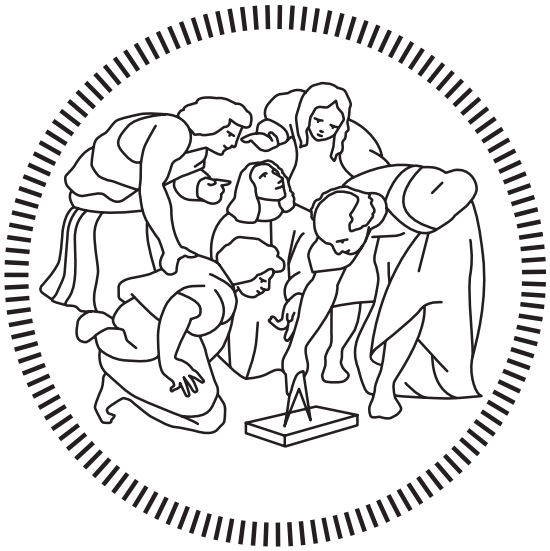
Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

Master of Science – Music and Acoustic Engineering



Speech separation and diarization with microphone arrays: integrating tasks to improve performance

Supervisor
Prof. Fabio ANTONACCI

Co-Supervisor
PhD Federico BORRA

Co-Supervisor
PhD St. Clara BORRELLI

Candidate
Matteo SCERBO – 899823

Academic Year 2020 – 2021

Abstract

In recent years, great progress has been made in the fields of Diarization and Speech Separation. In particular, many works have proposed new ways to exploit the shared aspects of these tasks for the advantage of both research areas. The goal of this thesis is to further investigate the integration of the mentioned fields. This was tackled by training a unified Deep Neural Network to extract clean speech features, useful towards separation as well as identification.

Our system for combined diarization and separation of speakers is composed of three independent modules. First, a microphone array is used to profit from spatial information in the sound signals, and a combination of direction-of-arrival estimation and beamforming achieves a preliminary isolation of each concurrently active speaker. The beamformed signals are then individually processed by a Deep Neural Network that jointly produces two outputs: a set of embeddings that characterize the target speaker, and a separation mask to be applied to the beamformed signal. Lastly, clustering is performed on the speaker embeddings to identify the speaker related to each embedding. The masked signals are assigned to identity-exclusive output channels based on the identification clustering.

We designed the Neural Network with feature sharing in mind. We chose a U-Net architecture, with the speaker embeddings being extracted from the bottleneck and the separation masks being the result of the decoder branch. For the purpose of validating the proposed method, as well as training the designed network, we created a dataset of simulated conversations in reverberant rooms. Results showed excellent diarization accuracy and significantly higher separation quality with respect to the unprocessed beamformer signal.

Sommario

I campi di *Diarization* — determinare chi ha parlato quando, in una conversazione — e *Speech Separation* — isolare una voce da rumori e altre voci — hanno visto grandi progressi negli anni recenti. In particolare, molte pubblicazioni hanno proposto nuovi modi di sfruttare gli aspetti condivisi di queste applicazioni, avanzando entrambi i campi di ricerca. Lo scopo di questa tesi è realizzare una completa integrazione di tali campi. Per questo scopo, abbiamo progettato e allenato una singola rete neurale che si occupi di ambo i compiti, estraendo dal segnale sonoro caratteristiche del parlato utili per l'identificazione come per l'estrazione.

Il nostro sistema, che effettua simultaneamente separazione e identificazione vocale, è composto da tre moduli indipendenti. Per prima cosa, tramite un array di microfoni, viene stimata la direzione d'arrivo delle voci di ogni interlocutore attivo in un determinato istante. Viene quindi applicato un beamformer verso ognuno di essi, compiendo una separazione preliminare. I segnali dei beamformer sono successivamente processati da una rete neurale, la quale produce due output: un embedding che caratterizza l'identità dell'interlocutore, e una maschera di separazione da applicare al segnale del beamformer per isolarne la voce. Infine, viene eseguito un clustering degli embedding per associare ognuno alla relativa identità, e i segnali mascherati sono assegnati a canali di output diversi in base all'identificazione.

La rete neurale è stata progettata tenendo a mente la condivisione delle feature tra i due compiti, e per questo è stata scelta un'architettura di tipo U-Net. Gli embedding vengono estratti dal collo di bottiglia, mentre le maschere di separazione sono l'output del decoder. Inoltre è stato prodotto un dataset di conversazioni simulate in stanze riverberanti, con il quale è stato allenato e testato il sistema. I risultati mostrano efficacia di identificazione eccellente e qualità di separazione significativamente superiore rispetto al segnale del beamformer.

Contents

Abstract ii

Sommario iii

Contents v

List of Figures vii

List of Tables viii

1 Introduction 1

2 Background and state-of-the-art 6

2.1 Background . . . 7

2.1.1 Beamforming . . . 7

2.1.2 Deep Neural Networks . . . 10

2.2 State-of-the-art . . . 18

2.2.1 Speaker separation . . . 18

2.2.2 Speaker diarization . . . 19

2.2.3 Integrating tasks . . . 20

3 Architecture 23

3.1 Problem formulation . . . 24

3.2 System layout . . . 25

3.2.1 Windowing / Overlap-add . . . 26

3.2.2 Speech detection and DOA evaluation . . . 26

3.2.3 Beamforming . . . 27

3.2.4 Neural network . . . 27

3.2.5 Clustering and channel selection . . . 33

4 Experimental setup 35

4.1 Dataset . . . 35

4.1.1 Room simulation . . . 35

4.1.2 Conversation simulation 37
4.2 System setup and training 39
4.2.1 Windowing, beamforming, and clustering 39
4.2.2 Training batch construction 40
4.2.3 Training parameters 42
4.3 Evaluation metrics 42
4.3.1 Identification performance 42
4.3.2 Separation performance 43

5 Results **44**
5.1 General results 44
5.2 Reverb-specific results 46
5.3 Overlap-specific results 51
5.4 Lightweight system 53

6 Conclusions and future work **55**
6.1 Conclusions 55
6.2 Future work 56

Acronyms **59**

Bibliography **62**

List of Figures

Figure 2.1 Some examples of microphone array configurations[1]. 7
Figure 2.2 A linear microphone array. 9
Figure 2.3 Frequency and spatial response of a delay-and-sum beamformer. Created using code from [9]. 11
Figure 2.4 The layer structure of a Deep Neural Network (DNN). 11
Figure 2.5 An artificial neuron. 12
Figure 2.6 An example of the convolution operation[13]. 14
Figure 2.7 An example of 2×2 pooling[14]. 15
Figure 2.8 A basic Convolutional Neural Network (CNN): blocks are built from series of convolution+nonlinearity followed by pooling. 15
Figure 2.9 The general architecture of a bottleneck autoencoder. 16
Figure 2.10 A Unet DNN. 17

Figure 3.1 An example configuration of the problem. In this case there are 4 sources, 6 microphones, and a source of noise. The first speaker's Direction Of Arrival (DOA) θ_1 is displayed. 24
Figure 3.2 General layout of the system. 26
Figure 3.3 The proposed network. A larger version of the image can be found in the appendix, as Figure 1. 28
Figure 3.4 Two examples of Ideal Ratio Mask (IRM). Overlaps and reverb are usually both present, here they were considered separately for clarity. 31

Figure 4.1 Two examples of simulated room shapes and a Room Impulse Response (RIR) from each. Crosses denote the positions of microphones, other shapes represent possible sources. 36
Figure 4.2 Percentages of the training data representing a lone speaker, a 2-speaker overlap, or a 3-speaker overlap. 39

Figure 5.1 Diarization Error Rate measured over all testing data. 45
Figure 5.2 Perceptual evaluation of speech quality (PESQ) measured over all testing data. 45
Figure 5.3 Word Error Rate measured over all testing data. 46

Figure 5.4 Comparison between ground-truth clean speech signals and signals isolated by the network's output masks. 47
Figure 5.5 Comparison between ground-truth IRMs and the predictions made by the network. 48
Figure 5.6 Diarization Error Rate for different wall materials. 49
Figure 5.7 PESQ for different wall materials. 50
Figure 5.8 Word Error Rate for different wall materials. 50
Figure 5.9 Diarization Error Rate for different overlap times. 52
Figure 5.10 PESQ for different overlap times. 52
Figure 5.11 Word Error Rate for different overlap times. 53
Figure 5.12 Performance comparison between full system and lightweight variant. 54

Figure 1 The proposed network. 58

List of Tables

Table 4.1 Different wall materials' absorption coefficients. 37
Table 4.2 Utterance counts for the different dataset partitions. 37

Chapter 1

Introduction

The objective of this thesis is to create a system with the capability of isolating and enhancing the voices of different — possibly simultaneous — speakers and assigning the isolated voices to the relative identities. This entails a couple of problems related to speech analysis which, as we will discuss, are deeply entangled.

Speech analysis and enhancement plays an ever more central role in Signal processing research. Microphones' decreasing sizes and costs make it easier and easier to embed them in all kinds of smart devices, and because of this, Human-Computer Interaction (HCI) is now gravitating towards speech for many everyday applications. There are countless examples of such applications: many smarthphones now offer voice interfaces, as do some home appliances; real-time speech processing enables automatic close-captioning and translation, transcription of any kind of meeting or conversation, and so on. The many problems faced by these kinds of applications constitute the many research fields of speech analysis. These can be Speech recognition (transcribing) as well as Speaker recognition (identifying), Denoising (removing undesired disturbances) as well as Speaker separation (separating the signals of concurrent speakers), and many more. All of these problems, which once could only rely on specifically designed deterministic approaches, have recently been greatly boosted by advancements in Machine Learning (ML).
The two problems that are key to the objective of this thesis are Speaker separation and Diarization.

The field of Speaker separation deals with isolating a voice from a mix of other voices and/or noise. It is of great use whenever a speech signal needs to be analysed, interpreted, and/or recorded. Some example applications are speech recognition, meeting transcription and enhancement, hearing aids, music information retrieval, and speech enhancement [4].

The field of Diarization deals with determining who spoke and when. Use cases of separation very often overlap with those of Diarization: in fact, Diarization applications also include speaker recognition and meeting transcription, as well as speech translation [17]. Furthermore, solutions to these two problems often rely on processing the same features, and using the same methods.

Seeing as these fields go hand in hand, being both required in the same situations and solved with the same tools, it is natural to investigate ways to integrate the two tasks, in order to save time and resources.

The speech analysis research community has focused on combining these subjects in recent times. Sarkar et al.[20] combined Diarization and Speech recognition, forgoing an explicit audio signal isolation step in favor of a neural network that directly outputs a transcription. Zhang et al.[30] explored the interaction of Sound dereverberation and Speaker identification by constraining a neural network with a bottleneck layer, subjected to a secondary loss function. Several works enhanced Speaker separation using extracted speaker features, like Wang et al.[24] and Žmolíková et al.[31], both of which make use of pre-enrolled speech signals pertaining to the speakers to be isolated.

There were even some works proposed during the writing of this thesis, highlighting the activity of this area of research: Han et al.[10] improve on Wang et al.[24] by not requiring pre-enrolled examples, instead deriving the reference embeddings from non-overlapped portions of the processed conversation; Zeghidour and Grangier[29] enhance Speaker separation by performing a Speaker identification clustering step first, and providing the results to a separation network.

While examining all these works, we singled out four recurring advantages. Each considered work exhibits some of these advangates, but none of them combines all four. The first consideration is that performing Speaker separation and Diarization in a joint fashion is not only efficient in terms of processing time, but also beneficial to the performance of both tasks. In other words, features that were extracted from a signal for the purpose of identification can aid in separation, and vice versa. Many propose, for example, to extract speaker embeddings meant for identification, and provide them to a separation module. We posit that a deeper and more thorough feature sharing would be better — more on that later. A second consideration is that incorporating a spatial processing step — based on a microphone array — in a neural network makes it heavily dependent on the array's configurations and characteristics. Keeping the spatial processing step separate from the neural network, on the other hand, allows the network to be independent from the array and thus able to be employed independently from hardware availability, with minimal re-training. The

third consideration is related to a problem inherent to speaker separation. If a network is asked to provide an output related to each speaker in a mix, the permutation of the outputs being unpredictable is an issue during the training process. While many works opt to tackle this *permutation problem* with a special Permutation Invariant Training (PIT) regimen, some show that it is possible to instruct a network as to which specific speakers to extract, thereby bypassing the problem entirely. The final consideration is related to system requirements: some systems require pre-enrolled speech recordings from the involved speakers. It is clearly advantageous for a system to operate without such pre-existing knowledge of the operating conditions. We propose a novel method which includes all mentioned enhancement possibilities, something no method in our knowledge had achieved. Features of high- and low-level, related to both the target speech and disturbances, are shared through a U-Net architecture, as will be detailed. We process spatial information in a separate module to ensure hardware independence. Said spatial information is leveraged to implicitly instruct a network as to which speaker to extract for any given call, thus bypassing the permutation problem without need for pre-enrolled data.

Our method isolates different speakers' voices and assigns each to an output stream related to the speaker's identity. It revolves around a single neural network designed to simultaneously isolate a target speaker's voice and produce an identifying embedding of its features. The neural network is preceded by a beamformer and followed by a clustering module. The beamformer, based on a linear microphone array, performs a preliminary separation. It is employed for each distinct speaker, so that the neural network only isolates and identifies one speaker at a time. The clustering module works on the speaker embeddings produced by the network, classifying signals based on speaker identity. The beamformer's independence keeps the neural network independent from the microphone array geometry and beamforming algorithm, allowing application on different hardware with little to no need for retraining. The clustering module's independence opens a possibility for more advanced clustering procedures, which might allow real-time processing and/or operation with an unknown number of involved speakers.

The designed neural network takes the magnitude spectrum of each beamformer signal as input, and employs a U-Net encoder+decoder approach. It produces a separation mask and a speaker embedding, both related to the speaker towards which the beamformer is pointed. The encoder branch extracts the target speaker's features with convolutional layers of gradually decreasing sizes, followed by a series of fully-connected layers which produce the speaker embeddings. The decoder branch uses

skipped connections from several levels of the encoder to create a separation mask to be applied to the beamformer spectrum.

The two network outputs, masks and embeddings, are respectively used for isolation and diarization. To produce the isolated signals, the masks are applied to the relative beamformer spectra, filtering out any undesired disturbance, interference, noise, or reverb. To assign the isolated signals to the correct output channels, producing the diarized audio stream, the speaker embeddings first go through spectral clustering to attain the guessed identities. They are then grouped into utterances based on contiguity, and the most prominent guessed identity of each group is picked for all embeddings of that group, controlling the choice of the output stream for all signals related to them.

All aspects of the system were tailored to address the recurring shortcomings of previous works. The U-Net architecture was chosen to maximize sharing of features of high- and low-level, related to both the target speech and disturbances. The system's modularity ensures hardware independence and enables customization to fit the deployment conditions. Spatial information is leveraged to bypass the permutation problem without need for pre-enrolled data.

For the purpose of training and testing this system, we created a dataset of simulated conversations in reverberant rooms. The dataset was created by first simulating the RIRs of a set of rooms with different shapes and wall materials, and subsequently employing clean speech from the datasets TIMIT[7] and LibriSpeech[16] to generate mock conversations in the simulated rooms.

To sum up, the designed system's benefits are manifold. The network that can handle virtually unlimited simultaneous speakers, without encountering the permutation problem facing many other systems. With the use of a modular structure, each module is relatively independent and can be individually improved and/or adapted to the available hardware or for the situation at hand. Shared neural features, at both high and low levels and regarding both the target speech and disturbances, maximize the mutual benefit of the handled tasks. Lastly, this system has no requirements for previous recordings from involved speakers.

The system's performance was evaluated separately for identification and isolation, and showed clear improvement in quality over the beamformer's preliminary isolation, as well as excellent identification accuracy.

We begin by introducing the basic concepts of beamforming, neural networks, and the state-of-the-art for the relevant tasks in Chapter 2. Chapter 3 establishes a formulation of the contemplated problem, and presents the system that we designed to handle it. Next, Chapter 4 describes the creation of our dataset as well as the

process used to train and test the system, and Chapter 5 lays out the results of our experiments. Finally, in Chapter 6 we draw some conclusions on our work, and make some suggestions for future works.

Chapter 2

Background and state-of-the-art

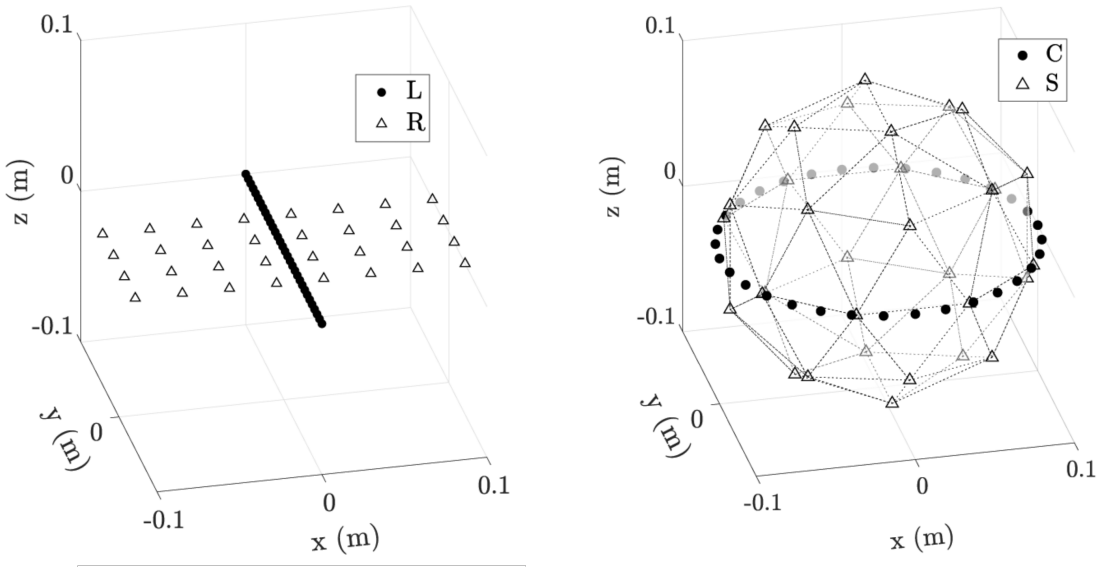
In Section 2.1, we will briefly introduce some of the techniques and methods upon which our system is based. These techniques are split into two main subjects:

- *Beamforming*, the use of multiple microphones to exploit spatial features of sound[23];
- *Machine Learning (ML)*, the use of self-improving algorithms that automatically learn the correct behaviour from data[8].

