POLITECNICO
MILANO 1863

Dipartimento di Elettronica, Informazione e Bioingegneria

Master Degree in Computer Science and Engineering

Synthetic Speech Detection through Convolutional Neural Networks in Noisy Environments

by:
Eleonora Landini

matr.:
916202

Supervisor:
Prof. Paolo Bestagini

Co-supervisor:
Dr. Clara Borrelli

Academic Year
2020-2021

Abstract

Nowadays, deepfakes are well known to a lot of people, from forensic analysis experts to teenagers on social media. Social networks have a big role in the spreading of these artificial generated media and there is a reason. Video, images and speech deepfakes are entertaining and they easily attract the interest of many because of the remarkable resemblance they have with respect to the real people they mimic.

To the genuine interest on this subject for the creation of entertainment content, we have to add the one influenced by the malicious means these technologies may be used for. Because of these spiteful intents, phenomena like people impersonation have the likelihood to spread worldwide at increasing speed, with the concurrent fast creation of new algorithms for media synthesis. Furthermore, the always evolving improvements affecting Machine Learning are a powerful aid for these purposes.

Understandably, the rise of deepfakes techniques coincides with much deeper investigations on synthetic media detection. Video deepfakes detection algorithms have the richest literature in this field. Conversely, audio deepfake detection has been less investigated and needs attention.

In this thesis we propose a method for the classification of synthetic speech excerpts in noisy environments based on Convolutional Neural Networks (CNNs). The proposed system is a composition of a preprocessing denoising DnCNN followed by a VGGish Convolutional Network that acts as classifier. The two networks are jointly trained in an end-to-end framework. Our results confirm our expectation, showing that our end-to-end approach outperforms other solutions based on disjoint denoising and classification.

Sommario

Oggigiorno i deepfake sono conosciuti molto bene da molte persone, a partire da esperti di analisi forense fino a ragazzi sui social media. I social network hanno un importante ruolo nella diffusione di questi contenuti multimediali, e per una ragione. Deepfake di video, immagini e registrazioni forniscono un ottimo intrattenimento e incuriosiscono facilmente molti grazie alle impressionanti somiglianze che hanno rispetto alle persone che imitano.

Al genuino interesse su questi file per la creazione di contenuti di intrattenimento, dobbiamo aggiungere quello guidato dagli scopi malevoli per cui si può utilizzare questa tecnologia. A causa di questi fini illegali, fenomeni come lo scambio d'identità hanno il potenziale di diffondersi molto rapidamente in tutto il mondo, con un veloce incremento nella creazione di algoritmi per la sintesi di contenuti multimediali. Inoltre, i continui sviluppi nel campo del Machine Learning sono un potente strumento per questi intenti.

Comprensibilmente, l'aumento di tecniche per generare deepfake coincide con maggiori indagini per il rilevamento di file multimediali creati artificialmente. La letteratura riguardante il rilevamento deepfake video è la più ricca in questo ambito. Al contrario, l'individuazione di deepfake audio è meno trattata e necessita di maggiore attenzione.

In questa tesi proponiamo un metodo per la classificazione di tracce vocali in ambienti rumorosi basato su reti neurali convoluzionali (CNN). Il sistema proposto è composto da una DnCNN usata come riduttore del rumore preliminare seguita da una rete convoluzionale VGGish che agisce come classificatore. Le due reti sono allenate congiuntamente in

una struttura end-to-end. I risultati confermano le nostre aspettative, mostrando che l'approccio end-to-end supera di gran lunga soluzioni basate su riduzione del rumore disgiunta dalla classificazione.

Acknowledgements

Vorrei ringraziare Paolo e Clara, come mi hanno detto di chiamarli, per la loro disponibilità a rispondere alle mie infinite domande, per la loro guida in questo lavoro e per avermi mostrato cosa significa fare un lavoro per cui si prova entusiasmo.

Devo assolutamente ringraziare i miei finanziatori. Grazie mamma. Grazie babbo. Grazie per le opportunità che mi avete sempre concesso fidandovi del mio giudizio e delle capacità in cui ogni tanto non credo neanche io. Grazie Ale, perché nonostante io sia una sorella vagabonda, tu mi trovi sempre.

E ora grazie a tutti i miei amici: quelli con cui ho condiviso una vita, quelli con cui ho condiviso una casa e quelli con cui ho condiviso stati di ansia altissimi. Vi voglio bene.

Non è per fare favoritismi, ma un ringraziamento speciale va a Cecilia, con cui ho condiviso ogni dettaglio piacevole e spiacevole di questi ormai tre, ultimi, anni universitari.

E.L.

Contents

List of Figures

List of Tables

Glossary

AI Artificial Intelligence. xiii, 6, 21

ANN Artificial Neural Network. 6

AR autoregressive. xv, 16, 44

ASR automatic speech recognition. 44

ASV Automatic Speaker Verification. 11, 12, 19, 20, 40, 41, 45, 79

CART classification and regression tree. 43

CLDNN Convolutional LSTM Deep Neural Network. 20

CNN Convolutional Neural Network. vii, viii, xv, 1, 6, 8–10, 20, 23–26, 31–33, 38, 49–51, 54, 57, 59–64, 66, 68–74, 76–79

CQCC constant Q cepstral coefficient. xiv, 19

CQT constant Q transform. 19

DFT Discrete Fourier Transform. 3

DKPLS Dynamic Kernel Partial Least Squares. 18

DNN Deep Learning Network. 8, 9, 16, 18, 20

DSP Digital Signal Processing. 12, 13, 15

FFT Fast Fourier Transform. 52

FT Fourier Transform. 3

GMM Gaussian Mixture Model. xv, 18, 19

HMM hidden Markov model. xiv, 16, 42

LPCC linear predictive cepstral coefficient. 17

LSF line spectral frequencies. 17

LSTM Long Short-Term Memory. 20

LTS letter-to-sound. 13, 14

MFCC mel-frequency cepstral coefficient. 16, 17, 19

MSE Mean Squared Error. 37, 38, 53

NLP Natural Language Processing. 12–15

NN neural-network. vii, xiii, xv, 1, 6–8, 10, 11, 20, 42–44, 75

PDE Partial Differential Equation. 27

PSOLA Pitch Synchronous Overlap and Add. 16, 17

RNN Recurrent Neural Network. xv, 7, 20, 43, 44

SAR shallow autoregressive. 42

SNR Signal to Noise Ratio. viii, 2, 3, 45–47, 59, 61, 72, 74, 79

SPSS statistical parametric speech synthesis. 43

STFT Short Time Fourier Transform. 2–4, 26, 52

SVM Support Vector Machine. xv, 19

TSU *threshold logic unit*. 7

TTS Text-To-Speech. vii, xiv, 12, 13, 18, 20, 41–44, 79

VAE variational auto-encoder. 42–44

VC Voice Conversion. vii, xiv, 12, 17, 18, 41, 43–45, 79

VCTK Voice Cloning Toolkit. 40

Introduction

A few years ago, no one would have been able to suggest being accustomed to see videos of people they knew, saying things they never said, perfectly moving their lips to the lyrics they were hearing. Nor anyone could have imagined seeing the president of the United States of America singing some upbeat new pop song while standing beside a debate podium. Nowadays, this is no big news for a lot of people. The multiple social media platforms most of us now use made this kind of content popular for its funny reactions and the curiosity it can spark into people, because of the resemblance it can achieve to the real subjects. The mass diffusion of these media concurred to the increasing interest in the techniques used to create these files, called *deepfakes*. Deepfakes are generated via Artificial Intelligence (AI) techniques applied to video, image and audio signals.

As entertaining as deepfake videos, images and sounds can be, they can also be the means of malicious intent. The broadcast diffusion of deepfakes may coincide with the spreading of fake news, people impersonation, and could end up in revenge porn as well. Because of their distribution on social media, synthetic videos are the first ones we think of when we hear of deepfakes, therefore, we find many examples of these in the literature: from computer graphics approaches, like [6, 7], to techniques employing NNs such as [8, 9, 10].

However, audio deepfakes are as common and as dangerous as their video counterpart. Nonetheless, forensics studies about their detection have been less investigated in the literature. For this reason, in this thesis we focus on speech synthetic content, for which there is equal

enthusiasm in the technology community: [11, 12] are just a few example of widespread tools for speech synthesis. Usually, speech deepfakes are produced using TTS and VC algorithms. These span from simple concatenation of waveforms, e.g. [13], to vocoders exploiting speech signals' source-filter model [14], comprising models employing the power of Machine Learning like the famous raw waveform generator model, the WaveNet [15], and the hidden Markov model (HMM)-based algorithm [16], joining HMMs techniques to a vocoder to reproduce speech waveforms.

With these always evolving synthesis techniques, the study of fake speech is nowadays central in forensic analysis. In [17], the authors state: "the objective of the audio forensic analysis is to establish as far as possible that the recording is a true 'acoustic representation' of events made at through a specific acquisition system and at a specific location." Hence, we can insert synthetic speech review as part of the audio forensic analysis. In fact, a speech deepfake excerpt could make its way into a hearing court and it could be identified as reliable source, thus eluding one of the three primary concerns in audio forensics: *authenticity*, [18]. The other two concerns mentioned in [18] are *interpretation* and *enhancement*. The latter is needed because recordings brought as evidence in trials suffer more than often from background noise, they risk being low in volume and have other noise sources tampering with the clarity of the sound file in exam.

In order to avoid inauthentic recordings and more specifically the usage of deepfakes for impersonation, audio forensic analysts need detection systems that perform well even in noisy environments. We know of the existence of multiple video deepfake analyzers in literature [19, 20, 21, 22], but there are not as many in the speech field [23, 24, 25]. For this reason we try to tackle the speech synthesis detection problem, taking into account the less recent and state-of-the-art technologies involving this field. Most classical detection techniques require two separate stages in order to detect whether a considered recording is actually produced by a human being or not: a feature extraction step and a classification one. Features are often extracted using techniques such as constant Q cepstral coeffi-

cients (CQCCs) from [26], the exploitation of sub-band analysis [27], the consideration of multiple different autoregressive (AR) orders at once and the consequent combination with bicoherence computation [25]. CNNs are also adopted for this stage, e.g. in [28]. As for classification of the extracted features, common Machine Learning approaches include Gaussian Mixture Models (GMMs) and Support Vector Machine (SVM) [27], while NNs inspired ones include e.g. the use of Recurrent Neural Networks (RNNs) [28]. Other models implement classification of synthesized speech without the separation of the two stages, like [23] and [29].

Since the aforementioned studies on speech deepfake detection had pristine real and fake recordings as main focus of their work, we expand the field of investigation to noisy inputs. This is necessary to move towards a much more realistic case study, and to act in prevention of deliberate noise contamination of speech deepfake excerpts in order to better avoid detection.

The focus of this thesis is therefore to apply a classification algorithm, by the means of a CNN called VGGish [30], to pristine and noisy recordings, with the aim of correctly assigning each track to the class it belongs to: the real or fake one. As we expect an important decrease in performances in presence of noisy inputs, we propose multiple solutions to improve these results. We explore the combination of different classic signal processing denoisers with the classification system. We then use a data driven kind of denoiser as pre-processing before the CNN, and finally we set an end-to-end experiment comprising the two CNNs training together.

