POLITECNICO
MILANO 1863

Dipartimento di Elettronica, Informazione e Bioingegneria

Master Degree in Computer Science and Engineering

Speaker-Independent Microphone Identification via Blind Channel Estimation in Noisy Condition

Author:
Antonio Giganti

Id. Number:
925671

Supervisor:
Prof. Paolo Bestagini

Co-supervisor:
M.Sc. Luca Cuccovillo

Academic Year
2020-2021

Abstract

In recent years, we have witnessed a radical change in the way information is exchanged, moving from simple text-based communications to the use of multimedia elements such as audio and/or video messages. This trend has been facilitated by the increasing speed of network connections as well as the low cost of mobile-phones.

However, technological advancements often pave the way to illegal use and new threatening scenarios that were previously unthinkable. For instance, voice recordings are often subject to tampering, whose aim ranges from deliberate manipulation to identity theft. It is no coincidence that the field of audio forensics is attracting interest among the scientific community, with an increasing number of publications on techniques for audio recordings analysis.

Within this context, a relevant problem in forensic investigations is that of *device identification*. Solving this problem means to recover information useful to trace the device that produced the speech recording under analysis. These traces are always left on each recording during the acquisition phase, and different methods in the literature have been proposed to extract them. From a forensic perspective, this information constitutes the *fingerprint* (or signature) of the adopted device and it is used as a discriminating element in the device identification process.

In this work, a method for device identification from speech recordings is proposed. The considered fingerprint is based on an estimate of the spectral fluctuations made by the device's microphone during the acquisition. In the literature this procedure is known as *channel estimation*. However, the process of extracting this fingerprint requires an audio recording as free as possible from external disturbances, such as speech or noise. It is well known that the presence of these undesirable signals causes a clear deterioration of the channel estimation, resulting in the unreliability of the device identification process. For this reason, we focus on the challenging problem of device identification in noisy condition exploiting denoising techniques based on neural networks.

In validating the effectiveness of the method, we formulate the problem in a *closed-set* scenario, where the number of possible devices is limited and known in advance. The results confirm our theoretical formulation, showing a significant increase in performances with respect to the model adopted as baseline, thus improving the final reliability of the device identification process in presence of speech recordings corrupted by noise.

Sommario

Negli ultimi anni, abbiamo assistito ad un cambiamento radicale nella modalità di scambio di informazioni, passando da semplici comunicazioni testuali all'utilizzo di elementi multimediali come audio e/o video messaggi. Questa tendenza è stata facilitata dalla crescente velocità delle connessioni di rete nonché dal basso costo dei telefoni cellulari.

Come spesso accade però, la tecnologia va di pari passo con l'illegalità, aprendo le porte a possibili scenari che prima d'ora erano impensabili. Dal furto d'identità, alla manipolazione volontaria, le registrazioni vocali sono spesso vittime di manomissioni atte a distorcere il loro reale contenuto. Non è un caso se nell'analisi forense di file multimediali, il settore dell'audio forense stia raccogliendo un crescente interesse tra la comunità scientifica, con un numero sempre maggiore di pubblicazioni riguardanti tecniche per l'analisi delle registrazioni audio.

In questo contesto, un problema rilevante nelle indagini forensi è quello dell'*identificazione del dispositivo*. Questo problema si pone l'obiettivo di recuperare informazioni utili per risalire al dispositivo che ha effettuato la registrazione vocale presa in analisi. Il dispositivo lascia sempre delle tracce intrinseche su ogni sua registrazione durante la fase di acquisizione e in letteratura sono stati proposti diversi metodi per estrarle. Nell'ottica forense, queste informazioni costituiscono l'*impronta* (o firma) del dispositivo adottato e verranno utilizzate come elemento discriminante nel processo di identificazione.

In questa tesi proponiamo un metodo per l'identificazione del dispositivo partendo da una registrazione vocale. L'impronta considerata si basa su una stima delle modifiche spettrali applicate dal microfono del dispositivo in fase di acquisizione. Questa procedura è nota in letteratura come *stima del canale*. Il processo di estrazione di questa impronta necessita però di una registrazione audio che sia il più possibile esente da componenti esterne, come il parlato o rumore. È ben noto infatti come questi segnali provochino un netto deterioramento della stima, con la conseguente diminuzione dell'affidabilità nell'identificare il dispositivo utilizzato. Per questo motivo, ci concentriamo sul complesso problema dell'identificazione del dispositivo in condizioni rumorose sfruttando tecniche di denoising basate su reti neurali.

Nel validare l'efficacia del metodo, formuliamo il problema in uno scenario *closed-set*, dove il numero di dispositivi possibili è limitato e noto in precedenza. I risultati ottenuti confermano la nostra formulazione teorica, ottenendo un notevole incremento rispetto al modello adottato come riferimento, migliorando quindi l'affidabilità finale nell'identificazione del dispositivo di acquisizione in presenza di registrazioni vocali corrotte da rumore.

Ringraziamenti

Un sentito grazie va *in primis* a tutta la mia famiglia, a mia sorella e ai miei parenti milanesi per avermi dato la possibilità di raggiungere questo obiettivo, sostenendomi lungo tutto il percorso accademico.

Grazie ai miei amici di una vita, perché la loro sincera vicinanza mi ha permesso di credere in me stesso ed arrivare a questo traguardo. Un ringraziamento particolare va ai miei compagni di studio, con i quali durante questo percorso si è instaurato un rapporto di amicizia che va oltre la semplice vita universitaria. Ringrazio tutti coloro che durante questo periodo accademico hanno lasciato la città perseguendo i loro obiettivi, ma nonostante ciò mi sono sempre accanto. Non posso dimenticare di ringraziare la città di Milano, che ormai considero come una seconda casa.

Un ringraziamento speciale va infine al mio relatore prof. Paolo Bestagini e al dott. Luca Cuccovillo per i preziosi consigli, il continuo supporto e la pazienza avuta lungo tutto questo lavoro di tesi a distanza. Grazie di cuore per l'enorme professionalità.

A.G.

Contents

Abstract **i**

Sommario **iii**

Ringraziamenti **v**

List of Figures **x**

List of Tables **xi**

Glossary **xii**

1 Introduction **1**

2 Theoretical Background **5**

2.1 Signal Processing 5

2.1.1 Time-Frequency Analysis 5

2.1.2 Mel-Frequency Cepstral Coefficients 8

2.1.3 RelAtive Spectral TrAnsform Filter 10

2.2 Machine Learning and Deep Learning Fundamentals . . . 11

2.2.1 Overview 11

2.2.2 Taxonomy 12

2.2.3 Architectures Used in this Work 14

2.3 Conclusive Remarks 27

3 State of the Art and Problem Statement **28**

3.1 State of the Art on Source Attribution 28

3.1.1 Device Identification 29

3.1.2 Device Identification in Noisy Conditions 31

3.2 Problem Statement 32

4 Proposed Method 36
4.1 Overview . . . 36
4.2 Building Blocks . . . 39
4.2.1 Clean Speech Model . . . 39
4.2.2 Denoising . . . 43
4.2.3 Blind Channel Estimation . . . 46
4.2.4 Feature Computation . . . 52
4.2.5 Identification . . . 54
4.3 Conclusive Remarks . . . 55

5 Experiments and Results 56
5.1 Experiment Setup . . . 56
5.1.1 Datasets . . . 56
5.1.2 Parameters and Routines Design . . . 58
5.1.3 Evaluation Metrics . . . 60
5.2 Preliminary Experiments for Method Design . . . 61
5.2.1 GMM . . . 61
5.2.2 Denoiser . . . 62
5.3 Results . . . 64
5.3.1 Baseline . . . 65
5.3.2 Ablation Studies . . . 65
5.3.3 Proposed Method . . . 71
5.3.4 Model Robustness . . . 71
5.4 Conclusive Remarks . . . 75

6 Conclusions and Outlook 76

Appendices 79
A AS Experiment Flowcharts . . . 79
A.1 AS Experiment I . . . 80
A.2 AS Experiment II . . . 80
A.3 AS Experiment III . . . 81
A.4 AS Experiment IV . . . 82
B Identification Reports . . . 83
B.1 AS Experiment I . . . 83
B.2 AS Experiment II . . . 84
B.3 AS Experiment III . . . 85
B.4 AS Experiment IV . . . 87

B.5 Proposed Method 88
B.6 MR Experiment I 89
B.7 MR Experiment II 90

List of Figures

1.1 Source attribution scenarios 3

2.1 Generic spectrogram 7
2.2 Hz-Mel relationship 8
2.3 Mel-Frequency Cepstral Coefficientss (MFCCs) extraction 10
2.4 Decision boundary for a linearly-separable problem . . . 15
2.5 Expectation-Maximization (EM) algorithm 21
2.6 Multi Layer Perceptron (MLP) architecture 22
2.7 Artificial neuron 23
2.8 Two-dimensional convolution operation 25
2.9 Convolutional Neural Network (CNN) architecture 26

3.1 Audio signal recording model 33
3.2 Channel responses example 35

4.1 Proposed method - simplified flowchart 37
4.2 Proposed method - flowchart 38
4.3 Phoneme spectrum basis matrix 44
4.4 Effect of the Denoising CNN (DnCNN) on the spectrogram 45
4.5 DnCNN architecture 46
4.6 Channel responses extracted using our modified algorithm 51
4.7 Feature components example 55

5.1 Effect of the pre-emphasis filter 62
5.2 AS Experiment I - confusion matrix 67
5.3 AS Experiment II - confusion matrix 68
5.4 AS Experiment III - confusion matrix 69
5.5 AS Experiment IV - confusion matrix 70
5.6 Proposed Method - confusion matrix 72

5.7 MR Experiment I - confusion matrix 73
5.8 MR Experiment II - confusion matrix 74

1 AS Experiment I - flowchart 80
2 AS Experiment II - flowchart 80
3 AS Experiment III - flowchart 81
4 AS Experiment IV - flowchart 82

List of Tables

5.1 MOBIPHONE mobile-phones list 58
5.2 Gaussian Mixture Model (GMM) design results 61
5.3 Baseline - accuracy results 65
5.4 Ablation studies settings 65

1 AS Experiment I - report 83
2 AS Experiment II - report 84
3 AS Experiment III - report - clean data 85
4 AS Experiment III - report - noisy data 86
5 AS Experiment IV - report 87
6 Proposed method - report 88
7 MR Experiment I - report 89
8 MR Experiment II - report 90

Glossary

AAC Advanced Audio Coding. 2

Adam Adaptive Moment Estimation. 59

AI Artificial Intelligence. 3

AIC Akaike Information Criterion. 22

AMR Adaptive Multi-Rate. 32, 58

ANN Artificial Neural Network. 12, 22, 24, 63

AWGN Additive White Gaussian Noise. 31, 32, 34, 44, 58, 62–64, 74

BED Band Energy Difference. 30, 63, 66

BFCC Bark Frequency Cepstral Coefficient. 29

BIC Bayesian Information Criterion. 22, 59

CNN Convolutional Neural Network. ix, 2, 24–26, 30–32, 46, 55, 64

CQT Constant-Q Transform. 31

DC Direct Current. 32

DCT Discrete Cosine Transform. 10, 40–42

DFT Discrete Fourier Transform. 6, 32, 41, 44

DL Deep Learning. 5, 11, 12, 14, 27, 60

DnCNN Denoising CNN. ix, 45, 46, 48, 58, 59, 63, 65, 67, 71–73, 76, 77

EM Expectation-Maximization. ix, 19, 21

FBP Filter Bank Power. 8, 41

FLAC Free Lossless Audio Codec. 57

GMM Gaussian Mixture Model. xi, 18, 19, 21, 29, 39, 40, 42, 47, 48, 55, 58–62, 65, 68, 69, 76, 77

GSV Gaussian Supervector. 29, 30

HHT Hilbert-Huang Transform. 5

HTK Hidden Markov Model Toolkit. 7, 40–42, 48, 76

IFT Inverse Fourier Transform. 8

IIR Infinite Impulse Response. 11

IMFCC Inverted MFCC. 30

LFCC Linear Frequency Cepstral Coefficient. 29

LPCC Linear Prediction Cepstral Coefficient. 29

LR Learning Rate. 59

LTAS Long-Term Average Spectrum. 66

MFCC Mel-Frequency Cepstral Coefficients. ix, 8, 10, 11, 29, 30, 40–42, 48, 76

MIR Music Information Retrieval. 8

ML Machine Learning. 3, 5, 11, 12, 14, 27, 59

MLP Multi Layer Perceptron. ix, 22, 24, 26

MSE Mean Square Error. 59

PCM Pulse-Code Modulation. 2, 57, 58

PSD Power Spectral Density. 30, 31

PSNR Peak SNR Ratio. 64

RASTA RelAtive Spectral TrAnsform. 10, 11, 39, 42, 48, 76

RASTA-MFCC RASTA filtered Mel-Frequency Cepstral Coefficients. 39, 40, 55

RBF Radial Basis Function. 17, 54, 60

RBF-NN Radial Basis Functions Neural Network. 29

SMO Sequential Minimal Optimization. 17, 30

SNR Signal-to-Noise Ratio. 32, 58, 60, 64, 72–74, 77

SSIM Structural Similarity Index. 64

STFT Short-Time Fourier Transform. 5–7, 33, 47

SVM Support Vector Machine. 14, 17, 29, 30, 54, 59, 60

TIMIT Texas Instruments/Massachusetts Institute of Technology. 57, 61, 62, 65

VGG Visual Geometry Group. 46

VQ Vector Quantization. 29

WAV Waveform Audio File. 57, 58

WT Wavelet Transform. 5

1 Introduction

Nowadays, the advancements in digital technologies led us to a massive spread of low-cost mobile devices. These are mostly used to acquire various forms of multimedia content such as videos, images, and audio recordings [1]. It happens more and more frequently that these digital materials are used as a piece of evidence in the court of law. Specifically, recorded audio through mobile devices is widely used due to the fact that its speech content may contain many pieces of evidence [2]. Just think at how many voice messages we exchange every day and how much sensitive information they might contain. Therefore, audio file authenticity procedures are required to assess the veracity of a multimedia object. Unfortunately, the vast majority of research has been carried out in the field of image forensics at the expense of audio forensics, which is comparatively less investigated [3].

Strictly concerning audio forensics, different methods have been proposed to solve various kinds of problems, ranging from speech and speaker recognition, acoustic environment identification, integrity verification, transcoding and codec identification, audio enhancement and double compression detection [4]. For instance, authors in [5] proposed a method for Advanced Audio Coding (AAC) encoding detection using a CNN, estimating also the bitrate from Pulse-Code Modulation (PCM) data. To assess integrity, the authors in [6] proposed an audio splicing detection method based on the estimation of inconsistencies in the reverberation time throughout a speech recording. The splicing tampering consists of replacing portions of audio with others from different recordings. They also addressed the problem of localization of the splicing point. Recent works in [7, 8] addressed the problem of AI-Synthesized speech signals detection by exploiting the weakness of the artificial generation of speech signals using neural networks. The audio-matching problem was addressed in [9] where the authors proposed a novel partial matching algorithm that balances time-granularity analysis and computational complexity with the amount of data to be analyzed.

One of the most important problems to be solved is that of *authenticity assessment*. This means verifying whether a recording is authentic or has been manipulated. Authenticity assessment using standard security approaches often means the adoption of digitally signing content right after signal acquisition or watermarking techniques, but this is not always applicable as it requires the modification of devices and workflows [4]. Pursuing this objective, most of the available passive[1] authentication methods relies on the extraction of a specific fingerprint directly from the audio recording. This fingerprint is often referable to the multiple tolerances of the hardware components adopted by the different manufacturers in the production process of the device, such as a mobile-phone.

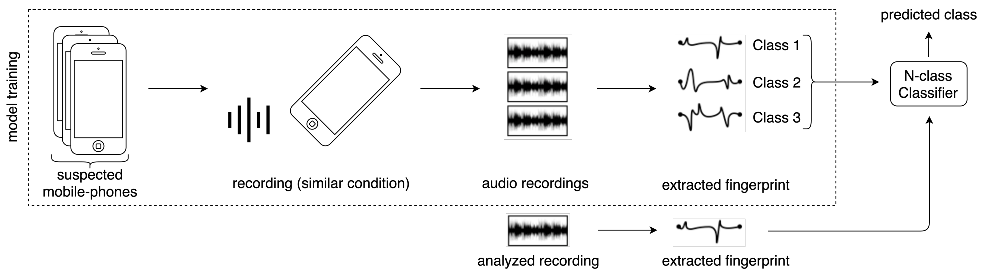
This extracted device fingerprint can be used in the context of *source attribution* to provide information about the acquisition device used to capture the audio signal under analysis. This task encompasses two known problems in forensics [10]:

- *source identification*: linking a file, i.e. audio recording, to one of several suspected acquisition devices (Fig. 1.1a);
- *source verification*: confirmation that an investigated file was acquired by a specific acquisition device (Fig. 1.1b).

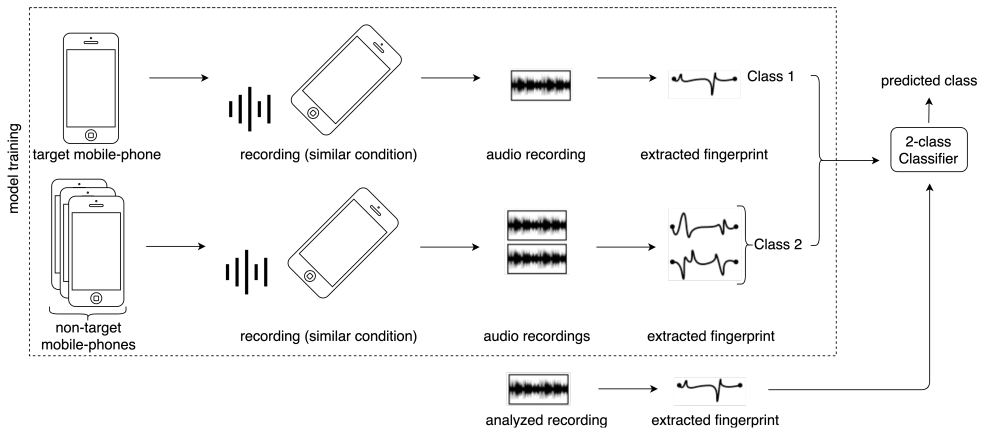
Our work focuses on *source identification*. More specifically, we propose a method to determine with which device the audio track under

[1]methods not relying on any extrinsic security mechanism such as digital sign or watermarking.

analysis was recorded. This scenario has two main constraints. The first is that the true device is supposed to be in a closed-set of known devices. The second one, we must have access to all the possible devices with the aim of producing some reference recordings, or at least, have the chance to access to a set of recordings whose source device is known.



(a) The device identification scenario.



(b) The device verification scenario.

Figure 1.1: The two main problems in source attribution.

In this work we develop a new and improved methodology to solve the source identification task in a *closed-set* scenario. Our work improves over a previous proposal by Cuccovillo et al. [11, 12], in which the authors proposed to perform microphone identification by blindly estimating the channel introduced by the microphone frequency response. In the original work, however, the estimation procedure was disrupted by the presence of any background noise, which greatly reduced its applicability in real case scenarios.

In this work, the blind estimation of the microphone frequency response integrates an explicit processing step for mitigating the influence of background noise. The algorithm is based on a composition of classic Machine Learning (ML), signal processing and Artificial Intelligence (AI) tools. It allows for better performances than the reference state-of-the-art techniques.

The rest of the work is organized as follows. Chapter 2 outlines the theoretical aspects necessary to fully understand the rest of the work. Chapter 3 gives an overview of the main state-of-the-art techniques addressing the device identification task. The chapter concludes by formally defining the problem that we solve in this work, relying upon a mathematical formulation. Chapter 4 describes the methodology that we devised to perform the device identification task. In particular, we will go into the details of the proposed architecture. Chapter 5 analyzes in details the results obtained in validating our method by means of the evaluation metrics. Chapter 6 summarizes possible system improvements and future developments to enhance performance. For the most interested readers, Appendix A sets out in detail the architectures used in each experiment while Appendix B contains the related identification statistics.

2 Theoretical Background

In this chapter we introduce the preliminary tools that will be used throughout this work. We make a short overview of the topics related to signal processing, to end up with a brief outline of some of the fundamental algorithms related to Machine Learning (ML) as well as Deep Learning (DL).

2.1 Signal Processing

In this section, we focus our attention on some signal processing tools which are an integral part of this work.

2.1.1 Time-Frequency Analysis

Time-frequency analysis refers to the techniques that study signals simultaneously, both in the time and in the frequency domain. These allow us to characterize signals that have a natural time-varying meaning such as a speech signal or music.

There are multiple time-frequency representations, such as the Wavelet Transform (WT), the Hilbert-Huang Transform (HHT) and the Short-

Time Fourier Transform (STFT). In the rest of the section we are going to focus our attention on the STFT and on its modifications.

2.1.1.1 Short-Time Fourier Transform

The basic idea used to build an STFT representation of a signal consists in partitioning it into shorter segments of equal length and performing the Discrete Fourier Transform (DFT) over them. The signal partitioning is performed through a sliding window function.

If we refer to $s(n)$ as a discrete signal, its STFT can be defined as [13]:

$$\mathrm{STFT}\{s(n)\}(k,l) = S(k,l) = \sum_{n=-\infty}^{+\infty} s(n)w(n-lR)e^{-j\frac{2\pi}{M}kn}, \tag{2.1}$$

where n is the current time instant, l is the time frame, k is the discrete frequency, denoted as "bin"; w is a window function composed by M samples, R is the hop size between two successive DFTs. At the time instant n, the sliding window mechanism allows taking information only on a portion of the signal around n. The time-frequency resolution trade-off can be adjusted by the window function length M. The longer the window, the higher the frequency resolution we have at the cost of a smaller resolution in time; vice versa for narrower windows. There exist many window functions in literature, each one with its own spectral characteristic[1].

The STFT of a signal is often visualized by means of a *spectrogram*, which is a two-dimensional representation of its magnitude [15].

When generating a spectrogram, the vertical axis often represents the frequency, the horizontal axis represents time, and the color of each pixel indicates the magnitude of a particular frequency in a specific time instant. In short, the procedure consists of creating a heat-map of the STFT magnitudes, as the one in Figure 2.1. This STFT magnitude is usually converted in a decibel scale, to better represent the sound intensity range that the human ear-system can cover. Usually, this mapping is done after squaring the values obtaining the so-called *log-power spectrogram*, i.e.:

$$P(k,l) = 10 \log |S(k,l)|^2. \tag{2.2}$$

[1]for a further explanation we refer the reader to [14].

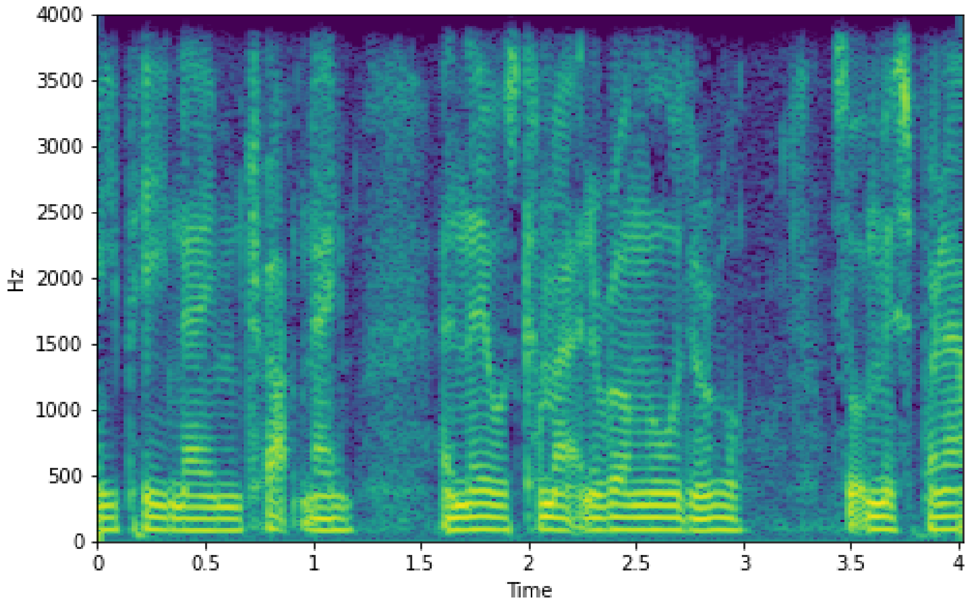


Figure 2.1: A generic spectrogram.

2.1.1.2 Mel Spectrogram

Different variants of STFT have been used in order to make this representation more informative depending on the needs. In the case of the *Mel spectrogram*, the frequency axis is mapped in a suitable way to emulate the human auditory system behavior. Indeed, our perception of pitch intervals is non-linearly correlated with linear intervals in frequency. For this reason, different scales that reflect this nonlinear relationship exists; one of the most used in the so-called *Mel scale* [16]. Due to their origin in listening experiments, several formulas exist that try to approximate the Hz-Mel frequency relationship. In this work we use the the one proposed by the Hidden Markov Model Toolkit (HTK) [17]:

$$m = 2595 \cdot \log\left(1 + \frac{f}{700}\right). \tag{2.3}$$

The inverse transform can be readily derived as:

$$f = 700\left(10^{\frac{m}{2595}} - 1\right). \tag{2.4}$$

In Figure 2.2, we can see how the relationship between Hz and Mel frequency is approximately linear for frequencies below 500Hz and highly non-linear on the opposite side. Indeed, by using this relationship, equally spaced points m_k, corresponds to frequency points f_k whose perceptual distance is equal [18]. The scales reference point is 1000, which means that 1000Hz equals 1000Mel.

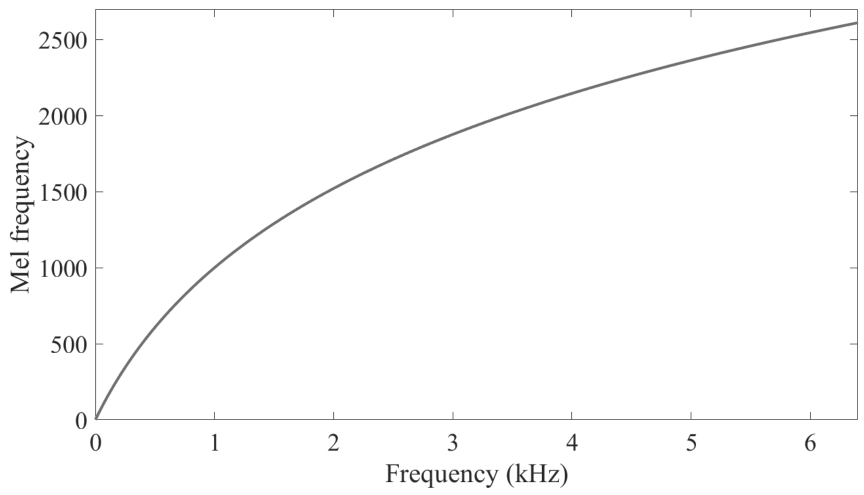


Figure 2.2: The Hz-Mel frequency relationship.

