Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Dept. of Telecommunications and Media Informatics

Ultrasound-based Silent Speech Interface using Deep Learning

BSc Thesis

Author
Eloi Moliner Juanpere

Supervisor
dr. Tamás Gábor Csapó

May 18, 2018

Contents

STUDENT DECLARATION

I, Eloi Moliner Juanpere, the undersigned, hereby declare that the present BSc thesis work has been prepared by myself and without any unauthorized help or assistance. Only the specified sources (references, tools, etc.) were used. All parts taken from other sources word by word, or after rephrasing but with identical meaning, were unambiguously identified with explicit reference to the sources utilized.

I authorize the Faculty of Electrical Engineering and Informatics of the Budapest University of Technology and Economics to publish the principal data of the thesis work (author's name, title, abstracts in English and in a second language, year of preparation, supervisor's name, etc.) in a searchable, public, electronic and online database and to publish the full text of the thesis work on the internal network of the university (this may include access by authenticated outside users). I declare that the submitted hardcopy of the thesis work and its electronic version are identical.

Full text of thesis works classified upon the decision of the Dean will be published after a period of three years.

Budapest, May 18, 2018

Eloi Moliner Juanpere
hallgató

Abstract

Silent Speech Interface (*SSI*) is a technology able to synthesize speech in the absence of any acoustic signal. It can be useful in cases like laryngectomy patients, noisy environments or silent calls. This thesis explores the particular case of *SSI* using ultrasound images of the tongue as input signals. A 'direct synthesis' approach based on Deep Neural Networks and Mel-generalized cepstral coefficients is proposed.

This document is an extension of Csapó et al. "*DNN-based Ultrasound-to-Speech Conversion for a Silent Speech Interface*" [1] . Several deep learning models, such as the basic Feed-forward Neural Networks, Convolutional Neural Networks and Recurrent Neural Networks are presented and discussed. A denoising pre-processing based on a Deep Convolutional Autoencoder has also been studied. A considerable number of experiments using a set of different deep learning architectures and an extensive hyperperameter optimization study have been realized.

The different experiments have been testing and rating several objective and subjective quality measures. According to the experiments, an architecture based on a CNN and bidirectional LSTM layers has shown the best results in both objective and subjective terms.

Resum

Silent Speech Interface (*SSI*) és una tecnologia capaç de sintetitzar veu partint únicament de senyals no-acústiques. Pot tenir gran utilitat en casos com pacients de laringectomia, ambients sorollosos o trucades silencioses. Aquesta tčsis explora el cas particular de *SSI* utilitzant imatges de la llengua captades amb ultrasons com a senyals d'entrada. Es proposa un enfocament de 'síntesis directa' basat en Xarxes Neuronals Profundes i coeficients *Mel-generalized cepstral*.

Aquest document és una extensió del treball de Csapó et al. "*DNN-based Ultrasound-to-Speech Conversion for a Silent Speech Interface*"[1] . Diversos models de xarxes neuronals són presentats i discutits, com les bŕsiques xarxes neuronals directes, xarxes neuronals convolucionals o xarxes neuronals recurrents. També s'ha estudiat un pre-processat reductor de soroll basat en un *Autoencoder* convolucional profund. S'ha portat a terme un nombre considerable d'experiments utilitzant diverses arquitectures de *Deep Learning*, així com un extens estudi d'optimització d'hyperparŕmetres.

Els diferents experiments han estat evaluar i qualificar a partir de diferentes mesures de qualitat objectives i subjectives. Els millors resultats, tant en termes objectius com subjectius, els ha presentat una arquitectura basada en una CNN i capes bidireccionals de LSTMs.

Introduction

Silent Speech Interface, or SSI, refers to a system enabling speech communication to take place when an audible acoustic signal is unavailable [2]. This system must be able to produce a digital representation of speech, which can be synthesized directly, by acquiring non-acoustic data.

This technology is still in experimental stage but it has several potential applications.

- Persons who have undergone a laryngectomy. "Each year, thousands of persons worldwide fall victim to a cancer of the larynx or a neurological problem leading to loss of voice, constituting a serious social handicap. Current speech restoration methods have not changed significantly for over 35 years and are unsatisfactory for many people" [3].
- Older citizens for whom speaking requires a substantial effort.
- Noisy environments where auditive contact is problematic. The non-acoustic *biosignals* are more robust against noise degradation than the speech signal.
- *Silent calls* in order to preserve privacy when making phone calls in public areas. Or in situations where the user must not disturb bystanders (e.g in conferences, theatres, or concerts).
- Military applications where confidentiality is fundamental.

The early idea of silent communication backed to a lip-reading computer *HAL 9000* in 1968 in Stanley Kubrick science-fiction film *2001-A Space Odyssey*. The first real SSI system was introduced by *Petajan* in 1984 [4], it consisted of an automatic lip-reading system. After that, as it is explained in [2], experimental SSI systems based on several types of technology have been described in the literature:

- Real-time characterization of the vocal tract using ultrasound (US) and optical imaging of the tongue and lips [5], [6].
- Non-audible murmur microphones, NAM, placed on the neck [7].
- Capture of the movement of fixed points on the articulators using Electromagnetic Articulography (EMA) sensors [8].
- Analysis of glottal activity using electromagnetic or vibration sensors [9].

- Surface electromyography (sEMG) of the articulator muscles or the larynx [10].
- Interpretation of signals from electro-encephalographic (EEG) sensors [11].
- Cortical implants for a "thought-driven" SSI [12].

This thesis is only focused in the first technology cited, ultrasound and optical imaging of the tongue and lips. One of the first approaches of SSI based on ultrasound used a simple neural network to predict spectral parameters, which can directly synthesize speech using a vocoder ('direct synthesis') [5]; but it didn't succeed in achieving convincing results. Other previous approaches consisted of 'recognition-and-synthesis', dividing the process in two steps; automatic speech recognition (ASR) based on tongue movement to identify the words spoken, and then synthesizing speech by a text-to-speech (TTS) stage [13], [14]. Nevertheless, state-of-the-art systems use 'direct synthesis' for several reasons. Firstly, it has much lower latency. Studies on delayed auditory feedback [15] have shown that delays over 50 ms can introduce mental stress and cause dysfluences in speech; 'direct synthesis' can reach delays below 50 ms, while it is almost impossible in ASR and TTS. Secondly, ASR and TTS depend on the language and lexicon, while 'direct synthesis' does not, so it can be easier to record more training data. Another important reason is that para-linguistic aspects of speech (e.g. age, mood or gender) are normally lost after ASR but they could be recovered via direct synthesis. Lastly, it has to be borne in mind that, when using two different submodels (ASR and TTS) one followed by the other, the error adds up and could distort much more the final speech signal output.

Deep neural networks have become very popular in signal processing since 2006 [16], they have proved to produce accuracy comparable to human performance in several pattern recognition tasks. In the area of SSI, some researchers have investigated deep learning. Diener, Janke and Schultz used a DNN for surface electromyographic (EMG) speech synthesis [10]. An objective and subjective improvement was found comparing their DNN result with previous Gaussian mixture model (GMM) approaches. Gonzalez and his colleagues [8] investigated direct speech reconstruction using different techniques, Gaussian mixture models (GMM) , DNNs and recurrent neural networks (RNNs). They were using articulator movement sensor data as input. They reached the best performance with the RNN, in particular a bidirectional RNN, which makes use of both past and future inputs for computing the outputs. Jaumard-Hakounet al. designed an approach focused on singing using ultrasound tongue images and lip images [17]; a Deep Autoencoder was used to reduce dimensionality and, after this, a deep neural network was used to predict spectral features. In the literature some other approaches of SSI can be found, focused on deep learning methods [18],[19], [20], but there are few relating to ultrasound technology.

The thesis is divided in 6 chapters. Chapter 1 talks about the characteristics of the input data, the ultrasound tongue images. Chapter 2 explains the mathematical background of the spectral coefficients and the vocoding process. In chapter 3 all the studied neural network types and architectures are presented. Chapter 4 describes the hyperparameter optimization process. In chapter 5 the objective and subjective results are shown. And the last chapter is about the conclusions.

Chapter 1

Input data

1.1 Ultrasound imaging

Ultrasound imaging, or sonography, has been used in medical applications since the 1940's. It is used to see internal body structures such as tendons, muscles, joints, blood vessels, and internal organs. It can be useful too for the case of the tongue.

This technology involves the use of ultrasounds, those are ultra high frequency sound waves, typically above 1 MHz. The sound is generated by a transducer made of a piezoelectric (pressure electric) crystal, which converts electricity into mechanical vibrations and vice versa. The crystal is heated to a very high temperature to create a dipolar molecular alignment in which molecules align north to south magnetically. When a voltage is applied to the crystal, the molecules first twist in one direction increasing the crystal's thickness, and then reverse direction decreasing the thickness. This mechanical vibration creates an ultra-high frequency sound wave pulse at the resonant frequency of the crystal, which is determined by the thickness of the crystal. [21]. The transducer acts subsequently as a microphone capturing the echoes generated by the tissue along the path of the emitted pulse. These echoes carry information about the acoustic properties of the tissue along the path. [22].

Ultrasound imaging is based on this *pulse-echo* principle. The later an echo is received, the deeper is the location of the structure giving rise to the echo. The larger the amplitude of the echo received, the larger is the average specific acoustic impedance difference between the structure and the tissue just above. An image is created by repeating this process. [22]. High frequency ultrasound waves have the same transmission properties as audible sound waves, but they have very short wavelengths. Short wavelengths are useful to capture the shapes of small objects and edges and, therefore, increasing spatial resolution. [21].

In the particular case of measuring the tongue, the transducer is placed beneath the chin. The sound wave travels upward through the tongue body until it reaches and reflects back downward from the upper tongue surface. The upper tongue surface interface is typically with the palate bone or airway, both of which have very different densities from the tongue and cause a strong echo. Within the tongue there are also weaker echoes between muscle, fat and connective tissue interfaces.

1.2 Tongue images

The recording environment of the database used as input data is explained in [23]. It basically used a 'Micro' ultrasound system (Articulate Instruments Ltd.) with a 2-4 MHz, 64 element 20mm radius convex ultrasound transducer at 80-100 fps. The resulting images have a spatial resolution of 842x64 pixels. Due to the fact that the vertical resolution is higher than the horizontal, in order to reduce redundancy and dimensionality, the images have been decimated in the vertical axis by a factor of 8. Therefore, the resulting image size is 106x64 pixels. Besides that, considering that the lower part of the images is useless for the prediction, the images have been cut and the final size is 96x64 pixels.