We start our work in Chapter 1, by giving a background overview on the mathematical approaches we are going to use in the setting of the experiments. Our research is centered around speech signals analyzed as images, hence we describe the time-frequency representation of signals, explaining the signal plus noise model and how a speech signal can be transformed into an image. Moreover, we lay down the fundamentals of how a Convolutional Neural Network is generated and how it works.

In Chapter 2 we go in further detail on what deepfakes are, how they can be generated and the state-of-the-art of detection and creation of

said files.

Chapter 3 represents the core of this thesis, containing the explanation of the solutions we briefly introduced above. Hence, what we conceived to solve the problem of fake speech detection and the basic methodology with which we intend to move forward.

In the Experimental Setup chapter, Chapter 4, we lay down in detail the sets of data used as input of the system we created, and all the parameters used to implement our experiments. Furthermore, we delineate the parameters we will use to evaluate the results presented in the following chapter.

Chapter 5 contains the most meaningful results obtained through our laid out systems. We also compare the performances of the different solutions, and identify the best solution among the proposed ones.

At last, in Chapter 6 we draw our conclusions on the developed work, and we outline some steps that could bring to further progress in this field of study.

1

Theoretical Background

In this chapter we explain some background concepts that will be useful to understand the following chapters. As mentioned above, our work will center around the classification of clean and noisy speech signals by means of a CNN.

We therefore organize this chapter by starting from the explanation of time-frequency representation of audio signals. We describe the signal and noise model we take into account and the transformations of the speech signal we require for the study we intend to look into.

Afterwards, in Section 1.2 we lay down the basics of NNs and then focus on CNNs.

1.1 Time-Frequency Representation of Audio Signals

Our work is centered around the classification of audio signals, therefore we hereby describe some notions of the way these can be represented in time and frequency domain.

We consider audio as a digital signal, obtained by sampling the analog ones at a certain frequency. Moreover, we do not only take into account pristine audio files but noisy ones too, hence we give an overview of the signal and noise model. This model is important for the computation of the SNR, which we will use to model the noisy signals in Chapter 4.

Moreover, we explain the steps to follow in order to obtain a 2D representation of audio signals: the log-mel spectrogram. We therefore outline the concepts of Short Time Fourier Transform (STFT), spectrogram and Mel scale.

1.1.1 Signal and Noise Model

We consider $x(t)$ as a generic sum of sinusoidal signals, as we can assume with sound excerpts, defined as such:

$$x(t) = \sum_{i=1}^{N} A_i(t) \cos \theta_i(t), \tag{1.1}$$

where $A_i(t)$ and $\theta_i(t)$ respectively represent the instantaneous amplitude and phase of the i-th sinusoidal component.

We then define the noise signal as $\nu(t)$ and we assume it to be well modeled by filtered white noise:

$$\nu(t) = (h_t * u)(t) \triangleq \int_0^t h_t(t-\tau)u(\tau)d\tau, \tag{1.2}$$

where $u(t)$ is the white noise and $h_t(\tau)$ is the impulse response of a time varying linear filter at time t, [31].

Therefore the signal and noise model can be defined as:

$$x_\nu(t) = x(t) + \nu(t). \tag{1.3}$$

Starting from this model we can compute the Signal to Noise Ratio (SNR), that represents the comparison between the desired signal and the disturbance, the noise.

$$\text{SNR} \triangleq \frac{P_x}{P_\nu}, \tag{1.4}$$

where P_x and P_ν are respectively the average power of the clean signal and of the noise signal. Knowing that the average power of a signal is equal to the expected value of the squared signal, $P_x = \text{E}[x(t)^2]$, we can deduce that:

$$\text{SNR} = \frac{\text{E}[x(t)^2]}{\text{E}[\nu(t)^2]}. \tag{1.5}$$

Finally, considering zero mean signals we can consider $\text{E}[x(t)^2]$ as equal to the square of the standard deviation σ_x^2, hence:

$$\text{SNR} = \frac{\sigma_x^2}{\sigma_\nu^2}, \tag{1.6}$$

where σ_x and σ_ν are respectively the standard deviation of the pristine signal and of the noise signal's one.

1.1.2 Short-Time Fourier Transform

The STFT, differently from the Fourier Transform (FT), is able to describe a signal on the time and frequency domain at the same time and this comes in handy with audio signals. The basic concept of this algorithm is dividing the signal in N samples long time frames using a windowing signal, and applying a Discrete Fourier Transform (DFT) to each windowed frame. Considering a discrete signal $x(n)$, its STFT is defined as follows:

$$X(m,k) = \sum_{n=0}^{N-1} x(n+mH)\ w(n)\ e^{-j\frac{2\pi nk}{N}}, \tag{1.7}$$

where m is the time frame, k is the frequency bin, H is the hop length, $w(n)$ is the window applied to $x(n)$ and N is the frame size. Therefore, the output of the STFT is the Fourier coefficient for the k-th frequency at the m-th time frame.

Having this information we can now compute the spectrogram of our signal. This is generated by computing the square value of the magnitude

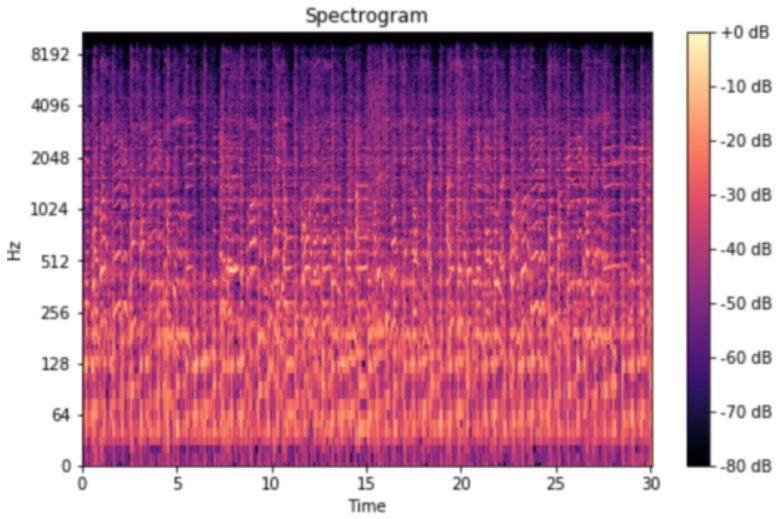


Figure 1.1: Spectrogram representation of an audio signal, from [1].

of each $X(m, k)$ STFT coefficient. We show an example of the result of this calculation in Figure 1.1. Here we can denote that time frames are put in the horizontal axis, frequency bins are on the vertical one and the intensity of the spectrogram's coefficients are color coded. Furthermore, the time axis is logarithm scaled and the intensity is measured in decibels. This aligns with the fact that human perceive sound with a logarithmic scale: better hearing difference between low frequency sound with respect to differences between high pitches, [1].

1.1.3 Mel-Scale

In [2] S. S. Stevens and J. Volkmann propose a new scale for pitch and linear frequency, the Mel-scale. This is a scale of pitches where the intervals represent how the human ear truthfully perceives equally distanced frequencies. Hence, the perceived Mel-frequency at frequency k is equal to:

$$\text{Mel}(k) = 2595 \cdot \log_{10}\left(1 + \frac{k}{700}\right). \tag{1.8}$$

In Figure 1.2 we display the graph Stevens and Volkmann used to describe the scaling as perceived by human ears.

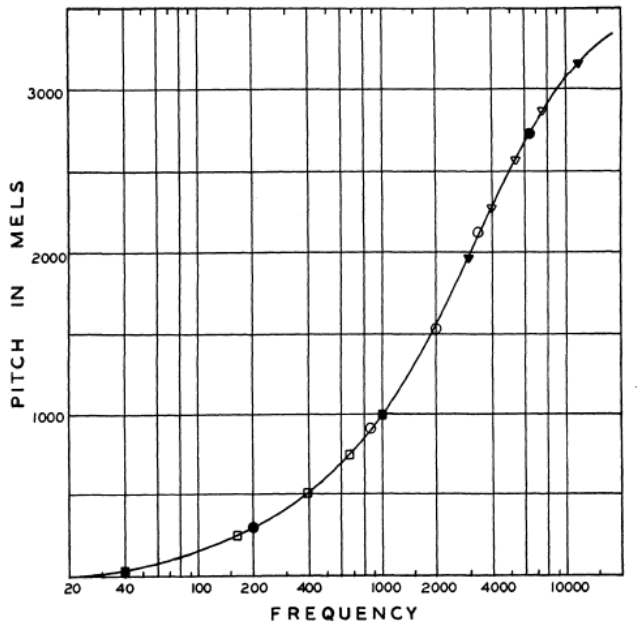


Figure 1.2: Scaling of perceived pitch with respect to linear frequencies proposed in [2].

1.1.4 Log-Mel Spectrogram

Finally we now define the log-mel spectrogram as a logarithmically scaled spectrogram where the frequencies have been scaled using the Mel-scale. An example of this 2D representation of audio signals can be found in Figure 1.3.

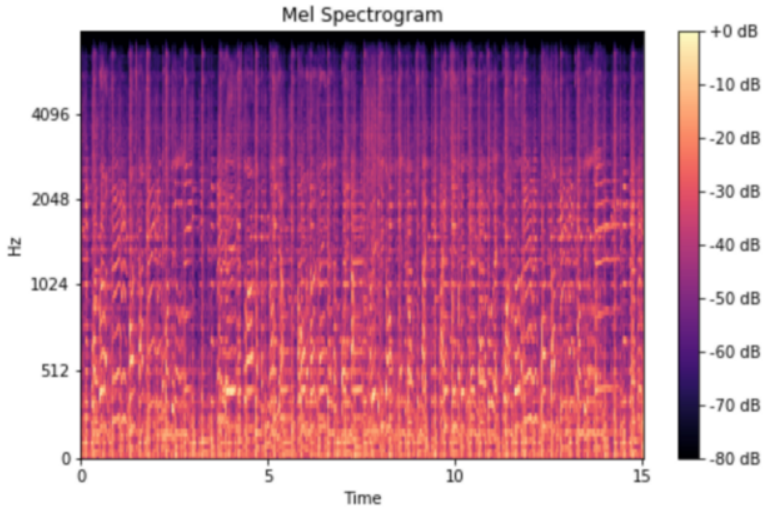


Figure 1.3: Log-mel spectrogram of a generic audio signal, from [1].

1.2 Convolutional Neural Network

In this section we try to give a brief explanation of how NNs and CNNs work and the difference between the disparate most commonly used typologies.

The networks we look into are examples of machine learning, a category of AI processes that aims to build models that can execute tasks without explicitly programming parameters to do so.

1.2.1 Neural Networks

In [32], Gurney writes that a neural-network (NN), also called Artificial Neural Network (ANN), is "an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the interunit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns". Therefore NNs aim to mimic the functioning of synapses' firing connection using particular mathematical and algebraic techniques.

