of the loss during the training**

In []: def plot\_performance(loss\_val, loss\_test, pesq\_test):

"""Model performance visualization"""

*# subplots*

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 5))

*# Loss on validation dataset*

ax1.plot(loss\_val, color='red')

‘ ax1.set\_title("Model loss on validation set")

ax1.set\_ylabel("Loss on validation set")

ax1.set\_xlabel("Epoch")

*# Pesq on test dataset*

ax2.plot(pesq\_test, color='red')

ax2.set\_title("Pesq on validation set")

ax2.set\_ylabel("Pesq on validation set")

ax2.set\_xlabel("Epoch")

*# Loss on test dataset*

ax3.plot(loss\_test, color='red')

ax3.set\_title("Model loss on training set")

ax3.set\_ylabel("Loss on training set")

ax3.set\_xlabel("Epoch")

plt.show()

**Augmentation using different combinations of voice + noise**

**In [ ]:**

@track\_time\_memory\_usage

def one\_to\_n\_dataset(voice\_samples, noise\_samples, n\_noise\_per\_voice: int, noise\_factor: float = 1.):

train\_set = []

test\_set = []

n\_noise\_samples = noise\_samples.shape[0]

for idx, voice\_sample in enumerate(voice\_samples):

for noise\_sample in noise\_samples[torch.randperm(n\_noise\_samples)[:n\_noise\_per\_voice]]:

train\_set.append(voice\_sample + noise\_factor \* noise\_sample)

test\_set.append(voice\_sample)

return torch.stack(train\_set), torch.stack(test\_set)

**In []:**

if run\_expensive\_functions:

train\_signals, train\_labels = one\_to\_n\_dataset(voice\_train, noise\_train, 5, 10)

test\_signals, test\_labels = one\_to\_n\_dataset(voice\_test, noise\_test, 5, 10)

Running function one\_to\_n\_dataset.

RAM initially used: 10.6557 GB (68.50 %)

We can take a look at a few noisy samples in the dataset to compare them with their labels (same track without noise).

**In [ ]:**

if run\_expensive\_functions:

@interact

def plot\_samples(i = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) -> None:

display(IPython.display.Audio((train\_signals[i]), rate=8000))

display(IPython.display.Audio((train\_labels[i]), rate=8000))

interactive(children=(Dropdown(description='i', options=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10), value=1), Output()), …

**In [ ]:**

if run\_expensive\_functions:

print\_tensors\_size\_in\_memory(train\_signals, train\_labels, test\_signals, test\_labels)

Size in memory of tensor train\_signals: 1939.0869 MB

Size in memory of tensor train\_labels: 1939.0869 MB

Size in memory of tensor test\_signals: 715.9424 MB

Size in memory of tensor test\_labels: 715.9424 MB

**In [ ]:**

if run\_expensive\_functions:

del train\_signals, train\_labels, test\_signals, test\_labels

gc.collect()

*# this actually does not work because a notebook keeps additional reference to*

*# objects like in Out[k], you have to restart the runtime.*

print\_ram\_usage()

# Run and Resu Its

In the run below we implement an online heuristic to update the learning rate in order to avoid oscillations.

**In[ ]:**

epochs = 20

loss\_list\_val, loss\_list\_test, pesq\_list\_test = [], [], []

test\_loss, val\_no\_impv, halving = 10, 0, False

nb\_pesq = PerceptualEvaluationSpeechQuality(8000, 'nb')

for t in range(epochs):

try:

print(f"Epoch {t + 1}\n-------------------------------")

loss\_test = train(train\_dataloader, model, loss\_fn, optimizer)

prev\_test\_loss = test\_loss

test\_loss, pesq\_err = test(test\_dataloader, model, loss\_fn, nb\_pesq)

loss\_list\_val.append(test\_loss)

loss\_list\_test.append(loss\_test)

pesq\_list\_test.append(pesq\_err)

if test\_loss >= prev\_test\_loss:

val\_no\_impv += 1

if val\_no\_impv >= 3:

halving = True

if val\_no\_impv >= 10:

print("No improvement for 10 epochs, early stopping.")

break

else:

val\_no\_impv = 0

if halving:

optim\_state = optimizer.state\_dict()

optim\_state['param\_groups'][0]['lr'] = optim\_state['param\_groups'][0]['lr'] / 2.0

optimizer.load\_state\_dict(optim\_state)

print(f"Learning rate adjusted to: {optim\_state['param\_groups'][0]['lr']:.6f}")

halving = False

except KeyboardInterrupt:

print("\nExecution stopped.")

break

print("Done!")

Epoch 1

-------------------------------

loss: 0.012994 [ 0/ 2118]

loss: 0.012089 [ 100/ 2118]

loss: 0.012211 [ 200/ 2118]

loss: 0.012775 [ 300/ 2118]

loss: 0.013709 [ 400/ 2118]

loss: 0.010598 [ 500/ 2118]

loss: 0.015674 [ 600/ 2118]

loss: 0.010068 [ 700/ 2118]

loss: 0.013917 [ 800/ 2118]

loss: 0.011314 [ 900/ 2118]

loss: 0.014564 [ 1000/ 2118]

loss: 0.013531 [ 1100/ 2118]

loss: 0.015261 [ 1200/ 2118]

loss: 0.011463 [ 1300/ 2118]

loss: 0.011307 [ 1400/ 2118]

loss: 0.013066 [ 1500/ 2118]

loss: 0.011072 [ 1600/ 2118]

loss: 0.010956 [ 1700/ 2118]

loss: 0.013513 [ 1800/ 2118]

loss: 0.010834 [ 1900/ 2118]

loss: 0.009656 [ 2000/ 2118]

loss: 0.012826 [ 2100/ 2118]

Test Error:

Avg loss: 0.013854

Avg pesq: 1.750832

Epoch 20

-------------------------------

loss: 0.008078 [ 0/ 2118]

loss: 0.007998 [ 100/ 2118]

loss: 0.006170 [ 200/ 2118]

loss: 0.008812 [ 300/ 2118]

loss: 0.007891 [ 400/ 2118]

loss: 0.006646 [ 500/ 2118]

loss: 0.008689 [ 600/ 2118]

loss: 0.008053 [ 700/ 2118]

loss: 0.008741 [ 800/ 2118]

loss: 0.007610 [ 900/ 2118]

loss: 0.006512 [ 1000/ 2118]

loss: 0.006227 [ 1100/ 2118]

loss: 0.006852 [ 1200/ 2118]

loss: 0.006970 [ 1300/ 2118]

loss: 0.006171 [ 1400/ 2118]

loss: 0.008709 [ 1500/ 2118]

loss: 0.009028 [ 1600/ 2118]

loss: 0.008439 [ 1700/ 2118]

loss: 0.005719 [ 1800/ 2118]

loss: 0.009585 [ 1900/ 2118]

loss: 0.008895 [ 2000/ 2118]

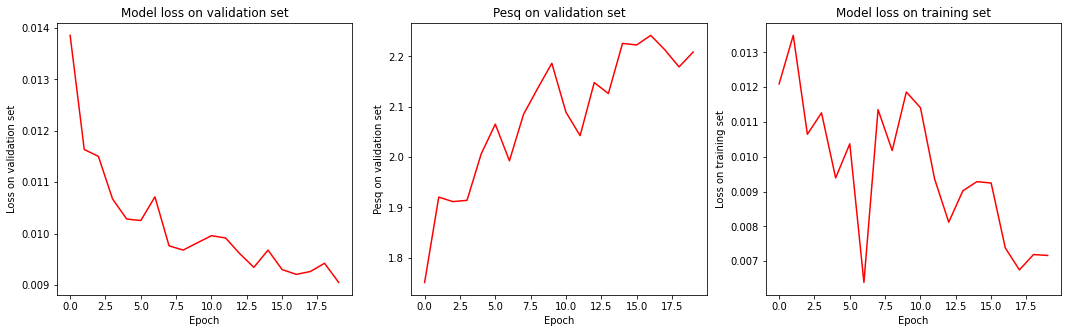
loss: 0.006766 [ 2100/ 2118]

Test Error:

Avg loss: 0.009053 Avg pesq: 2.208733 Done!