The figure 1.1 shows a sequence of eight subsequent ultrasound tongue images. As it can be seen, these images are composed of speckled areas and edges. The tongue can be detected as a bright and narrow line situated above a darker area. The thickness of the line is irrelevant because the tongue surface is the gradient from white to black at the lower edge. Even though, these images have a lot of noise and are far to be clear. Because of the particularities of ultrasound acoustic signals, the noise is multiplicative (Speckle noise), making it much harder to clean the images with any standard image processing technique. A lot of research about cleaning these type of images has been done, [24], [25], [26]. In the literature some research can be found about the field of tongue tracking

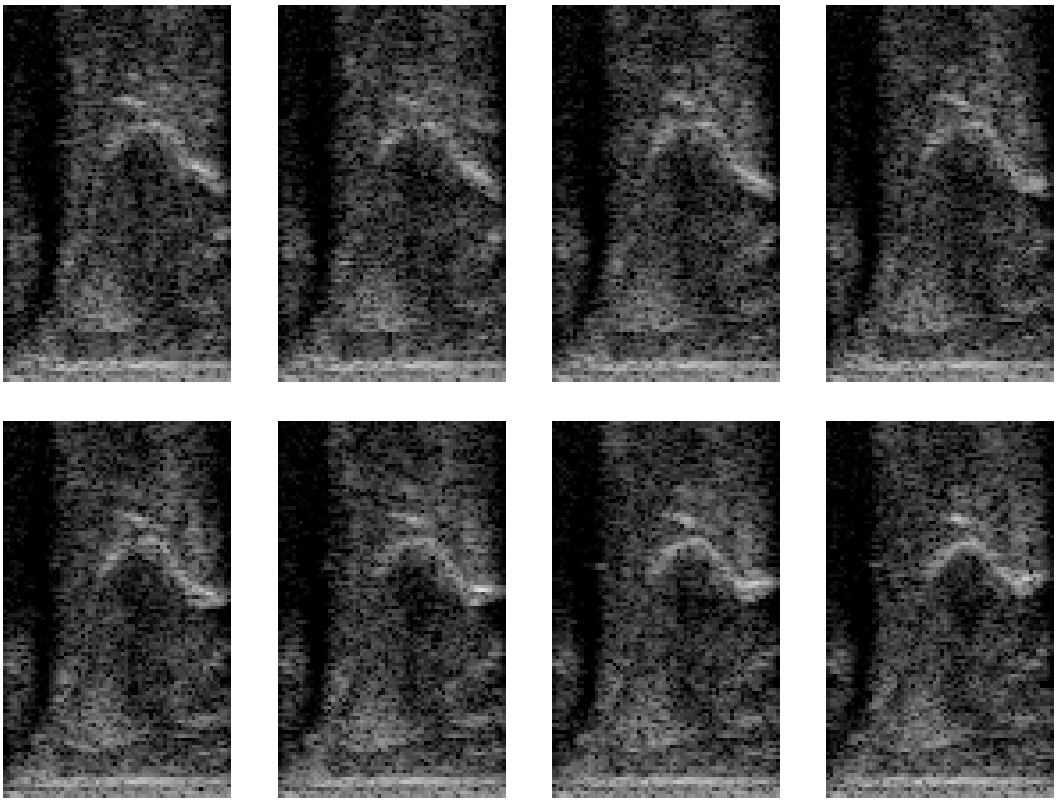


Figure 1.1*Sequence of eight subsequent ultrasound tongue images*

using active contours or other kinds of tracking algorithms [27], [28], [29]. Applying tongue tracking would significantly reduce the number of parameters and increase the neural network performance. Unfortunately there is no fully-automatic algorithm robust enough

to be included in this approach, most of the algorithms require of manual initialization or rectifying. Tongue tracking is actually a hard problem due to the different artifacts that can appear on the images, according to [21] there are five distinguished artifacts.

- Double edges: When the tongue is grooved it has a lower edge at midline, and a higher edge lateral to midline. Sometimes, both edges are visible in the ultrasound image, especially if the transducer is slightly off midline.
- Discontinuities: These can be caused by a slight transducer rotation or a possible asynchrony between the scan rate and the frame rate.
- Objects above the tongue surface: Reflections appearing above the tongue surface are artifacts and should be ignored.
- Inconsistent transducer placement: A serious artifact is created by the erroneous belief that two tongue images are in the same orientation, when they are not.
- Artifacts in swallowing: After swallowing the water-air interface produces a straight white line that is quite bright and should not be confused with the tongue.

Due to the complexity of implementing an image cleaning preprocessing or tongue tracking and its lack of robustness, no preprocessing algorithm has been applied to the input data. Even though an image denoiser based on a deep convolutional autoencoder has been implemented and tested, it is explained in detail in section 3.4.

Looking at the sequence in figure 1.1, we can notice that there is a huge correlation between one frame and the next. This correlation will be explored using the Recurrent Neural Networks.

Chapter 2

Target data

For this thesis, a 'direct synthesis' approach has been chosen. That means that the neural network, fed with the ultrasound images, predicts spectral coefficients which can be used to directly synthesize speech through a vocoder. The coefficients used are Mel-Generalized Cepstrum coefficients (MGC), but converted to Line Spectral Pair (LSP) representation. In the following sections these coefficients and the vocoding process are explained. The Speech Signal Processing Toolkit (SPTK) has been used to compute all the algorithms explained hereunder.

2.1 Mel-Generalized Cepstral coefficients

Cepstral analysis is a widely used tool in speech processing tasks. A cepstrum $c(n)$ of a signal $x(n)$ is defined as the inverse Fourier transform of the logarithmic spectrum.

$$\begin{aligned} X(e^{j\omega}) &= \mathcal{F}\left[x(n)\right] \\ c(n) &= \mathcal{F}^{-1}\left[\log X(e^{j\omega})\right] \end{aligned} \tag{2.1}$$

The case of Mel-Cepstral coefficients represents the spectral envelope with coefficients spaced from one another in the Mel scale, a perceptual scale of pitches judged by listeners to be equal in distance from one another. Therefore, the spectrum is 'mapped' with a Mel-scale triangular overlapping filter bank before applying the natural logarithm, figure 2.1.

In 1994, Keiichi Tokuda and his colleagues proposed a unified approach to the cep-

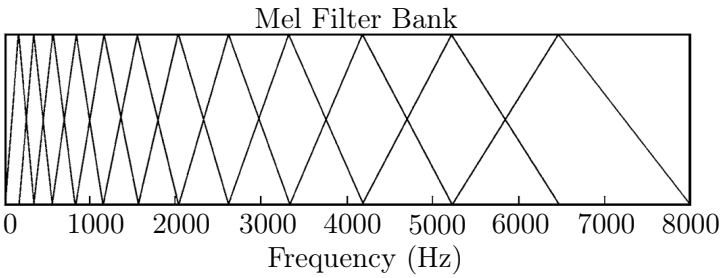


Figure 2.1*Example of Mel-frequency filter bank*

	α=0	$\lvert\alpha\rvert$ <1
γ=-1	all-pole	warped all-pole
γ=0	cepstral	Mel-cepstral
γ=1	all-zero	warped all-zero
-1 < γ < 1	generalized cepstral	Mel-generalized cepstral

Table 2.1: *Form of the coefficients depending on the parameters α and γ*

stral method and the linear prediction method [30], the Mel-generalized cepstral analysis (MGC). By varying two parameters, α and γ, it is possible to choose between the different available features. These coefficients, instead of the natural logarithm, use the generalized logarithmic function $s_\gamma(\omega)$, defined as:

$$s_\gamma(\omega) = \begin{cases} (\omega^\gamma - 1)/\gamma, & \text{for } 0 \leq |\gamma| \leq 1 \\ \log\omega, & \text{for } \gamma = 0 \end{cases} \tag{2.2}$$

The Mel-generalized cepstral coefficients (MGC) $c_{\alpha,\gamma}(m)$ are defined as the inverse Fourier transform of the generalized logarithmic spectrum calculated on a warped frequency scale $\beta_\alpha(\omega)$.

$$s_\gamma\left(X(e^{j\omega})\right) = \sum_{m=-\infty}^{\infty} c_{\alpha,\gamma}(m) e^{-j\beta_\alpha(\omega)m} \tag{2.3}$$

The warped frequency scale is defined as the phase response of an all-pass system, $\Psi_\alpha(z)$. It gives an approximation of the Mel-scale choosing the appropriate value of the parameter α-

$$\Psi_\alpha(z) = \left.\frac{z^{-1} - \alpha}{1 - \alpha z^{-1}}\right|_{z=e^{j\omega}} = e^{-j\beta_\alpha(\omega)} \tag{2.4}$$

where

$$\beta_\alpha(\omega) = \tan^{-1}\frac{(1-\alpha^2)\sin\omega}{(1+\alpha^2)\cos\omega - 2\alpha} \tag{2.5}$$

In [30] it is admitted that the speech spectrum $H\left(e^{j\omega}\right)$ can be modeled by M + 1 Mel-generalized cepstral coefficients as follows:

$$H(z) = s_\gamma^{-1}\left(\sum_{m=0}^{M} c_{\alpha,\gamma}(m)\Psi_\alpha^m(z)\right) \tag{2.6}$$

Varying the parameters α and γ the model spectrum takes different forms as it can be seen in the table 2.1

In this particular case, an order of $M = 24$, $\alpha = 0.42$, $\gamma = -0.33$ have been chosen.

2.2 Line Spectral Pair

The MGC coefficients have been converted to Line Spectral Pair (LSP) representation, because these have better interpolation properties and makes the coefficients more suitable to be predicted by a Deep Neural Network.

LSPs were originally minded for transforming LPC coefficients, but because, as explained in the previous section, MGCs are a variant of LPCs characterized by the parameters α and γ, Line Spectral Pair can be used to transform MGCs.

As explained in [31], two $(M+1)$th ordered polynomials $P(z)$ and $Q(z)$ are defined.

$$P(z) = H(z) - z^{-(M+1)}H(z^{-1}) \tag{2.7}$$

$$Q(z) = H(z) + z^{-(M+1)}H(z^{-1}) \tag{2.8}$$

$$H(z) = \frac{P(z) + Q(z)}{2} \tag{2.9}$$

Being $H(z)$ the z-transform of the MGC coefficients.

The LSP parameters are expressed as the zeroes (or roots) of $P(z)$ and $Q(z)$. If we denote the set of complex roots as θ_k, then the LSPs in radians ω_k are determined from 2.7 and 2.8:

$$\omega_k = \tan^{-1}\left(\frac{\mathbb{R}e(\theta_k)}{\mathbb{I}m(\theta_k)}\right) \tag{2.10}$$

2.3 Vocoder

For the synthesis phase, we assumed that the pitch cannot be estimated. However, a recent study by Hungarian researchers, including the supervisor of this thesis [32], has proven that F0 can be predicted from the ultrasound images using DNNs with convincing results. In this work, for simplicity reasons and because the main objective is to estimate the spectral parameters, the original F0 is used. A pulse excitation function is generated according to the pitch period, and when the pitch is not existent (marked as 0) a white Gaussian noise excitation is generated.

The MGC-LSP coefficients are firstly transformed to simple MGC coefficients, those coefficients are normalized in order to avoid any kind of issue. From the MGC coefficients, is derived a Mel-Generalized Log Spectral Approximation digital filter (MGLSA), this is used to filter the excitation signal previously generated.

The block diagram of the vocoder is represented in 2.2, it must be noticed that the gain parameter is the first LSP coefficient ω_0.

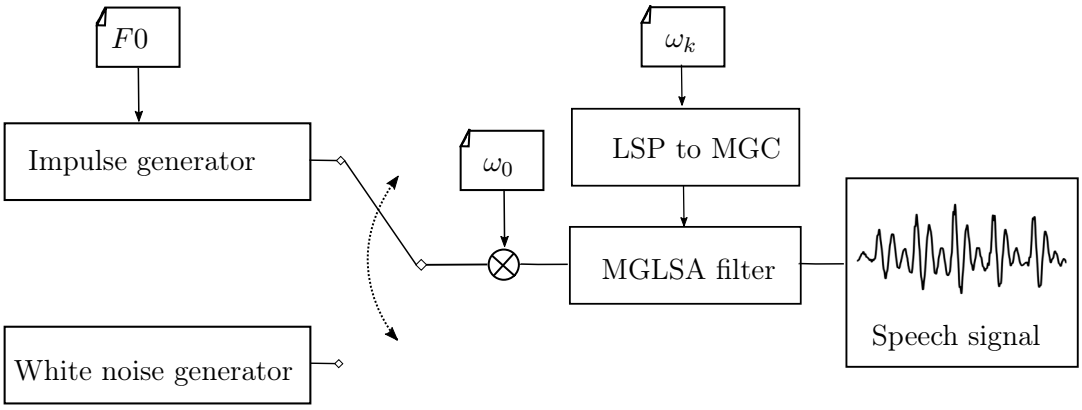


Figure 2.2 *Block diagram of the vocoder*

Chapter 3

Deep Learning Architectures

Throughout this thesis, several different deep learning architectures have been tested. This chapter shows the basic concepts about how they are structured, the types of layers they have and how they work.