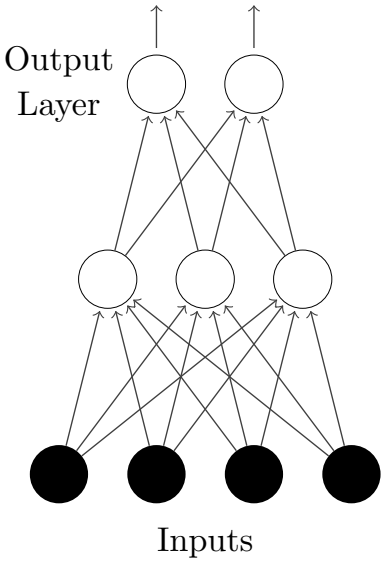


Figure 1.4: A simple NN.

In Figure 1.4 we show a simple NN. Each circle characterizes a node, the artificial network's representation of a neuron. Synapses are modeled

by weights that are multiplied to each input, depending on the importance of that node. What happens next can better be understood with the aid of a couple of equations:

$$a = \sum_{i=1}^{n} w_i x_i \tag{1.9}$$

$$y = \begin{cases} 1, & a \geqslant t \\ 0, & a < t \end{cases}. \tag{1.10}$$

In (1.9), x_i is the i-th input of the network, one of the full circles in Figure 1.4, w_i is the weight assigned to i-th input and a can be called *node activation*, or *threshold logic unit* (TSU), [32]. As shown in (1.10), if the value of the activation is greater than a threshold t, the output of the computation is 1 and the connection to the y node taken into consideration is activated, otherwise the output is zero and y is considered off.

Analyzing Figure 1.4 we can infer the typical structure of a NN. Each row of circles represents what we call a *layer* and this network has an input layer, an output layer and one hidden layer. Hidden layers are the ones not visible from the outside of the architectures and the amount and type of these concurs to the definition of the particular category of NN, e.g. convolutional, recurrent, deep.

Moreover, the structure portrayed in Figure 1.4 and the process explained in (1.9) and (1.10) define a feedforward NN, as the work flow of the net propagates from input to output. Conversely, another example of NN is the RNN, that is a backwards propagation network, as it introduces memory in the process: hidden layers preserve information about previous inputs that can influence the final outcome of the whole network.

Just like real neurons can sometimes change their behavior depending on their stimulus input, weights of NNs evolve as well. In our thesis we focus on updating rules involving supervised learning, that is applied when both the inputs and the target signals are available to the network. In this case the evolution of weights is related to how well fitting the outputs are with respect to the target signals. We measure the discrep-

ancy between targets and outputs with loss functions that influence the adjusting of the weights.

For the circumstance of our work it is relevant to mention a further distinction between different NNs, as the main problem we tackle is a classification one, different from the regression one. Classification problems occur when the targets of the considered network fall under determined categories and the net has the aim of identifying the right one for each input. An example of this could be the training of a net to select the correct item of clothing given an input dataset made of images of clothes.

1.2.2 Convolutional Neural Networks

CNNs fall into the category of Deep Learning Networks (DNNs), hence they employ many hidden layers in their implementation. The origin of this particular type of NN lays on the fully connected structure of common NNs, shown in Figure 1.5. A layer is said to be fully connected when

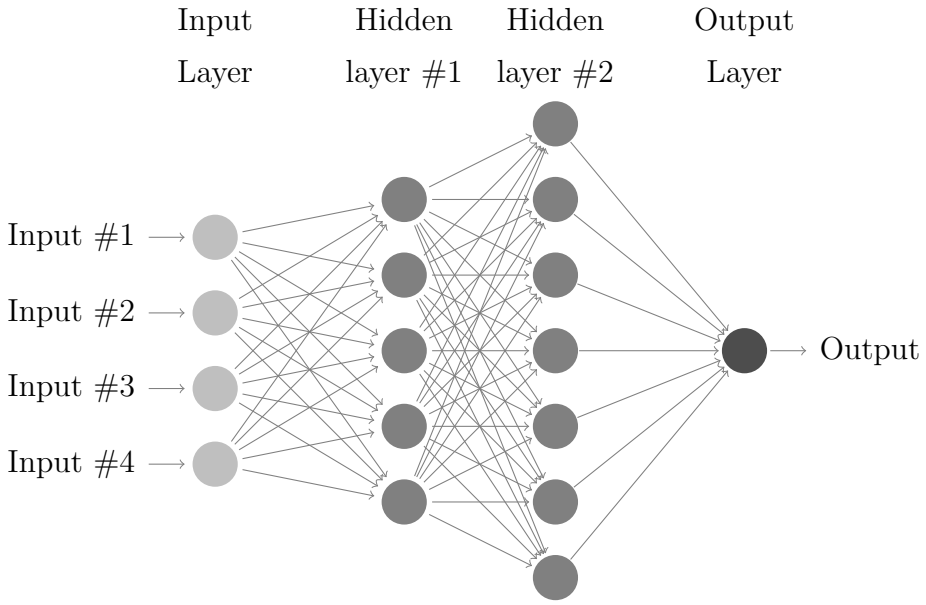


Figure 1.5: A fully connected NN.

all of its neurons are connected to each of the nodes on the previous layer and on the next one, e.g. both hidden layers in Figure 1.5. This consequently culminates in a very complex computation, highly demanding in computational power and decidedly time consuming, considering the

fact that DNNs are usually used for processing image inputs. Therefore, these may reach a high number of parameters starting from the input layer.

The innovation of the CNN lays in the convolutional layers: each one of their neurons is only connected to a subset of neurons in the previous layer, thus reducing the computation complexity of all the system and being able to focus on certain tasks, [33]. Typical CNNs are structured as displayed in Figure 1.6: many subsequent blocks composed of a convolution layer, followed by an activation state and a pooling one preceding one flattening stage and one fully connected layer.

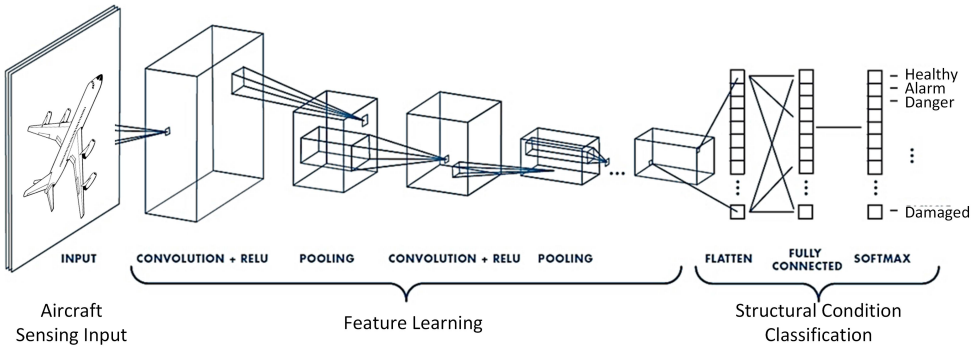


Figure 1.6: Example of architecture of a CNN, from [34].

Each block we just mentioned comprises multiple convolution kernels that learn feature representations of the inputs. Assuming that the network has a 2D input, so an image signal, the convolution for one pixel is computed as follows:

$$y(t,f) = (x * w)[t,f] = \sum_m \sum_n x[m,n] \cdot w[t-m, f-n], \qquad (1.11)$$

where $y(t,f)$ is the output flowing to the next layer, x represents the input image, w is the kernel or filter matrix and $*$ represents the convolution operator [35].

The second step is applying an element-wise non-linear activation function applied to the convolved results $y(t,f)$, thus generating what we call feature maps, [34]. It is important to mention that the kernel, w, is shared between the neurons belonging to the same feature map, thereby reducing the complexity of the network keeping the number of

parameters low, [33]. The non-linearity can be used to adjust or cut-off the generated output. This layer is applied in order to saturate the output or limiting the generated output. Among the many activation functions we can employ the most common are ReLu, sigmoid and tanh, [35].

Following the activation stage the system feeds this layer to a pooling layer, that performs a downsample in height and weight reducing the dimension of the activation maps without loss of information and the number of parameters of the net. We hence further decrease the overall computational complexity. Some common pooling operations are max pooling, average pooling, stochastic pooling and spectral pooling, [33]. This concludes the bundle of layers that are often repeated in CNNs.

The flatten layer is used to change the shape of the input, making it an array of 1 neuron depth and height, equal in length to the product between the length, depth and height of the input to that layer. This layer is used because the output layer must be a one-dimensional vector, [34].

One of the main negative implications of Deep CNNs is the arising of overfitting. The overfitting phenomenon occurs when the loss in the validation phase is higher than the one in the training one. This happens when the model is able to find the best fit to the training data to the extent that is impacts the performances with new data [36]. To attenuate this behavior we conventionally feed the output of the flattening stage to a regularization layer, usually *Dropout* or *DropConnect* techniques, [33], to randomly cut off a fraction of the nodes of the network.

The final layer is the fully connected one, that is fairly similar to the ones composing usual NNs. The major drawback of this layer, is that it includes a lot of parameters that need complex computational in training examples, [35].

A last activation layer, as shown in Figure 1.6, can be added at the end of the just described pipeline, depending on the specific implementation of the CNN in exam.

2

State of the Art

In this chapter we explain the state of the art relative to the technologies we focus on for this thesis, pointing out the basic assumptions we make in our work and illustrating the novelties our study may bring to the literature.

Therefore we hereby discuss the origin of the deepfake speech detection problem starting from the description of what synthetic speech is, how it was initially generated and how it is evolved during the recent years with the introduction of NNs. We then give an overview of old and new techniques used for speech synthesis detection.

2.1 Synthetic Speech Generation

Automatic Speaker Verification (ASV) is the task of recognizing an identity using the voice as input. It is a fairly common biometric technology used all around the world and its implementation dates a long time ago, e.g. in [37], from 1976, Rosemberg gives a review of the methods to im-

plement automatic techniques for speaker verification. Hence, the study of speech signals in this field is not new and it is still expanding and evolving. For sure, innovation and technology development come with a price, in fact it is now common knowledge that Automatic Speaker Verification (ASV) is vulnerable to manipulation via *spoofing*, also referred to as *presentation attacks*, [5]. There are four known main classes of spoofing attacks: impersonation, replay, speech synthesis and voice conversion.

Impersonation attacks involve one person mimicking another person's voice with their own and the vulnerability of ASV systems to this kind of attack is still uncertain, [5].

Replay attacks are the easiest ones to perform, they require inexpensive technology, a microphone and a tool for playback, and are really hard for ASV systems to detect. This kind of attack is carried out by recording a bonafide access attempt, presumably secretly from the target person, and then presenting said recording to the ASV system. Hence, they can be accomplished by not specialized attackers as well, [38].

Speech synthesis attacks, commonly mentioned as Text-To-Speech (TTS) attacks, are employed to create "intelligible, natural sounding artificial speech for any arbitrary text", [38]. They usually involve text analysis followed by speech waveform generation, often called front-end and back-end respectively.

Voice Conversion (VC) attacks work directly on speech inputs converting an attacker's natural voice into the target's one, [38]. This and the previous spoofing techniques are the focus of the next sections, them being the algorithms employed for the creation of the dataset in [5], the one we used for our thesis.

2.1.1 Text To Speech

As stated in [39], a TTS synthesizer is composed by two main blocks, Figure 2.1: a Natural Language Processing (NLP) module, with the task of producing a phonetic transcription of the text to read together with the wanted intonation and rhythm, usually referred to as *prosody*, and a Digital Signal Processing (DSP) module, that turns the information it

receives from the previous block into actual speech. We hereby explain each module.

