Convolutional-recurrent models for speech enhancement

Developing a single-channel denoising algorithm with deep learning

Project Report

Christie Laurent
Riccardo Miccini

Aalborg University
Department of Architecture, Design and Media Technology
Faculty of IT and Design

This report was typeset using the LaTeX typesetting system originally developed by Leslie Lamport, based on TeX created by Donald Knuth. All figures, unless otherwise noted, are generated using Matplotlib by J. D. Hunter for Python 3.

AALBORG UNIVERSITY

STUDENT REPORT

Department of Architecture, Design and Media Technology, Faculty of IT and Design
Aalborg University
http://www.aau.dk

Title:
Convolutional-recurrent models for speech enhancement - developing a single-channel denoising algorithm with deep learning

Theme:
Machine Learning, Signal Processing

Project Period:
Spring Semester 2019

Project Group:
1

Participant(s):
Christie Laurent
Riccardo Miccini

Supervisor(s):
Tsampikos Kounalakis
Cumhur Erkut

Lars Bøgild Christensen
Ulrik Kjems

Copies: 1

Page Numbers: 96

Date of Completion:
September 25, 2019

Abstract:

The aim of this work is to implement a single channel speech enhancement algorithm using deep neural networks. Several models are devised and inspected, whose architectures are inspired from state-of-the-art models such as fully-convolutional autoencoders and convolutional-recurrent network. The experimental adoption of Gated Recurrent Units (GRUs) and novel Temporal Convolutional Networks (TCNs) instead of Long-Short Term Memory (LSTM) layers is presented and discussed. The models are trained using several tiers of datasets, composed of speech data from news broadcasts mixed with pink noise and different real-world noises. Different data representations are tested, all of them based on spectrogram. The performance of each model is evaluated in terms of speech quality and intelligibility using Blind Source Evaluation, Perceptual Evaluation of Speech Quality (PESQ) and Short-Term-Objective-Intelligibility (STOI) metrics. The results are compared to those of a state-of-the-art baseline noise reduction system and relevant findings are exposed and discussed.

Contents

Preface

This report has been authored as part of a 15-ECTS semester project for Sound and Music Computing, with focus on Signal Analysis.

The project has been carried out in collaboration with *Retune DSP*. The opinions expressed in this document are the authors' own and do not reflect the view of Retune DSP and its personnel. All brands, product names, logos, or other trademarks featured or referred to in this document are the property of their respective holders, and their use does not imply endorsement.

We would like to thank our internal supervisors Cumhur and Tsampikos for their precious guidance and feedback, Ulrik and Lars from Retune DSP for the possibility of working with them as well as their useful pointers, and all the people from SMC at Aalborg University Copenhagen for their constant support.

Aalborg University, September 25, 2019

Christie Laurent
claure18@student.aau.dk

Riccardo Miccini
rmicci18@student.aau.dk

Chapter 1

Introduction

Speech enhancement is a field that has been researched upon extensively over these last decades. It groups several techniques aiming to improve intelligibility and quality of noisy speech signals. For the purpose of communication or for precise analysis in signal computing, the cleanliness of a signal is indeed an issue at stake. Such a perfect signal does not exist in real life, although one can approach it in anechoic chambers that are highly constrained environments. In general however, signals are composed of several parts. Let x be an ordinary noisy speech signal, comprising the following main components x_s, x_n, H explicited here below, such as $x = H * x_s + x_n$ — as usually stated for a linear problem in Bayesian statistics:

Speech: first, x contains one or several useful signals x_s — called *message(s)* or *target(s)* further on — which need to be extracted from the entire mixture; in the case of several messages this problem is known as the "cocktail party" problem.

Additive noise: x is then generally blurred by some additive noise x_n, that can be characterized by a quasi-stationary or slowly-evolving spectrum, e.g. traffic in the background, by aperiodic occurrences and a fast-evolving spectrum, for instance children cry, or a combination of both.

Convolutive noise: finally, convolutive effect **H** can occur as well. Best example in Speech Enhancement field is the reverberation due to room effect, which is intrinsically intertwined with the rest of the mixture. Reverberation can sometimes be neglected, especially for very short source-listener distances or for lowly reverberant rooms. In this case, **H** is the unity function.

In this work, we focus on signals containing a single useful message, additive stationary and non-stationary noises and do not consider room effect. Therefore, we define the noisy time-domain signal $x(t)$ as a sum of the clean speech signal

$x_s(t)$ with some additive noise $x_n(t)$:

$$x(t) = x_s(t) + x_n(t) \tag{1.1}$$

with $x_n(t) = x_{n,stat}(t) + x_{n,nonstat}(t)$, where $x_{n,stat}(t)$ is a stationary noise component and $x_{n,nonstat}(t)$ is a non-stationary noise signal. The aim of this work is then to output $x_s(t)$ given $x(t)$ in discarding any noise component.

This processing task can be handled with various methods, depending on the number of recordings available for a given situation — or *scene*. Traditionally, *signal processing methods* are employed when several versions of the scene are available. Here however, we tackle single-channel signals, meaning that we can not correlate different versions of a same signal to extract the message. Therefore, instead of cross-checking all available instances of a scene, for each scene, independently one from another, we let a machine learning (ML) system learn patterns in a given set of data and apply this knowledge to unknown situations. The expression Machine learning is a generic term that comprises computing techniques aiming at decoding patterns in large sets of data, in order to achieve tasks such as classification (i.e. recognition) and/or regression.

Deep learning is an emerging branch of machine learning, which has found applications in speech enhancement tasks for more than a decade [Alo+18], and has produced state-of-the-art systems. This project builds on such results. In particular, we attempt to reproduce the architecture described in [Zha+18], while exploring variations of this model, namely based on Gated Recurrent Units (GRUs). We also introduce Temporal Convolutional Networks (TCNs) as a replacement for the recurrent network components, and assess the results with established quantitative metrics for noise separation, as well as speech intelligibility and quality.

In Chapter 2, we begin with recalling the work related to speech enhancement and we present existing work in deep learning for this field of application; in Chapter 3, we recall deep learning and data processing core concepts and both basic and more complex tools and architectures; in Chapter 4 we detail the building blocks of our work: the datasets used, the processing chain of this data, the models under investigation; in Chapter 5, we present the experiments carried as well as their results; finally in Chapter 6, we draw conclusions regarding the results that were computed and observed in the previous chapters, and highlight limitations and possible improvements.

Chapter 2

Related work

Ever since Colin Cherry coined the expression *cocktail party problem* in 1953 [Che53], speech enhancement and separation systems have been extensively researched by engineers and scientists encompassing multiple disciplines. In particular, single-channel solutions have been especially sought after due to their usefulness in applications where microphone arrays are not viable due to cost or space constraints, such as with fast-moving consumer products and hearing aids respectively.

Research on speech enhancement is inherently entwined with source separation, and the former can be effectively considered a subset of the latter problem. However, given that the focus of this project is on signals containing only one desirable source, this study is confined to speech enhancement solutions and methods identified as separation methods but which can apply for enhancement as well.

The following subsections discuss some of the existing work on the subject, highlighting the inherent challenges and shortcomings of existing systems. Firstly, traditional approaches based on statistical or digital signal processing techniques are presented. Subsequently, an introduction to deep learning — in particular within the audio domain — is provided, culminating with cutting-edge systems which have been found to be particularly relevant or inspiring.

2.1 Classical Approaches

Improving audio signal quality has been researched extensively over the recent history. Multiple approaches exist but we focus here on classical — signal processing — methods adapted to single-channel speech enhancement and leave aside methods for multi-talker situations, but the one that can be assimilated to enhancement algorithms in the case where the two targets are speech and noise (where noise can be of any sort: voice, broad-band noise, impacts, etc). A more

comprehensive description of the methods presented here can be found in [Kol18, Chapters 1.1 and 1.2]

Most of these methods imply Discrete Fourier Transform (DFT); consequently, let us introduce the capital letter notation which denotes the Fourier transform of the time-domain signal, e.g. $X = DFT(x)$.

Several classical methods are based on the analysis of the signal gain and follow the steps shown on Figure 2.1 and that we explicit here below.

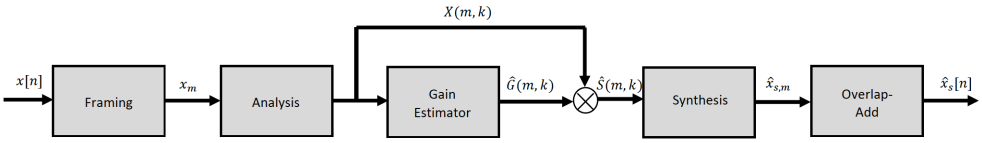


Figure 2.1: Classical gain-based speech enhancement system. [Kol18]

The noisy discretised time-domain signal $x[n]$ is first segmented into overlapping frames using a sliding window of length N. Each frame is then projected into the frequency domain by application of a Discrete Fourier Transform; let us call X this projection which is a Short-Time-Fourier-Transform (STFT). A gain $\hat{G}$ is then estimated (see following sub-sections) and applied multiplicatively and element-wise to X for every Time-Frequency (TF) bin (m,k) as follows:

$$\hat{S}(m,k) = \hat{G}(m,k)\, X(m,k) \tag{2.1}$$

where $\hat{S}$ refers to the enhanced speech spectrogram. $\hat{S}$ is then transformed back into time-domain where a overlap-add function reconstructs it as one-block. The output, denoted $\hat{x}_s[n]$, is supposed to be an enhanced version of the speech signal $x_s[n]$.

The different existing methods differ in regard to the way these gains are calculated. They can be divided in four families that are presented hereafter: spectral subtractive-based algorithms, statistical model-based algorithms, linear algebra-based algorithms, and finally machine learning-based algorithms.

2.1.1 Spectral subtractive-based algorithms

Spectral subtractive-based algorithms were amongst the first speech enhancement algorithms to be developed in the late 1970s [Bol79]. They are based on heuristics and consist in subtracting the absolute value of the noise term N from the mixture term X for each TF-bin and to modulate this difference with the mixture phase. They are simple to implement and effectively reduce noise in the processed mixture, as far as precise estimates of $|STFT(x_n)|$ bins are available and that the user deals with potential negative values that can occur in spectrograms

due to estimation errors. Regarding the former point, noise is usually estimated from non-speech periods prior to speech activity. As for the latter issue about negative values, one can arbitrarily put them at 0 or at values taken from neighboring frames. However, these precautions don't prevent this method to be prone to *musical noise*, which is a type of signal distortion due to estimation errors in the estimate of the noise magnitude.

2.1.2 Statistical model-based algorithms

In order to give a stronger mathematical frame to the denoising process, statistical algorithms were introduced. These algorithms are set as optimization problems, where an objective function is minimized. Minimum Mean Squared Error (MMSE) algorithms are amongst the most commonly used today. We present hereunder Wiener filters and the Short-Time Spectral Amplitude (STSA)-MMSE algorithm which belong to this class.

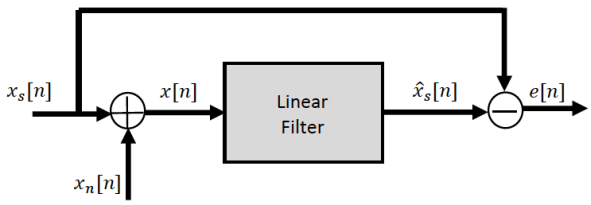


Figure 2.2: Linear optimization problem with mean-square error

Wiener filters Wiener filters are MMSE optimal linear filters for the linear estimation problem shown in Figure 2.2, where the observed signal $x[n]$ is given as in Equation 1.1 by $x[n] = x_s[n] + x_n[n]$, where $x_s[n]$ and $x_n[n]$ are assumed to be uncorrelated and stationary stochastic processes. Wiener filters can have either a Finite Impulse Response (FIR) or an Infinite Impulse Response (IIR) or even be non-causal. For the causal FIR Wiener filter, the estimated signal $\hat{x}_s[n]$ is given by

$$\hat{x}_s[n] = \underline{h}_0^T \, \underline{x}(n) \qquad \text{with} \quad \underline{h}_0 = (\underline{\underline{C}}_{\underline{ss}} + \underline{\underline{C}}_{\underline{nn}})^{-1} \, \underline{r}_{s\underline{X}_s},$$

where $\underline{\underline{C}}_{\underline{ss}}$ and $\underline{\underline{C}}_{\underline{nn}}$ denote the autocorrelation matrices of x_s and x_n, respectively, $\underline{r}_{s\underline{S}}$ denote the autocorrelation vector, and $\underline{x}(n) = \left[x[n],\, x[n-1],\, \ldots,\, x[n-L+1]\right]^T$ are the past L samples of the observed signal [Loi13].

An alternative to the time-domain Wiener filter is the frequency-domain Wiener

filter where G is found to be:

$$G_W(w) = \frac{\xi_w}{\xi_w + 1} \tag{2.2}$$

for a non-causal filter $\underline{h}$ of infinite duration.
This is of interest, since ξ_w refers directly to linear *a priori* Signal-to-Noise Ratio (SNR) — notion further explained under its logarithmic form in 3.4, at frequency w. Explicitly, this filter is real and positive, with values in between 0 and 1 and behaves as a low-pass filter regarding $SNR(w)$: for low SNRs (predominance of the noise upon speech), $X(w)$ is modulated by a coefficient close to 0. Inversely for high SNRs (predominance of the speech upon noise), $X(w)$ is almost unchanged. Another advantage is that it doesn't change the phase of the mixture. However some knowledge about speech and noise are required to be able to calculate the filter coefficients.

Although the Wiener filter is considered the optimal *complex* spectral estimator, it is not the optimal spectral *amplitude* estimator, which was researched upon at the time when phase was thought unsignificant for speech enhancement (see e.g. [WL82]); this belief led to the development of optimal spectral amplitude estimators, commonly known as STSA-MMSE estimators and presented in the next paragraph.

STSA-MMSE STSA-MMSE estimators are based on Bayesian statistics, where explicit assumptions are made about the probability distributions of speech and noise in the frequency domain.

After computing the Bayesian Mean Squared Error on the clean and cleaned speech, manipulating the equation thus obtained, and stating hypotheses about the speech and noise statistics (e.g. statistically independent, zero-mean, Gaussian distributed random variables, which is motivated by the fact that STFT coefficients become uncorrelated, and therefore independent under a Gaussian assumption, with increasing frame length), one ends up with an equation similar to 2.1, at the difference that $G_w = G(\phi_w, \gamma_w)$ depends now on two parameters, for a given gain function:

$$\phi_w = \frac{\mathbb{E}\{|S(w)|^2\}}{\mathbb{E}\{|N(w)|^2\}} \quad \text{and} \quad \gamma_w = \frac{X^2(w)}{\mathbb{E}\{|N(w)|^2\}}.$$

where $\mathbb{E}$ represents the expectation operator. The term ϕ_w is referred to as *a priori SNR* since $\phi_w \approx [SNR(w)]^3$, and γ_w is referred to as the *a posteriori SNR* as it reflects the SNR of the noisy speech signal. Therefore, STSA-MMSE is a function of *a priori* and *a posteriori* SNRs. When compared to the Wiener gain, the STSA-MMSE gain generally introduces less artifacts at low SNRs, partially due to the a posteriori SNR. Nonetheless, it also needs an *a priori* estimation of SNRs.

2.1.3 Linear algebra-based algorithms

The third class of enhancement algorithms contains subspace-based algorithms, which are primarily derived using linear algebra principles.

Subspace based algorithms The general underlying assumption behind these methods is that K-dimensional vectors of speech signals do not span the entire K-dimensional euclidean space, but are instead confined to a smaller M-dimensional subspace.
The aim of these algorithms is then to project the mixture into the clean speech space — or signal space, whose dimensions and main components are estimated by Principal Component Analysis (PCA). [HW+06]
These methods hold under standard assumptions, stating that speech x_s and noise signals x_n are stationary, uncorrelated and zero-mean random processes.

Like previous methods, these algorithms need an estimate of some statistics — the covariance matrix — of the clean speech, noise, and whole signal; plus, an estimation of the signal space dimension M is required. This second element is generally time-varying which makes it complex to estimate precisely; it has nonetheless a non-negligible impact if badly estimated: M overestimated means that some noise remains in the final product while M underestimated implies that some of the signal subspace is discarded. Consequently, the quality of these estimates highly influences the performance of subspace-based speech enhancement algorithms. Nevertheless, these algorithms are capable to improve speech intelligibility for hearing impaired listeners wearing cochlear implants especially [LLH05].

Non-negative Matrix Factorization Previous algorithms reach their limits when it comes to precisely estimate data statistics, particularly noise statistics, which is especially true when the noise signal is also a speech signal — i.e., non-stationary. In this very case, we can consider our initial single-channel denoising problem as a single-channel separation problem in the simplest "cocktail-party" situation, i.e. with two components to separate: the voice target and the voice noise. Non-negative Matrix Factorization (NMF), another algebra-based method, can then be considered for it allows to consider time-varying statistics. Plus, it deals with positive data and is therefore adapted to audio, which is not supposed to take negative values.

The underlying assumption of NMF is that a non-negative data matrix $V \in \mathbb{R}^{K\,x\,M}$ can be approximately factorized as $V \approx WH$ with $W \in \mathbb{R}^{K\,x\,L}$ a dictionary matrix and $H \in \mathbb{R}^{L\,x\,M}$ an activation matrix. With our notation, $V = \hat{S}$ our estimate speech signal under the chosen representation. NMF thus aims to reconstruct a good enough but compressed representation of the data of interest — the target speech in our case.

W can be seen as a set of basis vectors for the data matrix V and the columns of H represent how much of each basis vector, and at what time, is needed to represent each column of V. The dimension L — the size of the dictionary — is a tuning parameter that controls the accuracy of the approximation and is specified experimentally, after counting the number of different talkers and noises one may expect. In our case, where we consider one target and one noise signal, $L = 2$. V is calculated in solving an optimization problem (see 3.1.3) with a "divergence" as loss function, such as the Euclidean distance. This optimization problem is solved iteratively, based on the newest W and H that are updated at each iteration based on the former value of V,H and W. The clean speech signal is reconstructed in extracting the proper "word" from the dictionary and time occurrences from the activation matrix.

NMF is a simple but efficient technique for single-microphone multi-talker speech separation as well as for speech enhancement. However, NMF is a linear model and as such is limited in its model capacity. Second, NMF generally requires speaker dependent dictionaries, making NMF less suitable for speaker, or noise-type independent applications. Third, as signal-dependent activations Hs are required, applying NMF to real-time applications — where low latency is critical — is difficult. Finally, due to the computational complexity of the update equations, NMF performs badly for large datasets.

Even though the aforementioned methods have proven capable of improving the quality of a noisy speech signal, the conditions and estimates necessary for its validity make their usage questionable in reality. Plus, they generally do not improve speech intelligibility for normal hearing listeners [KL11].

2.1.4 Machine Learning

A different approach to the speech enhancement task is to consider it as a *supervised learning problem*. This approach states that the clean version of the input speech can be learned from the observations of a set composed of corresponding pairs of clean and noisy speech signals.

Specifically, instead of defining a clean-speech estimator straight from the beginning using mathematical principles, statistical assumptions and *a priori* knowledge, the estimator is determined by a parameterized mathematical model, that represents a large function space, potentially with universal approximation properties such as Gaussian Mixture Models (GMMs) [MB88], Support Vector Machines (SVMs) [HG03], or Artificial Neural Networks (ANNs) (see next sections for detailed information, particularly 2.2 and 3.1). The parameters of these ML-models are then found as the solution to an optimization problem with respect to an objective function evaluated on a representative dataset (see 3.1.3 for more on opti-

mization).

This approach is looser in regard to the amount of required conditions: neither restrictions e.g. about linearity, nor explicit assumptions e.g. about stationarity or uncorrelated signals, are imposed on the model. Instead, the ML-model is implicitly tasked to extract any signal feature which it finds to be relevant for solving the optimization problem. In our case, the task consists in retrieving a speech signal from a noisy observation — also called *mixture*.
ML techniques therefore offer the advantages to downsize the initial constraints of the problem and to be applicable to a larger panel of problems than traditional signal processing methods.

In the following sections, we confine our perspective to deep learning which is the main topic of this report.

2.2 Overview of Deep Learning

Deep learning is a branch of machine learning interested in algorithms consisting of multiple computational layers and featuring elements of non-linearity.

The predominant types of deep learning models currently researched fall under the generic name of *artificial neural networks*. Originally inspired by how information is processed within biological systems such as synaptic structures, ANNs are more commonly interpreted as universal *function approximators*, that is, systems capable of approximating any continuous function for a given compact interval, i.e. a subset of a Euclidean space. Deep learning methods are therefore capable of modelling very complex nonlinear relations by means of observing relevant examples, without making any assumption about the data.

Despite that notion — known as the *universal approximation theorem* — has been proven mathematically in 1989 [Cyb89] and in 1991 [Hor91] (for single-hidden layer networks using sigmoid non-linearities, and multi-layer feed-forward networks, respectively), the field has mostly lain dormant due to the computational costs associated with it, as well as the complexity of training.

In recent years, the development of pre-training techniques (based on *Restricted Boltzmann machines* (RBM) and *Autoencoders*) capable of speeding up convergence led to a renewed interest in the field [HS06]. Another major breakthrough which determined the resurgence of deep learning came from the adoption of GPU hardware for computing, which in turn allowed newer and better performing techniques to be developed [KSH12]. Since then, deep learning has gradually superseded traditional techniques in an ever-increasing number of fields such as computer vision, natural language processing, recommendation systems, and so forth. Furthermore, the release of high-level deep learning frameworks such as Tensorflow, Theano, and PyTorch has helped making the technology more accessible and

easier to deploy in real-world scenarios. For a more comprehensive discussion of deep learning, refer to Section 3.1.

One of the finest examples of deep learning applied to audio processing is undoubtedly WaveNet [Oor+16], a generative model for raw audio. This neural network uses a deep stack of multiple convolutional layers with dilated kernels to learn patterns in time-domain audio material such as music or speech, and is widely used in text-to-speech applications. In 2017, an autoencoder model based on WaveNet-style layers called NSynth [Eng+17] was used to derive latent representations of sounds from musical instruments, allowing users to interpolate between them, thereby paving the way for new synthesis paradigms. NSynth has been developed by the Google Magenta team, who combines the potential of deep learning techniques with music and arts.

The following section elaborates on deep learning-based solutions for speech enhancement.

2.3 State of the art

As established in the introduction chapter, challenges associated with speech enhancement may be divided into *denoising* and *dereverberation*. While the former deals with additive noise coming from undesired sources, the latter is concerned with the disturbance introduced by the properties of the enclosing environment and location of speaker and listener, which can be modeled as convolution with an impulse response.

Wang and Chen [WC17] offer an overview of the evolution of deep learning applied to speech separation; the most important milestones in regards to denoising are summarized below.

One of the first attempts at adopting supervised deep learning algorithms in speech enhancement came in late 2012 from [YD13], where noisy speech is passed through a 64-channel gammatone filterbank, whose output is then fed into a deep neural network (DNN) with the purpose of deriving acoustic features. These automatically extracted feature are then fed into a linear discriminator (namely, a *Support Vector Machine*) in order to estimate an *Ideal Binary Mask* (IBM). This latter is a two-dimensional representation where each dimension corresponds to time and frequency respectively, and its values may be either 1 or 0 depending on the SNR. Clean speech can then be estimated by applying the IBM to its respective noisy spectrogram. The entire procedure is documented in Figure 2.3.

An extension of the aforementioned system [Hea+13] generates an intermediate mask whose values represent the probability of speech in a given T-F cell. The mask is then fed into a second stage of DNNs which are trained to compute the IBM. This solution showed substantial improvement of intelligibility for hearing-

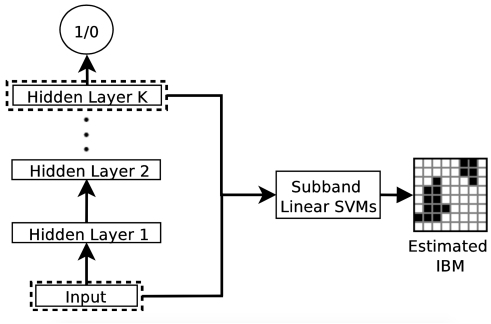


Figure 2.3: feature obtained with deep neural network used by linear SVM for IBM estimation [YD13]

impaired subjects in noisy conditions.

Subsequent algorithms focus on *spectral mapping* — that is, estimating a Time-Frequency (T-F) representation of the clean speech such as its magnitude spectrogram — instead of masking. In 2013, [Lu+] constructs a deep model composed of multiple autoencoder networks which are individually trained and later fine-tuned together. Similarly, [Xu+14] uses a log power spectrum as its target and applies pre-training with RBM instead of autoencoders, exceeding performances of traditional methods in terms of speech quality. The resulting complete denoising system is presented in Figure 2.4.