In Section 2.2, we present some works that tackled the subjects of our interest, briefly introducing their basic principles, features, and shortcomings. Again, these are split into categories, depending on which is the main focus of the work:

- *Speaker separation*, or isolating the sound of a speaker's voice from other voices, background noise, and/or reverb — also known as speaker isolation, or denoising/dereverberating in the case of a lone voice;
- *Speaker diarization*, or determining the time intervals of activity for one or more speakers (who spoke when);
- *Integrating tasks*, meaning the work offers some insights on methods and advantages related to combining tasks.

Lastly, we will make some considerations on the integration of the tasks we introduced, drawing inspiration from some of the works above.



(a) Linear (L) and Rectangular (R) arrays.

(b) Circular (C) and Spherical (S) arrays.

Figure 2.1. Some examples of microphone array configurations[1].

2.1 Background

2.1.1 Beamforming

By combining the signals of multiple microphones placed in different positions, it is possible to enhance or dampen sounds coming from chosen directions. Such a set of microphones is known as a *microphone array* — some examples are shown in Figure 2.1 — and the most widely applied technique for directional sound filtering is called *beamforming*. In order to explain how beamforming works, we first need to introduce some concepts regarding microphone arrays in general.

Microphone arrays

Let's start by considering the signal received at each microphone when there is a single sound source. The output of the m^{th} microphone can be modeled[23] as

$$y_m(t) = h_m(t) * s(t - \tau_m) + e_m(t), \tag{2.1}$$

where:

- $s(t)$ is the source signal, as measured at a reference point;
- $h_m(t)$ is the impulse response from the reference point to the m^{th} microphone's position;
- τ_m is the propagation delay between the reference point and the m^{th} microphone;

- $e_m(t)$ is the additive noise at the m^{th} microphone;
- the operator $*$ represents a convolution operation.

Let's further simplify the problem by assuming the signal to be narrow-band[1], centered around frequency ω_c. We can then represent this sinusoidal-like signal as a function of its amplitude and phase:

$$S(t) = \alpha(t)e^{j\phi(t)}. \tag{2.2}$$

Using the same notation, the microphone signals Equation (2.1) can be represented as

$$Y_m(t) = H_m(\omega_c)S(t)e^{-j\omega_c\tau_m} + E_m(t) \tag{2.3}$$

where $H_m(\omega_c)$ is the frequency response of the m^{th} microphone evaluated at frequency ω_c.

Adopting a vector representation, we can derive the complete array model for a single narrow-band source:

$$\mathbf{y}(t) = \mathbf{a}(\theta)S(t) + \mathbf{e}(t), \tag{2.4}$$

where

$$\mathbf{y}(t) = \begin{bmatrix} Y_1(t) \\ Y_2(t) \\ \vdots \\ Y_M(t) \end{bmatrix}, \quad \mathbf{a}(\theta) = \begin{bmatrix} H_1(\omega_c)e^{-j\omega_c\tau_1} \\ H_2(\omega_c)e^{-j\omega_c\tau_2} \\ \vdots \\ H_M(\omega_c)e^{-j\omega_c\tau_M} \end{bmatrix}, \quad \mathbf{e}(t) = \begin{bmatrix} E_1(t) \\ E_2(t) \\ \vdots \\ E_M(t) \end{bmatrix}. \tag{2.5}$$

Let's now consider the case of a linear mic array as shown in Figure 2.2 and a narrow-band sound. The array is composed of M microphones in a straight line with distance d between each of them. We will characterize the sound source by the angle θ between its origin point and the axis perpendicular to the array. We will also assume that the source is far enough from the array that its sound waves can be approximated as a planar wave when they reach the microphones. Taking the first microphone of the array as the reference point, the propagation delay of the m^{th} microphone's signal w.r.t. the reference can be expressed as

$$\tau_m = (m-1)\frac{d\sin(\theta)}{c}, \tag{2.6}$$

[1]Broad-band sources can be represented as a sum of narrow-band sources, and thanks to the superposition principle, all of the (linear) solutions we find can be applied to the case of multiple broad-band sources by simple addition.

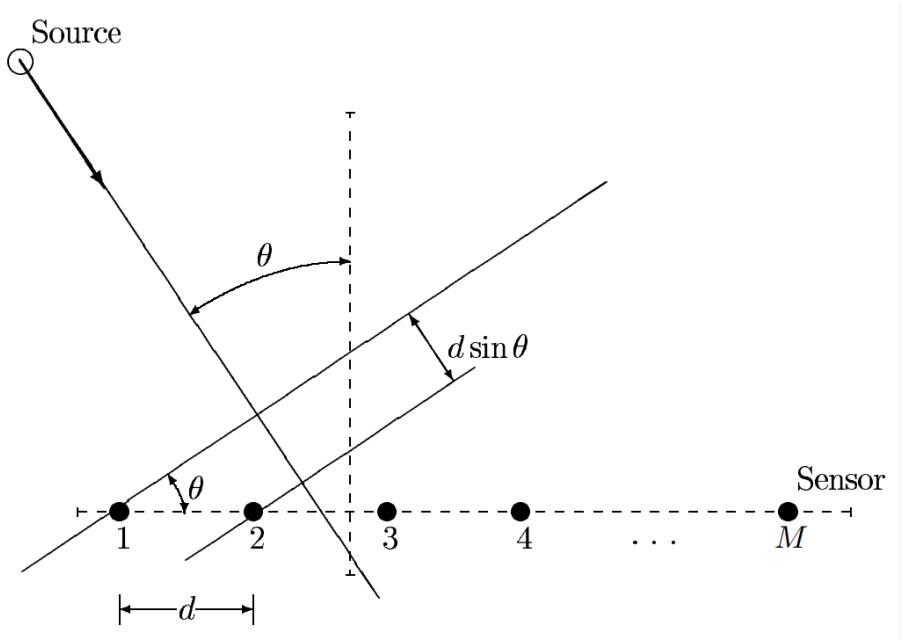


Figure 2.2. A linear microphone array.

where $c \approx 340\,\mathrm{m/s}$ is the speed of sound. All the microphones are assumed to be identical and omni-directional, and we are operating in ideal conditions (no reverb, sources are stationary, ...) therefore we can assume $H_1(\omega_c) = H_2(\omega_c) = \cdots = H_M(\omega_c) = 1$.

We can then simplify the propagation vector $\mathbf{a}(\theta)$ from Equation (2.5):

$$\mathbf{a}(\theta) = \begin{bmatrix} 1 \\ e^{-j\omega_c \frac{d\sin(\theta)}{c}} \\ \vdots \\ e^{-j\omega_c \frac{(M-1)d\sin(\theta)}{c}} \end{bmatrix}. \tag{2.7}$$

The elements of this vector can be seen as samples of a complex sinusoid,

$$e^{-j\omega_c \frac{d\sin(\theta)}{c} m}, \quad m = 0, 1, \ldots, M-1 \tag{2.8}$$

This makes sense, since at a given moment in time, the equally-spaced microphones "sample" the sound wave in space. For this reason the frequency of the sinusoid above is called *spatial frequency*:

$$\omega_s = \omega_c \frac{d\sin(\theta)}{c}. \tag{2.9}$$

This spatial frequency depends on the sound's DOA, and is the key to its isolation.

Beamforming

Now that we have introduced microphone arrays, we can delve into the methods for estimating the DOA of a sound signal and/or enhancing sound signals from a specific

DOA. We will only introduce the concept of non-parametric methods, a.k.a. spatial filtering. There exist other, more advanced parametric methods like MUSIC and ESPRIT beamformers, but this is only a basic introduction.

The general idea of spatial filtering is to combine the microphone signals linearly, applying weights to each. The beamformer signal is

$$B(t) = \sum_{m=1}^{M} W_m Y_m(t) = \mathbf{w}^H \mathbf{y}(t), \tag{2.10}$$

and from this, substituting the array signal from Equation (2.4), we can define the "spatial response" of the beamformer (the array-plus-weights system):

$$B(t) = \underbrace{[\mathbf{w}^H \mathbf{a}(\theta)]}_{\text{spatial response}} S(t) + \mathbf{w}^H \mathbf{e}(t). \tag{2.11}$$

The goal is to design a spatial filter $\mathbf{w}(\theta_0)$ that enhances the signal from DOA θ_0 and dampens signals from other directions.

The most basic approach, delay-and-sum beamforming, aims to pass the signals with DOA θ_0 undistorted and attenuate all the other DOAs as much as possible. That means solving the problem

$$\mathbf{w}(\theta_0) = \underset{\mathbf{w}}{\operatorname{argmin}}\ \mathbf{w}^H \mathbf{w} \qquad \text{subject to } \mathbf{w}^H \mathbf{a}(\theta_0) = 1, \tag{2.12}$$

to which the solution is given by

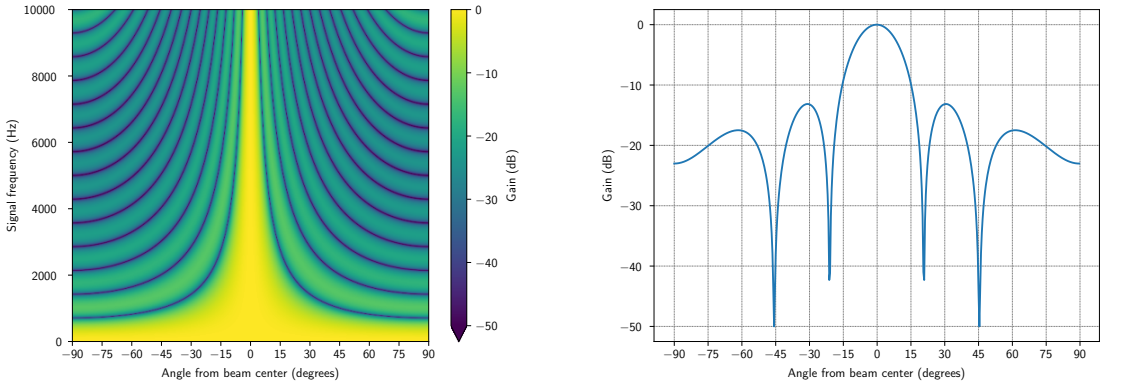
$$\mathbf{w}(\theta_0) = \frac{\mathbf{a}(\theta_0)}{\mathbf{a}^H(\theta_0)\mathbf{a}(\theta_0)} = \frac{\mathbf{a}(\theta_0)}{M}. \tag{2.13}$$

Because of the nature of $\mathbf{a}(\theta_0)$, this solution simply shifts the signals' phases (delaying them) before combining them to introduce constructive or destructive interferences — thus the name, delay-and-sum beamformer. The delays to be applied depend on the array geometry, the signals' frequency(ies), and of course the desired DOA.

Figure 2.3 shows the response of a delay-and-sum beamformer to incoming signals of different frequencies and DOAs. The microphone array considered to draw this response is a 16-microphone linear array with 3 cm spacing between microphones, and the beamformer is steered towards $\theta_0 = 0$.

2.1.2 Deep Neural Networks

In most cases, programming means carefully designing a set of rules that will produce certain outputs depending on the given inputs. If a large enough quantity of examples of desired outputs and inputs is available, it may be possible to instead design a system



(a) Frequency and spatial response

(b) Spatial response for a frequency of 2 kHz

Figure 2.3. Frequency and spatial response of a delay-and-sum beamformer. Created using code from [9].

that will go over that data and *learn* the set of rules by itself. Such systems are the focus of Machine Learning (ML). In recent years, thanks to the increasing amounts of available reference data, ML has proven capable of solving a great variety of problems, often outperforming hand-designed programs and deterministic algorithms. Some examples of these advancements will be presented in Section 2.2. There are a very large variety of ML methods, but we will only present Deep Neural Networks (DNNs), in particular Convolutional Neural Networks (CNNs) and some generic architecture structures involving them.

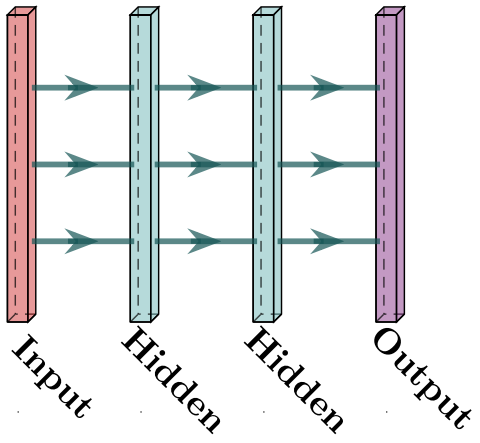


Figure 2.4. The layer structure of a DNN.

DNNs learn to approximate the behaviour of some function f^* by a combination of simpler functions, each of which may involve its own parameters. Each of these chained functions is referred to as a *layer*. The last layer in the network is called the

output layer, since its value constitutes the output of the network; previous layers are called *hidden layers*, since their outputs are not shown outside of the network; the overall length on the chain is named the *depth* of the network. An example of this structure is seen in Figure 2.4. In general, layers located deeper in a network are said to work with *high-level features*. Their operations are more "abstract" in nature, being concerned with the system's behaviour as a whole. The input layer, and in general the more "shallow" layers, work with *low-level features*. They focus on the basic details of small functions rather than complex processes. Taking as example a picture of a building, the lowest-level features would be edges and colors; higher-level features would be the placements of windows and shape of the roof; the highest-level features in a complex network might represent the age or architectural style of the building. Here we will look at some examples of functions that may be employed as layers.

The single elements of most layers are units called *artificial neurons*. Their name — which in turn gives *neural networks* theirs — is owed to some inspirations that these functions take from neurobiology.

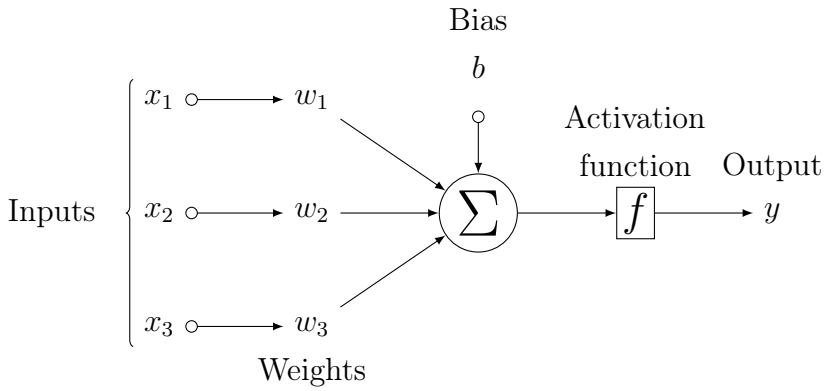


Figure 2.5. An artificial neuron.

An artificial neuron's output value is computed by applying a weight to each input value, summing the results to an optional bias value, and finally applying a (usually non-linear) *activation function* to the resulting value. Figure 2.5 illustrates the working of one such neuron, which can be defined as

$$y = f(b + \sum_n x_n w_n), \tag{2.14}$$

where x_n are the inputs, w_n are the weights, b is the bias, f is the activation function, and y is the neuron activation value. The activation function can be anything from a sign function to a hyperbolic tangent: a common example is a ReLU (Rectified Linear Unit), that lets positive elements pass undisturbed but clips negative elements to 0.

The most basic layer in any DNN is the *fully connected layer*. Each element of a fully connected layer's output is a neuron connected to every output element of the previous layer. However, there are more thoughtful and targeted ways to construct a layer. We will explore some possibilities later.

Training a network

The basic concept behind the training of neural networks is relatively simple. At first, all the parameters are set randomly. An example input is sent through the network, and the resulting output is compared to the desired reference. The comparison between the optimal output and the one given by the network is called *loss function*, *cost function*, or *error function*, and there are many ways of computing it, depending on the problem at hand. For example, if the desired output of the network is real-valued, one can simply use the difference between the desired result and the predicted one; if the desired output is supposed to fall into one of two categories, it is preferable to compute the binary cross-entropy of the desired result and the predicted one; so on. In any case, the next step is to compute the gradient of the error function w.r.t. the network's parameters. The gradient is computed through an algorithm called *back-propagation*: we will not detail how it works here, what is important to know is that the gradient is computed starting from the error value and going backwards through the network, retracing every operation that affected the output, and making note of every parameter's influence on the error. All parameters are then changed slightly, based on the respective gradients, in order to minimize the error. If we call the error function E, the updated parameters will be:

$$\mathbf{p}_1 = \mathbf{p}_0 - \epsilon \frac{\partial E}{\partial \mathbf{p}}. \tag{2.15}$$

where $\mathbf{p}_0$ are the previous parameters, $\frac{\partial E}{\partial \mathbf{p}}$ denotes the partial derivative of the error function w.r.t. the parameters, and ϵ is a coefficient that may be constant, or may be adapted through the training process. This updating process is repeated iteratively on all data samples, until the network's error is satisfyingly small.

Convolutional Networks

Whereas the fully connected layers described above are roughly equivalent to matrix multiplication of inputs and parameters, *convolutional layers* are related to the convolution operation. The convolution of a two-dimensional input X, using a two-dimensional kernel K, is defined as

$$Y(i,j) = (X * K)(i,j) = \sum_m \sum_n X(i-m, j-n) K(m,n). \tag{2.16}$$

An example of this operation is seen in Figure 2.6.