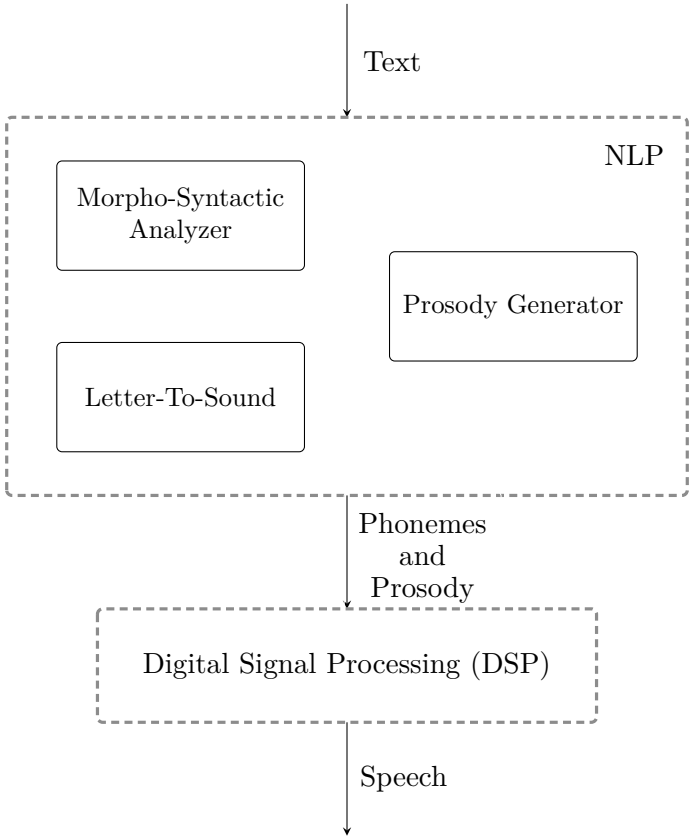


Figure 2.1: General architecture of a TTS algorithm.

2.1.1.1 Natural Language Processing Module

This component consists of three modules itself, shown in the upper part of Figure 2.1: a text analyzer, a letter-to-sound (LTS) module and a prosody generator.

The text analyzer has the duty of parsing the text and grouping words and small sentences to use them then as input of the LTS module. In particular, this module has a pre-processor that has the job to organize sentences in peculiar lists of words. It then sends these organized words to a morphological analysis module, that, for each word taken individually, proposes all possible part-of-speech categories it belongs to. Following this stage, we have a contextual analysis module that considers words in their context, lowering the amount of part-of-speech categories just mentioned to a short list with high probable hypothesis. Finally,

there is a syntactic-prosodic parser that finds the organization of the text into clause and phrase-like constituents. This text analyzing module represents a morpho-syntactic analyzer that has to be able to reduce a given sentence to something resembling a sequence of part-of-speech belonging to it, unveiling its internal structure. This is important for LTS and the prosody generator, and the correct functioning of the Natural Language Processing system. As for the LTS, in order to obtain an accurate phonetic transcription, we need the part-of-speech category of some words to be available. Furthermore, the dependency relationship between successive words is required for the same aim. Regarding the prosody generator, syntax is really important for natural prosody and, given the higher complexity in finding the generative aspects of semantics and pragmatics, speech synthesizers focus mainly on the first.

As aforementioned, the LTS module is the one responsible for automatic phonetization of the incoming organized text. What may seem an easy look-up task, is actually more complicated, since this process highly relies on the preliminary morpho-syntactic analysis brought out by the text analysis module. The proposed solutions for the LTS task span between *dictionary-based* and *rule-based* strategies. *Dictionary-based* resolutions embody the storing of a maximum amount of phonological knowledge into a lexicon. On the other hand *rule-based* transcription systems execute a different strategy. They transfer the majority of dictionaries' phonological information into a LTS set of rules. With this strategy, only those words that compose a rule of their own, because of their peculiar pronunciation, are stored in an exception dictionary.

The last block of the NLP module is the one belonging to prosody generation. As outlined above, prosody deals with certain properties of speech signals related to rhythm, audible changes in pitch, loudness and syllable length. Prosodic features create a segmentation of the speech chain into groups of syllables and indicate the relationships between these groups. The grouping is not always equal to the syntactic organization of the words. Being able to produce a natural prosodic utterance is complicated, therefore in [39] the author infers that the main focus of the generator is to obtain an acceptable segmentation and to translate it into

the *continuation of words* or *finality* marks. This is to be done ignoring the *relationship between words* mark or their contrastive meaning. Once the prosodic-syntactic structure of a sentence has been derived, it takes part into the obtaining of the exact duration of each phoneme and of silences, joint to the intonation to give to them.

2.1.1.2 Digital Signal Processing Module

The job of the DSP component is reproducing the phonemes and the other information transmitted by the NLP module as if it was emitted by a human vocal tract, [39], hence creating proper speech waveforms. There are three main historical approaches to this part of the computation: *formant synthesizers*, *articulatory synthesizers* and *concatenative synthesizers*.

The synthesizers belonging to the first category are also called rule synthesizers, and they describe speech as the dynamic evolution of many parameters, mostly referring to formant and anti-formant frequencies and bandwidths, together with glottal waveforms. Formants are although difficult parameters to estimate starting from speech data, therefore this kind of synthesis tends to be time-consuming [39] and whilst being able to produce intelligible speech, it leads to a far from natural resulting speech, [40].

Articulatory synthesis uses the accurate modeling of the human articulatory behavior to produce speech. Despite the promise of a high quality result, the process is rendered difficult by the articulatory model. This includes control parameters as lip aperture, lip protrusion, tongue tip position, tongue tip height, tongue position and tongue height. The two main obstacles in using this synthesizer lay in the difficulty of acquiring data for the model, needing instruments to measure all the just mentioned parameters, and in the one of finding a balance between the accuracy of the model and the complexity of its design and control. Generally, the results supplied by *articulatory synthesis* are worse than both *formant* and *concatenative* ones, [40].

The last historical alternative for synthesis is the *concatenative* one, the most famous one. This overcomes the difficulty in finding formants of

the referring synthesis, by following a data driven approach. *Concatenative synthesis* generates speech connecting natural, pre-recorded speech units [40]. Speech units may differ in length. With longer units there is an improvement in the naturalness of the produced sound, but with it comes the necessity for more memory. Diphones are the most popular type of unit. Afterwards these units need time and pitch scaling. Among the techniques used to implement time-pitch scaling of diphones are: Pitch Synchronous Overlap and Add (PSOLA), residual excited linear prediction, mel-frequency cepstral coefficients (MFCCs) synthesis, sinusoidal models, [41].

Unit selection synthesis (also called corpus-based concatenative synthesis) starts from the basis laid by *concatenative synthesis* techniques and tries to solve its artifacts problem. This is sometimes produced in the attempt to obtain some desired prosody, causing the speech to sound unnatural. This algorithm stores in the unit inventory multiple instances of each unit with varying prosodies. A unit selection algorithm is then developed to select the closest match to the target prosody, and concatenate it to the rest of the speech signal, [40].

Some HMM-based approaches have been proposed to further enlarge the variety of speaking styles. These operate with contextual HMMs trained on large datasets of acoustic features extracted from diphones and triphones, [25]. These models are usually called HMM-based speech synthesis models, and they are part of the "statistical parametric speech synthesis", another data-driven approach on the matter. In a similar way to the prior alternative, HMMs represent not only the phoneme sequences, but also various contexts of the linguistic specification, [16]. Acoustic parameters generated by the HMM are selected following this linguistic specification and are used to drive a vocoder, which allows to generate a speech waveform dependent on vocal tract parameters and excitation ones, [16].

One last type of speech synthesizer presented in literature is the WaveNet: a DNN for generating raw audio waveforms. This is a probabilistic and AR model that can capture the characteristics of many different speakers with equal fidelity, and can switch between them by

conditioning on the speaker identity, [15].

2.1.2 Voice Conversion

As mentioned before, the main focus of VC algorithms is to modify speech coming from one speaker, the *source speaker*, into one that is perceived as coming from another speaker, the *target speaker*. These systems only modify speaker-dependent characteristics of speech, such as formants, the fundamental frequency and prosodic components of speech, while carrying over the speaker independent speech content, [3]. VC techniques commonly include three phases, displayed in the second row of Figure 2.2: speech analysis, mapping, reconstruction, [42]. The speech analyzer extracts features containing the needed information from the source signal, the mapping component changes them into the ones of the target speaker and finally the reconstruction stage is responsible for re-synthesizing the speech signals into the time-domain scale, [3]. As shown in Figure 2.2, both target and source speech signals concur to the creation of a conversion model, then used in the mapping stage of the process.

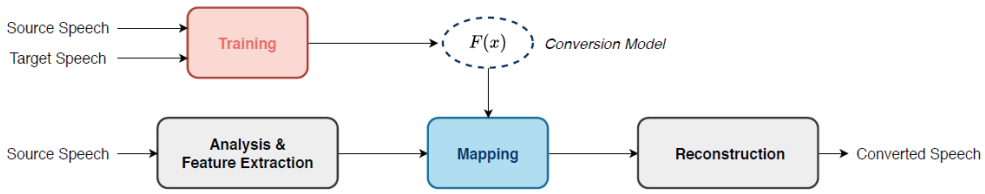


Figure 2.2: Flow chart of the general VC algorithm, from [3].

In the analysis phase, feature extraction has the aim of selecting parameters that well represent the individuality of the speaker. Typically used speech features include MFCCs, linear predictive cepstral coefficients (LPCCs), line spectral frequencies (LSF), [3]. Speech analysis and reconstruction are essential to Voice Conversion, although, like other signal processing techniques, they inevitably introduce artifacts. In literature, we find many approaches devoted to minimize such artifacts, [3]. These include PSOLA, vocoders, WaveNet vocoders [43] and recent development on neural vocoders[44].

The second component of the most common VC algorithms is the one corresponding to mapping. Feature mapping performs the modification of speech features from source to target speaker: spectral mapping seeks to change the voice timbre, while prosody conversion ought to modify the prosody features, such as fundamental frequency, intonation and duration. So far, spectral mapping remains the center of many voice conversion studies. Among the techniques employed to perform it we highlight GMMs [42], Dynamic Kernel Partial Least Squares (DKPLS) [45], Frequency Warping, that is able to fix the over-smoothing behavior introduced by the last two algorithms, and some recent DNNs mentioned in [3].

Finally, speech reconstruction operates on the modified parameters, coming from the mapping stage, as a sort of inverse function of speech analysis, and generates an audible speech signal.

2.2 Deepfake Speech Detection

As gathered from the previous section, we know of the existence of plenty of techniques to perform speech synthesis from text or from audio signals. These may be used for many decent reasons, such as improving accessibility through TTS models, or introducing new applications for leisure, and for the sake of exploring technology possibilities. Sadly these always evolving algorithms are also used for illegal purposes, like the ones we mentioned in the introduction. It is on these premises that synthetic speech detection has its origin, and in the next sections we try to outline the state-of-the-art technologies in the literature and the commonly used ones.