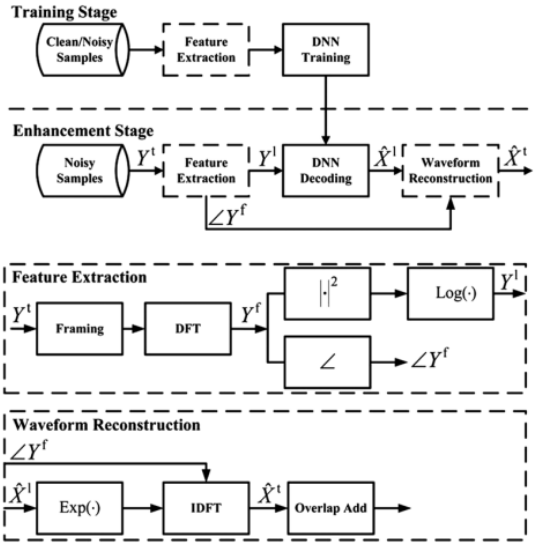


Figure 2.4: block diagram of denoising system based on spectral mapping [Xu+14]

In an effort to tackle phase reconstruction, [Fu+17] adopts a novel T-F representation based on rectangular notation rather than magnitude alone along with multi-metrics learning with a log power spectrum reconstruction term.

With the recent rise to popularity of more sophisticated classes of DNNs such as convolutional neural networks (CNN) and recurrent neural networks (RNN), successful attempts at adopting them have been performed.

In a 2014 work, recurrent models with Long-Short-Term-Memory (LSTM) layers are trained to estimate either an *Ideal Ratio Mask* (IRM, similar to the aforementioned IBM, except non-binary) or spectral mapping, proving more effective than dense DNNs [Wen+14].

In 2015, [KZ15] explores fully-convolutional autoencoders by comparing several empirically-derived architectures, paving the way to further research. Later on, [PL16] compares several encoder-decoder models, loosely based on autoencoders, called *Redundant Convolutional Encoder Decoder* (shown in Figure 2.5), achieving better performances than its fully-connected baseline; these models introduce a number of innovations such as skip connections between non-adjacent layers and no dimensionality reduction. Even more recently, [GP17] successfully applies fully-convolutional denoising autoencoders to source separation tasks.

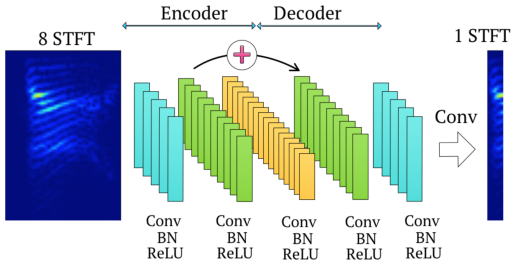


Figure 2.5: redundant Convolutional Encoder Decoder model [PL16]

RNN and CNN have also been used conjointly in hybrid architectures, showing promising results. In a recently presented convolutional-recurrent model, features are extracted from noisy input spectrograms using a convolutional layer; these features are then concatenated and fed into a bi-directional recurrent network, whose output sequence is used to estimate clean spectrograms [Zha+18].

Similarly, the model constructed in [Fak+18] uses convolutional layers applied to a narrow context window to compute features for a given time step, which is then fed into LSTM layers. A multi-task learning criterion is also introduced, based on a secondary RNN used for creating a language model. The overall system is documented in Figure 2.6

Both works show substantial improvement in perceptual quality and word recognition over state-of-the-art and production baselines.

Finally, more exotic approaches combining deep learning and traditional denoising techniques have been researched, including applying backpropagation to learn activation coefficients in deep NMF [LRHW15], and introducing a classic *decision-directed approach* as pre- and post-processing steps in an RNN trained to

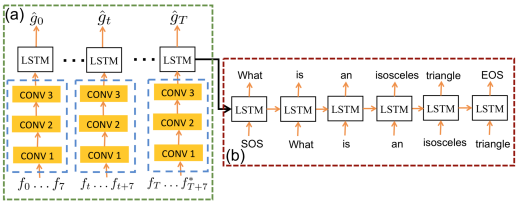


Figure 2.6: multi-task learning architecture [Fak+18]

estimate both spectral mapping and IRM [Tu+18].

Chapter 3

Methodology

This chapter aims at providing the necessary knowledge foundations upon which the project is based. In particular, the following aspects are covered: deep learning concepts, audio data representation and processing techniques, and evaluation metrics.

3.1 Deep Learning

Deep learning is a branch of machine learning investigating architectures of computational networks composed of multiple layers and aiming at solving complex problems.

The expression *deep learning*, although originally used to designate networks with more than one layer, is today employed for any network whose process results in solutions both complex and of relative high quality, independently of their number of layers.

The general architecture of a neural network consists of an *input* which is manipulated through a series of *layers* performing an arbitrary computation, before being outputted through the last layer. Each layer is composed of a set of neurons, also called *nodes* or *units*. These neurons have learnable parameters (e.g. weights and biases) that are used, along with the input of the layer, to compute an output according to a user-defined connectivity. Any of the layers that are not directly linked to the inputted data or to the outputted data are called *hidden*. During the training stage, the learnable parameters are iteratively estimated and updated by the network, as described in 3.1.3.

The deeper these networks are, the more complex they become even though the relation is not entirely linear since neural networks are parameterizable in many ways; and the more complex they are, the more difficult tasks they can achieve — ideally. This complexity stems from the increasing number of connections between

layers and can be estimated through the total number of parameters.

As mentioned in Section 2.2, the computation occurring inside the network nodes is a rather coarse model of the biological system — namely how neurons in the brain receive impulses from their neighbours through their dendrites and propagate them further through their axons. Nevertheless, deep learning does not set to rigorously replicate biological systems, and an abundance of network topologies and other heuristics have been developed, with some of them also inspired by other biological processes.

A common issue with deep learning — and machine learning as a whole — is *overfitting*, which is said to occur when a system becomes exceedingly good at modeling its training data, to the point of losing the ability to fit (i.e. predict or categorize) new data, also known as *generalization*. This is of particular relevance in the domain of speech enhancement, as noise signals may manifest themselves in countless different ways [WC17]. Over time, several remedies and mitigation strategies have been developed, some of which are reported below.

The following subsections will present some of the most relevant deep network architectures. In this section, we focus on characteristics related to the general architecture exclusively. Specific aspects such as the number of layers used in a model as well as other functional parameters are covered in Section 4.3.

3.1.1 Multilayer Perceptron

Multilayer perceptrons (MLP) are the most basic class of artificial neural networks.

MLPs comprise a minimum of three layers — an input layer, one or more hidden layers, and an output layer. Each layer is, in turn, composed of nodes which are connected with all the nodes from the previous layer forming a feed-forward network, as exemplified by Figure 3.1. Each node computes the weighted sum of its inputs, offsets it by a scalar bias, and applies a nonlinearity (or activation function).

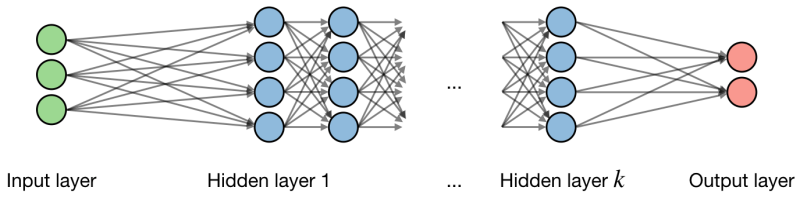


Figure 3.1: multi-layer perceptron architecture [AA18]

The node weights in multilayer perceptrons represent the strength of the synapses connecting each neuron, and the activation function models their firing rate. Stan-

dard "s-shaped" activation functions have been commonly used for such purpose, due to their ability of squashing a real-valued input into a confined range, therefore simulating saturation phenomena.

This can be expressed as:

$$y = \phi\left(\sum_i w_i x_i + b\right) \tag{3.1}$$

where x_i are the outputs of nodes in the previous layer, w_i are their respective weights, b is a bias, and $\phi()$ is the activation function. The optimal values of weights and biases for each neuron are learned during a training stage.

The presence of a nonlinear activation and multiple layers allows this system to work as a universal function approximator, capable of distinguishing data that is not linearly separable.

During the training stage, examples of relevant input data and their expected output (target) are fed into the network, which computes its output, one layer at a time. The network output is then compared with its target using a cost function, and the network parameters (its weights and biases) are updated in order to minimize said function. Common cost (also called loss) functions are *mean squared error* for regression problems and *cross entropy* for classification ones. A more comprehensive description of the training procedure, including algorithms used for minimizing cost functions, can be found in Section 3.1.3.

Architectures based entirely on MLPs are rarely used, and depending on the task, recurrent or convolutional networks (described in the following sections) prove more efficient and successful. Nevertheless, layers of neurons like those presented above are often found in the final stages of more sophisticated network architectures, especially when performing classification tasks [PLS16]. They are normally referred to as *dense* or *fully-connected* layers.

3.1.2 Activation Functions

Activation functions are a fundamental element in all deep learning systems. They allow networks to model complex patterns and relationships, that are not linearly separable.

An activation function is a transfer function that maps its real-valued input to an output. Such output is usually limited to a subset of the input domain.

For a function to be used effectively as activation, there are two main desirable properties. Firstly, as mentioned above, *nonlinearity* is what allow deep networks to act as universal function approximators; conversely, a deep network with activations $y(x) = x$ can be combined into a single layer (i.e. by multiplying the weight matrices together) thus collapsing into a linear regression model. Secondly, gradient descent-based optimization algorithms require the activation function to

be *continuously differentiable*. However, since landing exactly on a minimum is not paramount for a successful training (as it would most likely entail overfitting), it is acceptable for the cost function to have an undefined gradient at its minima.

There exist a number of activation functions that have been devised, from the biologically-inspired, to those which have been empirically observed [Sri+14b] to work better in practice. The most common ones are presented in Figure 3.2 and described below.

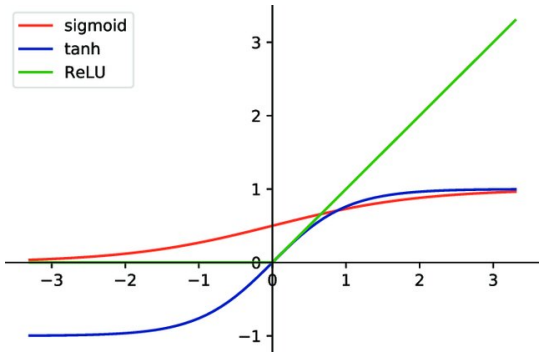


Figure 3.2: commonly-adopted activation functions

Logistic Commonly referred to as *sigmoid*, this function can be expressed in the form:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

While its usage was historically motivated by how it models neuron firing rates, it has been mostly superseded due to some of its drawbacks. In particular, its saturating behaviour causes the gradient at the asymptotes to quickly approach zero, preventing the network from learning further — a phenomenon known as *vanishing gradient*.

Tanh The *hyperbolic tangent* function is a scaled and shifted version of the logistic function, and can be expressed as:

$$\tanh(x) = 2\sigma(2x) - 1 \tag{3.3}$$

While this does not fully solve the vanishing gradient problem mentioned above, it partially mitigates it by providing stronger dynamics in its derivative. It does however ensure that its output is centered around zero, and is therefore preferred over the standard sigmoid.

Rectified Linear Unit Popularized by [GBB], the *ReLU* has been featured in state-of-the-art convolutional models such as [KSH12] where it improved training convergence time by a factor of 6. It can be expressed as:

$$f(x) = \max(0, x) \tag{3.4}$$

The reasons for its superior performances *may* lie in its lack of saturating regions, as well as in a sparser activation pattern [SWT14] (i.e. for zero-centered data, approximately half of the nodes will not contribute to the following layer). Performance boosts are achieved thanks to its less involved software implementation which does not call for expensive floating-point operations such as exponential.

However, depending on its parameters initialization, a given neuron might land into a region of the search space where the gradient is always zero, thereby halting its learning. This issue, similar to the vanishing gradient, is commonly referred to as *dying ReLU problem* .

Leaky ReLU This function is a variant of ReLU where the gradient is allowed to flow for the negative half of the input, by introducing a small amount of negative slope. It is defined as:

$$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{3.5}$$

Despite addressing the issues highlighted above, its effective benefits are not consistent and are similar to those achieved with a careful selection of other hyper-parameters [Kar].

3.1.3 Training and optimization

Optimization is the process that updates the network weights and biases at each learning iteration, with the purpose of minimizing the cost function. This is done by calculating its gradient (that is, the slope of the cost function across each of the dimensions projected by the network parameters) and then adjusting the weights by an arbitrary small amount (*learning rate*) proportional to the negative of the slope. This process is known as *gradient descent*, and can be expressed as:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta) \tag{3.6}$$

where θ is the vector of trainable parameters (e.g. weights and biases), η is the learning rate, $J(\theta)$ is the cost function, and $\nabla_\theta J(\theta)$ is its gradient with respect to the trainable parameters.

The cost function is calculated on a given set of input data, so the gradient takes into consideration each input sample. However, this may be unfeasible for vast datasets like those required for deep learning. Therefore, this method has

been generalized to operate on a subset of the original data, referred to as *mini-batch*. When the batch consists of a single, randomly picked sample, the process is called *stochastic gradient descent*.

The disadvantages of these two methods consist in having to compute multiple approximations of the gradient, which causes convergence to occur more slowly; however, the noise introduced by the differences in the gradients may cause the optimization to leap out of a suboptimal minimum.

The gradient of the cost function is calculated through a process known as *back-propagation*, which consists in recursively applying the *chain rule* across all the layers of the network.

The chain rule is used to calculate the derivative of composite functions — that is, functions in the form $f(x) = h(g(x)) = (h \circ g)(x)$. According to the chain rule, the derivative of the given expression is given by:

$$\frac{\partial f}{\partial x} = \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x} \tag{3.7}$$

This formula is the foundation of the algorithm, as deep networks are indeed the result of multiple nested operations such as weight multiplication, nonlinear activation, and cost function computation. As illustrated in figure 3.3, it is possible to compute the gradient at a given node, only using the local gradient and recursively applying the chain rule.

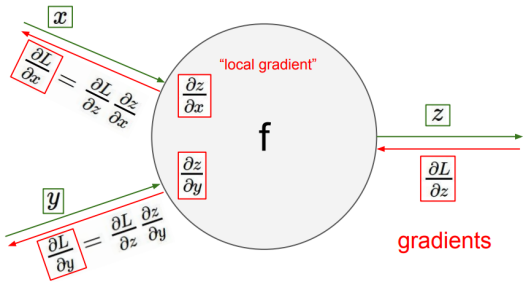


Figure 3.3: back-propagation on a single node [Kar]

Most deep learning libraries implementing back-propagation in their API provide forward- and backward-passes for their neurons and other computational block, allowing gradients to be computed in a modular fashion and without the need for an explicit analytic expression. This allows software implementations to leverage on optimization techniques (such as *memoization*) to quickly retrieve cached computation.

Further details about back-propagation may be found in [Kar; LeC+98; Nie15].

While gradient descent offers a solid base for optimizing a given function, there are a few issues which make it not ideal:

- a small fixed learning rate may cause convergence to occur very slowly, whereas a large value may prevent it entirely by repeatedly overshooting the minimum
- applying the fixed learning rate to all features at each iteration might not be the desirable strategy when the data is sparse and certain neurons do not fire very often
- in non-convex error functions, there is a substantial risk of falling into a suboptimal minimum or a *saddle point*, where the direction of the gradient is particularly ambiguous

To address these and other issues, real-world implementations of deep learning systems resort to more sophisticated optimization algorithms. The following paragraphs offer an overview of the most common ones, in increasing order of complexity. Further details about optimization algorithm can be found in [Rud16].

Momentum Optimal local minima are often surrounded by regions — called *ravines* — where the curvature is much steeper in one dimension than in others. Gradient descent algorithm may end up oscillating across the steepest slope of the ravine instead of reaching a minimum. A way of dampening these oscillations consists in adding a fraction γ of the previous update vector to the current one:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta_{t+1} &= \theta - v_t \end{aligned} \tag{3.8}$$

Nesterov Accelerated Gradient Adding momentum to the update vector may cause overshooting, as the algorithm cannot preemptively slow down when approaching an optimal minimum. NAG takes the concept presented in Momentum further, and calculates the gradient with respect to the estimated updated parameters, according to:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta_{t+1} &= \theta - v_t \end{aligned} \tag{3.9}$$

Adagrad This optimization algorithm scales the learning rate of each parameter based on the accumulated gradient along that parameter dimension up to the current iteration:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t \tag{3.10}$$

where g_t is short for $\nabla_\theta J(\theta_t)$ — i.e. the gradient vector at time step t, G_t is a diagonal matrix where each element is the sum of the squares of the gradients with respect to each parameter up to time step t, and ϵ is an arbitrary small number used to avoid division by zero.

Adadelta/RMSprop While Adagrad is effective at adapting to the characteristics of each dimension, the accumulation of past gradients in the term G_t in the form of squared (therefore positive) values causes the learning rate to become infinitesimally small, thereby halting learning. Adadelta and RMSprop seek to solve this issue by replacing the aforementioned term with a decaying running average of the squared gradient, as per:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{3.11}$$

where γ is a fractional term. Once G_t is replaced with $E[g^2]_t$, the denominator can be seen as the root mean squared error of the gradient. The update rule can therefore be rewritten as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\text{RMS}(g)_t} g_t \tag{3.12}$$

Adadelta differs from RMSprop in that it replaces the learning rate η with the RMS of the previous parameter update.

Adam This state-of-the-art algorithm combines adaptive learning rate and momentum features by keeping track of the first and second moments (mean and uncentered variance respectively) of the gradients:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{3.13}$$

where β_1 and β_2 are smoothing factors for the decaying average. The update rule is therefore formulated as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \tag{3.14}$$

3.1.4 Convolutional Neural Networks

Convolutional Neural Networks are a family of neural networks that was originally designed with the purpose of modeling image processing in the visual cortex of animals [HW62; Fuk88].
They were first used for a classification problem — hand-written digits [Lec+98], before being extended to more general pattern recognition tasks. CNNs have seen their performance improve and their architectures deepen since then [Alo+18, p. 15], for instance with AlexNet [KSH12] and VGG [SZ14]. CNNs are mostly applied to computer vision and image processing tasks, including image analysis (segmenting) and enhancing (denoising) — see [Cnn] for a comprehensive overview of CNNs applications; however, one can find some examples of CNNs designed for solving Graph problems [NAK16], for speech enhancement [Hou+17] and for medical imaging [Zha+17].

CNNs are feed-forward neural networks; they are variations of multilayer perceptrons presented in 3.1.1 that are meant to use minimal amounts of pre-processing. CNNs were designed in order to reduce the number of parameters considered and to make the whole learning process more efficient in assuming some properties of the input data. CNNs use convolution as processing tool, which comes down to successively filtering the data along one or more dimensions.

In such models, the filters — or *kernels* — used are multidimensional with their dimensions being kept small most of the time. The dimensionality of the whole data defines its volume, and depends on the data representation that is chosen (see 3.2); and so does the number of dimensions of these kernels, as well as the number of remaining dimensions — called *channels* or *depth*.

As an example, images are encoded spatially in two dimensions, and may have a third non-spatial dimension given by color encoding. The kernels are therefore applied to the whole volume of each layer's input by iteratively sliding them on the two spatial dimensions and considering the full depth of the data at each position. The purpose of such an operation — a convolution— is to output multidimensional activation maps, based on the structural elements of the original data acquired during the training stage.

The output of a convolutional layer is the stack of as many activation maps as there are filters. The size and the content of these activation maps depend on several user-defined hyperparameters such as padding, strides, and dilation. These are means to further encode the analogy with animal's vision by controlling local connections.

Padding Padding refers to the ways of considering boundary conditions on the data. For an image as example, it comes down to balance the weight of pixels at the border relatively to these in the middle of the image in adding one or more extra rows of pixels all around the picture and in filling them .

Stride The stride defines the spatial sliding step of the kernels. A stride of 1 makes the filters move one unit at a time; a stride of 2 makes them skip one unit each time it slides in the direction to which the stride is applied. If needed, different strides can be applied to the two spatial directions. A big stride will reduce the size of the activation maps accordingly.

Dilation Some recent development [YK15] introduced a new hyperparameter to the convolutional layers called *dilation*, which controls the space between the cells that are used during the computation of the new weights. In other words, it spatially interconnects input and weights so as to aggregate multi-scale contextual information.

Mathematically speaking, a dilation is defined as follows for a given layer:

$$w * x = \sum_{s+Lt} w(t)x(s) \tag{3.15}$$

where x is the output of the nodes in the previous layer, w are the corresponding weights, and L is the dilation factor.

As a result, it makes the kernels bigger and extends the receptive field while keeping the spatial resolution — unlike the stride — and without adding more trainable parameters.

Generally, several dilation rates are jointly used in order to merge information at different scales at the same time.

Receptive field The notion of receptive field $\mathcal{R}$ for convolutional neural networks is defined as the region in the input space that a particular unit in a network is looking at and is affected by [Luo+17]. Controlling this field is vital so as to provide the network units with the proper amount of information they need.

Padding, dilation, stride, and kernel size impact the receptive field of a CNN, whose general formula for a layer with input size i, with respect to its previous layer, is:

$$\mathcal{R} = 1 + \frac{i + 2p - k - (k-1)(d-1)}{s} \tag{3.16}$$

where p is the padding, k is the size of the filters, s the stride and d the dilation if any. This parameter should be carefully tuned so as to cover relevant regions of the input data.

CNNs have proven to perform admirably in many cases, especially for image processing tasks — see introduction of this section. Nonetheless, they show several weaknesses:

Data type While CNNs are known for processing efficiently images and videos; special considerations must be taken into account when processing other types of data such as audio signals, even though these can also be represented as two-dimensional matrices (see 3.2.1). However, while images are invariant along both spatial dimensions, audio signal representations such as spectrograms that display frequency in function of time are not invariant along their frequency axis. Therefore, an identical processing of these two types of data is not possible.

Time consideration Another major difference is the high dependency of speech signal upon time: there is a logic when one speaks, and words usually succeed so as to make sense. This property is not encoded in CNNs, that take inputs of fixed size without considering the past — or history, and generate fixed-size outputs

Learning method CNNs require vast amounts of training data in order to achieve good satisfactory performances. Moreover, they are often trained using supervised learning methods, thereby requiring labeled data which are not always viable.

In order to overcome some of these limitations and be able to process efficiently other types of data such as text and speech signals, recurrent neural networks (RNN) have been introduced.

3.1.5 Recurrent Neural Networks

Recurrent Neural Networks are a class of deep neural networks where the network outputs and/or internal representations are fed back along with its input. This allows the network to dynamically model the temporal relationships occurring in certain types of data, such as natural language, audio, and any kind of time-series data. RNNs are particularly suitable for modeling sequential data, and have found applications in translation, image captioning, sentiment analysis, and speech and handwriting recognition [MG08; ZWL18; Sow+18].

The most attributed origin of RNNs stems from the work of cognitive scientists Rumelhart et al. [RHW86] in 1986, who among other things popularized backpropagation on deep neural networks, and discovered how such networks could learn useful internal representation of their input data. The theoretical notions presented in the rest of this section are further expanded in the book *Deep Learning* by Goodfellow et al. [GBC16] as well as in the notes from Stanford's *CS230* course in deep learning [Ngb; AA18].

As introduced above, RNNs compute each new output based on their previous output and an internal state, which is commonly referred to as *memory*. In traditional RNNs, internal states for a given time step t are calculated as follows:

$$h_t = \phi_1(W_h h_{t-1} + W_x x_t + b_h) \tag{3.17}$$

where W_h and W_x are matrices of trainable coefficients applied to the previous hidden state h_{t-1} and current input x_t respectively, b_h is a matrix of trainable biases, and $\phi_1()$ is the recurrent activation function. These matrices represent the importance of a given internal state and previous output features for the computation of the current output, similar to probability matrices in Markov chain models. In RNNs, the computational nodes performing this operation may also be called *units*.

Once the internal state has been computed, the output at time step t may be formulated as:

$$y_t = \phi_2(W_y h_t + b_y) \tag{3.18}$$

where W_y and b_h are matrices of weights and biases respectively, and $\phi_2()$ is another activation function.

The operations performed by an RNN can be *unfolded* by repeatedly applying its definition, such that:

$$h_t = f(h_{t-1}, x_t) = f(f(h_{t-2}, x_{t-1}), x_t) = f(f(f(h_{t-3}, x_{t-2}), x_{t-1}), x_t) = \dots \quad (3.19)$$

where $f(h, x)$ summarizes equation 3.17. The equation clearly shows the recursive nature of the computation; however, for a finite number of time steps, the expression will eventually be dependant on the input x_{t-N} only. This makes it possible to represent it using a *directed acyclic graph*, which is how most deep learning frameworks encode computations.