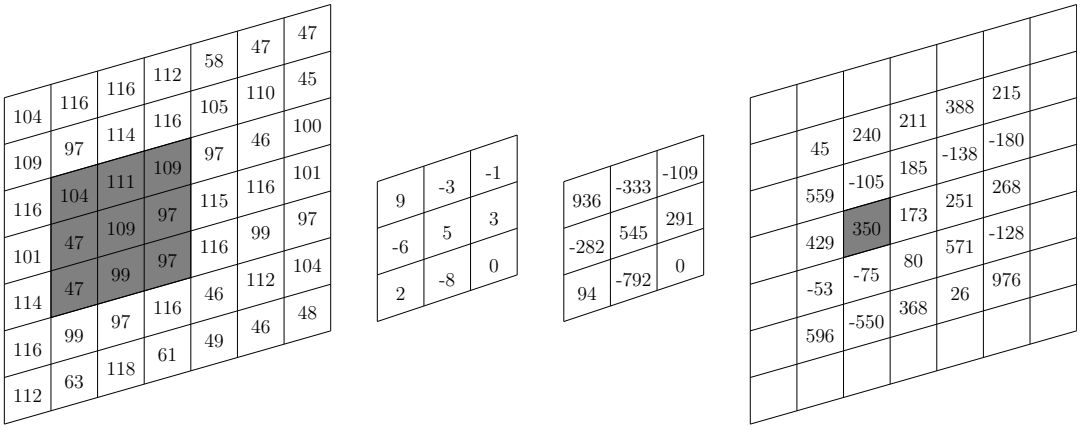


Figure 2.6. An example of the convolution operation[13].

A convolution layer has two very notable differences w.r.t. a fully connected layer. One is that each output element of a convolution layer only depends on a neighbourhood — as large as the kernel — of the relative input element, instead of depending on all input elements. The second is that all such "neighbourhoods" share the same processing parameters, i.e. the kernel parameters. Using fewer parameters reduces the memory requirements of the model, improves its statistical efficiency, and requires fewer operations during both forward and backward passes. Another effect of using the same parameters on different parts of the input is an equivariance to translation. If the convolutional layer "responds" in a certain way to a specific localized feature in the input, translating that feature in the input will generate the same response, translated, in the output.

Convolution layers have a certain number of *channels*: this simply refers to different kernels. Each kernel of a layer processes the input in parallel, and their results are concatenated to form the layer output, which can thus be seen as having a number of "channels". If a DNN employs at least one convolutional layer, it is called a Convolutional Neural Network (CNN).

Like fully connected layers, convolution layers may be followed by a nonlinear function. After that, however, another function is sometimes applied, called *pooling*. A pooling function computes, for each element of the input, a summary of neighbouring elements. One example is max pooling, that substitutes each element's value for the largest value in its vicinity. Figure 2.7 shows an example of this. This can help stabilize the network against small translations of the input: if a feature of the input is translated by only a few elements, the relevant pooling outputs do not change.

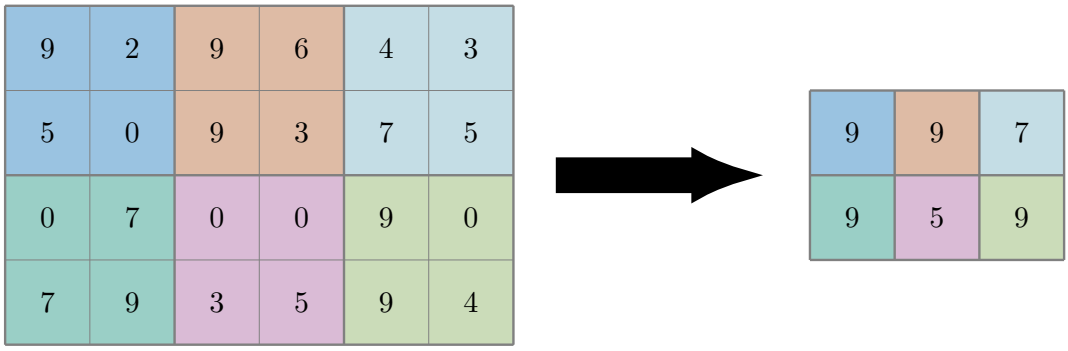


Figure 2.7. An example of 2×2 pooling[14].

To summarize, a convolution "block" is usually composed of a sequence of

- convolution layer,
- non-linear function,
- pooling layer.

Figure 2.8 shows a chain of such blocks. Notice that the width and height of the layers decreases after each pooling layer, while the number of channels in each convolution layer increases. The decreasing width and height are a result of the pooling layers, and

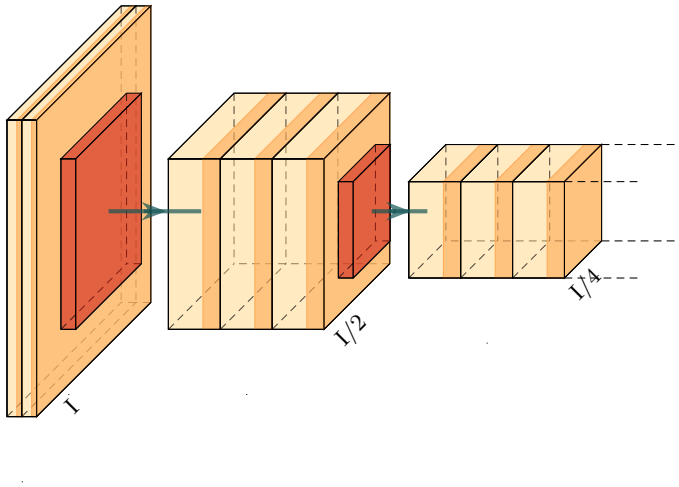


Figure 2.8. A basic CNN: blocks are built from series of convolution+nonlinearity followed by pooling.

indicate to the fact that the extracted features are decreasingly position-dependent as they travel through the network. The increasing number of channels signals an

increase in the number of parameters employed by the hidden layers, which makes them more computationally powerful.

Autoencoders

The term *autoencoder* refers to a particular network architecture, defined by the presence of one layer with a specific purpose. The most common way of employing an autoencoder is creating a "bottleneck" layer, which contains far fewer elements than the input, as shown in Figure 2.9. Autoencoders are trained to replicate the inputs in

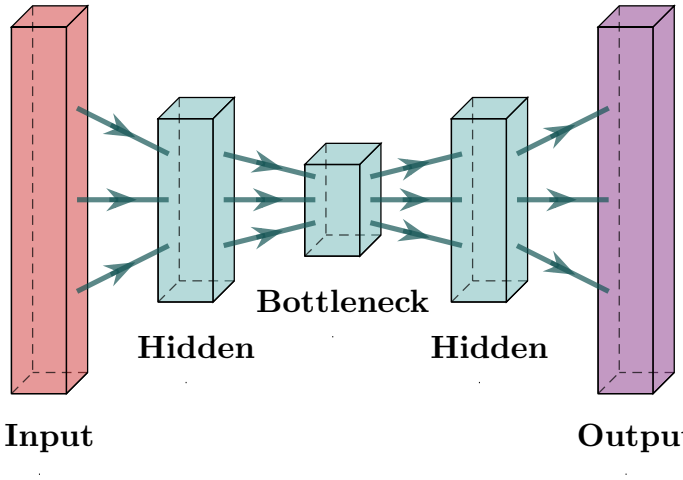


Figure 2.9. The general architecture of a bottleneck autoencoder.

the outputs, but they include a "restricted" hidden layer to prevent a simple copy of the input as-is: depending on the restrictions, the network should only learn to replicate inputs similar to the training data. Because it must learn which features should be copied, it often learns useful properties of the data. The restricted layer must learn to create a representation of the input, which is called a *code*. For this reason, the first "half" of an autoencoder network (the part producing the code) is termed *encoder*, and the second "half" (that turns the code back into the input) is termed *decoder*. This forces the network to condense the most meaningful high-level features of the input into the smaller feature space, and to reconstruct the whole input starting from those features. In doing so, it must learn to both recognize and re-create those features.

That learnt knowledge can then be applied to a variety of problems: one example is denoising. A *denoising* autoencoder is not simply trained to reproduce the exact input. Instead, the clean input is corrupted before being fed to the network, which

means the encoder must not only recognize the input's salient features, but tell them apart from the noise, in order to produce a clean reconstruction.

Another example application of autoencoders is *representation learning*. For this application, the autoencoder is trained with clean data, but only the encoder is of interest: since it is effectively a smaller-space representation of the input's features, it can itself serve a variety of purposes, such as clustering for classification. Using an autoencoder for this purpose can sometimes be a good alternative to hand-crafting a complex feature extraction algorithm.

U-Nets

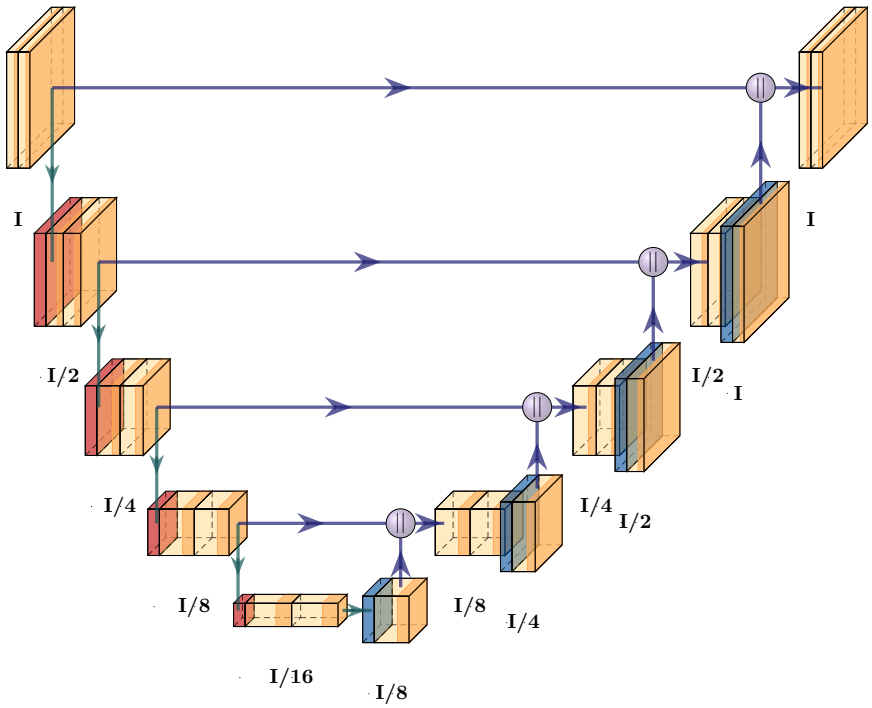


Figure 2.10. A Unet DNN.

Lastly, we'll introduce a very specific sub-type of autoencoders called *U-Nets*. U-Nets are mostly composed of convolutional layers, that shrink and expand in size to create a typical autoencoder bottleneck in the middle of the network. However, each layer in the decoder takes as input not only the output of the immediate predecessor, but also the output of its counterpart from the encoder.
A clear example is shown in Figure 2.10.

What this achieves is a merge of high- and low-level features. The encoder, like in common autoencoders, is able to learn and put to use high-level features of the data. The *skipped* connections between encoder and decoder allow the transmission of low-

level, high-definition features like edges and smaller details. Of course, this method is hardly useful for the exact-copy training of some autoencoder applications, but proves powerful in contexts like that of denoising autoencoders. Such applications require both a high-level understanding of the target data's features, and high-definition details of the input data to be preserved in the output.

2.2 State-of-the-art

We will now illustrate some state-of-the-art works in the fields of speaker separation and diarization. Many of them integrate the tasks in some fashion, so we will pay particular attention to how and why they do it. Finally, we will delineate our goal of integrating the tasks at hand in a way that differs from — and improves on — all existing works.

2.2.1 Speaker separation

In recent years, techniques for isolating or enhancing a speaker's voice in a noisy recording have been thoroughly dominated by DNNs. Large quantities of training data are relatively easy to acquire (or simulate), and the flexible performance of Deep Learning (DL) has long surpassed any other solution. That being said, the variety of proposed systems is large. While they are all characterized by some sort of DNN, there is great diversity in the architectures of the proposed networks, their purpose in the systems, and their training processes. We will present here some of the most prominent works that inspired this thesis. They are for the most part works that employ microphone arrays in addition to DNNs, since that allows exploiting spatial information and yields better results.

The system proposed by Yoshioka et al.[27] uses two separate DNNs in conjunction with a beamformer. The first network — which is a hybrid of a CNN and a Recurrent Neural Network (RNN) — takes as input the magnitude spectrum from a reference microphone, as well as the phase differences between that same microphone and all the others in the array, giving as output up to two masks. Each mask can be used to isolate a different speaker's voice from the recording, so the network can separate up to two simultaneously active speakers. Each separation mask is applied to all array signals, which are then used to estimate the voices' DOAs; a beamformer is pointed towards each found direction, and the beamformed signals are finally processed by a second network — a more standard RNN — for additional enhancement. It should be noted that the entire first DNN is only used to estimate the DOA and choose the

most appropriate beamformer, and has no other influence on the output signal. The whole system can operate in real time, with very low latency.

A recurring problem that surfaces in many approaches is the *permutation problem*. Let's take for example the first network designed by Yoshioka et al.[27], as just described. The network outputs two masks, which relate to two speakers in a mixture. The output order of the masks is irrelevant during regular use, but requires special handling during the network's training process. This is usually solved using a technique called PIT[28], which performs back-propagation based on the permutation of the outputs which offers the smallest error.

Wang and Wang[26] employed a similar approach to Yoshioka et al.[27], by feeding the amplitude and phase data from the array directly to a DNN that produces masks for initial separations. The masks, trained with PIT, are then used to estimate the DOAs and point beamformers. The beamformer signals are enhanced with a second DNN. However, the network approach is different from Yoshioka et al.[27]'s: the array microphones are considered two by two, and each pair's data is processed by the DNN. Moreover, the second network receives data from the first network's separation in addition to the beamformer signal. The number of speakers is assumed known in advance, and the second processing step is applied for each source. This system only operates offline.

Chen et al.[3] chose instead to apply the beamformers first, and process beamformed signals with a single DNN. A fixed set of beamformers that spans 360° is active at all times, producing B beamformed signals. Each beamformer signal is then processed by the network, which produces E different separation masks for each beamformer signal. N signals are then picked out of the $B * E$ resulting signals, with N being the number of speakers known in advance, by a process of magnitude spectrum correlation and spectral clustering. The assumption is that each speaker appears as one of the first E strongest sources in at least one of the B calculated beams. Of course, the permutation problem is present again, because the networks output multiple separation masks, targeting all the most energetic sources in the provided beamformer signal.

All of these systems achieve considerable isolation quality, but no identification, which would need to be carried out separately. In Section 2.2.3 we will see several advantages of combining identification and isolation into one system.

2.2.2 Speaker diarization

Much like speaker separation, the field of speaker diarization has also heavily shifted towards the use of DL. This can be very clearly observed in the review of diarization techniques provided by Park et al.[17]. The most widely used approach is that of

using a DNN to find speaker embeddings, often called *d-vectors*, and use them for clustering to identify the "owner" of each specific segment of audio. Wang et al.[25] find these embeddings with the aid of a RNN, for example, and perform spectral clustering with the addition of some refinement operations on the affinity matrix.

Fujita et al.[6] choose instead to incorporate both processes into one end-to-end network, which directly outputs diarization results, to address the issue of speech overlaps. They also introduce "self-attention layers" in their network, please refer to their paper for details on their function.

2.2.3 Integrating tasks

The focus of our work was to integrate speaker diarization and separation in one system in order to exploit the shared aspects of the two tasks. Finding the salient features that characterize a speaker's identity can aid in separating their voice from a mixture of speech and noise, and vice versa, isolating the target voice from unrelated sounds is key to correctly identifying the speaker. In the following sections, we will explore works that aim at similar task integrations, and conclude by presenting our takeaways from the examined literature.

Integrating diarization and transcription

Sarkar et al.[20] proposed a system that joins the tasks of diarization and speech recognition (what words were spoken) rather than speech isolation. Isolation can be seen as a step to enhance a subsequent recognition, but the clean signal may also have many other purposes, so this system comes short of our task by only providing a transcription of the speech. Nevertheless, it provides insight on the concept of joining tasks, showing that a neural network can sometimes profit by being jointly trained on two tasks rather than being split into two nets with different purposes.

Integrating identification and dereverberation

Zhang et al.[30] showed two interesting aspects regarding the dereverberation of speech. First, by training a bottleneck-based denoising autoencoder to generate speaker-identifying features, they note that the model can transform reverberant speech features to a less-reverberant feature space, which offers better speaker identification abilities. Then, they train another denoising autoencoder to turn reverberant features into non-reverberant ones, with the expectation of improving speaker identification by using these "cleaned" features. Finally, they propose a joint use of both these separate networks in parallel, to achieve the best possible identification performance.

Separation using speaker features

The system proposed by Wang et al.[24] is separated in two modules. The first one extracts speaker embeddings from the incoming signal, compares them with an inventory of embeddings extracted from a collection of pre-enrolled speaker signals, and selects the embeddings related to the current speaker(s). The relevant embeddings are passed to the second module, that generates masks to perform speech separation. This separation module is also trained with PIT, since it outputs two masks in irrelevant order. It is shown that providing these target-speaker-specific features aids in the separation of the target speech.

The SpeakerBeam system by Žmolíková et al.[31] extracts a speaker's voice from a mixture by providing a DNN with a different, clean utterance from the target speaker to be isolated. The permutation problem is avoided because the network only provides one output, being instructed on which speaker to locate and isolate. The extraction results are good, showing that the network learns to separate the different parts of the mixture and confront them with the reference clean utterance, so as to choose the correct parts to isolate.