2.1.2 Mel-Frequency Cepstral Coefficients

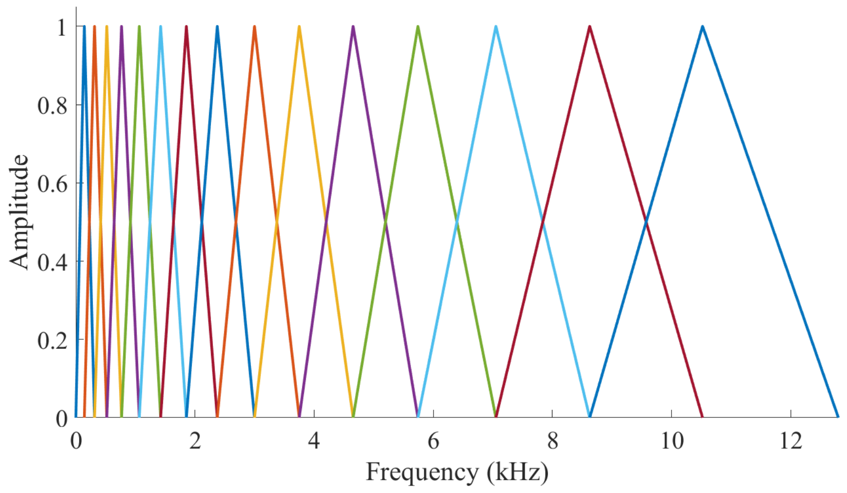
The Mel-Frequency Cepstral Coefficients (MFCC) are a well-known feature used for speech recognition and most recently adopted in the Music Information Retrieval (MIR) field. Their computation relies on the *cepstrum*, which is the outcome of computing the Inverse Fourier Transform (IFT) of the logarithm of a signal spectrum. This representation is a powerful tool for investigating periodic structures in the frequency domain and has a strong acceptance in human speech analysis.

At first, we construct the so-called *Mel-filter bank*. This is a set of triangular filters equally spaced on a Mel scale range. The construction procedure requires that the filters must overlap each other, such that the two extreme minima of each filter are located on the center frequency of the two adjacent ones, as in Figure 2.3a.

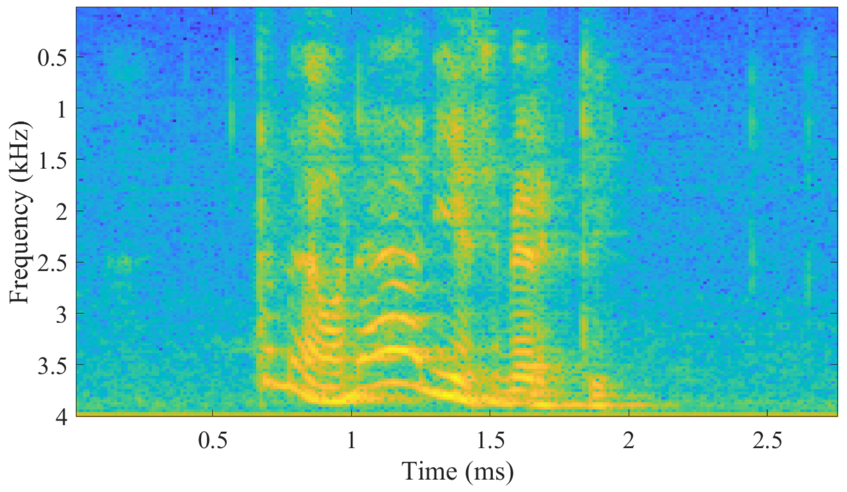
The extraction process of the MFCCs is illustrated in Figure 2.3. In specific, Figure 2.3b shows the log-power spectrogram of a speech segment[2]. When each window of that spectrogram is multiplied with the triangular filter bank, we obtain the Mel-weighted spectrum, illustrated in Figure 2.3c. This operation enables us to calculate the Filter Bank Power (FBP), which is a list of the power from each triangular filter.

Here we see that the main shape of the spectrogram is retained, but

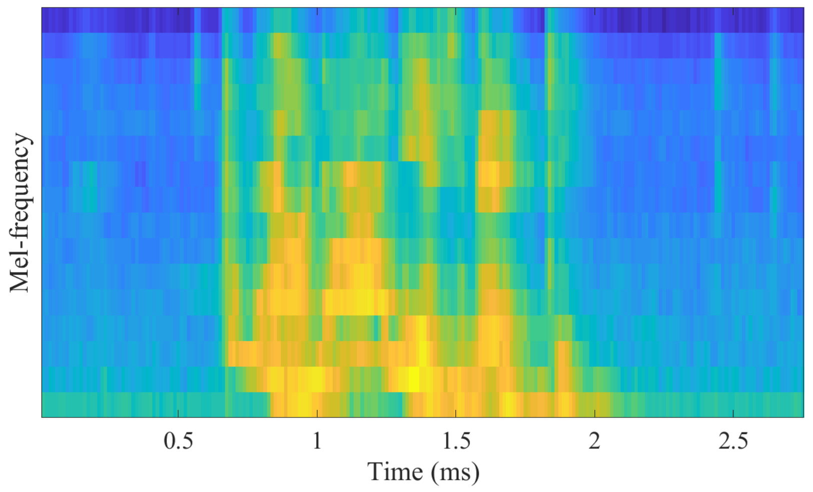
[2]different version of the MFCCs computation exists in literature [19]; here we report the one used in this work, in which we use the power spectrogram as input of the filter bank.



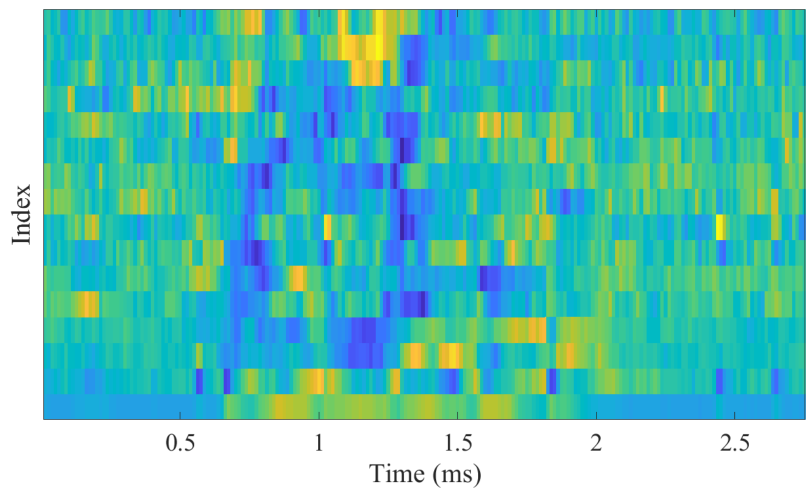
(a) The triangular Mel filter bank.



(b) A spectrogram of a speech segment.



(c) The spectrogram after multiplication with the Mel filter bank.



(d) The extracted MFCCs.

Figure 2.3: The main steps of the MFCCs extraction procedure; images courtesy of [18].

the fine-structure has been smoothed out. This procedure in practice removes the details related to the harmonic structure. Indeed, the identity of phonemes such as vowels is determined based on macro-shapes in the spectrum. The MFCCs preserve that type of information and remove "unrelated" information such as the pitch. Figure 2.3d illustrates the outcome after the Mel-weighted spectrogram is multiplied with a Discrete Cosine Transform (DCT) to obtain the final MFCCs. The purpose of the DCT is to decorrelate the signal since the triangular filter bank is highly overlapped, hence containing much frequency cross-information.

We can see that, differently from the Mel-weighted spectrogram, the MFCCs does not retain the original spectrum shape. This is justified by the fact that it is based on an abstract domain, in which we have information about the spectral envelope of the speech signal and not on the harmonic content, as in the spectrogram [18].

2.1.3 RelAtive Spectral TrAnsform Filter

Hermansky et al. in [20] proposed a filtering approach for a robust representations for speech recognition and enhancement. This procedure was meant to decouple the temporal properties of a generic channel effects from the temporal properties of a speech signal convolved with that channel. They called this filtering procedure RelAtive Spectral TrAnsform (RASTA). For some time, it was successfully used to model human voice for speaker (and speech) identification.

The main motivations that support the RelAtive Spectral TrAnsform (RASTA) filtering procedure lie in the speech signal characteristics and how a human perceives it. In particular, it has been proved that human listeners do not seem to pay much attention to a slow change in the frequency characteristics of a channel. In addition, steady background noise does not severely impair human speech communication. Thus, they band-pass filtering time trajectories of speech feature vectors using a filter with a sharp spectral zero at the zero frequency.

In principle, the RASTA processing can be done on time trajectories of any parameters, of course, with different effects depending on the domain of application. In our work, we are going to perform RASTA filtering of MFCCs vectors.

The RASTA filter is an Infinite Impulse Response (IIR) filter with transfer function in the z-domain as follow:

$$H(z) = 0.1z^4 * \frac{2 + z^{-1} - z^{-3} - 2z^{-4}}{1 - 0.94z^{-1}}. \tag{2.5}$$

Having a long time constant for integration, this filter has a memory effect. This characteristic allows an enhancement of the spectral transitions and makes also the result dependent on the previous short segment of speech, such as phoneme or syllable. In addition, this filtering accounts also for channel equalization and noisy speech enhancement, which are the reasons for its adoption in this work.

2.2 Machine Learning and Deep Learning Fundamentals

In this section, we give a brief introduction to the core ideas and concepts behind Machine Learning (ML) and Deep Learning (DL), explaining the algorithms used in this work.

2.2.1 Overview

Machine Learning (ML) is a field of study which allow computers ("machines") to learn from data or experience in order to make predictions that are based on this knowledge. These machines are not explicitly programmed, although they make data-driven decisions for carrying out a certain task. These programs or algorithms are designed in a way that they learn and improve over time when are exposed to new data. Deep Learning (DL) instead, is a subcategory of machine learning where the used algorithms attempt to find a mathematical representation of information processing that emulates the structure and function of the brain. For this reason, they are called Artificial Neural Network (ANN) [21].

2.2.2 Taxonomy

ML problems can be broadly divided in three classical categories:

- *Supervised Learning*;
- *Unsupervised Learning*;
- *Reinforcement Learning.*

In addition to these three main categories, there are two main subcategories of supervised learning: *Semi-Supervised Learning*, which can be considered a mix between supervised and unsupervised learning, and *Self-Supervised Learning* that can be regarded as autonomous learning in that the model does not necessarily require sample data classified in advance by humans.

2.2.2.1 Supervised Learning

Supervised learning is focused on predicting a target value given input observations. In machine learning, we call the model inputs *features.* The target values instead are often called *labels.* The latter are the elements on which the supervised models are trained to make a prediction. Supervised learning problems can be categorized into two major subcategories [22]: *regression* analysis and *classification.*

In regression analysis, the labels are continuous variables. In classification, the labels are so-called *class labels*, which can be understood as discrete class or group-membership indicators [23].

2.2.2.2 Unsupervised Learning

While supervised learning is based on labeled data, unsupervised learning aims to model the hidden structure in data without label information. The main tasks addressed in unsupervised learning are:

- *Feature Learning*;
- *Dimensionality Reduction*;
- *Clustering.*

Feature learning techniques replace manual feature engineering and allow a machine to automatically discover the representations needed to perform a specific task.

Instead, *dimensionality reduction* aims at achieving a data transformation from a high-dimensional space into a low-dimensional one so that the low-dimensional representation still preserve some meaningful properties of the original data. This is crucial since working in high-dimensional spaces can be undesirable: raw data are often sparse as a consequence of the curse of dimensionality, and analyzing the data in this way is usually computationally intractable [22].

Lastly, *clustering* methods can be seen as a task similar to classification but without labeling information in the training dataset. Without this information, the main goal is to group data by similarity and define distinct groups based on similarity thresholds. These algorithms can be divided into three major groups [23]: prototype-based, density-based, and hierarchical clustering. While in the first type, the amount of clusters is defined a-priori, in a density-based clustering this number is not fixed but assigned by identifying regions of a high density of data. In hierarchical clustering, however, a distance metric is used to group examples in a tree-like fashion, in such a way that examples at the root are more related to each other. The depth of the tree defines the number of clusters to be used.

2.2.2.3 Reinforcement Learning

Reinforcement learning is concerned with developing reward systems to model complex decision processes and learning a series of actions that lead to a particular outcome. In a reinforcement learning algorithm, a so-called agent learns how to act by interacting with its environment. The agent receives rewards for performing correctly and penalties for performing incorrectly. The agent learns without human intervention by maximizing its reward and minimizing its penalty.

Since its discussion is beyond the scope of this work, a more comprehensive treatment of the subject is left to the reader in [24] and [25].

2.2.3 Architectures Used in this Work

In this section, we will see how some algorithms can help us in solving general tasks that interest this work. In particular, we will start with two more general ML algorithms and then move on to addressing two of the basic building blocks which forms some of the DL architecture used in this field.

2.2.3.1 Support Vector Machine

Support Vector Machine (SVM) is a supervised ML algorithm mostly used for linear and non-linear classification problems.

Its working principle is based on the creation of a hyperplane in a multidimensional feature space that separates data points into two or more classes. The position and orientation of this hyperplane depend on the features position in such a way that the distance between the nearest points belonging to a different classes is maximized. These points are called *support vectors* and decide the exact placement of the separating hyperplane, maximizing the region[3] between the possible classes. The resulting separation hyperplane will be located in the middle point from the support vectors. In Figure 2.4 we give a graphical example for a two-class classification problem with a two-dimensional feature space. The margin is represented by the dashed lines whereas the final decision boundary is equispaced between them and represented by a single solid line.

The learning step is therefore the process of finding the hyperplane with the maximum margin (optimal solution), thus maximizing the distance of the closest point to the hyperplane.

In a simplified two-class classification scenario, using a linear model in order to separate these data points, we can define an hyperplane as

[3]called functional *margin* in this context.

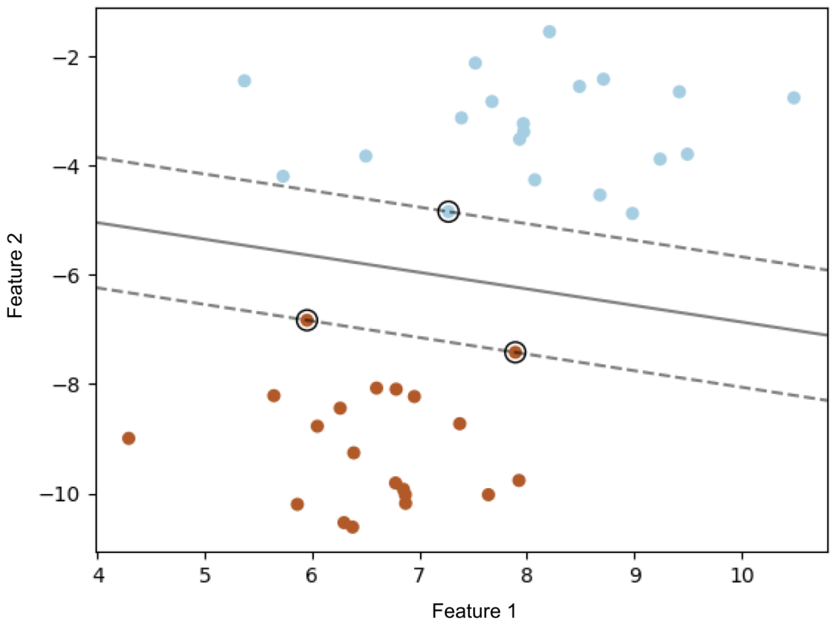


Figure 2.4: The decision boundary for a linearly-separable problem, with three support vectors (rounded dots) on the margin boundaries; image courtesy of [26].

follow [22]:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b, \tag{2.6}$$

where $\mathbf{x}$ and $\mathbf{w}$ are N-dimensional vectors, $\boldsymbol{\phi}$ denotes a fixed feature space transformation and b a bias parameter. The training data set is formed by N input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with the corresponding target values $t_1, \ldots, t_N$, where $t_n \in \{-1, 1\}$. Assuming that the training dataset is linearly separable in the feature space, there exist at least one choice of parameters $\mathbf{w}$ and b such that $y(\mathbf{x}_n) > 0$ for points having $t_n = +1$ and $y(\mathbf{x}_n) < 0$ for points having $t_n = -1$. For the calculation of the optimal solution, we are only interested in data points that are correctly classified, so the ones that satisfy the relation $t_n y(\mathbf{x}_n) > 0$ for all n. Knowing that the distance of a point $\mathbf{x}_n$ to the decision surface in the Equation (2.6), is given by $\frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$, we can narrow down the calculation to the only correct points, such that:

$$\frac{t_n y\left(\mathbf{x}_n\right)}{\|\mathbf{w}\|} = \frac{t_n\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}\left(\mathbf{x}_n\right) + b\right)}{\|\mathbf{w}\|}, \tag{2.7}$$

where $||\cdot||$ represents the norm of a vector. The margin is given by the perpendicular distance to the closest point $\mathbf{x}_n$ from the data set (the support vector), and our final goal is to optimize the parameters $\mathbf{w}$ and b in order to maximize this distance. Thus the maximum margin solution is found by solving:

$$\arg\max_{\mathbf{w},b}\left\{\frac{1}{\|\mathbf{w}\|}\min_{n}\left[t_n\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}\left(\mathbf{x}_n\right)+b\right)\right]\right\}, \tag{2.8}$$

where the factor $\frac{1}{\|\mathbf{w}\|}$ is outside the optimization over n because $\mathbf{w}$ does not depend on n. This optimization problem leads us to a very complex solution. We can overcome this by turning this problem into a simpler one. We can make a rescaling $\mathbf{w} \to k\mathbf{w}$ and $b \to kb$ in such a way that the distance from any point $\mathbf{x}_n$ to the decision surface is unchanged [25]. This allows us to formulate the problem as follows:

$$t_n\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}\left(\mathbf{x}_n\right)+b\right)=1, \tag{2.9}$$

for the points that are closest to the surface, so for the support vectors. All the other data points have to satisfy the constraints:

$$t_n\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}\left(\mathbf{x}_n\right)+b\right)\geqslant 1. \tag{2.10}$$

This is known as the canonical representation of the decision hyperplane. In this formulation, the optimization problem simply requires to maximize $\|\mathbf{w}\|^{-1}$, which is equivalent to minimize $\|\mathbf{w}\|^2$, i.e.:

$$\arg\min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2, \tag{2.11}$$

subject to the constraints given in Equation (2.10). This is an example of a quadratic programming problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints. Its solution is obtained through the use of the Lagrange multipliers $a_n \geqslant 0$, with one multiplier a_n for each of the constraints in Equation (2.10), leading to the Lagrangian function:

$$L(\mathbf{w},b,\mathbf{a})=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{n=1}^{N}a_n\left\{t_n\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}\left(\mathbf{x}_n\right)+b\right)-1\right\}, \tag{2.12}$$

with $\mathbf{a}=(a_n,\ldots,a_N)^{\mathrm{T}}$. The minus sign in front of the Lagrange multiplier term is reasonable because we are minimizing with respect to $\mathbf{w}$ and b, and maximizing with respect to $\mathbf{a}$. For an extensive discussion on this subject, the reader can refer to [22] and [25].

One of the most popular approaches to train a support vector machines is called Sequential Minimal Optimization (SMO), and it is based on the concept of chunking[4], considering only two Lagrange multipliers at a time. In this simplified case, the sub-problem can be solved analytically, avoiding the use of numerical quadratic programming. Heuristics are given for choosing the pair of Lagrange multipliers to be considered at each step. SMO is found to have a complexity scaling with the number of data points that is between linear and quadratic depending on the particular application [22].

In most cases, a well-known problem is that the resulting feature space is non-linearly separable such that the hyperplane that divides the class cannot be described in a linear form as stated in Equation (2.6). To solve this problem, a data transformation is carried out using the so-called *kernel*, that mapping the data points in a higher dimension in which they become linearly separable. This procedure is known in the literature as the *kernel trick* or kernel substitution. The concept is based on a kernel formulated as an inner product in a feature space. This allows an algorithm formulated in the same way to replace that scalar product with some other choice of kernel. There exist different kernels, each one with specific characteristics. The most popular are the Polynomial kernel, the Gaussian kernel, the Radial Basis Function (RBF), the Hyperbolic Tangent kernel, the Sigmoid kernel and the ANOVA RBF kernel. Among them, the most commonly used is the RBF.

Whenever performing the tuning of a Support Vector Machine (SVM), it is possible to change not only the kernel but also some other hyperparameters with the final goal of improving the total accuracy of the system. The *regularization coefficient*, usually denoted with C, tells the SVM optimization how much miss-classification rate we can accept. The lower this parameter, the bigger the hyperplane's margin will be, so the number of the miss classified point will be higher, giving more flexibility. Another important hyperparameter is *gamma*, usually denoted with γ and relevant for the RBF kernels. It controls the influences that the points have on the hyperplane position. A low gamma implies that for finding the optimal hyperplane we have to consider points that are dis-

[4]the technique of *chunking* exploits the fact that the value of the Lagrangian function remains the same if we remove the rows and columns of the kernel matrix corresponding to Lagrange multipliers that have zero value.

tant from the plausible hyperplane, and vice versa.

2.2.3.2 Gaussian Mixture Model

The Gaussian Mixture Model (GMM) is an unsupervised method for solving clustering problems. We often refer to this problem as a soft-clustering [22] because, differently for the classical K-means algorithm[5], for each point in the cluster we can associate an uncertainty measure, i.e. a probability. In addition, such information allows us to use a probability distribution-based model instead of a classical distance-based model for solving more generic clustering problems.

GMM assumes that there are a certain number K of Gaussian distributions, usually called *components* or *mixtures*, that represent the clusters, with the goal of grouping together data points belonging to the same distribution. A GMM is parameterized by two types of values, the mixture weights and the mixture means and covariances. Generalizing for a K-dimensional input $\mathbf{x}$, the Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right), \tag{2.13}$$

where π_k represent the k-th component weight coefficient[6] and are subject to the constrains $0 \leqslant \pi_k \leqslant 1$ along with $\sum_{i=k}^{K} \pi_k = 1$, so that the total probability distribution normalizes to 1. $\mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$ instead, represents a single mixture, and is a multivariate Gaussian distribution with its own mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$. This single multivariate Gaussian distribution represents a single cluster and can be generally formulated as:

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{K/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}, \tag{2.14}$$

where $\boldsymbol{\mu}$ is a K-dimensional mean vector, $\boldsymbol{\Sigma}$ a K $\times$ K covariance matrix and $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$ [22].

Let us introduce a K-dimensional binary random variable $\mathbf{z} \in \{0,1\}$ with $\sum_{k=1}^{K} z_k = 1$. A single element z_k of $\mathbf{z}$ is equal to 1 and all the other

[5]a hard-clustering algorithm because each point is associated to one and only one cluster [27].

[6]also called *mixing coefficient* or *prior*.

are 0, thus there are K possible states for the vector $\mathbf{z}$ according to which element is nonzero. This vector can be considered as a selector index for a specific mixture. We can define a marginal distribution $p(\mathbf{z})$ and a conditional distribution $p(\mathbf{x} \mid \mathbf{z})$ noting that the marginal distribution over $\mathbf{z}$ is specified in terms of the mixing coefficients π_k, such that:

$$p(z_k = 1) = \pi_k. \tag{2.15}$$

The conditional distribution of $\mathbf{x}$ given a particular value for $\mathbf{z}$ instead, is a Gaussian of the form:

$$p(\mathbf{x} \mid z_k = 1) = \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right). \tag{2.16}$$

From Equation (2.15) and Equation (2.16) we can obtain the conditional probability of $\mathbf{z}$ given $\mathbf{x}$ using the Bayes' theorem, i.e.:

$$\begin{aligned} p\left(z_k = 1 \mid \mathbf{x}\right) &= \frac{p\left(z_k = 1\right) p\left(\mathbf{x} \mid z_k = 1\right)}{\sum_{j=1}^{K} p\left(z_j = 1\right) p\left(\mathbf{x} \mid z_j = 1\right)} \\ &= \frac{\pi_k \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)}. \end{aligned} \tag{2.17}$$

This quantity indicates the *posterior* probability once we have observed the $\mathbf{x}$ data. Basically, π_k is the *prior* probability of $z_k = 1$ and $p\left(z_k = 1 \mid \mathbf{x}\right)$ can be viewed as the *responsibility* that the mixture k gives for "explaining" the observation in $\mathbf{x}$.

Training a GMM for a clustering problem means finding the right mean vector $\boldsymbol{\mu}_k$, the covariance matrix $\boldsymbol{\Sigma}_k$ and the mixing coefficient π_k for each of our Gaussian mixtures. This problem is tackled using an iterative statistical algorithm called EM [28].

E-M Algorithm The EM algorithm for Gaussian mixtures has the goal to maximize the likelihood function with respect to the parameters[7]. Suppose we have a dataset of observations $\mathbf{x}_1, \ldots, \mathbf{x}_N$. This iterative procedure for Gaussian mixtures can be formulated in this way [22]:

1. Initialize the parameters and evaluate the initial value of the log likelihood.

[7]speaking about GMM, these parameters are its mixing coefficient π_k, the mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$ that characterize a single multivariate Gaussian distribution.

2. **Expectation-step**: evaluate the posterior probabilities (Equation (2.17)) using the current parameter values:

$$\gamma\left(z_{nk}\right)=\frac{\pi_k \mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)}{\sum_{j=1}^K \pi_j \mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)}.$$

3. **Maximization-step**: re-estimate the parameters using the current posterior probabilities:

$$\boldsymbol{\mu}_k^{\text{new}}=\frac{1}{N_k} \sum_{n=1}^N \gamma\left(z_{nk}\right) \mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k^{\text{new}}=\frac{1}{N_k} \sum_{n=1}^N \gamma\left(z_{nk}\right)\left(\mathbf{x}_n-\boldsymbol{\mu}_k^{\text{new}}\right)\left(\mathbf{x}_n-\boldsymbol{\mu}_k^{\text{new}}\right)^{\mathrm{T}}$$

$$\pi_k^{\text{new}}=\frac{N_k}{N},$$

where

$$N_k=\sum_{n=1}^N \gamma\left(z_{nk}\right).$$

4. Evaluate the log likelihood:

$$\ln p(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi})=\sum_{n=1}^N \ln \left\{\sum_{k=1}^K \pi_k \mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right\},$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

A graphical example of the algorithm is proposed in Figure 2.5, where a mixture of two Gaussians was used. Figure 2.5a shows the data points in green, together with the initial configuration of the mixture model in which the one standard-deviation contours for the two Gaussian components are shown as blue and red circles. Figure 2.5b shows the result of the initial Expectation-step, in which each data point is depicted using a proportion of blue color equal to the posterior probability of having been generated from the blue component, and a corresponding proportion of red color given by the posterior probability of having been generated by the red component. To obtain these posterior probabilities the Equation (2.17) was used. In addition, we can see that points that have a

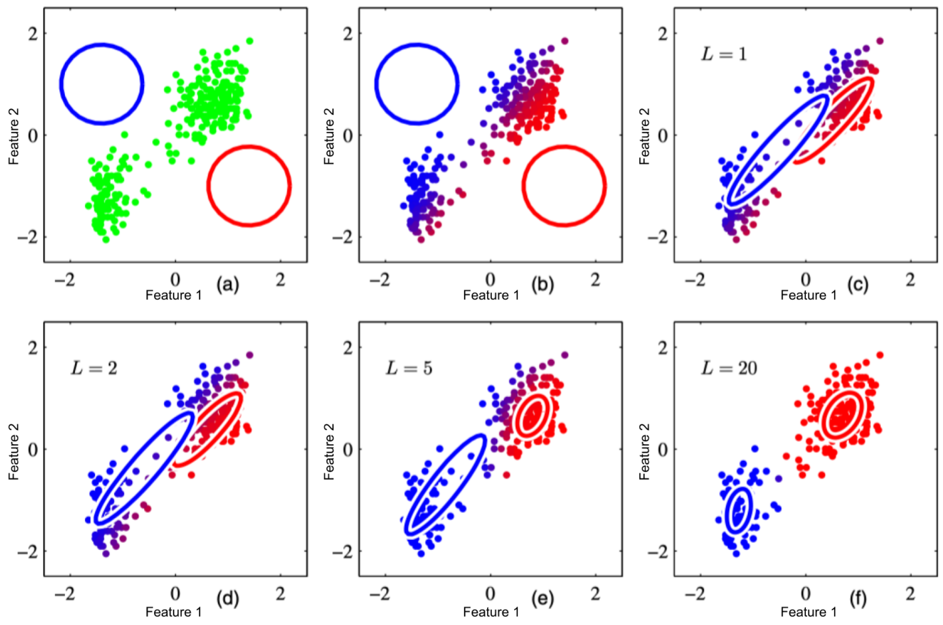


Figure 2.5: An example of the EM algorithm for a mixture of two components; image courtesy of [22].

significant probability of belonging to either cluster appear purple. The situation after the first Maximization-step is shown Figure 2.5c, in which the mean of the blue Gaussian has moved to the mean of the data set, weighted by the probabilities of each data point belonging to the blue cluster. We can see how it has moved to the center of mass of the blue data. Similarly, the covariance of the blue Gaussian is set equal to the covariance of the blue data. Analogous results hold for the red points. Figure 2.5d, 2.5e, and 2.5f show the results after 2, 5, and 20 complete cycles of EM, respectively. In plot Figure 2.5f the algorithm is close to convergence.