Similarly to how CNNs share parameters across spatial dimensions (thereby exhibiting shift-invariant behavior), recurrent networks share parameters across time steps, hence making the model resilient to different sequence lengths and orders of appearance. This characteristic is particularly desirable for processing data that is not constrained in size, and where information can occur at different positions, such as natural language.

Networks based on recurrent units may be classified based on the nature of the input they take and output they produce. In particular, with T_x and T_y the lengths of the input and output sequences respectively, there can be:

- **one-to-one** $T_x = 1,\ T_y = 1$, no recurrence (traditional ANN)
- **one-to-many** $T_x = 1,\ T_y > 1$, a sequence of data is generated from a single input (e.g. automatic image captioning, music generation from random seed)
- **many-to-one** $T_x > 1,\ T_y = 1$, a single output is generated for a sequence of data composed of multiple time steps (e.g. sentiment analysis)
- **synced many-to-many** $T_x = T_y$, an output is generated from each input time step (e.g. video frame tagging, name entity recognition)
- **many-to-many** $T_x \neq T_y$, inputs and outputs are sequences of arbitrary lengths (e.g. machine translation, chat-bots)

When using RNNs to model *long-term dependencies* — that is, patterns encompassing numerous time steps — gradients must be propagated over many stages, causing them to explode or vanish. This can be seen as the result of applying multiplications and nonlinear activations repeatedly; while there exist regions in the parameter space where the gradients are stable, these are typically not the ones that will allow the network to learn these long-term relationships, as short-term ones will have exponentially larger magnitude which would hinder learning.

While multiple ways to overcome this issue have been developed, the most effective and widely adopted one comes in the form of *gated units*. This technique consists in providing alternative paths through time for gradients to flow through, without causing it to vanish or explode. Its two major variants are illustrated in Figure 3.4 and presented below.

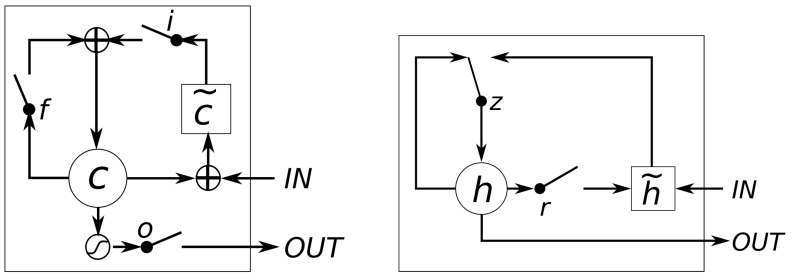


Figure 3.4: major types of gated units: LSTM and GRU [Chu+14]

3.1.5.1 Long Short-Term Memory

LSTM networks were proposed in 1997 by Hochreiter et al. [HS97], and have recently become the dominant class of RNNs. They address the vanishing gradient problem by introducing a series of paths, called *gates*, where gradients from previous time steps can flow unmitigated. An LSTM network can therefore learn when to accumulate, propagate, or flush data, in order to maintain rich gradients and model long-term dependencies.

In order for this to happen, the computational units described by equation 3.17 are replaced by more complex *cells* featuring internal loops, hidden as well as internal states, and analog gates controlling the flow of information. Such gates can be expressed in the form:

$$\Gamma = \sigma(Wx_t + Uh_{t-1} + b) \tag{3.20}$$

where W, U, and b are trainable coefficients specific to the given gate, and σ is the logistic activation function. It is important to note how the state of the gate is based on the current input, as well as the hidden state (which doubles up as output) at previous time step, of the cell.

Hidden states acting as outputs may seem counter-intuitive, but it is easily explained when considering how different categories of RNNs (one-to-many, many-to-many, and so forth) expose different amounts of hidden states depending on the length of their output. For example, in a many-to-one network only the last hidden state is used as output.

Each LSTM cell typically comprises the following gates:

- **Input gate** Γ_i controls how much content is added to the memory cell
- **Output gate** Γ_o controls how much of the memory cell content is exposed
- **Forget gate** Γ_f controls how much of the existing memory cell content is discarded

Given these building blocks, the output of the cell at time step t is given by:

$$h_t = \Gamma_o \odot c_t \tag{3.21}$$

where $\odot$ represents element-wise vector multiplication and c_t is the internal state (memory) of the cell for the current time step. In turn, this latter value depends on how much new content $\tilde{c}_t$ and content from the internal state at the previous time step c_{t-1} is allowed by their respective gates; this can be expressed as:

$$c_t = \Gamma_i \odot \tilde{c}_t + \Gamma_f \odot c_{t-1} \tag{3.22}$$

where the dependency on the internal state at the previous time step denotes the aforementioned internal loop.

Finally, the new content $\tilde{c}_t$ is defined as:

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{3.23}$$

where W_c, U_c, and b_c, are trainable coefficients.

Given the vast number of trainable parameters in each cell (multiple weights and biases for each gate, plus weights and biases for the new internal state), LSTM networks require substantial amounts of processing power and make troubleshooting particularly cumbersome. Therefore, a recently proposed, more streamlined variant is presented below.

3.1.5.2 Gated Recurrent Units

GRUs, introduced in 2014 by Cho et al. [Cho+14], are a simplified variant of LSTM networks, comprising only two gates and a no separate internal state (cell memory). In fact, for GRU cells, $c_t = h_t$, meaning that internal state is the same as the cell hidden state or output, as opposed to LSTM which feature a specialized output gate. The two gates Γ_u and Γ_r are called *update* and *reset*: the former combines the functionalities of the input and forget gates found in LSTMs, whereas the latter is applied to the previous hidden state, in order to modulate the amount of previous information to consider when computing the current state.

With these notions in mind, the characterizing equations for GRU cells can be written as:

$$c_t = \Gamma_u \odot \tilde{c}_t + (1 - \Gamma_u) \odot c_{t-1} \tag{3.24}$$

for the current hidden state (or output), and

$$\tilde{c}_t = \tanh(W_c x_t + U_c(\Gamma_r \odot c_{t-1}) + b_c) \tag{3.25}$$

for the *candidate* state. The cell output is therefore computed as a combination of its candidate state and its previous hidden state.

GRUs have been shown to have similar performances as LSTMs while benefitting from their increased simplicity [Chu+14].

3.1.6 Autoencoders

Autoencoders (AEs) are a family of neural network models that aim to learn compressed latent variables of high-dimensional data in an unsupervised manner and to reconstruct original data from this latent representation. They were popularized by Hinton et al. in [HS06].

They can be compared to PCA and NMF models mentioned in 2.1, as they reduce the number of features dimensions. However, unlike models that aim to classify the data and are only concerned with the compression of input, AEs output data of the same dimensionality as the input for a purpose of reconstruction or transformation of this data.

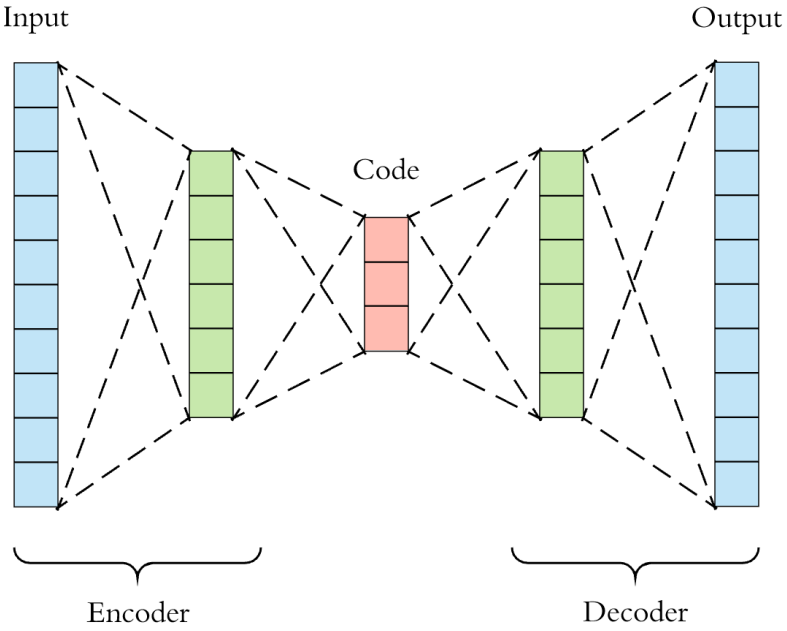


Figure 3.5: Auto-Encoder

An AE is a model made of two networks as shown on Figure 3.5. Firstly, an *encoder* compresses the data and outputs a downsized version of the input — the

code — in the latent space — or bottleneck. Secondly, a *decoder* reconstructs data from this latent representation. A solid intermediate representation is paramount for a good decompression process through the decoder.

Encoder and decoder are usually built symmetrically. The simplest AE has an MLP-like structure. This structure can however be customized according to the task or type of data, using any of the primitives mentioned in the previous sections, such as convolutional layers, or by adjusting the number of hidden layers.

Mathematically speaking, the model contains an encoder function $f_\phi()$ and a decoder function $g_\theta()$ parameterized by ϕ and θ respectively. For a given input x, the low-dimensional representation $z = f_\phi(x)$ is outputted in the bottleneck layer where it is reused as input by the decoder to reconstruct the input $\hat{x}$ such as $\hat{x} = g_\theta(z)$.

The set of parameters ϕ and θ are jointly learned by the network so that $g \circ f$ operates as an identity function. This is performed by computing the loss function, e.g. *mean squared error*, on x and $\hat{x}$.

Autoencoders may be used to denoise data if the network is fed with processed noisy data x and the loss function is computed on clean data x_s and predicted $\hat{x}_s$. Using AEs for speech denoising has been widely experimented, with state-of-the-art examples of it in section 2.3.

Other applications include improving the robustness of the network against overfitting (sparse, contractive AE) or manipulating the latent data z in order to produce variations of the input data (variational AE) [HZG17; Kuz+].

3.1.7 Batch Normalization

As seen earlier, data is fed into the model by subsets of the whole data, called *mini-batches* (or simply batches on most machine learning frameworks), and during training, weights and biases are updated on an individual batch basis. The computation performed on these batches are therefore propagated from layer to layer and amplified accordingly, which makes the later layers highly dependent on the weights of the earlier layers and may cause weights range instability. This provokes two unwanted phenomena, mentioned in [IS15; Nga]: firstly, it slows down the convergence of the network since each layer must readjust its weights to the varying incoming distribution — problem known as *internal covariate shift*; secondly, it pushes data into the saturating region of the activation function, worsening the *vanishing gradient* problem.

Batch normalization is a method introduced by Ioffe and Szegedy in [IS15]. It ensures the homogeneity of data distribution in every batch by keeping means and variance close to expected statistics, for instance 0 and 1 respectively, according to

the following formula for each activation input x_i:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B + \epsilon} \tag{3.26}$$

where μ_B and σ_B are the batch mean and standard deviation, and ϵ ensures numerical stability.

Simply normalizing each input of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the function. Therefore, $\hat{x}_i$ is also linearly transformed to ensure that the normalizing effect is maintained despite the changes in the network during backpropagation, and that the whole transformation can represent the identity function:

$$y_i = \lambda_i * \hat{x}_i + \beta_i \tag{3.27}$$

where λ_i and β_i are parameters coding mean and standard deviation that are learned by the network. If the network manages to estimate perfectly λ_i and β_i for each layer i, then it reconstructs an exact version of the input $\hat{x}_i$.

Batch normalization improves convergence speed [LeC+98] and mitigates the vanishing gradient problem by ensuring that the values within the network don't reach a saturation region. It also grants the user with more liberty in initializing the parameters and setting the range for learning rates — larger than without batch normalization. Moreover, it has been shown to provide regularization effect as side effect.

3.1.8 Residual networks

The vanishing/exploding gradient being an issue at stake in CNNs during learning, a solution has been implemented quite recently, that addresses the connectivity of the layers of the network [He+15] and thus its whole architecture. The thus transformed network is named Residual Neural Network (ResNet), for a reason that is presented here below.

The vanishing/exploding gradient is a problem whose probability to occur increases as a regular CNN deepens.One measure to deal with it consists in putting in contact some layers that are not immediately successive, through an additional "shortcut-connection".

Mathematically speaking, for a given set of layers — called *system*, let call $\mathcal{F}$ its original mapping function, i.e. without shortcut-connection, such as an input x outputs from the system as $\mathcal{F}(x)$; let $\mathcal{H}$ be the desired mapping function in our newly connected system, such as an input x outputs the system as $\mathcal{H}(x)$. In this newly designed system, an additional connection modifies the system's internal processing, in transforming the version of x that doesn't feed the immediate next

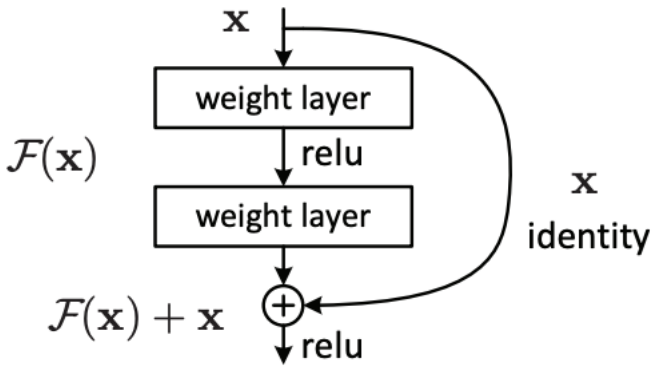


Figure 3.6: residual learning: a building block with residual function being the identity function [He+15]

layer with a chosen function C; the new general mapping function is now $\mathcal{H}$ such as $\mathcal{H}(x) = \mathcal{F}(x) + C(x)$ when $+$ stands for element-wise addition. This can be reformulated into $\mathcal{F}(x) = \mathcal{H}(x) - C(x)$.

This is the *residue*, which gives its name to this class of networks.

C is generally chosen to be the identity function, as in Figure 3.6, so layers have to learn modifications to the identity mapping rather than the entire transformation. Note that in case of the dimensionality of the data increases when processed with $\mathcal{F}$, $C(x)$ is zero-padded to match the new data size.

The underlying assumption is that a network should be able to learn $\mathcal{H}$ as good as $\mathcal{F}$ — at least asymptotically, but may have more ease to learn residual mappings than straight ones [He+15]. If this hypothesis is verified, which seems to be the case since it has repeatedly proved to benefit very deep networks [Bai18], then ResNets should be able to attenuate the vanishing gradient phenomenon.

3.1.9 Dropout

As mentioned earlier, overfitting is a well-known issue in deep learning, and some methods have been developed for preventing it from occurring, among which *dropout* which was originally introduced by Hinton et al. in [Hin+12] and was further explained in [Sri+14a].

During the training process, dropout shuts down a portion of all nodes and associated activations at each iteration for every hidden layer and training sample, with a probability p. To put it in other words, dropout randomly changes more or less subtlety the internal connectivity of the network at each step of the training. It thus introduces noise into the data and makes the network learn more robust features, preventing it from relying on certain combination of features which are

not representative of the data.

This method alters the convergence speed of the network since it increases the number of iterations required to converge. However, it speeds up the training time for each epoch due to less computation being performed overall.

3.2 Data Representation

Human's auditory system is constantly stimulated in real life, either by natural or virtual sounds. In order to be able to analyze audio, researchers have developed various representations of sound since the mid 1900s. These representations, respectively Time representation, Discrete Fourier Transform and Short-Time Fourier Transform are presented in the following subsections.

A sound wave is a continuous longitudinal compression wave that propagates in a material medium, causing tiny pressure variations in the environment that depend on both space and time. It is characterized by its length, amplitude, frequential content and spectrum, which correlate perceptively to duration, loudness, pitch and timbre.

Sound waves can be encoded in many ways to make it available for later usage. Nonetheless, one can not afford to store the signal as it is with all its subtleties, for the sake of storing space. Therefore, an audio signal is encoded in a compressed *digital* format, stored on support, then uncompressed through a decoding stage when needed. This is the chain of reproduction of sound, whose encoding stage is further explained here under.

3.2.1 Time Domain Representation

Many encoding methods have been deployed since the first recording in 1877 by T. Edison in order to store audio. During the encoding stage, the audio signal is uniformly sampled in time at a frequency called *sample rate* usually set so as to encode as high frequency as the humans can hear, i.e. up to 20 kHz. In taking into consideration Nyquist-Shannon's law [Smi03] that says than frequencies up to half the sample rate can be encoded, we arrive close to the common value that is generally used and that is 44 100 Hz.

The signal is then quantized, meaning that every sample is attributed a value into a discrete range determined by the capacity of the support.

Let x be this digitally encoded signal. x is now readable by some software. This representation is convenient to detect intensity peaks and onsets. However, it does not convey much information about the frequency content. Therefore, spectral analysis methods were invented.

3.2.2 Discrete Fourier Transform

The need of a complementary tool for analysing signals has arisen surprinsingly early in history, with Gauss and Fourier in the beginning of the XIXth century. However spectrum analysis and more specifically the Discrete Fourier Transform have been theorized in 1965 only by Cooley and Tuckey.

DFT is an operation that projects the time-domain signal into a sparse complex space, where frequencies form an orthogonal basis in the destination space. The DFT X of the signal x for frequency k is given by the following formula:

$$X(k) = \sum_{m=0}^{N-1} x(m) e^{-2i\pi m.\lambda_k} \tag{3.28}$$

where $\lambda_k = \frac{k}{N}$ is the normalised frequency after discretization of frequencies. Note that an optimized algorithm for the computation of DFT is called Fast Fourier Transform (FFT).

DFT thus makes it possible to extract frequency-related features from the signal under analysis. Whereas time-domain analysis hardly allows the user to extract any frequency in a complex signal, DFT provides an easy way to have access to every frequency up to $sr/2$. In order to observe the DFT, its absolute value is plotted against unnormalized frequencies.

However, caution must be taken when interpreting the DFT, for this representation is not error-free. As a matter of fact, DFT can not be applied to an infinite signal in practice. Therefore, only a segment of finite length and implicitly framed by a window w is processed at a time. This implicit process is a convolution of w and x in the time-domain resulting in their multiplication in the frequency-domain [Smi03]. It thus impacts the precision and the resolution of the signal's spectral representation.

First, the length of the fragment directly affects the precision of the frequencies detected. The longer the time frame, the more matter it provides for the analysis and the more precise the frequencies estimation *for a stationary signal*. However, this last condition is crucial since DFT is averaging frequencies on the whole frame it is fed with: consequently, a signal having a fast-evolving spectrum and being analysed with a large window will result in some mixture in no way interpretable. The window size is therefore chosen heuristically by the user to provide enough content for the analysis without compromise the analysis validity. Another issue arises from windowing. As we have seen before, windowing is unavoidable and must be applied carefully; even worse is the fact that it comes at a price, in altering the frequential content of the signal's DFT. This is due to the edges of non-infinite signals, that are transformed through DFT at the same time as the signal itself. Below on Figure 3.7 is the DFT of a rectangular window in contrast with the DFT of

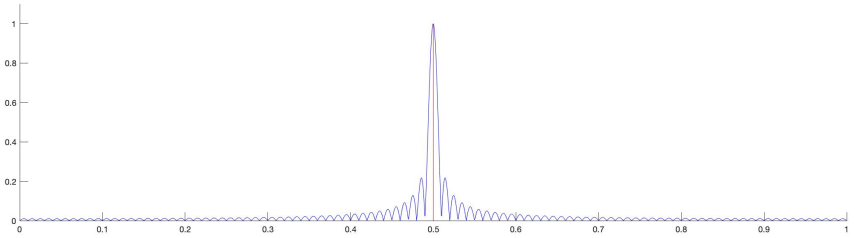


Figure 3.7: DFT of an "infinite window" (red) against DFT of a finite rectangular window (blue)

an infinite window - very paradoxical appellation! We can observe two phenomena on the window DFT representation, commonly named *spectral leaks* and *spectral spread*.

Spectral spread this relates to the width of the peak at $f0$ in the DFT representation of the window, where $f0$ is the resonant frequency of the window defined by its size. In comparison, the resonant peak of an infinite window is centered on 0 — or wherever, and is infinitely thin and high. Hence, when multiplied with the signal's DFT, a finite window's DFT widens the signal's *true* peaks which impacts the frequential resolution of the analysis.

Spectral leaking this phenomenon corresponds to the non-zero values of the DFT wherever $f \neq f0$. When multiplied with the signal's DFT, it results into the presence of secondary lobes on each side of a frequency peak, which introduces frequency noise.

These two weaknesses can be controlled to some extent by manipulating the type of window that is used. Nonetheless, it doesn't provide any space for a dynamic representation of s. Therefore a third signal's representation was developed in the half XX^{th}, that combines advantages of both time-domain and frequency-domain methods introduced here above: the spectrogram.

3.2.3 Short-Time Fourier Transform

In case one wants to analyse the spectrum of a 5-minute long fast-changing signal, it is highly recommended *not* to rely on DFT since it averages frequencies on the frame it is given to analyse. So as to solve this issue, one can apply DFTs to successive frames of the signal.
Furthermore, it is the user's interest to define an *overlap* in order to ensure continuity of the analysis. In this way, some frequential information is coupled with a range of temporal landmarks.
This process is called the Short-Time Fourier Transform and is an improved version

of the DFT introduced by Dennis Gabor in 1946. The STFT is calculated for every Time-Frequency (TF) bin as follows:

$$STFT_{h,s}[n,k] \triangleq \sum_{m=0}^{N-1} s[m].h[m-n].e^{-2i\pi m.\lambda_k} \tag{3.29}$$

where h is the window considered , n and k index a T-F bin in time and frequency respectively.

The data thus obtained is complex once again; in order to observe it, one needs to process it a bit further. Various possibilities exist depending on the exponent α used on the absolute value of the STFT:

$$S_{h,s}[n,k] \triangleq |STFT_{h,s}[n,k]|^{\alpha} \tag{3.30}$$

A magnitude spectrogram and a power spectrogram are obtained for $\alpha = 1$ for $\alpha = 2$ respectively. Other representations use customized coefficients, such as $\alpha = 1/6$.

This representation is very suitable for analysis for it gives a comprehensive overview of the signal under analysis: the spectrum is easily readable over time in a fashion that reveals the T-F bins the most energetic. This representation fits signals fast-evolving in intensity and/or spectrum — this second feature being nonetheless constrained by the size of the window employed.

However, STFT is not perfect: it admittedly inherits the advantages of time- and frequency-representations, but also some of their defaults. For instance, the better resolution in frequency, the worse the resolution in time, according to the inequality of Heisenberg-Gabor that states that a signal can't have small support in time and frequency simultaneously [Bus07]. Therefore, similarly to the DFT, some compromise between time and frequency resolutions must be researched upon so as to retain the properties of the signal.
In addition, it introduces interferences between bins, in frequency and/or time depending on the dimension of the bins and on the window used.
Finally, the STFT doesn't retain the energetic properties of the signal: one can not easily get back its energy per frequency band or per time frame. It means that the reconstruction of the signal from its STFT is more complex than with the previous methods.

Some other data representations can be expressed as functions of the spectrogram, such as the mel-scale and the dB-scale that are presented hereunder.

The dB-scale is based on the logarithmic function that is known to model fairly the human's audition. Particularly, this scale conveys the well-known phenomenon of different perceived distance between two pairs of octave-spaced sounds depending on the average pitch of each pair.

The *log-spectrogram* is simply a magnitude spectrogram which is passed into the logarithm function. As a result, it spaces low frequencies and compresses high frequencies. It is used for denoising in [KZ15].

The mel-scale is another scale based on the results on an experiment about pitch perception conducted in the 30s by Stevens et al. [Ste37]. Even though this scale has been questioned due to the controversial methodology of the experiment [Gre97], it is still very often used today for modeling human sound perception [Lu+; ZL18]. The name *mel* comes from the word *melody* to indicate that the scale is based on pitch comparison. The main concept of this scale is for a given sound as reference, another sound having double pitch should be perceived twice as high as the reference. It is done the following way: for low-pitched sounds, mels equal Hz; above $f = 500$Hz, mels follow a human-made trajectory that increases slower than Hz.

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency [Mer76; PCCB97]. The resulting coefficients, referred to as MFCC, or some derivatives are oftentimes used for denoising today [Far18; FZG14].

3.3 Data processing

The data resulting from the application of the STFT function presents some properties that are not ideal, when feeding it into a neural network model; this section features further processing techniques aimed at mitigating this issue.