**In[ ]:**

plot\_performance(loss\_list\_val, loss\_list\_test, pesq\_list\_test)

****

**In[ ]:**

with torch.no\_grad():

for X, y in test\_dataloader:

n = noise\_test.shape[0]

X, y = (

get\_data(X, n, "test").to(device),

voice\_test[y].to(device)

)

pred = model(X)

break

compare\_batch(X, y, pred)

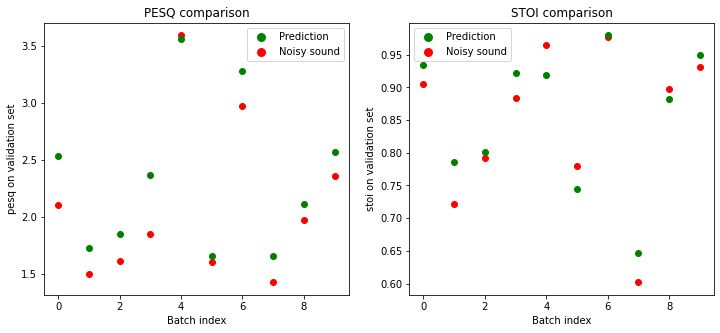
#

Mean PESQ between noisy signal and target signal: 2.0956

Mean PESQ between denoised signal and target signal: 2.3272

Mean STOI between noisy signal and target signal: 0.8456

Mean STOI between denoised signal and target signal: 0.8567

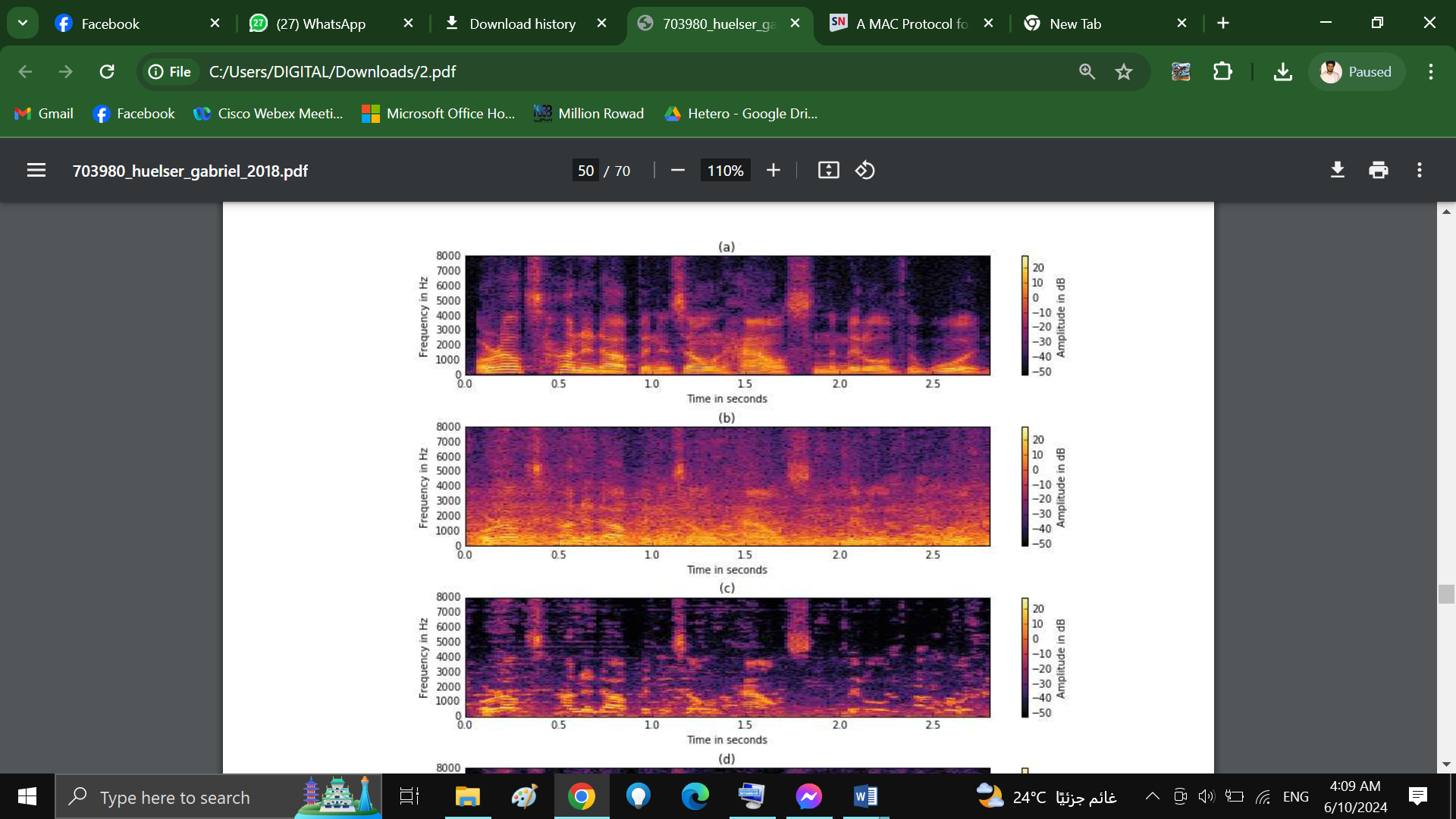
****

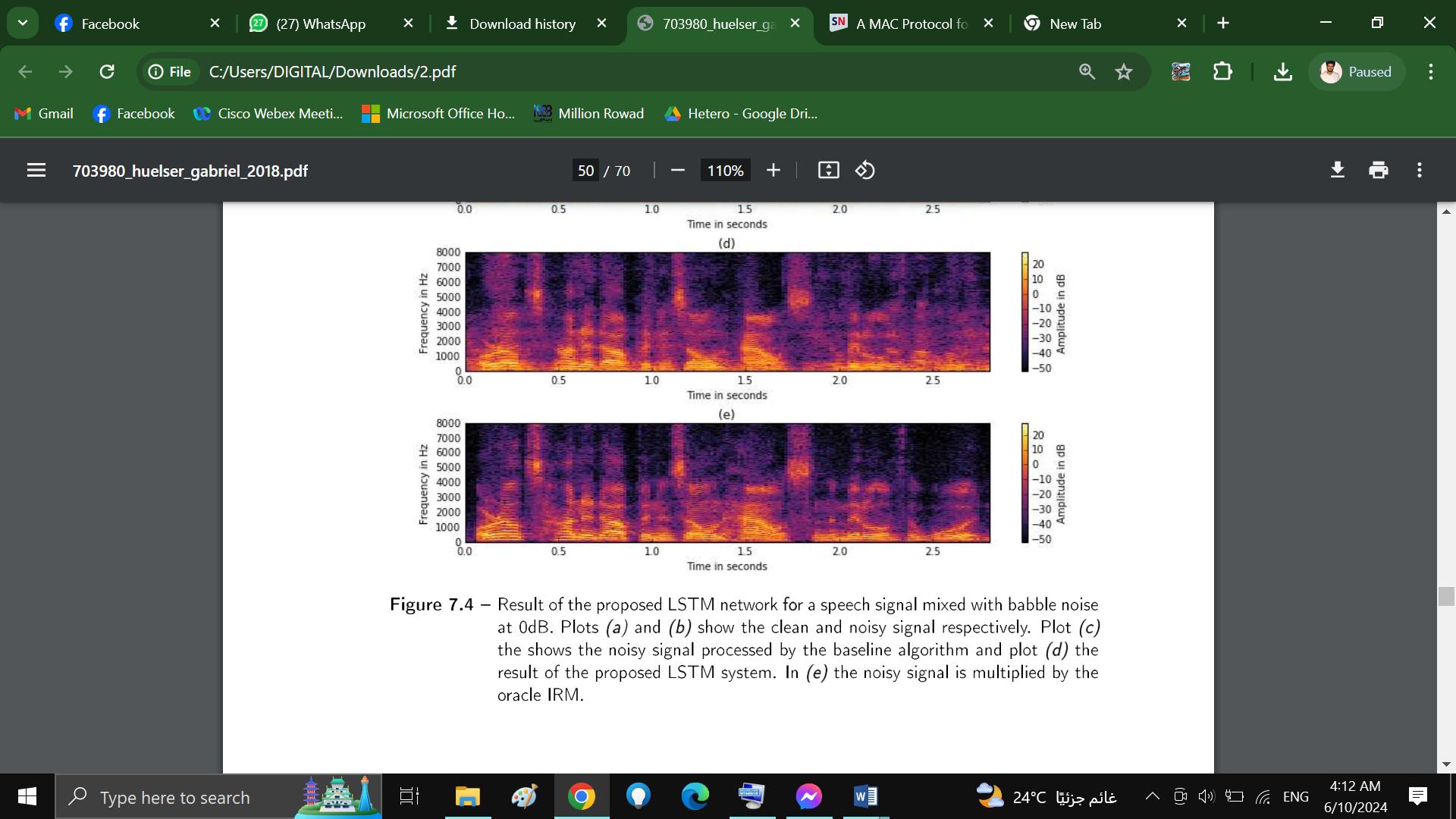
You can listen to one element in the batch (chosen randomly) using the widgets below.