Key concepts

Here we pointed out works showing that

- a neural network can profit by being jointly trained on two tasks rather than being split into two nets with different purposes;
- by training a bottleneck layer to generate speaker-identifying features, a model can be made to transform noisy speech features to a cleaner feature space;
- working the other way around, training a model to turn noisy features into clean ones can improve speaker identification;
- providing target-speaker-specific features aids in the separation of the target speech;
- a network can learn to separate the different parts of a mixture, confront them with a clean utterance, and choose the correct parts to isolate.

We now set out to design a single bottleneck-based network that jointly provides speech separation and speaker identification. Our goal is to fully share the speech feature extraction process between the two tasks, considering both high- and low-level features related to both the target and disturbances. We hope to solve the permutation

problem by training the network to provide a single output, relative to the one speaker that a beamformer is directly aimed towards.

Chapter 3

Architecture

As shown in Section 2.2.3, joining the tasks of speech separation and diarization can yield better results than performing them separately, so we set out to design a joint system that handles both aspects in an integrated fashion. However, maintaining a certain degree of modularity in the system has several advantages.

For example, separating the spatial processing (i.e. beamforming) in a standalone module allows for a flexible separation system that does not depend on the array geometry nor the beamforming algorithm used; a system that can scale with the available hardware. Moreover, DL-based beamformers have not yet surpassed the quality of state-of-the-art beamforming algorithms, and relying on a well-established deterministic algorithm eases the training process for the separation network.

This chapter will define the structure of the system we designed. Starting by defining the problem at hand in Section 3.1, we will then describe the overall layout of the modular system in Section 3.2, going over each module's purpose and inner working.

The network's architecture, and the reasoning behind it, is detailed in Section 3.2.4. As previously stated, the network was designed to join the separation and identification tasks, and is the core module of the system. Rather than extracting speaker embeddings and feeding them to a separate isolation module like some works at the state-of-the-art do, we opted to employ a single network with two outputs: a clustering-ready embedding of the target speaker's features, and an isolation mask to be applied to the mixture spectrum. Both outputs are related to the speaker spotlighted by the beamformer signal that the network takes as input.

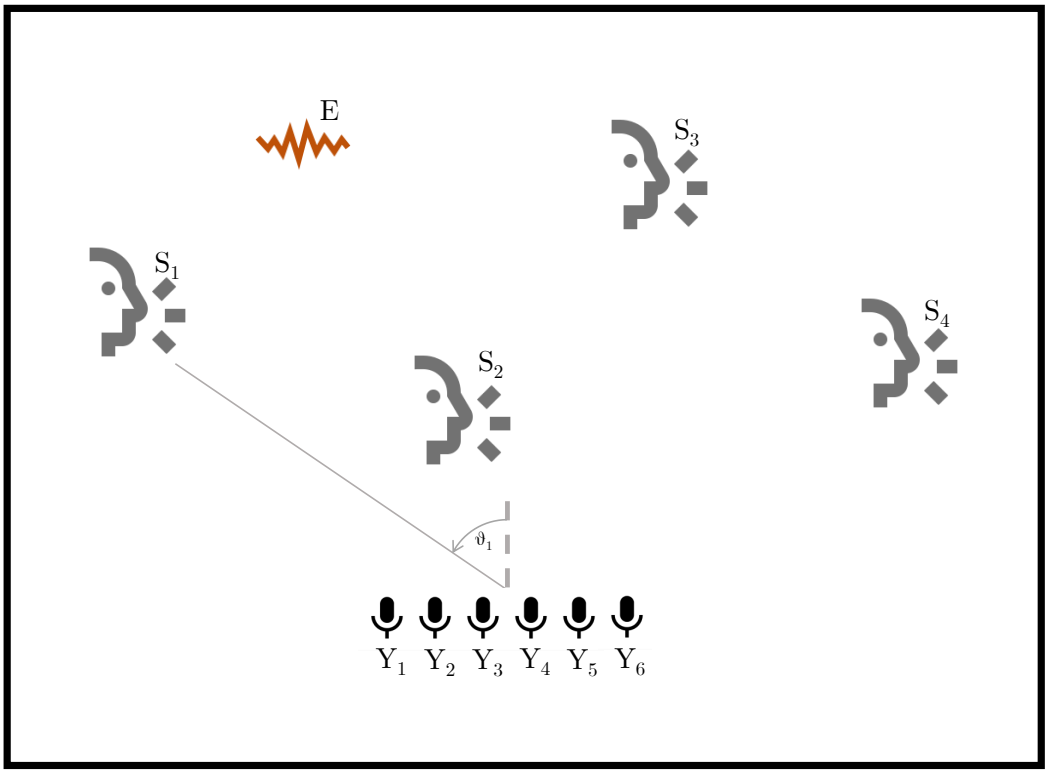


Figure 3.1. An example configuration of the problem. In this case there are 4 sources, 6 microphones, and a source of noise. The first speaker's DOA θ_1 is displayed.

3.1 Problem formulation

The goal of this thesis is to create a system that can process the multi-channel microphone array recording of a conversation and produce a multi-channel recording where each speaker's isolated voice is assigned to a different channel based on identity. Let's consider a situation with N speakers scattered around a M-microphone linear array. We will characterize each speaker n's position w.r.t. the array by the angle θ_n to the line perpendicular to the array, as in Figure 3.1. The speakers usually take turns, but multiple speakers may also be active simultaneously for arbitrarily long periods of time. They may change locations over the course of the conversation, but their speech should be assigned to the identity-unique channel regardless.

Using the notation introduced in Equation (2.1), the n^{th} speaker's contribution to the m^{th} microphone recording, $y_{m,n}(t)$, can be calculated as

$$y_{m,n}(t) = h_{m,n}(t) * s_n(t - \tau_{m,n}), \tag{3.1}$$

where:

- $s_n(t)$ is the n^{th} speaker's speech signal,
- $h_{m,n}(t)$ is the impulse response from the n^{th} speaker's position to the m^{th} microphone's position,
- $\tau_{m,n}$ is the propagation delay between the n^{th} speaker's position and the m^{th} microphone's position.

Applying the principle of superposition, the signal $y_m(t)$ recorded at each microphone can be seen as a sum of the individual speakers' contributions:

$$y_m(t) = \sum_{n=1}^{N} y_{m,n}(t) = \sum_{n=1}^{N} h_{m,n}(t) * s_n(t - \tau_{m,n}) + e_m(t), \tag{3.2}$$

where $e_m(t)$ is an additive noise at the m^{th} microphone. We will refer to the array recording using vector notation, with

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_M(t) \end{bmatrix}. \tag{3.3}$$

Our system's ideal output will then be

$$\mathbf{s}(t) = \begin{bmatrix} s_1(t) \\ s_2(t) \\ \vdots \\ s_N(t) \end{bmatrix}, \tag{3.4}$$

where $s_n(t)$ is speaker n's sound signal.

3.2 System layout

Figure 3.2 shows the overall architecture of the designed system. The following sections will detail each module's function, which can be summarized as:

- Windowing and Overlap-add are employed to elaborate recordings in chunks;
- Speech detection and DOA evaluation are employed to determine when speakers are active, how many, and where from;
- a separate beamformer is applied towards each active speaker, and each beamformer produces a rough preliminary separation of the targeted speaker signal;

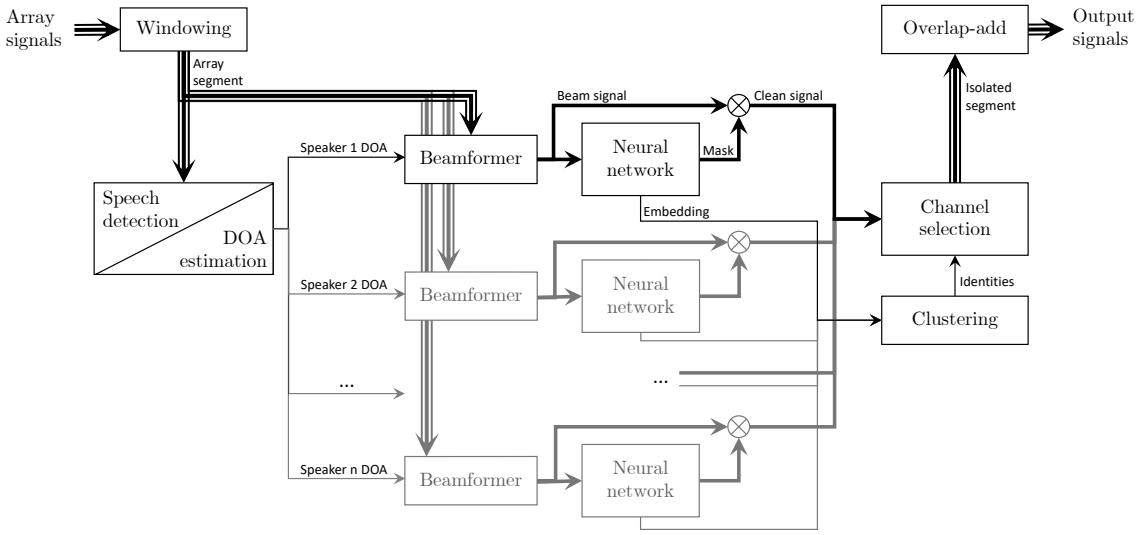


Figure 3.2. General layout of the system.

- a neural network uses the beamformers' outputs to generate speaker embeddings for identification and spectral masks for isolation;
- clustering is performed on the resulting speaker embeddings to assign an identity to each isolated segment, and assign it to the appropriate output channel.

3.2.1 Windowing / Overlap-add

The incoming microphone array recording is first split into overlapping segments known as *windows*. We will denote this using the superscript ^{W}, for example $\mathbf{y}^W(t)$ will refer to a single window of the the entire recording $\mathbf{y}(t)$. Each segment is processed by the system, and the resulting output segment is likewise added to the outgoing audio stream. This technique is known as *overlap-add*, since the overlapping segments are added one on top of the other to recreate the segmented signal. Our implementation runs offline, but the entire system could be rendered capable of real-time processing with some modifications to the clustering module (see Section 3.2.5).

3.2.2 Speech detection and DOA evaluation

Each array segment $\mathbf{y}^W(t)$ is analyzed with a combination of speech activity detection and DOA estimation. Our experiments relied on oracle prediction for both, with the confidence that state-of-the-art deterministic algorithms — such as MUSIC for DOA estimation and a simple DNN for speech activity detection — can handle both tasks reliably in reasonable conditions.

Speech activity detection, as the name implies, determines whether the given segment contains speech, as well as the number N_a of concurrently active speakers. Note that N_a does not correspond to the total number N of speakers involved in the conversation. The segment is ignored if no speech is detected, otherwise the DOA estimation determines the location of each active speaker.

The result is a list of directions

$$\theta_1, \theta_2, \ldots, \theta_{N_a} \tag{3.5}$$

of all speakers active during the given segment. Each of these directions is then processed separately by the next module, the beamformer.

3.2.3 Beamforming

The beamforming module takes the target direction θ_n provided by the DOA module and picks the best choice out of a set of pre-computed delay-and-sum beamformers (see Equation (2.13)). The choice is made by finding the beamformer pointed in the direction closest to the required θ_n. The chosen beamformer filters $\mathbf{h}_n^B(t)$ are then applied to the microphone array segment $\mathbf{y}^W(t)$, producing the signal $b_n^W(t)$:

$$b_n^W(t) = \frac{1}{M} \sum_{m=1}^{M} h_{m,n}^B(t) * y_m^W(t). \tag{3.6}$$

The beamforming filters $\mathbf{h}^B(t)$ are the time-domain counterpart to the beamforming weights $\mathbf{w}$ introduced in Equation (2.10), and are thus applied through convolution instead of multiplication.

3.2.4 Neural network

The beamformer signal $b_n^W(t)$ is transformed with a Short Time Fourier Transform (STFT) obtaining the spectrum $B_n^W(f,t)$. The magnitude spectrum, $|B_n^W(f,t)|$, is used as input for the neural network, which returns a speaker embedding $\boldsymbol{\varrho}_n$ and a separation mask $M_n^W(f,t)$. The beamformer signal spectrum is then multiplied by the separation mask, obtaining the spectrum of the isolated target speech

$$S_n(f,t) = B_n^W(f,t) M_n(f,t). \tag{3.7}$$

This spectrum is anti-transformed to $s_n^W(t)$ and sent to the channel selection module, together with the relative embedding $\boldsymbol{\varrho}_n$, to be assigned to the correct output channel.

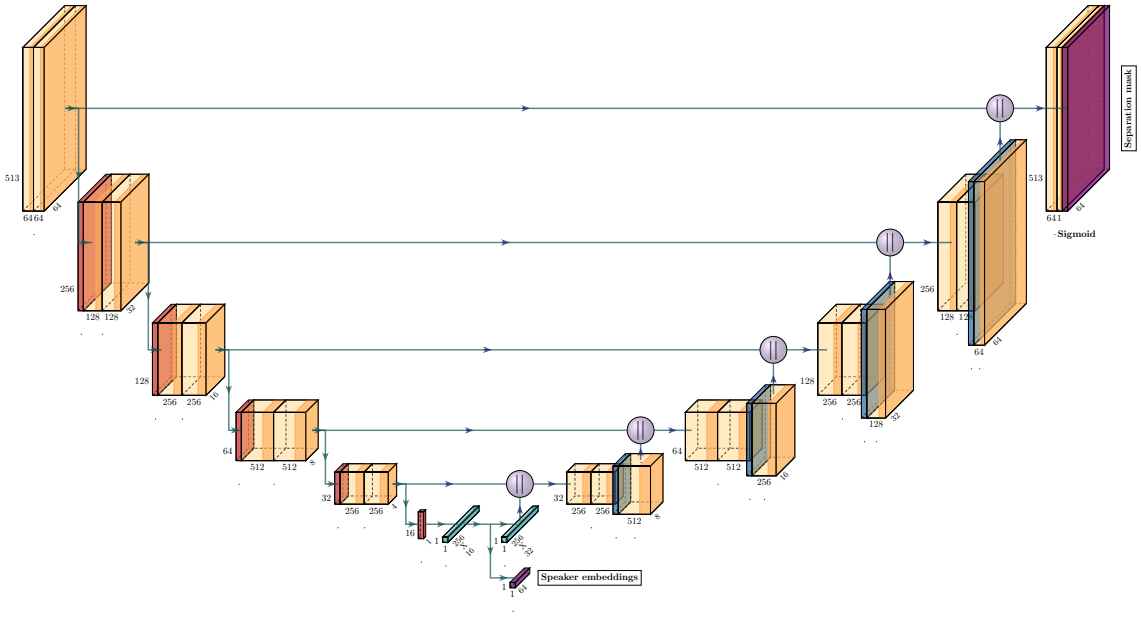


Figure 3.3. The proposed network. A larger version of the image can be found in the appendix, as Figure 1.

Network architecture

Figure 3.3 shows an overview of the proposed network.
The basic architecture of the network is a U-Net: as introduced in Section 2.1.2, this means that the network is divided in two parts. The first half (the *encoder*) takes the magnitude spectrum $|B_n^W(f,t)|$ and applies a series of convolution layers of decreasing width and increasing depth (number of channels), ending with a pair of fully-connected layers that provides the speaker embedding $\boldsymbol{\varrho}_n$. The second half (the *decoder*) of the network takes intermediate results — commonly referred to as *skipped connections* — from various layers in the encoder, and applies a series of transpose-convolution layers of increasing width and decreasing depth, providing the separation mask $M_n^W(f,t)$. This can be seen as a reversal of the encoder's process.

Both parts are mostly composed of *double convolution* blocks defined as:

1. 3×3 convolution with 1×1 padding,
2. batch normalization,
3. ReLU;
4. 3×3 convolution with 1×1 padding,
5. batch normalization,

6. ReLU.

The encoder alternates this block with pooling layers, repeating both five times:

1. double convolution, 2×2 pooling;
2. double convolution, 2×2 pooling;
3. double convolution, 2×2 pooling;
4. double convolution, 2×2 pooling;
5. double convolution, 2×∞ pooling[1];
6. flattening to one dimension, fully-connected layer, batch normalization, PReLU[2];
7. fully-connected layer, element-wise normalization.

The final fully-connected layer, being the output layer for the speaker embedding $\boldsymbol{\varrho}$, features no batch normalization nor non-linearity, and its outputs are instead normalized individually.
The decoder alternates convolution blocks and transpose-convolution layers with 2×2 kernel and 2×2 stride, which double the dimensionality along both axis, inverting the pooling operation. In the decoder, each block takes as input a concatenation of the previous block's output and the corresponding skipped connection of the same size (refer to Figure 3.3). The decoder's blocks are:

1. fully-connected layer, batch normalization, PReLU, un-flattening to 3 dimensions[3];
2. double convolution, transpose-convolution;
3. double convolution, transpose-convolution;
4. double convolution, transpose-convolution;
5. double convolution, transpose-convolution;
6. double convolution, sigmoid activation function.

In the last convolution block, the second convolution layer only has one channel, and the closing non-linearity is a sigmoid instead of the usual ReLU. This is used as the output layer for the separation mask $M_n^W(f,t)$.

[1]The last pooling layer, denotes as 2×∞, reduces the dimensionality down to only one sample along the time axis, and by the normal factor of 2 along the frequency axis.

[2]Parametric ReLU: $\mathrm{PReLU}(x) = \max(0,x) + a * \min(0,x)$, with a being a learnable parameter.

[3]The linear output is first folded into two dimensions (channels and frequency), then repeated along the time dimension, replicating the shape that enters the last pooling layer.

Goal of the architecture

The goal of this single-network architecture is to fully share the speech feature extraction process between the two branches, with a focus on the inclusion of both high- and low-level features. By extracting speaker embeddings and feeding the correct one to a separate isolation module, the shared features would be exclusively related to the target speaker, and only high-level. Only sharing features related to the target speakers means that the isolation module would need to learn to differentiate between the target speech's features and those of disturbances. On top of that, only sharing high-level features means that the isolation module would also need to learn to translate them into low-level features in order to include their information in the output. All of this is redundant, because the identification module must already have learned to separate noise and different speech features to "locate" the target speaker in the mix, extracting both high- and low-level features throughout the process. We believe that sharing features of all levels would avoid redundant re-learning; moreover, allowing back-propagation from both cost functions should strengthen the shared aspects of the problem, improving training and ensuring a better merge of the tasks.