Traditional approaches for synthetic speech detection focus on extracting meaningful features from speech samples able to discriminate between fake and bonafide audio tracks, [25]. Specifically, it is common knowledge in this field that methods which choose effective and spoof-aware features outperform more complex classifiers, [46].

In [26] we find full supporters of this thesis, and their results point towards new generalized techniques to identify speech deepfakes. The

authors choose as main countermeasure approach artifact detection, implementing relatively standard feature extraction and statistical pattern recognition techniques. The hypothesis on which the paper is based on is that "the design of a spoofing countermeasure system which exploits a feature representation different to that of typical ASV systems, may offer greater robustness to spoofing, in addition to greater generalization to unforeseen spoofing attacks". The new extracted features are referred to as CQCCs, and are created by coupling the constant Q transform (CQT) for spoofing detection with traditional cepstral analysis, e.g. MFCCs. CQT employs geometrically spaced frequency bins, differently from the frequently used Fourier-based approaches, which impose regular spaced frequency bins and hence a variable Q factor. Therefore, CQT ensures a constant Q factor across the entire spectrum. The coupling with cepstral analysis facilitates the use of a conventional GMM for spoofing detection.

Moving forward from this, it has been noticed that the distribution of traces of synthetic speech algorithms happen unevenly across the frequency bands. Therefore, we find in the literature the exploitation of sub-band analysis for the means of detection of fake speech signals. Multiple examples of these techniques are reported in [27], where we find the comparison of 19 speech front-end features and their relative performances, evaluated with the implementation of classifiers based on GMM and SVM. Among the many, the authors investigate spectral flux based features and spectral sub-band centroids, which represent centroid frequencies of sub-bands, with properties similar to formant frequencies.

A different approach was proposed in [47], based on the assumption that synthetic or converted voice is quite likely to be either effortlessly predicted, due to its generation by means of a simplified model, or very difficult to predict, in the eventuality of the presence of artifacts in the signal. Hence, the suggested method is based on the analysis of prediction error: various parameters of prediction error are measured, capturing the features which will help to distinguish human from spoofed sources.

Due to the accomplishments in speech synthesis and classification tasks, recent studies have started introducing Deep Learning approaches. We find an example of this in [23], where the authors propose a speech

classification algorithm based on CNNs, considering the challenging scenario of state-of-the-art powerful TTS services, such as Google, Amazon, Microsoft and IBM. The algorithm comprises a shallow CNN applied to spectrograms, computed on short chunks of recorded audio signals. The CNN acts as a classifier, and it outputs a two-element vector indicating the analyzed audio window's likelihood of belonging to each class, i.e. bonafide or fake. This study can be identified as the inspiration of our own, implementing different CNNs and input datasets from the ones chosen for our study, but having the same baseline survey structure.

Another similar investigation was carried out in [28], where a CNN plays the role of a feature extractor, and subsequently an RNN is employed to capture the consistent spoofing properties. Thanks to back propagation, the CNN feature extractor and the RNN classification network are optimized simultaneously.

We think it is significant to present one last state-of-the-art approach, where raw waveforms are employed as inputs, conversely to the more typical log-mel ones. [29] implements the Convolutional LSTM Deep Neural Network (CLDNN) architecture, which combines three different types of neural networks into a single model. CLDNN performs time-frequency convolution to reduce spectral variance, long-term temporal modeling by using a Long Short-Term Memory (LSTM) NN, and finally classification using a DNN. This model is fed with an input in the form of a sequence of frames, and directly outputs a likelihood for the whole sequence.

We reckon it is meaningful mentioning that the ASVSpoof challenges conducted over the years were lengthily taken advantage of by many of the presented studies. Influenced by this trend, we too exploit the dataset created in the occasion of the ASVSpoof 2019 challenges and we will describe it in detail in Chapter 4.

We finally focus our attention on the latest literature on speech synthesis detection in noisy environment, core of our work. As a matter of fact, the literature results fairly lacking on the topic, although including a couple of articles were noise mitigation is mentioned in detection problems. E.g., in [48] noise reduction is computed through a spectral noise

subtraction method prior to the feature extraction stage, and afterwards the audio features are subjected to a further reduction of noise, by mean and variance normalizing. Despite proposing denoising methods in order to have a signal as pristine as possible, the authors focus their study on the robustness of the system with respect to unknown type of spoofing attacks, rather than on robustness to noise. An ulterior example can be found in [49], where the authors propose to perform AI-synthesized speech detection using bispectral analysis. In the computation of bicoherence they included "some averaging to ensure stable estimates". Once again, whilst being useful to the investigation on the performances of the suggested method, noise is not the main focus of the article. Therefore, considering the lack of studies carried out on noisy environment, we chose to concentrate our thesis on this specific topic, in hopes of building a system robust to noise.

3 Method

In this chapter we report all the details related to the method we propose to solve the speech spoof detection problem, which is the goal of this thesis. We first describe in detail the problem we want to solve. Then, we provide a thorough description of the proposed solution. In doing so, we illustrate the main blocks composing our pipeline and how we connect them to reach our goal.

3.1 Problem Formulation

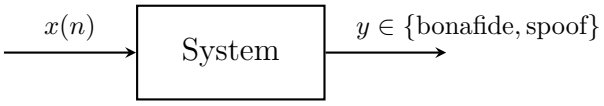
In this section we explain the main problem explored in this thesis. The aim of our study is to identify whether a certain piece of speech recording we analyze is spoofed or "bonafide". Let us denote a speech recording as $x(n)$. The considered problem consists of assigning to the signal $x(n)$ a label y such that represents the class the recording belongs to: *bonafide* or *spoof* (see Figure 3.1a). The *bonafide* class represents the recordings with real human sources (as described in [5]). Conversely, the *spoof* class

represents the synthesized recordings using different methods of voice conversion or text-to-speech.

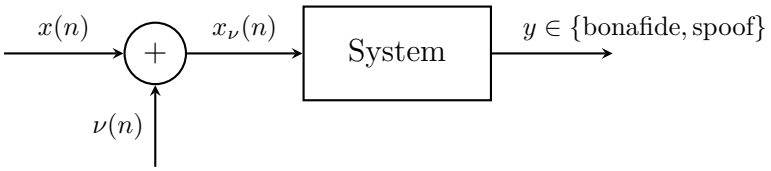
In addition, we want to investigate whether it is possible to solve the aforementioned problem in the presence of noisy inputs. Therefore, instead of the sole recording $x(n)$, we also consider the case in which the input of our system is:

$$x_\nu(n) = x(n) + \nu(n), \tag{3.1}$$

where $\nu(n)$ represents an additive noise term as per the model presented in Chapter 1 (see Figure 3.1b).



(a) Clean signal problem formulation.



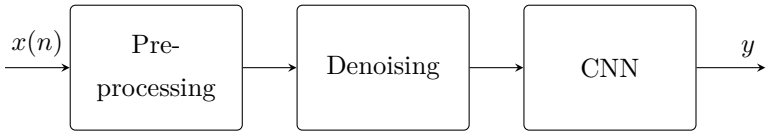
(b) Noisy signal problem formulation.

Figure 3.1: Schematic representation of the speech spoof detection problem considered in this work, in case of clean (a) or noisy signals (b).

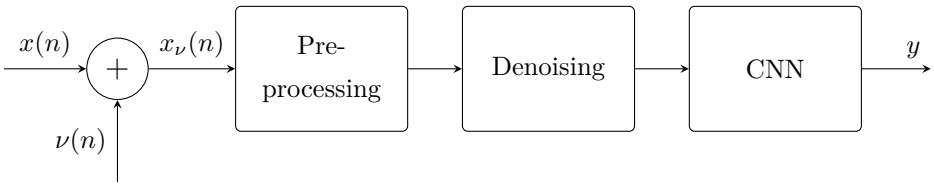
In the next section we propose different solutions based on a CNN and we describe how the proposed system works.

3.2 Proposed Pipeline

In this section we present the pipeline we employ to solve the problem we just defined. The aforementioned audio recording is used as input of a pre-processing system, that has the role of transforming the audio signal into a logarithmic mel-spectrogram. This signal is then fed to the denoising stage of our pipeline. The output of this block is the input of the last block, which is the classification one, executed by the CNN, outputting the classification result. This general pipeline is shown in Figure 3.2.



(a) Pipeline with clean input.



(b) Pipeline with additive noise.

Figure 3.2: Clean (a) and noisy (b) basic pipeline.

Starting from this essential schematic, we turn on and off different sections of it, to fit the solutions we chose to pursue. The pre-processing stage is common to all of our implementations, therefore moving forward we will consider the output of this block as the input of all the representations, and we will denote it as $S(n)$, considering both the pre-processed clean recordings and the noisy ones.
In general, the different settings we employ are the following:

- CNN - We evaluate the performances of the sole CNN used as a classifier. The network is fed with the logarithmic mel-spectrogram of the recording under analysis and it outputs the result of the binary classification.

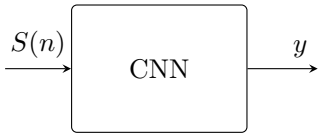


Figure 3.3: Standalone CNN.

- Independent denoising and CNN - Each recording is denoised first and then fed to the binary classifier network. This is done to help the CNN's processing of noisy recordings.

- Joint denoising and CNN - The classifier and the denoiser are directly connected and trained altogether in order to process the

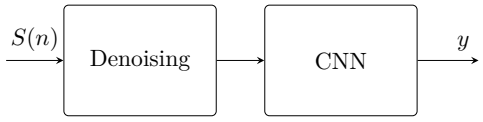


Figure 3.4: Denoising stage disjointed from the CNN.

input. This approach improves the previous one, as the denoiser is tailored to the used classifier.

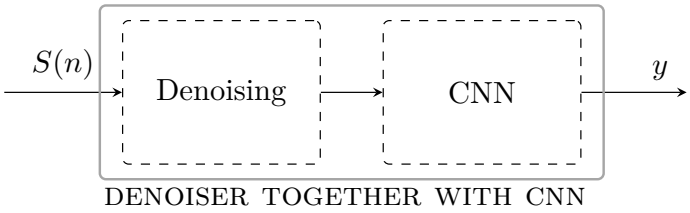


Figure 3.5: Denoiser training jointly with the CNN.

In the next sections we describe the building blocks composing these pipelines, followed by the training strategies.