Firstly, the vast majority of neural network models operate with real-valued data, whereas the time-frequency representation is complex-valued. This is trivially solved by taking the absolute value (also called *magnitude* or *modulus*) of the spectrograms. For a speech excerpt, it may look like on 3.8:

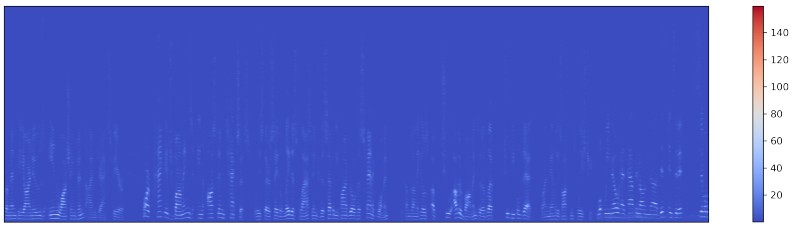


Figure 3.8: a magnitude spectrogram

While being a step in the right direction, the magnitude spectrum of a signal still manifests the aforementioned properties. The following sections will outline some of these characteristics along with ways to address them.

3.3.1 Dynamic Range Processing

The dynamic range of a signal is the ratio between its loudest and its quietest parts. If a given channel only supports a limited dynamic range, it is important to match that of the signal accordingly. The two most common operations on the dynamic range are *compression* and *expansion*.

As mentioned before, a deep network can approximate any continuous function over a compact interval. If this interval is likened to the dynamic range of a channel, the need for further corrective manipulation quickly arises [KZ15].

To fully understand why this is the case, it is important to consider the histogram of a magnitude spectrogram (i.e. the one in the example above). Figure 3.9 highlights two distinct properties: a substantial positive skewness and the lack of a limited interval. This makes sense intuitively considering how sparse the spectrogram in figure 3.8 tends to appear. In particular, the magnitude spectrogram data seems to closely follow a *log-normal distribution*.

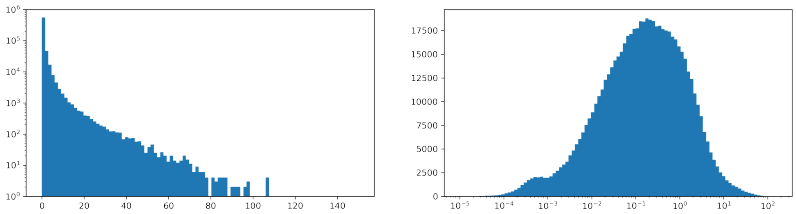


Figure 3.9: histogram of a magnitude spectrogram, with linear x-axis and log y-axis, and log x-axis and linear y-axis respectively

There exist multiple ways of addressing this issue. They can be divided into two classes: *logarithmic* and *power* transformations.

The former is widely adopted [KZ15; WC17; Tu+18], and can be generalized as:

$$f(x) = \alpha \log_\beta(x) \tag{3.31}$$

where α and β are scaling factors.

For $\alpha = 20$ and $\beta = 10$, this equation describes the data in terms of power ratio in decibels (dB) (given a unit reference signal). When applied to the example spectrogram, the result appears as on figure 3.10.

The observable increase in dynamic range is certainly valuable, and the skewness in the data distribution has been compensated.

When implemented using floating-point arithmetic, the logarithm operator may cause numerical instability, especially when dealing with very small values. Introducing a lower and upper threshold can effectively mitigate the issue — at the cost of data integrity [PLS16]. This is clearly noticeable in the histogram (Figure 3.10), where part of the distribution has been clipped, causing the smallest possible sample to be over-represented.

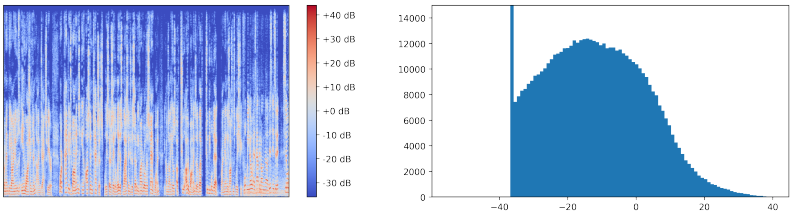


Figure 3.10: power spectrogram and its respective histogram, with linear x and y axes

A more sophisticated variant is the so-called *μ-law*, which is defined as:

$$f(x) = \text{sign}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \quad \text{for} -1 < x < 1 \tag{3.32}$$

where μ is typically 255. This has been used historically on telephony lines to optimize quantization error on time-domain signals, and increase SNR at lower bit rates. However, since this algorithm is only defined between 1 and -1, a more general approach must be taken.

Power transformations are also common [Wen+14], and can be generalized with the following formula:

$$f(x) = x^{\alpha} \tag{3.33}$$

where α is a factor determining the amount of compensation. Negative skewness is treated with $\alpha > 1$ whereas positive skewness is treated with $0 < \alpha < 1$.

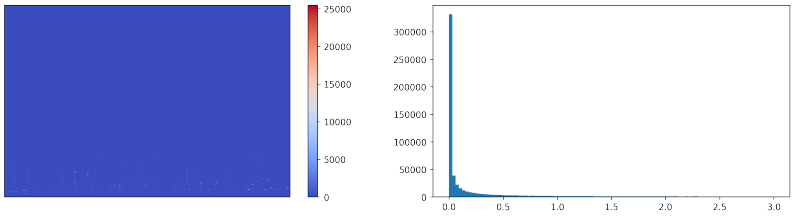
Power transformations are ubiquitous in the field of psychophysics, where they are used to describe the relationship between a stimulus and its perceived magnitude. Typical values of α are 1 for magnitude spectrum, 2 for power spectrum, and 0.667 for *auditory spectrum*, which is a rough model of perceptual loudness [Wen+14].

Figure 3.11 shows these former two along with 0.167, which has been empirically chosen for its even skewness.

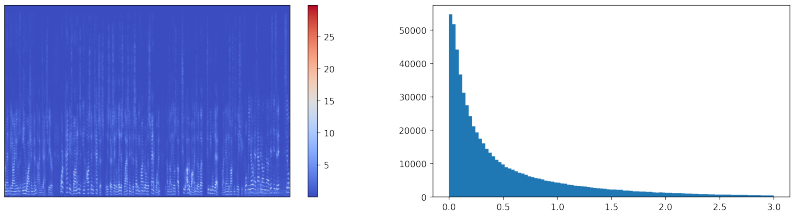
3.3.2 Real/Imaginary spectrogram

A common scenario when using real-valued spectrograms as prediction target is that the phase information has to be estimated. The phase information from the noisy signal is often used. However, it may be desirable to have the network encode and generate the said phase data [Fu+17].

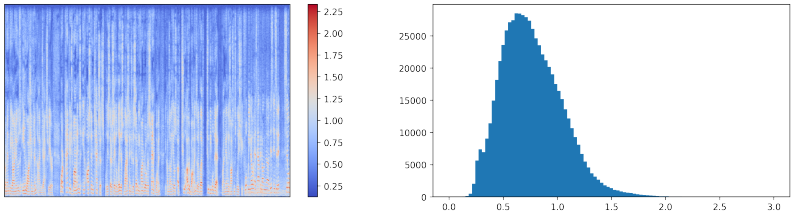
Since neural network are operated with real values in the vast majority of cases, one possible way of encoding complex values is by storing their real and imaginary parts in two separate indexes along a new, extra dimension. An input spectrogram of shape (F, T) where F is the number of frequency bins and T is the number of



(a) $\alpha = 2$



(b) $\alpha = 0.667$



(c) $\alpha = 0.167$

Figure 3.11: spectrograms and histograms for different values of α

time frames is converted into a real/imaginary spectrogram of shape $(F, T, 2)$, with the extra dimension being the number of channels such that:

$$S_{\text{Re/Im}} = \begin{cases} \text{Re}(S) & \text{for first channel} \\ \text{Im}(S) & \text{for second channel} \end{cases} \tag{3.34}$$

The higher dimensionality of this encoding fits the requirements of convolutional neural networks, which are meant to operate with image data that is inherently 3-dimensional (height, width, color channel).

While this method handles phase reconstruction, it does so by obfuscating other useful characteristics of the spectrogram such as its magnitude, which would be otherwise derived using the Pythagorean theorem. The resulting spectrogram is therefore less intuitive for humans than e.g. a power spectrogram in dB, although it is debatable whether this has any impact on artificial neural networks.

A significant drawback of this data representation is that the dynamic range compression techniques presented above may distort the vectors represented by the underlying complex numbers.

3.4 Evaluation

While training deep models do not require much manual intervention, it is more difficult to define effective models since they rely on the tuning of numerous training parameters. Furthermore, much like traditional denoising techniques, a network may only work satisfactorily for a given set of condition. Evaluation thus aims at quantitatively asses the performance of a trained model, in order to improve its behavior.

Within the scope of this project we use metrics that are largely adopted in the literature [Kol18; Zha+18; Fak+18; Fu+17; PL16], and that are adopted to the process of speech denoising, i.e. correlated with speech intelligibility (SI) and perceived speech quality (SQ). These concepts are not equivalent: while SI measures *what* the message is — how understandable it is, SQ assesses *how* it is said; the latter is highly subjective [Loi11] and can be seen as a combination of factors such as naturalness, loudness, listening effort, as well as intelligibility itself [GK08]. Therefore, it is worth noting that commonly-adopted speech quality algorithms provide a mere approximation.

These metrics are presented below. Other evaluation metrics such as Dau [DPK96] and segmented SNR [Fu+17] exist, but are less commonly employed in the literature. Speech recognition metrics such as *word error rate* (WER) are more widespread [Tu+18; Zha+18; Fak+18] and particularly useful for determining the effectiveness of a given denoising algorithm in combination with a speech recognition system, but require one such system as well as a transcript of the source speech as reference.

3.4.1 Signal-to-Noise Ratio

Signal-to-Noise Ratio (SNR) is a measure that quantifies the power ratio of the signal to the noise. It is usually expressed in dB so as to fit the human perception of loudness which is roughly logarithmic.

$$\text{SNR} = 10 \log_{10} \left(\frac{P_{\hat{x}}}{P_{x_n}} \right) \tag{3.35}$$

where $\hat{x}$ is the estimated speech signal and x_n the known noise, and P the notation for power quantity.

This metric can be calculated in a variety of ways, such as over the entire signal, for single utterances, or on T-F spectrogram units. However, it does not account for the actual content of the estimated signal, i.e. its similarity to the original clean speech. The following section describes more sophisticated metrics addressing this shortcoming.

3.4.2 Source Evaluation

In the case of a mixture of signals (such as speech plus environmental noises), one can evaluate the predominance off a component against the others. In particular, *Blind Source Evaluation* metrics are widely adopted in source separation tasks, but can also be computed for speech denoising applications by considering the noisy component as a second source.

These metrics have been introduced by Vincent et al. [VGF06] and are used in the evaluation of speech enhancement systems by [Fak+18; Zhe+18; Kol18; GP17].

First, the signal must be arbitrarily divided into constitutive elements that are supposed to be independent and known.
Let $\hat{x}$ be an estimation of the mixed signal x.
As in [VGF06], we set the hypothesis that $\hat{x}$ can be decomposed as follows:

$$\hat{x} = x_s + x_{n,interf} + x_{n,noise} + x_{n,artif} \tag{3.36}$$

where

$x_s = f(x)$ is a version of x_s modified by an allowed distortion f,

$x_{n,interf}$, $x_{n,noise}$ and $x_{n,artif}$ are error terms for interference, noise and artifacts respectively.

Under the hypothesis that the ear splits the speech signal into the same four components as our analytical decomposition 3.36, these four terms should represent, respectively: the parts of $\hat{x}$ perceived as coming from the wanted source s; from other unwanted sources — either other speech signal or noise; from sensor noises — i.e. additive stationary noise; and finally from other causes such as forbidden distortions of the sources and/or "burbling" artifacts — i.e. complex effects and non-stationary elements.

In our case, as presented in Chapter 1, we consider a speech signal degraded by either pink noise or additive real-life noise that contains stationary and non-stationary components. If relevant, this latter element can itself be decomposed into speech signals and other.
Here are the conventions to apply the Source Evaluation metrics to this project. The speech signal is here synonym for $\hat{x}_s$; $x_{n,noise}$ contains stationary noise components from the noise file and from the speech file if necessary; likewise, $x_{n,artif}$ includes non-stationary noise components from both the noise and speech files if relevant, excluding any speech signal; any other speech signal is put into $x_{n,interf}$.

The four ratios derived from Source Evaluation expressed in dB are computed as put hereinbelow and their meaning is explicited accordingly to our conventions presented here above.

Signal-to-Distortion Ratio (SDR)

$$\text{SDR} \triangleq 10\log_{10}\left(\frac{\|x_s\|^2}{\left\|x_{n,interf} + x_{n,noise} + x_{n,artif}\right\|^2}\right) \tag{3.37}$$

This ratio quantifies the energy of the target to all other components, that can be considered as *noise* under its most general meaning.

Signal-to-Interferences Ratio (SIR)

$$\text{SIR} \triangleq 10\log_{10}\left(\frac{\|x_s\|^2}{\left\|x_{n,interf}\right\|^2}\right) \tag{3.38}$$

This ratio quantifies the energy of the target to the energy of the interference source(s). In other words, it measures how dominant the target speech is comparatively to other speech signals.

Signal-to-Noise Ratio (SNR)

$$\text{SNR} \triangleq 10\log_{10}\left(\frac{\left\|x_s + x_{n,interf}\right\|^2}{\left\|x_{n,noise}\right\|^2}\right) \tag{3.39}$$

This is a second definition of SNR based on the decomposition in Equation 3.36 that aims to make this metric independent of SIR. It measures the energy of all speech signals to stationary noise energy.

Signal-to-Artifacts Ratio (SAR)

$$\text{SAR} \triangleq 10\log_{10}\left(\frac{\left\|x_s + x_{n,interf} + x_{n,noise}\right\|^2}{\left\|x_{n,artif}\right\|^2}\right) \tag{3.40}$$

Finally, SAR estimates the energy ratio of all speech signals plus stationary noise relatively to non-stationary noise energy.

While these metrics are largely used in speech enhancement tasks, they lack correlation with auditory perception and are known to be unstable when computed with different numbers of delays and time frames. We resolve this issue in considering two other full-reference performance measures hereafter presented: PESQ and STOI.

3.4.3 Perceptual Evaluation of Speech Quality

Unlike Source Evaluation methods, Perceptual Evaluation of Speech Quality (PESQ) is a measure aiming to assess the speech quality with perception as core concept. It is a speech envelope-based method designed for telecommunication initially, and as an ITU recommendation is publicly available at [Itu]. It scores SQ in the range [1, 4.5] most usually.

It was developed in 2000 conjointly by two research teams which proposed their own algorithm in the frame of a competition organized by ITU-T, aiming to replace the existing models (BSD, PSQM and MNB) that didn't handle properly speech distortion, packet loss and background noise when evaluating speech quality. PESQ is the result of the merging of these two algorithms and was first introduced in [Rix+01]. It is still much employed today when it comes to evaluate the *perceived quality* of a speech signal [PL16; Zha+18; Fak+18; Tu+18; Kol18; Lu+].

The structure of PESQ is shown in Figure 3.12. The model begins by level aligning both signals to a standard listening level. They are filtered (using a FFT) with an input filter to model a standard telephone handset. The signals are then aligned in time and processed through an auditory transform which is a psycho-acoustic model mapping the signals into a representation of perceived loudness in time and frequency. The transformation also involves equalizing for linear filtering in the system and for gain variation. Two distortion parameters are extracted from the difference between the transforms of the signals, called disturbance, and are aggregated in frequency and time before being mapped to a prediction of subjective mean opinion score (MOS).

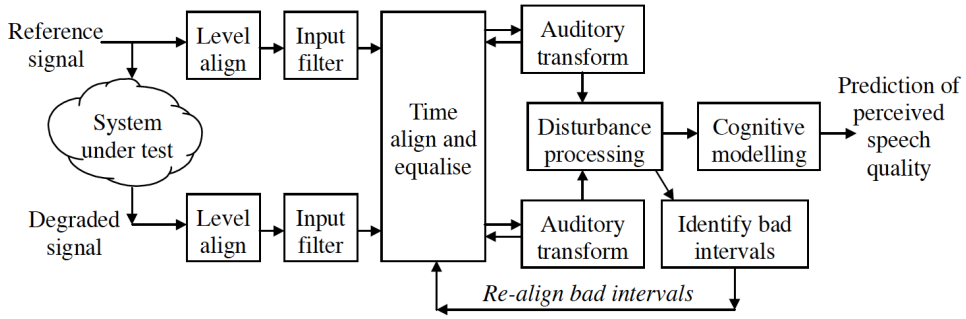


Figure 3.12: block diagram of PESQ model [Rix+01]

PESQ can handle a certain number of situations, among which background (environmental) noise, and noise processing. These can be assessed by presenting PESQ with both the clean or unprocessed original and the noisy degraded signal.

Nonetheless, it presents large missing application fields that have not been tested by ITU. Despite this, PESQ is still very much used to assess the quality of

encoded human voice; this may be due to its time and resource efficiency compared to conducting MOS tests with an extensive sample.

Within the scope of this project, the most relevant deficiency appears when considering multiple simultaneous talkers situations, such as when using babble noise at low SNRs levels. Moreover, PESQ is unreliable when it comes to considering the artifacts resulting from denoising algorithms. Consequently, PESQ figures must be considered carefully.

3.4.4 STOI

The *Short-Time-Objective-Intelligibility* measure is derived from the SNR calculation. It was first presented in [Taa+10] and further improved in [Taa+11]. It is considered state-of-the-art when it comes to correlation with speech intelligibility. In contrast to other conventional intelligibility models which tend to rely on global statistics across entire sentences, STOI is based on short time segments.

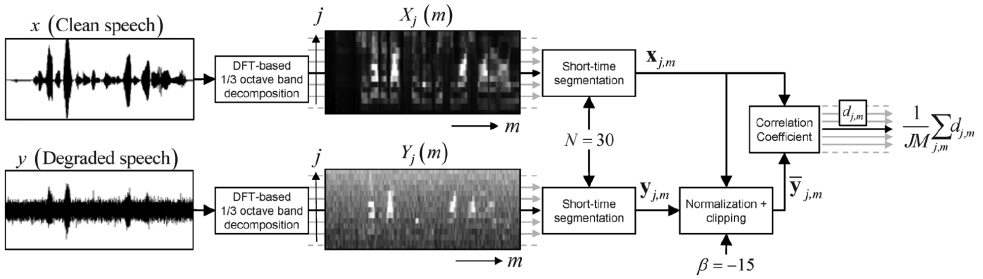


Figure 3.13: block diagram of STOI [Taa+10]

Just like PESQ, STOI is a full-reference algorithm, which means it is a function of both the clean and degraded speech. These signals are processed as illustrated on Figure 3.13. They are first decomposed into DFT-based, one-third octave bands. Next, short-time (a few hundreds milliseconds) temporal envelope segments of the clean and degraded speech are compared by means of a correlation coefficient. Before that, the short-time degraded speech temporal envelopes are normalized and clipped. These short-time intermediate intelligibility measures are then averaged to one scalar value, which is expected to have a monotonic increasing relation with the speech intelligibility. The resulting scale is [0,1] with 0 total non-understanding and 1 perfect intelligibility.
The length of these segments was chosen in order to consider low temporal modulations (above 2–3 Hz) that are important for speech intelligibility, in line. Note that it is about the same order of magnitude as the maximum temporal integration properties of the auditory system.

STOI has proven to be highly competitive with other existent methods for speech intelligibility evaluation (Dau, CSTI, etc.) for a given set of noisy (addi-

tive) situations. Nonetheless, let's consider cautiously the figures outputted with this method, that are valid under the hypothesis that frequency bands contribute independently to intelligibility. Plus, this method presents a default — at least one— and also some weaknesses that were noted to be hopefully fixed in [JT16].

The said default is related to the mapping function used to output STOI. This function is sigmoid-shaped, which implies that uncertainty is bigger for low- or high-on-scale STOI values than for average values. This implies that two STOIs in the high range of the scale — near 1, should be considered equivalent since their uncertainty widths overlap. This makes difficult to compare two situations that result in close STOIs located at the extremities of the scale.

When it comes to weaknesses, the relevance of STOI decreases when the complexity of the data increases, for instance in situations involving negative SNRs and complex noises, as shown in [JT16]. Then, STOI does not take into account some types of absolute threshold in quiet. Therefore, its predictions may not be accurate for operations which significantly reduce the level per band and do not have a strong impact on its temporal envelope (e.g., as with low-pass or high-pass filtering). Finally, STOI shows only weak links to the properties of the auditory system, despite the choice of the window length.

Chapter 4

Experimental Setup

This chapter offers a detailed overview of the software tools, data, and models used. A description of the experiments that have been carried out, as well as a discussion of their results, is deferred to the next chapter.

4.1 Dataset

During the implementation and initial experimentation phases, a collection of news casts from the National Public Radio[1] were used. The dataset, offered by Retune DSP but otherwise available on the radio's website, comprises 33 news report broadcast of approximately 5 minutes each, stored in just as many files.

Each file features speech content from multiple speakers — mostly one per broadcast — with a female-to-male ratio of 1.75. The audio content is predominantly recorded in studio conditions with the exception of interview excerpts which may contain background noise and different speakers.

In order to simulate realistic noisy conditions, the DEMAND dataset of real-world noisy environment recordings [TIV13] was used. This dataset contains multi-channel recordings from environments belonging to the following categories: domestic, nature, office, public indoor, public road, transportation. For each category, there exist three different 5-minute recordings, composed of 16 channels each; only the first channel was used.

The noise samples were divided into stationary and non-stationary, and narrow- and broad-band. An empirical analysis was performed using a *spectral bandwidth* envelope; noises with an average bandwidth higher than 1500 Hz are considered broad-band, and noises with a spectral bandwidth deviation higher than 250 Hz are considered non-stationary. Moreover, in order to have similar levels of perceived loudness when compared to pink noise, a gain factor for each noise file was

[1] `https://www.npr.org/`

computed, using A-weighting curves [FM33].

4.1.1 Splits

The aforementioned raw clean data was assigned into four distinct datasets, which have been used during different stages of the research.

Each dataset varies in terms of amount of source clean data, type of noise applied, and fragment overlap, in order to cater the different needs of each experimental stage.

The splits are described below, in increasing order of complexity:

- **DS0** Used mostly for model troubleshooting, this small dataset is obtained from a single clean speech file and pink noise as 15 dB of SNR, for a total of 1150 audio fragments (5 minutes)
- **DS1** This dataset is composed of approximately one hour of raw content, mixed with pink noise at 5, 15 and 25 dB of SNR, giving a total of 80 000 audio fragments (6 hours).
- **DS2** Similarly derived from one hour of clean speech content, this dataset features three real-world noises — all stationary, with one narrow-band and two broad-band — at 5, 15 and 25 dB of SNR, giving a total of 220 000 training fragments (17 hours)
- **DS3** This last dataset is constructed the same source speech of **DS2** mixed with four real-world noises — one for each combination of stationarity and bandwidth — at 0, 5, and 15 dB of SNR, for a total of 290 000 training fragments (22 hours)

Except for **DS0**, each dataset is further divided into training, validation, and testing subsets; the testing subset is explicitly defined, while the validation subset is derived from the training one at run-time. It is important to note how, while several files have been used across multiple datasets, care has been taken to ensure that the testing subsets are composed of novel clean speech and noises. For a more comprehensive description of the dataset setup, along with a list of all the processing arguments, refer to the experiment training logs in the Appendix A.

4.1.2 Processing

In order to obtain the datasets mentioned above, the clean speech data is fed into a processing pipeline, aimed at replicating the setup of the CHiME Corpus dataset presented in [Chr+10]. The pipeline is composed of the following operations:

Noising Each clean audio file is mixed with a noise source — either pink noise or recording — whose signal power is adjusted to match a given SNR level. This SNR figure is mostly indicative, due to being calculated over the entirety of the clean speech and noise files which are non-stationary. All data is downsampled to 16 KHz upon loading, using a band-limited sinc interpolation algorithm.

Time-Frequency The mixture signal is then converted to the time-frequency domain using short-time Fourier transform with the following parameterization: FFT window of 512 samples, Hann window with 75% overlap; the DC bin is always discarded, leaving 256 frequency bins.

Dynamic range processing A further level of processing, taken from those presented in Section 3.3, is applied to the T-F data; the best fit for this step will be investigated in Section 5.1.

Fragmentation The processed spectrograms are divided into fragments of length `frag_len` composed of overlapping rectangular sliding windows with a hop size `frag_hop`. For a spectrogram composed of N_{frames} STFT frames, the number of resulting fragments is:

$$N_{\text{frags}} = 1 + \left\lfloor \frac{N_{\text{frames}} - \texttt{frag\_len}}{\texttt{frag\_hop}} \right\rfloor$$

For `frag_len`, a value of 32 time frames (270 ms) has been chosen, in order to provide large enough a context while accounting for computational limitations, whereas `frag_hop` depends on the application. This latter parameter doesn't affect the audio characteristics of the data, but provides a easy way of controlling the overall amount of training samples.

Normalization This last step is optional, and consists in subtracting the mean and scaling by the standard deviation, which are then stored separately.