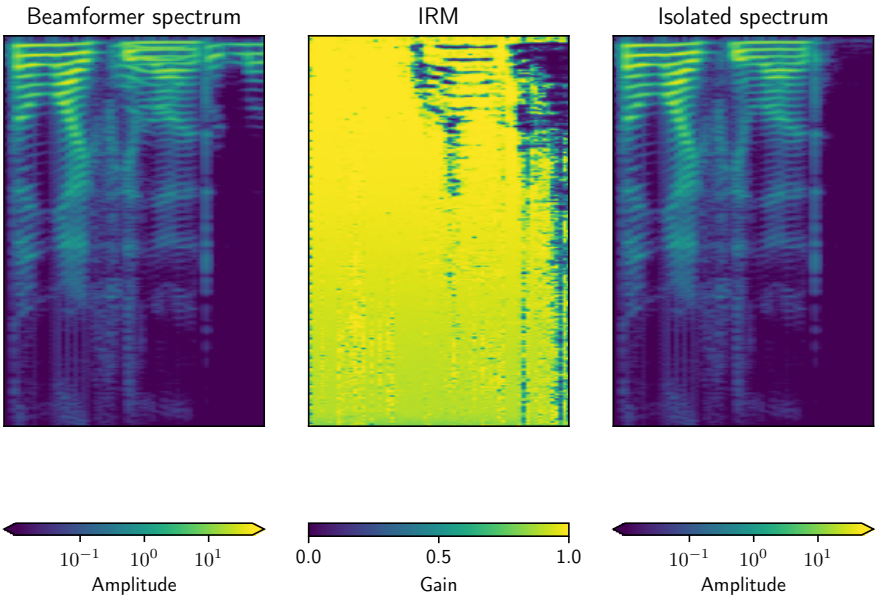
Training losses

The loss function used to train the network is a weighted sum of a loss function related to the masks and one related to the embeddings. We will detail both here, starting with the masks.

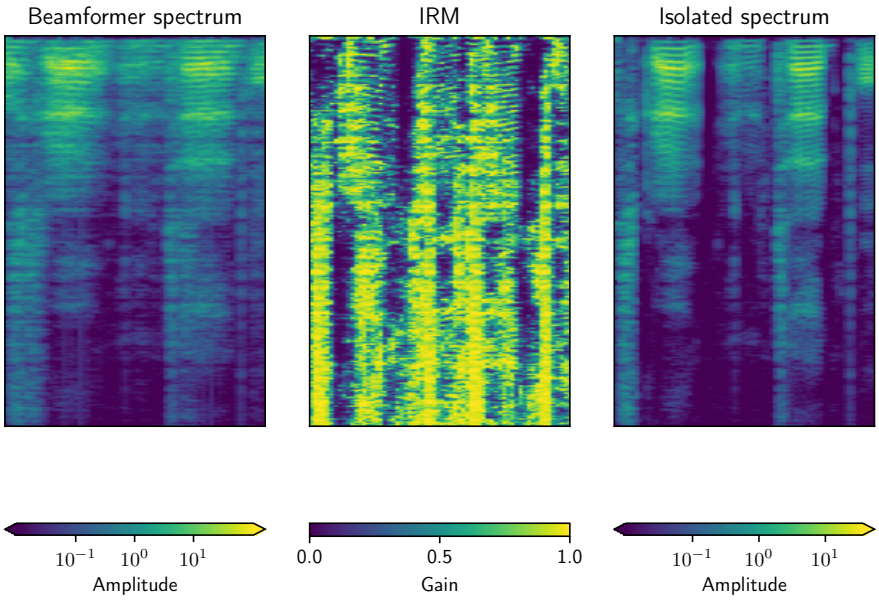
The training objective for the separation masks is the Ideal Ratio Mask (IRM), which is defined as

$$\mathrm{IRM}_n(f,t) = \sqrt{\frac{|S_n^W(f,t)|^2}{|S_n^W(f,t)|^2 + |N^W(f,t)|^2}} \tag{3.8}$$

where $|S_n^W(f,t)|^2$ is the desired clean signal's energy and $|N^W(f,t)|^2$ is the collective disturbance's energy (other speakers, ambient noise, white noise, ...). Figure 3.4 shows two examples of IRMs with the respective beamformer spectra and clean speech spectra. The signal spectra are presented here in log scale for readability, but the network operates in linear scale. IRM time-frequency bins range in values from 0 to 1, with smaller values removing spectral components from the masked signal, and high values letting them through. In Figure 3.4a, which only displays a speaker overlap with no reverb, it is apparent that the leftmost (earlier in time) part of the spectrum is identical between the beamformer spectrum and the clean, isolated speech. This is reflected by the IRM retaining only high values in that area. In the rightmost (later in time) part of the spectrum, the beamformer presents additional spectral features



(a) Beamformer spectrum, IRM, and clean speech spectrum in a situation with a partial overlap and no reverb.



(b) Beamformer spectrum, IRM, and clean speech spectrum in a situation with reverb and no overlaps.

Figure 3.4. Two examples of IRM. Overlaps and reverb are usually both present, here they were considered separately for clarity.

compared to the clean speech. These features pertain to a secondary speaker, and the IRM removes them by taking on low values in the appropriate time-frequency bins. Thoroughly analysing Figure 3.4b, which displays a situation with reverb but no speaker overlaps, proves trickier. Suffice it to say that once again, low-valued bins in the mask dampen the undesired spectral features. Reverb and speaker overlaps are presented here separately for the sake of illustrating the IRM's behaviours, but of course reverb is always somewhat present, as is background noise.

Returning to the matter of network loss functions, the objective IRM is compared to the network output using the self-adjusting smooth L1 loss introduced by Fu et al.[5]. The common smooth L1 loss is defined as

$$L_{\text{smooth}}(x) = \begin{cases} 0.5\frac{x^2}{\beta}, & \text{if } |x| < \beta \\ |x| - 0.5\beta, & \text{otherwise} \end{cases} \tag{3.9}$$

where x is the difference between the predicted value and the reference ground-truth (in this case, the network output and the ideal IRM respectively), and β is a scalable parameter. Fu et al.[5]'s method adjusts the β parameter during training, based on the running statistics of the absolute loss. Please refer to their paper for further details.

The loss function for embeddings is a standard triplet loss[22] defined as

$$L_{\text{triplet}}(\boldsymbol{\varrho}_A, \boldsymbol{\varrho}_P, \boldsymbol{\varrho}_N) = \max(0, ||\boldsymbol{\varrho}_A - \boldsymbol{\varrho}_P||^2 - ||\boldsymbol{\varrho}_A - \boldsymbol{\varrho}_N||^2 + \alpha), \tag{3.10}$$

where α is a margin value and $\boldsymbol{\varrho}_A, \boldsymbol{\varrho}_P, \boldsymbol{\varrho}_N$ are the embeddings of, respectively, an *anchor* input A, a *positive* input P of the same identity as A, and a *negative* input N of a different identity from A. This means that for every input-output pair, the output is compared to two outputs from different inputs, one being a different utterance from the same speaker and the other being an utterance from a different speaker. For the triplet selection approach, see Section 4.2.2.

To sum up, the network is provided with triplets of beamformer spectra

$$|B_A^W(f,t)|, |B_P^W(f,t)|, |B_N^W(f,t)| \tag{3.11}$$

and the relative ideal isolation masks

$$\text{IRM}_A(f,t), \text{IRM}_P(f,t), \text{IRM}_N(f,t); \tag{3.12}$$

it produces a triplet of embeddings

$$\boldsymbol{\varrho}_A, \boldsymbol{\varrho}_P, \boldsymbol{\varrho}_N \tag{3.13}$$

and one of masks

$$M_A^W(f,t), M_P^W(f,t), M_N^W(f,t). \tag{3.14}$$

The total training loss is given by:

$$L_{\text{total}} = w_e L_{\text{triplet}}(\boldsymbol{\varrho}_A, \boldsymbol{\varrho}_P, \boldsymbol{\varrho}_N) + w_m \sum_{n=A,P,N} \sum_{f,t} L_{\text{smooth}}(M_n^W(f,t) - \text{IRM}_n(f,t)), \tag{3.15}$$

where w_e, w_m are weights applied to each term.

3.2.5 Clustering and channel selection

After the whole recording has been processed, spectral clustering[15] is applied to the speaker embeddings $\boldsymbol{\varrho}$. The process of spectral clustering can be decomposed as follows:

1. calculate the affinity matrix A, by computing the Euclidean distance between each pair of embeddings $\boldsymbol{\varrho}_a, \boldsymbol{\varrho}_b$;
2. calculate the Laplacian L of A;
3. calculate the first k eigenvectors (the eigenvectors corresponding to the k smallest eigenvalues) of L;
4. consider the matrix formed by the calculated eigenvectors (each row defines the features of an embedding);
5. apply clustering based on these features.

For the last step, we used K-Means clustering. K-Means partitions all embeddings into N clusters by locating N cluster *centroids*, where each centroid is the mean of the features of embeddings pertaining to that centroid's cluster. This is done by starting with N random centroids, and iteratively alternating between re-calculating the centroid positions as means of the clusters and re-assigning embeddings to the nearest centroid, until assignments no longer change.

The clustering results assign an identity to each processed segment, and according to this assignment the isolated segments $s_n^W(t)$ are overlap-added to the outgoing audio stream. Thus is finally achieved the objective, the $\mathbf{s}(t)$ signal introduced in Section 3.1.

For our experiments, the total number of speakers present was assumed known in advance, but a different clustering approach may guess the appropriate number of clusters on its own. Just as well, a different clustering approach may allow processing of the embeddings as they come, thus enabling real-time operation of the whole system.

Utterance-aware clustering

The first and last segments of each utterance, where overlaps may be present and/or the target speech may only take up a small fraction of the segment, are troublesome for the feature extraction and thus for the clustering accuracy. To compensate for this, we considered the DOA and speaker change information found by the DOA module, estimating which utterance contained each segment. We then grouped embeddings by utterance, and for each utterance we observed the prevailing cluster over all its embeddings, taking it as the true cluster for the whole utterance. The results in Chapter 5 present both the clustering accuracy of lone embeddings, and that of utterance-grouped embeddings, showing much higher performance for the latter.

Chapter 4

Experimental setup

This chapter will outline the training and testing process used in our experiments, starting with the construction of a dataset of simulated microphone array recordings of conversations in reverberant rooms in Section 4.1. Then, Section 4.2.1 will detail how the dataset was used differently during training and testing, Section 4.2.2 will explain how training batches were assembled, and Section 4.2.3 will list the training parameters and settings. Section 4.3 will describe the metrics used for evaluation.

4.1 Dataset

The simulated conversation dataset was constructed in two steps. First, a RIR dataset was produced by simulating a set of rooms with different shapes and wall absorption coefficients. Then, the simulated RIRs were used in conjunction with clean speech from the TIMIT[7] and LibriSpeech[16] datasets to create simulated conversations, with varying amounts of overlap between speakers and different numbers of total involved identities.

4.1.1 Room simulation

The simulated rooms were exclusively constructed with irregular geometry, in particular parallel walls were avoided to prevent reflection artefacts in the image-source-based acoustic engine, pyroomacoustics[21]. In each room, we considered a 16-microphone linear array, with 3 cm spacing between microphones, and several possible source positions around the array. Sources were placed on only one side of the linear array, in a 180° spread (±90° of the line perpendicular to the array), at 15° intervals. Distances from sources to the array were multiples of 1 m, and more were added in larger rooms that allowed it. RIRs were simulated between every source-microphone pair. The

array positions were individually picked for each room geometry, in order to ensure a variety of arrangements — flush to a wall, in the middle of a room, by a corner, etc. Figure 4.1 shows two of the room geometries, and one RIR from each.

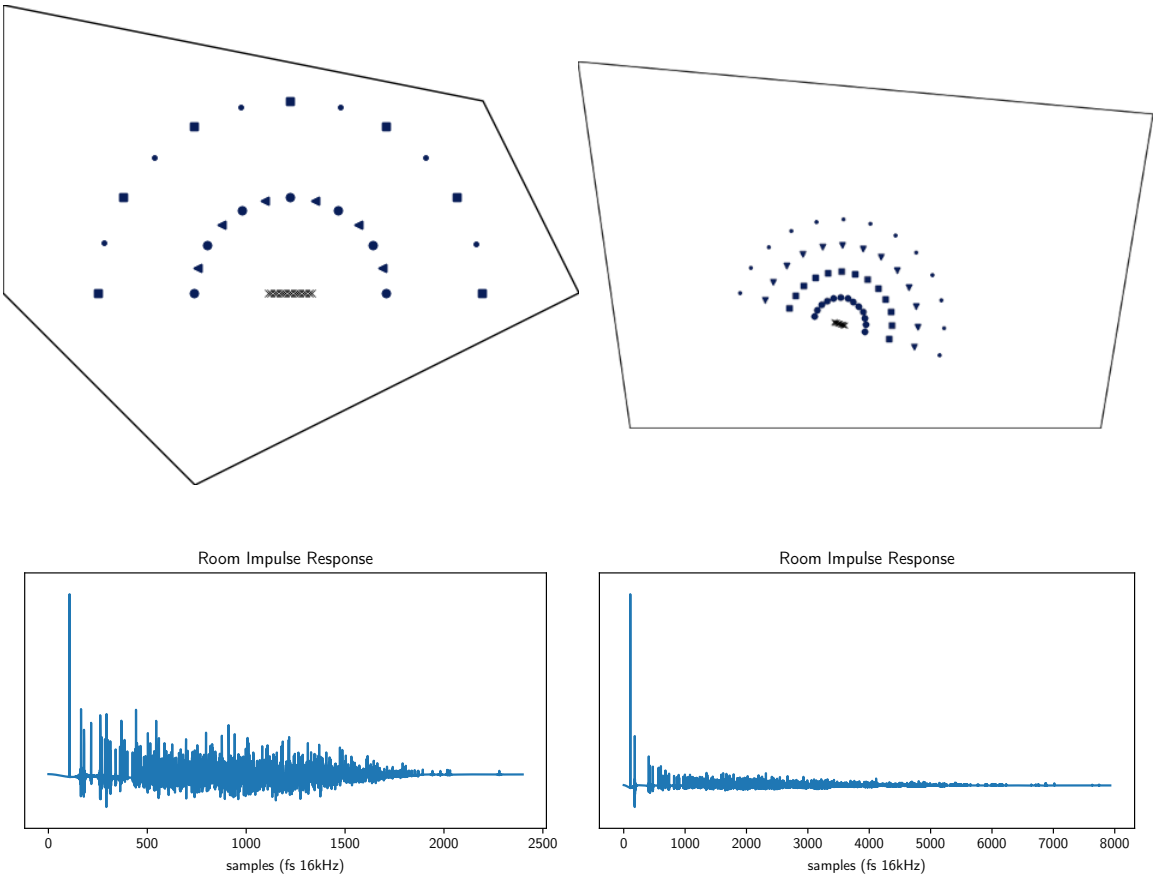


Figure 4.1. Two examples of simulated room shapes and a RIR from each. Crosses denote the positions of microphones, other shapes represent possible sources.

For each room geometry and arrangement, four sets of RIRs were simulated, with four different levels of reverb. For one group (no reverb), direct RIR were simulated, forgoing reverb entirely. The other three groups used wall materials taken from pyroomacoustics' materials database[21]. Their characteristic absorption coefficients for different frequency bands are detailed in Table 4.1.

Over the four different materials, and the six different room geometries, the obtained T60[1] values ranged from 0.05 s to 0.5 s.

[1]The time it takes for a sound to decay by 60dB, a common measure for reverberation time.[12]

Material (group)	Absorption coefficients						
	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz	8 kHz
wooden_lining (low damping)	0.27	0.23	0.22	0.15	0.10	0.07	0.06
curtains_cotton_0.5 (std damping)	0.3	0.45	0.65	0.56	0.59	0.71	0.71
curtains_fabric_folded (high damping)	0.12	0.60	0.98	1.00	1.00	1.00	1.00

Table 4.1. Different wall materials' absorption coefficients.

4.1.2 Conversation simulation

To create the simulated conversations, the clean utterance datasets — TIMIT and LibriSpeech — were first scanned, retrieving all identities and their corresponding utterances. Utt.s from the original partition of LibriSpeech named *train-clean-100* and the original partition of TIMIT named *TRAIN* were used to create the training simulations, while utt.s from the original partition of LibriSpeech named *test-clean* and the original partition of TIMIT named *TEST* were used for the testing simulations. Table 4.2 reports the total number of *diverse* utterances considered for training and testing simulations. Keep in mind that each utterance was employed several times in different conversation conditions.

	LibriSpeech	TIMIT	Total
Training	train-clean-100	TRAIN	
	28 539 utt.s	4 620 utt.s	33 159 utt.s
Testing	test-clean	TEST	
	2 620 utt.s	1 680 utt.s	4 300 utt.s

Table 4.2. Utterance counts for the different dataset partitions.

The following simulation process was repeated several times for each possible combination of

- room shape,
- wall material,
- number of involved identities (3, 5, or 7),
- white noise energy level, and

- overlap amounts.

The noise energy level was one of $\mathrm{SNR}_{\mathrm{db}} = 10, 20, \ldots, 90, 100, \infty$. White noise was only added to training simulations, and not testing simulations. As for the amount of overlap, several different ranges were defined. By "range" we mean the possible range for the delay between any two consecutive utterances, which could be positive (silence) or negative (overlap). Furthermore, some simulations were created with a constant overlap of two speakers, and some with three.

Iteratively, random utterances were assigned to random positions, convolved with the appropriate RIRs, and added to the simulated array recording in a consecutive fashion, with a random delay between each. A series of checks ensured that

- no two utterances from the same identity were ever played simultaneously,
- the amount of overlap or silence between consecutive utterances never strayed outside of the desired range,
- simultaneous speakers were never less than 0.3 rad apart w.r.t. the microphone array,
- all identities had at least one turn speaking, and all utterances were picked fairly.