Noisy signal

Target signal

Denoised signal





**Figure 7.4** - Result of the proposed LSTM network for a speech signal mixed with babble noise at OdB. Plots *(a)* and *(b)* show the clean and noisy signal respectively. Plot *(c)* the shows the noisy signal processed by the baseline algorithm and plot *( d)* the result of the proposed LSTM system. In *(e)* the noisy signal is multiplied by the oracle IRM.

In table 7.2 the PESQ and STOI improvements are indicated for each of the noise cate­ gories included in the test set. It can be seen that the biggest improvements compared to the baseline system are achieved for noise categories, which involve transient signals

such as 'machine gun' and 'workshop'. Categories for which the differences between the proposed model and the baseline algorithm are not significant, e.g. the categories 'f16' and 'leopard', typically consist of more stationary and narrow-band noise. The drop in

performance for these noise types could be explained by the fact that the predictions of the proposed system are based on an acoustic context spanning 11 overlapping frames, which in some cases may not be enough to distinguish between the harmonic structure

of a vowel and that of a sinusoidal masker. This issue could be addressed by applying a recursive evaluation scheme for the mask weights similar to [3] [76], by which information from the preceding mask predictions are taken into account.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **noise type** | **method** | **PESQ** | **STOI** |  |
|  | babble | LSTM | 0.3 (±0.06) | 0.02 (±0.005) |  |
|  |  | baseline | 0.1 (±0.03) | -0.03 (±0.005) |  |
|  | buccaneer | LSTM | 0.5 (±0.07) | 0.04 (±0.005) |  |
|  |  | baseline | 0.2 (±0.04) | -0.01 (±0.003) |  |
|  | destroyer engine | LSTM | 0.4 (±0.06) | 0.04 (±0.006) |  |
|  |  | baseline | 0.3 (±0.05) | 0.01 (±0.002) |  |
|  | destroyer ops | LSTM | 0.4 (±0.06) | 0.03 (±0.004) |  |
|  |  | baseline | 0.2 (±0.03) | -0.02 (±0.002) |  |
|  | f16 | LSTM | 0.5 (±0.06) | 0.05 (±0.006) |  |
|  |  | baseline | 0.3 (±0.05) | 0.01 (±0.003) |  |
|  | factory | LSTM | 0.5 (±0.07) | 0.04 (±0.006) |  |
|  |  | baseline | 0.2 (±0.04) | -0.02 (±0.003) |  |
|  | leopard | LSTM | 0.4 (±0.06) | 0.03 (±0.004) |  |
|  |  | baseline | 0.4 (±0.05) | -0.01 (±0.001) |  |
|  | machine gun | LSTM | 0.3 (±0.04) | 0.03 (±0.006) |  |
|  |  | baseline | -0.1 (±0.02) | -0.01 (±0.001) |  |
|  | workshop | LSTM | 0.5 (±0.07) | 0.05 (±0.007) |  |
|  |  | **baseline** | **-0.0 (±0.04)** | **-0.01 (±0.003)** |  |

**Table 7.2** - Comparison of the proposed LSTM network with the baseline algorithm. The results of the measured evaluation criteria are shown individually for each of the categories from the test. The 95% confidence interval is given in brackets next to the mean value.

* + 1. **Comparison of Features**

In this experiments different feature target combinations for the described network were implemented and evaluated. The phase sensitive filter as described in chapter 5 was used in [36] in the context of speaker seperation. The authors reported the PSF to perform better than the IRM in terms of signal to distortion ratio. Similar to the experiments in chapter 5 the spectral masks were compared in an oracle scenario, meaning that the optimum masks calculated from the reference signal were used. However, it is not proven that these results correlate with the suitability of the respective time-frequency masks as learning targets for deep learning models.

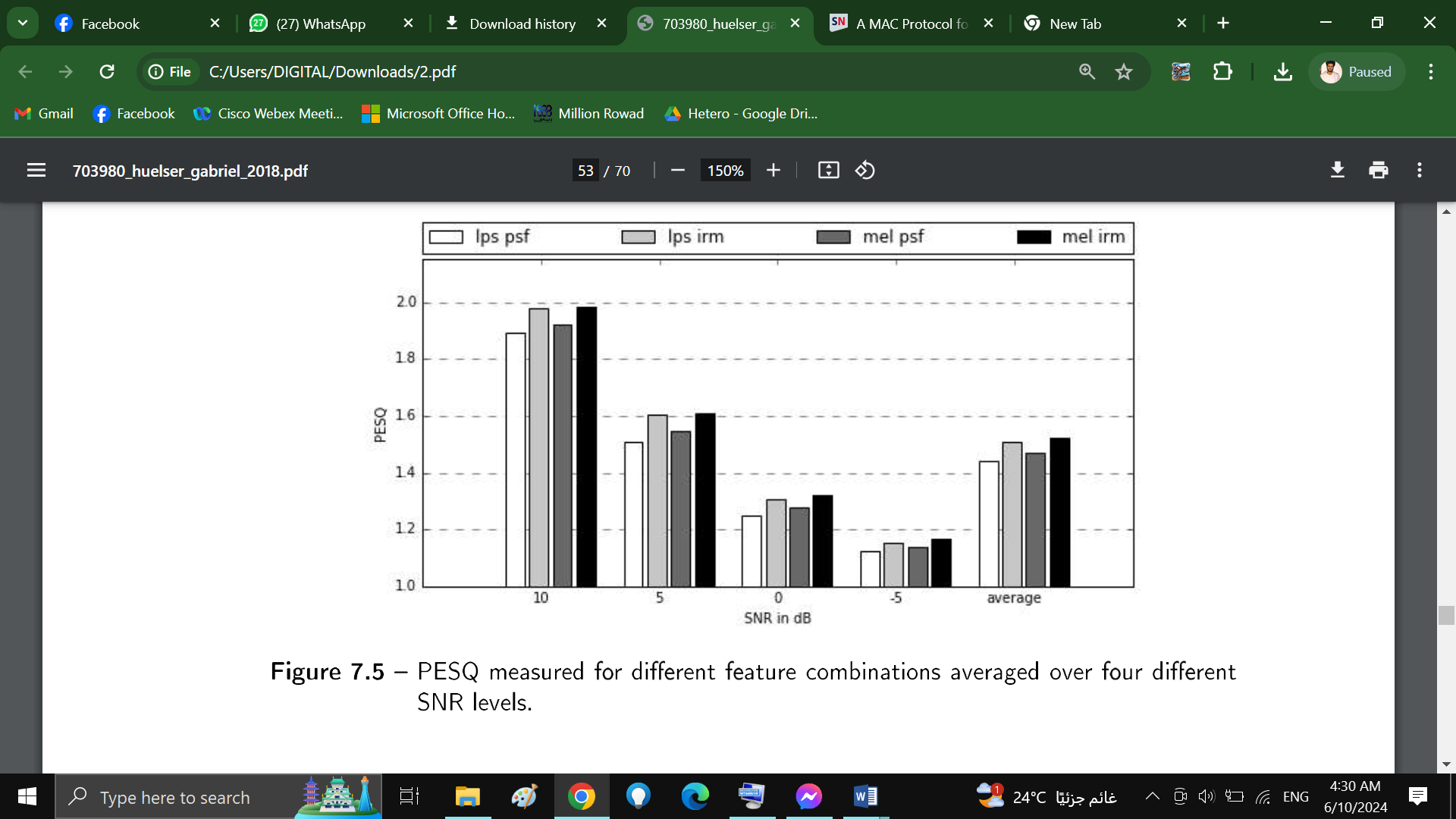
Table 7.3 shows the result for four different combinations. The column 'features' rep­ resents the input features to the neural network model, the column 'targets' represents the type of spectral mask used as the output feature.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | model | features | targets | ace[%] | 6PESQ | 6STOI |  |
|  | LSTM 2x512 | lps | psf | 64 | 0.3 | 0.04 |  |
|  | LSTM 2x512 | lps | 1rm | 70 | 0.3 | 0.04 |  |
|  | LSTM 2x512 | mel | psf | 67 | 0.3 | **0.05** |  |
|  | LSTM 2x512 | mel | 1rm | **72** | **0.4** | 0.04 |  |

**Table 7.3** - Objective evaluation of different feature-target combinations.

The results are fairly similar for all combinations in terms of PESQ and STOI and the best word accuracy scores are obtained using IRM as output target. The observed su­ periority of the IRM over the PSF mask contradicts the oracle experiment in chapter 5, where the PSF performed best in terms of PESQ and SNR. This leads to the assumption that although the PSF is theoretically better in terms of the resulting speech quality, the IRM is more suitable as a learning target since it only requires information about the magnitude, which is more or less explicitly provided by the input features. In contrast, the phase information included in the PSF is not explicitly given by the model input and can only be estimated from the harmonic structure of the input spectrum. In table 7.3 it can also be seen that in general, mel spectra achieve better results than log power spectra except for the STOI values, where PSF masks yield the best results. However, from the best to the worst result for the STOI metric the difference is only 0.8%.