It is important to notice that the EM algorithm takes many more iterations to reach (approximate) convergence compared to the K-means algorithm and that each cycle requires significantly more computation. It is therefore common to run the K-means algorithm in order to find a suitable initialization for a GMM that is subsequently adapted using the EM.

The number of mixtures K is not known a priori. Indeed, it is typical to guess the number of components and fit that model to the data using the EM algorithm. A much more rigorous approach to solve this problem is based on the adoption of some information-theoretic criteria as the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC). Based on these scores we can select the right number of mixtures other than the covariance matrix type that will be used.

A more theoretical formulation of this problem is addressed in [22] and in [25].

2.2.3.3 Multi Layer Perceptron

The Multi Layer Perceptron (MLP) is a specific class of feed-forward ANNs used for different kinds of problems. Despite the name, the model comprises multiple layers of logistic regression models [22] (with continuous non-linearities) rather than multiple perceptrons (with discontinuous non-linearities).

Any MLP is composed of layers containing multiple nodes, called *neurons*. A common structure involves an ordered stack of these layers, for instance, an input layer, one or more hidden layers and a final output layer. The choice of the topology, as well as the type of input data, determines the effectiveness of the network. For example, the *fully-connected* MLP structure provides that the neurons of a layer are connected to all the neurons of both the previous and next layer, as we can see in the Figure 2.6.

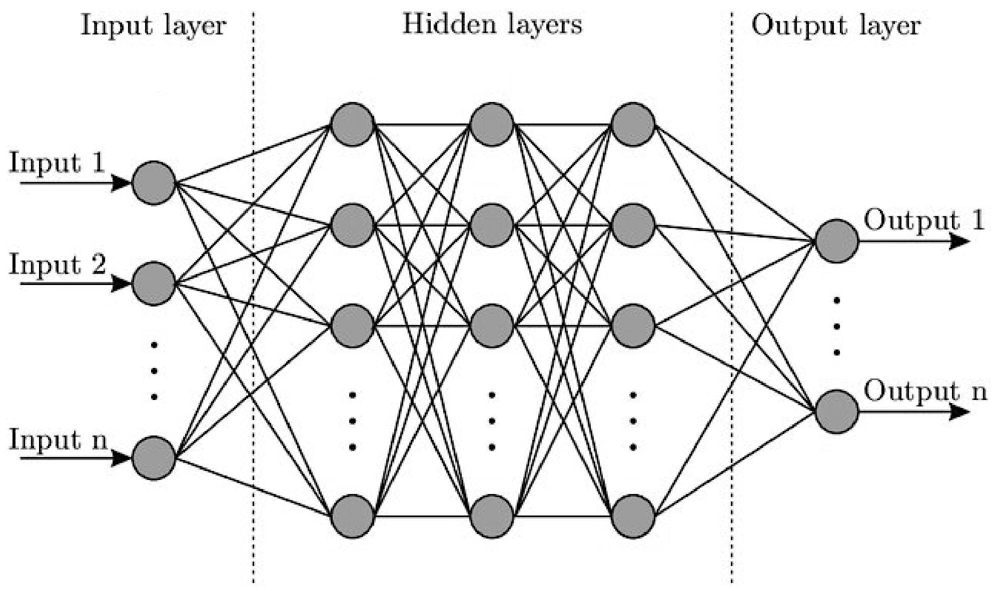


Figure 2.6: A fully-connected MLP architecture.

Let us consider $\mathbf{x}$ our input vector with N elements $x_1, \ldots, x_N$, i the input signal index and j the neuron index in the layer. The *activation* *procedure* for a specific neuron j can be seen as a composition of three components:

- *Activation value*: the input of a neuron is composed of a sum of all the elements of $\mathbf{x}$ weighted by a weight vector $\mathbf{w}$, i.e.:

$$z_j = f(\mathbf{x}, \mathbf{w}, b) = \sum_{i=1}^{N} w_{ij} x_i; \tag{2.18}$$

- *Bias*: $b_j = -w_{0j} * 1$, utilized to offset the result;
- *Activation function*: $g_j(\cdot)$, a non linear function.

Therefore, the final output for a single neuron is obtained as:

$$y_j = g_j(z_j) = g_j\left(\sum_{i=1}^{N} w_{ij} x_i - b_j\right). \tag{2.19}$$

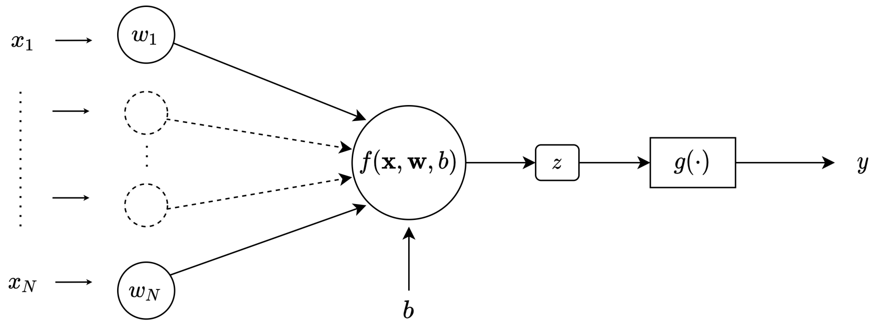


Figure 2.7: The basic elements of an artificial neuron.

In the beginning, the weights and biases are “randomly” assigned. The weights help determine the importance of any given variable, in such a way that the larger ones contributing more significantly to the output compared to the other inputs. As we see from Figure 2.7, all inputs are multiplied by their respective weights and summed. The final non-linear output function gives the y_j. The fact that the output function is non-linear is important because if it were not, any composition of neurons would still represent a linear function. The non-linearity is what allows sufficiently large networks of neurons to represent arbitrary functions [21].

The output function can be different from network to network. Several function exists, giving different benefit. The most common are listed below [21]:

- *Sigmoid*, $\sigma(x) = \frac{1}{(1+e^{-x})}$;
- *ReLU*, $\mathrm{ReLU}(x) = \max(0, x)$;
- *Softplus*, $\mathrm{softplus}(x) = \log(1 + e^{x})$;
- *tanh*, $\tanh(x) = \frac{e^{2x}-1}{e^{x}+1}$.

The training process has the purpose of getting the right weights and biases for the model, minimizing a specific *loss function.* To do that, a maximum likelihood framework is used, involving the solution of a non-linear optimization problem. This requires the evaluation of derivatives of the log likelihood function with respect to the network parameters that can be obtained using the technique of *error back-propagation* [22]. The term *back-propagation* is explained by the fact that the weights and biases update procedure starts from the final layer, returning back up to the initial one.

The output function for the output layer is different from the one used in the input and hidden layer; usually a *softmax* function is used, where the resulting value can be interpreted as a probability. This function is a generalization of the logistic function to multiple dimensions, i.e.:

$$\sigma(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}. \tag{2.20}$$

It is important to note that according to the task we need to solve, the output layer can assume different configuration. In specific:

- for *Classification problems*: the number of neurons are of the same number of the final number of classes. The output values must be considered as the probabilities that the analyzed data belongs to the class;
- for *Regression problems*: the number of neurons are of the same number of the domain of the regression function. By convention, the output values are considered as the regression values.

2.2.3.4 Convolutional Neural Network

The Convolutional Neural Network (CNN) is a specific feed-forward ANN inspired by the behavior of the visual cortex. As in the MLP case, a CNN is composed of an input layer, a variable number of hidden layers and an output layer. These hidden layers perform the convolution operation between the input signal, i.e. image, with a series of filters that contain the weights and biases that are learned during the training phase. These filters are shared on the entire layer reducing the complexity of the network keeping the number of parameters low [29]. The filter size determines the respective *receptive field.* The result of each convolution is called *feature map* and represents a specific feature extracted at all locations of the input [30].

We can think of the convolution operation as an approximation of the visual cortex behavior, in the way that a finite number of neurons respond to certain portions of the visual field. For a generic two-dimensional signal, i.e. image, this operation is defined as the scalar product of a weight matrix, the kernel, with every neighborhood in the input as shown in Figure 2.8, i.e.:

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t), \tag{2.21}$$

with $f(x,y)$ the original signal, ω is the kernel, $g(x,y)$ is the output signal obtained from convolution, i.e. the feature map, and $*$ is the convolution operation done on a two-dimensional rectangular block.

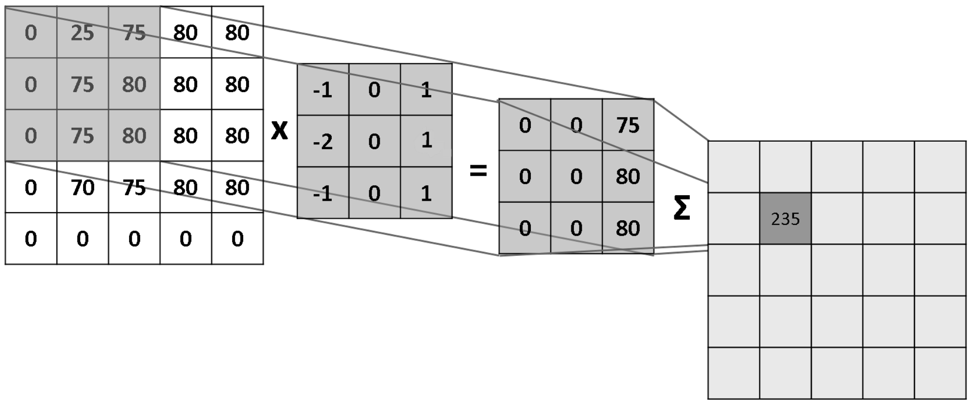


Figure 2.8: An example of a two-dimensional convolution operation.

In Figure 2.9 we can see a generic CNN structure. After the first convolution, the resulting feature map $g(x,y)$ firstly passes through one of the non-linear activation functions previously seen in Section 2.2.3.3. Then, the so-called *pooling layer* is used to decrease the dimension of the feature map reducing the complexity, replacing the output at a certain location with a summary statistic of the nearby outputs. There exist many types of pooling layers, depending on what kind of statistics is used. The most popular ones are the max pooling and the average pooling that take the maximum value and the average value respectively around a certain area. After that, a stack of these layers is generally used to learn the hidden structure of the signal. Indeed, we can think of the CNN as a process that permits a gradually increase in the ability to detect complex patterns as the hidden structure grows.

Often, the increase in hidden structure can lead to a problem known as *overfitting*. This happens when the model tends to find the best fit to the specific data used in training instead of learning their generic statistics [31]. This problem may be alleviated using *dropout* layers. The key idea is to temporarily remove neurons from the network during training, along with all its incoming and outgoing connections, with the aim of improving the performance on supervised learning tasks [32].

After the feature learning phase, the signal is usually flattened in a one-dimensional signal. The resulting signal is then passed to an output layer that is often made by a fully-connected MLP, with the goal of learning the right final mapping (an image class in this case) starting from data that have a high level of abstraction at that point.

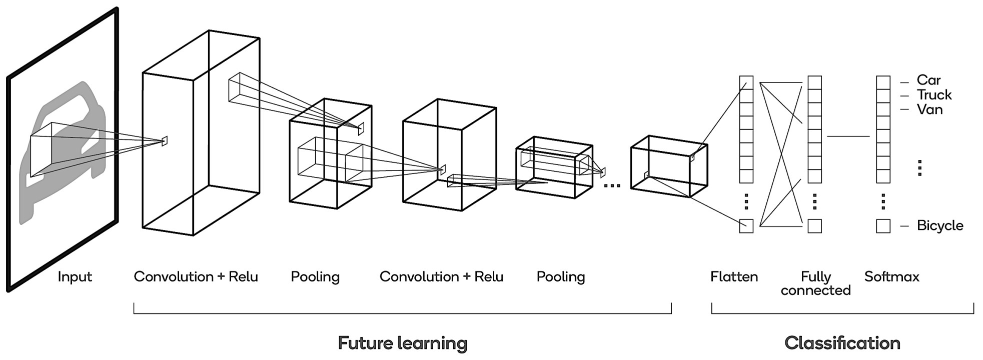


Figure 2.9: A generic CNN architecture.

Also in this case, the training procedure aims at finding the right weights and biases, minimizing a loss function using the back-propagation procedure.

2.3 Conclusive Remarks

In this chapter, we introduced some of the signal processing tools that are used throughout this work. We also exposed the main motivation behind ML and described its three broad categories which are supervised learning, unsupervised learning and reinforcement learning. Lastly, in order to give a better comprehension of our method, we talked about some of the ML and DL algorithms adopted.

3 State of the Art and Problem Statement

In this chapter we introduce the reader to the current challenges concerning the Audio Forensic field. In particular, we cover the problem of device identification from a speech recording. We report a general overview of the current state-of-the-art in this field, briefly describing some of the most remarkable works. Finally, we end up by reporting the formal definition of the problem addressed in this work.

3.1 State of the Art on Source Attribution

In a real-world scenario, different devices (i.e., telephone handsets, mobile-phones, desktop microphones) hardly share exactly the same frequency response due to the tolerance in the nominal values of the electronic components and the different designs criteria used by the various manufacturers [33]. Thanks to this, we can characterize a specific device by exploiting its own intrinsic traces and use these as signature, or fingerprint. Therefore, the way we extract this signature from the audio file is crucial to the recognition of the audio recording source.

Considering that the vast majority of the audio acquisition systems use mobile-phones as an essential tool for recording and given also the fact that our work focuses on this particular scenario, here we proposed a review of methods explicitly focused on mobile-phone identification from a recorded audio signal.

3.1.1 Device Identification

The most common methods for device identification involve the extraction of the features directly using the speech audio file and computing the MFCCs. This methodology was adopted by Hanilçi et al. in [33]. In this case, the dataset used for the recognition task comprises 14 different mobile-phone devices and the SVM classifier achieve 96.42% of accuracy. They also explored the use of Vector Quantization (VQ) for the mobile-phone identification task.

Hanilçi et al. in [34] explored the performances of different acoustic features for the same task. The final results led to the conclusion that in most of the cases, MFCCs perform better compared to other cepstral based features such as Linear Frequency Cepstral Coefficients (LFCCs), Bark Frequency Cepstral Coefficients (BFCCs), and Linear Prediction Cepstral Coefficients (LPCCs). However, LPCCs perform better identification if mean and variance normalization is adopted. Benefits in identification results were obtained by adding the corresponding delta features[1] to the original cepstral features.

Kotropoulos et al. in [36], built a device fingerprint by first extracting the MFCCs feature vector at the frame level. Then, in order to model the probability density function of these vectors, they trained a GMM with diagonal covariance matrices. Finally, having trained a single GMM for each device, a Gaussian Supervector (GSV) is built by concatenating the mean vectors and the main diagonals of the covariance matrices of all components, in order to construct a specific template for each device. Maximum identification accuracy of 97.6% has been achieved on 21 mobile-phones from seven different brands using a Radial Basis Functions Neural Network (RBF-NN) classifier. Jiang et al. in [37] proposed a kernel-based projection method that maps the raw GSV onto another di-

[1]an estimate of the first and second order derivatives [35].

mensional space where the useful microphone response information and the useless speech signal can be separated more easily. The projected GSVs outperform the raw GSVs in the microphone identification task. GSVs were also used for mobile-phone device verification in [38].

Verma et al. in [39] proposed to characterize a mobile device by concatenating MFCC and Inverted MFCC (IMFCC) feature vectors to characterize a mobile-device. The reason in using the IMFCC lies in its intrinsic ability to extract features from the high frequency region of the audio, as opposed to the traditional MFCC. This work addresses also double compression detection problem.

Da Luo et al. in [10] proposed the adoption of a new audio descriptor for attribution task of digital speech recordings. The proposed descriptor is called Band Energy Difference (BED) and relies on deviations that some frequency bands present when compared to adjacent bands. This behavior has been proven to be consistent across multiple devices and various recording conditions, including different speakers and recording environments. Similarity between frequency bands was also investigated by Lin et al. in [40]. The authors tried to understand which frequency ranges are most significant in device identification. To do this, they extracted the energy time variation from different sub-bands obtained by dividing the spectrogram's frequency axis of an audio recording in multiple parts. These energy variations are then analyzed by a CNN architecture with multiple inputs, one for each sub-band. Each input implements an attention mechanism [41] to detect the most relevant information carried by a single sub-band and dependent on the device under analysis. The specific attention mechanism adopted [42] was also found to be very useful in the analysis of noisy signals, leading to an increased robustness of the system in the device identification problem.

Speech-free segments of the audio were used in [43] and in [44], where the authors used them to estimate the Power Spectral Density (PSD) in order to characterize the mobile-phone device. Aggarwal et al. in [45] used the noise estimate to extract large numbers of MFCC feature vectors that were clustered using the K-means algorithm. Then the feature vectors corresponding to the centroid of these clusters were classified using SMO based SVM classifier.

Noisy data was also used by Shen et al. in [2]. In this recent work, the authors did not adopt MFCCs instead, they designed a novel neural network to extract suitable features directly from the device noise. They also built a new recording source dataset which contains some of the latest mobile phones and tablet devices. As far as we know, this is the most recent and updated dataset available in the literature that contains recordings from different mobile-phones as well as from different speakers.

Mobile-phone clustering was addressed by Li et al. in [46]. They proposed a method that aims to merge the recordings acquired by the same device into a single cluster without having prior information about the recordings. Deep auto-encoder networks were used to extract intrinsic signatures of a device.

Baldini et al. in [47] made an evaluation of the influence of different entropy measures in the microphone identification task. The entropy measures were extracted from the spectrogram of the audio recordings of the mobile-phones stimulated with three different types of sounds. In this work, they also extensively analyzed the features selection process with the aim of identifying the most discriminating entropy measures related to the type of data to be analyzed.

3.1.2 Device Identification in Noisy Conditions

The papers which we presented up to this point all focused their attention on obtaining rich features which could convey the influence of the microphone frequency response on the recordings, to obtain some sort of device fingerprint. The extraction of these device fingerprints in presence of noise, however, has been greatly underestimated in the literature, even if the presence of unwanted noise in the audio file inevitably causes the masking of signatures that are present in different parts of the spectrum for a specific device. In particular, the Additive White Gaussian Noise (AWGN), having uniform PSD can mask the entire spectrum, making no distinction in contribution between frequencies. To our knowledge, device identification in noisy conditions was tackled only by the few works which we are going to present hereafter.

Qin et al. in [48] extracted spectral distribution features from the Constant-Q Transform (CQT) domain, which has a higher frequency resolution in the mid-low frequency. This choice is justified by the fact that this frequency interval contains the greatest discriminatory power, especially in case of the distinction of different devices from the same manufacturer. A CNN was used by Baldini et al. in [49] and [50]. In these works, the authors proposed a method for both identifications and authentication problems. The operating principle is based on the stimulation of the built-in microphone with non-speech sounds at different frequencies. That is because it was empirically found that the best identification result is obtained by using a specific part of the frequency magnitude component located in a frequency band between Direct Current (DC) (0 Hz) and the adopted stimulus frequencies (1 KHz or 2 KHz). After that, the correspondent DFT was calculated and the classication was performed using a CNN. One of the latest works addressing this problem was proposed by Verma et al. in [1]. They proposed a CNN-based system for capturing the device signature using DFT information. The most remarkable contribution of this work was that they address the problem of speaker-independent scenario[2]. The final prediction for a single audio file was obtained by combining the single prediction at frame level. In addition to variable Signal-to-Noise Ratio (SNR) noise-corruption (AWGN) studies, they also investigated the effect in device identification of double Adaptive Multi-Rate (AMR) compressed audio recordings.

3.2 Problem Statement

The goal of this work is to develop a new and improved methodology to solve the source identification task. This means being able to link an audio recording to one of the available devices. In particular, we want to address the problem in a closed-set scenario, thus a finite number of possible devices.

More formally, given a set of J audio recordings $\mathcal{R} = \{x_1, x_2, \dots, x_J\}$ and a set of K possible device models $\mathcal{D} = \{c_1, c_2, \dots, c_K\}$, our goal is to associate $x \in \mathcal{R}$ to a specific $c \in \mathcal{D}$ producing the pair (x_j, c_k), indicating that the j-th recording was recorded using the k-th device.

In this work, this identification task relies on the estimation of the channel response associated with a specific device, i.e. a mobile-phone microphone, that is used as a device fingerprint.

The channel response is extracted from the considered recording $x \in \mathcal{R}$ after some signal processing procedure, in order to enhance its infor-

[2]i.e. make the device identification independent from the speaker speaking in the audio recording.

mative content. Here, we formulate the channel estimation problem from a speech recording, which is the main core of this work.

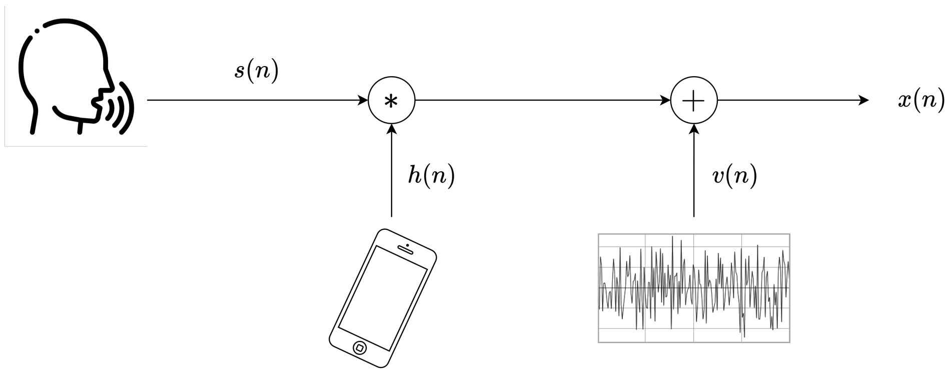


Figure 3.1: The recorded audio signal model.

Let us consider a generic digital audio recording of a speaker $x(n)$, $n \in [0; N-1]$ as shown in Figure 3.1. The clean speech signal $s(n)$ is recorded by a device characterized by an impulse response $h(n)$. The device may also additionally corrupt the signal introducing some noise $v(n)$. The recorded audio signal can be modeled in the discrete time domain as follows [51]:

$$x(n) = s(n) * h(n) + v(n), \tag{3.1}$$

where $x(n)$ is the recorded audio signal, $s(n)$ is the speech-only audio signal at the microphone, $h(n)$ is the impulse response of the microphone, $v(n)$ is the noise contribution and $*$ denotes the convolution operator. Moving to the time-frequency domain, using the STFT, the equation in (3.1) can be expressed as:

$$X(k,l) = S(k,l)H(k,l) + V(k,l), \tag{3.2}$$

with k and l indicating the frequency and time frame correspondingly and $X(k,l), S(k,l), H(k,l), V(k,l)$ are complex number sequences.

If we assume that the frame length of the STFT is large compared to the impulse response $h(n)$ [51, 52], we can consider that the microphone response $H(k,l)$ is constant over time. This means that this response varies more slowly than the speech content[3], leading us to the following approximation:

$$X(k,l) \approx S(k,l)H(k) + V(k,l). \tag{3.3}$$

[3]the microphone response does not vary significantly with l.

In addition, if we assume to be in an ideal noiseless case, i.e. $V(k,l) = 0$, we can also write:

$$|X(k,l)|^2 \approx |S(k,l)|^2 |H(k)|^2. \tag{3.4}$$

Passing to the logarithms we obtain an additive relationship, i.e.:

$$\log|X(k,l)| \approx \log|S(k,l)| + \log|H(k)|. \tag{3.5}$$

If an estimate $\hat{S}(k,l)$ of the initial speech signal $S(k,l)$ is present, then we can compute an estimate $\hat{H}(k)$ of the channel frequency response, i.e.:

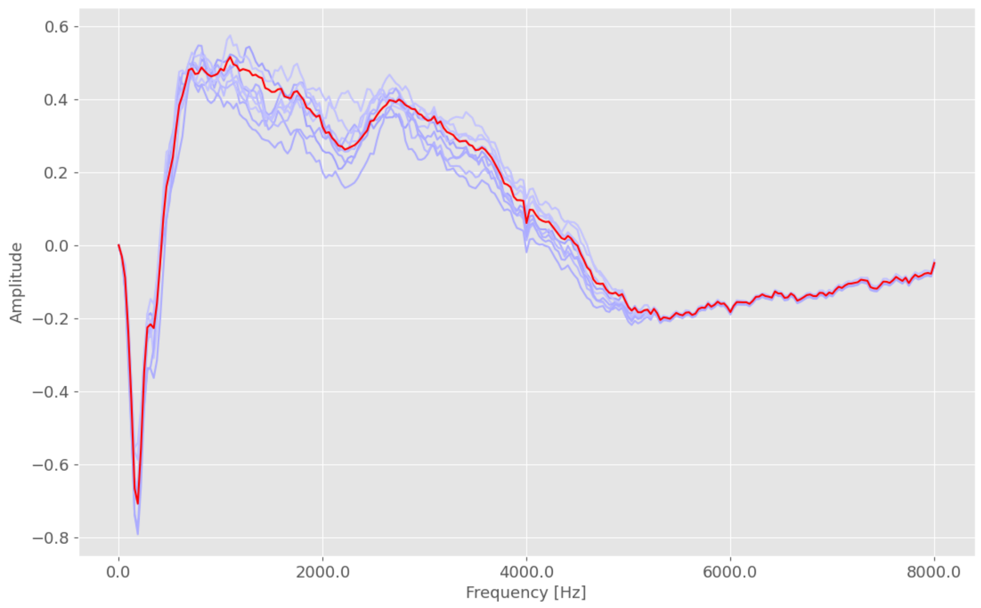
$$\hat{H}(k) = \frac{1}{L}\sum_{l=1}^{L}(\log|X(k,l)| - \log|\hat{S}(k,l)|), \tag{3.6}$$

where L is the total number of time frames.