3.1 Deep Feed-forward Neural Network

Deep Feed-forward Neural Networks, or Fully connected Neural Networks, or Multilayer Perceptrons are the most classic deep learning models. "The goal of a feed-forward network is to approximate some function f, it defines a mapping $c = f(x;\theta)$ and learns the value of the parameters θ that result in the best function approximation"[33]. In our particular case of *SSI* the network must be able to find the best function that maps the ultrasound images x to the MGC-LSP coefficients c.

Those networks are composed of basic units called perceptrons, or neurons. The structure of a perceptron is represented in figure 3.1. A perceptron is nothing else than a linear classifier, each one has several inputs x_n escalated by some weights ω_n. All the weighted inputs are added and the result is mapped through an activation function g. Different functions are used for this purpose, the most common are *tanh*, *sigmoid*, *softsign*, *identity* or *ReLu*. In our experiments, the Rectified Linear Unit (*ReLu*) function has been applied in almost all the layers, the shape of this function is shown in figure 3.2.

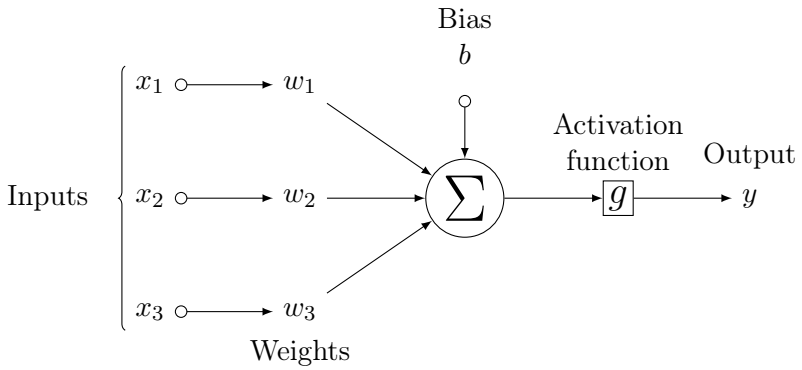


Figure 3.1 *Structure of a basic perceptron unit*

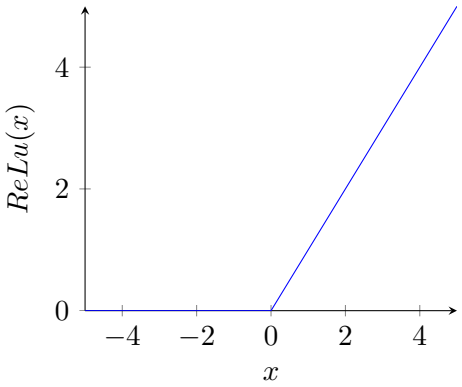
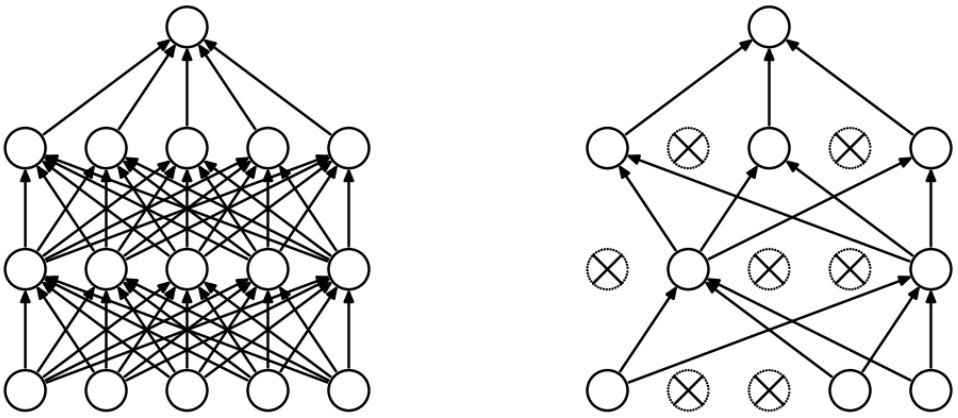


Figure 3.2 *Plot of the Rectified Linear Unit function*



(a) *Standard Neural Network* **(b)** *After applying dropout*

Figure 3.3 *Representation of Dropout regularization. Figures extracted from [35]*

In a Fully Connected Neural Network all neurons are grouped in layers. The input layer has the input data as inputs, and the output layer has the predictions as outputs, between these two layers there are one or more hidden layers. All the perceptrons are connected in a way that every output of each neuron in one layer is used as input for each neuron in the consecutive layer.

As mentioned earlier, the goal of neural networks is to learn the value of the set of parameters θ that result in the best function approximation. In these kind of networks, the set θ consists on all the weights $w_k^{i,j}$ and biases $b^{i,j}$ from every i neuron in every j layer. These parameters are computed in the training process using the *back-propagation algorithm* [34].

One common problem in Deep Learning is *overfitting*. This word references the scene when the neural network has fitted too much to the training dataset and, when it is evaluated with a different dataset, the results are worse than expected. One method used in this thesis to prevent *overfitting* is called *Dropout* [35]. This method consists in 'dropping out units' from hidden and visible layers in a random way. Figure 3.3 shows a visual representation of what dropout regularization does.

The architecture of the neural network for this SSI application consists of one input

layer with all the images reshaped as a one-dimensional array, four hidden layers with 500, 1000, 1000 and 1000 neurons respectively and the output layer with 25 outputs. All those hyperparameters have been optimized as it is explained in section 4.1. A diagram of this network is shown in figure 3.4. Even though it is not shown in the figure, dropout has been applied after each hidden layer, the probabilities are $p = 0.1$ in each layer.

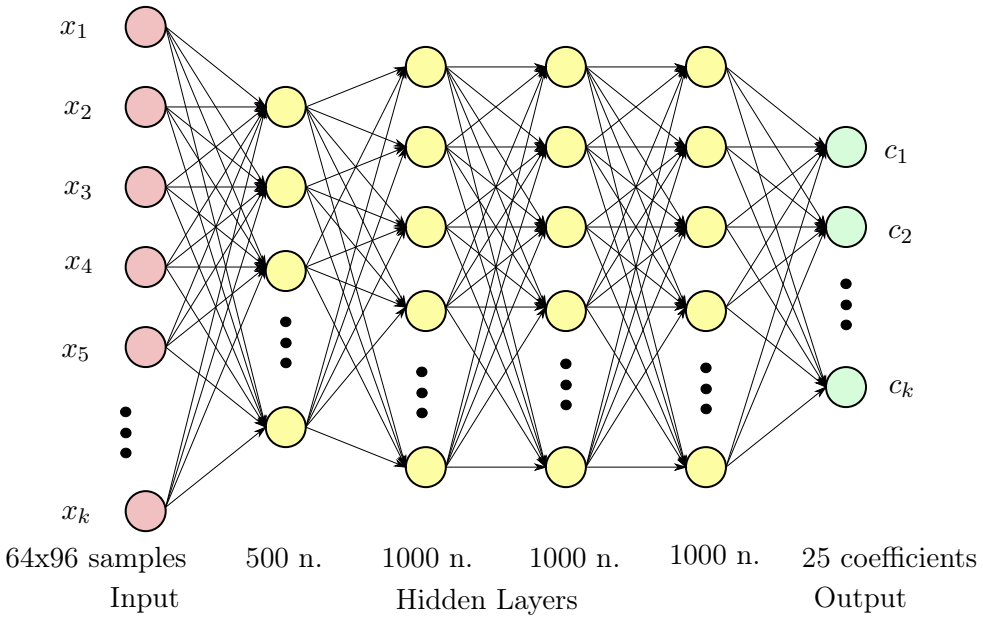


Figure 3.4*Diagram of the deep feed-forward neural network*

3.2 Convolutional Neural Network

Convolutional Neural Networks are a kind of neural networks for processing data that have a known grid-like topology. "Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers" [33]. Those networks have become very successful in analyzing visual imagery applications. They have a two or three dimensional input data followed by a combination of convolutional layers and pooling layers; after that, these networks have some fully-connected layers at the end.

The main part of a Convolutional Neural Network are the convolutional layers. The idea behind these layers is to apply a filter-bank to the data. Each layer is composed on k three-dimensional kernels (or filters) with an mxn pre-defined size and a depth d, the value of the depth d has to be the same as the depth of the input. The computation is based on multi-dimensional convolutions between the input image and all the k kernels in the layer.

One important factor is how does the network lead with the convolution in the borders. One option is using padding (e.g. zero padding, mirroring), this way the resulting image would have the same size as the input. The other option is the stride, this means that the borders are not computed, so the resulting image will be slightly smaller. Therefore, having an input data of dimensions $a \times b \times c$ and k kernels with size $m \times n \times c$, the total size of

the output data would be $a \times b \times k$ in case of padding or $(a-2) \times (b-2) \times k$ using stride.

The parameters that the neural network would have to learn in one layer are the weights of each of the k kernels, a total of $k \times m \times n \times c$ parameters. This number is much less than the number of parameters that would be needed if the same image was used as input of a Fully Connected Neural Network.

Between convolutional layers, pooling layers are interleaved. These are used to reduce the number of parameters and simplify the network. It is also referred to as a downsampling layer. In this category, there are also several layer options, with maxpooling being the most popular. MaxPool consists on taking only the maximum value from a fixed-length window, figure 3.5 shows a graphical example.

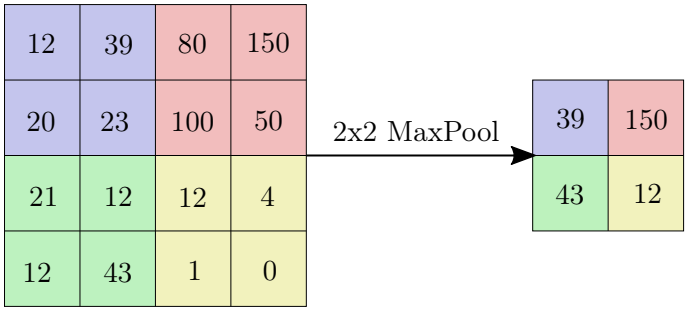


Figure 3.5 *Example of a 2x2 MaxPool operation*

The architecture explored in this thesis consists in two convolutional layers, two pooling layers and, after flattening the data, a 4-layer fully connected end (3 hidden layers and the output layer). The CNN is represented in figure 3.6. All the hyperparameters of this network have been optimized as will be explained in section 4.2.