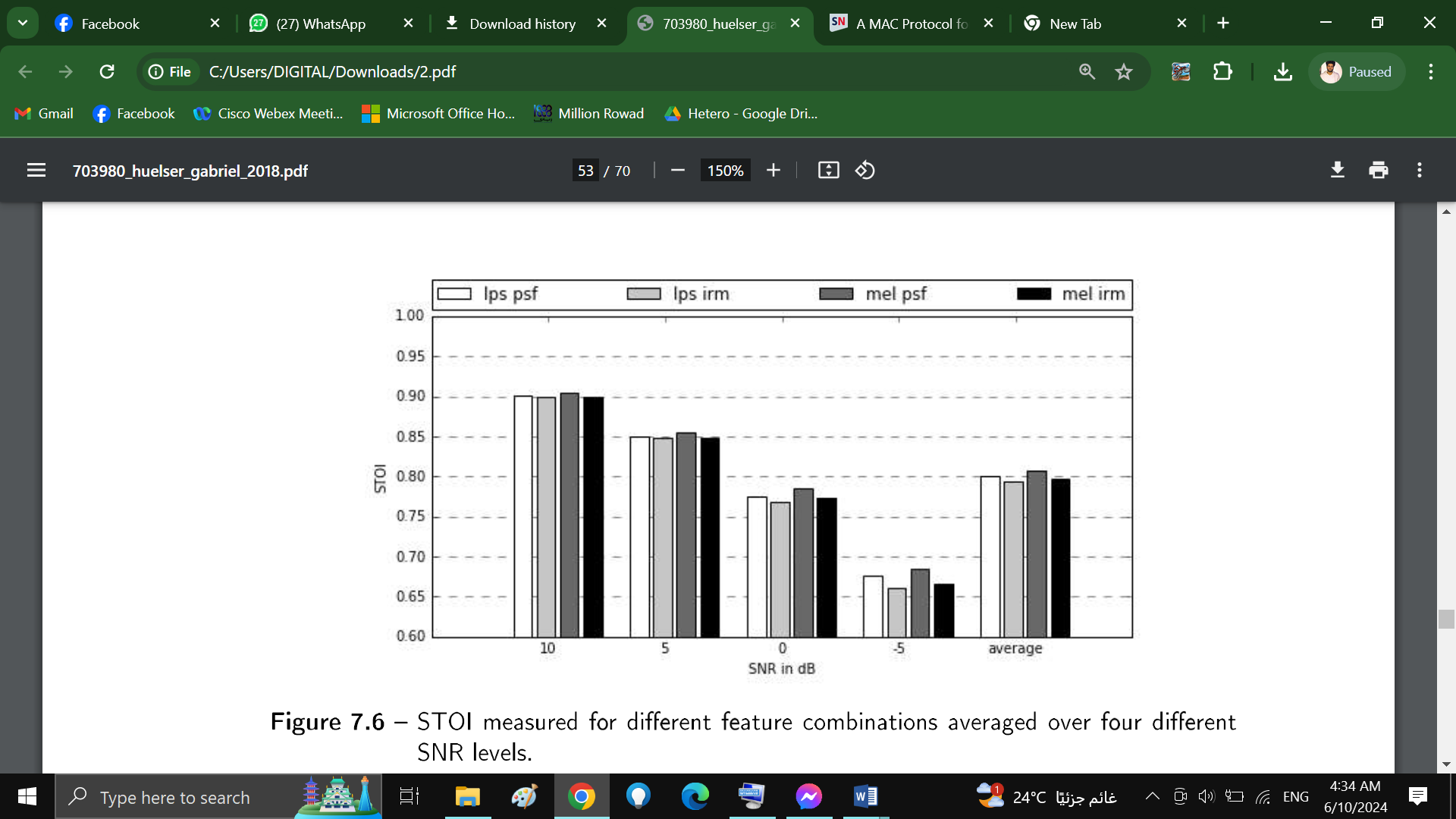
Figure 7.5 shows the PESQ measurements in detail. The results are averaged over each othe four different SNR levels.



**Figure 7.5** - PESQ measured for different feature combinations averaged over four different SNR levels.

It can be observed that the relations between the different feature combinations stay fairly equal over all noise levels, since the tendencies are independent from the underlying SNR level. However, since the results are more articulate for higher SNR levels, it can be concluded that, the lower the SNR of the test signal, the lower the differences between the feature combinations.

The results for the STOI metric is shown in figure 7.6. As already observed in table 7.3, the results are very similar for all four feature combinations, with the combination mel spectra and phase sensitive filter performing best for each of the four SNR levels.



**Figure 7.6** - STOI measured for different feature combinations averaged over four different SNR levels.

## Comparison of Layer Sizes

For this experiments the above mentioned network consisting of two LSTM layers and a sigmoid output layer was used to evaluate the performance for different layer sizes. Mel spectra were used as input features and ideal ratio masks as output features, since this configuration showed the most promising results in the previous experiment. Table 7.5 shows the achieved speech accuracy score as well as the average results for PESQ and STOI for different numbers of hidden units in the first two layers.

When looking at the speech accuracy score it is clearly visible that the performance improves with the increase in the number of hidden units per layer.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | model | features | targets | ace[%] | ,0.PESQ | ,0.STOI |  |
| LSTM 2x64 | mel | Irm | 69 | 0.2 | 0.03 |  |
| LSTM 2x128 | mel | Irm | 69 | 0.3 | 0.03 |  |
| LSTM 2x256 | mel | Irm | 71 | 0.3 | 0.04 |  |
| LSTM 2x384 | mel | Irm | 70 | 0.4 | 0.05 |  |
| LSTM 2x512 | mel | Irm | 72 | 0.4 | 0.04 |  |
|  | LSTM 2x1024 | mel | Irm | **74** | **0.4** | **0.05** |  |

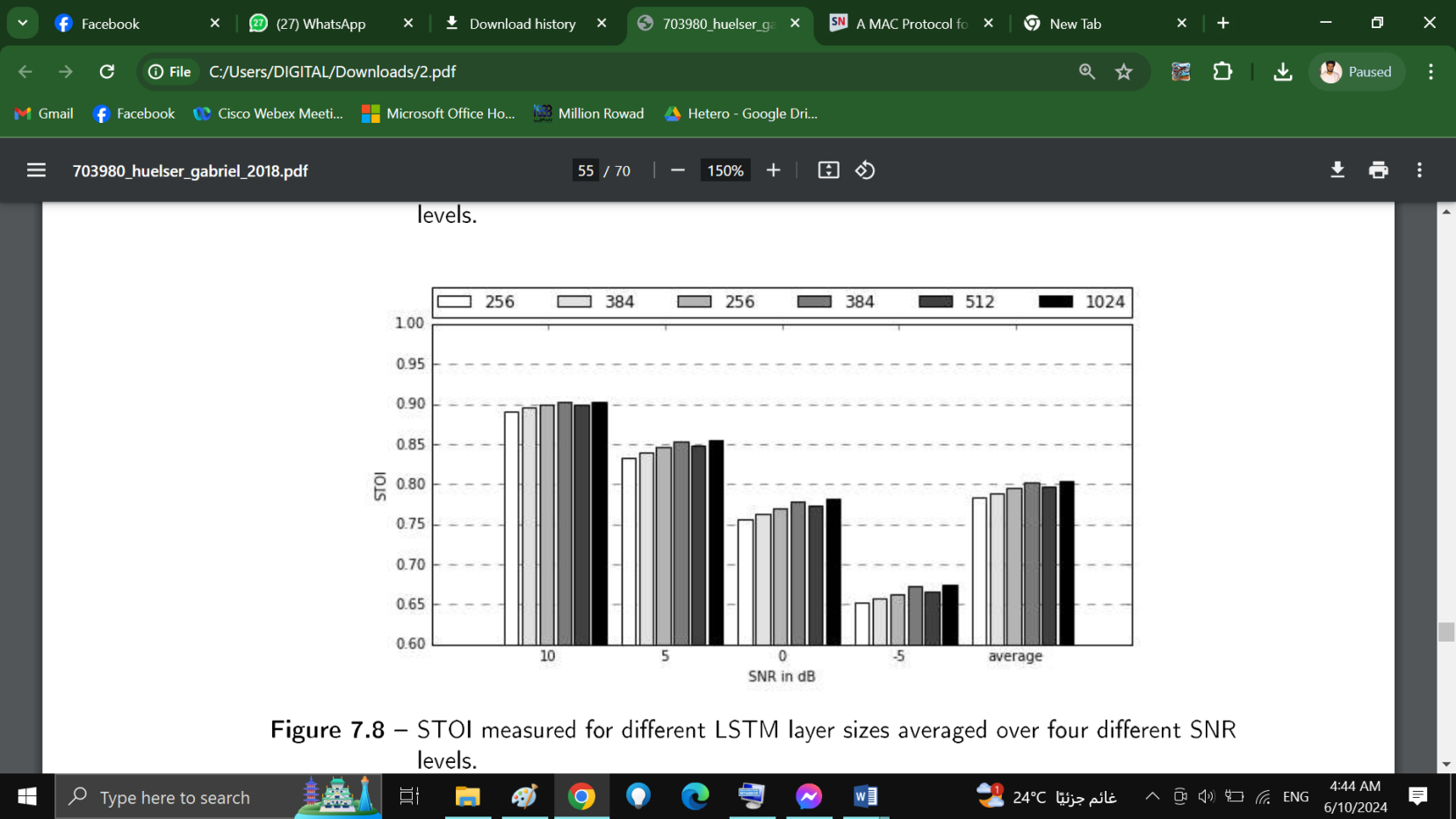
**Table 7.4** - Objective evaluation of different layer sizes