Clean samples to be used as targets are generated according to the same procedure except for the noising stage. All the signal processing is performed using LibROSA [McF+15].

4.2 Framework

A software framework has been implemented, in order to fulfill the most common tasks within the project, such as processing data, training models, and collect results. Since these operations are often inter-dependant, they have been integrated into a command-line interface application, with sub-commands for each task. Each

sub-command interprets a different set of mandatory and optional arguments, allowing for quick tweaking of most common hyper-parameters.

The entire code base has been written in Python 3.6 and uses the Keras library (with Tensorflow backend) for high-level neural network and deep learning functionalities, as well as Numpy, Pandas, Matplotlib and other component of the Scipy ecosystem [Oli07] for data handling and visualization. Further internal libraries have been developed, and divided into data generation, model utilities, general utilities, and signal processing functions.

Along with the aforementioned scripts, a series of Jupyter notebooks has been created, in order to perform exploratory data analyses, test out processing functions and new features, evaluate training outcomes, and navigate logs and metrics reports. While the notebooks are not part of the framework code base, they rely on the same internal and external dependencies as the CLI application, thereby ensuring reproducibility.

Among the activities related to the experiments, the most resource-intensive ones — namely data pre-processing, model training, and evaluation — have been carried out on an ad-hoc workstation comprising three *Nvidia TITAN X Pascal* graphics cards with 12 GB of memory each, and an *Intel Xeon E5-2620 v3* hexa-core CPU. All training sessions have been optimized using the state-of-the-art *Adam* gradient descent algorithm (see Section 3.1.3), with an initial learning rate of 0.005. A scheduler function takes care of halving the learning rate every 150 epochs. Models are left to train for a number of epochs (dependant on the specific experiment stage); if training loss — or validation loss in the last experiment — has been stagnant for more than 25 epochs, training is halted altogether.

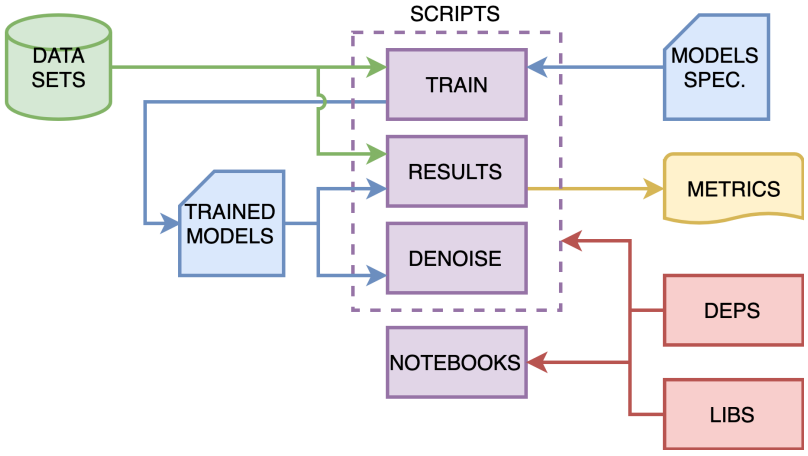


Figure 4.1: software framework for the project, with highlighted data, models, results, and code elements

Figure 4.1 shows the infrastructure in its entirety, whereas the following sub-

sections describe each command script in details. A comprehensive list of input arguments is provided within each script and is accessible through the `--help` flag.

4.2.1 Training

This script is responsible for loading the dataset and training a model on it.

A dataset is specified by passing the path of a directory containing clean speech files, as well a series of processing arguments such as SNR, noise type, dynamic range processing, etc. The noisy-clean pairs — to be used as input and target respectively — are fed into the network by a *data generator* object, which takes care of retrieving the cached pre-processed T-F data from storage and dividing it into randomly distributed batches.

Models can be provided in the form of JSON data structures defining their layer-wise structure, or as pre-trained HDF5 archives to undergo further training. The former can be further parameterized using a simple templating system.

Before training begins, comprehensive logs detailing most parameters and arguments, including those obtained at run-time, are stored as JSON, in order to efficiently troubleshoot failures. During training, a number of processes monitor the model performance, periodically storing snapshots, adjusting the learning rate, and preemptively terminating the process in case of overfitting. Training progresses can be followed in real-time through *TensorBoard*, a tool for visualizing several metrics such as validation and parameter distribution.

4.2.2 Results

This script is used to evaluate the performances of a model against a number of quantitative metrics, such as those described in section 3.4.

Similarly to training, the script accepts a directory — most often different from the one used during training — of clean speech and a series of noising settings. The path to a previously trained model is also required. Once the speech has been processed, the loaded model calculates the predicted clean speech fragments, which are then converted back to time domain following an inverse processing pipeline.

Metrics are obtained for all combinations of clean speech files and noise variations, yielding a multi-dimensional table structure, which is stored as serialized Pandas object for further analysis. They are calculated using the `sklearn`, `mir-eval`, `pystoi` and `pypesq` Python toolboxes.

4.2.3 Denoising

This script is meant to showcase and use the functionalities of a given trained models.

Users can specify a trained model path and a noisy audio file. The script will load the model, process the source file to fit the desired T-F representation (without artificially applying further noise), and predict clean fragments. Subsequently, an inverse processing takes place and the resulting time-domain data is stored as a WAV file.

4.3 Models

In this section, several model architectures, among those that have been devised and investigated, are presented and described.

Four model architectures have been investigated, each of them representing a deep neural network composed of multiple layers, which are parameterized depending on the desired complexity. These models can be divided in *convolutional autoencoders* and *hybrid* networks. The former category identifies autoencoders consisting entirely of convolutional layers, where the dimensionality reduction is achieved by downsampling along the spatial dimensions, as well as diminishing the number of filters. The latter refers to other topologies comprising an initial convolutional stage which feeds automatically extracted features into a *sequence model*, most often in the form of an RNN (in this case we refer to them as *convolutional-recurrent* models). The initial computing stage common to all hybrid networks, can be seen in Figure 4.2

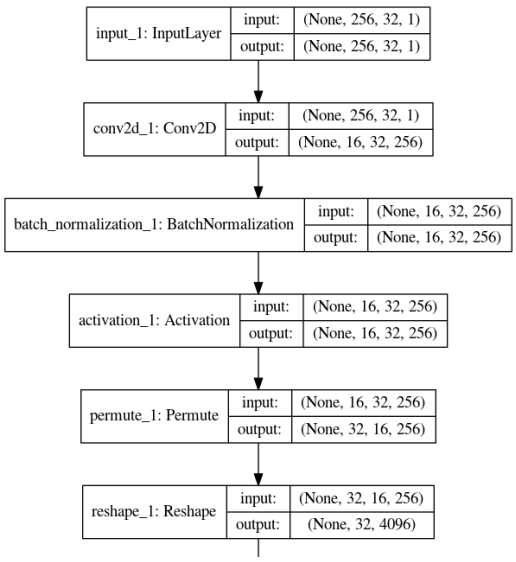


Figure 4.2: computation graph for convolutional stage in hybrid models, highlighting data dimensionality

In order to ensure meaningful comparisons between models, each of them has

been devised so as to accept input data in the T-F domain (see Section 3.29) of shape (F, T, C), where F is the number of frequency bins, T is the number of STFT frames, and C is 2 for real/imaginary representation and 1 in all other cases. Since the target of each model consists of reconstructed clean spectrograms, the shape of the output data is the same as the input. Moreover, all models have been trained against the same loss function, which is a *mean squared error* over a *slice*, i.e. a subset of the STFT frames.

The software implementation of the models is composed of a python parser script and a series of model descriptors. These latter are JSON documents describing the structure of the network, layer by layer, using Keras *functional API* specification. The syntax has been enhanced to support template arguments that can be passed during training, allowing models to be easily parameterized. Examples of template arguments are bias initial values, kernel and stride shape in convolutional layers, number of neurons, and so forth.

The following subsections will elaborate on each architecture in detail.

4.3.1 Convolutional Autoencoder

Autoencoders have become a popular tool for speech denoising tasks; in particular, the proposed network architecture is inspired by the work in [GP17; KZ15; PL16].

AEs — which are thoroughly described in Section 3.1.6 — are composed of two sub-networks called encoder and decoder, where the former compresses the data into a lower-dimensionality space (called *latent space* or *bottleneck*) and the latter reconstructs it.

The encoder in the model presented here consists of stacks of 2D convolution, batch normalization, and nonlinear activation, connected sequentially. Despite commonly adopted in autoencoders, fully-connected layers are not used in the bottleneck since previous works show how the performance gain is not sufficient to motivate the increased complexity. Instead, the first convolutional layer features a large number of channels, and data dimensionality is reduced by downsampling along the frequency and time dimensions, as well as progressively reducing the number of convolution channels. Downsampling with strides of more than one was chosen over other methods such as *max pooling* or *average pooling* since it allows to further limit the number of trainable parameters.

The decoder follows the same approach, stacking sequences of transposed 2D convolution, batch normalization, and nonlinear activation. Transposed convolution is an upsampling technique where the input pixels are spaced out onto a sparser tensor before applying the kernel, causing the result of the operation to have the desired dimension sizes. Deep learning APIs commonly provide transposed convolution layers with the same parameterization as their regular counter-

parts, making it easier to match them. Layers in the decoder are parameterized similarly to the encoder, in a symmetrical fashion. An additional transposed convolution layer without nonlinear activation is present at the end of the decoder, in order to combine the large number of generated features into the desired number of output channels C.

The AE model presented here consists of 3 stacks of convolutional layers for each sub-network. The choice of convolutional parameters such as kernel shape, strides, number of channels, and activation was informed by the existing literature [PLS16; Han+15] as well as experimental results. Overall, the best results have been achieved with kernels of 3×3 and strides of 2×2; this setup considers both time and frequency dimension to be shift invariant, and features a small receptive field. Figure 4.3 shows the implemented autoencoder, along with its input and output dimensions.

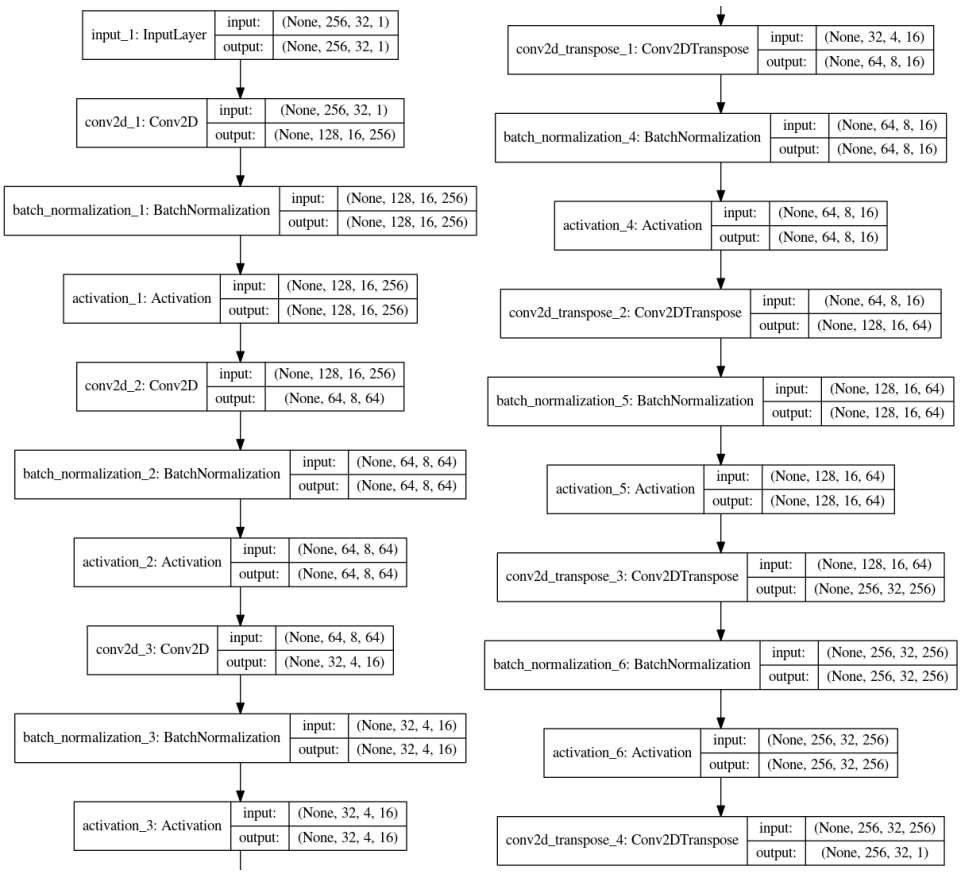


Figure 4.3: computation graph for convolutional autoencoder, highlighting data dimensionality

The activation function was set to ReLU for the vast majority of the tests, whereas the number of channels was adapted to achieve the desired amount of

trainable parameters (i.e. complexity). Moreover, depending on the input representation, the bias parameters were initialized to either zero or one, in order to avoid the aforementioned *dying ReLU* problem.

Fully-convolutional autoencoders are not quite able to encode the sequential relationships existing within speech audio [GBC16]; therefore, the following section will present models based on recurrent neural networks.

4.3.2 Convolutional-Recurrent Model

In [Zha+18], Zhao et al. introduced a novel architecture called *convolutional-recurrent neural network*, substantially outperforming state-of-the-art methods. It aims at modelling the complex long-term patterns occurring within audio spectrograms, while limiting the computational burden using regularized layers.

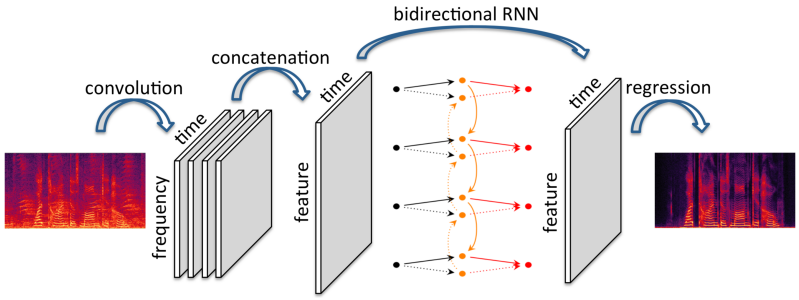


Figure 4.4: block diagram of convolutional-recurrent network [Zha+18]

As shown in Figure 4.4, the model features a multi-stage pipeline composed of convolutional, recurrent, and fully-connected layers.

Firstly, STFT fragments are fed into a convolutional layer, in order to efficiently extract local patterns across time and frequency. The layer comprises 256 channels and kernels of size 32×11 moving across strides of 16×1 and padding set to *same*. These parameters were chosen as to take advantage of the similarities between adjacent frequency bins, provide a large enough context window (receptive field) to each kernel calculation, and limit the overall number of trainable parameters.

The features extracted in the first layer are then concatenated along the frequency axis and fed into a bi-directional recurrent neural network — that is, an RNN with recurrent connections in either direction. In order to avoid the vanishing gradient problem, LSTM layers are used as recurrent units. The output of the RNN component comprises all the hidden representations, as for synchronized sequence-to-sequence use cases.

Finally, a fully connected layer takes care of predicting each clean speech frame.

Since the implementation complexity of the model and the training time required by LSTM layers posed serious limitations to its adoption in this project, two

simplified version were implemented. These models opted out of the bi-directional mechanism — allowing the model to respect causality and be suitable for real-time application — and used an overall smaller input fragment size (the original paper used 500 STFT time frames). Moreover, GRU recurrent units, reportedly known for being more computationally efficient, were used in the RNN component of one of the two architectures. Being most similar to what is described in the aforementioned paper, the other model (based on LSTM units) was used as a baseline against its GRU counterpart, autoencoder architectures, and TCN architectures described below.

4.3.3 Temporal Convolutional Network

Temporal Convolutional Networks (TCNs) are a novel class of neural networks that associate convolutional and recurrent networks. They are based on the idea that a combination of both network types can outperform RNNs in terms of convergence and final performance for a certain number of datasets and tasks, e.g. language modelling. Therefore, TCNs should be regarded as a natural starting point for sequence modeling tasks [Bai18]. Such hybrid models have been implemented for some years [Shi+15] but were first named TCNs in [Lea+16] and later formalized and compared to other traditional structures in [Bai18].

The distinguishing characteristics of TCNs are:

- the convolutions in the architecture are causal, meaning that there is no information "leakage" from future to past
- the architecture can handle input and output sequences of arbitrary length, just like a RNN
- the history size can be user-controlled by properly using residual layers and dilated convolutions (see 3.1.4 for details about ResNets and dilation)

To accomplish the first point, the TCN uses causal convolutions, where an output at time t is convolved only with elements from time t and earlier in the previous layer. To achieve the second point, the TCN uses a 1D fully-convolutional architecture where each hidden layer is the same length as the input layer, and zero padding of length $(\textit{kernel size} - 1)$ is added to keep subsequent layers the same length as previous ones.

TCNs general structure is based on CNNs architecture, but traditional 2D convolutional layers are substituted with 1D dilated convolutions as in [Oor+16]. This is meant to expand the receptive field so as to cover a larger portion of the input sequence. Its effect on the receptive field of each layer can be observed in Figure 4.5, where each node in the upper layers cover a greater and greater portion of their previous ones.

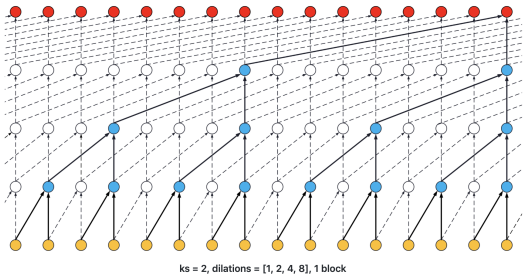


Figure 4.5: effect of dilations on the receptive field of 1D convolution in a TCN architecture

The other main elements at the heart of TCNs are the residual blocks [He+15]. Residual blocks with skip connections are used to allow great depth — which in turn extends the receptive field introducing memory into the network —- while maintaining a stable gradient that does not vanish within e.g. a few dozen layers.

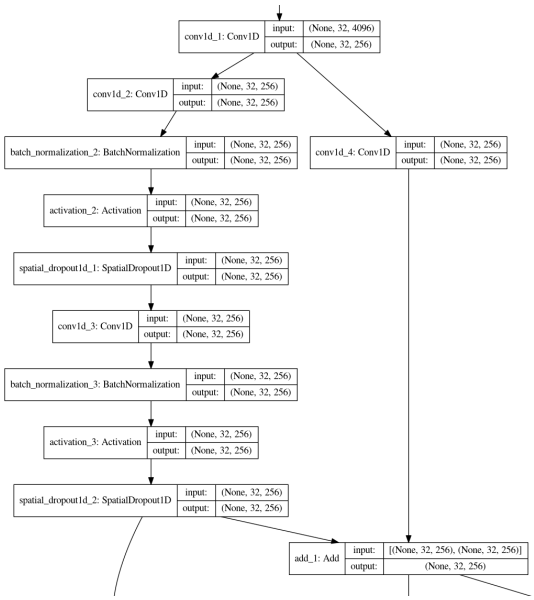


Figure 4.6: computational graph of a single residual block within TCN; arrows on the right connect to the following residual block, whereas the arrow on the left connects to a summator node, together with the output of all other residual blocks

Within each of its residual block, the TCN has 2 layers of dilated causal convolution with kernel size k, ReLU non-linearity [NH10], and batch normalization, which helps to constraint our data in a given range, speeding up the learning process [IS15] and providing regularization. In addition, a spatial dropout [Sri+14a] — similar to regular dropout but operating on an entire 1D channel — is applied for further regularization. This is depicted in Figure 4.6.

On top of a flexible receptive field size and of the ability to take inputs of various sizes, TCNs are remarkable on a few more points.
First, they surpass RNNs regarding their computational speed since several TCNs layers can work at the same time whereas RNNs' layers must wait for the other layers having processed the input data — in sequential order [Bai18]. This implies less storage requirements while training. This advantage becomes a drawback during the evaluation stage, where all the memory is required to re-create the hidden state, unlike RNNs. Then, TCNs stabilize the gradient and help avoiding the famous vanishing/exploding gradient problem. However, a TCN carefully designed for a given problem, it has little chances to be applicable as such to another problem that needs a different size of memory. Finally, let note than TCNs are brand-new architectures, which implies it has been tested only for a limited number of applications (see [Bai18, Sec. 4.]).

Our hybrid model based on TCN — referred to as `conv-tcn` — features 2 stacks n of 4 residual blocks each, where each block is parameterized by kernel size $k = 3$ and dilation rate d, which is progressively spaced, i.e $d = [1, 2, 4, 8]$. The corresponding receptive field $\mathcal{R}$ is calculated as follows:

$$\mathcal{R} = n \times \max(d) \times k, \tag{4.1}$$

and here takes the value $\mathcal{R} = 2 * 8 * 3 = 48$.

Chapter 5

Experiments

This section presents the experiments conducted with the aforementioned model architectures and datasets.

In this project, several aspects of a speech enhancement system have been investigated, such as data representation and pre-processing strategies, model architecture and complexity, and choice of data. However, due to the vast amounts of tunable parameters, testing out all the most sensible combinations would cause the training process to be unreasonably time-consuming.

In order to avoid the substantial computational burden, a multi-stage process has been devised, with the aim of tuning a subset of all parameters at a time, while maintaining sensible values for the others. This allows for ill-conceived parameterizations to be ruled out along the process, thereby maintaining focus on effective strategies only. While our overall focus is maximizing speech enhancement performances using the presented models, each stage — or phase — is concerned with optimizing different aspects.

Figure 5.1 presents the experimental procedure, exposing the parameters, models, and datasets adopted at each stage.

In the following sections, the nature of each experiment is documented and the findings obtained from each stage are reported and discussed, divided by area of impact.

Moreover, an evaluation of the performances of each model is carried out, under a variety of scenarios (such as data training set or hyper-parameter tuning), using the metrics introduced in Section 3.4. For reference purposes, the convolutional-recurrent model based on LSTM units (documented in Section 4.3.2) will serve as baseline.

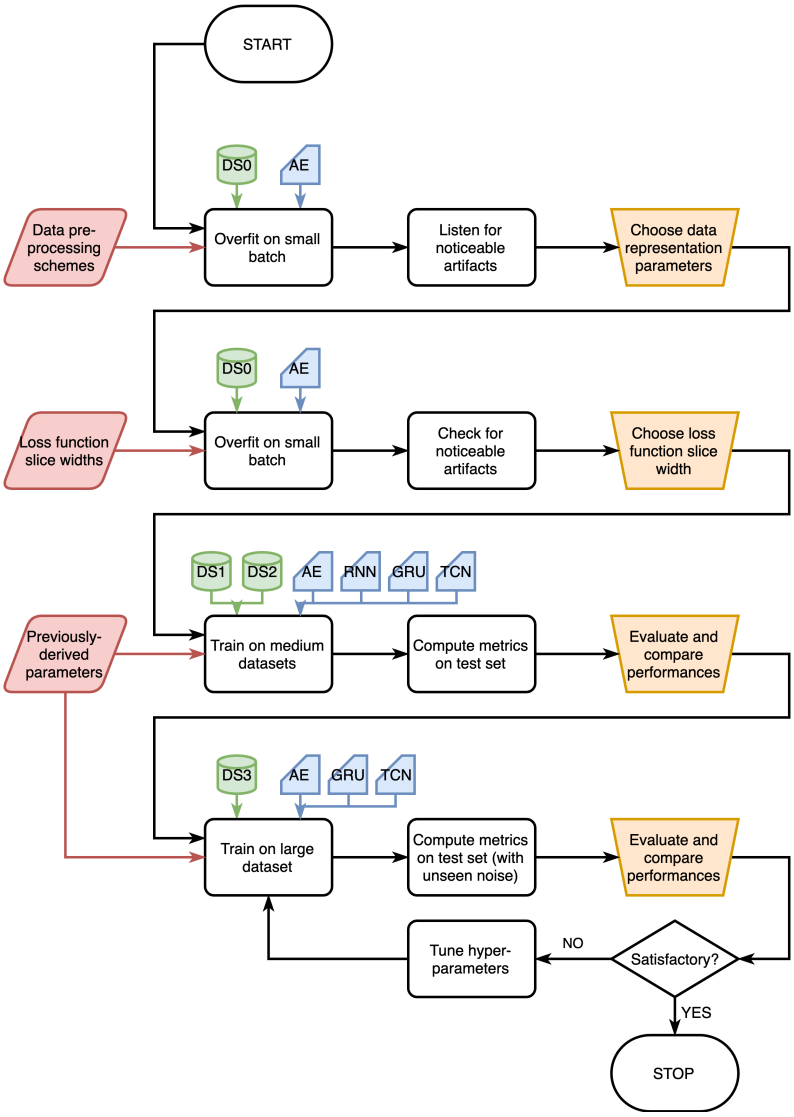


Figure 5.1: flowchart documenting the experimental procedure

5.1 Choice of data processing scheme

During this initial phase, different pre-processing techniques — among those described in Section 3.3 — are evaluated, in order to find the ones able of encoding the data in the most effective way.