We also took care to include each identity in at least one simulation. This process was repeated until the simulation reached the desired length — 120 s for training simulations, 30 s for testing simulations. With the simulated array recording ready, a different white noise signal was added to each channel of it, with the desired energy level.

For each simulation, two multi-channel recordings were produced: the simulation of the microphone array recording, with reverberant overlapped speech, and the "target" recording with each identity's clean speech isolated in a different channel. The clean recording was produced by employing a direct, reverb-free RIR between the sound source and a reference microphone, in order to maintain a similar propagation delay as that of the array simulation.

The resulting training dataset contains 121 463 utterances from 3 374 simulations, for a total of 258 hours of speech. The testing dataset contains 432 simulations, or 2.2 hours of conversations. Everything was done using a sampling rate of $F_s = 16\,\mathrm{kHz}$.

Figure 4.2 shows the percentages of the training data that represent a lone speaker, a 2-speaker overlap, or a 3-speaker overlap. Part of the data contains silence: this is because the beginnings and ends of utterances are padded with a second of silence.

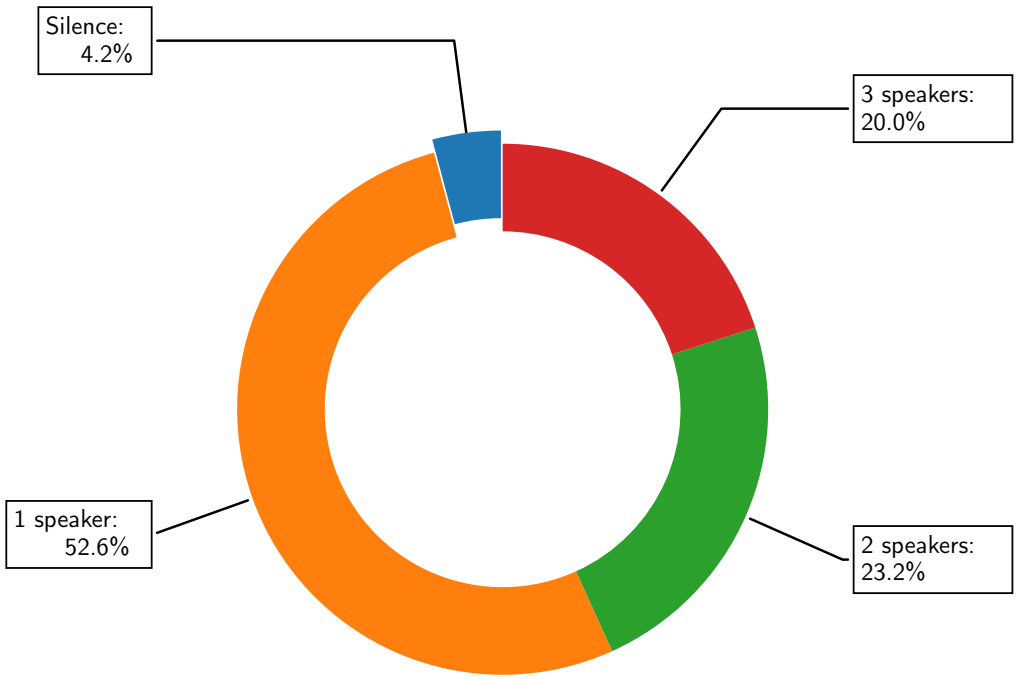


Figure 4.2. Percentages of the training data representing a lone speaker, a 2-speaker overlap, or a 3-speaker overlap.

This is done in order to ensure the network doesn't only train on segments from the middle of an utterance, which contain speech throughout, but also experiences segments from utterance boundaries, in which speech only fills part of the segment.

4.2 System setup and training

In this section we will first detail the general setup of the system used for training and testing, then the process and data used to train the network, and finally the parameters we used for the network implementation.

4.2.1 Windowing, beamforming, and clustering

Testing setup

The overall system setup, used for testing, processes conversation recordings by first dividing them in one-second-long segments. Rather than exactly one second, segments were cut at 16384 samples or 1.024 s in order to make the sample count divisible by

the STFT window length, 1024. Segments are advanced with a hop size of 4094, or 1/4 of a segment, for an overlap of 75%. Each segment is processed individually.

Using oracle prediction (i.e. consulting the dataset annotation), the system determines how many speakers are active within the segment, and their directions. For each located speaker, the appropriate beamformer is applied, and the beamformer's output is transformed with a STFT using a window length of 1024 samples i.e. 64 ms, a hop size of 256 samples i.e. 16 ms, and a Hamming windowing function. The resulting spectrum is fed to the neural network, producing the relative mask and embedding; the mask is applied to the spectrum and the isolated signal is reverse-transformed. The segment's temporal location, speaker embedding, and isolated signal are appended to a list, proceeding until the whole recording is processed.

After this is done, the embeddings are clustered using a SciKit[18] implementation of spectral clustering, and the resulting identities are also stored in the same list entries. All isolated signal segments are finally filtered with a Hamming window, and overlap-added at the appropriate temporal location, to the channel dictated by the assigned identity.

Training setup

During the training process, only the DNN was involved, to decrease the computational load. The training data was prepared in advance, each training utterance being cut into segments and processed by the appropriate beamformer just like it would during testing. Section 4.2.2 will detail the construction process for the training batches.

Dataset splits

Simulations for testing and for training were produced separately, using different utterances as introduced in Section 4.1.2, and with additional preparation for the training data as was just mentioned. Among the training data, seven identities were selected for validation, and all segments related to them were removed from the rest. These were used to monitor the training process, never being used for back-propagation.

4.2.2 Training batch construction

Mini-batches

As already mentioned, the training simulations were prepared in advance, with each utterance being divided in segments and processed by a beamformer. To create the training batch, the prepared training dataset was shuffled and preliminarily divided

in groups of 32. Then, mini-batches were built as follows. For each segment of a group two more segments are selected. The first shares the same identity of the anchor segment, while the second one contains speech from a different, randomly picked, identity. During the training phase, the complete mini-batch is considered for the computation of the masking loss L_{smooth}, while only a selection of triplets is considered in the computation of the embedding loss $L_{triplet}$. The selection process of these triplets is detailed in the following section.

Triplet selection

The triplet loss (see Equation (3.10)) works by comparing data points, as the name suggests, in groups of three. As Schroff et al.[22] explain, generating all possible triplets in the dataset would results in many triplets with small error (i.e. the embeddings of same-identity segments are already close and that of the different-identity segment is farther). These "easy" triplets would not contribute to the training, and result in slower convergence if included in the training data, so selecting the right triplets — ones that are hard enough to improve the model — is paramount.

For this purpose, we used an *online semi-hard triplet selection* approach, meaning that triplets were selected during training, within each mini-batch. The approach works as follows:

1. all elements of the mini-batch pass through the network for the forward-pass, generating their embeddings;

2. all possible anchor-positive-negative triplets in the mini-batch are located, and their loss values evaluated;

3. triplets are selected for the backward-pass if their negative embedding is close to the anchor embedding (within a margin), and yet farther from it than the positive embedding.

This is where the mini-batch generation approach described above comes into play. It ensures that each mini-batch contains *at least* b viable triplets for the triplet loss (where b is the batch size reported in Section 4.2.3), while also ensuring that every segment in the dataset is considered at least once in every epoch. Given the number of identities in the dataset, if mini-batches were generated with no regard for triplets, they would have a very low chance of presenting any same-identity pairs at all, let alone enough of them to make a meaningful choice of triplets.

4.2.3 Training parameters

To complete the overview of our training process, here we will detail all settings and parameters that were used to obtain the results presented in Chapter 5.

The size of the spectra used as inputs for the network, and of course the masks output by the network, are 513×64 (frequency×time bins). The embeddings were given a dimensionality of 64, and the margin for the online triplet loss was 1.

The first convolution block of the network has 64 channels, with the following steps having 128, 256, 512, and 256 again. Each layer's dimensions can be found in the appendix's Figure 1. As a further experiment, a more light-weight network was trained starting with only 16 channels in the top blocks, and descending to 32, 65, 128, and 256. The performance comparison can be seen in Chapter 5.

The two terms of the loss — mask and embedding — were weighted equally:

$$w_e = w_m = 1. \tag{4.1}$$

Training was carried out with an Adam optimization algorithm and a learning rate of 0.01, for 10 epochs, with mini-batches of 32×3 (32 triplets, see Section 4.2.2).

4.3 Evaluation metrics

Since the system performs two different tasks, the metrics will reflect that: we will be using separate metrics to evaluate the diarization correctness and the perceived quality of the masked signal.

4.3.1 Identification performance

To evaluate diarization, we compute the Diarization Error Rate (DER)[2]. DER is the *de facto* standard metric for evaluating and comparing speaker diarization systems, defined as follows:

$$\text{DER} = \frac{\text{false alarm} + \text{missed detection} + \text{confusion}}{\text{total}} \tag{4.2}$$

where *false alarm* is the duration of non-speech incorrectly classified as speech, *missed detection* is the duration of speech incorrectly classified as non-speech, *confusion* is the duration of speaker confusion (intervals assigned to the incorrect identity), and *total* is the total duration of speech in the reference annotation. All durations are usually measured in seconds, but being a ratio, this metric is independent from the unit used to measure conversation and overlaps.

We will show both the DER based on the utterance-aware clustering approach mentioned in Section 3.2.5, and the "naive" DER based directly on the embeddings. This metric is meant to evaluate the performance of embeddings-based identification of segments, and not the accuracy of segmentation in locating speaker changes. For that reason, the annotation used as *ground truth* was created by assigning each segment to the true speakers active during it, rather than exactly annotating the start and end of each utterance.

4.3.2 Separation performance

For the assessment of the quality/intelligibility of the mask-isolated signal, we compute two different metrics. We start with the well-known metric PESQ[19]. PESQ was developed to assess speech quality as perceived by a user of a telecommunications system, and can be seen as an objective voice quality testing metric. It compares the signal to be evaluated, in this case the mask-isolated beamformer signal, with a clean reference, in this case the clean speech from the original dataset. Please refer to the original publication for the exact definition.

We also consider the Word Error Rate (WER), using a speech recognition algorithm included in Sphinx[11]. In particular, the output signals of the network are fed to the speech-to-text agent and the resulting transcriptions are compared with the ground-truth from the original speech dataset.

The WER is defined as:

$$\text{WER} = \frac{S + D + I}{N} \tag{4.3}$$

where S is the number of substitutions (words that were misinterpreted as the wrong word), D is the number of deletions (words that were omitted entirely), I is the number of insertions (words that do not appear in the reference transcription, and did not substitute any word), and N is the number of words in the reference transcription.

Chapter 5

Results

Having illustrated the system's architecture, the approach used for training and testing it, and the metrics used to evaluate its performance, here are the results obtained in our tests.

Section 5.1 details the average metrics values over all of our testing data, which is comprised of several simulations with different room geometries, wall materials, numbers of involved speakers, and overlap times, as detailed in Section 4.1.2. Some of these parameters showed no influence on the system's performance (i.e. room geometry and number of involved speakers), while significant differences can be seen when grouping the testing data based on wall material (Section 5.2) or overlap times (Section 5.3).

In all results, the signal used to evaluate both PESQ and WER was the one based on utterance-aware clustering, and not the one that used "naive" clustering.

5.1 General results

DER performance

Figure 5.1 shows the Diarization Error Rate. It is evaluated based on both a simple spectral clustering of the network's embeddings, and the utterance-aware clustering approach mentioned in Chapter 3. Since the DER is fundamentally the percentage of signal that was incorrectly assigned, the objective is reducing it to zero. It is then obvious that the utterance-aware approach, achieving a DER under 0.01, is a tremendous improvement over the clustering of isolated embeddings, which performs a 0.24 DER.

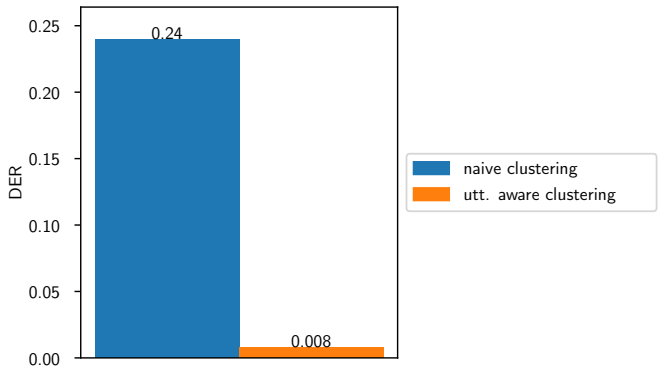


Figure 5.1. Diarization Error Rate measured over all testing data.

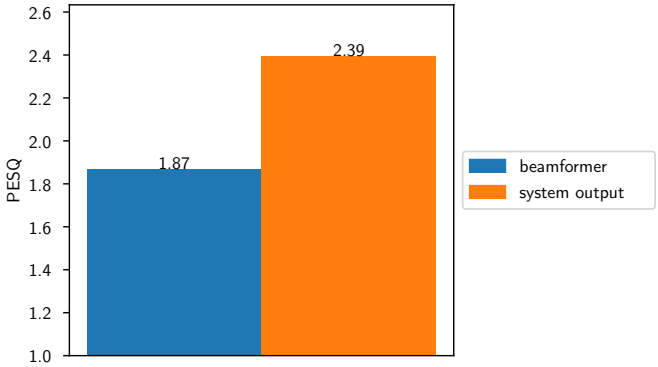


Figure 5.2. PESQ measured over all testing data.

PESQ performance

Figure 5.2 shows the PESQ grade, calculated w.r.t. the clean signal from the original dataset. It is evaluated for the raw beamformer signal, and for the network-isolated speech signal. PESQ values range from 1 to 5, with 1 being an incomprehensibly corrupted signal, and 5 a perfect transmission. The system performance of 2.39 is far from perfect, but still much higher than the beamformer's 1.87.

WER performance

Figure 5.3 shows the Word Error Rate. It is evaluated for the beamformer signal, the network-isolated speech, and the clean signal from the original dataset. Like the DER, the WER is fundamentally the percentage of error, and the objective is reducing it to zero. For this metric, we include the performance of the objective signal, which serves as reference for the maximum possible performance from the system. This would have been pointless for DER and PESQ, since the reference would of course have scored 0

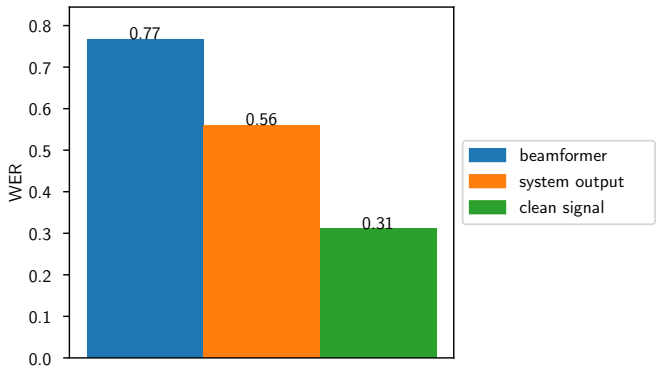


Figure 5.3. Word Error Rate measured over all testing data.

and 5 respectively, but when an automatic speech recognition algorithm is applied even the reference produces imperfect transcriptions with an average score of 0.31. With that in mind, the system's score of 0.56 is a considerable improvement on the beamformer's 0.77.

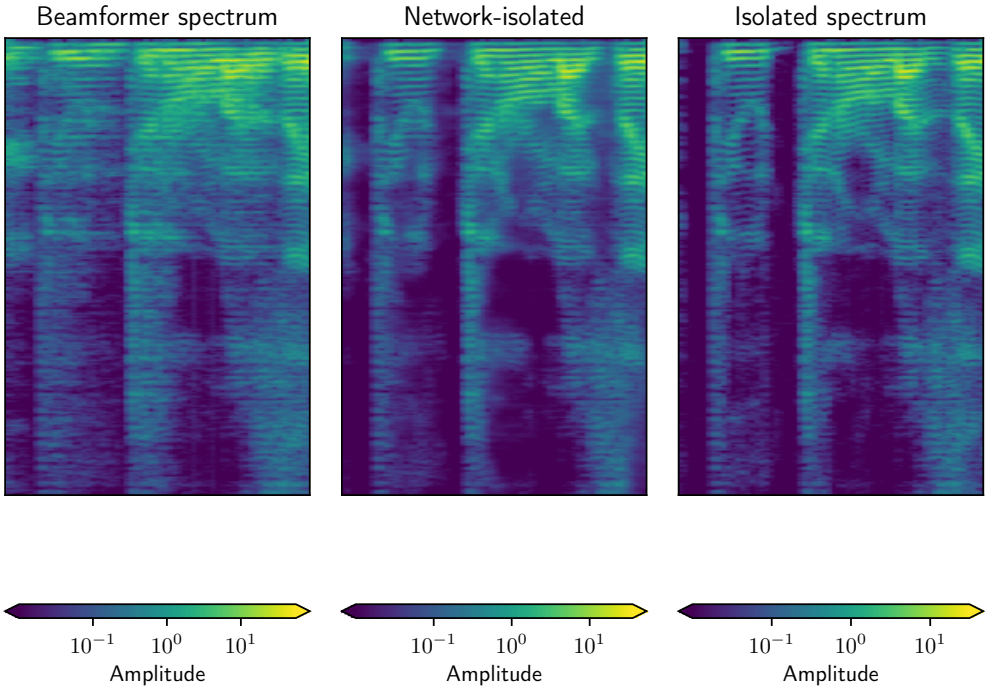
Spectrum examples

Figure 5.4 shows a pair of comparisons between a beamformer signal, the signal isolated by the network's mask, and the reference clean speech signal. In the first comparison, there are no overlaps between speakers, only reverb. The network clearly reduces the effects of reverb greatly when compared to the beamformer. In the second comparison there is a severe overlap of speakers, in addition to the reverb. the network successfully recognizes all parts of the spectrum related to the target speaker and those related to disturbances, greatly enhancing the clean speech signal.