3.2.1 Building Blocks

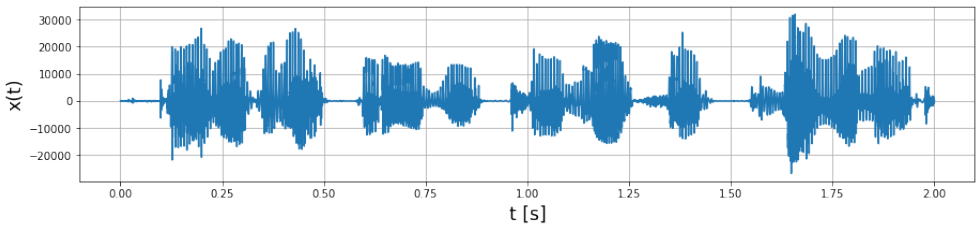
In this section we explain in depth how the blocks composing our pipeline work and what their purpose is for the means of this thesis. We start from a step necessary to use the CNN we intend to implement, the pre-processing stage. This stage turns the speech recordings into log-mel spectrogram, hence 2D images-like representations. We then proceed to describe all the filters we want to denoise the pre-processed signals with. We want to maintain the process fixed for all the different solutions we propose for our method, therefore the aforementioned image shaped signals are used as inputs of each denoising filter. To support our idea, we draw inspiration from [50], where a Non-Local Means algorithm is used on speech spectrograms to improve the noise robustness in speech recognition. The main intent in choosing multiple classical signal processing denoisers relies on our debate regarding whether these can be sufficient in solving our degradation issue in the classification problem, or if it is necessary to employ data driven strategies, to pair up with the CNN. In the end, we explain how the CNN chosen for this thesis, the VGGish, works in general.

3.2.1.1 Pre-Processing

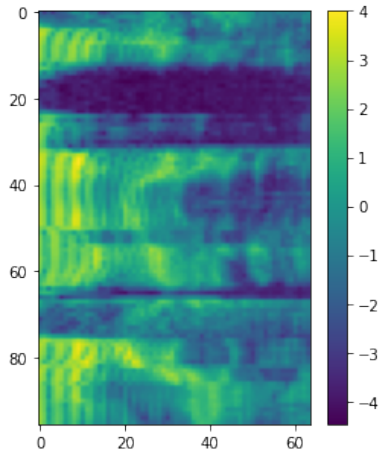
Many audio classification tasks are solved by means of data-driven techniques. Classic old-fashioned solutions typically involve a feature extraction step followed by a classification step [25, 26, 47]. Conversely, since the rise of deep learning, multiple end-to-end methods that analyze the input audio stream and directly provide a classification result have been proposed e.g. in [23, 29]. Many of these methods exploit Convolutional Neural Networks applied to a time-frequency representation of the audio signal, rather than directly using the audio waveform [30]. For this reason, our pipeline starts with a pre-processing step that turns an input audio recording into a 2D time-frequency representation.

This is done by loading the voice recordings, shown in Figure 3.6a and extracting from them an N×M log-mel spectrogram. What was initially a speech recording is now an image, just like we can see in Figure 3.6b. Formally, given the audio track under analysis x, we compute the log-mel spectrogram S following this procedure:

- We compute the STFT of x on N different windows and we take its magnitude X.
- We compute a Mel filter-bank of length M, compatible with the length of the window of the just calculated STFT, f_{mel}.
- We create the Mel spectrogram computing the scalar product between X and f_{mel}.
- We apply the natural logarithm to the last result, thus obtaining $S = \log(X \cdot f_{mel})$, the log-mel spectrogram.



(a) Audio track loaded without any kind of processing.



(b) Log-Mel spectrogram of the above recording.

Figure 3.6: Audio track before (a) and after (b) pre-processing.

3.2.1.2 Denoisers: Total Variation

The first type of image denoiser we take into account is the one presented by Chambolle in [51], where a fairly fast algorithm is proposed to solve the same problem described in [52] by Rudin, Osher and Fatemi.

In the latter, the authors explain their aim as to denoise images by minimizing the total variation norm of the estimated solution. To do so they derive a constrained minimization algorithm as a time dependent non-linear Partial Differential Equation (PDE), where the constraints are determined by the noise statistics. Basically, let us consider a noisy image $v(x, y)$ with $x, y \in \Omega$, where Ω is the surface of the image and x, y represent the spatial coordinates on it. They suggest to retrieve the original image $u(x, y)$ by solving this minimization problem:

$$\arg \min_{\hat{u} \in \mathrm{BV}(\Omega)} \|x\|_{\mathrm{TV}(\Omega)} + \frac{\lambda}{2} \int_{\Omega} (v(x, y) - \hat{u}(x, y))^2 \, dx \, dy, \tag{3.2}$$

where $\hat{u}(x, y)$ represents the denoised image, $\mathrm{BV}(\Omega)$ is the set of functions with bounded variation over the domain Ω and $\mathrm{TV}(\Omega)$ represents the total variation over the domain. λ is a given Lagrange multiplier: it is related to the aforementioned noise statistics and controls the degree of filtering of the obtained solution, [53]. The minimum of these equations exists and it is unique. In Figure 3.7 we display an example of how this denoiser works when applied to spectrograms.

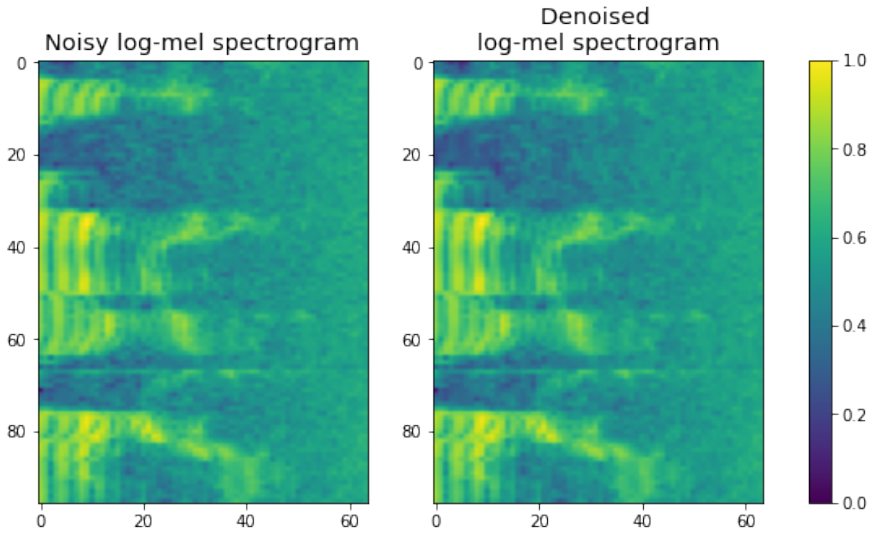


Figure 3.7: Spectrograms before and after Total Variation denoising.

3.2.1.3 Denoisers: Wavelet

This denoiser uses wavelet transform and thresholding. This is a non-linear technique that is considered simple, since it uses one wavelet coefficient at a time [54], yet effective. Starting from Donoho and Johnstone's study in [55], we can explain this method as follows.

Let us denote $\mathbf{Y} = \mathcal{W}\, v(x, y)$ as the matrix of wavelet coefficients, obtained by applying the two-dimensional dyadic orthogonal wavelet transform operator $\mathcal{W}$ to the noisy image $v(x, y)$. Subsequently, one of the computed wavelet coefficients and a given threshold are compared and, if the first is less than said threshold, it is set to zero, otherwise it is left as is or modified, depending on the thresholding rule. This way, the threshold acts as a parameter that distinguishes between insignificant coefficients, likely due to noise, and significant ones. Thresholding basically creates a region around zero, where the coefficients are considered negligible. Outside of this region, the thresholded coefficients are kept to full precision.

The thresholding rules taken into account are: *VisuShrink* from [55] and *BayesShrink* thresholding method, proposed in [54]. The latter is implemented by using a single algorithm for both denoising and compression. The authors suggest to use lossy compression for denoising because of the structural correlations usually present in pristine signals, that can

be exploited by coders to get a concise representation. White noise does not have this characteristic, it is not easily compressible, therefore a good compression model can yield a good model to distinguish signal from noise. In Figure 3.8 an example of the output of this denoiser with a spectrogram as input.

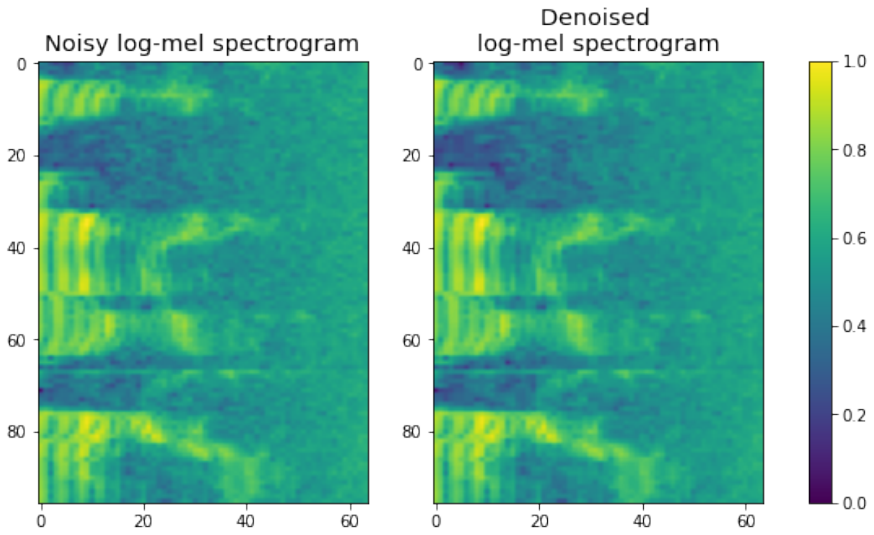


Figure 3.8: Spectrograms before and after wavelet denoising.

3.2.1.4 Denoisers: Bilateral

Bilateral filtering, proposed by Tomasi and Manduchi in [56], smooths images preserving their edges by means of a non-linear combination of nearby image values. In order to do so, it uses both spatial similarity and range similarity, combining them. It is a neighboring filter, therefore within the chosen area of pixels, the weight assigned to these decreases proportionally to the distance from the pixel we are trying to fix. In addition to this traditional way of filtering, the authors implement the concept for colour similarity as well. Hence, as the range of similarity drops, the weight of the corresponding pixel drops too. It was demonstrated that the sole use of the range similarity would only remap the colors in the image, whilst in combination with the domain one, it manages to maintain the edges of objects. In [56] is moreover shown that this kind of filter works well both for black and white and coloured images, working in the CIE-Lab color space. An example of the results obtained

using this denoiser on spectrograms is shown in Figure 3.9.

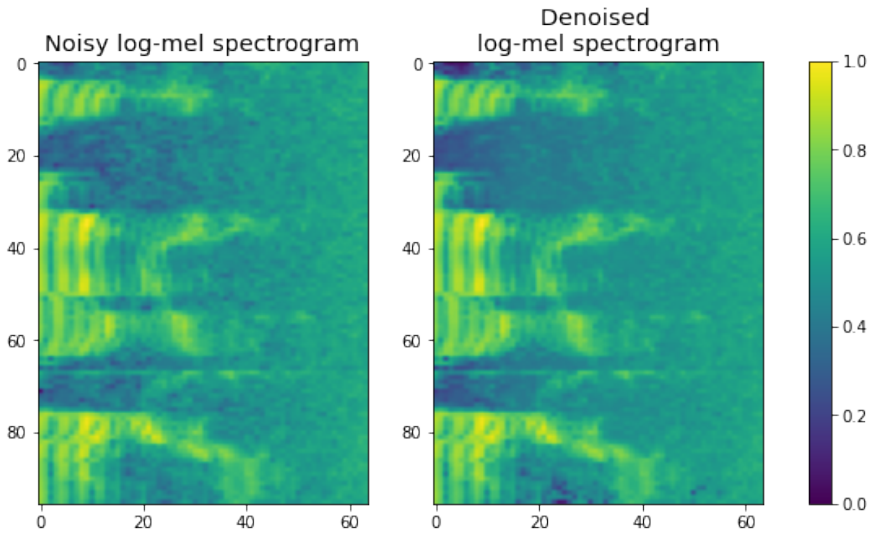


Figure 3.9: Spectrograms before and after bilateral denoising.