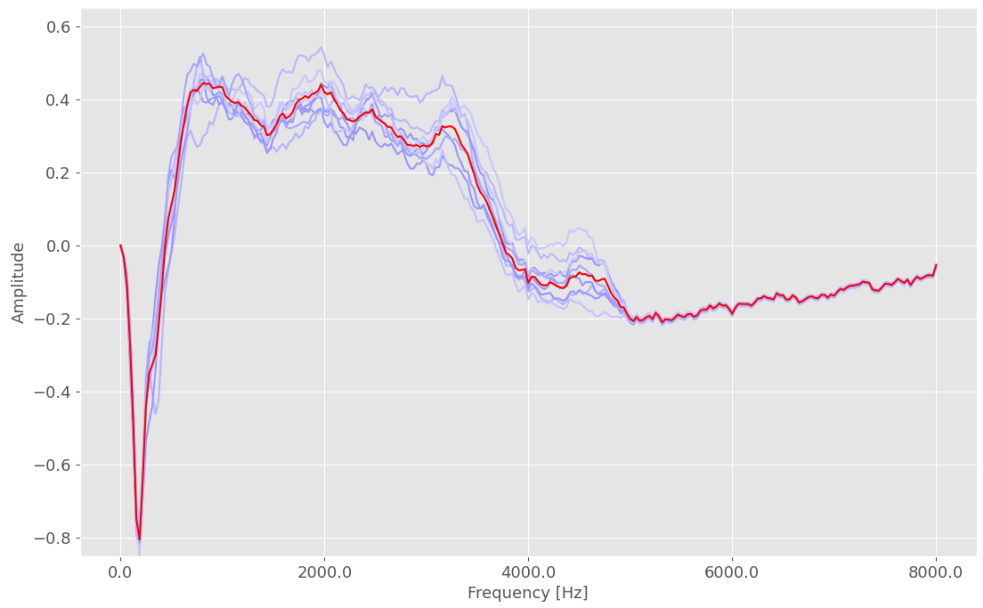
As we can see in Figure 3.2 if we consider different devices the resulting frequency channel responses differ greatly from one another.

Unfortunately, in a real scenario, we are always dealing with speech audio recordings affected by various kinds of noise signals. In particular, the AWGN breaks down the majority of the modern methods adopted for passive fingerprint extraction, such as the channel response. In addition, most of the anti-forensic attacks relies on the voluntary noise injection to the audio data, with the goal of obfuscating any device fingerprint, making its extraction more challenging [53].

Our goal is therefore to limit the negative influences of the noise, in the context of passive fingerprint extraction, trying to recover its discriminant power for our final device identification.



(a) Sony Ericsson c902.



(b) Samsung E2121B.

Figure 3.2: The extracted channel response from two different mobile-phones.

4 Proposed Method

In this chapter we present the proposed method for device identification. We first provide a general overview of the system. Then, we perform a detailed analysis of all its elementary parts.

4.1 Overview

The proposed method is an improved version of a blind channel estimation algorithm proposed by Cuccovillo et al. in [11, 12, 54].

Our system can be thought of as a composition of different principal blocks that are shown in Figure 4.1. Starting from an audio recording, we obtain the relative frequency representation by performing a time-frequency transform. After that, a specific signal processing technique is applied in order to remove unwanted noisy component. From the resulting denoised frequency representation, we are able to compute a reliable estimation of the speech signal that is present in the initial recording. The estimation procedure is carried out using a pre-computed clean speech model. The estimated speech component is then removed from the original signal. With the remaining information, we can compute a specific

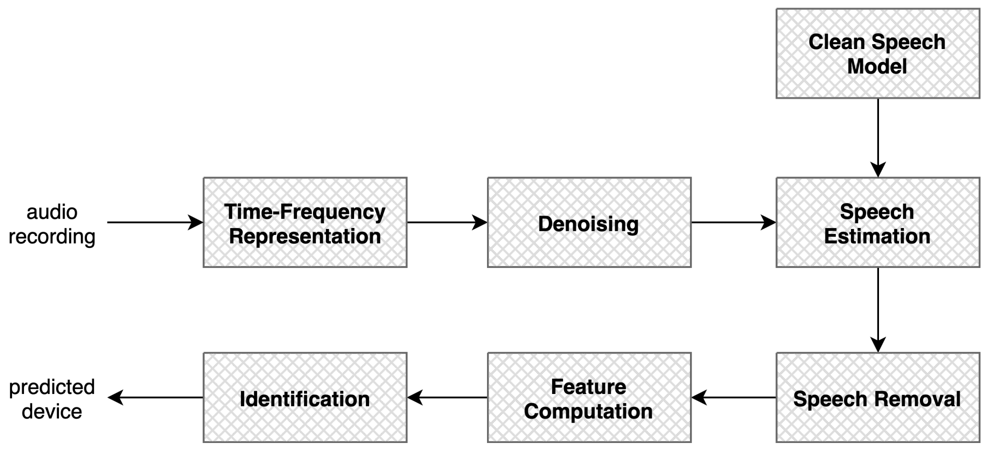


Figure 4.1: The main components of our proposed method.

feature that is adopted in the identification problem. The final identification output represents a prediction of the device that was used to capture the initial audio recording.

In Figure 4.2 a more detailed version of the proposed method is shown, with a clear reference of the main components introduced in Figure 4.1.

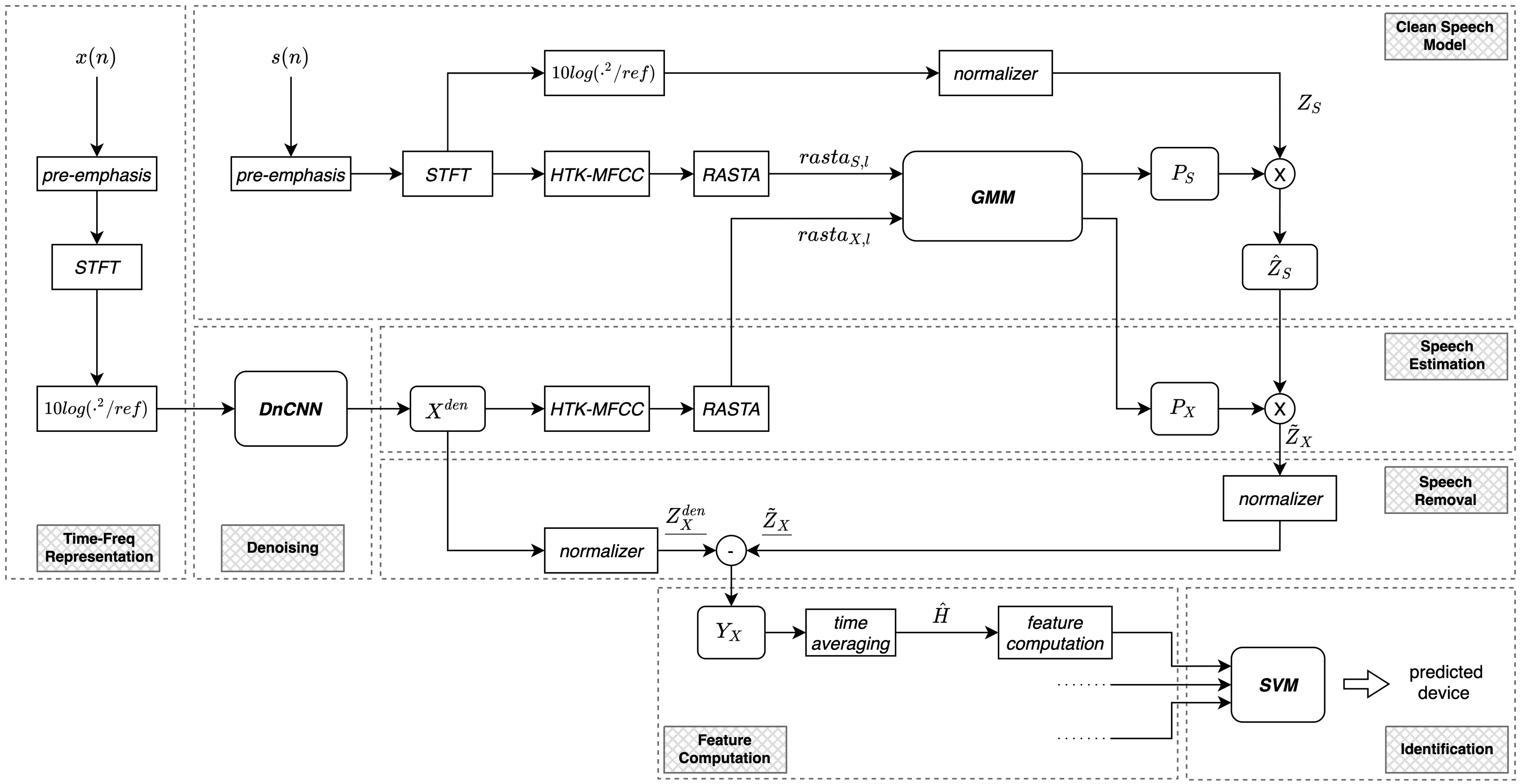


Figure 4.2: The proposed method flowchart.

4.2 Building Blocks

In the following sections, we explain all the various blocks that make up our system, describing in detail their functionalities and the relationships with the other blocks.

4.2.1 Clean Speech Model

There exist several robust methods for channel response estimation when both the input signal and the observed signal are known. Indeed, as we have seen in Section 3.2, if we assume to know the magnitude spectrum of the speech signal, we can easily estimate our channel response $\hat{H}(k)$ using Equation (3.6). However, in practical scenarios only the observed signal is available. This leads to the need for a *blind channel estimation*[1] procedure.

In this specific scenario, the input speech signal $S(k, l)$ is unknown, but we can try to get an estimate $\hat{S}(k, l)$ from the observed audio recording $X(k, l)$. The quality of the estimated channel depends on the match between the estimated speech spectrum $\hat{S}(k, l)$ and the real one $S(k, l)$. Since the absolute level of the speech is unknown, the channel $\hat{H}(k)$ can be estimated only within an unknown scale factor [51].

Different methodologies for modeling speech from a clean-speech signal can be found in the speaker recognition literature. Authors in [51, 52] perform RASTA filtering in the cepstral domain leading to the use of the so called RASTA filtered Mel-Frequency Cepstral Coefficients (RASTA-MFCC). This feature has shown to be robust to the distortion introduced by the acquisition device, i.e. microphone, reducing the channel effect[2]. Using the RASTA-MFCC as a feature, the authors propose the use of a classifier to find a finer grid of generic speech spectrum components. They call this estimated grid a set of *classes of average speech spectra*. For a greater self-explainability, from now on we refer to it as the *phoneme spectrum basis*, in that it aims to be a composition of different spectra related to the different phoneme sounds that may be found on a generic speech corpus.

We use a GMM density estimator to provide this general structure. Thanks to the estimated probability, the GMM then acts as a selector

[1]blind because the clean input signal is unknown.

[2]this property is explained in Section 2.1.3.

for the different phonemes spectrum that are present in the phoneme spectrum basis. A convex combination of these phoneme spectra is used to approximate the clean speech signal component $\hat{S}(k,l)$ of our observed recorded signal $X(k,l)$.

In this way we try to eliminate all the unnecessary components that could influence our channel estimation, making this procedure as independent as possible from the speaker involved[3].

It should be noted that, differently from the original work, we adapt the algorithm to use a sampling rate of $F_s = 16\text{kHz}$, doubling the original one. This is in line with the new types of devices, which allow a higher sampling rate compared to the one used in the original work.

We decompose this first block in five main steps:

1. **HTK-MFCCs computation**. We compute the MFCCs adopting the HTK specification in [17]. These HTK parameters interest the *Pre-emphasis* filter and the *lifter* coefficient as well as the adopted DCT definition and the number of filters in the Mel filter bank. It is worth saying that the HTK Hz-to-mel conversion formula used in this work and reported in Equation (2.3) has been found to give a smoothed channel estimation compared to the one obtained using the classic one provided in [55] [11].

   In line with Equation (3.1), we denote $s(n)$ our speech-only audio recording. $s(n)$ is used as a training audio for the GMM density estimator. From the recording, we compute the MFCCs as follow:

   i *Pre-emphasis filtering*: in speech processing, a pre-emphasis filter is often used to compensate for the energy drop at high frequency, equalizing the levels of the formant resonances [56, 57]. Indeed, in every magnitude spectrum of a speech signal, a majority of the energy is located in the lower end of the spectrum. The filtered signal results in a more flattened spectrum and is more suitable for future signal processing applications.

   We use the standard first-order auto-regressive pre-emphasis filter, i.e.:

$$s(n) = s(n) - \alpha \cdot s(n-1), \tag{4.1}$$

[3]the RASTA-MFCC features has been proven to be independent also from the speaker's gender and its peculiar timbre [20, 11].

where α denotes the pre-emphasis coefficient. We use $\alpha = 0.97$ since it matches the pre-emphasis filter used in the HTK implementation of the MFCCs computation.

ii *Windowing*: the speech recording $s(n)$ is split in a 50% overlapped frames with 32ms length[4] and windowed using an *hanning* window[5]. We want to point out that this time condition on frames is crucial for a successful clean speech estimation, as we can assume stationary characteristics for a single phoneme within this interval. We denote this windowed time frame as $s_l(n)$.

iii *Magnitude spectrum computation*: we compute the DFT of the l-th frame and take its absolute value:

$$MAG_l(k) = |S_l(k)| = |\mathcal{F}(s_l(n))|, k \in \left[1; \frac{N_{fft}}{2} + 1\right], \quad (4.2)$$

where MAG stands for magnitude and $\mathcal{F}(\cdot)$ represents the DFT of a signal. The boundaries of k indicate that we only take the first half part of the considered spectrum for symmetry reasons.

iv *Triangular filter bank application*: as described in Section 2.1.2, in order to calculate the MFCCs we have to map our power spectrum onto the Mel-scale. We use a frequency range that goes from 0Hz to 4kHz. On this, we adopt 26 different Mel bands [17].

We denote with h_{triang} the uniformly spaced triangular overlapping filter bank on this Mel-scale. Thanks to this filter, we compute the relative $FilterBankPower(FBP)$ for each of the l-th recording frames, as follow:

$$FBP_l = h_{triang} \cdot MAG_l(k)^2, \quad (4.3)$$

resulting in a list of Mel powers.

v *MFCC computation*: we log-compress and decorrelate these $FBPs$ using a DCT:

$$\mathbf{f}_{\mathrm{MFCC}_l} = \mathrm{dct}(\log(FBP_l)). \quad (4.4)$$

[4]that corresponds to 512 samples using a sampling rate of 16kHz.

[5]$w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right)$, with N denotes the window length.

To comply with the HTK [17] specifications, we use the dct *type iii*[6] definition in [58]. The MFCCs are the amplitudes of the resulting spectrum. The first 13 are taken.

vi *Cepstral liftering*: we perform a sinusoidal cepstral liftering[7] of the MFCCs with $L = 22$:

$$\mathbf{f}_{\text{HTK-MFCC}_l}(k) = \mathbf{f}_{\text{MFCC}_l}(k) \cdot \left(1 + \frac{L}{2} \sin\left(\frac{\pi}{L}(k+1)\right)\right). \quad (4.5)$$

2. **RASTA filtering**. Each one of the $\mathbf{f}_{\text{HTK-MFCC}_l}$ is RASTA filtered:

$$\mathbf{f}_{\text{RASTA-HTK-MFCC}_k}(l) = \mathbf{f}_{\text{HTK-MFCC}_k}(l) * \text{RASTAfil}, \quad (4.6)$$

where $*$ symbol denotes the convolution operation and RASTAfil represents the RASTA filter explained in Section 2.1.3. Note that we are filtering each of the k-th MFCCs time-series independently for each l-th frame. We obtain L_s different $\mathbf{f}_{\text{RASTA-HTK-MFCC}_l}$ features, since we have L_s frames on the training recording.

3. **GMM training**. We denote with $rasta_{S,l}$ the $\mathbf{f}_{\text{RASTA-HTK-MFCC}}$ referred to the l-th frame of the training audio recording $s(n)$. We use these as feature vectors (excluding the first coefficient[8], $k = 0$) to train a GMM with $M = 1024$ mixtures. After the training procedure we obtain:

i *GMM parameters*: the mean $\boldsymbol{\mu}_i$, covariance matrix $\boldsymbol{\Sigma}_i$, and weight π_i of each i-th mixture.

ii *GMM mixture probability*: similarly to the Equation (2.17), we get the relative mixture probability, that is, the probability that the feature vector $rasta_{S,l}$ belongs to the i-th mixture:

$$p(z_i = 1 \mid rasta_{S,l}) = \frac{\pi_i \mathcal{N}\left(rasta_{S,l} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)}{\sum_{m=1}^{M} \pi_m \mathcal{N}\left(rasta_{S,l} \mid \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m\right)}, \quad (4.7)$$

where z_i is the selection variable[9] for the i-th component of the GMM and $\mathcal{N}\left(\cdot \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)$ denotes a multivariate Gaussian distribution, as in Equation (2.14). These probabilities are used

[6]often referred as "the inverse DCT".

[7]filtering in the cepstrum domain.

[8]this coefficient represents the average log-energy of the input signal, which only carries little speaker-specific information [59].

[9]as we have seen in Section 2.2.3.2, $z_i \in \{0, 1\}$; i.e. $z_i = 1$ denotes that the i-th mixture is generating the current frame.

to build a matrix $P_S \in \mathbb{R}^{M \times L_s}$ where M indicates the number of mixtures and L_s indicates the total number of frames of the training audio recording $s(n)$. Formally we can write:

$$P_S = \left(p\left(z_i = 1 \mid rasta_{S,l}\right)\right), i \in [1; M], l \in [1; L_s]. \qquad (4.8)$$

4. **Log-power spectrum normalization**. In order to avoid issues with signal level differences that may arise in the identification process, the log-power spectrum of each frame is normalized by subtracting its mean value[10], i.e.:

$$Z_{S,l} = \log\left(|S_l|^2\right) - \frac{1}{N_{fft}} \sum_{k=1}^{N_{fft}} \log\left(|S_l(k)|^2\right), \forall l \in L_s, \qquad (4.9)$$

where all the $Z_{S,l}$ form the matrix $Z_S \in \mathbb{R}^{L_s \times N_{fft}}$ that contains the normalized log-power spectrum of all the training audio frames.

5. **Phoneme log-power spectrum basis computation**. We combine the relative mixture probabilities P_S and the normalized log-power spectrum Z_S matrices to obtain a weighted set of M average clean speech log-power spectra over all the available training recording frames L_s and thus, the phoneme spectrum basis we have been searching for, i.e.:

$$\hat{Z}_S = Z_S \cdot P_S. \qquad (4.10)$$

In this phoneme spectrum basis matrix $\hat{Z}_S \in \mathbb{R}^{M \times N_{fft}}$ the i-th row indicates the i-th mixture that provides the log-power spectrum associated with a specific phoneme, each column instead, represents a specific frequency bin of the correspondent spectrum. In Figure 4.3 a graphical representation of the phoneme spectrum basis used in this work is shown.

4.2.2 Denoising

The main contribution of this work is carried out by the denoising block. Working directly on the time-frequency domain as in the Equation (3.2),

[10]this process should not be confused with cepstral mean subtraction which aims to neutralize the channel [51]; this normalization only affects the level of the log-power spectrum.

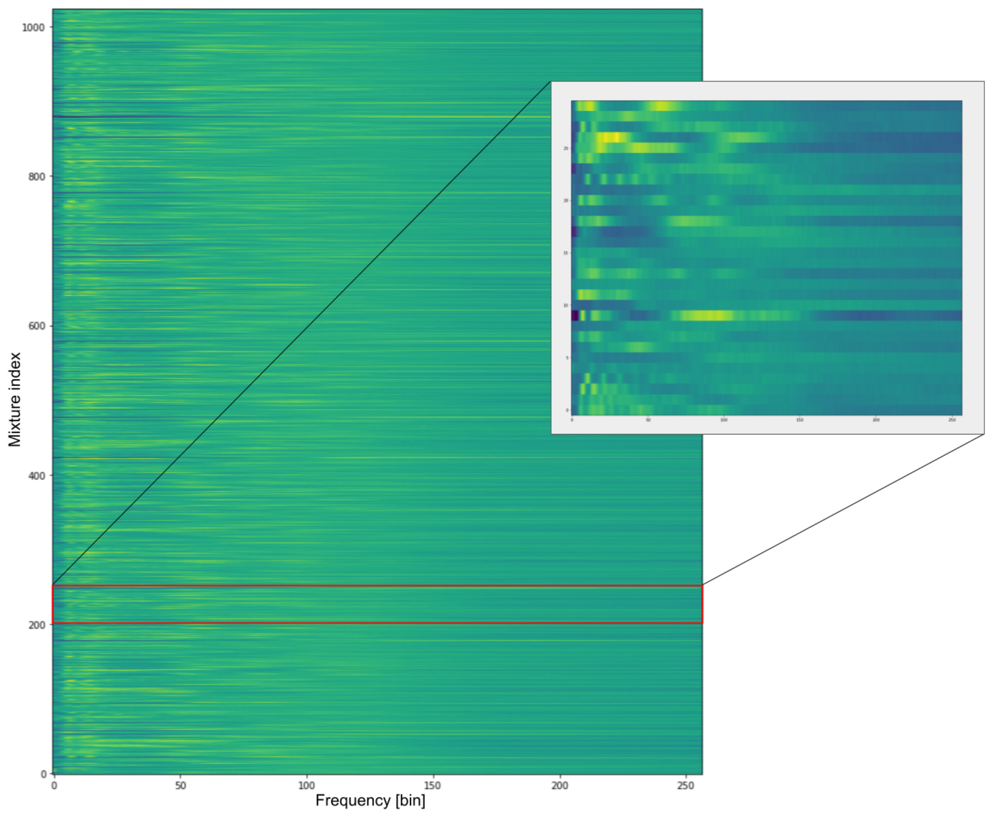


Figure 4.3: The phoneme spectrum basis matrix. In detail, we can see its hidden structure, where the spectrum values are shown using a heat-map; from here it can be easily seen that a specific phoneme spectrum can be selected by choosing the respective row, that is associated with one of the 1024 Gaussian components. The columns instead represent the different frequency bins of the correspondent DFT.

the observed signal can be seen as the sum of multiple contributes:

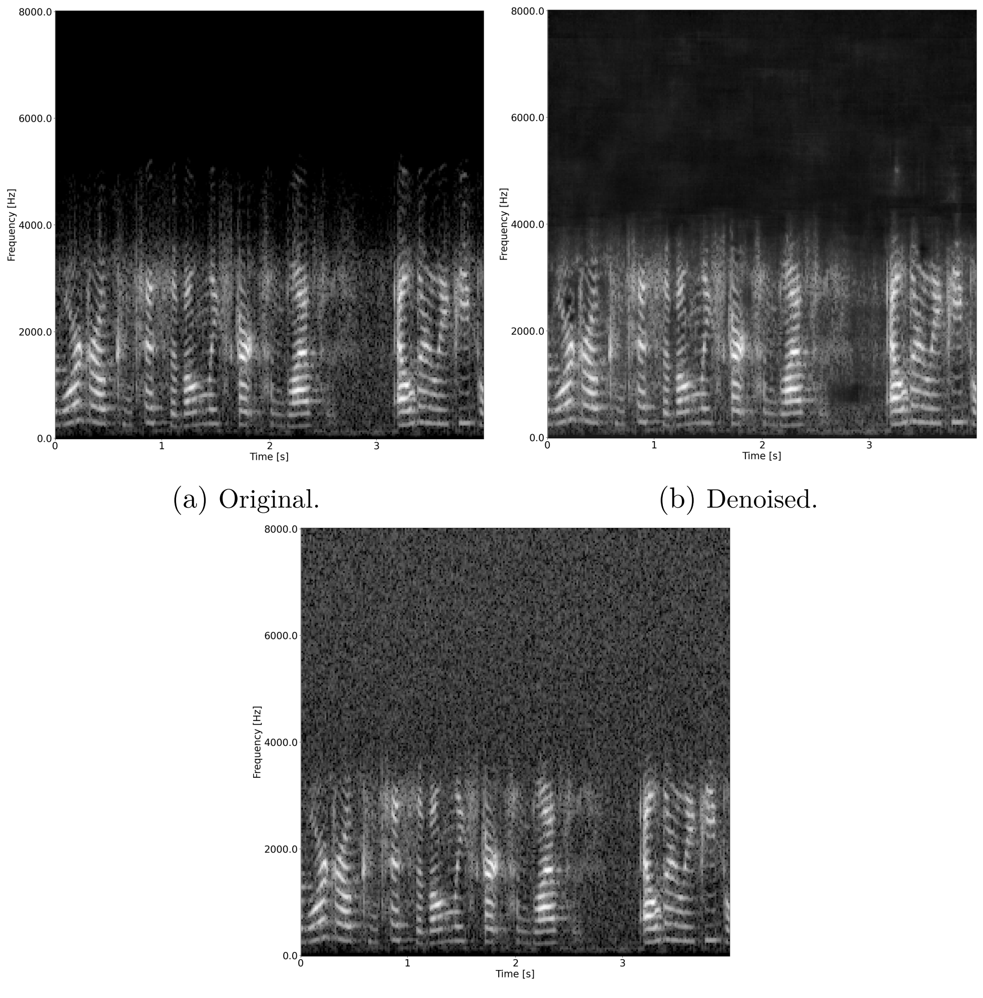
$$X(k,l) \approx S(k,l)H(k) + V(k,l), \tag{4.11}$$

where $X(k,l)$ is the only recording frame we can observe, $S(k,l)$ is the speech-only frame, $H(k)$ is the transfer function of the microphone that we want to preserve and $V(k,l)$ is the AWGN contribution that we aim to suppress. Our goal is therefore to lead us back to the noiseless case, as assumed in Equation (3.4) since the noise contribution corrupts our channel estimation irreversibly.

Formally, given a spectrogram $X(k,l)$ as input to our denoiser, we estimate a new spectrogram $X^{den}(k,l)$ where the additive noise component $V(k,l)$ is nearly zero, attempting to preserve the intrinsic spectral characteristics of the device's microphone. Formally, we can write:

$$\begin{aligned} X^{den}(k,l) &= \mathrm{DEN}\left(X(k,l)\right) \\ &= \mathrm{DEN}\left(S(k,l)H(k) + V(k,l)\right) \\ &\approx S(k,l)H(k), \end{aligned} \tag{4.12}$$

where DEN$(\cdot)$ stands for the denoising operation carried out by our denoiser. A visual example of the denoising process performance is shown in Figure 4.4.



(a) Original.

(b) Denoised.

(c) Noisy.

Figure 4.4: The spectrogram denoising result performed by the DnCNN. The noise corrupted spectra in 4.4c is processed to obtain a denoised version 4.4b. Our goal is to obtain a denoised spectrogram as similar as possible to the original one, in 4.4a.