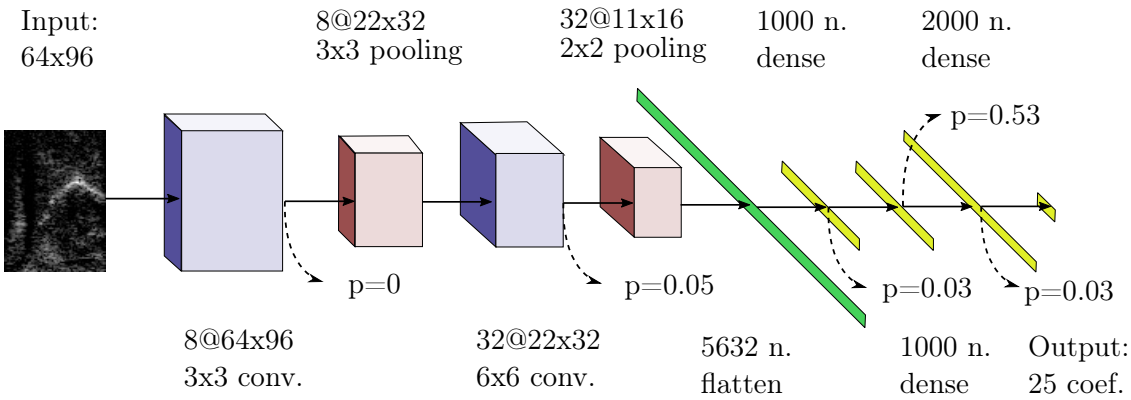


Figure 3.6 *Diagram of the convolutional neural network. Blue layers represent convolutional layers, red layers are pooling layers, the green layer represents a flattening layer and the yellow ones are dense layers. The discontinuous arrows mean that dropout has been applied on that layer, the parameter p shows the dropout probability. The dimensions of the figure are not totally proportional to the real network, but are meant to be indicative.*

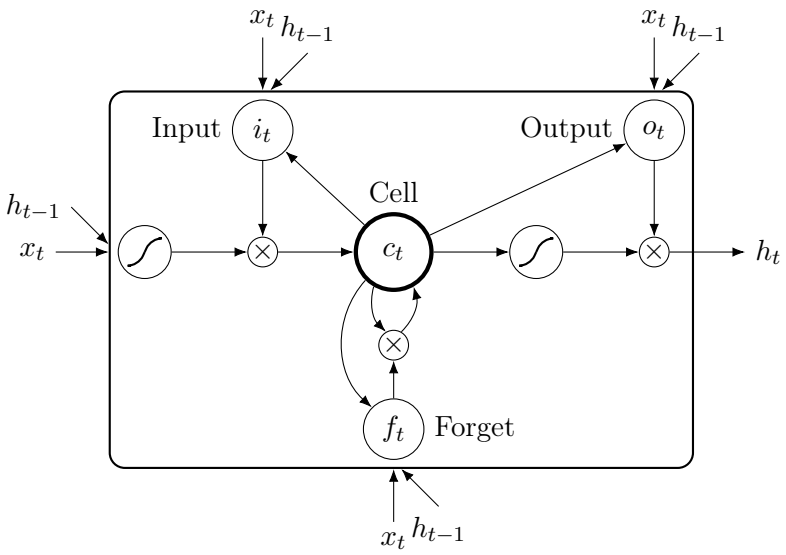


Figure 3.7*Structure of an LSTM unit*

3.3 CNN-LSTM Recurrent Neural Network

Long Short-Therm Memory units (LSTMs) are a kind of units for Recurrent Neural Networks. "A recurrent neural network is a neural network that is specialized for processing a sequence of values $x^1, ..., x^\tau$" [33]. Recurrent Neural Networks allow the networks to have temporal information. These are useful for many different applications, like image/video captioning, word prediction, translation or image processing. They can be appropriate for our particular case of SSI as well, making use of the correlation between image samples can increase the accuracy of our system.

LSTMs where firstly introduced by Sepp Hochreiter and Jurgen Schmidhuber in 1997 [36]. They introduced an efficient recurrent architecture designed to overcome some back-flow problems existent in previous RNN approaches. The structure of an LSTM unit is represented in figure 3.7. An LSTM hidden layer consists of recurrently connected sub-nets, called memory blocks. Each block contains a set of internal units or cells whose activation is controlled by three multiplicative gates: the input gate, forget gate, and output gate.

The effect of the gates is to allow the cells to store and access information over long periods of time. When the input gate remains closed (its activation is close to zero), the activation of the cell will not be overwritten by the new inputs arriving in the network. Similarly, the cell activation is only available to the rest of the network when the output gate is open. The forget gate was designed later [37] to learn to reset memory blocks once their contents are out of date and hence useless.

Our particular recurrent neural network architecture is shown in figure 3.8. It consists of three distinguished sections: a CNN, an LSTM layer and a fully connected end. The CNN part has two convolutional layers and two pooling layers. It is followed by an LSTM layer in order to incorporate time information. After this, a fully connected feedforward network with three hidden layers and the output is added. The input data is passed to the network in batches of 32 frames, this number was chosen as a trade-off between the

recurrent information imported to the network and its computational complexity which affects the training time.

As will be explained in detail in section 4, only the hyperparameters of the LSTM and fully connected sections have been optimized. The CNN section has the same architecture as in figure 3.6.

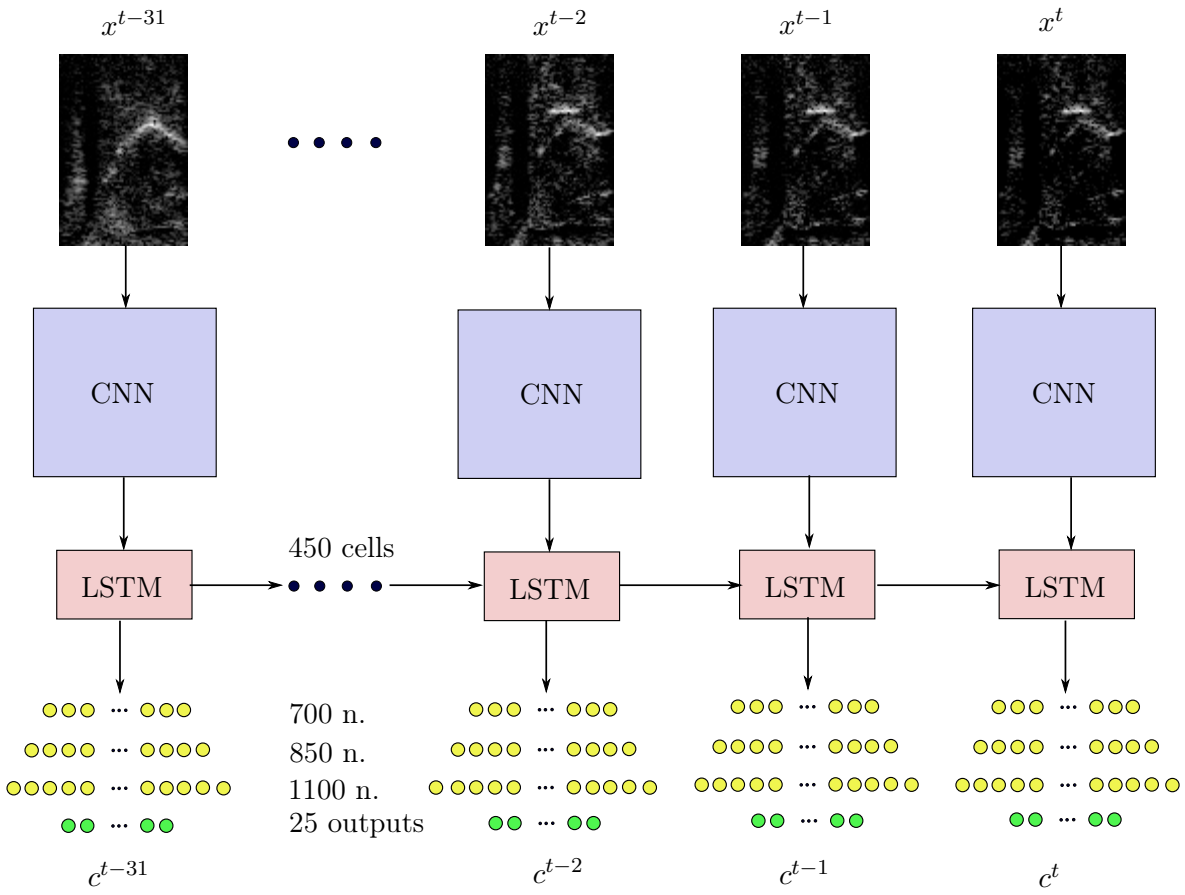


Figure 3.8 *Diagram of the CNN-LSTM Neural Network*

3.3.1 Bidirectional CNN-LSTM Recurrent Neural Network

An improvement to the CNN-LSTM network has been explored by adding bidirectional information. That means that not only the past images are used for the prediction but also the future images are used. This has been achieved using two LSTM layers in parallel, concatenating their outputs and then using a fully-connected end as the previous network. Figure 3.9 shows the architecture of this network, the CNN sections are the same as before but all the hyperparameters relating to the LSTM layers and the Dense layers have been to be optimized again. The main problem about this network is that it requires a higher delay. This network uses batches of 32 frames, having a sampling frequency of 81.5 Hz, it requires $\frac{32}{F_s} = 392ms$ more than the others. Therefore, this may not be an appropriate architecture for real time applications, but in other scenes where low delay is not required it might have a better performance.

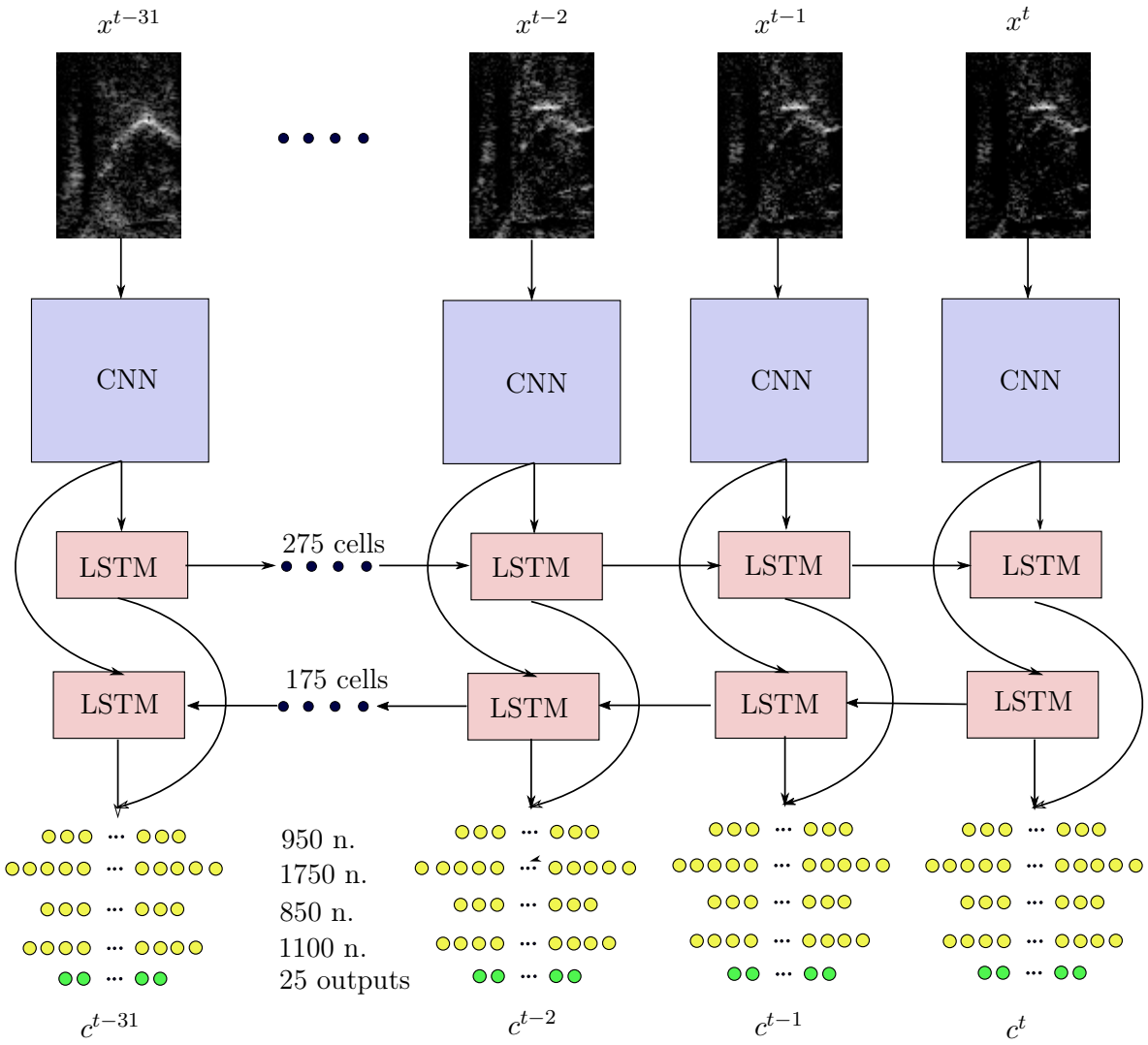


Figure 3.9 *Diagram of the bidirectional CNN-LSTM Neural Network*

3.4 Denoising Convolutional Autoencoder

An autoencoder is a neural network trained with objective of copying its input to its output. The network can be split in two parts: encoding and decoding. The encoding produces a code $h = f(x)$ with much fewer coefficients than the inputs. The decoder produces a reconstruction $\hat{x} = g(h)$. In these networks, the input parameters are the same as the outputs, so they can be classified as an unsupervised learning algorithm.