5.1.1 Procedure

The six schemes described in Table 5.1 have been tested. Moreover, an optional *Z-normalization* (zero-mean, unit-variance) step was introduced, bringing the total number of parameter combinations to $6 \cdot 2 = 12$.

Name	Description
`exp1`	Magnitude spectrum ($\alpha = 1$)
`exp2`	Power spectrum ($\alpha = 2$)
`exp066`	Auditory spectrum ($\alpha = 0.667$)
`exp016`	Even skewness spectrum ($\alpha = 0.167$)
`db`	Power spectrum, in decibels
`reim`	Real/Imaginary complex spectrum

Table 5.1: pre-processing schemes (α is the exponent applied to each spectrogram T-F unit)

Since this process also acts as *sanity check*, it was decided to follow the common practice of allowing a simple model to overfit on a small batch of data [AA18]. A fully-convolutional autoencoder like the one presented in Section 3.1.6 was trained on the smallest dataset **DS0** introduced earlier.

The model featured 3×3 kernels with strides of 2 along both dimensions, and a number of filters of 256, 64, and 16 for each convolutional layer respectively, comprising in total approximately 300 000 trainable parameters. The input shape was set to 256×32, corresponding to around 270 ms of data. Depending on the distribution of the pre-processed input data (either zero-centered or always positive), convolutional biases were initialized to 1 or 0.

Given that computation occurring in the other models is very similar to that of the AE (i.e. convolutional layers, ReLU activation, etc), it was deemed sufficient to perform this on one of the models only, and apply the findings later with the other ones too, without having to repeat this process.

A training session of up to 500 epochs was issued for each pre-processing scheme, using the regular mean squared error metric as loss function (i.e. no slice); validation loss calculation was disabled in order to increase speed.

Once the AE model was trained for each combination, a listening evaluation on the reconstructed training set was performed, so as to rule out any pre-processing

scheme that would introduce artifacts.

5.1.2 Results

According to the results of this first experimental stage, the choice of representation and pre-processing has profound impact on the network ability to reconstruct the data.

	Normalization	
	Yes	No
exp016		✓
exp066		✓
exp1		✓
exp2		
reim		✓
db		✓

Table 5.2: outcome of manual selection for different pre-processing schemes, based on presence of audible artifacts

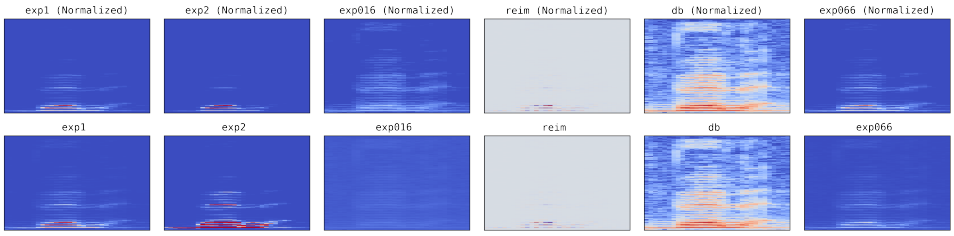


Figure 5.2: reconstructed fragments from different different pre-processing schemes

Figure 5.2 shows sample spectrograms from each combination, while Table 5.2 summarizes their outcome in term of suitability. As expected, out of the 12 processing scheme tested, only a handful of them yielded satisfactory performances.

In particular, none of the attempts at performing fragment-wise Z-normalization proved to work, producing noticeable artifacts when combined with any of the data representations. These artifacts would manifest in the form of a slight gating of the noise at different frames, as well as a heavy dynamic range compression causing noise to dominate otherwise silent portions of the audio.

This may be explained by the fact that, while the global SNR level is constant, noise and speech figure at different power ratios in each fragment. In turn, this causes identical noises to be scaled differently, leading to the model not being able to learn the relationships in the data.

Among the pre-processing schemes without normalization, `exp2` turned out to be defective too, presenting substantial distortion in the reconstructed speech. It is also worth noting how this data representation is susceptible to invalid numeric results when going through the post-processing stage, as the square root operation is only defined for positive numbers. This has been addressed by clipping negative numbers in the predicted spectrogram to zero before feeding the data to the post-processing function.

While the same precaution had to be adopted for `exp066`, the latter proved quite effective at predicting clean speech. In general, given $b^n = x$, any negative b and non-integer n yield a complex result; while the input data for any `exp*` representation is bounded at zero, predicted data might contain spurious negative values, causing the aforementioned issue. This however does not constitute a problem with `exp016` because the inverse of 0.166 — which is used in the post-processing — is the integer 6.

Decibel (logarithmic) representation also proved successful, particularly after introducing a initial bias value of 1 for convolutional layers. Similarly, `reim` representation resulted in satisfactory reconstruction.

It is worth noting how comparing the performances of the different pre-processing schemes using the training loss value is inconclusive, due to it being dependent on the order of magnitude of the data. This means that there is no clear way of determining the best technique, without resorting on the evaluation metrics. As such, it was decided to proceed with the following stage with all the satisfactory setups, and subsequently perform a more selective pick.

5.2 Choice of loss function slice

A second hyperparameter that was deemed significant is the loss function *slice* width.

This stems from the fact that — in low-latency use cases — only the newest (latest) STFT frame of a given predicted fragment is used, while all of the others are discarded. In turn, this raised doubts as to whether these unused frames should affect the cost function at all, leading to this part of the experiment. However, it was decided to use centered slices instead of rightmost ones, in order to provide context data on both sides.

5.2.1 Procedure

In order to investigate this matter, the pre-processing schemes which achieved satisfactory reconstruction performances were employed on further training sessions, parameterized as per previous description, and with a slice applied to the loss function. Any of the data outside of this slice would be used merely as context,

without contributing to the loss value. The extra slice widths that have been tested are 10% and 50% of the data width (3 and 16 STFT frames respectively), as well as a single frame.

Given that most of the resulting trained models ended up generating uncorrupted predictions, a simple quantitative metric, akin to SDR (see Section 3.4.2), was used to determine the best combinations of data pre-processing and loss function slice:

$$\mathrm{SDR} = 10 \log_{10} \frac{1}{TF} \sum_{f,t} \frac{{s_{f,t}}^2}{(s_{f,t} - \hat{s}_{f,t})^2} \tag{5.1}$$

where s and $\hat{s}$ are the original and reconstructed clean speech spectrograms, and f, t are frequency and time frame indexes reaching up to F and T respectively.

5.2.2 Results

Table 5.3 shows the combinations which did not introduce substantial artifacts. It is interesting to notice how a window composed of a single frame always resulted in corrupted output data. Similarly, `exp016` did not give any successful results with smaller slices. The real/imaginary data representation also yielded mixed results. Consequently, these setups have been discarded.

	Slice width			
	1 frame	10%	50%	100%
`exp1`		✓	✓	✓
`exp016`				✓
`reim`		✓		
`db`		✓	✓	✓
`exp066`		✓	✓	✓

Table 5.3: outcome of observations for different pre-processing schemes and loss function slice width, based on presence of audible artifacts

When using a smaller slice width, care must be taken in ensuring that individual output fragments are merged using the relevant portion — that is, the window considered by the loss function. Therefore, in order to compare the performances in a fair and realistic scenario, a fragment hop of a single time frame was used; for slices larger than that, the leftmost frame was used.

Figure 5.3 shows a selection of reconstructed fragments. Interestingly, the `conv-ae` model seems to attempt to reconstruct parts of the fragment not covered by the loss function, indicating how the extra context is indeed considered in the computation.

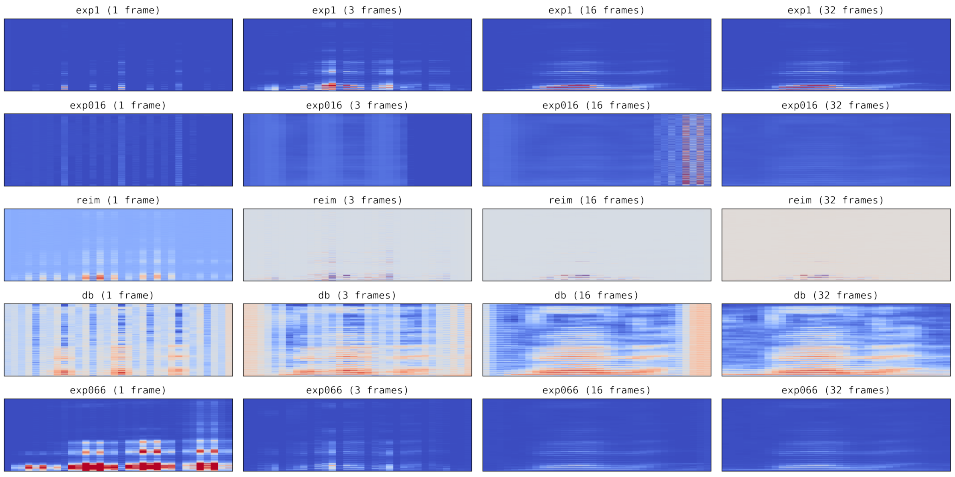


Figure 5.3: reconstructed fragments from different different pre-processing schemes and loss function slices

Based on hearing tests on all combinations of processing and loss function slice, it was decided to not proceed further with `exp016` and `reim`, as they did not seem to behave consistently. In order to determine the best combinations among the remaining ones, the simple SDR metric was computed, using the training set **DS0** as input data.

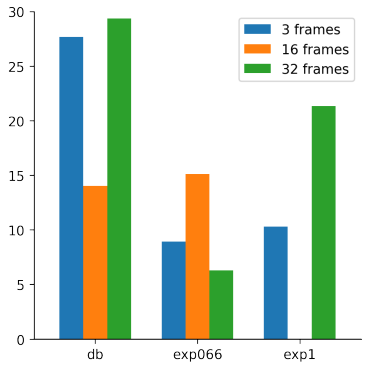


Figure 5.4: SDR for reconstructed fragments from different different pre-processing schemes and loss function slices

The results, summarized in Figure 5.4, did not prove particularly conclusive, especially for the `exp066` auditory spectrum representation, with was subsequently discarded.

Since the small difference between 3 and 32 frames in the `db` representation seemed particularly encouraging, it was decided to follow that initial intuition, and investigate smaller loss function slices. In order to maintain a certain level of

comparison, the same thought was applied to `exp1` representation, despite the less impressive results.

Summing up the results of the previous two experimental stages, it was chosen to adopt `db` and `exp1` (logarithmic and magnitude representations respectively) with a slice width of 10% of the original fragment width, corresponding to 3 time frames.

5.3 Preliminary training

This subsequent stage consists in training the aforementioned four models on two new datasets, **DS1** and **DS2**, with the pre-processing hyperparameters that were selected beforehand. While the former dataset — processed with pink noise — was useful in the development stages, the latter aimed at testing the models in a more realistic scenario.

5.3.1 Procedure

The models were trained for every retained data representation. As in previous stages, the input shape was set to 256×32, corresponding to around 270 ms of data, and the bias was initialized at 0 or 1 depending on the distribution of the pre-processed input data (either zero-centered or always positive).

The convolutional-recurrent hybrids were designed on a similar basis, while the parameters of `conv-ae` were adapted to the computational limitations of the hardware. Thus, on the one hand, `conv-ae` featured 3×3 kernels with strides of 2 along both dimensions, a number of filters of 256, 64, and 16 for each convolutional layer respectively, and represented approximately 1 000 000 trainable parameters; on the other hand, all hybrid models were set with 256 convolutional filters, `conv-gru` and `conv-rnn` (the baseline) were given a kernel size of 32×11 and strides of 16×1 while `conv-tcn` took a kernel of 3.

Each model was trained in batches of 256 samples; however, in order to fit the `conv-ae` model into the 12 GB available on the GPU, its batch size was reduced at 200. The RNN, GRU and TCN models comprised approximately 10, 9, and 5 million trainable parameters respectively.

Since making the models overfit was not the purpose of the training anymore, the validation dataset was enabled. This, along with the larger train set, consequently lengthened the trainings: while the hybrid models trained for 150 seconds/epoch on average, `conv-ae` took around 6 minutes per epoch. Therefore, it was launched with 200 epochs as upper limit against 300 for the other models.

The trained models were then evaluated against the metrics presented earlier, through the results script, using the 20-minute test set. For the results of models

trained on **DS2**, a combination of seen and unseen, and broad- and narrow-band noises was used, in order to assess the generalization capabilities of a given model.

5.3.2 Results

In this section, the impact of two essential elements in deep learning are studied in regard to the performance of the models. On the one hand, the influence of the parameters related to noise is highlighted, including noise types, SNR levels, frequency range and usage during training. On the other hand, the impact of the data representation is evaluated. All these parameters are used to discriminate models one against another.

For clarity sake, we introduce the following notation to compare models against one another, relatively to all aforementioned variables: further on, *better* means "better in the sense of the metrics", except for mean square error which is not considered in the analysis but is shown for completeness sake. Inversely, *inferior* means "inferior in the sense of the metrics".

	mse	sir	sar	sdr	stoi	pesq
conv-rnn	0.98	0.68	0.96	0.91	0.96	0.85
conv-gru	1.12	0.68	0.89	0.84	0.95	0.78
conv-tcn	1.40	0.65	**0.74**	0.69	**0.94**	0.78
conv-ae	**2.19**	**0.61**	**0.74**	**0.66**	0.95	**0.77**

Table 5.4: Ratio of model performances comparing SNR between 5 and 15 dB for dataset **DS2**, data representation `db`. But for MSE, values lower than 1 indicate better performance for broad-band noises. Highlighted in **bold**, are the values for most impacted models and underlined, the values for most impacted metrics.

5.3.2.1 Impact of SNR level

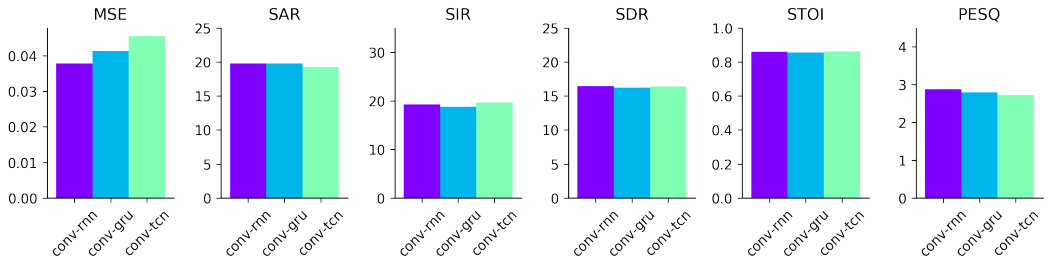
The metrics were calculated over two SNR levels: 5 dB and 15 dB; the former aims at putting the model under high constraint while the latter is meant to be an "easy" case.

Unsurprisingly, the tests issued better metrics for the highest SNR level, and this without any exception for all combinations of other parameters. This was expected for a high SNR implies lower masking of the target in the mixture, therefore a higher SIR as featured in Table 5.4. Let notice that SIR and STOI are respectively the most and least affected metrics with an improvement of 53 and 5 points in average, when comparing $SNR = 15$ dB to $SNR = 5$ dB. Intelligibility thus seems to be particularly robust to noise intensity ratio variation. When it comes to comparing models against one another, `conv-ae` and `conv-tcn` yield comparable performance

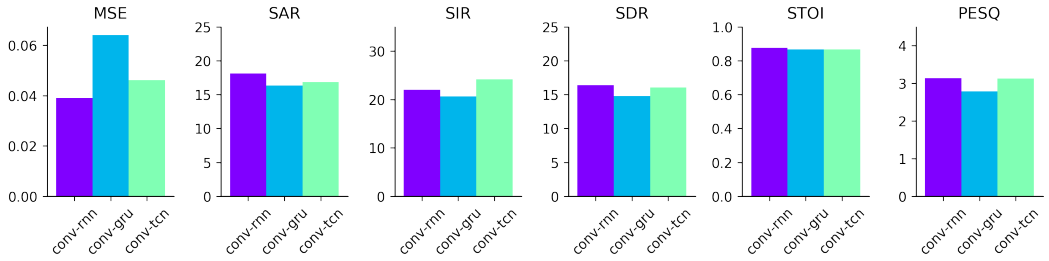
progress between $SNR = 5$ and $SNR = 15$; this evolution, which is the steepest among models with 37 points, implies that these two models may have difficulties to generalize to harder noise conditions in further experiments. On the contrary, our baseline seems more robust to noise variation with *only* 17 points in comparaison.

5.3.2.2 Pink noise vs real-world noise

The datasets **DS1** and **DS2** offer the possibility for comparing the models against different types of stationary noises, without any other change in the parameterization.



(a) DS1 - pink noise



(b) DS2 - real-world noise (seen)

Figure 5.5: performance metrics trained on **DS1** and **DS2** at $SNR = 5dB$, for various models with `exp1` representation

Let us compare the trained models for a given representation depending on the dataset. Due to a design error which caused it to output unusable results on **DS1**, `conv-ae` is not considered here.

- `db`: for all models, considering all metrics averaged on the files under test and for every SNR considered, `db` representation yielded better results on average when applied to real-life noises, compared to when applied to pink noise. The only metric being the exception is SIR, which turned out to be slightly higher on data with pink noise.

- `exp1`: the results obtained with the magnitude spectrum representation are

more mitigated, as illustrated on Figure 5.5. While this representation seems to work particularly well with data from datasets for both `conv-gru` and `conv-rnn` models, it appears to yield weaker SIR and PESQ for `conv-tcn` on **DS1**. The other metrics STOI, SAR and SDR seem to be higher on **DS1** than on **DS2** for all models, but for STOI on `conv-tcn` which is relatively equal for both datasets.

It appears that no conclusive trend regarding the impact of noise emerges straightforwardly from these results independently from the data representation. Therefore, a further step will consist in choosing a data representation.

5.3.2.3 Impact of frequency range

In **DS2**, we compare models performances relatively to the variation of noise considered. The results were run on noises with either a broad or a narrow frequency range; for conciseness, we use 'narrow noise' or 'narrow-band noise', and 'broad noise' or 'broad-band noise' to designate a noise with a narrow frequency range or with a broad frequency range, respectively.

	mse	sir	sar	sdr	stoi	pesq
conv-rnn	1.05	0.93	0.85	0.86	0.98	1.00
conv-gru	1.18	0.98	0.79	0.82	0.98	0.99
conv-tcn	1.16	0.92	0.73	0.76	**0.97**	**0.96**
conv-ae	**1.80**	**0.87**	**0.68**	**0.70**	0.98	**0.96**

Table 5.5: Ratio of performances comparing narrow-band and broad-band noises for $SNR = 5$ dB, data representation `db`, dataset **DS2**. But for MSE, values lower than 1 indicate better performance for broad-band noises. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metric

Broad-band noises result in better metrics than narrow ones, whichever model is considered. The trend is particularly visible for `conv-ae` as model (gain of 21 points in average on all metrics but MSE) and SAR as metric (gain of 32 points in average on the models), as can be seen in Table 5.5; on the contrary, the change is far more neglectable for `conv-rnn` as model (gain of 9 points in average on all metrics but MSE) and for our state-of-the-art metrics PESQ and STOI (gain of 2 points for both in average on all models). The phenomenon intensifies for `exp1` relatively to `db`.

5.3.2.4 Impact of usage in training

In **DS2**, we compare models performances relatively to the variation of noise considered. The results were run on noises that the network had trained on, and on unknown noises; they are referred to as 'seen' and 'unseen' respectively.

As expected, seen noises seem generally better processed than unseen noises, as illustrated in Table 5.6. Let notice that SIR is clearly the most altered metric with a gain of 49 points in average on the models for seen noises relatively to unseen noises. This implies that new noises are less well detected and suppressed by the network, which results in more interference to the target speech.

	mse	sir	sar	sdr	stoi	pesq
conv-rnn	0.99	**1.56**	1.03	1.14	1.03	1.13
conv-gru	0.95	1.47	1.04	1.14	1.03	1.10
conv-tcn	0.89	1.44	1.07	1.20	1.03	**1.16**
conv-ae	**0.61**	1.50	**1.12**	**1.33**	1.03	1.15

Table 5.6: Ratio of performances comparing seen and unseen noises for SNR=5 dB, data representation `db`, dataset **DS2**. But for MSE, values higher than 1 indicate a better performance for seen noises. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metrics (but MSE).

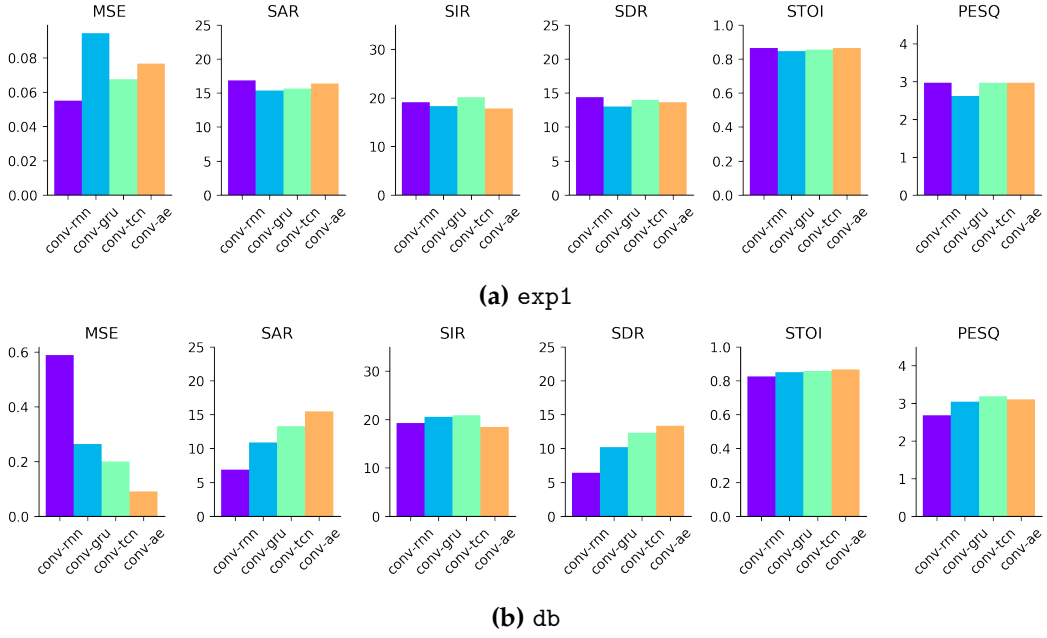
When it comes to models, `conv-ae` seems to be the most affected model with an averaged gain of 23 points (calculated on all metrics but MSE) against 18 in average for all other models. It means that `conv-ae` is not as good at generalizing to new data as the other networks.

5.3.2.5 Magnitude vs logarithmic representation

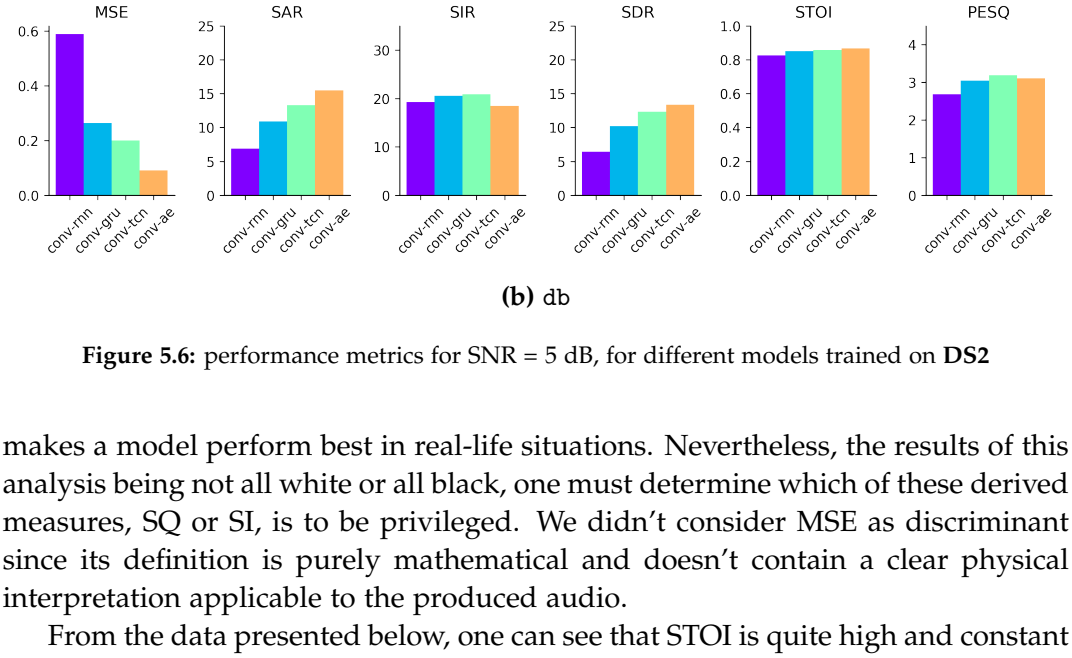
Let compare models performances relatively to data representations. On the one hand, our `conv-rnn` on data representation `exp1` yielded better results for all metrics (averaged on the files under test), for both SNRs and both datasets; our `conv-gru` resulted in similar dominance of `exp1` on `db` for **DS1** only. On the other hand, `conv-tcn` for both datasets and `conv-gru` for **DS2** show similar behaviours in outcoming dominant PESQ and SIR for `db` and dominant STOI, SAR and SDR for `exp1`. `conv-ae` follows a close pattern but is barely discernible. The observations for **DS2** are displayed on Figure 5.6 and Table 5.7, which represent absolute scores for both data representations and relative performances respectively.