As we will see in Section 4.2.3.1, this operation is made before the feature extraction, to get the best results on the channel estimation process.

After preliminary studies on audio denoising covered in Section 5.2.2, we chose to tackle this problem in the time-frequency domain. By working with the time-frequency representation of the audio signal we can take great advantage of the well-established methods for image denoising.

In particular, we decided to adopt an architecture known as Denoising CNN (DnCNN). The network is a very deep feed-forward CNN proposed by Zhang et al. in [60] and inspired by the Visual Geometry Group (VGG) network [61]. We chose to use a kernel size of 3×3 with 15 stacked convolutional hidden layers.

For a better comprehension of the DnCNN architecture, in Figure 4.5 the stacked layers structure is shown.

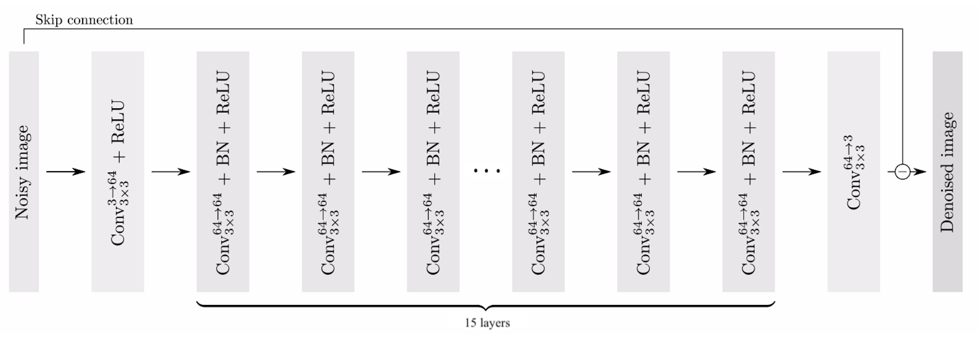

Skip connection
Noisy image
$\mathrm{Conv}^{3\to64}_{3\times3}$ + ReLU
$\mathrm{Conv}^{64\to64}_{3\times3}$ + BN + ReLU
$\mathrm{Conv}^{64\to64}_{3\times3}$ + BN + ReLU
$\mathrm{Conv}^{64\to64}_{3\times3}$ + BN + ReLU
$\mathrm{Conv}^{64\to64}_{3\times3}$ + BN + ReLU
$\mathrm{Conv}^{64\to64}_{3\times3}$ + BN + ReLU
$\mathrm{Conv}^{64\to64}_{3\times3}$ + BN + ReLU
$\mathrm{Conv}^{64\to3}_{3\times3}$
Denoised image
15 layers

Figure 4.5: The architecture of the DnCNN network.

4.2.3 Blind Channel Estimation

This section concerns the denoising, the speech estimation and the speech removal blocks. The main goal of this part is the extraction of the device's frequency channel response from an observed speech recording corrupted by noise.

We remark the fact that we refer to this as a blind channel estimation problem because the clean input signal is unknown. The observed signal can be considered as the composition of speech, noise and microphone contributes.

Let us denote with $x(n)$ our observed audio recording. The blind channel estimation procedure on an observed recording can be split into two different sub-processes:

1. *Speech Estimation*;

2. *Channel Estimation.*

4.2.3.1 Speech Estimation

We assume that $x(n)$ is a noise-corrupted speech audio recording. We aim at suppressing both the noise and the speech contributions. For the speech part we use the pre-trained GMM introduced in Section 4.2.1, to obtain spectral information from the observed recording. In specific, we use the triplets $(\pi_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ associated with each mixture and the phoneme spectrum basis contained in the $\hat{Z}_S$ matrix. For the noise part instead, we use a denoiser acting in the time-frequency domain.

This procedure interests many blocks of our system. To better analyze the process, we can divide it into eight different steps:

1. **Re-sampling**. The observed recording has to be re-sampled to $F_s = 16$kHz - if needed - in order to match the training audio recording sampling rate. This is necessary as the GMM estimation procedure works properly only if the training and the testing sampling rate are equal.

2. **Pre-emphasis filtering**. As before we perform a pre-emphasis filtering adopting the filter in Equation (4.1).

3. **Denoising**. The denoising part needs some preliminary data manipulation in order to be properly done. Thus, for the sake of clarity we decompose this procedure into three parts:

   i *STFT computation*: the observed recording $x(n)$ is transformed into time-frequency domain using the STFT. This time-frequency transformation is made adopting a 50% overlapping time frame with a window length of 512 samples. For the frequency analysis, we use the same number of points of the time analysis window, obtaining 256 frequency bins for symmetry reasons.

   ii *Spectrogram splitting*: the resulting spectrogram is split into non-overlapping chunks X of dimension 256×256. Each spectrogram matrix represents a duration of 4.096 seconds of our observed recording. It is important to notice that a time frame window of 512 samples at $F_s = 16$kHz, corresponds to a frame duration of 32ms. This is important because, as explained in Section 4.2.1 point 1b, this is a necessary condition in order to get a proper phoneme estimation for modeling a clean speech signal.

We denote with L_x the 256 time frames composing a single spectrogram chunk.

iii *Denoising*: all the spectrograms are converted into log-power spectrograms before the denoiser block. Using these spectrograms, our pre-trained DnCNN denoising net is responsible for suppressing the unwanted noise contribute contained in them. The resulting denoised spectrogram X^{den} is used for the next signal processing steps.

It is important to stress that only one spectrogram at a time is employed to perform a single channel estimation procedure. Indeed, multiple spectrogram from the same recording will reveal very similar channel response as they come from the same device. This characteristics impose the maximum audio length adopted in the channel estimation procedure. As explained above, in this case we adopt 4.096 seconds.

4. **HTK-MFCCs computation**. From the denoised spectrogram X^{den}, we compute a HTK-MFCC matrix. Following the steps done in Section 4.2.1 point 1d, 1e and 1f, for each time frame we compute the $\mathbf{f}_{\text{HTK-MFCC}_l}$ feature vector referred to the l-th time frame of the denoised spectrogram's chunk X^{den} resulting from the observed recording $x(n)$.

5. **RASTA filtering**. All the $\mathbf{f}_{\text{HTK-MFCC}_l}$ are RASTA filtered to obtain L_x different $\mathbf{f}_{\text{RASTA-HTK-MFCC}_l}$ feature vectors.

6. **GMM mixture probability computation**. Using the same notation adopted earlier, with $rasta_{X,l}$ we refer to the $\mathbf{f}_{\text{RASTA-HTK-MFCC}_l}$ feature associated to the l-th frame of the denoised spectrogram X^{den}. Using the pre-trained GMM, we compute the probability that the $rasta_{X,l}$ feature vector belongs to the i-th mixture. Thus, in a specular way to Equation (4.7) we can write:

$$p(z_i = 1 \mid rasta_{X,l}) = \frac{\pi_i \cdot \mathcal{N}\left(rasta_{X,l} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)}{\sum_{m=1}^{M} \pi_m \cdot \mathcal{N}\left(rasta_{X,l} \mid \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m\right)}. \tag{4.13}$$

Even in this case, these probabilities are used to build a matrix $P_X \in \mathbb{R}^{M \times L_x}$ where $M = 1024$ indicates the number of mixtures and L_x indicates the total number of time frames of the denoised spectrogram.

7. **Log-power spectrum normalization**. As before, we normalize the log-power spectrum of each l-th time frame of the denoised spectrogram by subtracting its mean value. Note that since the X^{den} spectrogram is already log-power, we do not have to perform the log of the squared magnitude. We can write:

$$Z_{X,l}^{den} = \left(|X_l^{den}|\right) - \frac{1}{N_{fft}} \sum_{k=1}^{N_{fft}} \left(|X_l^{den}|\right), \forall l \in L_x, \tag{4.14}$$

where all the $Z_{X,l}^{den}$ form the matrix $Z_X^{den} \in \mathbb{R}^{L_x \times N_{fft}}$ that contains the normalized log-power spectrum of all the frames composing the denoised spectrogram X^{den}.

8. **Speech-only power spectrum estimation**. By using the relative mixture probability matrix P_X as a selection matrix on the phoneme spectrum basis $\hat{Z}_S$, we are able to estimate the non-filtered ideal clean-speech spectrogram of the observed recording. Formally, we can write:

$$\tilde{Z}_X = P_X^t \cdot \hat{Z}_S, \tag{4.15}$$

with $\tilde{Z}_X \in \mathbb{R}^{L_x \times N_{fft}}$. Using this weighted combination approach rather than a Maximum Likelihood selection can be advantageous when there is any uncertainty in the classification [52].

4.2.3.2 Channel Estimation

This final procedure for the blind channel estimation part can also be divided in three different steps:

1. **Spectrum Normalization**. To avoid issues with signal level differences, both the estimated speech-only power spectrogram $\tilde{Z}_X$ and the denoised normalized log-power spectrogram Z_X^{den} need to be normalized between 0 and 1:

$$\begin{cases} \underline{\tilde{Z}_X} = \underset{[0,1]}{\operatorname{norm}}(\tilde{Z}_X) \\ \underline{Z_X^{den}} = \underset{[0,1]}{\operatorname{norm}}(Z_X^{den}) \end{cases} \tag{4.16}$$

where $\underset{[0,1]}{\text{norm}}(\cdot)$ stands for:

$$\underset{[0,1]}{\text{norm}}(\cdot) = \frac{(\cdot) - \min(\cdot)}{\max(\cdot) - \min(\cdot)}. \tag{4.17}$$

2. **Speech Removal**. We compute our speech-free spectrogram by spectral subtracting the estimated ideal speech spectrum $\tilde{Z}_X$ from the observed denoised one, as in Equation (3.6), i.e.:

   $$Y_X = \underline{Z_X^{den}} - \underline{\tilde{Z}_X}. \tag{4.18}$$

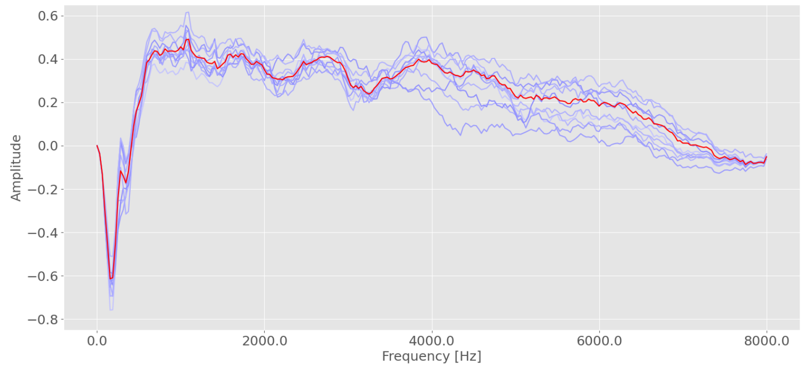
   Note that, in order to perform a proper spectral subtraction, both the spectrograms have to be log-power.

3. **Time Averaging**. We estimate our final channel frequency response by averaging the speech-free spectrogram Y_X through time, i.e.:

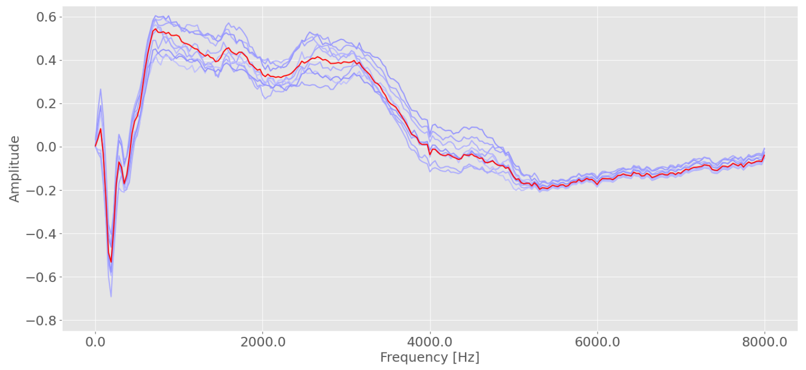
   $$\hat{H} = \frac{Y_X^t \cdot \mathbf{1}}{L_x}, \tag{4.19}$$

   where $Y_X^t \in \mathbb{R}^{N_{fft} \times L_x}$, $\hat{H} \in \mathbb{R}^{N_{fft} \times 1}$ and $\mathbf{1}$ is an all-ones vector of dimension $L_x \times 1$.

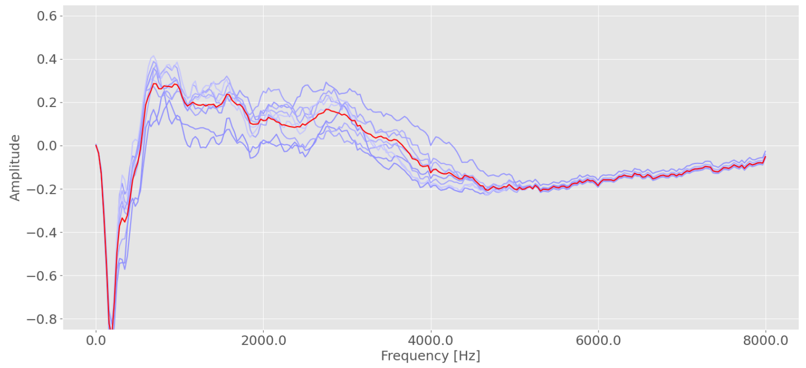
In Figure 4.6 we can see how this channel estimation gives a recognizable pattern for a specific device. The “coloring” effect is distinguishable from device to device, for each individual channel estimate.



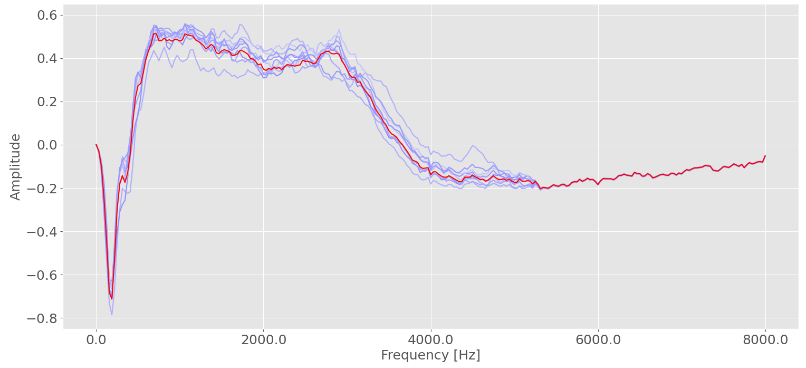
(a) Apple iPhone 5.



(b) Vodafone joy 845.



(c) HTC Sensation xe.



(d) LG GS290B.

Figure 4.6: Multiple channel responses extracted using our modified algorithm. Each figure refers to a specific device model. We can notice a remarkable trend (red line) on each single channel response (blue line) for a specific device.

4.2.4 Feature Computation

The channel response $\hat{H}$ computed in Equation (4.19) brings just a portion of all the information available within the observed recording $x(n)$. To maximize the available information, we build a specific custom feature denoted with f and composed by three multidimensional feature vector, namely f_1, f_2, f_3 [11, 12].

In particular, f_1 contains all the information available from the channel estimation algorithm, f_2 describes the correlation between the estimated channel response and the original log-power spectrum of the observed audio recording, while f_3 only describes the properties of the observed audio recording.

As preliminary step, from the normalized log-power denoised spectrogram of the observed recording $\underline{Z_X^{den}}$, computed in Equation (4.16), we derive $\hat{p}$, i.e.:

$$\hat{p} = \frac{\underline{Z_X^{den}}^t \cdot \mathbf{1}}{L_x},$$

where $\mathbf{1}$ is an all-ones vector of dimension $L_x \times 1$. Here, $\hat{p}$ represents the approximated normalized power of observed recording $x(n)$.

Feature 1 The first feature f_1 is responsible for enhancing the information obtained by the channel estimation algorithm.

This feature can be computed as follow:

1. Apply a variable gain between the estimated channel response $\hat{H}$ and the average value of $\hat{p}$, dependent on the power of the channel, i.e.:

$$h_1 = \frac{\hat{H} + \overline{\hat{p}}}{\|\hat{H}\|^2},$$

   where $\overline{\hat{p}}$ denotes the average value of $\hat{p}$ and $\|\cdot\|$ denotes the l^2-norm.

2. Compute the first derivative h_1' of h_1, as the inter-sample difference of h_1:

$$h_1'(k) = h_1(k) - h_1(k-1), \forall k \in [2, K],$$

   where K is the total number of frequency bins.

3. Compute the second derivative h_1'' of h_1, as the inter-sample difference of h_1':

$$h_1''(k) = h_1'(k) - h_1'(k-1), \forall k \in [3, K].$$

4. The final feature f_1 is obtained concatenating the three component:

$$f_1 = [h_1, h_1', h_1''] \, .$$

Feature 2 The second feature f_2 is responsible for enhancing the descriptive power of the correlation between the estimated channel response and the original log-power spectrum of the observed recording.

This feature can be computed as follow:

1. Compute h_0 as the division between the estimated channel and $\hat{p}$:

$$h_0 = \hat{H}./\hat{p},$$

   where the operation $(a)./(b)$ performs the element-wise division by dividing each element of a by the corresponding element of b.

2. Apply a variable gain on h_0, dependent on its power:

$$h_2 = \frac{h_0}{||h_0||^2}.$$

3. Also in this case, we compute the first and the second derivative of h_2, resulting in h_2' and h_2''.

4. Apply a variable gain on h_2' and h_2'', dependent on their respective powers:

$$\begin{cases} \widetilde{h_2'} = \frac{h_2'}{||h_2'||^2} \\ \widetilde{h_2''} = \frac{h_2''}{||h_2''||^2} \end{cases}$$

5. The final feature f_2 is obtained concatenating the three components:

$$f_2 = \left[h_2, \widetilde{h_2'}, \widetilde{h_2''}\right] \, .$$

Feature 3 The third feature f_3 is responsible for decrease the redundant information between f_1 and f_2 and reducing the noise due to the content of the recording. For doing this, the feature is computed by manipulating the approximated normalized power $\hat{p}$ of the observed recording. We want to clarify that $\hat{p}$ is generated both by the channel and by the content. Thus, we must ensure that this influence is not strong enough to affect by any means the outcome of the final device identification.

This feature can be computed as follow:

1. Apply a variable gain on the sum between $\hat{p}$ and the average value of the estimated channel response $\hat{H}$, dependent on the power of $\hat{p}$:
$$h_3 = \frac{\hat{p} + \overline{\hat{H}}}{||\hat{p}||^2}.$$

2. Normalize h_3 between 0 and 1:
$$\underline{h_3} = \underset{[0,1]}{\operatorname{norm}}(h_3),$$
where $\underset{[0,1]}{\operatorname{norm}}(\cdot)$ is the same function define in Equation (4.17).

3. Also in this case, we compute the first and the second derivative of $\underline{h_3}$, resulting in $\underline{h_3}'$ and $\underline{h_3}''$.

4. Compute h_4, which contains the absolute value of each component of $\underline{h_3}''$:
$$h_4(k) = \left|\underline{h_3}''(k)\right|.$$

5. Normalize between 0 and 1 both $\underline{h_3}'$ and h_4:
$$\begin{cases} \widetilde{h'_3} = \underset{[0,1]}{\operatorname{norm}}(\underline{h_3}') \\ \widetilde{h_4} = \underset{[0,1]}{\operatorname{norm}}(h_4) \end{cases}$$

6. The final feature f_3 is obtained concatenating the three components:
$$f_3 = \left[\underline{h_3}, \widetilde{h'_3}, \widetilde{h_4}\right].$$

Final Feature The complete feature vector for each recording is computed by collecting together the three features computed so far, i.e.:

$$f = [f_1, f_2, f_3]\,. \tag{4.20}$$

An example of the three components of the final feature f is shown in Figure 4.7.

4.2.5 Identification

Finally, we use these features as discriminant elements for our final closed-set identification problem. For doing this, we adopt a multi-class SVM with an RBF kernel.

More specifics on the SVM parameters and the adopted data manipulation will be discussed in Section 5.1.2.

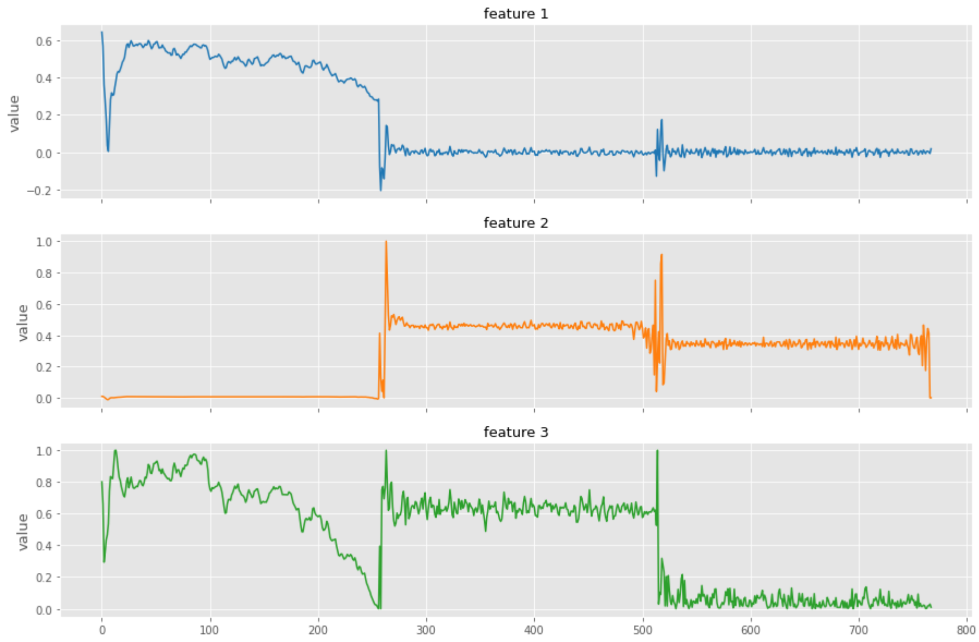


Figure 4.7: The three parts f_1, f_2, f_3, that make up the final feature f.

4.3 Conclusive Remarks

In this chapter, we tackled the problem of device identification from a speech recording in noisy conditions. The main goal was to overcome the negative influence that an additive noise signal has on the channel response extraction. This negative effect was reduced by providing a denoising CNN architecture working on spectrograms of the analyzed recording.

The key principle of the final identification algorithm is based on a modified blind channel estimation method. To suppress the speech content, this method exploits the phoneme characteristics by training a GMM density estimator with RASTA-MFCC features. The resulting GMM is then used with frame-based features obtained from an observed noisy speech signal to find the best matching of clean speech. The observed signal that has undergone a channel influence is first denoised in the time-frequency domain and then used along with the estimated clean speech to perform a spectral subtraction. Then, the temporal mean of the remainder after the subtraction is considered as the unknown channel response. From this, a suitable feature vector is constructed and used as discriminant feature for the final device identification task.

5 Experiments and Results

In this chapter we validate our proposed methodology. After a brief presentation of the adopted datasets and some preliminary design criteria, we go through the final results, performing an in-depth comparison between each experiment.

5.1 Experiment Setup

This section describes the datasets used in our work as well as the data manipulation decisions employed during all the experiments. Lastly, we give a brief explanation of the system's evaluation metrics.

5.1.1 Datasets

LibriSpeech This dataset was introduced by the Center for Language and Speech Processing & Human Language Technology Center of Excellence of the Johns Hopkins University in Baltimore (USA) [62]. The LibriSpeech[1] corpus is derived from audio-books that are part of the

[1] `https://www.openslr.org/12`.

LibriVox [63] project, a volunteer effort that is currently responsible for the creation of public domain audio books, the majority of which are in English. Most of the recordings are based on texts from Project Gutenberg [64], also in the public domain.

It contains 1000 hours of English speech stored in multiple files with a Free Lossless Audio Codec (FLAC) format. The sampling frequency is 16kHz, with bit depth of 16bit, mono channel and variable bitrate. From this corpus, we consider the subset *train-clean-100*, which contains speech tracks for about 100 hours of recording. These tracks are coming from speakers with low error rates on automatic transcription, thus are easily intelligible.

TIMIT The Texas Instruments/Massachusetts Institute of Technology (TIMIT) is a corpus of read speech, specifically designed to provide speech data for the acquisition of acoustic-phonetic knowledge and for development and evaluation of automatic speech recognition systems. The Texas Instruments/Massachusetts Institute of Technology (TIMIT) contains speech from 630 speakers representing 8 major dialect divisions of American English, each speaking 10 phonetically rich sentences. The recordings are stored in a Waveform Audio File (WAV) format, with a sampling frequency of 16kHz, bit depth of 16bit, PCM encoded in mono channel.

MOBIPHONE To the best of our knowledge, the only publicly available dataset that brings together audio recording from different mobile-phone devices is the so-called MOBIPHONE[2]. This dataset was proposed by Kotropoulos et al. in [36] where the authors used it for the same objective of this works, that is mobile-phone device identification.

This dataset contains 504 audio recordings from 21 mobile-phones produced by 7 different manufacturers. The list of the available devices is presented in Table 5.1. As we can see, the list includes some of the major companies in the mobile market, like Samsung, Apple and LG. The recordings consist of utterances from 12 male and 12 female speakers randomly chosen from the TIMIT dataset [65]. Each speaker reads 10 sentences approximately of 3 seconds long. The first two sentences are the same for every speaker, but the remaining are different. Basically, 10

[2]`https://www.dropbox.com/sh/9n7fy7moi825bgk/WFLBKxUitV.`

utterances per speaker were recorded for each device. These utterances were concatenated in a single 30 seconds long recording, to give a total of 24 recordings for each device. The recordings were captured in a silent controlled environment with the same recording equipment. The raw recordings were in AMR format and later converted into WAV format. The sampling frequency is 16kHz, with bit depth of 16bit, PCM encoded, stereo with a bit rate of 512kbps.

MOBIPHONE$_{\mathbf{awgn}}$ This is a modified version of the original MOBIPHONE dataset. This was created by adding an AWGN signal to all the original audio data. This noise-corruption is used to simulate a real scenario, in which a noisy recordings is adopted to perform audio forensic tasks, such as our device identification. In order to do this, we use a specific function provided by the *audiomentation* v0.16.0 Python library [66]. The noise level was set to 30dB SNR for all the recordings, as it appears to be a suitable choice for a standard noise corruption. The augmented dataset has the same technical specifics as the original version.

5.1.2 Parameters and Routines Design

To perform our experiments we had to train three main architectures:

- **GMM** for the *Clean Speech Model*;
- **DnCNN** for the spectrogram *Denoising*;

Table 5.1: Mobile-phones available in the MOBIPHONE dataset [36].

Manufacturer	**Model**		**Manufacturer**	**Model**
HTC	desire c		Apple	iPhone5
HTC	sensation xe		Samsung	E2121B
LG	GS290		Samsung	E2600
LG	L3		Samsung	GT-I8190 mini
LG	Optimus L5		Samsung	GT-N7100 (Galaxy Note2)
LG	Optimus L9		Samsung	Galaxy GT-I9100 s2
Nokia	5530		Samsung	Galaxy Nexus S
Nokia	C5		Samsung	e1230
Nokia	N70		Samsung	s5830i
Sony Ericsson	c902		Vodafone	joy 845
Sony Ericsson	c510i		-	-

- **SVM** for the final device *Identification*.