A convolutional autoencoder uses a convolutional neural network for the encoding and decoding parts. The encoding is made of convolutional and pooling layers, while the decoding part is the symmetrical mirror of the encoding, made of convolutional layers and up-sampling layers instead of pooling.

Autoencoders are often used for dimensionality reduction, in order to use the code h as input for another neural network. In this SSI application, the autoencoder is using a denoising algorithm, so it is not the code h what will be extracted but the reconstruction image $\hat{x}$. Because of the fact that the accuracy of the autoencoder is obviously not perfect,

the reconstruction image $\hat{x}$ is something like a simpler version of the original x. This means that some of the noises and artifacts are lost in the denoising process but the tongue shape, the important part of the images, is preserved. This way, we avoid 'distracting' the neural network with useless artifacts.

Figure 3.10 shows the diagram of the autoencoder implemented and a comparison between the input image and the reconstruction. The encoder consists of two convolutional layers (colored in blue), both followed by a pooling layer (colored in red). The decoder contains two convolutional layers with the same number of kernels and sizes as the encoder, both followed by up-sampling layers (colored in green) with the same sizes as the poolings in the input. Just before the output, another convolution layer is added in purpose to adapt the image dimensions. On the bottom are shown the input and output images decimated by 8, due to the fact that the input of the posterior neural network will have these dimensions.

As shown in the figure, the input images are not decimated by 8 as in the previous networks, this decision was made in order to make a special use of the autoencoder for the image resizing. In order to prevent possible aliasing issues, doing the decimation at the reconstructing image is way more convenient than doing it in the original one. Considering the image dimensions, bigger poolings are done in the vertical axis than in the horizontal, this way a higher 'low-pass effect' in the vertical axis appears in the resulting image.

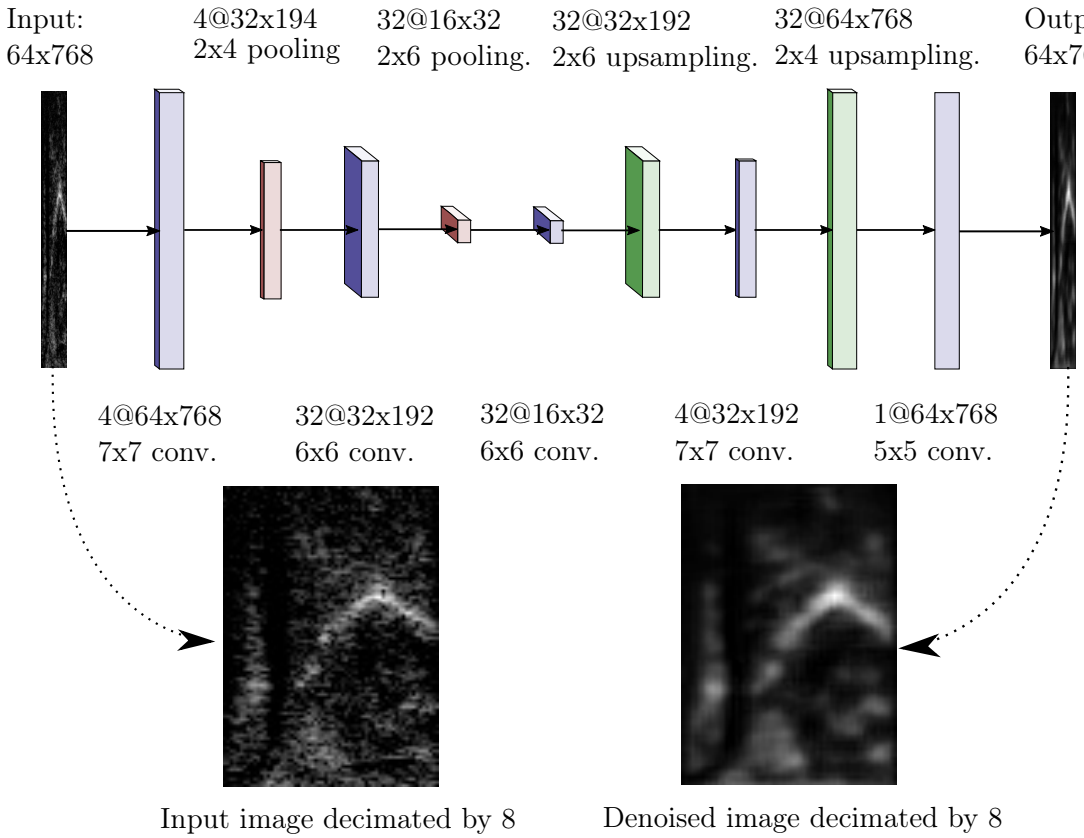


Figure 3.10 *Diagram of the denoising convolutional deep autoencoder*

Chapter 4

Hyperparameter optimization

Hyperparameter optimization is a must do task in any deep learning project. It consists of finding the values for all the hyperparameters, parameters that have to be fixed before the training process, in the way that maximizes the accuracy of the network.

The classic method to do it is by grid search, which is simply exhausting searching along the entire hyperparameter space. This method is severely influenced by the curse of dimensionality as the number of hyperparameters increase. The random search method is an alternative of grid search to explore the randomly sampled hyperparameters in the hyperparameter space. It can outperform Grid search, when only a small number of hyperparameters affects the final performance, but it is also time-consuming and inefficient.

Instead of these, Bayesian Optimization has been used. "Bayesian optimization constructs a probabilistic surrogate model to define the distribution over the unknown black box function, and a proxy optimization is performed to seek the next location to evaluate where the posterior distribution is developed based on conditioning on the previous evaluations. Acquisition function is applied to the posterior mean and variance to express the trade off between exploration and exploitation" [38]. Bayesian Optimization has been implemented using the *hyperas* module in *keras* .

This chapter shows how hyperparameter optimization has been conducted for all the different architectures presented in chapter 3.

4.1 Deep Feed-forward Neural Network

The first attempt to train a neural network has been the most basic, the fully-connected Feed-forward Neural Network. It is also the most easy to train and the one with the least hyperparameters. Table 4.1 shows all the hyperparameters validated and their optimal values. It must be noticed that some of the hyperparameters have continuous values and others are discretized. For example, the dropout probabilities can get any value between 0 and 1, but the number of neurons is restricted in a discrete space where the options are limited (e.g. 250, 500, 750, 1000, etc.) .

In this experiment, a total of 895 evaluations have been executed. Figure 4.1 shows how the validation accuracy has varied during the evaluations. To make this plot, all the

Hyperparameter	Optimal value
Number of hidden layers	4
Number of Neurons in the first layer	500
Number of Neurons in the second layer	1000
Number of Neurons in the third layer	1000
Number of Neurons in the fourth layer	1000
Dropout probability after the first hidden layer	0.09
Dropout probability after the second hidden layer	0.07
Dropout probability after the third hidden layer	0.15
Dropout probability after the fourth hidden layer	0.07
Optimizer	adamax
Batch size	64
Validation accuracy	0.28

Table 4.1: *Hyperparameters from the fully-connected Neural Network*

accuracy values have been sorted in descending order. The horizontal axis shows the index of the evaluation ordered from best to worse and the vertical shows its validation accuracy. Looking at the plot, the Feed forward neural network is colored in blue and we can see that there are only few evaluations that have reached good results (above 0.27).

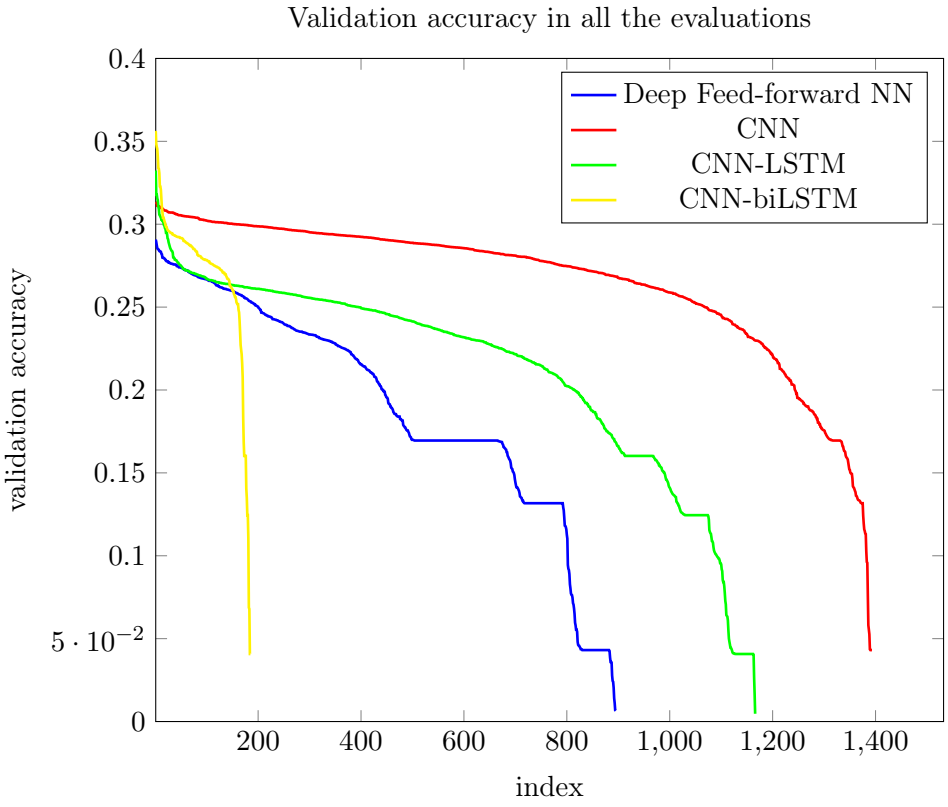


Figure 4.1 *Plot of the validation accuracy results in the the different neural networks for all the evaluations sorted from best to worse*

4.2 Convolutional Neural Network

The convolutional neural network has been the second type of deep learning network to explore. The structure of the network is shown in figure 3.6 in the previous chapter, it combines convolutional and pooling layers and has a fully-connected ending.

This network needs way more hyperparameters to be validated, so a higher number of evaluations is required, a total of 1394 evaluations have been executed. Despite this huge number, the whole computing time had not been very long (less than two days), because convolutional layers have few parameters to train and the training time is quite short. Table 4.2 shows all the hyperparameters evaluated. As can be seen in the table all the features of the dense layers have been to be evaluated again.

Hyperparameter	Optimal value
Number of convolutional layers	2
Number of kernels in the first conv. layer	8
Number of kernels in the second conv. layer	32
Size of the kernels in the first conv. layer	(3, 3)
Size of the kernels in the second conv. layer	(6, 6)
Dropout after the first conv. layer	0
Dropout after the second conv. layer	0.05
Pooling window size in the first pooling layer	(3, 3)
Pooling window size in the second pooling layer	(2, 2)
Number of hidden dense layers	3
Number of Neurons in the first dense layer	1000
Number of Neurons in the second dense layer	1000
Number of Neurons in the third dense layer	2000
Dropout probability after the first hidden layer	0.03
Dropout probability after the second hidden layer	0.55
Dropout probability after the third hidden layer	0.03
Optimizer	adam
Batch size	64
Validation accuracy	0.31

Table 4.2: *Hyperparameters from the Convolutional Neural Network*

The red plot in Figure 4.1 shows the validation accuracy variations in the same way as in the previous case, sorting all the evaluations in descending order. Looking at the figure, we can notice that the results are way better than in the fully-connected network. The highest point is 0.31 while in the fully-connected was 0.28 and in general there are many evaluations with relatively high accuracies (above 0.3). This may be caused because changing some parameters as the number of kernels or the kernel sizes does not affect too much the validation accuracy of the whole network. In this case, it could be said that there is a lower intrinsic dimensionality in the hyperparameter space.