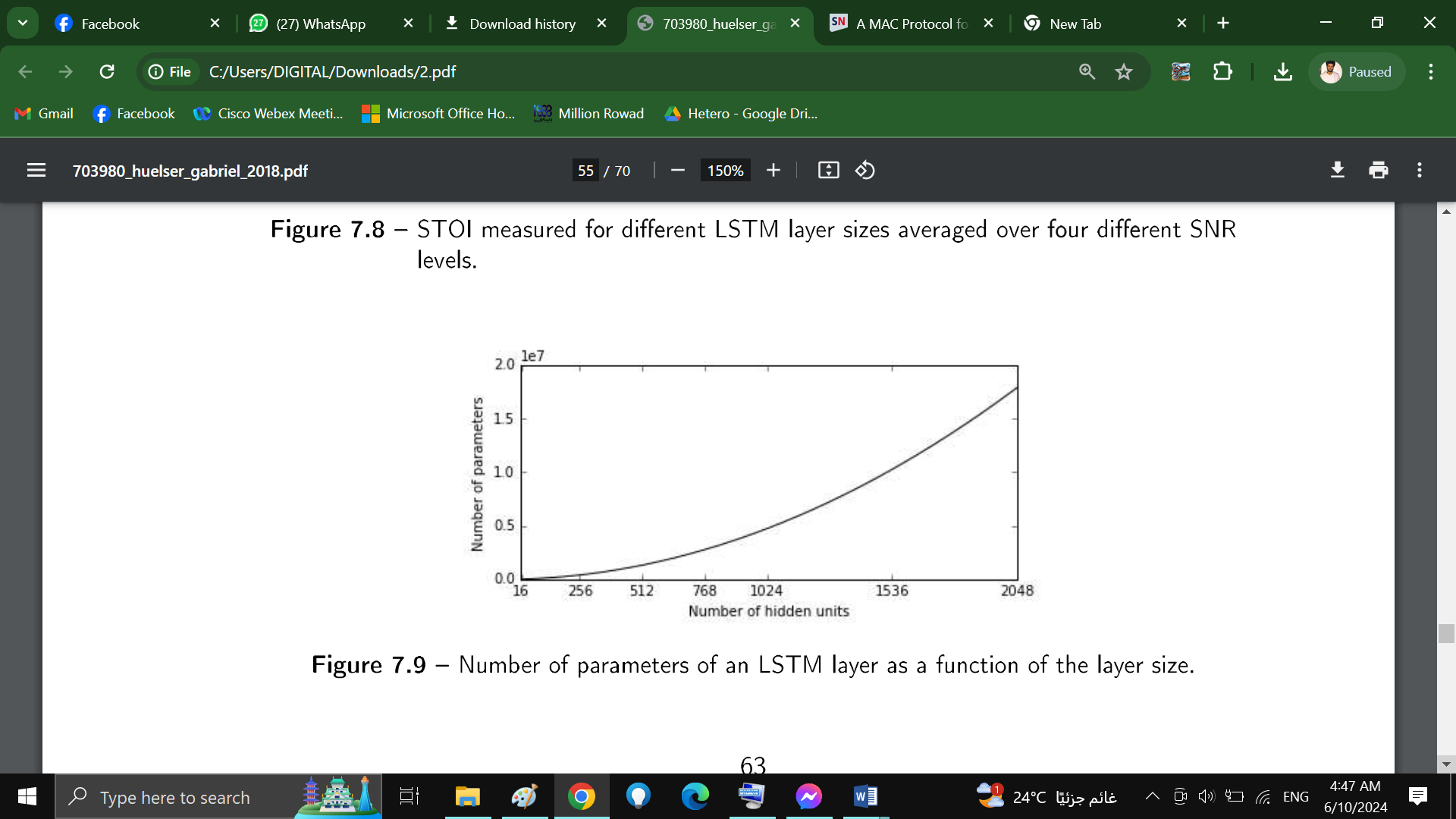
The other metrics, also shown in figures 7.7 and 7.8, show a clear rise in performance from 64 to 256 hidden units. lnterestingely PESQ and STOI show a dip in performance at 2x512 hidden units while the development stays linear for the speech accuracy score at this configuration.

However, above 384 hidden units a saturation can be observed, with only minor differ­ ences in terms of PESQ and STOI. This is an important observation when considering that total number of parameters increases exponentially in relation to the number of hidden units (see figure 7.9). As an example, an increase from 256 to 512 hidden units per layer results in an increase from 985,601 to 3,543,809 total parameters for the corresponding NN model.



**Figure 7.7** - PESQ measured for different LSTM layer sizes averaged over four different SNR levels.

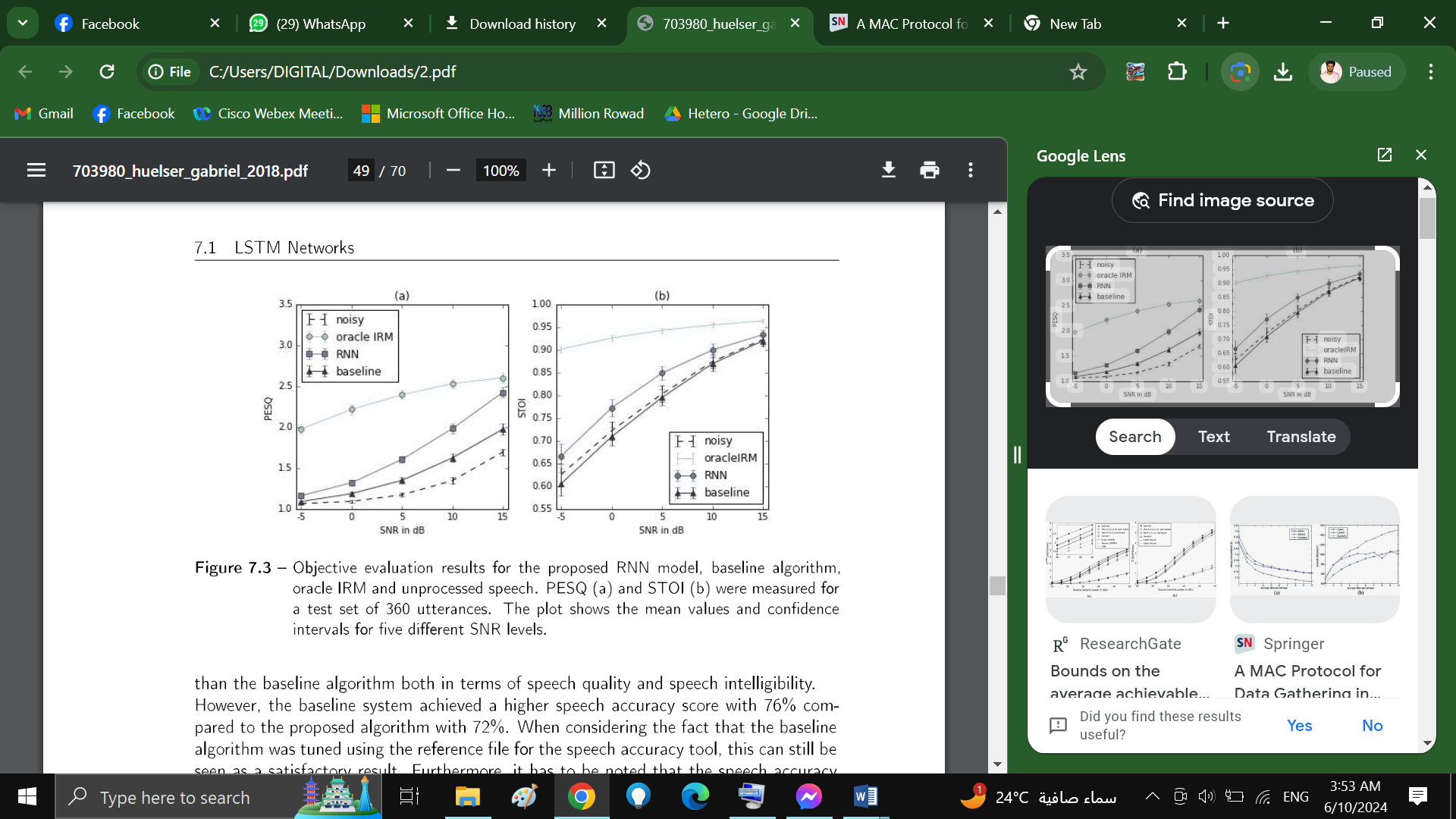
 **Figure 7.8** - STOI measured for different LSTM layer sizes averaged over four different SNR levels.