GMM The GMM was trained on a speech audio file created by randomly merging multiple speech tracks taken from the LibriSpeech dataset until reaching a total duration of 45 minutes. The GMM was trained using $M = 1024$ mixtures and a diagonal covariance matrix. The granularity adopted (number of mixtures M) and the covariance type was chosen based on [11] and confirmed by an additional model selection procedure, based on a BIC score. The combination with the lower score was taken. The GMM was initialized by the use of the K-means algorithm. We used the *scikit-learn* v0.24.1 [67] ML library to build our GMM model.

DnCNN The DnCNN was trained using a custom dataset. This dataset contains spectrograms matrices extracted from the MOBIPHONE and the MOBIPHONE$_{\text{awgn}}$dataset. To build this dataset, each audio recording was filtered by a pre-emphasis filter and then its log-power spectrogram was computed using a 50% overlapping time frame with a window length of 512 samples. For the frequency analysis, we adopted the same number of points of the time analysis window, thus obtaining 256 frequency bins for symmetry reasons. Then the resulting spectrogram was sliced into multiple non-overlapped square-shaped matrices of dimension 256×256. Each segment covers a duration of 4.096 seconds. For computing the time-frequency representation of the audio we used the function provided by the *Librosa* v0.8.0 library [68]. The weight initialization of the DnCNN was performed using the Kaiming-He initialization [69]. We used a batch-size of 20 and a Mean Square Error (MSE) (squared L2 norm) loss function with Adaptive Moment Estimation (Adam) optimizer. It is important to note that the network needs as input the pair clean/noise spectrogram, with the aim of finding a mapping between the two to compute the denoised spectrogram. Thus, the loss function returns the error between the predicted (denoised) and the original (clean) spectrogram. The Learning Rate (LR) was dynamically adapted during the training. More specifically, the LR was decreased by a factor of 1.3 whenever the validation loss did not improve for 4 epochs. This allowed us to start with a relatively high LR value, i.e. 10^{-3}, boosting the learning process. We imposed an early stopping on the training procedure when the validation loss did not decrease for more than 13 epochs. These values have been proven to be a good compromise during preliminary tests. The model providing the best validation loss was selected. Our final network with 30dB SNR noisy spectrograms as input converged at epoch 116. A dataset split policy of 80% for train and 20% for evaluation was used on a total number of 7188 spectrogram's pair. Data shuffling was performed at every epoch. The network was developed using the *PyTorch* v1.8.0 [70] DL framework. We performed the experiments using a workstation equipped with one Intel® Xeon E5-2630 v2 (12 Cores @2.6GHz), RAM 126 GiB, and two Quadro *P*6000 (3840 CUDA Cores @1530MHz), 24GiB, running Ubuntu 20.04.2. LTS. The training procedure took about 8 hours.

SVM The SVM classifier was trained using the custom features resulting from our modified channel estimation procedure. We adopted a RBF kernel type and we left the default values for the regularization and gamma parameters. A dataset split policy of 80% for train and 20% for test was used, with data shuffling before the final classification (i.e. device identification). Unfortunately, since the audio recordings of the mobile-phone model *Samsung s*5830*i* were not of full length, we had to limit our resulting spectrogram chunks to a total number of 85 for each device, corresponding to 85 custom features. As for the GMM, the *scikit-learn* implementation of the algorithm was used.

5.1.3 Evaluation Metrics

The *accuracy* of the resulting device identification task was taken as a metric for evaluating the performance of all the experiments, comparing the predicted mobile-phones models to the corresponding annotated ground labels.

We also evaluated the device identification task by means of a *confusion matrix*, an $N \times N$ matrix where N is the number of classes predicted by our SVM classifier. In our case the number of classes were the number of the different mobile-phones available in the MOBIPHONE dataset, i.e. 21. Each element $n_{i,j}$ of the matrix displays the number of elements with predicted class j and having true class i. We used a normalized version of the confusion matrix, dividing each value of the matrix by the sum of the values of the row it belongs to.

5.2 Preliminary Experiments for Method Design

In this section we explain some of the reasons behind the design criteria of our method.

5.2.1 GMM

The GMM density estimator used in this work has a huge impact on the overall accuracy of the system since it is responsible for modeling a generic clean speech, learning to distinguish the different phoneme spectra that compose a generic speech corpus.

In this sense, the choice of the training audio data adopted is crucial. To make comparisons, the training procedure was done using two different datasets and performing the final identification task using our baseline algorithm proposed by Cuccovillo et al. in [11].

From the identification task result in Table 5.2, we can notice the benefits in using the LibriSpeech [62] instead of the TIMIT [65], which is the one adopted in the original version of the baseline.

Table 5.2: Identification accuracy for a different version of our GMM's train audio recording.

	Accuracy	
Dataset	*Original*	*Pre-emphasized*
TIMIT	0.96195	0.97381
LibriSpeech	0.98095	0.98636

Another modification applied to the GMM training interested the pre-emphasis filter. It has been proven that the final device identification benefits from the adoption of this filtering on the training audio data, as we can see from the results in the right column of Table 5.2. This improvement is justified by the fact that this signal filtering procedure gives more discriminant power also to speech formants that have the major energy component on the higher part of the spectra, balancing the overall spectral energy. This was the rationale behind the adoption of the pre-emphasis filter pre-processing in all the experiments. Figure 5.1 shows its spectral effect on a log-power spectrogram.

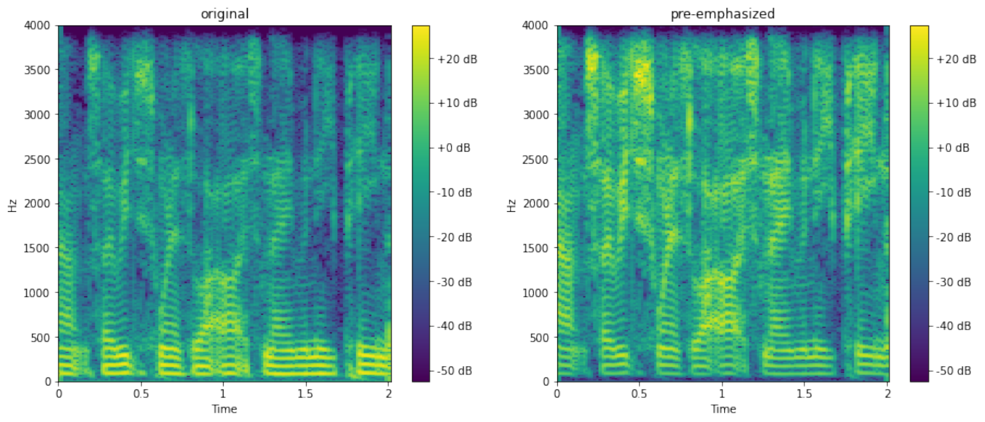


Figure 5.1: The speech pre-emphasis effect. From the spectrogram's heat-map we clearly see the spectral energy balancing before and after the filtering process.

It is also important to notice that the use of two different datasets avoids mutual influences between the train and the test data. For this reason, we trained the GMM using the LibriSpeech corpus and tested the classification baseline with recordings belonging to the MOBIPHONE-dataset, which contains utterances from the TIMIT dataset. The TIMIT dataset has completely different speech content compared to the LibriSpeech, and this aspect increases the generalization of the model as we adopted not only different corpus but also different speakers.

5.2.2 Denoiser

Here we expose the main motivations behind our choice of addressing the device identification problem in noisy conditions. We also explain the reasons that led us to address the denoising problem in the time-frequency domain rather than on the raw audio waveform.

Motivation We know that our mobile-phone device identification problem is based on a blind channel estimation procedure. Unfortunately, this process has weaknesses when using signals corrupted by various kinds of perturbation, such as reverb, music, background sounds and noise.

In this work we chose to address the presence of the additive noise, in specific by considering the AWGN case, trying to limit its negative impact to our task. To confirm this problem, several device identification tests were done using AWGN-corrupted data. The accuracy of our baseline algorithm dropped to 60.0% from the original 96.2% as we can see from Table 5.3. In addition, we implemented Band Energy Difference (BED) [10], which is a very popular feature used to perform device identification. Even in this case, the identification accuracy dropped down from 98.5% to 60.1%.

At first, we tried to solve this problem by using a multi-scene data-augmentation approach on training phase. We fed our system with half of audio recording corrupted by noise and we left the remaining unchanged. Unfortunately, this approach was ineffective, while at the same time leading us to a low accuracy value of 47.6%.

Our final idea fell on the adoption of an "intelligent" denoising process. The adjective "intelligent" refers to the ability of suppressing as much as possible the additive noise contribution, leaving the intrinsic device fingerprint nearly unchanged.

From Audio Waveform Initially, we tried to tackle the denoising task working directly in the waveform domain, and used an ANN developed by Facebook AI Research [71] that is based on an encoder-decoder architecture with skip-connections. This network was an adaptation for a causal speech enhancement problem of the DEMUCS architecture [72], initially developed for music source separation. Even if this architecture was capable of removing various kinds of background noise including stationary and non-stationary ones, it was not really suitable for our specific denoising task. After some tests indeed, even with low denoising power settings, the denoiser turned out to take away much of the intrinsic information that makes up the device fingerprint. The best accuracy achieved was 35.2% forcing us to find an alternative solution to solve this problem.

From Spectrogram After some research, we decided to perform a denoising process using a time-frequency representation of audio data. Thus, we represented the audio as a two-dimensional image as explained in Section 2.1.1.1, and performed denoising on it. By doing so, we took great advantage of the numerous architectures present in the literature facing the image denoising problem.

Among the many architectures available, we decided to use an architecture known as Denoising CNN (DnCNN) [60], designed to tackle the AWGN contribution on images, but is also widely used for superresolution and JPEG image deblocking problems. In order to increase the denoising performance, the authors decided to use batch normalization [73] and a *residual learning* [74] approach with only one residual unit. Rather than directly outputting the denoised image, the net has been designed to predict the residual image, i.e. the noisy component. Then, the denoised image has been obtained by subtracting this residual from the noisy image, as seen in Figure 4.5.

The motivation behind the adoption of residual learning strategies on image denoising relies on the assumption that the residual mapping is much easier to be learned than the original unreferenced mapping [60]. With such a residual learning strategy, extremely deep CNN can be easily trained to improve the overall accuracy [74].

The use of a CNN-based architecture on a time-frequency representation is justified by the fact that the convolution operation in time makes the model equivariant to shifts in time. Using convolution across the frequency axis instead makes the model equivariant to frequency [30]. These properties reflects the AWGN spectral characteristics.

On image processing, the architecture achieved a Peak SNR Ratio (PSNR) of $29,02$ and a Structural Similarity Index (SSIM) of $0,8190$ for a Gaussian denoising task on the BSD68 [75] dataset with an SNR of 25dB [60]. The effectiveness of the network is affected by the kernel size and the model depth: high noise level usually requires a larger effective patch size to capture more context information for restoration. In this work, we chose to use a kernel size of 3×3 with 15 stacked convolutional hidden layers, replicating the structure used in [60] for Gaussian denoising.

5.3 Results

This section is entirely devoted to illustrate the final identification performance of our method, starting from the baseline adopted to finally analyze the final configuration. To validate our design process, different ablation studies are carried out.

5.3.1 Baseline

We adopted the algorithm proposed by Cuccovillo et al. in [11, 12] as a baseline for our device identification task. In this implementation, we used the pre-emphasis filtering neither for the train audio of the GMM nor for the test recordings, as not mandatory in the implementation specifications. As in the original work, the TIMIT dataset [65] was used to train the GMM clean speech estimator.

Table 5.3: The identification performance of our baseline in [11, 12].

Train Data	**Test Data**	**Accuracy**
MOBIPHONE	MOBIPHONE	0.96195
	MOBIPHONE$_{\text{awgn}}$	0.60000

Despite the good performance obtained in the clean case, from Table 5.3 we can see how badly the noise disturbance contributed to the final outcome, giving an accuracy in identification of 0.60000. For this reason, the following sections are entirely devoted to demonstrate the proposed approach adopted to deal with this signal perturbation.

5.3.2 Ablation Studies

In this section, we validate our proposed methodology following a bottom-up approach. We are going to add the main blocks that form our system one by one, to prove that they are all necessary.

Table 5.4: Summary of the various settings in our ablation studies.

	Speech Removal	Denoising (DnCNN)	Feature Computation
AS Experiment I	-	-	-
AS Experiment II	-	✓	-
AS Experiment III	✓	-	-
AS Experiment IV	✓	✓	-
Proposed Method	✓	✓	✓

For better comprehension, in Table 5.4 we created a grid showing the different settings setup used during the experiments. In doing this, we advise the reader to refer to them as we go on the experiments.

All the architecture flowcharts and the identification reports related to each experiment are available in Appendix A and Appendix B respectively.

5.3.2.1 AS Experiment I – Averaging the Spectrum

For this first experiment, we performed device identification by making a time-frequency representation of the signal and then computing the channel response directly from the raw data. Indeed, we left the recording untouched, without performing neither denoising nor speech removal. The corresponding reduced system architecture can be found in the appendix, in Figure A.1.

This is the simplest setup we used, since only the time averaging block was acting, being the only thing required for the computation of the channel response from a spectrogram. Indeed, we did not use the custom features for the final identification but the estimated channel response directly.

As a side note, we can consider this procedure equivalent to computing the Long-Term Average Spectrum (LTAS), which is one of the naive approaches for excluding the hypothesis that two recordings were acquired by the same device. In addition, the Band Energy Difference (BED) [10] feature can be seen as an approximation of the derivative of the channel obtained in the same way as the experiment under consideration.

In Figure 5.2 the confusion matrix of this experiment is shown. We can clearly see the poor performance obtained adopting this workflow for the noisy speech audio input, having an accuracy of 0.09244. From here, we notice how the noise confused the identification process, making the final prediction fall on only two mobile-phone models. Also the clean audio input case was addressed, giving 0.26611 of accuracy and confirming the bad design of such a system.

The related identification report is also provided in the appendix for completeness, in Table B.1.

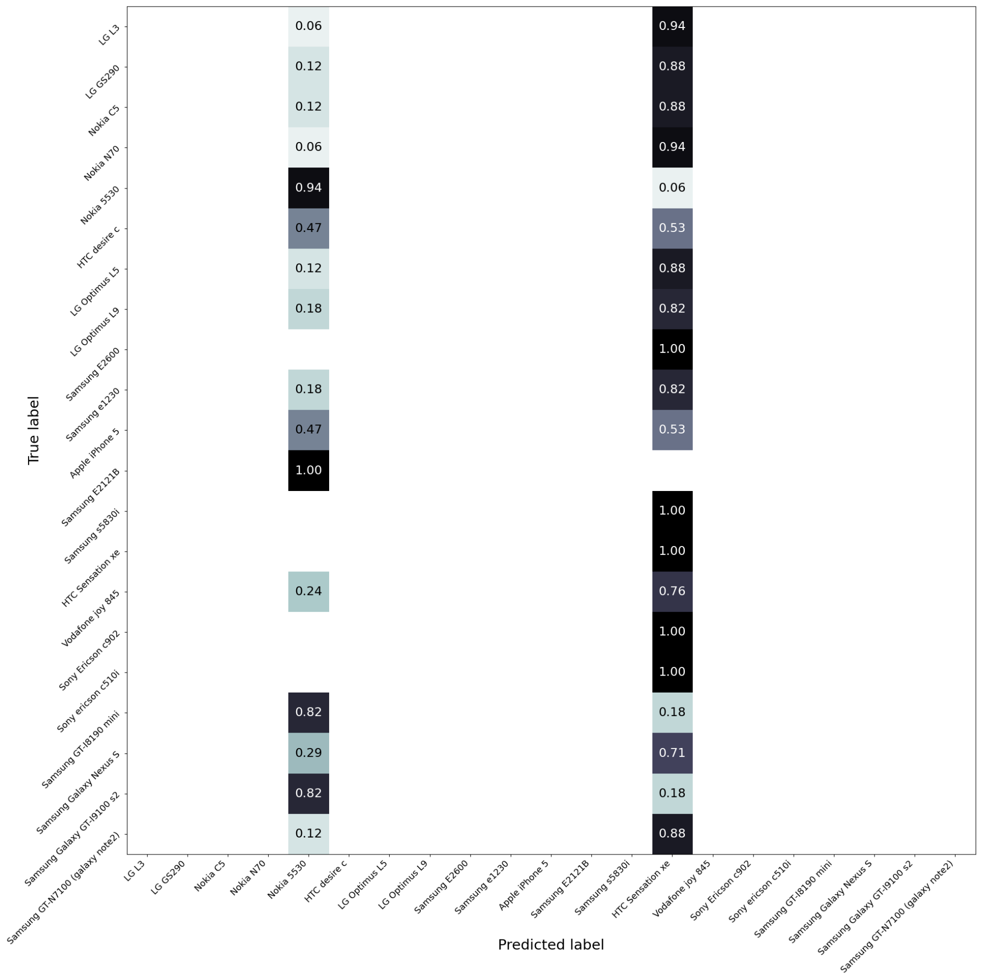


Figure 5.2: AS Experiment I confusion matrix - accuracy 0.09244.

5.3.2.2 AS Experiment II – Averaging of the Denoised Spectrum

In this case, we introduced the DnCNN denoiser with the aim to improve the performance when using the noisy recordings in the MOBIPHONE$_{\text{awgn}}$ dataset. Before retrieving the final channel response we performed a signal processing step in order to clean the recording, i.e. its time-frequency representation. Therefore, the log-magnitude spectrogram was denoised by the DnCNN and the time averaging block did the rest by computing the final channel response as before. Even in this case we used the raw estimated channel to perform the final identification. The corresponding reduced system architecture can be found in the appendix, in Figure A.2.

Comparing the confusion matrix in Figure 5.3 with the one in the previous experiment, we can see how the signal process tried to clean up the signal, with the aim of bringing back useful information for identification,

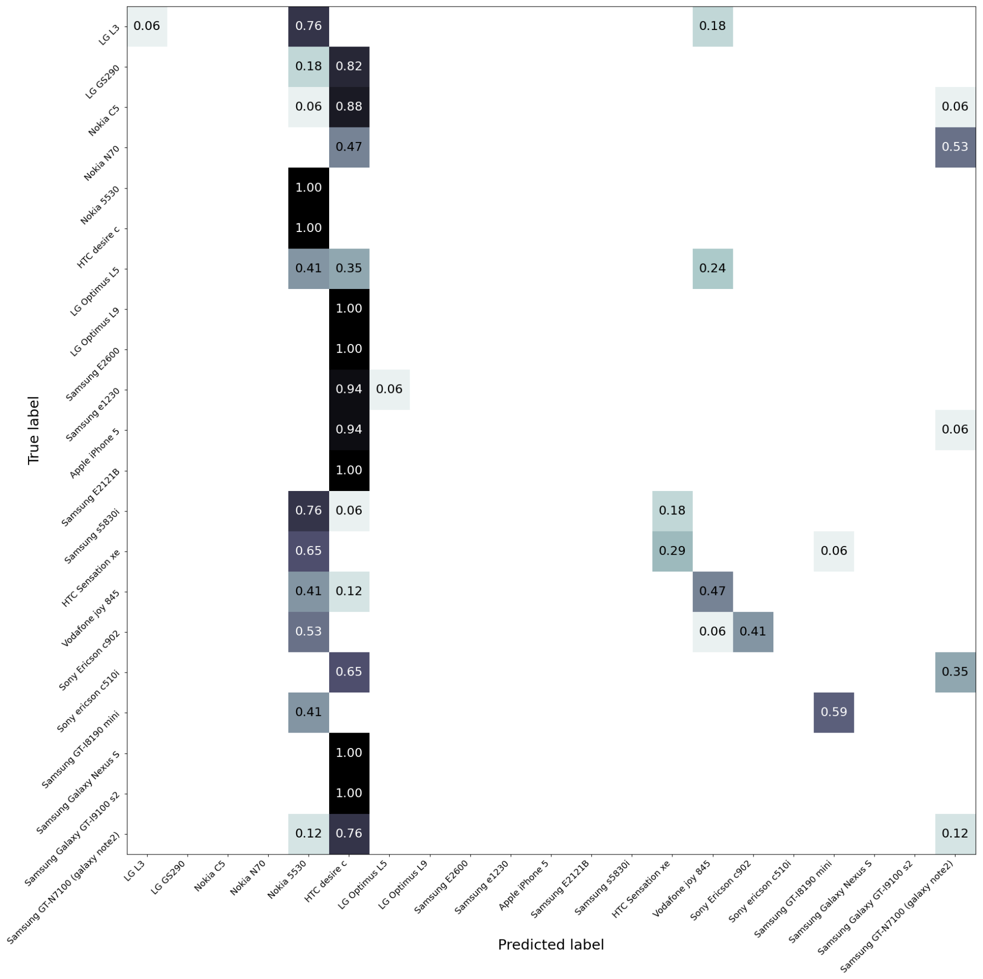


Figure 5.3: AS Experiment II confusion matrix - accuracy 0.14006.

but with still poor results.

The related identification report is also provided in the appendix for completeness, in Table B.2.

5.3.2.3 AS Experiment III – Blind Channel Estimation

The previous method seemed to be ill-posed as it focused on the problem of the noise component while it did not address the speech component, which contributes negatively to our channel estimation.

For this reason, in this experiment we wanted to test the impact of the speech removal procedure. It is important to notice that this procedure is based on a previous speech estimation performed by the pre-trained GMM architecture, responsible for modeling a generic clean speech corpus. The corresponding reduced system architecture can be found in the appendix, in Figure A.3.

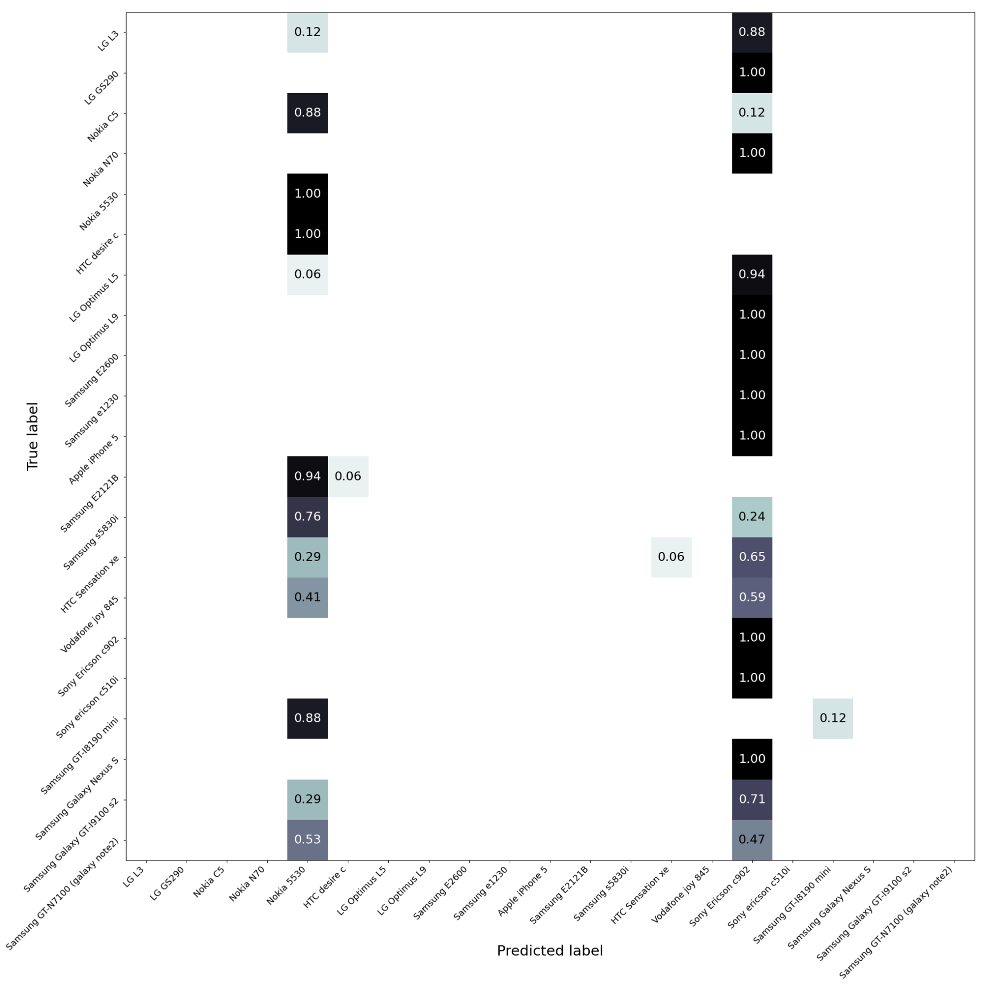


Figure 5.4: AS Experiment III confusion matrix - accuracy 0.10364.

By testing this configuration on clean data contained in MOBIPHONE, we obtain a final identification accuracy of 0.94118. If we compare this with the one obtained in AS Experiment I using the same test data, we can notice that the adoption of the GMM along with the final speech spectral subtraction turned out to be a good way to address this problem, achieving an improvement of 0.67507. Therefore, we can confirm that the speech estimation and removal procedure is the right way to follow.

Unfortunately, the confusion matrix in Figure 5.4 illustrates the limit of this configuration in addressing the problem when tested with noisy speech recordings. Indeed, using the MOBIPHONE$_{\text{awgn}}$ recordings, we obtained an accuracy of 0.10364, similar to the one achieved in AS Experiment I, thus without performing the spectral subtraction.

The related identification reports are also provided in the appendix for completeness, in Table B.3.1 and in Table B.3.2, respectively.

5.3.2.4 AS Experiment IV – Blind Channel Estimation from Denoised Spectrum

Driven by the previous bad result on noisy data, we did a further step by adding a denoising stage to the AS Experiment III configuration. Therefore, we expected a clear improvement in testing with noisy speech audio recordings. Indeed, the denoising process should mitigate the negative contribution of noise, at the same time the speech removal should limit the one of the speech. The corresponding reduced system architecture can be found in the appendix, in Figure A.4.