From the observations listed here above, let note that the metrics seem to split into two separate groups. The first one is composed of PESQ and SIR, and is further referred to as subjective Speech Quality (SQ) metrics. The second one is constituted of STOI, SAR and SDR and is further referred to as objective Speech Intelligibility (SI) metrics. Among these two new performance measures, PESQ and STOI are considered as most representative of their respective group since they are state-of-the-art metrics. MSE seems to change inversely to the general level of other metrics.

The previous comparisons offer support to select the data representation that



(a) `exp1`



(b) `db`

Figure 5.6: performance metrics for SNR = 5 dB, for different models trained on **DS2**

makes a model perform best in real-life situations. Nevertheless, the results of this analysis being not all white or all black, one must determine which of these derived measures, SQ or SI, is to be privileged. We didn't consider MSE as discriminant since its definition is purely mathematical and doesn't contain a clear physical interpretation applicable to the produced audio.

From the data presented below, one can see that STOI is quite high and constant for all models, datasets and representations; SI is therefore not discriminatory. On the contrary, PESQ seems to vary on a larger amplitude. Accordingly, we made the choice to give preference to SQ upon SI. Finally, ensuing from the previous hierarchisation, `db` representation was selected for all models for the following reasons: for `conv-ae`, `conv-tcn` and `conv-gru` on the basis of their performances under this representation; and for `conv-rnn` in order to make it comparable to the other models.

5.4 Fine-tuning

Given the results of the preliminary evaluation conducted in the previous experimental stage, a fine-tuning of the remaining hyperparameters was carried out, with the purposes of further improving the performances of the models, as well as their generalization abilities. The dataset of choice for this final stage was **DS3**, which features more challenging data due to the lower SNR levels, as well as the

	mse	sir	sar	sdr	stoi	pesq
conv-rnn	0.09	0.99	**2.46**	**2.24**	**1.05**	**1.11**
conv-gru	**0.36**	**0.89**	1.41	1.27	0.99	0.86
conv-tcn	0.34	0.96	1.18	1.14	1.00	0.93
conv-ae	0.85	0.97	1.06	1.02	1.00	0.96

Table 5.7: Ratio of performances between data representations `exp1` and `db`, all noises included for SNR = 5 dB on dataset **DS2**. But for MSE, ratios bigger than 1 feature a better performance for `exp1`, and the value itself indicates to what extent. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metrics (but MSE).

presence of non-stationary noises which differ greatly from pink noise.

5.4.1 Procedure

Each model was adjusted on an individual basis, and different optimization strategies were adopted. This stage was informed by similar procedures performed in the literature [KZ15; PL16] as well as insights obtained during the previous phases. Network depth was increased in the following ways:

- `conv-ae`: more convolutional layers were added to both encoder and decoder; however, in order not to dramatically increase the computational requirements of the model, 1×1 (depthwise) kernels were used every second layer, a technique adopted in Google's *Inception* model [Sze+14]
- `conv-gru`: experiments have been conducted by increasing the size of the convolutional output (i.e. by reducing the strides across the frequency axis) with the intent of producing richer features, increasing the number of time steps considered by the recurrent network, and by stacking multiple GRU-based networks sequentially
- `conv-tcn`: different receptive fields were tested on in changing the number of dilations and stacks; plus, the convolutional output was enlarged just as for `conv-gru`.

Moreover, dropout was introduced on fully connected layers as well as on the convolutional layers within TCN, to avoid overfitting. In this stage, the training batch size was set to the maximum allowed by each model, depending on its resulting number of trainable parameters and layer dimensionality. Most often, this resulted in up to 1024 samples per batch for the hybrid models and 200 for the autoencoder-based one.

Finally, due to its role as baseline model, `conv-rnn` was not tine-tuned, but merely retrained on the larger, more complex dataset.

During this stage, evaluation of the models was performed continuously, in order to steer further tuning in the right direction. As such, the results discussed herein are an overview of the most interesting outcomes.

5.4.2 Results

During the last stage of our procedure, after analyzing the impact of the stationarity of the noise on the performance of the models, the best performing models of each type are optimized in playing on the network depth and on hyperparameters such as the dropout rate. However, due to either the poorness of some results `conv-rnn` and `conv-gru` and to the high time-consuming requirement for `conv-ae`, a single optimized version of these three models was trained. Therefore, they do not offer ground to be compared in this very section, and the research on these hyperparameters is here pursued and contrasted for several model versions for `conv-tcn` only.

5.4.2.1 Impact of network depth

In this paragraph, the influence of the network depth is investigated for `conv-tcn`.

An important notion of `conv-tcn` is the receptive field, as introduced in 4.3.3. We chose to tune it through two parameters, namely the dilation rate d, and the number of stacks (or layers) n. The reference model is the one presented in 4.3.3, which features $(n_d, n) = (4, 2)$ with a kernel k of 3. This last feature being not varied, it is not mentioned further on to avoid overloading text. Three experiments were conducted on **DS2**, all based on the notion of receptive field.

- **Same receptive field:** Two models with respectively $(n_d, n) = (2, 4)$ and $(n_d, n) = (3, 4)$ were trained and compared to the referent `conv-tcn`. According to Equation 4.1, the receptive field of the model is $\mathcal{R} = 48 = \mathcal{R}_{ref}$. When comparing our reference and $(n_d, n) = (2, 4)$, one sees that both models perform relatively equally for STOI and PESQ, as illustrated on Figure 5.7.

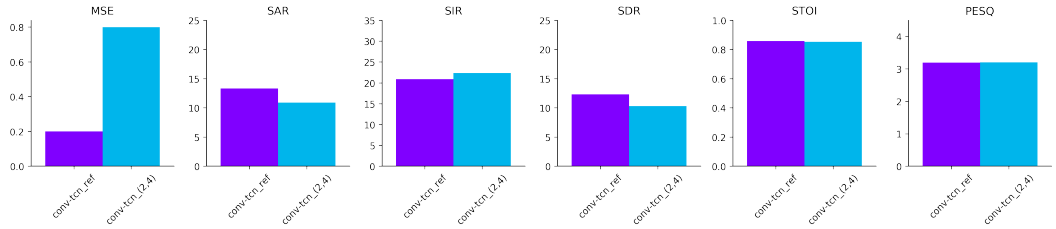


Figure 5.7: Compared performance of two `conv-tcn` models of same receptive field.

- **Smaller receptive field:** A model with $(n_d, n) = (2, 4)$ was trained and monitored on Tensorboard. Its receptive field is $\mathcal{R} = 24 = \mathcal{R}_{ref}/2$. This model proved to perform as good as with a receptive field twice as big. It is coherent with the findings of Bai et al. in [Bai18], who states that TCNs are relatively insensitive to hyperparameter variations, as long as the receptive field size is big enough. This may imply that our receptive field may have been reduced from the beginning for an equal performance; nonetheless, we chose to take Bai's word for it in generously sizing our network to prevent any drawback mentioned in [Bai18].
- **Bigger receptive field:** Two models with $(n_d, n) = (4, 4)$ and $(n_d, n) = (2, 16)$ were trained and monitored on Tensorboard. Their receptive field is $\mathcal{R} = 96 = 2 * \mathcal{R}_{ref}$. Both of them were found to show skyrocketing divergences on the validation loss early in the training. Therefore, these combinations were discarded as not stable enough. This is surprising for it reveals that $\mathcal{R}$ has not only a lowest limit, but also a highest one.

Based on these experiments, we brain-stormed about the value for $\mathcal{R}$ for it is recommended in the literature to set it depending on the history size of the input data and on its dimensionality [Bai18]. Our data being reasonably sized, and because we potentially considered varying the number of time frames per fragment in future experiments, the receptive field stayed finally fixed at $\mathcal{R} = 48 = \mathcal{R}_{ref}$ with $n \leq 4$.

5.4.2.2 Impact of training hyperparameters

	mse	sir	sar	sdr	stoi	pesq
conv-tcn-n4d4	1.05	0.96	0.96	**0.96**	1.0	0.99
conv-tcn-drop	0.91	**0.94**	1.01	1.00	1.0	**0.98**
conv-tcn-feat	**0.82**	0.96	**1.05**	1.03	1.0	0.99

Table 5.8: Ratio of performances comparing various `conv-tcn` models to `conv-tcn-ds2` on dataset **DS3** for SNR = 5 dB and data representation `db`. But for MSE, ratios bigger than 1 feature a better performance of the newest models, and the value itself indicates to what extent. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metrics (but MSE).

Four `conv-tcn` models were tested on **DS3** with same receptive field $\mathcal{R} = \mathcal{R}_{ref}$. One is the exact `conv-tcn` used on **DS2**, and is designated by `conv-tcn-ds2` further on; the others differ from the previous `conv-tcn-ds2` as follows:

- `conv-tcn-n4d4`: changing the complexity of the network (number of parameters: 6.6 million) in setting $(d, n_d) = (4, 4)$.

- `conv-tcn-drop`: adding a dropout rate of 0.2.
- `conv-tcn-feat`: changing the convolutional output with kernel size of 16×5 and strides of 8×1.

The performances of these three models relatively to `conv-tcn-ds2` are displayed in Table 5.8. All three models seem to outcome results in the same order of magnitude as `conv-tcn-ds2` but with slightly degraded SQ; while `conv-tcn-n4d4` appears least performant for all metrics, `conv-tcn-drop` and `conv-tcn-feat` seem to maintain and even improve SI — though moderately. Based on the same assumption as previously, one selects the best `conv-tcn` model on SQ consideration basis, meaning that `conv-tcn-ds2` is selected as optimal `conv-tcn` model.

5.4.3 Performance Evaluation

The best model for each type of architecture has been selected; this subsection aims at discriminating these models against one another regarding the task considered, i.e. speech denoising.

To this end, a general comparison of their performances is first conducted before addressing the impacts related to the noises, namely their stationarity, their frequency band, and finally the knowledge of the network about them (if the network was trained on it or not). All plots and samples comforting this analysis are available either in this subsection or in A as attachment to the present report.

Given the similar trend observed before, regarding performances at different SNR levels, the following analysis focuses on $SNR = 5$ dB, which is the "difficult" case.

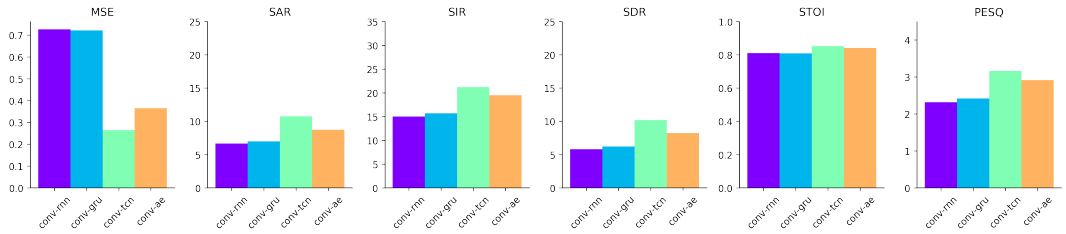


Figure 5.8: performance metrics for SNR = 5 db, for different optimized model and baseline (`conv-rnn`), trained on **DS3**

5.4.3.1 General behaviour

The absolute performances of the four models are presented in Figure 5.8, while the Table 5.9 displays the metrics of the three optimized models relatively to the baseline `conv-rnn`.

It appears that conv-rnn (baseline) is indeed outperformed by the three other models: conv-tcn clearly shows the best metrics, with an improvement on the baseline of 44 points (calculated on all metrics but MSE); conv-gru is only slightly better than conv-rnn and stays in the same order of magnitude with an averaged improvement of 4 points; finally, conv-ae performs just in between with an averaged improvement score of 27 points. One can notice that all optimized models provide a real enhancement of SQ particularly, which turned to be clearly audible when testing samples of the predicted data in suppressing part of the introduced noise and in diminishing the intensity and frequency of the artefacts specific to conv-rnn.

	mse	sir	sar	sdr	stoi	pesq
conv-gru	0.99	1.05	1.05	1.07	1.00	1.04
conv-tcn	**0.36**	**1.41**	**1.62**	**1.75**	**1.05**	**1.36**
conv-ae	0.50	1.30	1.31	1.42	1.04	1.26

Table 5.9: Ratio of performances comparing conv-gru, conv-tcn, conv-ae to baseline conv-rnn on dataset **DS3**, for SNR = 5 dB and data representation db. But for MSE, ratios bigger than 1 feature a better performance of the considered model relatively to conv-rnn, and the value itself indicates to what extent. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metrics (but MSE).

5.4.3.2 Impact of noise stationarity

The last dataset **DS3** made the networks see stationary and non-stationary noises. This subsection therefore aims at discriminating our models against one another based on the impact of this new noise component on their performances.

	mse	sir	sar	sdr	stoi	pesq
conv-rnn	0.98	1.03	0.95	0.94	1.00	**1.02**
conv-gru	1.00	**1.06**	**0.93**	**0.92**	1.00	**1.02**
conv-tcn	**1.09**	1.04	**0.93**	0.93	1.00	1.01
conv-ae	1.02	0.98	0.94	**0.92**	**0.99**	1.00

Table 5.10: Ratio of performances comparing stationary and non-stationary noises on dataset **DS3** for SNR = 5 dB and data representation db. But for MSE, ratios bigger than 1 feature a better performance on stationary noises, and the value itself indicates to what extent. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metrics (but MSE).

Table 5.10 displays the metrics for a comparison on the stationarity of noises in **DS3**. One can observe that there is no significant internal disparity for all models

in terms of STOI and PESQ in regard to the stationarity of noises — the differences between the absolute values being in their variation range. Therefore, contrarily to our expectations, the non-stationarity of a noise doesn't seem to make the learning more difficult for a network.

5.4.3.3 Impact of noise frequency range

As for **DS2**, we conduct here a discriminative analysis of our models based on their performances regarding noise frequency range.

As shown in Table 5.11, **DS3** seems to balance the performances of the models obtained on narrow-band and broad-band noises, compared to **DS2**.

One can even see a trend inversion for SQ, for which most of the metrics take values superior to 1, meaning that the networks — notably including `conv-rnn` (baseline) — perform of equal quality on narrow-band noises and broad-band noises. Across all metrics, SAR remains the most affected metric but has relatively improved much, with now a difference of 11 points between broad- and narrow-band noises (against 32 points on **DS2**).

	mse	sir	sar	sdr	stoi	pesq
conv-rnn	1.00	1.01	0.91	0.93	1.00	**1.03**
conv-gru	1.03	**1.06**	0.91	0.95	1.00	1.01
conv-tcn	1.00	1.01	0.90	0.91	**0.99**	1.00
conv-ae	**1.05**	0.97	**0.89**	**0.89**	**0.99**	1.00

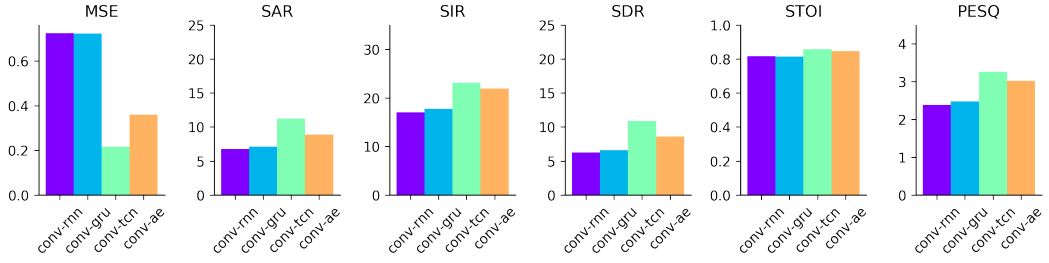
Table 5.11: Ratio of performances comparing narrow-band and broad-band noises on dataset **DS3** for SNR = 5 dB and data representation `db`. But for MSE, ratios bigger than 1 feature a better performance on narrow-band noises, and the value itself indicates to what extent. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metrics (but MSE).

5.4.3.4 Impact of usage in training

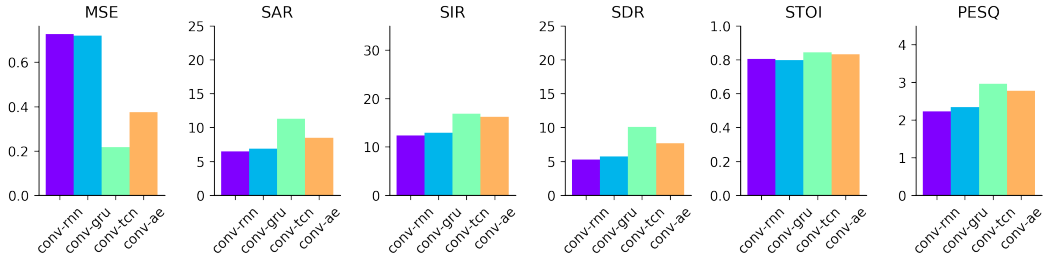
As for **DS2**, we conduct here a discriminative analysis of our models based on their performances on noises either seen or unseen during the training.

Once again, seen noises are better processed than unseen noises, as illustrated in Figure 5.9 and in Table 5.12. Let notice that SIR is once again clearly the most altered metric with a gain of 38 points (calculated in average on the models) for seen noises relatively to unseen noises. This makes a difference of 11 points between the two datasets, and hints for a general improvement in the capability of the networks to generalize to new data.

When it comes to models, all of them perform now similarly with an averaged gain of 13 points (calculated on all metrics but MSE) for seen noises relatively to



(a) seen noise



(b) unseen noise

Figure 5.9: performance metrics for SNR = 5 db, for different optimized model and baseline (`conv-rnn`), trained on **DS3**

unseen noises, against 18 on **DS2**. It means that each model individually improved its capability to generalize to new data. Remarkably, the baseline has the biggest progress for STOI and PESQ conjointly.

	mse	sir	sar	sdr	stoi	pesq
conv-rnn	1.00	1.38	1.04	**1.18**	1.01	1.07
conv-gru	1.00	1.37	1.03	1.16	**1.02**	1.06
conv-tcn	**0.97**	**1.40**	1.03	1.10	**1.02**	1.11
conv-ae	**0.96**	1.35	**1.05**	1.12	**1.02**	**1.09**

Table 5.12: Ratio of performances comparing seen and unseen noises on dataset **DS3** for SNR = 5 dB and data representation `db`. But for MSE, ratios bigger than 1 feature a better performance on seen noises, and the value itself indicates to what extent. Are highlighted in **bold** the values for most impacted models and are underlined the values for most impacted metrics (but MSE).

Chapter 6

Conclusions

The models resulting from the successive stages of parameter selection, training, and fine-tuning, yielded promising results, and state-of-the-art performances may be achieved with longer training time and a more extensive dataset.

Most notably, the hybrid model based on temporal convolutional networks fared similarly to our baseline (which uses LSTM recurrent units), despite being a feed-forward network. Given the scalability limitations posed by recurrent networks — due to their computation and bandwidth requirements [Cul17] — this is particularly interesting in that it paves the way for alternative approaches that may be employed on embedded devices such as hearing aids.

In their current state, most of our trained models are capable of suppressing substantial amounts of stationary and non-stationary noise, with the artifacts introduced being distinctively audible. These artifacts mostly manifest in the form of distorted, synthetic-sounding voice (similar to ring modulation processing), and are particularly prevalent in models whose training have stopped prematurely.

In general, when considering subjective hearing tests, the noise suppression seems quite aggressive if compared with speech enhancement algorithms fitted into consumer products such a mobile phones. This results in the sporadic emergence of the noise signal, causing a gating effect.

As such, the resulting audio quality is not deemed ripe for use in production, and further investigation is certainly necessary.

6.1 Synthesis

Throughout the project, a number of algorithms and methods pertaining to the fields of deep learning, signal processing, and statistics have been investigated, deployed, and assessed, with the purpose of deriving a state-of-the-art single-channel speech enhancement system.

We conducted a critical analysis of related works and literature within speech enhancement, denoising, and source separation, in order to understand the best practices.

After selecting several interesting approaches to replicate and investigate further, we took a step away from the literature and incorporated novel yet promising techniques — namely TCNs, GRUs, and real/imaginary and other alternative representations. The derived model architectures have then been trained and refined over multiple phases.

Subsequently, an extensive evaluation — using the established quantitative metrics and tools — allowed us to shed light on the behavior, patterns, and performances of the models under investigation. These findings have then been compared to our baseline, evaluated critically, and summarized herein. Moreover, a comprehensive and full-featured framework for processing data, designing and training models, and extracting metrics has been developed and extensively used throughout the project.

6.2 Limitations

While striving to follow the best practices exposed in previous works, this project and its results present several limitations and shortcomings.

In terms of data processing, several papers apply a *room impulse response* (RIR) to the noisy mixture, in order to introduce convolutive noise (as opposed to additive only) and provide more natural-sounding noisy signals [Tu+18; Fak+18]. Others prefer to mix clean speech and recorded noises at a predetermined, natural level, and then extract single utterances and sort them by local SNR, with the purpose of avoiding noisy mixtures that would not be occurring in real-world scenarios [Chr+10].

While applying an RIR is somewhat trivial and has indeed been implemented within the software framework, several other factors must have been considered for it to be used effectively, such as compensating for delay and speaker loudness depending on distance from mic. Therefore, convolutive noise has been ruled out of the project scope.

Furthermore, it is known that A-weighting is a somewhat coarse method for modelling human ear sensitivity, and there exist more sophisticated models that account for non-linear perception of loudness [FHTR14].

As such, our data augmentation doesn't necessarily reflect realistic conditions in terms of nature of noise and SNR levels, which may hinder the effectiveness of the models in real-world applications.

In terms of optimizing each model, `conv-ae` could not be investigated extensively due to the long training time required by it. While introducing intermediate

1×1 convolution layers drastically decreased training time, allowing us to reach satisfactorily performances, it would still be considerably slower than its hybrid counterparts.

When it came to the latter models, in particular for TCN-based ones, learning was hindered by abrupt spikes in the cost function value, which would be difficult to recover from, causing training to halt prematurely. This issue has been extensively investigated by tweaking its hyper-parameters throughout the experiment phases, allowing for a stable model to be constructed and trained. Nevertheless, adjusting its receptive field size further was particularly demanding and somewhat erratic, with subsequent trainings being kept under constant scrutiny.

While the first experimental stages shed light on the effectiveness of various parameter combinations, a more extensive comparison would be necessary to determine the best setup. In fact, due to their inherent differences, a given preprocessing scheme may fare better with a specific model; however, testing all the possible combinations would be unfeasible.

Moreover, discarding the real/imaginary `reim` data representation effectively limits us to speech reconstructions based on noisy signal phase, which sets a bottleneck in the performances.

It is worth noting how, in some cases, the metrics computed for each model fall short of listening tests when it comes to evaluating the perceived intelligibility and quality of the predicted speech. Indeed, extensive listening tests are paramount for an adequate performance assessment.

Finally, as always the case with deep learning techniques — and machine learning in general — a larger amount of controlled data (e.g. comprising anechoic recording only, no silent parts, etc) along with more processing power would have certainly benefited learning and overall performances alike. Similarly, a larger test sample would have increased the confidence in the reported metrics.

6.3 Outline

On top of addressing the aforementioned shortcomings, there are a number of paths that have not been explored but are believed to potentially improve our results.

There exist more sophisticated input features based on spectral representation, such as *log-mel spectrum* or *mel-frequency cepstral coefficients* (MFCC), which are commonly used in music information retrieval and speech processing contexts. The former, in particular, is already used in deep learning speech enhancement systems [Wen+14; HZG17] due to its ability to compress the input space without sacrificing data integrity.

Concerning the devised network architectures, TCN could certainly benefit from further regularization mechanisms to stabilize its gradient.

As a novel deep learning technique which is experiencing rapid growth in tasks concerning sequence data, *attention layers* [Vas+17] may be investigated and adapted for usage in the hybrid models used here.