3.2.1.5 Denoisers: Non-Local Means

The fourth algorithm we choose to use to clean our noisy data is the Non-Local Means for denoising images, introduced in [53]. Just like in many algorithms used on images, here the denoising is obtained by averaging. Although, while the bilateral algorithm computes the average over the gray levels of a specified region, Non-Local Means has a different strategy. Given a noisy image, $v = \{v(i) |\ i \in \Omega\}$, where i is the considered pixel positioned in coordinates $x, y \in \Omega$, the estimated value NL$[v](i)$ for i is computed as the weighted average of all the pixels in the image. The family of weights $w(i, j)$ depends on the similarity between the pixels i and j and this similarity is evaluated as a decreasing function of the weighted Euclidean distance.

What makes this filter different from other local filters or frequency domain filters, is the use of all available self-predictions in the image. In addition to comparing the gray level in a single point, it compares the geometrical configuration in a whole neighborhood too, making this algorithm more robust than neighborhood filters, [53]. Figure 3.10 reports an example of functioning of this denoiser with spectrograms.

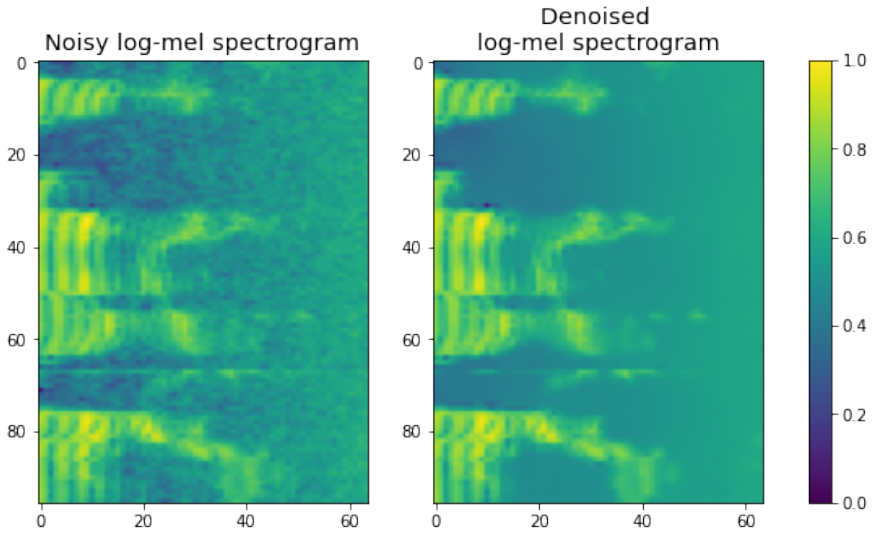


Figure 3.10: Spectrograms before and after Non-Local Means denoising.

3.2.1.6 Denoisers: DnCNN

The DnCNN net was firstly introduced in [4], it is a Deep CNN that works with residual learning. This means that rather than outputting directly the cleaned image $\hat{u}$, the system is designed to predict the residual $\hat{v}$, that is the difference between the noisy observation and the clean image. The net integrates both residual learning, for the denoising problem, and batch normalization, to speed up the training. This implementation is reported to be computationally efficient and to achieve effective denoising. The general architecture of the DnCNN is shown in Figure 3.11. In

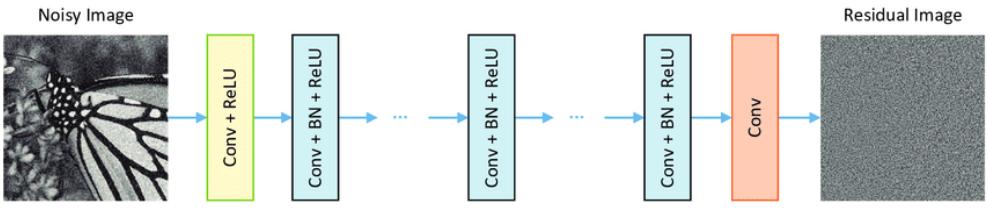


Figure 3.11: Architecture of the DnCNN proposed in [4].

particular, considering a DnCNN with depth D the network is composed by the following layers:

- One 2D convolutional layer with 64 filters of size $3 \times 3 \times c$, where c represents the image's number of channels, therefore $c = 1$ for grey images and $c = 3$ for coloured ones. This is followed by *ReLu*, used

for non-linearity.

- $(D-2)$ blocks composed by the series of a 2D convolutional layer, Batch Normalization and *ReLu*. The convolutional layer has 64 filters of size $3 \times 3 \times 64$.
- One last 2D convolutional layer with c filters of size $3 \times 3 \times 64$ utilized to reconstruct the output.
- One subtraction layer that compares the input image and the output of the previous layer, outputting the residual image.

The depth of the net depends on the noise level of the analyzed image. In Figure 3.13 there is an example of the DnCNN's architecture with input shape $96 \times 64 \times 1$.

In Figure 3.12 we present an example of the DnCNN network applied to a spectrogram.

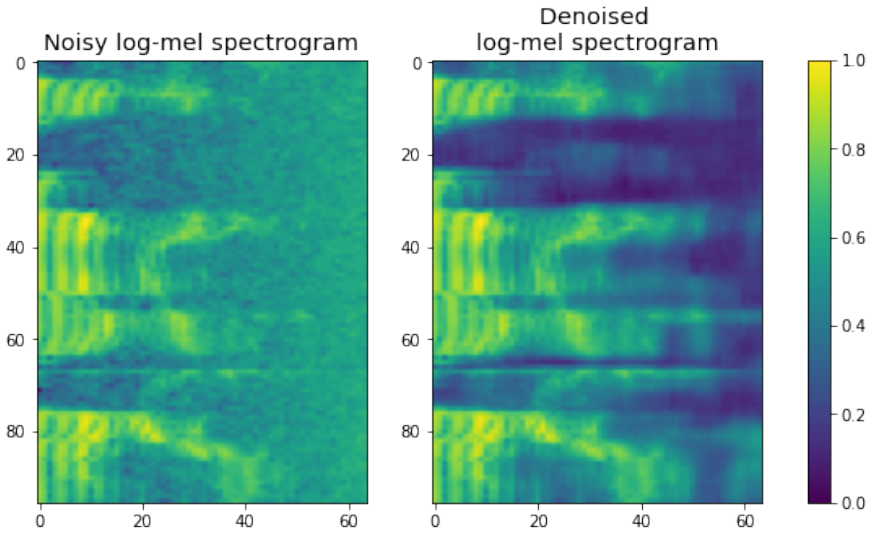


Figure 3.12: Spectrograms before and after DnCNN denoising.

3.2.1.7 Classifier: VGGish

Created to classify audio artifacts, the VGGish net is a pretrained CNN based on the image classification net VGG [57]. As reported in [30], the dataset which the net is pretrained on is called YouTube 100-M: it contains 70 million videos, for a total of 5.24 million hours, each video

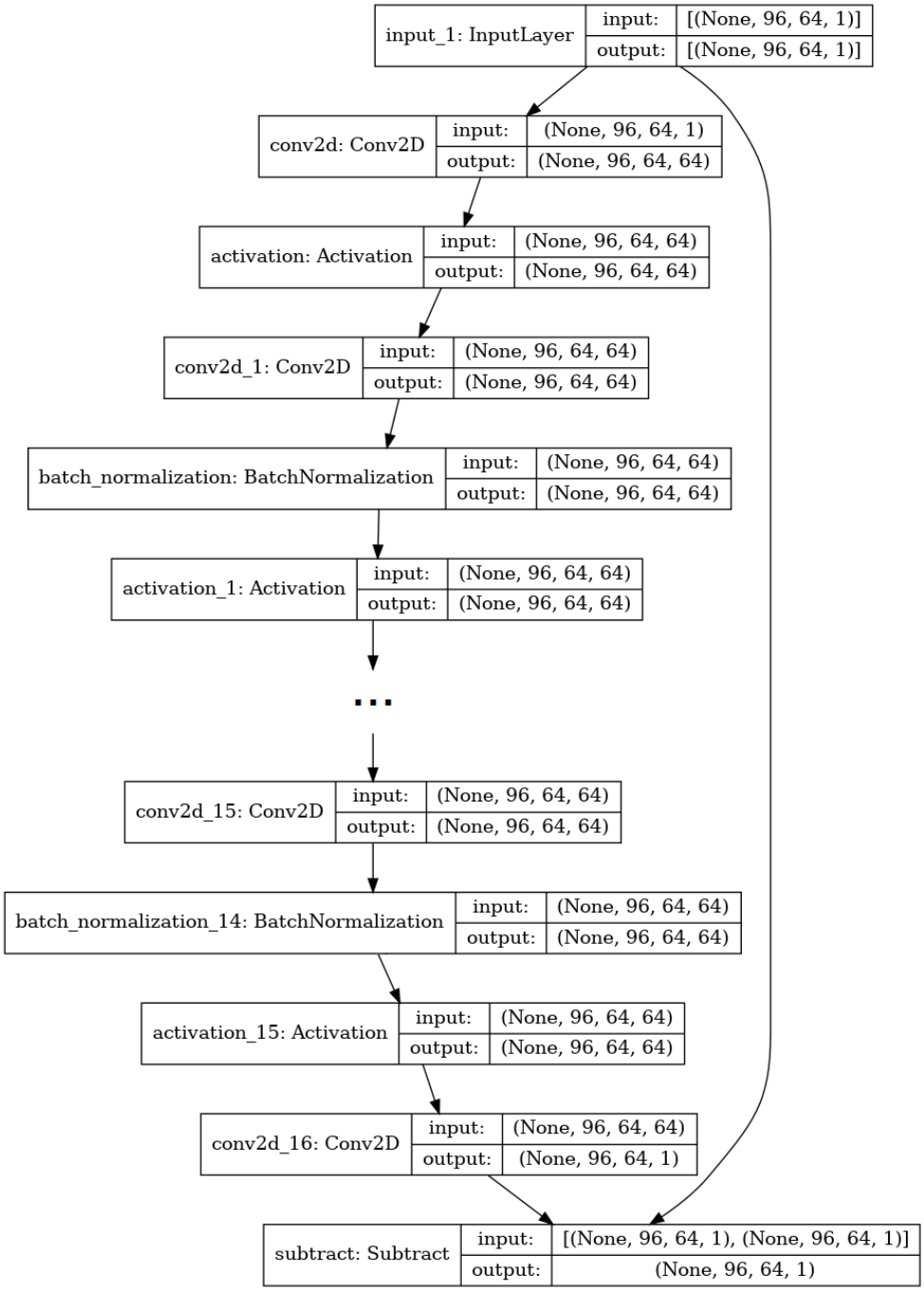


Figure 3.13: Architecture of the DnCNN.

tagged from a set of 30871 labels. As stated by the authors of the latter mentioned article: “Our dataset’s size allows us to examine networks with large model capacity, fully exploiting ideas from the image classification literature”. They in fact use modified image oriented classification CNNs in order to compare their performances on the problem of speech recognition. To do so they compute log-mel spectrograms of multiple

frames, thus creating the 2D image-like signals to use as input of the net. The VGGish net consists mainly of convolutional layers followed by maxpooling ones and concludes with fully connected layers.