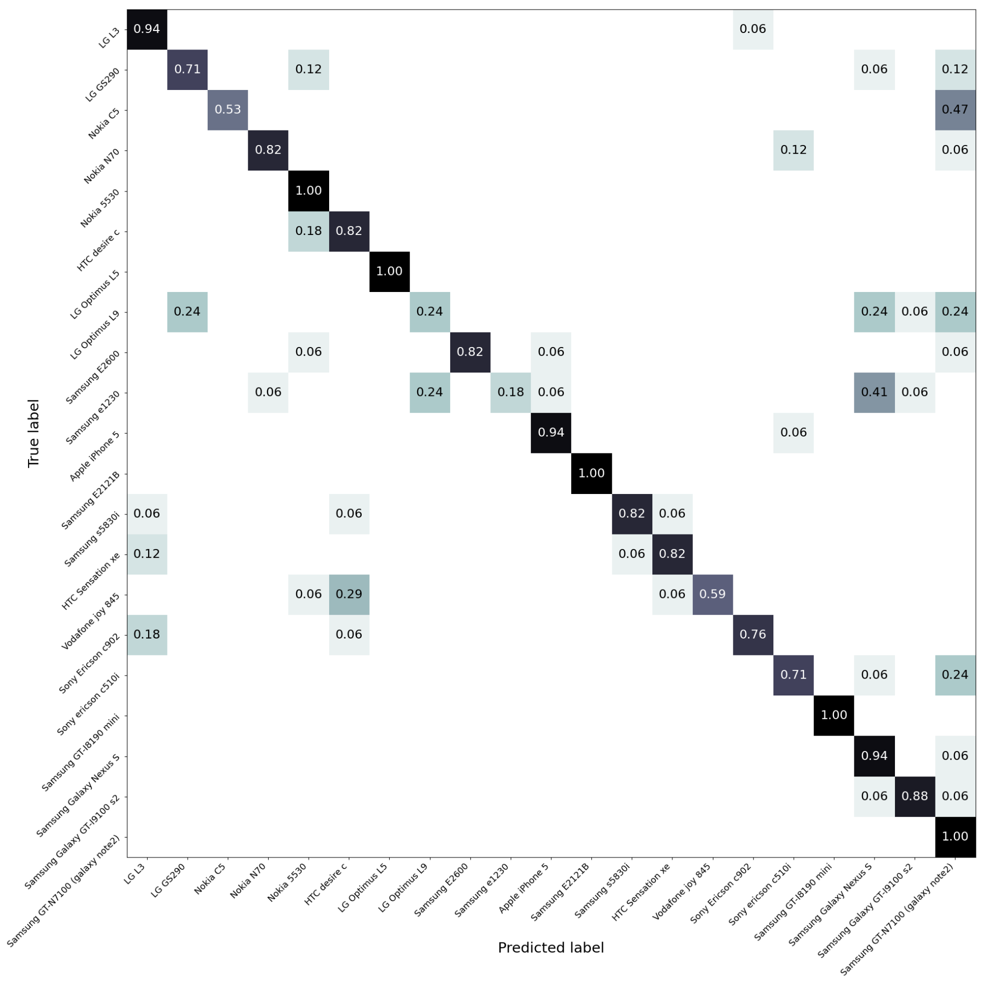


Figure 5.5: AS Experiment IV confusion matrix - accuracy 0.78711.

As we can see from the confusion matrix in Figure 5.5, the improvement after a denoising stage was quite effective, passing from an accuracy of 0.10364 - AS Experiment III with noisy data - to 0.78711. Thus, the benefit of the DnCNN was noticeable.

The related identification report is also provided in the appendix for completeness, in Table B.4.

5.3.3 Proposed Method – Advanced Custom Feature from the Blind Channel Estimation of the Denoised Spectrum

As final step, we introduced the custom feature rather than the raw estimated channel response. The channel response used until now allowed us to understand the main trend of the design process. The adoption of the custom feature computed in Section 4.2.4 confirmed the results and enhanced the informative power of the channel extraction process. The complete flowchart in Figure 4.2 shows all the steps of the algorithm.

Performing the identification task with this feature led to an improvement of 0.01961 on the overall accuracy from AS Experiment IV. Indeed, we obtain a final device identification accuracy of 0.80672, as we can see from the confusion matrix in Figure 5.6.

Nevertheless, we would like to point out that both the adopted features were well-posed since they resulted in a fixed-length property to represent variable-length speech recordings. This is a desirable characteristic since in principle the intrinsic fingerprint of a device should be independent from the amount of acquired data.

The related identification report is also provided in the appendix for completeness, in Table B.5.

From this final result, we can see that there was still a margin of improvement. The reasons behind this behavior are explained in the following Section 5.3.4.

5.3.4 Model Robustness

With the next experiments we quantify the robustness and the generalization performance of our proposed method.

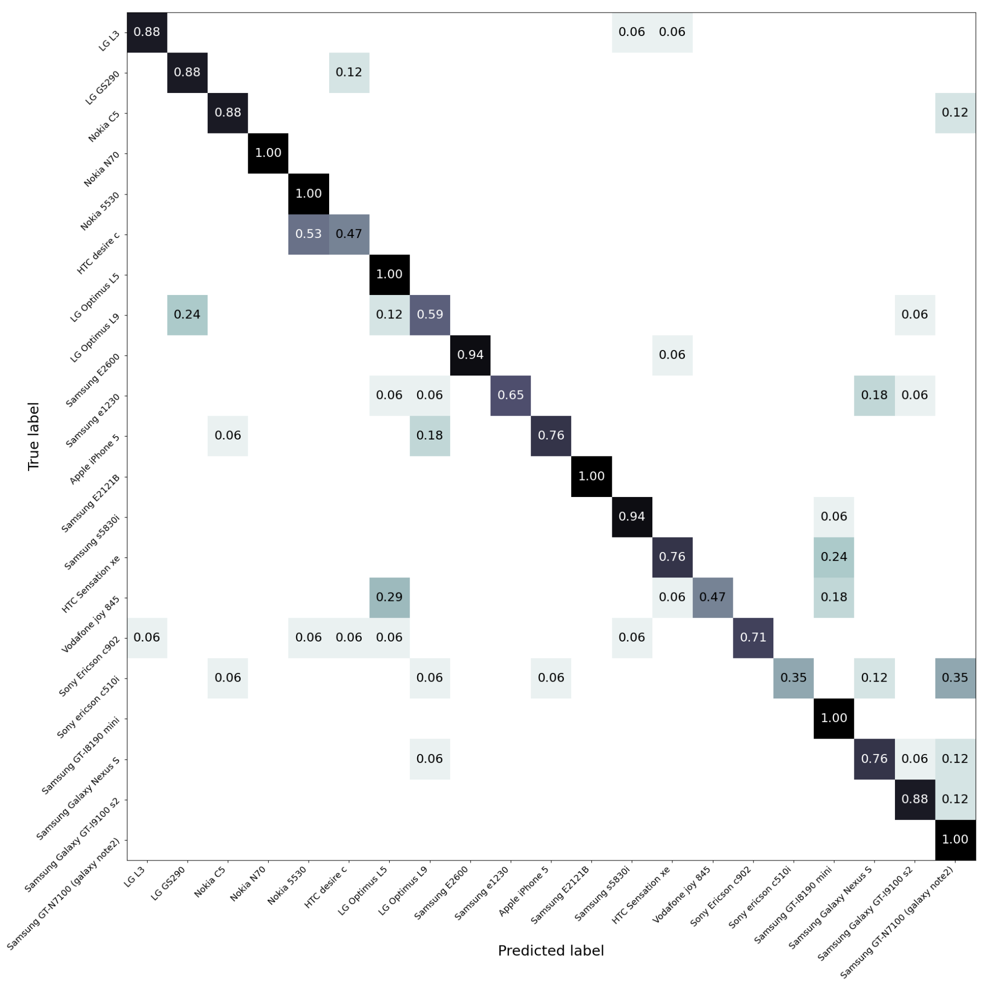


Figure 5.6: Proposed Method confusion matrix - accuracy 0.80672.

5.3.4.1 MR Experiment I – Different SNR

The goal of the first robustness experiment was to point out the DnCNN denoising performance on spectrograms that have an unknown SNR.

For this purpose, we tested the model with clean recordings, such as the ones contained on the MOBIPHONE dataset. This implied that the DnCNN had to perform a denoising process with a really high SNR, even if it was trained to address a 30dB SNR.

From the confusion matrix in Figure 5.7 the result may look bad, but this was actually expected. Indeed, the DnCNN was *not* capable of tackle a perfect *blind Gaussian denoising task*[3] and works fine only within the training SNR.

The related identification report is also provided in the appendix for

[3]a denoising process where the SNR is unknown.

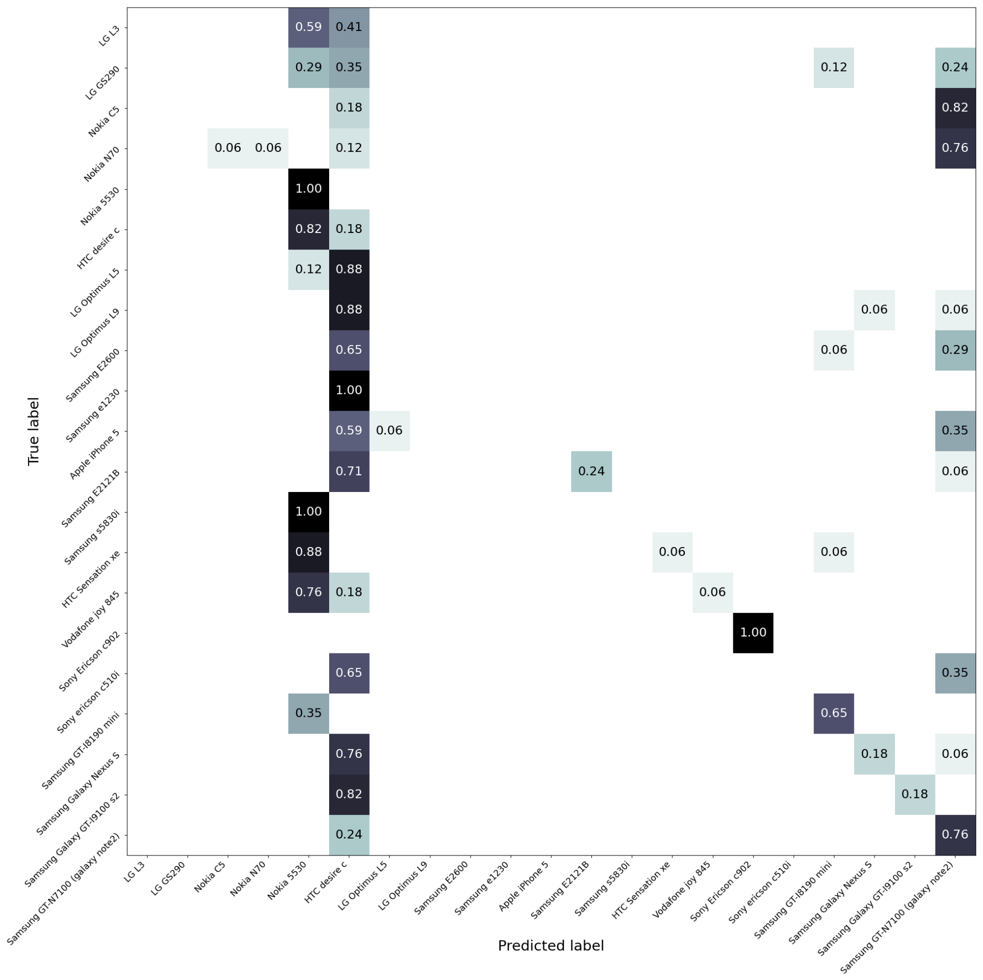


Figure 5.7: MR Experiment I confusion matrix - accuracy 0.20728.

completeness, in Table B.6.

In Section 6 we propose an alternative solution to limit this problem and preserve the intrinsic information of the device.

5.3.4.2 MR Experiment II – Residual Information

We have seen that the DnCNN performs badly if tested with SNR substantially different from the training one, as it is much more aggressive in removing the information we use as a fingerprint of the device.

In this experiment, we tried to quantify how much of this intrinsic information was picked up by the denoiser in making the denoising process on different SNR.

The removed information can be found by analyzing the residual signal. As we know, the residual is the estimation objective of the DnCNN. Since the net aims to estimate the noise corruption, this residual signal should correspond to AWGN noise, thus a signal that has a uniform power across the frequency band. We expect this signal to be identical for all the analyzed devices, since the noise corruption applied was the same for all the recordings in the MOBIPHONE$_{\text{awgn}}$. As a consequence, if we estimate the channel response from this signal by performing a time averaging on its spectrogram, the algorithm is supposed to give bad result in the identification task.

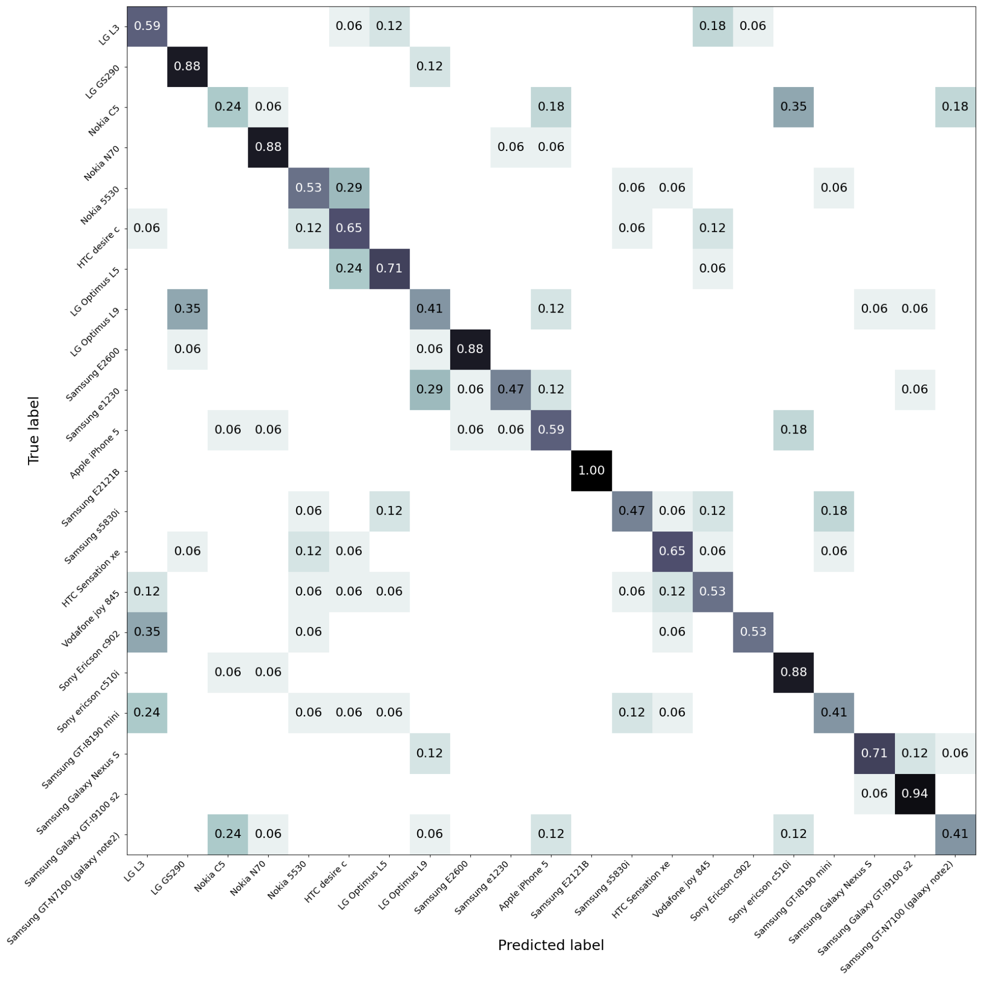


Figure 5.8: MR Experiment II confusion matrix - accuracy 0.63585.

Unfortunately, as we can see from the confusion matrix in Figure 5.8, the denoising process for a substantially different SNR also took away part of the device fingerprint. This behavior partially justifies the margin of improvement that we had in the previous robustness experiment as well as on our proposed method in Section 5.3.3.

The related identification report is also provided in the appendix for completeness, in Table B.7.

5.4 Conclusive Remarks

In this chapter, we evaluated the proposed methodology through simulations and experiments. We introduced the datasets involved and the evaluation metrics used in this work. After that, we focused on the preliminary studies that affected our method design. A detailed analysis of the final system was done following a bottom-up approach, validating its performance through some ablation studies. The last part of this chapter was entirely devoted to the robustness aspect and the generalization ability of the final algorithm.

6 Conclusions and Outlook

In this work, we proposed an approach to solve the device identification problem using speech recordings corrupted by additive noise.

The proposed method is based on the extraction of a device fingerprint. A feature computed from an estimated device channel response was chosen for this purpose. Since the clean speech signal was unavailable, a blind channel estimation was performed by creating a reliable estimator of a generic speech signal. Using a GMM architecture trained on a RASTA filtered HTK-MFCCs from clean speech frames, we were able to create an efficient representation of the speech component. The only objective of this procedure was the removal of unnecessary information, leaving only the parts of the signal that characterize the device allowing it to be unequivocally distinguished.

At first, we processed the real recording by getting rid of the negative noise contribution. We performed this task in the time-frequency domain using a residual denoiser known as DnCNN adapting it for this task. Then, the GMM architecture allowed us to perform a speech removal on real data also working in the time-frequency domain.

The algorithm was tested using noise corrupted speech recordings from the MOBIPHONE dataset, lowering the original SNR.

The final algorithm was tested with a different speech corpus used in the training phase of the GMM, thus solving the problem in a speaker-independent perspective.

In addition, the algorithm involved a built-in pre-emphasis filtering for all audio data. This signal processing turned out to be very beneficial in the clean speech estimation procedure.

For what concerns experiments, the denoiser addition led us to an accuracy improvement from 0.10 to 0.78 on the identification task. Furthermore, adopting custom features, the accuracy reached 0.81.

After having performed some detailed ablation studies on the model configuration, we concluded that the proposed approach seemed to be the most appropriate to tackle this problem, even though an ample margin of improvement exists.

Based on the results discussed throughout this work, future research on this topic should focus on different aspects.

Generalization An extensive study to evaluate the influence of completely different training data in modeling clean speech is required. Specifically, it is of significant interest to investigate how and if the different languages or speakers' age impact the performance of the algorithm, thus studying its resiliency.

There is also the need for a broader and updated dataset that contains recordings from recent mobile-phones devices. This will facilitate the development of more flexible algorithms in this field.

To achieve robustness, particular attention should be paid at compression, reverberation, different kinds of noise and the minimum recording duration, in order to state the usage limits of the method on real data.

Since our DnCNN fails in denoising signals that have a SNR substantially different from the one used for training, the proposed method could be adjusted as follow. In the modified method, multiple versions of the DnCNN denoiser must be trained, each one with data having different SNR. A preliminary SNR estimator should be placed before the entire workflow, as it will serve for selecting the pre-trained DnCNN model that is more suitable for the denoising process. This could help us creating a model characterized by a good generalization power for recordings with different SNR.

Challenging scenarios Since the effectiveness of voluntary manipulation goes in parallel with technological progress, a deep study should be carried out to cope with possible anti-forensic attacks. An example of an attack could be done using the recent method proposed by Borsos et al. in [76]. They proposed an algorithm to perform a microphone style swapping with the aim to transform audio recorded with a microphone as if it was recorded with another, given only a few seconds of audio from the latter. It is clear how this could affect the performance of our algorithm.

Different approaches The microphone-based fingerprinting techniques can be integrated with other approaches, such as - for instance - Sensor Fusion [77, 78].

Another possibility to widen the model could consists in exploiting other ranges of signal's frequency spectrum, i.e. ultrasound response, and combing them with the presented method, which is mainly based on voice.

Appendix

A AS Experiment Flowcharts

A.1 AS Experiment I

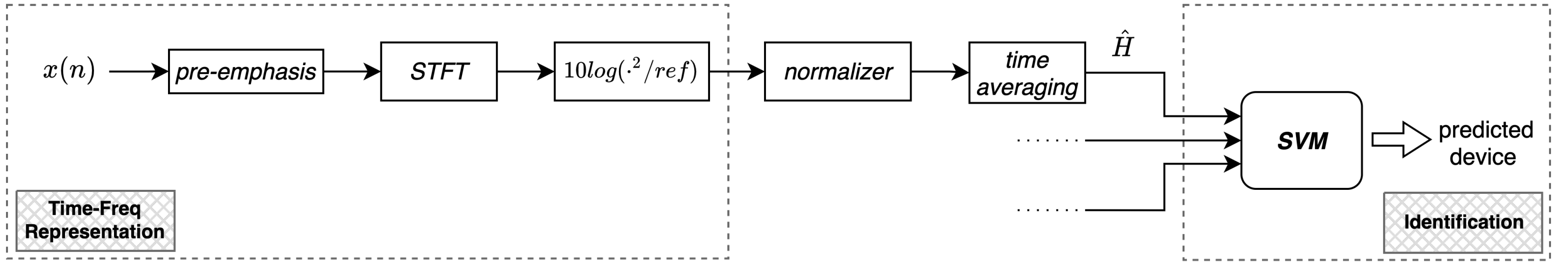


Figure 1: AS Experiment I flowchart.

A.2 AS Experiment II

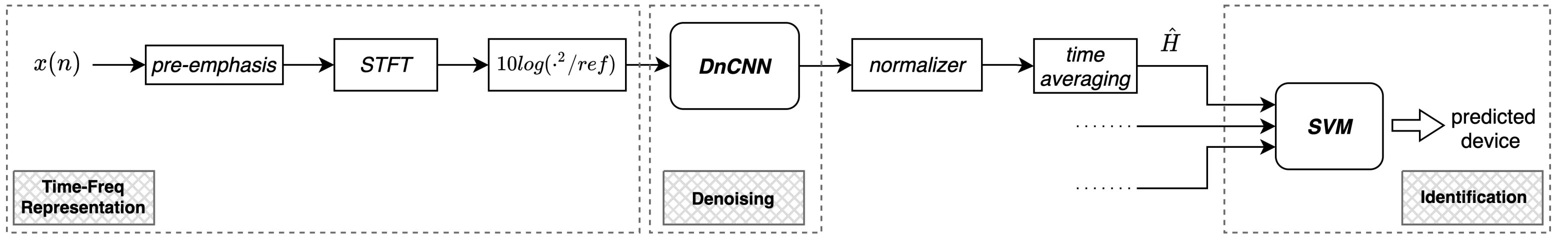


Figure 2: AS Experiment II flowchart.

A.3 AS Experiment III

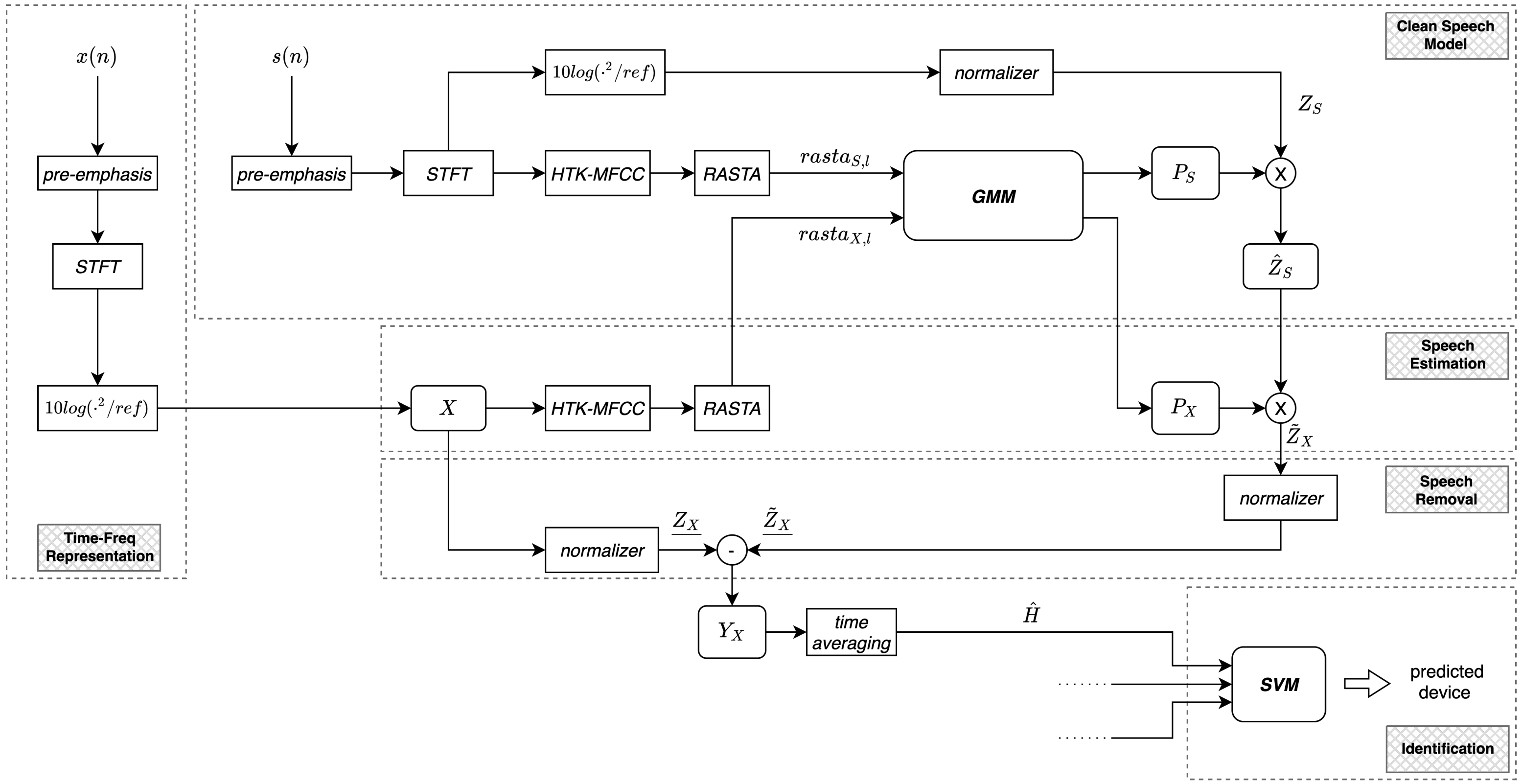


Figure 3: AS Experiment III flowchart.

A.4 AS Experiment IV

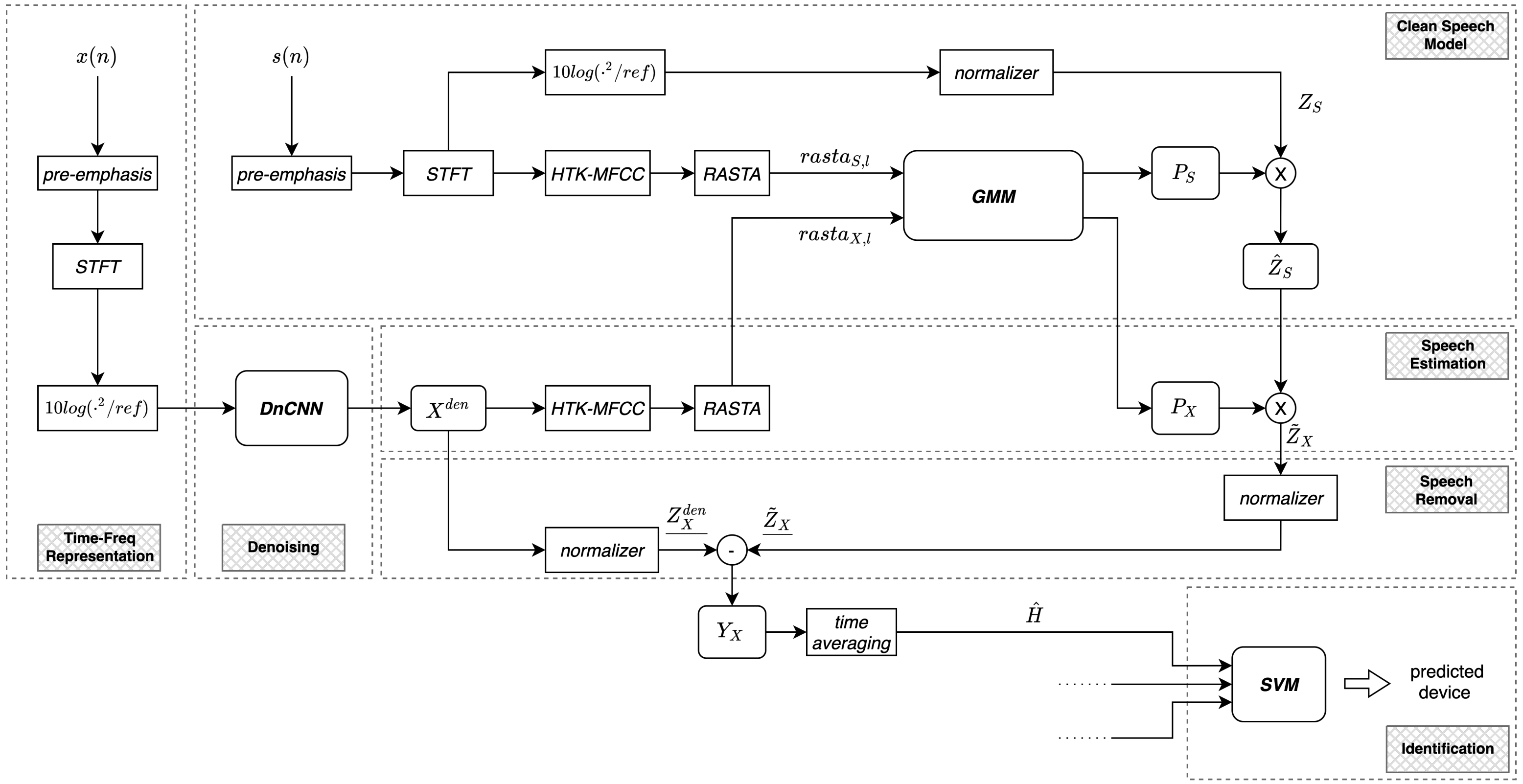


Figure 4: AS Experiment IV flowchart.