4.3 CNN-LSTM Recurrent Neural Network

The Recurrent Neural Network explored is a combination of a CNN section followed by an LSTM layer, being the outputs of the LSTM connected to a set of fully-connected layers at the end. The diagram of the network is represented in figure 3.8.

Hyperparameter	Optimal value
Number of neurons in the LSTM layer	450
Activation function in the LSTM layer	tanh
Number of hidden dense layers	3
Number of Neurons in the first dense layer	700
Number of Neurons in the second dense layer	850
Number of Neurons in the third dense layer	1100
Dropout probability after the first hidden layer	0.05
Dropout probability after the second hidden layer	0.51
Dropout probability after the third hidden layer	0.45
Optimizer	adamax
Batch size	2
Validation accuracy	0.33

Table 4.3: *Hyperparameters from the CNN-LSTM Neural Network*

The CNN section is the same one as the presented in section 4.2, with the same hyperparameters as its optimal result. So the hyperparameters evaluated have only been, as shown in table 4.3, the ones referring to the LSTM layer and to the three hidden dense layers.

In this experiment, 1167 evaluations have been executed. The LSTM layer has a substantial number of parameters to train, so this network needs more epochs to converge than the others. Therefore, the hyperparameter optimization process has required way more time than the others to have enough evaluations (between three and four days of continuous training). The green plot in figure 4.1, as in the previous sections, shows the accuracy in all the evaluations sorted from best to worse. In this case, the best result has a higher accuracy than the CNN network, that means that adding temporal information can slightly increase the performance of the SSI system. Looking at figure 4.1, it seems like this network's performance is worse than the CNN's because most of the evaluations in the mid-range of the plot have a worse accuracy. However, this conclusion is deceitful because in this set of hyperparameters there are not many which contribution to the accuracy could be considered redundant, while in the CNN some hyperparameters from the convolutional layers could be considered like that. Therefore, the only important thing to consider when comparing the accuracy plots are the best results.

4.3.1 Bidirectional CNN-LSTM Recurrent Neural Network

The bidirectional CNN-LSTM Recurrent Neural Network has required a hyperparameter optimization similar than the simple CNN-LSTM. In this case, considering that it has two LSTM layers, the number of parameters to train is even bigger, so the training is slower.

Due to the high computational time, the number of evaluations is smaller (only 200). The set of hyperparameters evaluated, as shown in table 4.4, is almost the same as before, the only change is that now there are two LSTM layers and the number of neurons in both must be evaluated.

Hyperparameter	Optimal value
Number of neurons in the left LSTM layer	275
Number of neurons in the right LSTM layer	175
Activation function in the LSTM layer	relu
Number of hidden dense layers	4
Number of Neurons in the first dense layer	950
Number of Neurons in the second dense layer	1750
Number of Neurons in the third dense layer	850
Number of Neurons in the fourth dense layer	1100
Dropout probability after the first hidden layer	0.19
Dropout probability after the second hidden layer	0.17
Dropout probability after the third hidden layer	0
Dropout probability after the fourth hidden layer	0
Optimizer	rmsprop
Batch size	1
Validation accuracy	0.34

Table 4.4: *Hyperparameters from the CNN-biLSTM Neural Network*

The accuracy in all the evaluations is represented, as in the other cases, in figure 4.1 colored in yellow.

4.4 Denoising Convolutional Autoencoder

The autoencoder has been optimized as well as the other network but, being a network for a different purpose, the hyperparameter optimization has been done in a special way.

This network, as can be seen in figure 3.10 only has convolutional, pooling and up-sampling layers. The number of hyperparameters is not large because the decoding section is the mirror of the encoding, so all the dimensions must be the same. The hyperparameters chosen are shown in table 4.5.

Hyperparameter	Optimal value
Number of convolutional layers	2
Number of kernels in the first conv. layer	4
Number of kernels in the second conv. layer	32
Size of the kernels in the first conv. layer	(7, 7)
Size of the kernels in the second conv. layer	(6, 6)
Size of the kernels in the last conv. layer	(5, 5)
Pooling window sizes in both layers	(2, 4) and (2,6)
Validation accuracy	0.18

Table 4.5: *Hyperparameters from the Denoising Convolutional Autoencoder*

Due to the fact that there is no hyperparameter that affects directly to the network performance and it seems to converge appropriately with all the combinations, the validation accuracy does not almost vary, as shown in figure 4.2, it remains in 0.18 in all the evaluations. Because of the low number of parameters and the futility of this hyperparameter optimization, it has not been necessary to execute a big number of evaluations, with only 37 has been enough. The architecture chosen has been the one that has a slightly better performance, but the difference between this and the worst is insignificant.

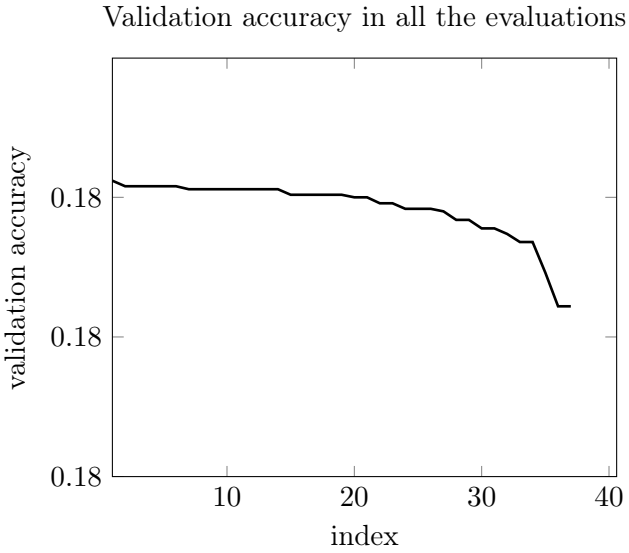


Figure 4.2 *Plot of the validation accuracy results in the autoencoder for all the evaluations sorted from best to worse*

4.5 Optimization using the denoised images

After using the autoencoder, the images after the denoising process have different properties than their original versions, so the neural networks used to decode them must be different. Therefore, the optimal architecture of each neural network must be found again, another hyperparameter optimization is required in all the cases. Tables 4.6, 4.7, 4.8, 4.9 show the sets of hyperparameters and their optimal values.

Figure 4.3 shows the accuracy in all the evaluations in the same way as figure 4.1. Looking at the figure, all the the plots have a very similar shape compared to the non-denoised case, but they present slighlty better accuracy results.

Based on these hyperparameter optimizations, the potentially best systems are the CNN-LSTM and CNN-biLSTM, both using the denoising autoencoder.

Hyperparameter	Optimal value
Number of hidden layers	4
Number of Neurons in the first layer	2000
Number of Neurons in the second layer	1000
Number of Neurons in the third layer	1000
Number of Neurons in the fourth layer	2000
Dropout probability after the first hidden layer	0.15
Dropout probability after the second hidden layer	0.35
Dropout probability after the third hidden layer	0.14
Dropout probability after the fourth hidden layer	0.39
Optimizer	adamax
Batch size	64
Validation accuracy	0.30

Table 4.6: *Hyperparameters from the fully-connected Neural Network using the denoising autoencoder*

Hyperparameter	Optimal value
Number of convolutional layers	2
Number of kernels in the first conv. layer	16
Number of kernels in the second conv. layer	16
Size of the kernels in the first conv. layer	(4, 4)
Size of the kernels in the second conv. layer	(5, 5)
Dropout after the first conv. layer	0.05
Dropout after the second conv. layer	0.01
Pooling window size in the first pooling layer	(4, 4)
Pooling window size in the second pooling layer	(2, 2)
Number of hidden dense layers	3
Number of Neurons in the first dense layer	1125
Number of Neurons in the second dense layer	1500
Number of Neurons in the third dense layer	875
Dropout probability after the first hidden layer	0.07
Dropout probability after the second hidden layer	0.03
Dropout probability after the third hidden layer	0.07
Optimizer	adam
Batch size	64
Validation accuracy	0.33

Table 4.7: *Hyperparameters from the Convolutional Neural Network using the denoising autoencoder*

Hyperparameter	Optimal value
Number of neurons in the LSTM layer	450
Activation function in the LSTM layer	tanh
Number of hidden dense layers	3
Number of Neurons in the first dense layer	900
Number of Neurons in the second dense layer	900
Number of Neurons in the third dense layer	1100
Dropout probability after the first hidden layer	0.04
Dropout probability after the second hidden layer	0.02
Dropout probability after the third hidden layer	0.18
Optimizer	rmsprop
Batch size	1
Validation accuracy	0.35

Table 4.8: *Hyperparameters from the CNN-LSTM Neural Network using the denoising autoencoder*

Hyperparameter	Optimal value
Number of neurons in the left LSTM layer	275
Number of neurons in the right LSTM layer	225
Activation function in the LSTM layers	relu
Number of hidden dense layers	3
Number of Neurons in the first dense layer	1250
Number of Neurons in the second dense layer	1025
Number of Neurons in the third dense layer	1100
Dropout probability after the first hidden layer	0.35
Dropout probability after the second hidden layer	0.06
Dropout probability after the third hidden layer	0.05
Optimizer	adam
Batch size	2
Validation accuracy	0.36

Table 4.9: *Hyperparameters from the CNN-biLSTM Neural Network using the denoising autoencoder*

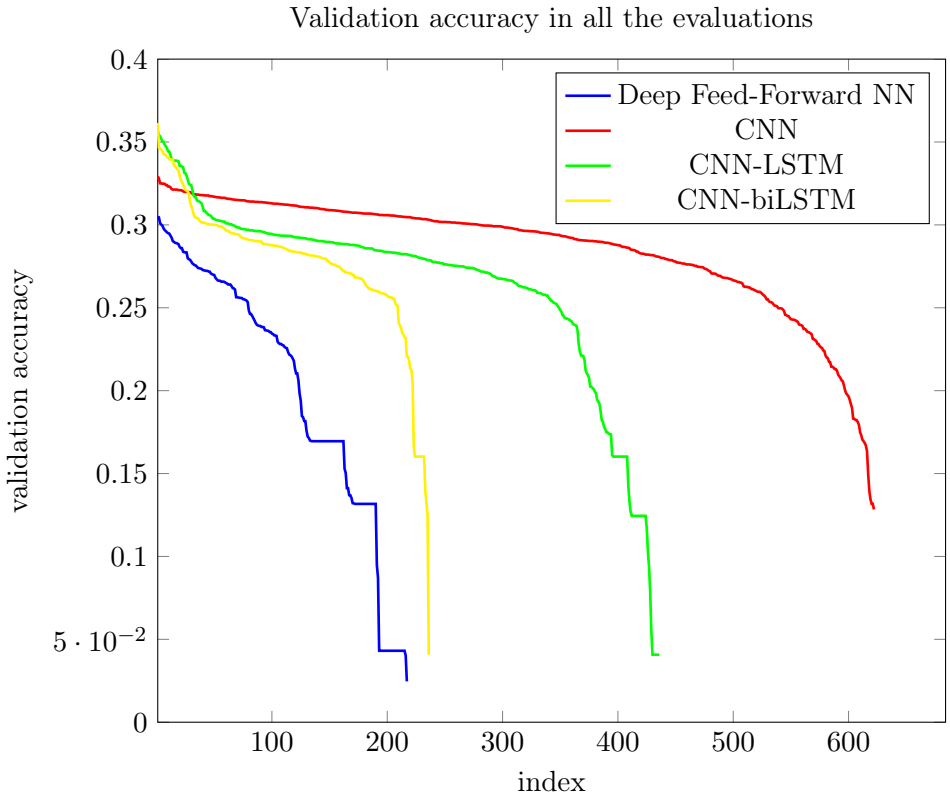


Figure 4.3 *Plot of the validation accuracy results in the the different neural networks using the denoised images for all the evaluations sorted from best to worse*

Chapter 5

Results

This chapter shows the test results in the different Neural Network architectures previously presented. After the hyperparameter optimization process, the hypothesis would be that, using a different test database, the best results would appear in those networks that have presented better validation accuracy metrics, which have been the CNN-LSTM and its bidirectional version, both using the denoising autoencoder. This chapter shows objective and subjective experimental results that will help to decide which could be the best choice for a real *SSI* system.