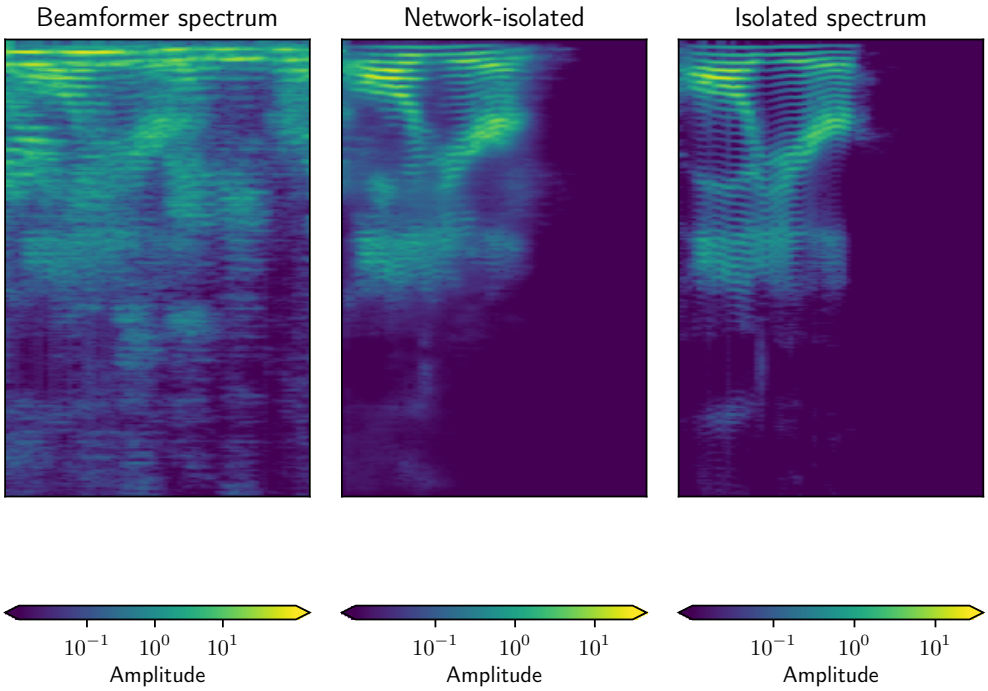
Figure 5.5 shows three comparisons between objective IRMs and the relative masks produced by the network. The first comparison only includes reverb, with no speaker overlaps. The second includes overlaps, but no reverb, and the third includes both. In all situation, it is apparent that the network is successful in distinguishing between the features of the targeted speaker and those of disturbances.

5.2 Reverb-specific results

For the following evaluations, the testing data was divided in four groups based on the wall absorption coefficients used in the room simulations. That is the only criterion in the division of these groups — each of them contains simulations spanning different room geometries (and thus T60 values), overlap amounts, and so on. At the end of

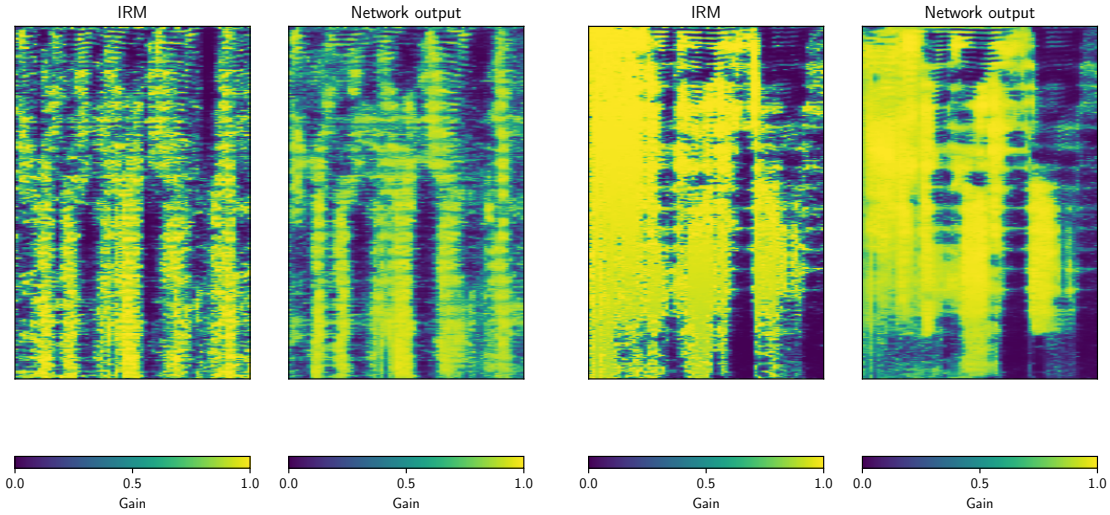


(a) in a situation with reverb, but no overlap



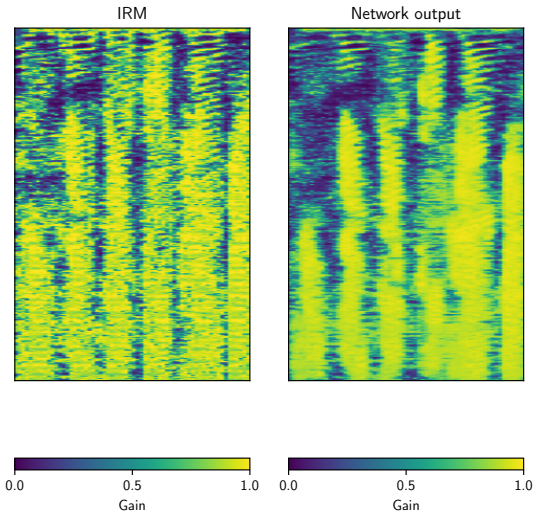
(b) in a situation with both overlap and reverb

Figure 5.4. Comparison between ground-truth clean speech signals and signals isolated by the network's output masks.



(a) in a situation with reverb, but no overlap

(b) in a situation with overlap, but no reverb



(c) in a situation with both overlap and reverb

Figure 5.5. Comparison between ground-truth IRMs and the predictions made by the network.

the section we will make some considerations on the relation of wall materials and T60 values. One group (no reverb) consists of simulations that exclusively used direct RIR, thus introducing no reverb. The other three groups are based on the wall materials as detailed in Table 4.1.

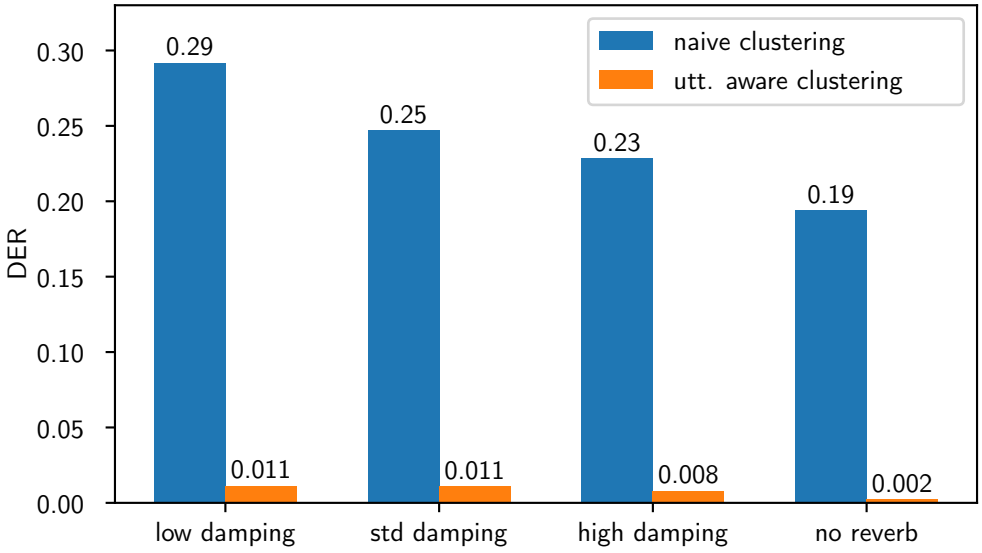


Figure 5.6. Diarization Error Rate for different wall materials.

Figure 5.6 shows the Diarization Error Rate based on simple clustering and on utterance-aware clustering. Just like for the general results, the utterance-aware approach is a colossal improvement over the naive clustering. As for the relationship with reverb levels, both approaches seem to suffer in situations with lower reverb damping, going from 0.19 to 0.29 with the naive approach, and from 0.002 to 0.011 with the utterance-aware approach.

Figure 5.7 shows the PESQ grade for the beamformer signal and for the network-isolated speech. Both suffer in low-damping situations, but the system always outperforms the beamformer. Something to note is that the beamformer abruptly drops in performance in situations with any reverb at all, dropping off from the reverb-free score of 2.97 to 1.7 even for the high-damping scenarios. The system, on the other hand, drops off performance more gradually, going under 2.0 only with the introduction of stronger reverb. The reason for the beamformer's behaviour is a lack of directionality in the beamformer's response. As Figure 2.3 showed, the main lobe in the beamformer's response is narrow for high frequencies, but gets progressively wider for lower frequencies, eventually failing to provide any directionality for near-0-frequency signals. Furthermore, all considered material coefficients (reported in

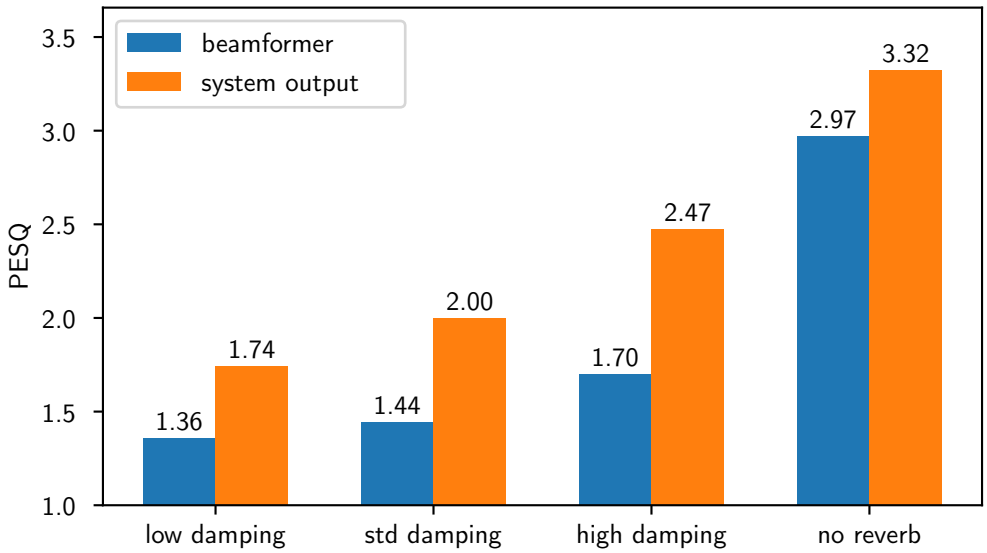


Figure 5.7. PESQ for different wall materials.

Table 4.1) offer relatively low damping in the low frequencies, even ones that achieve "perfect" damping in the high frequencies.

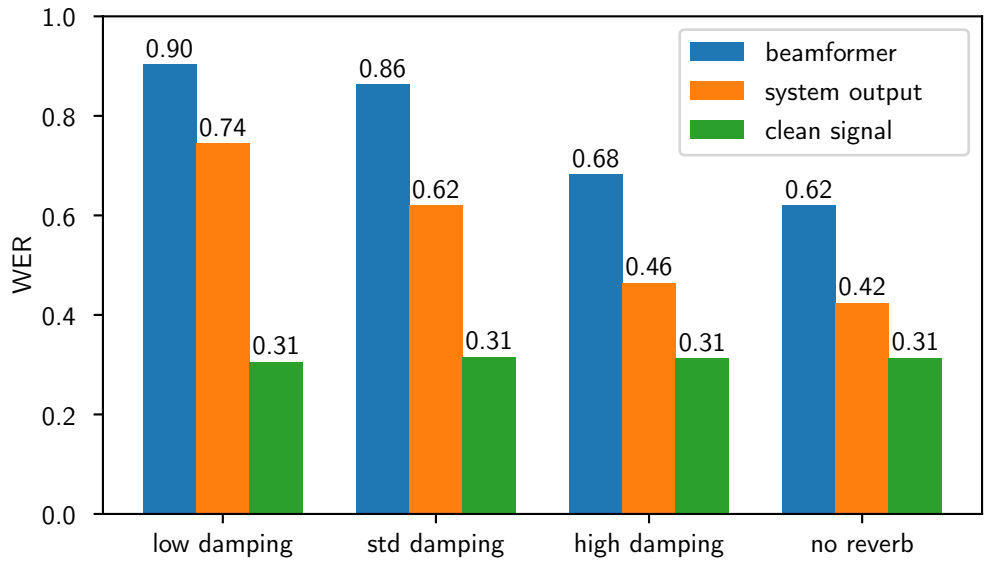


Figure 5.8. Word Error Rate for different wall materials.

Figure 5.8 shows the Word Error Rate for the beamformer signal, the network-isolated speech, and the clean signal from the original dataset. The clean signal's performance is obviously unaffected by the simulation settings, being 0.31 in all cases. The beamformer and the system, as we saw for the previous results, suffer from

the presence of strong reverb. This time the beamformer does not show an abrupt improvement in the no-reverb scenarios, instead showing poor performance in most situations, from 0.62 to 0.9. The system, on the other hand, gets rather close to the ideal performance when reverb is low or absent (0.46, 0.42) and quickly drops off when stronger reverb is introduced, reaching 0.74.

The system clearly benefits from high-damping environments, for both identification and isolation. It is important to note that the same distinction cannot be made when grouping the testing data by room geometry, *nor by T60*. In fact, the test results for T60-based groups (which are not reported here) did not show any correlation between T60 and system performance, for any evaluation metric. The performance's independence from room geometry and T60 suggests that the system is more susceptible to the reverb's behaviour in different frequency bands, rather than simple reverb duration.

5.3 Overlap-specific results

For this set of evaluations, the testing data was divided in four groups based on the amount of overlaps present in simulations. One group of simulations was created with "realistic" overlaps, meaning that the delay between each pair of consecutive utterances was randomly chosen between -0.1 s and 1.0 s, with negative delay meaning overlap and positive delay meaning silence between speakers. Another group of simulations used more "severe" overlaps, with delays being randomly chosen between -1.0 s and 1.0 s. The third group of simulations is characterized by constant overlap of two speakers, with virtually no time spent with fewer or more than that, and the fourth group does the same with three speakers.

It should be noted that, while our simulation setup considers at most three simultaneous speakers, the number of simultaneous speakers the system can manage is virtually unlimited. With the DNN only aiming at isolating the one speaker that the beamformer is aimed towards, permutation of the network outputs is not a problem as it is in systems that isolate all speakers "at once". The bottleneck for maximum simultaneous speaker capacity falls instead on the DOA module's ability to locate all sources accurately, the beamformer's ability to dampen all non-targeted sources, and the training data experienced by the network.

Figure 5.9 shows the Diarization Error Rate based on simple clustering and on utterance-aware clustering. Once again, the utterance-aware approach is hardly comparable to the naive clustering. This time, both approaches show a little less dependence to the different conditions. The utterance-aware approach goes from 0.004

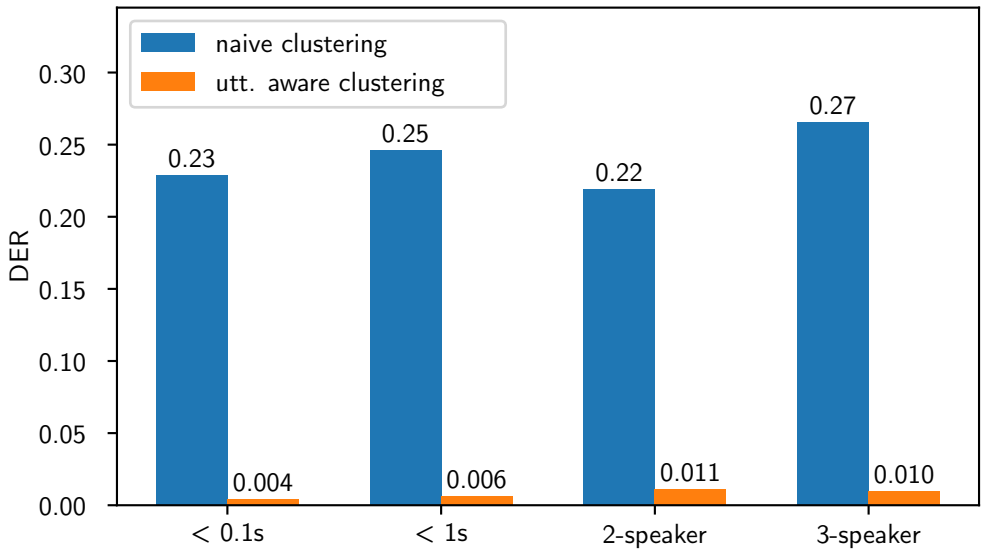


Figure 5.9. Diarization Error Rate for different overlap times.

to 0.011 when overlaps increase, but stays relatively still at 0.01 when the overlaps involve three speakers instead of two. The naive approach moves similarly up and down, seemingly unaffected by the amount of overlap introduced.