**Figure 7.9** - Number of parameters of an LSTM layer as a function of the layer size.

## **7.2 Bidirectional LSTMs**

The above stated network structure is now modified to employ bidirectional LSTMs, since they are considered to produce better results especially in the context of speech processing [52], [36], [53]. Figure 7.10 shows the measurements for a bidirectional LSTM network consisting of two layers with 512 hidden units each. Since each layer incorporates a forward layer as well as a backward layer the number of hidden units doubles, resulting in a total of 1024 trainable units. As a reference the measurements are compared to that of a LSTM network with 1024 hidden units per layer. Again the mismatched test set consisting of 360 utterances is used for evaluation and the results are given as the mean value over each of the four different SNR levels. It can be observed



**Figure 7.10** - Objective evaluation results for an LSTM network with 1024 hidden units per layer, BLSTM network with 512 hidden units per layer and unprocessed speech. PESQ (a) and STOI (b) were measured for a test set of 360 utterances.

that the bidirectional network performs slightly better than the standard LSTM network. In comparison to the noisy test signals, the BLSTM model achieved an improvement of

0.54 in terms of PESQ and 0.05 in terms of STOI. In comparison the LSTM improved the PESQ by 0.46 and the STOI by 0.04. The measurements correlate with the speech accuracy score where the BLSTM network obtains 74.1% compared to 73.7% for the LSTM network. Furthermore it should be noted that the BLSTM network has a smaller parameter space than the LSTM network with 9.4 million compared to around 13.4 million. This means that the use of bidirectional layers improves the performance while reducing the total number of parameters of the NN model, which is especially important in case the hardware resources are limited e.g. for FPGA implementations.

### **Comparison of Layer Sizes**

Different feature combinations namely mel/psf and mel/irm as well as different layer sizes were investigated similar to the previous experiments. The results are shown in table 7.5. Note that in the case of bidirectional LSTMs the number of hidden units doubles since each layer consists of one forward and one backward layer. In table 7.5 the total number of hidden units per layer is given in brackets.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | model | features | targets | ace [%] | PESQ | STOI |  |
| BLSTM | 2x192 (384) mel | psf | 63 | 0.3 | 0.06 |  |
| BLSTM | 2x256 (512) mel | psf | 67 | 0.4 | 0.06 |  |
| BLSTM | 2x512 (1024) mel | psf | **71** | 0.4 | 0.07 |  |
| BLSTM | 2x768 (1536) mel | psf | 68 | 0.4 | 0.06 |  |
| BLSTM | 2x32 (64) mel | 1rm | 64 | 0.2 | 0.03 |  |
| BLSTM | 2x64 (128) mel | 1rm | 69 | 0.3 | 0.04 |  |
| BLSTM | 2x128 (256) mel | 1rm | 71 | 0.4 | 0.06 |  |
| BLSTM | 2x192 (384) mel | 1rm | 72 | 0.4 | 0.06 |  |
| BLSTM | 2x256 (512) mel | 1rm | **74** | 0.4 | 0.06 |  |
| BLSTM | 2x512 (1024) mel | 1rm | 72 | **0.5** | 0.06 |  |
|  | BLSTM | 2x768 (1536) mel | 1rm | 71 | 0.5 | **0.06** |  |

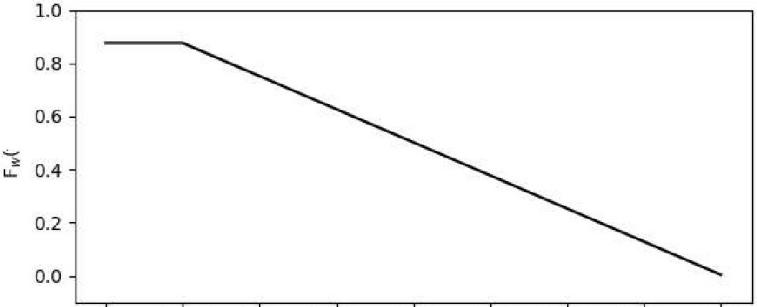
**Table 7.5** - Objective evaluation of different layer sizes and feature combinations. Each BLSTM layer consists of one forward and one backward layer which doubles the number of hidden units. The total number of hidden units is given in brackets.

When considering only the speech accuracy score both configurations seem to have an optimum layer sizes after which the performance starts to decrease. For the combina­ tion mel spectra/psf masks this is 2x512 hidden units and for mel spectra/irm masks the optimal size is smaller with 2x256. This indicates that the phase sensitive filter requires more complicated networks to be estimated precisely than the ideal ratio mask, which can be explained by the fact that psf masks incorporate additional phase information which is not explicitely contained by the mel spectrum.

When looking at the PESQ and STOI measurements, the performance linearly increases with the number of hidden units per layer, unlike the speech accuracy there is no observ­ able border indicating a decrease in performance.

As mentioned before, one should also consider the number of parameters of the resulting network when choosing the layer size, since it increases exponentially with the number of hidden units per layer.