5.1 Objective measurements

Firstly, an objective test has been executed, the metrics of this test are the loss and the accuracy after predicting the spectral coefficients in the test database. The loss is the Mean Square Error (MSE) between the target coefficients and the predicted ones, while the accuracy is the precentage of well predicted coefficients. It must be noticed that these results are calculated with the spectral parameters error, it would make no sense to compare the speech signals in time domain because no phase information has been preserved during the encoding. Table 5.1 shows the loss (mean square error) and accuracy for all the Neural Networks.

The test database consists of nine different audios from a female Hungarian speaker. The environment used to record these audios is the same as in the train/validation database.

Neural Network type	loss (MSE)	accuracy
Fully-Connected Feed-Forward NN	0.017	0.25
CNN	0.015	0.28
CNN-LSTM	0.013	0.31
CNN-biLSTM	0.013	0.33
Fully-Connected Feed-Forward NN using denoising autoencoder	0.016	0.28
CNN using denoising autoencoder	0.015	0.28
CNN-LSTM using denoising autoencoder	0.015	0.32
CNN-biLSTM using denoising autoencoder	0.013	0.34

Table 5.1: *Objective test results for all the Neural Network types*

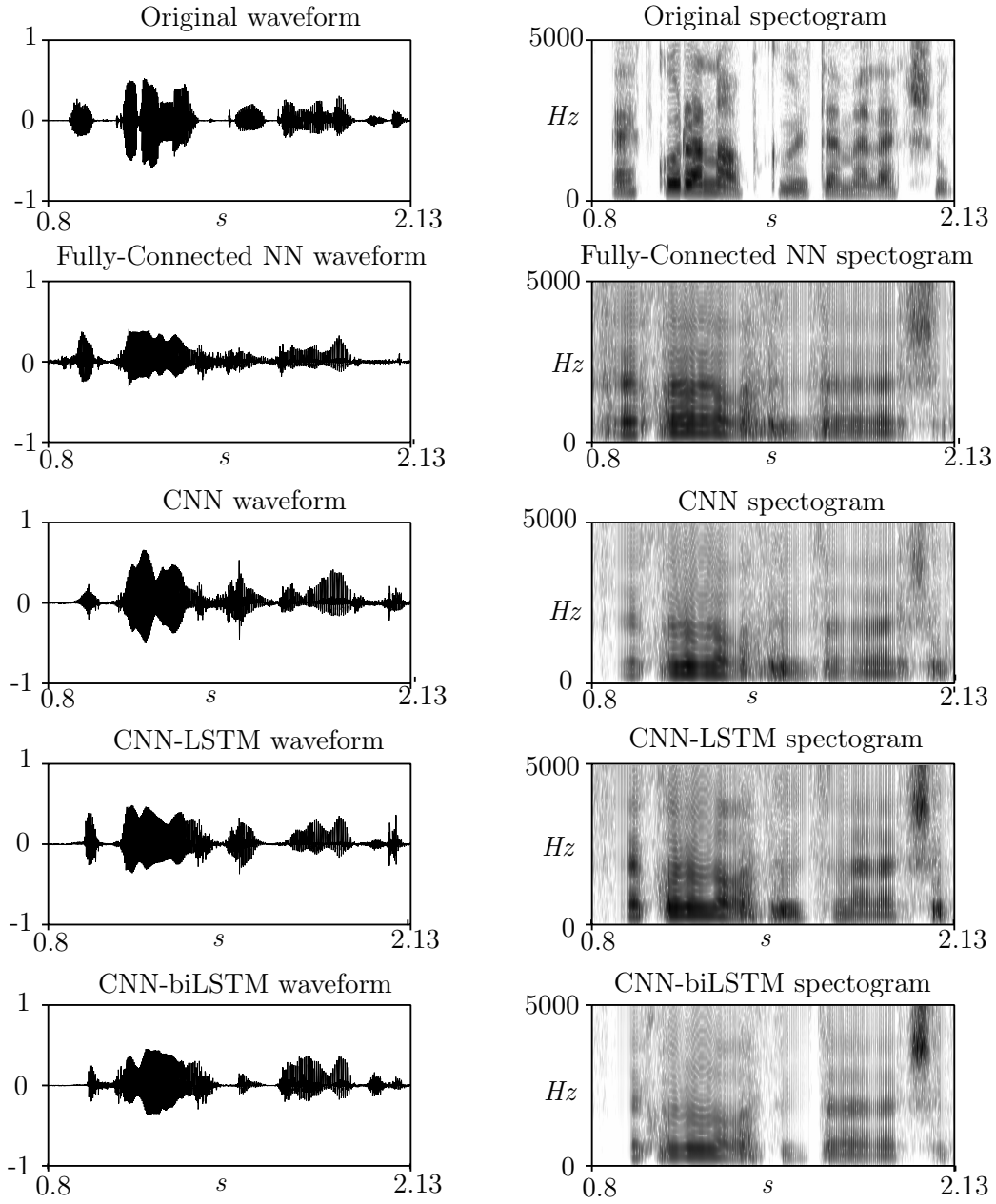


Figure 5.1 *Representation of an audio sample predicted by the different NNs.*

Figures 5.1 and 5.2 show the time-domain signal and spectrum from an 1.3 s original audio sample and the speech signals synthesized with the spectral coefficients predicted by the studied Neural Networks. Figure 5.1 shows all the networks without any preprocessing while figure 5.2 shows the results of the networks using the denoising autoencoder. It has been necessary to test the networks with and without the denoiser because its usefulness cannot be proven a priori.

Looking at figures 5.1 and 5.2 no definitive conclusion can be extracted, all waveform envelopes look far from the original one, but seems like those networks with LSTM layers can predict better the gain coefficient ω_0 and their envelopes look closer to the natural

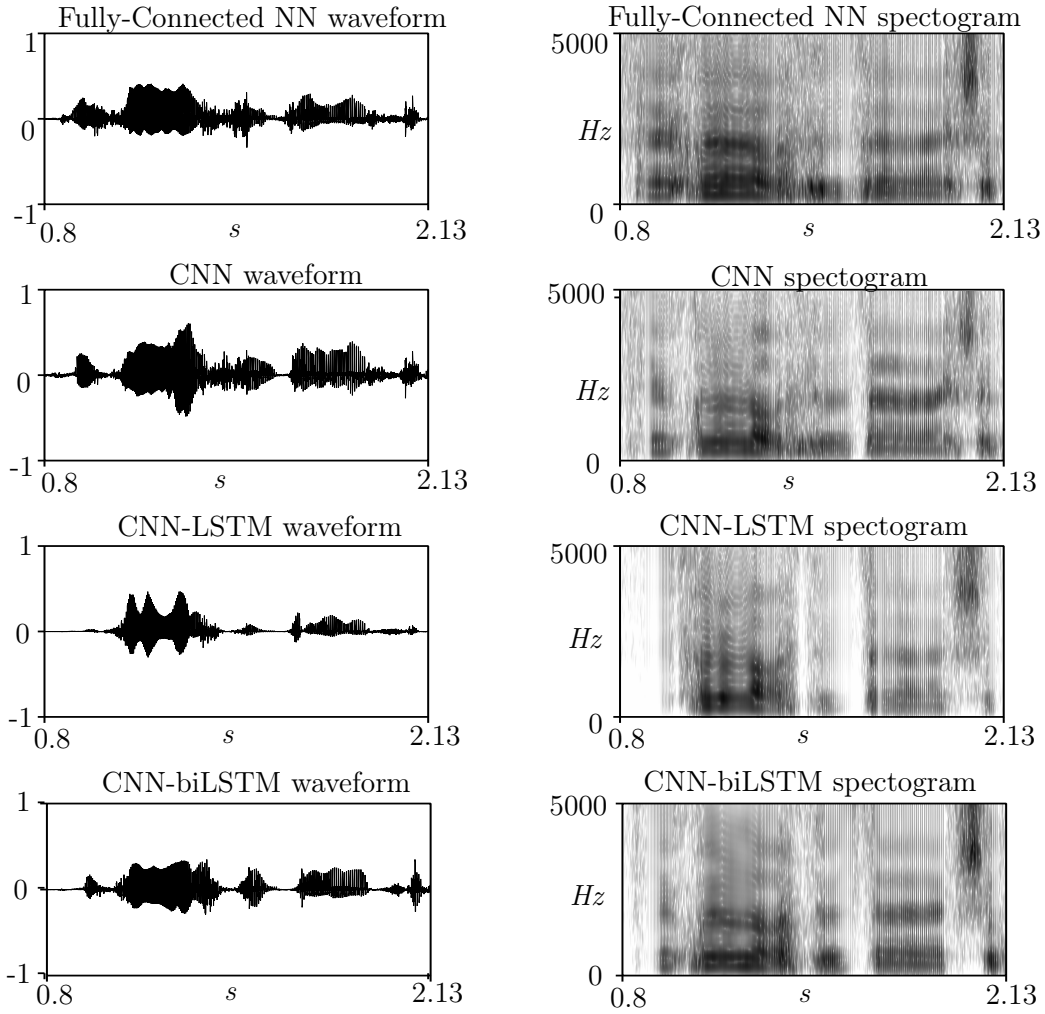


Figure 5.2 *Representation of an audio sample predicted by the different NNs using the denoising autoencoder.*

speech. Paying attention at the spectograms it is hard to realize which network does the best prediction of the formants, but seems like the LSTM networks have slightly better results, specially in high frequencies. Considering the loss and accuracy results, the worst network seems to be the basic fully-connected one, because it has the worst results in both denoised and non-denoised cases. Therefore, this proves that adding convolutional layers to the network increases the performance. Also CNN-LSTM networks have better test performance, as it could be expected after the hyperparameter optimization process. Comparing the results between using and non using the autoencoder, we can see that the results are slightly better using the denoiser for all four cases. This certifies that preprocessing and 'simplifying' the images helps to increase the network performance. The highest accuracy result, with a value of 0.34, belongs to the CNN-biLSTM network with preprocessing which is the most complex network of the list.

5.2 Listening subjective tests

In order to have a subjective opinion of the results, a MUSHRA (MUlti-Stimulous test with Hidden Reference and Anchor) listening test has been conducted [39]. This test contains the nine sentences from the test database and lets the listeners rate the naturalness of the speech. The test should not be 'exhausting' for the listeners, so it has been unfeasible to include all eight different audio samples in the test, instead of that, a smaller selection has been chosen. Due to the fact that the objective results have been better in the tests with the denoising autoencoder, the listening test includes only the audios of the four networks with preprocessing.

The advantage of using MUSHRA, instead of MOS (Mean Opinion Square), is that it requires fewer participants to obtain statistically significant results. This is because all four different systems are presented at the same time, altogether with the original sentence and a dummy version of it (with constant F0 and distorded spectrum), in order to add higher and lower references. This way, the test can be exposed to only trained listeners who know what typical artifacts sound like. In the test, the listeners have to rate the naturalness of each sample in a randomized order, from 0 (highly unnatural) to 100 (highly natural). The same process is repeated for all the nine sentences in the test database. Altogether, the listeners have to rate a total of 54 samples (6 types $\times$ 9 sentences).