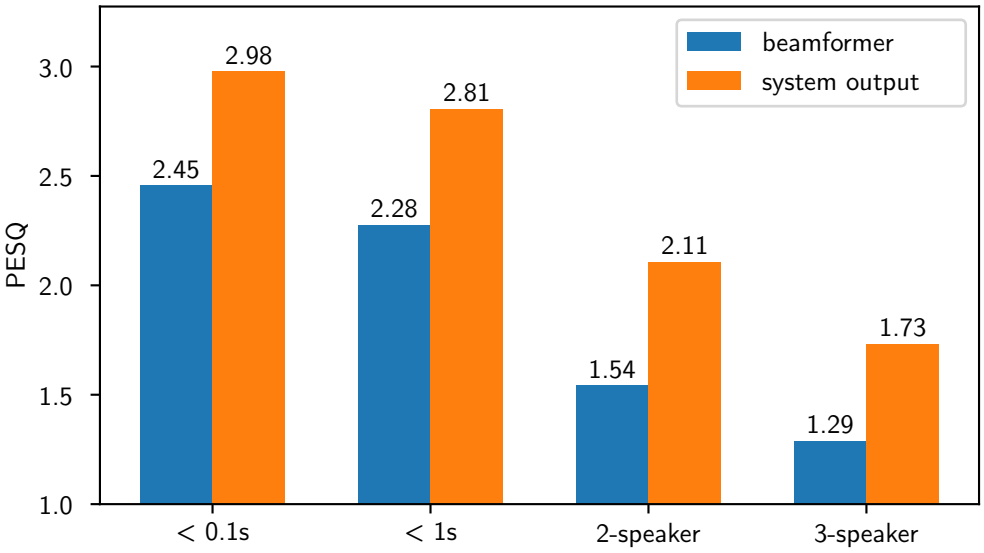


Figure 5.10. PESQ for different overlap times.

Figure 5.10 shows the PESQ grade for the beamformer signal and for the network-isolated speech. This time the difference between two-speaker and three-speaker scenarios is apparent, but still not as significant as the difference with more "realistic"

scenarios. Both the beamformer and the system are affected equally, dropping respectively from $2.45, 2.28$ to $1.54, 1.29$ and from $2.98, 2.81$ to $2.11, 1.73$.

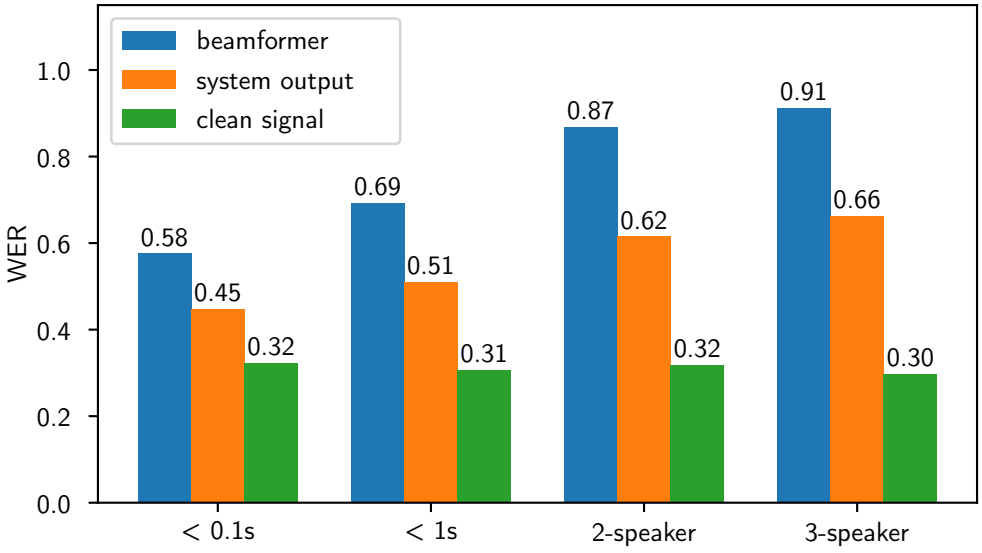


Figure 5.11. Word Error Rate for different overlap times.

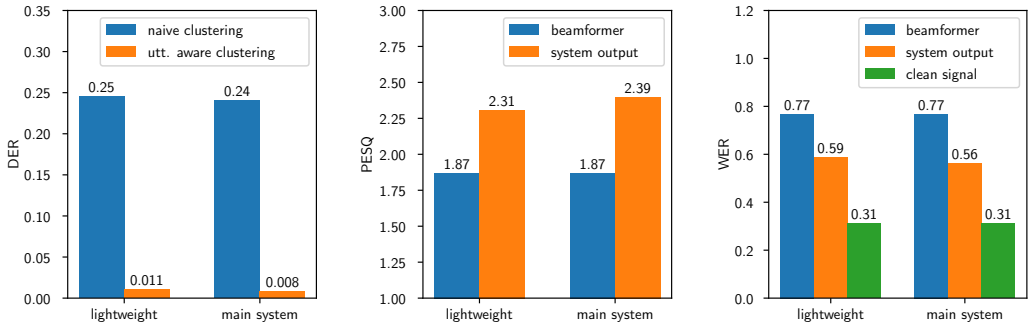
Figure 5.11 shows the Word Error Rate for the beamformer signal, the network-isolated speech, and the clean signal from the original dataset. The clean signal's performance is only shown to vary a little because the different groups of simulations include different utterances. The beamformer performance substantially drops in the presence of overlaps, reaching highs of 0.87 and 0.91 with two and three speakers present. The system, as always, outperforms the beamformer. It shows some weakness to overlaps, although not as much as it does to bad reverb conditions.

For both identification and isolation, the dependence on overlap amounts seems to be less severe than that on reverb conditions. The DER went from 0.002 to 0.011 at the change of reverb, from 0.004 to 0.011 at the change of overlaps; PESQ went from 3.32 to 1.74 at the change of reverb, from 2.98 to 1.73 at the change of overlaps; WER went from 0.42 to 0.74 at the change of reverb, from 0.45 to 0.66 at the change of overlaps.

5.4 Lightweight system

As anticipated in Section 4.2.3, a secondary network with fewer parameters was trained and tested. Figure 5.12 shows the performance comparison between that light-weight network and the full one. The beamformer and clean signal performances are obviously

unchanged, being independent from the network. The DER, with the utterance-aware approach, went from 0.008 up to 0.011; PESQ went down from 2.39 to 2.31; WER went up from 0.56 to 0.59.



(a) Diarization Error Rate (b) PESQ (c) Word Error Rate

Figure 5.12. Performance comparison between full system and lightweight variant.

While there is a small drop in performance for the smaller network, it is clearly a good substitution if computational power is scarce. Future works could further investigate how fewer-parameters versions of the system behave in different conditions, including with different array setups and beamforming algorithms as well as different room and conversation characteristics.

Chapter 6

Conclusions and future work

6.1 Conclusions

The focus of research in the fields of Diarization and Speech Separation is quickly shifting towards the integration of the two tasks for the advantage of both. For this reason, in this thesis we explored a novel approach for the integration of said tasks, by designing a system that simultaneously isolates different speakers' voices from a mix and assigns each to a channel based on the speaker's identity. For the purpose of training the designed system, we also produced a dataset of simulated conversations in reverberant rooms.

The proposed system is based on a beamformer for preliminary separation, and a DNN for identification and for the final separation. The relative independence of these two modules allows each to be improved or customized individually. The beamformer is based on a linear microphone array and a delay-and-sum algorithm, producing a set of signals targeting each concurrent speaker. Each of these signals is then handled by the custom-designed DNN.

The neural network takes a STFT of the partially-separated speaker as input, and employs a U-Net approach to handle both identification and further separation. The Encoder branch of the network gradually reduces the data in size, extracting features that finally produce speaker embeddings to be used for clustering, for identification. The Decoder branch of the network takes several skipped connections from the Encoder branch — both low- and high-level features. It produces an isolation mask to be applied to the input spectrum, exploiting the information about which features are related to the target speaker and which to disturbances.

Identification of the speaker embeddings is carried out by first performing Spectral clustering, and subsequently grouping the results based on the utterance of origin (with utterances being located based on the detected DOA and speaker change detection).

The groups of clustering results are analyzed and the most prominent guess is selected for each utterance. The system's final output is created by applying the isolation masks to the beamformer signals, and recording the resulting isolated signals to the identity-specific output channels based on the results of the embedding clustering.

The dataset was created in two steps. First, acoustic simulation software was employed to generate the RIRs of a set of rooms with different geometries and wall materials, considering a linear microphone array and several sound sources around it. Next, clean utterances from the datasets LibriSpeech and TIMIT were used to simulate conversations in the generated rooms, with different amounts of overlap between speakers and number of speakers involved in each conversation.

The designed system's performance was evaluted separately for identification accuracy — using DER — and isolation intelligibility — using PESQ and WER. Testing results show a clear improvement in quality over the simple beamformer approach, and excellent identification accuracy. Targeted testing also shows that the system has some weakness to excessive overlaps between speakers, and/or excessively reverberant environments. Finally, additional testing showed that a light-weight version of the network employing fewer parameters only slightly reduces performance, making it an acceptable substitute for quick-and-dirty use cases.

6.2 Future work

In this thesis we deliberately focused on the design of the DNN and employed very basic methods for DOA estimation, beamforming, and clustering. This decision was taken knowing that the modular architecture would allow for independent improvement of those parts. The following are some possibilities for future work.

- A different microphone array geometry, such as a circular array, would enable 360° coverage instead of just 180°.
- A more advanced beamforming technique, such as the Capon method, might improve the preliminary speaker separation.
- If the network's input included phase information (in particular phase differences between microphones) instead of just magnitude, it might aid the network in separating different sources.
- Including ambient noise and non-vocal disturbances in the network's training dataset might improve its robustness and versatility.

- As shown by Wang et al.[25], the results of spectral clustering can be improved through a series of refinement operations. In our case, these operations may be extended to include a DOA-aware step, or in some other capacity, information related to the utterance limits, as we employed for the utterance-aware clustering approach.
- Additionally, with a sufficiently adaptive clustering approach, it would be possible to operate the system in real time with latency proportional to the time segment fed to the network, and/or let the system guess the number of involved identities.

Appendix

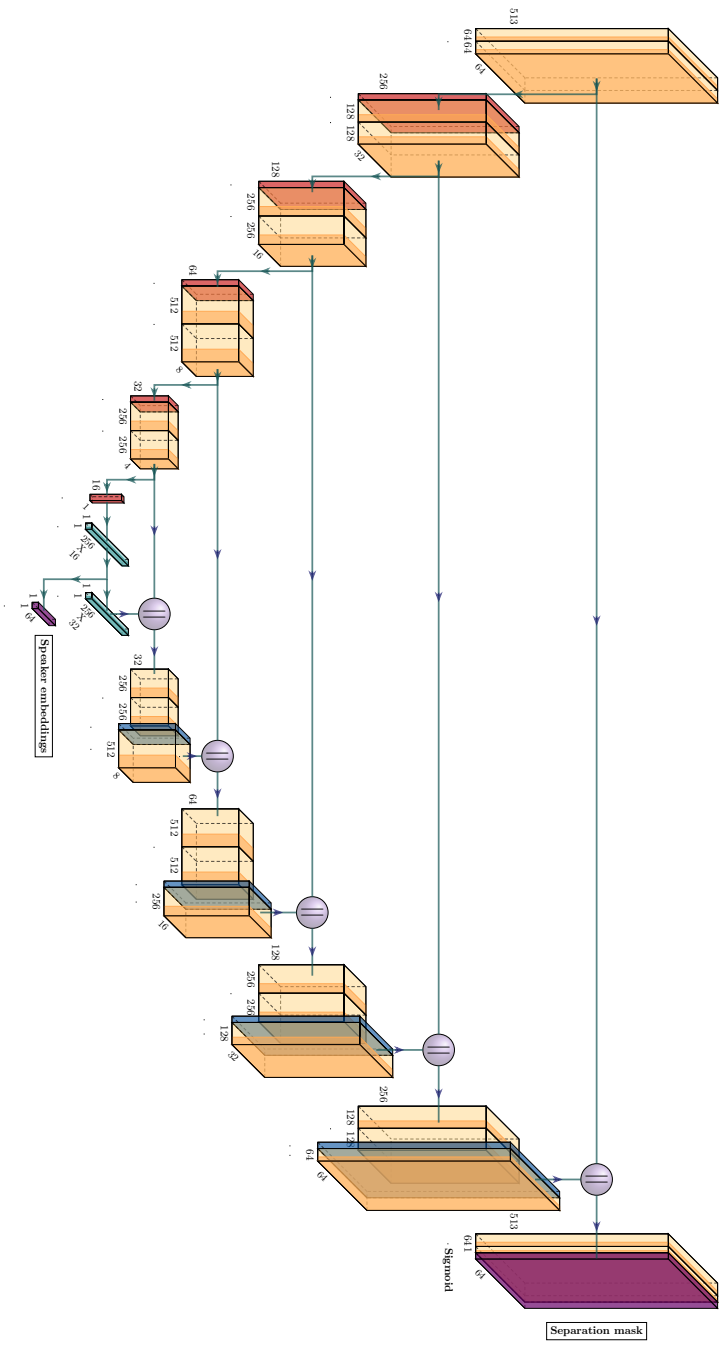


Figure 1. The proposed network.

Acronyms

HCI	Human-Computer Interaction
DOA	Direction Of Arrival
DL	Deep Learning
ML	Machine Learning
PIT	Permutation Invariant Training
RNN	Recurrent Neural Network
DNN	Deep Neural Network
DNNs	Deep Neural Networks
CNN	Convolutional Neural Network
CNNs	Convolutional Neural Networks
RIR	Room Impulse Response
DER	Diarization Error Rate
WER	Word Error Rate
STFT	Short Time Fourier Transform
PESQ	Perceptual evaluation of speech quality
IRM	Ideal Ratio Mask

Bibliography

[1] Miguel Blanco Galindo, Philip Coleman, and Philip Jackson. "Microphone array geometries for horizontal spatial audio object capture with beamforming". In: *Journal of the Audio Engineering Society (AES)* (2020).

[2] Hervé Bredin. "pyannote. metrics: A Toolkit for Reproducible Evaluation, Diagnostic, and Error Analysis of Speaker Diarization Systems." In: *INTERSPEECH*. 2017, pp. 3587–3591.

[3] Zhuo Chen et al. "Cracking the cocktail party problem by multi-beam deep attractor network". In: *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE. 2017, pp. 437–444.

[4] Zhuo Chen et al. *Multi-channel speech separation*. US Patent 10,839,822. Nov. 2020.

[5] Cheng-Yang Fu, Mykhailo Shvets, and Alexander C Berg. "RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free". In: *arXiv preprint arXiv:1901.03353* (2019).

[6] Yusuke Fujita et al. "End-to-end neural speaker diarization with self-attention". In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE. 2019, pp. 296–303.

[7] John Garofolo et al. *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. Version V1. 1993. DOI: 11272.1/AB2/SWVENO. URL: https://hdl.handle.net/11272.1/AB2/SWVENO.

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

[9] Andrew Greensted. URL: http://www.labbookpages.co.uk/audio/beamforming/delaySum.html (visited on 03/25/2021).

[10] Cong Han et al. "Continuous Speech Separation Using Speaker Inventory for Long Multi-talker Recording". In: *arXiv preprint arXiv:2012.09727* (2020).

[11] David Huggins-Daines et al. “Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices”. In: *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*. Vol. 1. IEEE. 2006, pp. I–I.

[12] *Acoustics — Measurement of room acoustic parameters — Part 2: Reverberation time in ordinary rooms*. Tech. rep. International Organization for Standardization, June 2008.

[13] MartinThoma. URL: `https://github.com/MartinThoma/LaTeX-examples/tree/master/tikz/convolution-linear` (visited on 03/25/2021).

[14] MartinThoma. URL: `https://github.com/MartinThoma/LaTeX-examples/tree/master/tikz/max-pooling` (visited on 03/25/2021).

[15] Andrew Ng, Michael Jordan, and Yair Weiss. “On spectral clustering: Analysis and an algorithm”. In: *Advances in neural information processing systems* 14 (2001), pp. 849–856.

[16] Vassil Panayotov et al. “Librispeech: an asr corpus based on public domain audio books”. In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, pp. 5206–5210.

[17] Tae Jin Park et al. “A Review of Speaker Diarization: Recent Advances with Deep Learning”. In: *arXiv preprint arXiv:2101.09624* (2021).

[18] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[19] Antony W Rix et al. “Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs”. In: *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*. Vol. 2. IEEE. 2001, pp. 749–752.

[20] Amitrajit Sarkar et al. “Says who? deep learning models for joint speech recognition, segmentation and diarization”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 5229–5233.

[21] Robin Scheibler, Eric Bezzam, and Ivan Dokmanić. “Pyroomacoustics: A python package for audio room simulation and array processing algorithms”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 351–355.

[22] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[23] T.E. Tuncer and B. Friedlander. *Classical and Modern Direction-of-Arrival Estimation*. Elsevier Science, 2009. ISBN: 9780080923079. URL: https://books.google.it/books?id=1aQbxKJI2CsC.

[24] Peidong Wang et al. "Speech separation using speaker inventory". In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE. 2019, pp. 230–236.

[25] Quan Wang et al. "Speaker diarization with lstm". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 5239–5243.

[26] Zhong-Qiu Wang and DeLiang Wang. "Combining spectral and spatial features for deep learning based blind speaker separation". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.2 (2018), pp. 457–468.

[27] Takuya Yoshioka et al. "Low-latency speaker-independent continuous speech separation". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 6980–6984.

[28] Dong Yu et al. "Permutation invariant training of deep models for speaker-independent multi-talker speech separation". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 241–245.

[29] Neil Zeghidour and David Grangier. "Wavesplit: End-to-end speech separation by speaker clustering". In: *arXiv preprint arXiv:2002.08933* (2020).

[30] Zhaofeng Zhang et al. "Deep neural network-based bottleneck feature and denoising autoencoder-based dereverberation for distant-talking speaker identification". In: *EURASIP Journal on Audio, Speech, and Music Processing* 2015.1 (2015), pp. 1–13.

[31] Kateřina Žmolíková et al. "SpeakerBeam: Speaker aware neural network for target speaker extraction in speech mixtures". In: *IEEE Journal of Selected Topics in Signal Processing* 13.4 (2019), pp. 800–814.