* 1. **Weighted Loss Function**



0 1000 2000 3000 4000 5000 6000 7000 8000

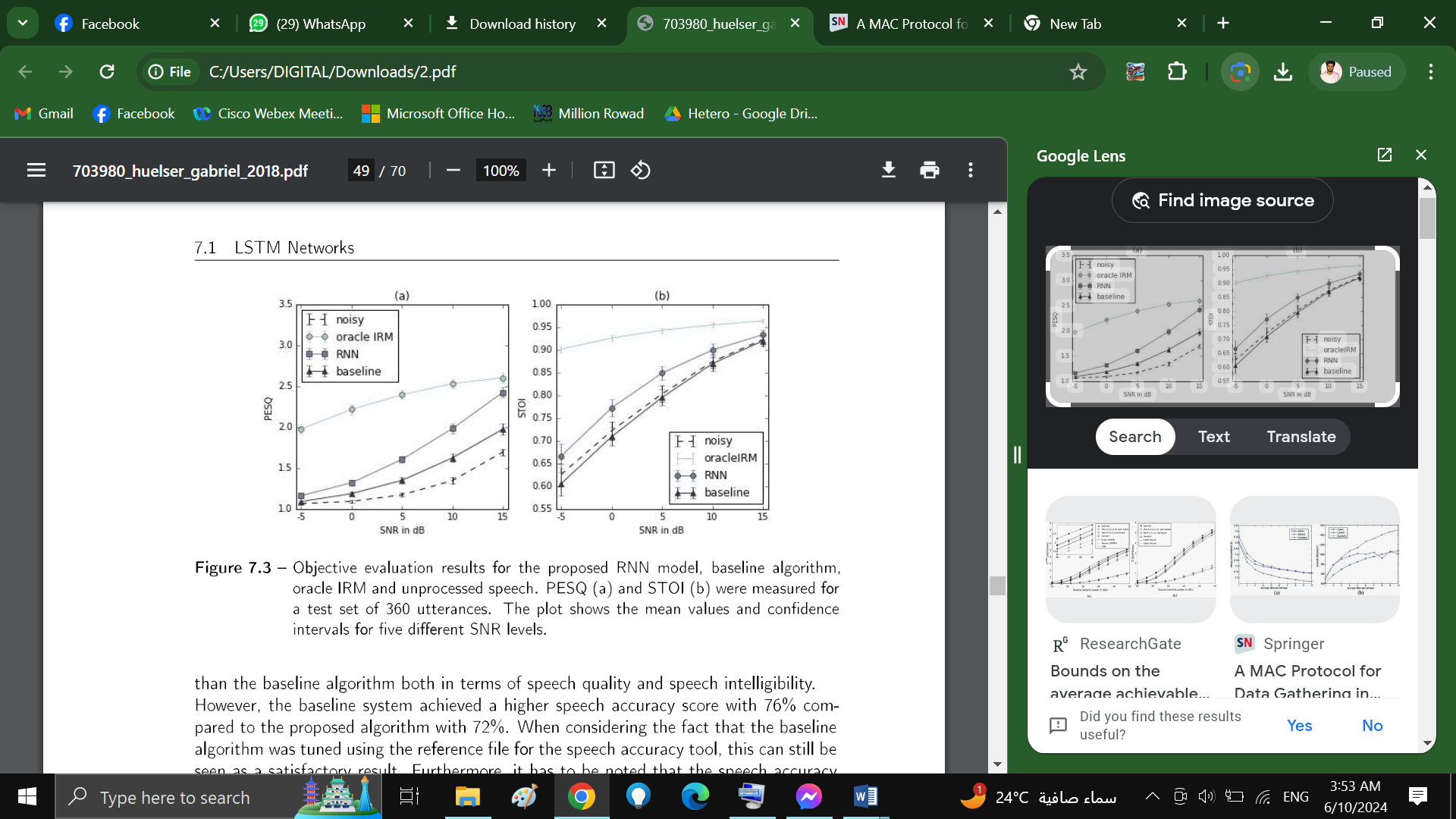
fin Hz

**Figure 7.11** - Loss weight as a function of frequency.

pitch increments correspond to equally spaced pitch intervals on the frequency scale up to a point between 500 and 1000 Hz [55], [56]. The frequency weighting has the consequence that the error of a single frequency bin corresponding to a low frequency is more significant than that of a single high frequency bin. This not only matches the frequency resolution of the human auditory system, but also helps to increase the quality of voiced speech which is mainly located between 100 and 4000 Hz.

Figure 7.12 shows the PESQ and STOI measurements in comparison to the same BLSTM model trained using a conventional MSE loss.

Additionally, in table 7.6 the *Cortana* speech accuracy as well as the average improve-



**Figure 7.12** - PESQ and STOI measurements for a 2x512 BLSTM network trained with the weighted cost function (wblstm) and the conventional MSE loss (blstm).

ments in terms of PESQ and STOI are compared for two different BLSTM strtuctures with and without the modified MSE loss.

It can be observed that while the frequency weighting of the cost function results in an increase of up to 3% in speech accuracy, the differences in terms of PESQ and STOI are minimal and even indicate a slight decrease in performance in the case of the weighted cost function.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | model | features | targets | ace [%] | PESQ | STOI |  |
|  | BLSTM 2x256 (512) | mel | Irm | 74 | 0.4 | 0.06 |  |
|  | wBLSTM 2x256 (512) | mel | Irm | **76** | 0.4 | 0.06 |  |
|  | BLSTM 2x512 (1024) | mel | Irm | 72 | **0.5** | 0.06 |  |
|  | wBLSTM 2x512 (1024) | mel | Irm | 75 | 0.4 | **0.07** |  |

**Table 7.6** - Objective evaluation of the weighted loss function. While BLSTM refers to a bidirectional network trained using a conventional MSE loss, wBLSTM denotes the same network structure trained using the above mentioned weighted MSE.

## **Matched Training**

To further improve the speech accuracy score, noise data extracted from the speech platform test file is used to form an extended training set. For this, hand labeled sec­ tions from the test file, in which no speech is present are mixed with clean speech from the dataset. This resulted in additional training data consisting of 6000 utterances (17 hours), increasing the total number of utterances included in the training data from 48,000 to 54,000.

Using the matched training data, the system improves by up to 8% in additional speech accuraccy. Table 7.7 shows the detailed results with and without the extended training set. It can be seen that the matched training set has no effect on the PESQ and STOI measurements for the 360 utterances contained in the test set, this means that the generalization capabilities are not effected by an extension of the training data while the improvements are large in case of the particular test case.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | model | features | targets | train data | ace [%] | PESQ | STOI |  |
|  | BLSTM 2x256 | mel | 1rm | unmatched | 74 | 0.4 | 0.06 |  |
|  | BLSTM 2x256 | mel | 1rm | matched | 80 | 0.4 | 0.06 |  |
|  | wBLSTM 2x256 | mel | 1rm | unmatched | 76 | 0.4 | 0.06 |  |
|  | wBLSTM 2x256 | mel | 1rm | matched | 82 | 0.4 | 0.06 |  |
|  | wBLSTM 2x512 | mel | 1rm | unmatched | 75 | 0.4 | 0.07 |  |
|  | wBLSTM 2x512 | mel | 1rm | matched | 83 | 0.4 | 0.07 |  |

**Table 7.7** - Objective evaluation for different RNN models with and without the matched training set.

Table 7.8 summarizes the results from the previous experiments, LSTM, BLSTM, BLSTM with weighted MSE and the same model trained with the extended training data are com­ pared in terms of speech accuracy, PESQ and STOI improvement. It can be observed that the adaption of the training data to match the *Cortana* reference file brings the largest improvement in terms of speech accuracy, which clearly shows that the configu­ ration of the training data is more effective and more important than fine-tuning of the network topology, hyper-parameters or training procedure.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | model | train data | ace [%] | PESQ | STOI |  |
| LSTM 2x512 | unmatched | 72 | 0.4 | 0.04 |  |
| BLSTM 2x256 | unmatched | 74 | 0.4 | 0.06 |  |
| wBLSTM 2x256 | unmatched | 76 | 0.4 | 0.06 |  |
|  | wBLSTM 2x256 | matched | 82 | 0.4 | 0.06 |  |

**Table 7.8** - Comparison of different optimization approaches. The matched training data brings the biggest improvements in terms of speech accuracy.

## **7.5 Convolutional Neural Networks**

For this experiment a convolutional neural network based on the structure used in [38] is implemented because promising results have been reported. Figure 7.13 shows the basic structure of the model, which is made up of two 2D convolutional layers with a max pooling operation in between. The stride of the convolutional layers is set to [1,1] and that of the pooling layer to [2,2]. After the second convolutional layer the output is flattened an fed into the first fully-connected layer of the feed-forward part. After two layers with 1024 hidden units using Relu activation follows the sigmoid output layer. The training procedure is the same as in the previous experiments, except that the input is given as matrices of size 128 x 11 instead of single time-step vectors, as it is the case for the mentioned LSTM models.