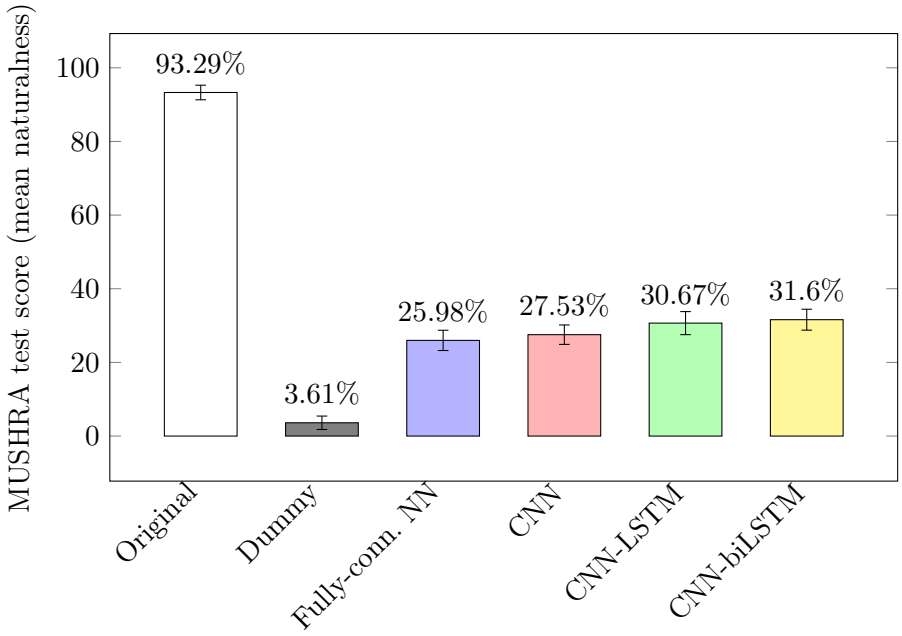


Figure 5.3 *Graph of the listening test scores. The higher the number, the more natural the system was found. The error bars show the 95% confidence intervals.*

Altogether 13 listeners participated in the listening test, being 4 of them speech researchers and 7 of them university students. The test took, on average, 11 minutes to complete. Figure 5.3 plots the results in a bar graph, showing the mean and the 95% confidence intervals. As it can be seen, all four Neural Networks have very similar results

(around 30%). The CNN-biLSTM was rated as the most natural, followed by the CNN-LSTM. These results show a substantial improvement compared to the fully-connected NN (26%) and CNN (27%). As the averages show, the listeneres were able to notice the differences between the four systems.

Chapter 6

Conclusions

This thesis described different deep learning experiments for the *Silent Speech Interface* application. The neural networks implemented had to predict Mel-Generalized Cepstrum features in Line Spectral Pair representation from ultrasound images of the tongue. Four different types of neural networks were investigated, including 1) a fully-connected feed-forward neural network, 2) a convolutional neural network, 3) a combination of a CNN and an LSTM layer, and 4) the addition of a second non-causal LSTM layer. In addition to this, a denoising deep autoencoder was designed for a preprocessing purpose. An exhaustive hyperparameter optimization process was conducted for all the networks (with and without denoising preprocessing).

After carrying out the objective and subjective tests, we found that the bidirectional CNN-biLSTM network was preferred in both objective and subjective terms. Even though, the difference between it and its causal version (the unidirectional CNN-LSTM network) is almost indistinguishable. Therefore, due to the fact that the CNN-biLSTM network would need more delay, the CNN-LSTM network would be better for a practical system.

It was proven that it is possible to synthesize a speech signal from only ultrasound images as input parameters using deep learning methods. The speech results were not totally satisfying for the listeners but, taking into account the limitations of the training database and the complexity of the problem, the resulting audios are considered as acceptable. The hypothesis that adding more complexity to the neural network would increase its performance was supported by both objective and subjective tests, verifying that every attempt to improve the system (the convolutional layers, the LSTMs and the denoising autoencoder) helped to obtain better results.

For a comercial purpose *Silent Speech Interface* application, more work needs to be done in order to obtain satisfying results. Several things can be studied in the future in order to improve the efficiency. One idea that has not been implemented in this thesis is adding multimodal articulatory data, specifically images of the lips. Another improvement would be implementing more complex image preprocessing (e.g. tongue tracking algorithms).

Acknowledgements

I would first like to thank my thesis supervisor Tamás Gábor Csapó of the Faculty of Electrical Engineering and Informatics at BME. His guidance helped me in all the time of research and gudiance of this thesis. He was always helpful every time I was stuck in any kind of issue.

I thank the MTA-ELTE 'lendület' Lingual Articulation Research Group for providing the tongue ultrasound recordings.

The Titan X GPU used for this research was donated by NVIDIA Corporation.

I would also like to thank all the listeners who participated in the subjective test.

List of Figures

List of Tables

Bibliography

[1] Tamás Gábor Csapó, Tamás Grósz, Gábor Gosztolya, László Tóth, and Alexandra Markó. Dnn-based ultrasound to speech conversion for a silent speech interface. In *INTERSPEECH*, pages 3672 – 3676, 2017.

[2] B. Denby, T. Schultz, K. Honda, T. Hueber, J.M. Gilbert, and J.S. Brumberg. Silent speech interfaces. In *Silent Speech Interfaces*, volume 52, pages 270 – 287, 2010.

[3] J. Cai, T. Hueber, S. Manitsaris, P. Roussel, L. Crevier-Buchman, M. Stone, C. Pillot-Loiseau, G. Chollet, G. Dreyfus, and B. Denby. Vocal tract imaging system for post-laryngectomy voice replacement. In *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 676–680, May 2013.

[4] Eric David Petajan. *Automatic Lipreading to Enhance Speech Recognition (Speech Reading)*. PhD thesis, Champaign, IL, USA, 1984. AAI8502266.

[5] B. Denby and M. Stone. Speech synthesis from real time ultrasound images of the tongue. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages I–685–8 vol.1, May 2004.

[6] T. Hueber, G. Aversano, G. Cholle, B. Denby, G. Dreyfus, Y. Oussar, P. Roussel, and M. Stone. Eigentongue feature extraction for an ultrasound-based silent speech interface. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 1, pages I–1245–I–1248, April 2007.

[7] Y. Nakajima, H. Kashioka, K. Shikano, and N. Campbell. Non-audible murmur recognition input interface using stethoscopic microphone attached to the skin. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, volume 5, pages V–708–11 vol.5, April 2003.

[8] J. A. Gonzalez, L. A. Cheah, A. M. Gomez, P. D. Green, J. M. Gilbert, S. R. Ell, R. K. Moore, and E. Holdsworth. Direct speech reconstruction from articulatory sensor data by machine learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(12):2362–2374, Dec 2017.

[9] D R Brown III, K Keenaghan, and S Desimini. Measuring glottal activity during voiced speech using a tuned electromagnetic resonating collar sensor. volume 16, page 2381, 2005.

[10] L. Diener, M. Janke, and T. Schultz. Direct conversion from facial myoelectric signals to speech using deep neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, July 2015.

[11] Fabrice Wendling, Fabrice Bartolomei, Jean-Jcques Bellanger, and Patrick Chauvel. Interpretation of interdependencies in epileptic signals using a macroscopic physiological model of eeg. In *Clinical neurophysiology*, volume 112, pages 1201–18, 08 2001.

[12] Anton Nijholt, Desney S. Tan, Gert Pfurtscheller, Clemens Brunner, Jose del R. Millan, Brendan Allison, Bernhard Graimann, Florin Popescu, Benjamin Blankertz, and Klaus-Robert Müller. Brain-computer interfacing for intelligent systems. In *IEEE Intelligent Systems*, volume 23, pages 72–79, 05 2008.

[13] Jun Wang, Ashok Samal, and Jordan Green. Preliminary test of a real-time, interactive silent speech interface based on electromagnetic articulograph. 06 2014.

[14] Thomas Hueber, Elie-Laurent Benaroya, Gérard Chollet, Bruce Denby, Gérard Dreyfus, and Maureen Stone. Development of a silent speech interface driven by ultrasound and optical images of the tongue and lips. volume 52, pages 288 – 300, 2010. Silent Speech Interfaces.

[15] Andrew Stuart, Joseph Kalinowski, Michael Rastatter, and Kerry Lynch. Effect of delayed auditory feedback on normal speakers at two speech rates. In *The Journal of the Acoustical Society of America*, volume 111, pages 2237–41, 05 2002.

[16] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. volume 18, pages 1527–1554, Cambridge, MA, USA, July 2006. MIT Press.

[17] Aurore Jaumard-Hakoun, Kele Xu, Clémence Leboullenger, Pierre Roussel-Ragot, and Bruce Denby. An articulatory-based singing voice synthesis using tongue and lips imaging. In *ISCA Interspeech 2016*, volume 2016, pages 1467–1471, 2016.

[18] Lorenz Diener, Christian Herff, Matthias Janke, and Tanja Schultz. An initial investigation into the real-time conversion of facial surface emg signals to audible speech. In *Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, volume 2016, pages 888–891, 08 2016.

[19] Florent Bocquelet, Thomas Hueber, Laurent Girin, Christophe Savariaux, and Blaise Yvert. Real-time control of an articulatory-based speech synthesizer for brain computer interfaces. volume 12, pages 1–28. Public Library of Science, 11 2016.

[20] Sandesh Aryal and Ricardo Gutierrez-Osuna. Data driven articulatory synthesis with deep neural networks. volume 36, pages 260–273, London, UK, UK, March 2016. Academic Press Ltd.

[21] Maureen Stone. A guide to analysing tongue motion from ultrasound images. volume 19, pages 455–501. Taylor & Francis, 2005. PMID: 16206478.

[22] Jens E Wilhjelm, Martin Kristensson, and Ole Trier Andersen. Medical diagnostic ultrasound -physical principles and imaging. 04 2018.

[23] Tamás Gábor Csapó, Andrea Deme, Tekla Etelka Graczi, Gergely Varjasi, and Alexandra Markó. Synchronized speech, tongue ultrasound and lip movement video recordings with the micro system. In *CAPSS*, 2017.

[24] B. Mazumdar, A. Mediratta, J. Bhattacharyya, and S. Banerjee. A real time speckle noise cleaning filter for ultrasound images. In *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*, pages 341–346, 2006.

[25] A. Hazra, J. Bhattacharyya, and S. Banerjee. Real time noise cleaning of ultrasound images. In *Proceedings. 17th IEEE Symposium on Computer-Based Medical Systems*, pages 379–384, June 2004.

[26] S. Raslain, F. Hachouf, and S. Kharfouchi. Using 2d arma-garch for ultrasound images denoising. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2672–2676, Sept 2017.

[27] A. Roussos, A. Katsamanis, and P. Maragos. Tongue tracking in ultrasound images with active appearance models. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 1733–1736, Nov 2009.

[28] H. Wang, S. Wang, B. Denby, and J. Dang. Automatic tongue contour tracking in ultrasound sequences without manual initialization. In *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 200–203, Dec 2015.

[29] L. Tang and G. Hamarneh. Graph-based tracking of the tongue contour in ultrasound sequences with adaptive temporal regularization. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pages 154–161, June 2010.

[30] Keiichi Tokuda, Takao Kobayashi, Takashi Masuko, and Satoshi Imai. Mel-generalized cepstral analysis - a unified approach to speech spectral estimation. In *ICSLP*, 1994.

[31] Ian Vince McLoughlin. Line spectral pairs. volume 88, pages 448 – 467, 2008.

[32] Tamás Gábor Csapó, Tamás Grósz, Gábor Gosztolya, László Tóth, and Alexandra Markó. F0 estimation for dnn-based ultrasound silent speech interfaces. In *ICASSP*, 2018.

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[34] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. volume 15, pages 1929–1958, 2014.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural computation*, volume 9, pages 1735–80, 12 1997.

[37] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. volume 12, pages 2451–2471, Cambridge, MA, USA, October 2000. MIT Press.

[38] Pushparaja Murugan. Hyperparameters optimization in deep convolutional neural network / bayesian approach with gaussian process prior. volume abs/1712.07233, 2017.

[39] ITU-R. Recommendation bs.1534: Method for the subjective assessment. Technical report, International Telecommunication Union, 2001.