B Identification Reports

B.1 AS Experiment I

Table 1: AS Experiment I report.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	0.0	0.0	0.0	17.0
LG GS290	0.0	0.0	0.0	17.0
Nokia C5	0.0	0.0	0.0	17.0
Nokia N70	0.0	0.0	0.0	17.0
Nokia 5530	0.15686	0.94118	0.26891	17.0
HTC desire c	0.0	0.0	0.0	17.0
LG Optimus L5	0.0	0.0	0.0	17.0
LG Optimus L9	0.0	0.0	0.0	17.0
Samsung E2600	0.0	0.0	0.0	17.0
Samsung e1230	0.0	0.0	0.0	17.0
Apple iPhone 5	0.0	0.0	0.0	17.0
Samsung E2121B	0.0	0.0	0.0	17.0
Samsung s5830i	0.0	0.0	0.0	17.0
HTC Sensation xe	0.06667	1.00000	0.12500	17.0
Vodafone joy 845	0.0	0.0	0.0	17.0
Sony Ericsson c902	0.0	0.0	0.0	17.0
Sony Ericsson c510i	0.0	0.0	0.0	17.0
Samsung GT-I8190 mini	0.0	0.0	0.0	17.0
Samsung Galaxy Nexus S	0.0	0.0	0.0	17.0
Samsung Galaxy GT-I9100 s2	0.0	0.0	0.0	17.0
Samsung GT-N7100	0.0	0.0	0.0	17.0
accuracy	0.09244	0.09244	0.09244	0.09244
macro avg	0.01064	0.09244	0.01876	357.0
weighted avg	0.01064	0.09244	0.01876	357.0

B.2 AS Experiment II

Table 2: AS Experiment II report.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	1.00000	0.05882	0.11111	17.0
LG GS290	0.0	0.0	0.0	17.0
Nokia C5	0.0	0.0	0.0	17.0
Nokia N70	0.0	0.0	0.0	17.0
Nokia 5530	0.15888	1.00000	0.27419	17.0
HTC desire c	0.0	0.0	0.0	17.0
LG Optimus L5	0.0	0.0	0.0	17.0
LG Optimus L9	0.0	0.0	0.0	17.0
Samsung E2600	0.0	0.0	0.0	17.0
Samsung e1230	0.0	0.0	0.0	17.0
Apple iPhone 5	0.0	0.0	0.0	17.0
Samsung E2121B	0.0	0.0	0.0	17.0
Samsung s5830i	0.0	0.0	0.0	17.0
HTC Sensation xe	0.62500	0.29412	0.40000	17.0
Vodafone joy 845	0.50000	0.47059	0.48485	17.0
Sony Ericsson c902	1.00000	0.41176	0.58333	17.0
Sony Ericsson c510i	0.0	0.0	0.0	17.0
Samsung GT-I8190 mini	0.90909	0.58823	0.71429	17.0
Samsung Galaxy Nexus S	0.0	0.0	0.0	17.0
Samsung Galaxy GT-I9100 s2	0.0	0.0	0.0	17.0
Samsung GT-N7100	0.10526	0.11765	0.11111	17.0
accuracy	0.14006	0.14006	0.14006	0.14006
macro avg	0.20468	0.14006	0.12756	357.0
weighted avg	0.20468	0.14006	0.12756	357.0

B.3 AS Experiment III

B.3.1 Clean Data

Table 3: AS Experiment III report - clean data.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	1.00000	1.00000	1.00000	17.0
LG GS290	0.89474	1.00000	0.94444	17.0
Nokia C5	0.9375	0.88235	0.90909	17.0
Nokia N70	0.93333	0.82353	0.87410	17.0
Nokia 5530	0.88889	0.94118	0.91428	17.0
HTC desire c	0.93750	0.88235	0.90909	17.0
LG Optimus L5	1.00000	1.00000	1.00000	17.0
LG Optimus L9	0.86667	0.76470	0.81250	17.0
Samsung E2600	1.00000	1.00000	1.00000	17.0
Samsung e1230	0.92308	0.70588	0.80000	17.0
Apple iPhone 5	0.94444	1.00000	0.97143	17.0
Samsung E2121B	1.00000	1.00000	1.00000	17.0
Samsung s5830i	1.00000	1.00000	1.00000	17.0
HTC Sensation xe	1.00000	1.00000	1.00000	17.0
Vodafone joy 845	1.00000	1.00000	1.00000	17.0
Sony Ericsson c902	1.00000	1.00000	1.00000	17.0
Sony Ericsson c510i	0.83333	0.88235	0.85714	17.0
Samsung GT-I8190 mini	1.00000	1.00000	1.00000	17.0
Samsung Galaxy Nexus S	0.83333	0.88235	0.85714	17.0
Samsung Galaxy GT-I9100 s2	0.89474	1.00000	0.94444	17.0
Samsung GT-N7100	0.89474	1.00000	0.94444	17.0
accuracy	0.94118	0.94118	0.94118	0.94118
macro avg	0.94201	0.94118	0.93995	357.0
weighted avg	0.94201	0.94118	0.93995	357.0

B.3.2 Noisy Data

Table 4: AS Experiment III report - noisy data.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	0.0	0.0	0.0	17.0
LG GS290	0.0	0.0	0.0	17.0
Nokia C5	0.0	0.0	0.0	17.0
Nokia N70	0.0	0.0	0.0	17.0
Nokia 5530	0.13934	1.00000	0.24460	17.0
HTC desire c	0.0	0.0	0.0	17.0
LG Optimus L5	0.0	0.0	0.0	17.0
LG Optimus L9	0.0	0.0	0.0	17.0
Samsung E2600	0.0	0.0	0.0	17.0
Samsung e1230	0.0	0.0	0.0	17.0
Apple iPhone 5	0.0	0.0	0.0	17.0
Samsung E2121B	0.0	0.0	0.0	17.0
Samsung s5830i	0.0	0.0	0.0	17.0
HTC Sensation xe	1.00000	0.05882	0.11111	17.0
Vodafone joy 845	0.0	0.0	0.0	17.0
Sony Ericsson c902	0.07359	1.00000	0.13710	17.0
Sony Ericsson c510i	0.0	0.0	0.0	17.0
Samsung GT-I8190 mini	1.00000	0.11765	0.21053	17.0
Samsung Galaxy Nexus S	0.0	0.0	0.0	17.0
Samsung Galaxy GT-I9100 s2	0.0	0.0	0.0	17.0
Samsung GT-N7100	0.0	0.0	0.0	17.0
accuracy	0.10364	0.10364	0.10364	0.10364
macro avg	0.10538	0.10364	0.03349	357.0
weighted avg	0.10538	0.10364	0.03349	357.0

B.4 AS Experiment IV

Table 5: AS Experiment IV report.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	0.72727	0.94118	0.82051	17.0
LG GS290	0.75000	0.70588	0.72727	17.0
Nokia C5	1.00000	0.52941	0.69231	17.0
Nokia N70	0.93333	0.82353	0.87410	17.0
Nokia 5530	0.70833	1.00000	0.82927	17.0
HTC desire c	0.66667	0.82352	0.73684	17.0
LG Optimus L5	1.00000	1.00000	1.00000	17.0
LG Optimus L9	0.50000	0.23529	0.31200	17.0
Samsung E2600	1.00000	0.82353	0.90322	17.0
Samsung e1230	1.00000	0.17647	0.30000	17.0
Apple iPhone 5	0.88889	0.94118	0.91428	17.0
Samsung E2121B	1.00000	1.00000	1.00000	17.0
Samsung s5830i	0.93333	0.82353	0.87410	17.0
HTC Sensation xe	0.87500	0.82353	0.84848	17.0
Vodafone joy 845	1.00000	0.58823	0.74074	17.0
Sony Ericsson c902	0.92857	0.76470	0.83871	17.0
Sony Ericsson c510i	0.80000	0.70588	0.75000	17.0
Samsung GT-I8190 mini	1.00000	1.00000	1.00000	17.0
Samsung Galaxy Nexus S	0.53333	0.94118	0.68085	17.0
Samsung Galaxy GT-I9100 s2	0.88235	0.88235	0.88235	17.0
Samsung GT-N7100	0.43590	1.00000	0.60714	17.0
accuracy	0.78711	0.78711	0.78711	0.78711
macro avg	0.83633	0.78711	0.77819	357.0
weighted avg	0.83633	0.78711	0.77819	357.0

B.5 Proposed Method

Table 6: Proposed method report.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	0.93750	0.88235	0.90909	17.0
LG GS290	0.78947	0.88235	0.83333	17.0
Nokia C5	0.88235	0.88235	0.88235	17.0
Nokia N70	1.00000	1.00000	1.00000	17.0
Nokia 5530	0.62963	1.00000	0.77273	17.0
HTC desire c	0.72727	0.47059	0.57143	17.0
LG Optimus L5	0.65385	1.00000	0.79070	17.0
LG Optimus L9	0.62500	0.58823	0.60606	17.0
Samsung E2600	1.00000	0.94118	0.96970	17.0
Samsung e1230	1.00000	0.64706	0.78571	17.0
Apple iPhone 5	0.92857	0.76470	0.83871	17.0
Samsung E2121B	1.00000	1.00000	1.00000	17.0
Samsung s5830i	0.88889	0.94118	0.91428	17.0
HTC Sensation xe	0.81250	0.76470	0.78788	17.0
Vodafone joy 845	1.00000	0.47059	0.63100	17.0
Sony Ericsson c902	1.00000	0.70588	0.82759	17.0
Sony Ericsson c510i	1.00000	0.35294	0.52174	17.0
Samsung GT-I8190 mini	0.68000	1.00000	0.80952	17.0
Samsung Galaxy Nexus S	0.72222	0.76470	0.74286	17.0
Samsung Galaxy GT-I9100 s2	0.83333	0.88235	0.85714	17.0
Samsung GT-N7100	0.58621	1.00000	0.73913	17.0
accuracy	0.80672	0.80672	0.80672	0.80672
macro avg	0.84270	0.80672	0.71000	357.0
weighted avg	0.84270	0.80672	0.71000	357.0

B.6 MR Experiment I

Table 7: MR Experiment I report.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	0.0	0.0	0.0	17.0
LG GS290	0.0	0.0	0.0	17.0
Nokia C5	0.0	0.0	0.0	17.0
Nokia N70	1.00000	0.05882	0.11111	17.0
Nokia 5530	0.17172	1.00000	0.29310	17.0
HTC desire c	0.02055	0.17647	0.03681	17.0
LG Optimus L5	0.0	0.0	0.0	17.0
LG Optimus L9	0.0	0.0	0.0	17.0
Samsung E2600	0.0	0.0	0.0	17.0
Samsung e1230	0.0	0.0	0.0	17.0
Apple iPhone 5	0.0	0.0	0.0	17.0
Samsung E2121B	1.00000	0.23529	0.38095	17.0
Samsung s5830i	0.0	0.0	0.0	17.0
HTC Sensation xe	1.00000	0.05882	0.11111	17.0
Vodafone joy 845	1.00000	0.05882	0.11111	17.0
Sony Ericsson c902	1.00000	1.00000	1.00000	17.0
Sony Ericsson c510i	0.0	0.0	0.0	17.0
Samsung GT-I8190 mini	0.73333	0.64706	0.68750	17.0
Samsung Galaxy Nexus S	0.75000	0.17647	0.28571	17.0
Samsung Galaxy GT-I9100 s2	1.00000	0.17647	0.30000	17.0
Samsung GT-N7100	0.20312	0.76470	0.32099	17.0
accuracy	0.20728	0.20728	0.20728	0.20728
macro avg	0.37518	0.20728	0.17325	357.0
weighted avg	0.37518	0.20728	0.17325	357.0

B.7 MR Experiment II

Table 8: MR Experiment II report.

mobile-phone model	**precision**	**recall**	**f1-score**	**support**
LG L3	0.43478	0.58823	0.50000	17.0
LG GS290	0.65217	0.88235	0.75000	17.0
Nokia C5	0.40000	0.23529	0.29630	17.0
Nokia N70	0.78947	0.88235	0.83333	17.0
Nokia 5530	0.52941	0.52941	0.52941	17.0
HTC desire c	0.45833	0.64706	0.53658	17.0
LG Optimus L5	0.66667	0.70588	0.68571	17.0
LG Optimus L9	0.38889	0.41176	0.31000	17.0
Samsung E2600	0.88235	0.88235	0.88235	17.0
Samsung e1230	0.80000	0.47059	0.59259	17.0
Apple iPhone 5	0.50000	0.58823	0.54054	17.0
Samsung E2121B	1.00000	1.00000	1.00000	17.0
Samsung s5830i	0.61538	0.47059	0.53333	17.0
HTC Sensation xe	0.64706	0.64706	0.64706	17.0
Vodafone joy 845	0.50000	0.52941	0.51428	17.0
Sony Ericsson c902	0.90000	0.52941	0.66667	17.0
Sony Ericsson c510i	0.57692	0.88235	0.69767	17.0
Samsung GT-I8190 mini	0.58333	0.41176	0.48276	17.0
Samsung Galaxy Nexus S	0.85714	0.70588	0.77419	17.0
Samsung Galaxy GT-I9100 s2	0.80000	0.94118	0.86486	17.0
Samsung GT-N7100	0.63636	0.41176	0.50000	17.0
accuracy	0.63585	0.63585	0.63585	0.63585
macro avg	0.64849	0.63585	0.62989	357.0
weighted avg	0.64849	0.63585	0.62989	357.0

Bibliography

[1] V. Verma and N. Khanna, "Speaker-independent source cell-phone identification for re-compressed and noisy audio recordings," *Multimedia Tools and Applications*, Jan. 2021.

[2] X. Shen, X. Shao, Q. Ge, and L. Liu, "RARS: Recognition of Audio Recording Source Based on Residual Neural Network," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 575–584, 2021.

[3] M. A. Qamhan, H. Altaheri, A. H. Meftah, G. Muhammad, and Y. A. Alotaibi, "Digital Audio Forensics: Microphone and Environment Classification Using Deep Learning," *IEEE Access*, vol. 9, pp. 62719–62733, 2021.

[4] M. Zakariah, M. K. Khan, and H. Malik, "Digital multimedia audio forensics: past, present and future," *Multimed Tools Appl*, 2017.

[5] D. Seichter, L. Cuccovillo, and P. Aichroth, "AAC encoding detection and bitrate estimation using a convolutional neural network," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Shanghai), pp. 2069–2073, IEEE, Mar. 2016.

[6] D. Capoferri, C. Borrelli, P. Bestagini, F. Antonacci, A. Sarti, and S. Tubaro, "Speech Audio Splicing Detection and Localization Exploiting Reverberation Cues," in *2020 IEEE International Workshop on Information Forensics and Security (WIFS)*, (New York, NY, USA), pp. 1–6, IEEE, Dec. 2020.

[7] C. Borrelli, P. Bestagini, F. Antonacci, A. Sarti, and S. Tubaro, "Synthetic speech detection through short-term and long-term

prediction traces," *EURASIP Journal on Information Security*, vol. 2021, p. 2, Apr. 2021.

[8] A. K. Singh and P. Singh, "Detection of AI-Synthesized Speech Using Cepstral & Bispectral Statistics," *arXiv:2009.01934 [cs, eess, stat]*, Apr. 2021. arXiv: 2009.01934.

[9] M. Maksimovi, P. Aichroth, and L. Cuccovillo, "Detection and localization of partial audio matches in various application scenarios," *Multimedia Tools and Applications*, vol. 80, pp. 22619–22641, June 2021.

[10] D. Luo, P. Korus, and J. Huang, "Band Energy Difference for Source Attribution in Audio Forensics," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 9, p. 11, 2018.

[11] L. Cuccovillo, "Classification of Microphones of Mobile Devices via Blind Channel Estimation," Master's thesis, Politecnico di Milano, Milan, 2011.

[12] L. Cuccovillo, S. Mann, M. Tagliasacchi, and P. Aichroth, "Audio tampering detection via microphone classification," in *2013 IEEE 15th International Workshop on Multimedia Signal Processing (MMSP)*, (Pula (CA), Italy), pp. 177–182, IEEE, Sept. 2013.

[13] J. O. Smith, *Spectral Audio Signal Processing.* `http://-ccrma.stanford.edu/~jos/sasp/`, 2011. online book, 2011 edition.

[14] P. S. R. Diniz, E. A. B. da Silva, and S. L. Netto, *Digital Signal Processing: System Analysis and Design.* Cambridge University Press, 2 ed., 2010.

[15] M. Müller, *Fundamentals of Music Processing.* Springer International Publishing, 2015.

[16] P. Knees and M. Schedl, *Music Similarity and Retrieval*, vol. 36 of *The Information Retrieval Series.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[17] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, "The htk book," *Cambridge University Engineering Department*, vol. 3, 2002.

[18] "Cepstrum and MFCC - Introduction to Speech Processing - Aalto University Wiki." `https://wiki.aalto.fi/display/ITSP/Cepstrum+and+MFCCs`.

[19] V. Tyagi and C. Wellekens, "On desensitizing the mel-cepstrum to spurious spectral components for robust speech recognition," in *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 1, pp. I/529–I/532 Vol. 1, 2005.

[20] H. Hermansky and N. Morgan, "RASTA processing of speech," *IEEE Transactions on Speech and Audio Processing*, vol. 2, pp. 578–589, Oct. 1994.

[21] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd ed ed., 2010.

[22] C. M. Bishop, *Pattern recognition and machine learning*. Information science and statistics, New York: Springer, 2006.

[23] S. Raschka and V. Mirjalili, *Python Machine Learning, 3rd Ed.* Birmingham, UK: Packt Publishing, 3 ed., 2019.

[24] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. Adaptive computation and machine learning, Cambridge, Mass: MIT Press, 1998.

[25] T. M. Mitchell, *Machine Learning*. McGraw-Hill series in computer science, New York: McGraw-Hill, 1997.

[26] "Support Vector Machines - scikit-learn." `https://scikit-learn.org/stable/modules/svm.html#id15`.

[27] J. A. Hartigan, *Clustering algorithms*. John Wiley & Sons, Inc., 1975.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data Via the *EM* Algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, pp. 1–22, Sept. 1977.

[29] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," in *2017 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0588–0592, 2017.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[31] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM Comput. Surv.*, vol. 27, p. 326327, Sept. 1995.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[33] C. Hanilçi and F. Erta, "Recognition of Brand and Models of Cell-Phones From Recorded Speech Signals," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, p. 11, 2012.

[34] C. Hanilçi and F. Ertas, "Optimizing acoustic features for source cell-phone recognition using speech signals," in *Proceedings of the first ACM workshop on Information hiding and multimedia security - IH&MMSec '13*, (Montpellier, France), p. 141, ACM Press, 2013.

[35] "Deltas and delta-deltas - Introduction to Speech Processing - Aalto University Wiki." `https://wiki.aalto.fi/display/ITSP/Deltas+and+Delta-deltas`.

[36] C. Kotropoulos and S. Samaras, "Mobile phone identification using recorded speech signals," in *2014 19th International Conference on Digital Signal Processing*, (Hong Kong, Hong Kong), pp. 586–591, IEEE, Aug. 2014.

[37] Y. Jiang and F. H. F. Leung, "Source Microphone Recognition Aided by a Kernel-Based Projection Method," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 2875–2886, Nov. 2019.

[38] L. Zou, Q. He, and J. Wu, "Source cell phone verification from speech recordings using sparse representation," *Digital Signal Processing*, vol. 62, pp. 125–136, Mar. 2017.

[39] V. Verma, P. Khaturia, and N. Khanna, “Cell-Phone Identification from Recompressed Audio Recordings,” in *2018 Twenty Fourth National Conference on Communications (NCC)*, (Hyderabad), pp. 1–6, IEEE, Feb. 2018.

[40] X. Lin, J. Zhu, and D. Chen, “Subband Aware CNN for Cell-Phone Recognition,” *IEEE Signal Processing Letters*, vol. 27, pp. 605–609, 2020.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” *arXiv:1706.03762 [cs]*, Dec. 2017. arXiv: 1706.03762.

[42] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, 2018.

[43] C. Hanilçi and T. Kinnunen, “Source cell-phone recognition from recorded speech using non-speech segments,” *Digital Signal Processing*, vol. 35, pp. 75–85, Dec. 2014.

[44] V. Pandey, V. K. Verma, and N. Khanna, “Cell-phone identification from audio recordings using PSD of speech-free regions,” in *2014 IEEE Students’ Conference on Electrical, Electronics and Computer Science*, (Bhopal), pp. 1–6, IEEE, Mar. 2014.

[45] R. Aggarwal, S. Singh, A. K. Roul, and N. Khanna, “Cellphone identification using noise estimates from recorded audio,” in *2014 International Conference on Communication and Signal Processing*, (Melmaruvathur, India), pp. 1218–1222, IEEE, Apr. 2014.

[46] Y. Li, X. Zhang, X. Li, Y. Zhang, J. Yang, and Q. He, “Mobile Phone Clustering From Speech Recordings Using Deep Representation and Spectral Clustering,” *IEEE Transactions on Information Forensics and Security*, vol. 13, pp. 965–977, Apr. 2018.

[47] G. Baldini and I. Amerini, “An Evaluation of Entropy Measures for Microphone Identification,” *Entropy*, vol. 22, p. 1235, Oct. 2020.

[48] T. Qin, R. Wang, D. Yan, and L. Lin, “Source Cell-Phone Identification in the Presence of Additive Noise from CQT Domain,” *Information*, vol. 9, p. 205, Aug. 2018.

[49] G. Baldini and I. Amerini, “Smartphones Identification Through the Built-In Microphones With Convolutional Neural Network,” *IEEE Access*, vol. 7, pp. 158685–158696, 2019.

[50] G. Baldini, I. Amerini, and C. Gentile, “Microphone Identification Using Convolutional Neural Networks,” *IEEE Sensors Letters*, vol. 3, no. 7, p. 4, 2019.

[51] N. D. Gaubitch, M. Brookes, P. A. Naylor, and D. Sharma, “Single-microphone blind channel identification in speech using spectrum classification,” in *2011 19th European Signal Processing Conference*, pp. 1748–1751, 2011.

[52] N. D. Gaubitch, M. Brookes, and P. A. Naylor, “Blind Channel Magnitude Response Estimation in Speech Using Spectrum Classification,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, pp. 2162–2171, Oct. 2013.

[53] A. T. Ho and S. Li, eds., *Handbook of Digital Forensics of Multimedia Data and Devices.* Chichester, UK: John Wiley & Sons, Ltd, July 2015.

[54] L. Cuccovillo and P. Aichroth, “Open-set microphone classification via blind channel analysis,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2074–2078, 2016.

[55] M. Slaney, *A MATLAB Auditory Toolbox: Toolbox for Auditory Modeling Work, version 2.* Interval Research Corporation, 1998.

[56] “Pre-emphasis - Introduction to Speech Processing - Aalto University Wiki.” `https://wiki.aalto.fi/display/ITSP/Pre-emphasis`.

[57] L. R. Rabiner and R. W. Schafer, *Theory and applications of digital speech processing.* Upper Saddle River: Pearson, 1st ed ed., 2011. OCLC: ocn476834107.

[58] J. Zhou, “On discrete cosine transform,” *arXiv:1109.0337 [cs, math]*, Sept. 2011. arXiv: 1109.0337.

[59] K. S. Rao, V. R. Reddy, and S. Maity, *Language Identification Using Spectral and Prosodic Features by K. Sreenivasa Rao, V. Ramu Reddy, Sudhamay Maity.* SpringerBriefs in Speech Technology, Studies in Speech Signal Processing, Natural Language Understanding, and Machine Learning, Cham: Springer International Publishing : Imprint: Springer, 1st ed. 2015. ed., 2015.

[60] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," *IEEE Transactions on Image Processing*, vol. 26, pp. 3142–3155, July 2017.

[61] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015. arXiv: 1409.1556.

[62] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206–5210, 2015.

[63] "LibriVox." `https://librivox.org/`.

[64] "Project Gutenberg." `https://www.gutenberg.org/`.

[65] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, "Timit acoustic-phonetic continuous speech corpus," *Philadelphia Linguistic Data Consortium*, 1993.

[66] "Audiomentations - a python library for audio data augmentation." `https://github.com/iver56/audiomentations`.

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[68] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[69] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv:1502.01852 [cs]*, Feb. 2015. arXiv: 1502.01852.

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[71] A. Defossez, G. Synnaeve, and Y. Adi, "Real time speech enhancement in the waveform domain," in *Interspeech*, 2020.

[72] A. Défossez, N. Usunier, L. Bottou, and F. Bach, "Music Source Separation in the Waveform Domain," *arXiv:1911.13254 [cs, eess, stat]*, Apr. 2021. arXiv: 1911.13254.

[73] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167 [cs]*, Mar. 2015. arXiv: 1502.03167.

[74] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

[75] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, pp. 416–423 vol.2, 2001.

[76] Z. Borsos, Y. Li, B. Gfeller, and M. Tagliasacchi, "MicAugment: One-shot Microphone Style Transfer," *arXiv:2010.09658 [cs, eess, stat]*, Oct. 2020. arXiv: 2010.09658.

[77] European Commission. Joint Research Centre., *Microphone smart device fingerprinting from video recordings: Project AVICAO Authors and Victims Identification of Child Abuse On line.* LU: Publications Office, 2018.

[78] S. Milani, L. Cuccovillo, M. Tagliasacchi, S. Tubaro, and P. Aichroth, "Video camera identification using audio-visual features," in *2014 5th European Workshop on Visual Information Processing (EUVIP)*, pp. 1–6, 2014.