**Input:** 128xll mel bins

**convolutional layer 1:** 52 kernel [5xl]

**max pooling:** [4x4]

**convolutional layer 2:** 78 kernel [5xl]

**dense layer:** 1024 units, Relu

**dense layer:** 1024 units, Relu

**output layer:** 1024 units, sigmoid

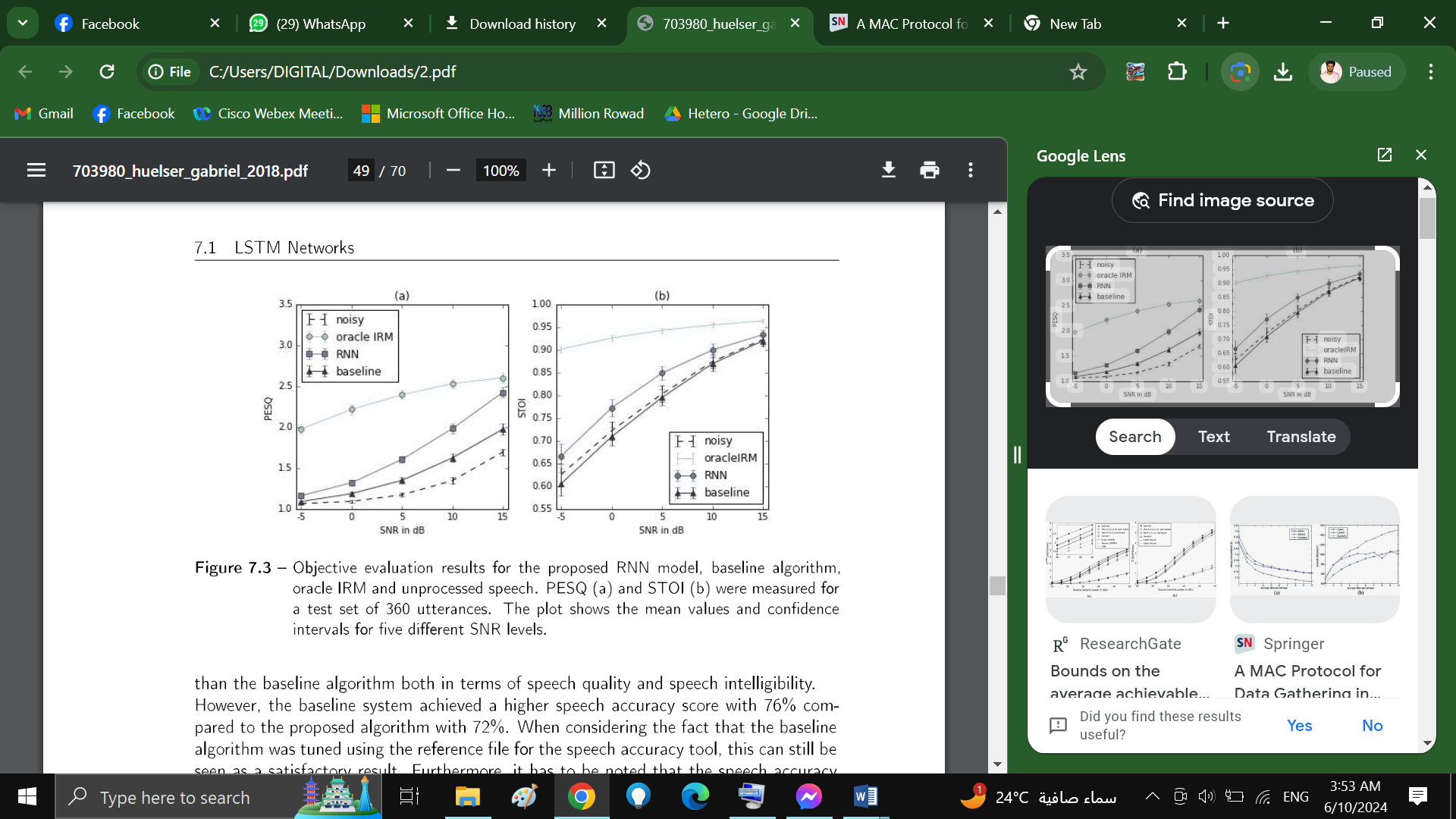
**Output:** 257 IRM bins

**Figure 7.13** - Structure of the investigated CNN.

Table ?? shows the results achieved with the CNN compared to the baseline method and the 2x256 BLSTM network. It can be observed that, although the measurements in terms of PESQ and STOI show a better performance of the CNN model compared

to the baseline system, the RNN performs significantly better than the CNN. More sig­ nificantly, the resulting speech accuracy of 43% is only +1% higher than that of the

unprocessed cortana reference file which scores at 42% accuracy. The poor performance of the CNN is logical when considering the fact that they are inspired by the princi­ ples of visual perception and are mostly used for image processing tasks such as object recognition. Furthermore, the employed filter kernels are only able to represent patterns along the frequency axis and not along the time axis, which means that the temporal



**Figure 7.14** - PESQ (a) and STOI (b) measurements for the proposed CNN model compared to a BLSTM with 256 hidden units per layer, the baseline algorithm and the corresponding unprocessed speech signals.

information can only be modeled by the two dense layers on top of the convolutional layers. It should also be noted that the described structure employs a large number of

|  |  |  |  |
| --- | --- | --- | --- |
| method | ace [%] | .6.PESQ | .6.STOI |
| baseline | **77** | 0.2 | -0.01 |
| BLSTM 2x256 | 74 | **0.5** | **0.05** |
| CNN 2x[5,1] | 43 | 0.4 | 0.04 |

**Table 7.9** - Objective evaluation of the investigate CNN structure compared to a BLSTM with 256 hidden units per layer and the baseline algorithm.

parameters compared to the above mentioned RNNs. In sum the CNN has a total of 19,545,535 parameters compared to 3,543,809 parameters used for an LSTM network with two layers of 512 hidden units. Although the number of parameters could be re­ duced by optimizing the network structure, e.g. the kernel sizes, stride size or number of filter kernels, it is unclear if the model will achieve results comparable to those by the proposed RNN models. Nevertheless, the advantage of CNNs in comparison to RNNs is that the computational effort needed for the prediction of a single mask frame is much lower, since only a single forward pass through the network is necessary as opposed to 11 sequential passes needed by the proposed LSTM network.

In this chapter the experimental results for different network topologies and training procedures are presented and discussed. The next chapter concludes this thesis by summarizing the results and giving an outlook on future potential of the findings.

**Conclusion and future work**

As a final remark, the feasibility of a real-time implementation of the proposed algorithm for the use within mobile devices such as smartphones or cloud speakers, has yet to be explored. Most of the presented algorithms employ neural network models defined by a large number of parameters, which would require too much memory to be implemented on conventional DSP chips. But even if enough memory would be available, a real-time implementation would still be challenging since the temporal context of the input data has to be processed sequentially. However, with the growing interest in artificial intelligence and deep learning, it can be expected that more and more research and development will be done in terms of custom hardware for neural network inference and training.

**future work**

**Sound Enhancement:**

Sound enhancement refers to the process of improving the quality and clarity of audio signals. This can involve various techniques such as:

Noise Reduction: Removing or reducing unwanted background noise, hiss, or interference from the audio signal to improve the signal-to-noise ratio.

Equalization: Adjusting the relative volume of different frequency ranges to enhance the overall balance and fidelity of the sound.

Dynamic Range Compression: Reducing the dynamic range of the audio signal to bring up the level of quieter parts and prevent clipping of louder parts.

Stereo Widening: Enhancing the perceived spatial separation and depth of the audio, creating a more immersive listening experience.

Room Correction: Compensating for the acoustic characteristics of the listening environment to improve the perceived sound quality.

**Video Enhancement:**

Video enhancement focuses on improving the visual quality and clarity of video content. Some common techniques include:

Resolution Upscaling: Increasing the spatial resolution of the video, making it appear sharper and more detailed.

Noise Reduction: Removing or reducing unwanted visual noise, grain, or artifacts to improve the overall image quality.

Color Correction: Adjusting the color balance, saturation, and contrast to enhance the visual appeal and accuracy.

Sharpening: Applying edge-enhancement algorithms to make the video appear crisper and more defined.

Frame Rate Interpolation: Generating additional intermediate frames to create a smoother and more fluid motion.

As for potential future work in this area, here are some ideas to consider:

**Multimodal Integration:** Exploring ways to integrate sound enhancement and video enhancement techniques to create a more cohesive and immersive multimedia experience.

**Adaptive Algorithms:** Developing intelligent algorithms that can dynamically adapt the enhancement processes based on the content and user preferences.

**Machine Learning-based Approaches:** Leveraging the advancements in machine learning and deep learning to further improve the quality and accuracy of sound and video enhancement.

**Real-time Processing:** Enhancing the efficiency and speed of the enhancement algorithms to enable real-time processing, particularly for live streaming and interactive applications.

**Cross-device Optimization:** Ensuring that the enhanced audio and video content can be seamlessly delivered and experienced across a wide range of devices, from smartphones to high-end displays.

**Personalization and Customization:** Allowing users to customize the enhancement settings and preferences to better suit their individual needs and preferences.

**Reference**

- IEEE Xplore Digital Library - This is a great resource for accessing technical papers and articles on speech enhancement from the IEEE. You can search by keywords or browse by relevant topics.

- Google Scholar - Searching for terms like "speech enhancement" or specific techniques can surface academic papers and studies on the topic.

- Github - There may be open source speech enhancement projects or libraries with associated documentation.

- Manufacturer/vendor websites - Companies working on speech enhancement technologies may have technical whitepapers or user guides available.

- Online tutorials and blog posts - Searching for tutorial-style content can sometimes uncover helpful overviews and explanations.

- The REVERB workshop website - This focuses on speech enhancement for reverberant environments: https://reverb2014.dereverberation.com/

- The International Speech Communication Association (ISCA) website - They have a special interest group on speech enhancement: https://www.isca-speech.org/iscaweb/

- The IEEE Signal Processing Society's Technical Committee on Audio and Acoustic Signal Processing-They cover speech enhancement research: https://signalprocessingsociety.org/get-involved/technical-committees/audio-and-acoustic-signal-processing

- The IWAENC (International Workshop on Acoustic Echo and Noise Control) website - They cover speech enhancement research: https://iwaenc.org/

- The ICASSP (IEEE International Conference on Acoustics, Speech and Signal Processing) website - They have a speech enhancement track: https://2023.ieeeicassp.org/

- The arXiv preprint repository - Searching for "speech enhancement" can surface recent research papers: https://arxiv.org/

- DSPRelated.com

- Mathworks.com

- The websites of academic institutions with relevant research groups

- "Speech Enhancement: Theory and Practice" by Philipos C. Loizou

- "Multirate Systems and Filter Banks" by P. P. Vaidyanathan

- "Speech and Audio Signal Processing: Processing and Perception of Speech and Music" by Ben Gold and Nelson Morgan