The cost function used in training may be adapted to consider a windowed (e.g. *hann*, *flat top*, etc) portion of the input fragment rather than a slice — i.e. a subset — of it. Furthermore, it may include several reconstruction terms calculated on different input features [Fu+17]. Similarly, one or more of such terms could be based on one of the evaluation metrics (e.g. SDR, STOI) [Gao+16]. These latter concepts are part of a broad spectrum of techniques called *multi-metrics learning*.

Finally, for the sake of adhering with best practices in the literature, it may be useful to integrate evaluation metrics based on speech recognition — computed using an external tool such at HTK [Noa]. In that regard, some of the most common metrics are *word error rate* (WER) and *equal error rate* (EER) [Zha+18; Fak+18; Tu+18]. These are particularly useful in cases where the speech enhancement system under investigation is meant to act a a front-end for an automatic speech recognition system.

Bibliography

[AA18] Afshine Amidi and Shervine Amidi. *CS 230 - Recurrent Neural Networks Cheatsheet*. 2018. URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks (visited on 05/06/2019).

[Alo+18] Md. Zahangir Alom et al. "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches". In: *CoRR* abs/1803.01164 (2018). arXiv: 1803.01164. URL: http://arxiv.org/abs/1803.01164.

[Bai18] Kolter J. Zico Koltun Vladlen Bai Shaojie. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: *arXiv:1803.01271 [cs]* (Mar. 2018). arXiv: 1803.01271. URL: http://arxiv.org/abs/1803.01271 (visited on 04/17/2019).

[Bol79] Steven Boll. "Suppression of acoustic noise in speech using spectral subtraction". In: *IEEE Transactions on acoustics, speech, and signal processing* 27.2 (1979), pp. 113–120.

[Bus07] Busch. *Heisenberg's uncertainty principle*. Vol. 452. Elsevier, 2007, pp. 155–176.

[Che53] E. Colin Cherry. "Some Experiments on the Recognition of Speech, with One and with Two Ears". In: *The Journal of the Acoustical Society of America* 25.5 (Sept. 1953), pp. 975–979. ISSN: 0001-4966. DOI: 10.1121/1.1907229. URL: https://asa.scitation.org/doi/10.1121/1.1907229 (visited on 04/26/2019).

[Cho+14] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *arXiv:1406.1078 [cs, stat]* (June 2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078 (visited on 05/06/2019).

[Chr+10] Heidi Christensen et al. "The CHiME corpus: a resource and a challenge for computational hearing in multisource environments". In: *INTERSPEECH*. 2010.

[Chu+14] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *arXiv:1412.3555 [cs]* (Dec. 2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555 (visited on 05/06/2019).

[Cnn] *https://github.com/kjw0612/awesome-deep-vision*. URL: https://github.com/kjw0612/awesome-deep-vision.

[Cul17] Eugenio Culurciello. *Computation and memory bandwidth in deep neural networks*. May 2017. URL: https://medium.com/@culurciello/computation-and-memory-bandwidth-in-deep-neural-networks-16cbac63ebd5 (visited on 05/26/2019).

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function". en. In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274 (visited on 04/17/2019).

[DPK96] Torsten Dau, Dirk Püschel, and Armin Kohlrausch. "A quantitative model of the effective signal processing in the auditory system. I. Model structure". In: *The Journal of the Acoustical Society of America* 99.6 (1996), pp. 3615–3622.

[Eng+17] Jesse Engel et al. "Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders". In: *arXiv:1704.01279 [cs]* (Apr. 2017). arXiv: 1704.01279. URL: http://arxiv.org/abs/1704.01279 (visited on 04/27/2019).

[Fak+18] Rasool Fakoor et al. "Constrained Convolutional-Recurrent Networks to Improve Speech Quality with Low Impact on Recognition Accuracy". en. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Calgary, AB: IEEE, Apr. 2018, pp. 3011–3015. ISBN: 978-1-5386-4658-8. DOI: 10.1109/ICASSP.2018.8462042. URL: https://ieeexplore.ieee.org/document/8462042/ (visited on 04/17/2019).

[Far18] Osama S. Faragallah. "Robust noise MKMFCC–SVM automatic speaker identification". In: *International Journal of Speech Technology* 21.2 (June 2018), pp. 185–192. ISSN: 1572-8110. DOI: 10.1007/s10772-018-9494-9. URL: https://doi.org/10.1007/s10772-018-9494-9.

[FHTR14] R. Faventi, H. Hopper, and M. Torrente Rodriguez. "Low power transmission plastic gear trains: which parameters affect the subjective acoustic quality?" en. In: *International Gear Conference 2014: 26th–28th August 2014, Lyon*. Elsevier, 2014, pp. 208–218. ISBN: 978-1-78242-194-8. DOI: 10.1533/9781782421955.208. URL: https://linkinghub.

elsevier . com / retrieve / pii / B9781782421948500252 (visited on 05/25/2019).

[FM33] Harvey Fletcher and W. A. Munson. "Loudness, Its Definition, Measurement and Calculation*". en. In: *Bell System Technical Journal* 12.4 (Oct. 1933), pp. 377–430. ISSN: 00058580. DOI: 10 . 1002 / j . 1538 - 7305 . 1933 . tb00403 . x. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6771028 (visited on 05/24/2019).

[Fu+17] Szu-Wei Fu et al. "Complex spectrogram enhancement by convolutional neural network with multi-metrics learning". In: *arXiv:1704.08504 [cs, stat]* (Apr. 2017). arXiv: 1704.08504. URL: http://arxiv.org/abs/1704.08504 (visited on 04/17/2019).

[Fuk88] Kunihiko Fukushima. "Neocognitron: A hierarchical neural network capable of visual pattern recognition". In: *Neural Networks* 1 (1988), pp. 119–130.

[FZG14] Xue Feng, Yaodong Zhang, and James Glass. "Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition". In: *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2014, pp. 1759–1763.

[Gao+16] Tian Gao et al. "SNR-Based Progressive Learning of Deep Neural Network for Speech Enhancement". In: *INTERSPEECH*. 2016. DOI: 10 . 21437/Interspeech.2016-224.

[GBB] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectier Neural Networks". en. In: (), p. 9.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GK08] Volodya Grancharov and W.Bastiaan Kleijn. "Speech Quality Assessment". en. In: *Springer Handbook of Speech Processing*. Ed. by Jacob Benesty, M. Mohan Sondhi, and Yiteng Arden Huang. Springer Handbooks. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 83–100. ISBN: 978-3-540-49127-9. DOI: 10 . 1007 / 978 - 3 - 540 - 49127 - 9_5. URL: https://doi.org/10.1007/978-3-540-49127-9_5 (visited on 05/25/2019).

[GP17] Emad M. Grais and Mark D. Plumbley. "Single Channel Audio Source Separation using Convolutional Denoising Autoencoders". en. In: *arXiv:1703.08019 [cs]* (Mar. 2017). arXiv: 1703.08019. URL: http://arxiv.org/abs/1703.08019 (visited on 04/17/2019).

[Gre97] Donald D Greenwood. "The Mel Scale's disqualifying bias and a consistency of pitch-difference equisections in 1956 with equal cochlear distances and equal frequency ratios". In: *Hearing Research* 103.1 (1997), pp. 199 –224. ISSN: 0378-5955. DOI: https://doi.org/10.1016/S0378-5955(96)00175-X. URL: http://www.sciencedirect.com/science/article/pii/S037859559600175X.

[Han+15] K. Han et al. "Learning Spectral Mapping for Speech Dereverberation and Denoising". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.6 (June 2015), pp. 982–992. ISSN: 2329-9290. DOI: 10.1109/TASLP.2015.2416653.

[He+15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[Hea+13] Eric W. Healy et al. "An algorithm to improve speech recognition in noise for hearing-impaired listeners". eng. In: *The Journal of the Acoustical Society of America* 134.4 (Oct. 2013), pp. 3029–3038. ISSN: 1520-8524. DOI: 10.1121/1.4820893.

[HG03] Barbara Hammer and Kai Gersmann. "A note on the universal approximation capability of support vector machines". In: *Neural Processing Letters* 17.1 (2003), pp. 43–53.

[Hin+12] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv:1207.0580 [cs]* (July 3, 2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580 (visited on 04/25/2019).

[Hor91] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T (visited on 04/26/2019).

[Hou+17] Jen-Cheng Hou et al. "Audio-Visual Speech Enhancement based on Multimodal Deep Convolutional Neural Network". In: *CoRR* abs/1703.10893 (2017). arXiv: 1703.10893. URL: http://arxiv.org/abs/1703.10893.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507. URL: https://doi.org/10.1126/science.1127647.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". en. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.8.1735. URL: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735 (visited on 05/06/2019).

[HW+06] Kris Hermus, Patrick Wambacq, et al. "A review of signal subspace speech enhancement and its application to noise robust speech recognition". In: *EURASIP Journal on Advances in Signal Processing* 2007.1 (2006), p. 045821.

[HW62] D. H. Hubel and T. N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". en. In: *The Journal of Physiology* 160.1 (Jan. 1962), pp. 106–154. ISSN: 00223751. DOI: 10.1113/jphysiol.1962.sp006837. URL: http://doi.wiley.com/10.1113/jphysiol.1962.sp006837 (visited on 05/09/2019).

[HZG17] Wei-Ning Hsu, Yu Zhang, and James Glass. "Learning Latent Representations for Speech Generation and Transformation". In: *arXiv:1704.04222 [cs, stat]* (Apr. 2017). arXiv: 1704.04222. URL: http://arxiv.org/abs/1704.04222 (visited on 04/17/2019).

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv:1502.03167 [cs]* (Feb. 2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167 (visited on 04/17/2019).

[Itu] *Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs*. https://www.itu.int/itu-t/recommendations/rec.aspx?rec=5374. Accessed: 2019-05-20.

[JT16] Jesper Jensen and Cees H. Taal. "An Algorithm for Predicting the Intelligibility of Speech Masked by Modulated Noise Maskers". en. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.11 (Nov. 2016), pp. 2009–2022. ISSN: 2329-9290, 2329-9304. DOI: 10.1109/TASLP.2016.2585878. URL: http://ieeexplore.ieee.org/document/7539284/ (visited on 05/05/2019).

[Kar] Andrej Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/neural-networks-1/ (visited on 04/28/2019).

[KL11] Gibak Kim and Philipos C. Loizou. "Gain-induced speech distortions and the absence of intelligibility benefit with existing noise-reduction algorithms." In: *The Journal of the Acoustical Society of America* 130 3 (2011), pp. 1581–96.

[Kol18] Morten Kolbæk. "Single-Microphone Speech Enhancement and Separation Using Deep Learning". In: *arXiv:1808.10620 [cs, eess]* (Aug. 2018). arXiv: 1808.10620. URL: http://arxiv.org/abs/1808.10620 (visited on 04/17/2019).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[Kuz+] Boris Kuznetsov et al. "Sound Morphing using Real/Imaginary Spectrogram Representation and Variational Autoencoders". en. In: (), p. 23.

[KZ15] Mike Kayser and Victor Zhong. "Denoising Convolutional Autoencoders for Noisy Speech Recognition". en. In: (2015), p. 6.

[Lea+16] Colin Lea et al. "Temporal Convolutional Networks for Action Segmentation and Detection". In: *CoRR* abs/1611.05267 (2016). arXiv: 1611.05267. URL: http://arxiv.org/abs/1611.05267.

[LeC+98] Yann LeCun et al. "Efficient BackProp". en. In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50. ISBN: 978-3-540-49430-0. DOI: 10.1007/3-540-49430-8_2. URL: https://doi.org/10.1007/3-540-49430-8_2 (visited on 05/07/2019).

[LLH05] Philipos C. Loizou, Arthur Lobo, and Yi Hu. "Subspace algorithms for noise reduction in cochlear implants". In: *The Journal of the Acoustical Society of America* 118.5 (Nov. 2005), pp. 2791–2793. ISSN: 0001-4966. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1343472/ (visited on 05/08/2019).

[Loi11] Philipos C. Loizou. "Speech Quality Assessment". English. In: *Multimedia Analysis, Processing and Communications* 346 (2011), pp. 623–654. ISSN: 1860-949X. DOI: 10.1007/978-3-642-19551-8_23. URL: https://sfx.aub.aau.dk/sfxaub?ctx.

[Loi13] Philipos C. Loizou. *Speech Enhancement : Theory and Practice, Second Edition*. English. 2013. ISBN: 978-1-4665-0421-9.

[LRHW15] Jonathan Le Roux, John R. Hershey, and Felix Weninger. "Deep NMF for speech separation". en. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. South Brisbane, Queensland, Australia: IEEE, Apr. 2015, pp. 66–70. ISBN: 978-1-4673-6997-8. DOI: 10.1109/ICASSP.2015.7177933. URL: http://ieeexplore.ieee.org/document/7177933/ (visited on 05/06/2019).

[Lu+] Xugang Lu et al. "Speech Enhancement Based on Deep Denoising Autoencoder". en. In: (), p. 5.

[Luo+17] Wenjie Luo et al. "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks". In: *CoRR* abs/1701.04128 (2017). arXiv: 1701.04128. URL: http://arxiv.org/abs/1701.04128.

[MB88] Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*. Vol. 84. M. Dekker New York, 1988.

[McF+15] Brian McFee et al. "librosa: Audio and Music Signal Analysis in Python". In: 2015. DOI: 10.25080/majora-7b98e3ed-003.

[Mer76] Paul Mermelstein. "Distance measures for speech recognition, psychological and instrumental". In: *Pattern recognition and artificial intelligence* 116 (1976), pp. 374–388.

[MG08] T. U. Munich and Germany. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks". In: 2008.

[NAK16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning Convolutional Neural Networks for Graphs". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 2016, pp. 2014–2023. URL: http://jmlr.org/proceedings/papers/v48/niepert16.html.

[Nga] Andrew Ng. *C2W3L06 Why does Batch Norm Work*. URL: https://www.youtube.com/watch?v=nUUqwaxLnWs (visited on 05/20/2019).

[Ngb] Andrew Ng. *CS230 Deep Learning*. URL: https://cs230.stanford.edu/ (visited on 05/06/2019).

[NH10] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[Nie15] Michael A. Nielsen. "Neural Networks and Deep Learning". en. In: (2015). URL: http://neuralnetworksanddeeplearning.com (visited on 05/08/2019).

[Noa] *HTK Speech Recognition Toolkit*. URL: http://htk.eng.cam.ac.uk/ (visited on 05/26/2019).

[Oli07] Travis E. Oliphant. "Python for Scientific Computing". In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.58. URL: http://ieeexplore.ieee.org/document/4160250/ (visited on 04/17/2019).

[Oor+16] Aaron van den Oord et al. "WaveNet: A Generative Model for Raw Audio". In: *arXiv:1609.03499 [cs]* (Sept. 2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499 (visited on 04/17/2019).

[PCCB97] H P. Combrinck and E C. Botha. "On The Mel-scaled Cepstrum". In: (Oct. 1997).

[PL16] Se Rim Park and Jinwon Lee. "A Fully Convolutional Neural Network for Speech Enhancement". en. In: (Sept. 2016). URL: https://arxiv.org/abs/1609.07132v1 (visited on 04/20/2019).

[PLS16] J. Pons, T. Lidy, and X. Serra. "Experimenting with musically motivated convolutional neural networks". In: *2016 14th International Workshop on Content-Based Multimedia Indexing (CBMI)*. June 2016, pp. 1–6. DOI: 10.1109/CBMI.2016.7500246.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0 (visited on 05/06/2019).

[Rix+01] A.W. Rix et al. "Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs". en. In: *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*. Vol. 2. Salt Lake City, UT, USA: IEEE, 2001, pp. 749–752. ISBN: 978-0-7803-7041-8. DOI: 10.1109/ICASSP.2001.941023. URL: http://ieeexplore.ieee.org/document/941023/ (visited on 04/25/2019).

[Rud16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". en. In: (Sept. 2016). URL: https://arxiv.org/abs/1609.04747v2 (visited on 05/02/2019).

[Shi+15] Xingjian Shi et al. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *CoRR* abs/1506.04214 (2015). arXiv: 1506.04214. URL: http://arxiv.org/abs/1506.04214.

[Smi03] Julius O Smith. *Mathematics of the discrete Fourier transform (DFT): with music and audio applicaitons*. English. OCLC: 60379068. Place of publication not identified]; Stanford, Calif.: W3K Pub. ; Center for Computer Research in Music and Acoustics, 2003. ISBN: 978-0-9745607-0-0.

[Sow+18] Daouda Sow et al. "A sequential guiding network with attention for image captioning". In: *arXiv:1811.00228 [cs]* (Nov. 2018). arXiv: 1811.00228. URL: http://arxiv.org/abs/1811.00228 (visited on 05/14/2019).

[Sri+14a] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (visited on 04/17/2019).

[Sri+14b] Rupesh Kumar Srivastava et al. "Understanding Locally Competitive Networks". In: *arXiv:1410.1165 [cs]* (Oct. 2014). arXiv: 1410.1165. URL: http://arxiv.org/abs/1410.1165 (visited on 04/17/2019).

[SWT14] Yi Sun, Xiaogang Wang, and Xiaoou Tang. "Deeply learned face representations are sparse, selective, and robust". In: *arXiv:1412.1265 [cs]* (Dec. 2014). arXiv: 1412.1265. URL: http://arxiv.org/abs/1412.1265 (visited on 05/13/2019).

[SZ14] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (Sept. 2014).

[Sze+14] Christian Szegedy et al. "Going Deeper with Convolutions". In: *arXiv:1409.4842 [cs]* (Sept. 2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842 (visited on 05/25/2019).

[Taa+10] Cees H. Taal et al. "A short-time objective intelligibility measure for time-frequency weighted noisy speech". In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. Dallas, TX, USA: IEEE, 2010, pp. 4214–4217. ISBN: 978-1-4244-4295-9. DOI: 10.1109/ICASSP.2010.5495701. URL: http://ieeexplore.ieee.org/document/5495701/ (visited on 04/17/2019).

[Taa+11] C. H. Taal et al. "An Algorithm for Intelligibility Prediction of Time–Frequency Weighted Noisy Speech". In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.7 (Sept. 2011), pp. 2125–2136. ISSN: 1558-7916. DOI: 10.1109/TASL.2011.2114881.

[TIV13] Joachim Thiemann, Nobutaka Ito, and Emmanuel Vincent. *Demand: A Collection Of Multi-Channel Recordings Of Acoustic Noise In Diverse Environments*. en. type: dataset. June 2013. DOI: 10.5281/zenodo.1227121. URL: https://zenodo.org/record/1227121 (visited on 04/22/2019).

[Tu+18] Yan-Hui Tu et al. "A Hybrid Approach to Combining Conventional and Deep Learning Techniques for Single-channel Speech Enhancement and Recognition". en-US. In: (Feb. 2018). URL: https://www.microsoft.com/en-us/research/publication/a-hybrid-approach-to-combining-conventional-and-deep-learning-techniques-for-single-channel-speech-enhancement-and-recognition/ (visited on 04/17/2019).

[Vas+17] Ashish Vaswani et al. "Attention Is All You Need". In: *arXiv:1706.03762 [cs]* (June 2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on 05/26/2019).

[VGF06] E. Vincent, R. Gribonval, and C. Fevotte. "Performance measurement in blind audio source separation". In: *IEEE Transactions on Audio, Speech and Language Processing* 14.4 (July 2006), pp. 1462–1469. ISSN: 1558-7916. DOI: 10.1109/TSA.2005.858005. URL: http://ieeexplore.ieee.org/document/1643671/ (visited on 04/17/2019).

[WC17] DeLiang Wang and Jitong Chen. "Supervised Speech Separation Based on Deep Learning: An Overview". In: *arXiv:1708.07524 [cs]* (Aug. 2017). arXiv: 1708.07524. URL: http://arxiv.org/abs/1708.07524 (visited on 04/17/2019).

[Wen+14] Felix Weninger et al. "Discriminatively trained recurrent neural networks for single-channel speech separation". en. In: *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. Atlanta, GA, USA: IEEE, Dec. 2014, pp. 577–581. ISBN: 978-1-4799-7088-9. DOI: 10.1109/GlobalSIP.2014.7032183. URL: http://ieeexplore.ieee.org/document/7032183/ (visited on 04/17/2019).

[WL82] David L. Wang and Jae S. Lim. "The unimportance of phase in speech enhancement". English. In: 30 (1982), pp. 679–681. ISSN: 0096-3518. DOI: 10.1109/TASSP.1982.1163920. URL: https://sfx.aub.aau.dk/sfxaub?ctx.

[Xu+14] Yong Xu et al. "An Experimental Study on Speech Enhancement Based on Deep Neural Networks". en. In: *IEEE Signal Processing Letters* 21.1 (Jan. 2014), pp. 65–68. ISSN: 1070-9908, 1558-2361. DOI: 10.1109/LSP.2013.2291240. URL: http://ieeexplore.ieee.org/document/6665000/ (visited on 05/06/2019).

[YD13] Yuxuan Wang and DeLiang Wang. "Towards Scaling Up Classification-Based Speech Separation". en. In: *IEEE Transactions on Audio, Speech, and Language Processing* 21.7 (July 2013), pp. 1381–1390. ISSN: 1558-7916, 1558-7924. DOI: 10.1109/TASL.2013.2250961. URL: http://ieeexplore.ieee.org/document/6473841/ (visited on 05/06/2019).

[YK15] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *arXiv preprint arXiv:1511.07122* (2015).

[Zha+17] Zizhao Zhang et al. "MDNet: A Semantically and Visually Interpretable Medical Image Diagnosis Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. 2017, pp. 3549–3557. DOI: 10.1109/CVPR.2017.378. URL: https://doi.org/10.1109/CVPR.2017.378.

[Zha+18] Han Zhao et al. "Convolutional-Recurrent Neural Networks for Speech Enhancement". In: *arXiv:1805.00579 [cs, eess]* (May 2018). arXiv: 1805.00579. URL: http://arxiv.org/abs/1805.00579 (visited on 04/17/2019).

[Zhe+18] Kai Zhen et al. "On Psychoacoustically Weighted Cost Functions Towards Resource-Efficient Deep Neural Networks for Speech Denoising". In: *arXiv:1801.09774 [cs, eess]* (Jan. 2018). arXiv: 1801.09774. URL: http://arxiv.org/abs/1801.09774 (visited on 04/17/2019).

[ZL18] Ruibin Zhang and Jingen Liu. "An Improved Multi-band Spectral Subtraction using Mel-scale". English. In: 131 (2018), pp. 779–785. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.04.324. URL: https://sfx.aub.aau.dk/sfxaub?ctx.

[ZWL18] Lei Zhang, Shuai Wang, and Bing Liu. "Deep Learning for Sentiment Analysis : A Survey". In: *arXiv:1801.07883 [cs, stat]* (Jan. 2018). arXiv: 1801.07883. URL: http://arxiv.org/abs/1801.07883 (visited on 05/14/2019).

[Lec+98] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[Ste37] S. S. Stevens. "A Scale for the Measurement of the Psychological Magnitude Pitch". In: *Acoustical Society of America Journal* 8 (1937), p. 185. DOI: 10.1121/1.1915893.

Appendix A

Attachments description

Any further data that may be useful for evaluating the project and its findings will be delivered in the form of compressed archive. The following sections describe their content and ways to access it.

A.1 Source

This directory contains the project source code, as described in Section 4.2. In order to run any of its content, a Python environment must be set up, using either `conda` or `pipenv`, and the provided dependency files. The content of this directory is also available online at `https://github.com/Christie22/SingleChannelDenoising`

A.2 Video

This directory hosts the project video presentation.

A.3 Samples

This directory contains audio samples generated by each of the models described in the training and fine-tuning experimental stages. Each sample is 30 seconds in length, and is composed of a clean (unprocessed), a noisy, and a predicted (cleaned) part. All noisy mixtures are generated using speech from the test set (not used during training) and are mixed at 5 dB of SNR.

Within the folder, the following structure is observed:

```
dataset/processing/model_name/filename
```

A.4 Analyses

This directory contains various analyses performed on the data and models, which were worthy of sharing. Of particular interest, a characterization of the real-world noises used, and the complete list of plots derived from the result script, i.e. metrics evaluation.

A.5 Models

This directory contains detailed graphs, parameters descriptions, and summary for the models described in the training and fine-tuning experimental stages. Within the folder, the following structure is observed:

```
dataset/processing/model_name/filename
```

Due to storage constraints, trained model files have not been included.

A.6 Results

This directory contains the data generated by the result script, i.e. metrics evaluation, for each of the models described in the training and fine-tuning experimental stages. The data is stored in the form of serialized Python pandas object, using the standard pickle library, and can be easily accessed using the `pandas.read_pickle()` function. An aggregate visualization of the data in this folder is also available in the analyses.

A.7 Logs

This directory contains log files documenting most of the model training sessions, in JSON format. Each log file may contain more than one training session, and stores information regarding dataset, data processing, model parameterization, optimizer, etc. Due to the vast amount of data, it is recommended to use an online JSON viewer such as `http://jsonviewer.stack.hu/`.