We proceed now in describing more specifically the network's architecture. The presented model defines layers up to and including a 128-wide embedding layer that does not include a final non-linear activation, so the embedding value is considered in its pre-activation stage. Every 2D convolutive layer has a $3 \times 3 \times F$ shape, where F represents the number of filters used in the layer. Moreover, all the 2D maxpool layers have 2×2 pool sizes and strides equal to 2, and all activations are *ReLu*. Taking into account the aforementioned information, the VGGish is structured as follows:

- one 2D convolutive layer with 64 filters, followed by a 2D maxpool layer;
- a series of 2D convolutive layers with 128 filters each and a 2D maxpool layer;
- two 2D convolutive layers, both with 256 filters, followed by a 2D maxpool layer;
- two 2D convolutive layers, both with 512 filters, followed by a 2D maxpool layer;
- one flattening layer;
- two fully connected layers with 4096 output nodes;
- one fully connected layer with 128 output nodes.

In Figure 3.14 we display the whole architecture.

3.2.2 Training Strategies

In the next sections we explain how the building blocks are combined to obtain the multiple training pipelines. We begin by describing the classification net, VGGish, used by itself either on the dataset as is or on the denoised one. We then move forward talking about the DnCNN used

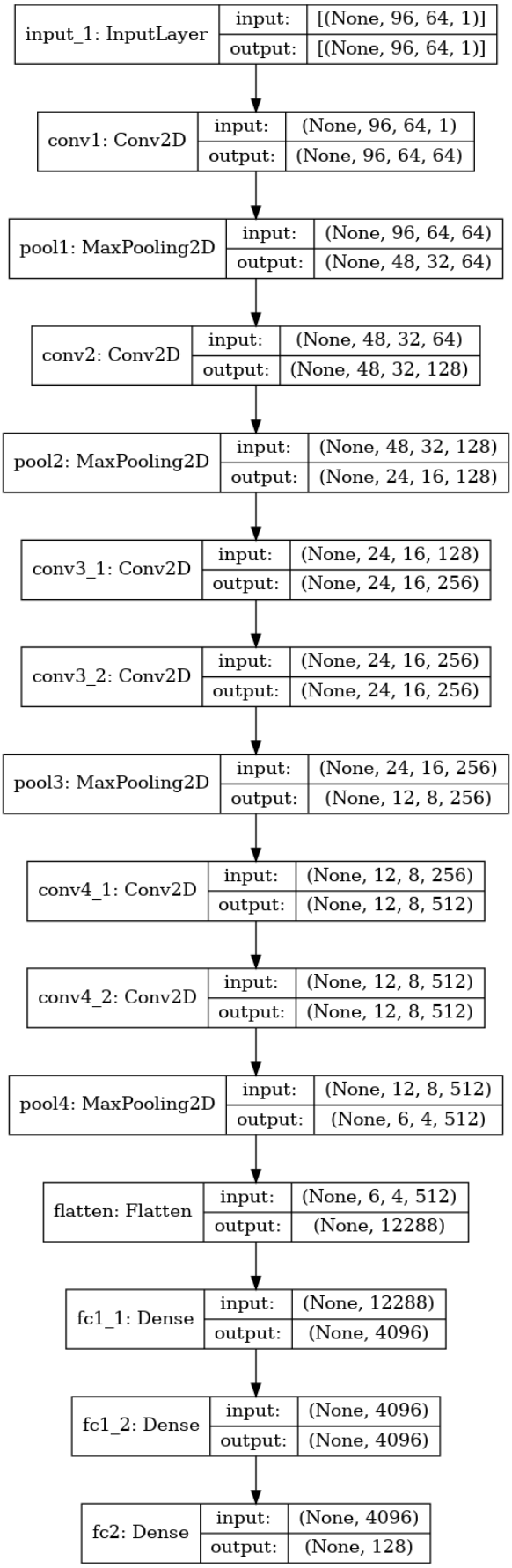


Figure 3.14: Structure of the VGGish network.

as denoiser and we describe its use in our solution. In the end we propose the third training strategy we adopt: the two nets combined together to create an end-to-end system.

3.2.2.1 Standalone VGGish

This is the training from which we start our experiments. Starting from the baseline VGGish architecture displayed in Figure 3.14, we modify it to align it to our purposes. The problem we focus on is distinguishing the *bonafide* excerpts from the *spoof* ones, therefore we add to the original last embedding layer a dense layer with *softmax* activation and 2 output nodes, turning the net into a binary classifier just as we desired. Depending on the considered scenario, the input of this net is:

- the pre-processed clean dataset;
- the pre-processed noisy datasets;
- the denoisers' output.

To compute the loss (l_{vggish}), we use the binary cross-entropy loss function. The cross entropy between two probability distributions p and q is the average number of bits needed to encode data coming from a source with distribution p when we use model q, [58]. Given our binary discrete problem, let us identify p_i as the true probability and the given distribution q_i as the predicted value of the current model, where i represents the class taken into account. Let us then consider $p \in \{y, 1-y\}$ and the model's probability of $y = 1$ as following:

$$q_{y=1} = \hat{y}. \tag{3.3}$$

Hence, for $y = 0$:

$$q_{y=0} = 1 - \hat{y}. \tag{3.4}$$

Consequently, $q \in \{\hat{y}, 1-\hat{y}\}$ and we can use the cross entropy as a measure of dissimilarity between p and q, using this formula:

$$l_{vggish} = H(p,q) = -\sum_i p_i \ \log(q_i) = -\hat{y}\log\hat{y} - (1-y)\log(1-\hat{y}). \tag{3.5}$$

3.2.2.2 Standalone DnCNN

As explained above, we use the DnCNN as a data driven denoiser. In this scenario it is used by itself, not considering that its output may be used for classification afterwards. The network is therefore trained to map noisy log-mel spectrograms into their clean noiseless version. In order to accomplish this, the output of the net has to have the same shape as the input. In particular, for our net we consider a depth $D = 17$, same as the one chosen in [4] for Gaussian denoising.

The loss we consider in this case (l_{dncnn}) is the Mean Squared Error (MSE) loss function. This loss function measures the expected squared distance between the predictions given from our predictor for a specific value, and what the true value really is. The MSE therefore gives us an approximation of the similarity between the predictions and the original signal. Let us consider y_i as the correct label for the i-th element of the n-elements-long array composing the input of the network; we identify as $\hat{y}_i$ the prediction of the same sample. Therefore, the MSE is defined as:

$$l_{dncnn} = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3.6}$$

3.2.2.3 End-To-End System

The last case of training strategies we present is the end-to-end one. In this scenario we connect the DnCNN output to the VGGish input, thus obtaining a single network. We start the training from pre-trained models for both the DnCNN and the VGGish. The datasets fed to this net are the unfiltered clean and noisy ones. As we are interested in controlling both the denoising effect and the classification accuracy, we use a loss function that considers two inputs: the binary classification prediction and a log-mel spectrogram, with the same shape of the input. More specifically, let us define the output of the DnCNN as *dncnn_output* and the output of the last VGGish layer as *vggish_output*, both shown in Figure 3.15. Now, we identify the loss related to the *vggish_output* as l_{vggish}, computed with the binary classification function described above, and the one related to the *dncnn_output* as l_{dncnn}, computed with the

aforementioned MSE loss function.

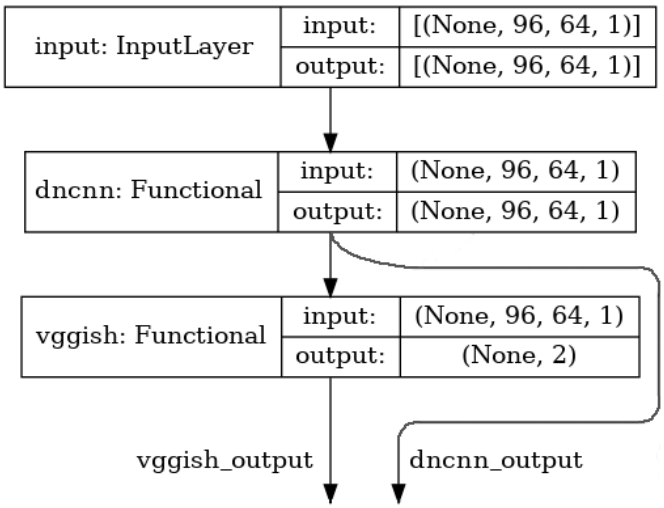


Figure 3.15: End-to-end network architecture, presenting one input and two outputs.

The loss function of the end-to-end system is a weighted sum of l_{vggish} and l_{dncnn}.

$$l_{end-to-end} = \omega_{dncnn} \cdot l_{dncnn} + \omega_{vggish} \cdot l_{vggish}, \tag{3.7}$$

where ω_{dncnn} is the weight factor assigned to l_{vggish}, and ω_{vggish} is the weight factor assigned to l_{dncnn}. In particular, ω_{dncnn} is computed by dividing the mean value of the losses of the pretrained VGGish by the one of the losses of the pretrained DnCNN, while ω_{vggish} is equal to 1.

4

Experimental Setup

In this chapter we break down all the experiments putting into action the just described method. We first thoroughly explain the creation of the datasets, their origin, the differences amongst them, and we describe the use we are going to make of them in each different training strategy. We then proceed to report all the details about the training setup, i.e. all the parameters we used in our experiments.

4.1 Dataset

In this section we show in detail how the datasets are created, where they come from and how they are used for the experimental part of the thesis. We already introduced in Chapter 3 the difference between the clean and noisy datasets, but we are now going to explain how each noisy dataset was created and all the different types of dataset we use.

4.1.1 ASVspoof

The main dataset used in our experiments draws inspiration from the dataset ASVSpoof 2019, created and described in [5]. The article displays the specifics of the datasets created for the ASVSpoof event, aimed to set up platforms to evaluate different spoofing countermeasures solutions. The spoofing attacks considered for the challenge are: speech synthesis, voice conversion and replay, all described in Chapter 2. The said dataset was created to be suited for the study of ASV replay spoofing and countermeasures, and fake audio detection, as well.

The ASVspoof 2019 database is based upon the Voice Cloning Toolkit (VCTK) corpus [11], a multi-speaker English speech database recorded in a hemi-anechoic chamber at a sampling rate of 96 kHz. The speech database was built using recordings from 107 speakers (46 male, 61 female), with sampling frequency equal to 16 KHz. Two different case scenarios are taken into account: logical access and physical access. Logical access (LA) control infers a situation where a system protected by ASV is being reached by a remote user. It is assumed in this case that the microphone in use is chosen by the user, and not controlled by the authentication system designer. Conversely, physical access implies the use of ASV to protect access to a space or facility. Therefore, in this scenario the authentication system is in control of the microphone, not the user. We can observe the difference between the two scenarios in Figure 4.1.

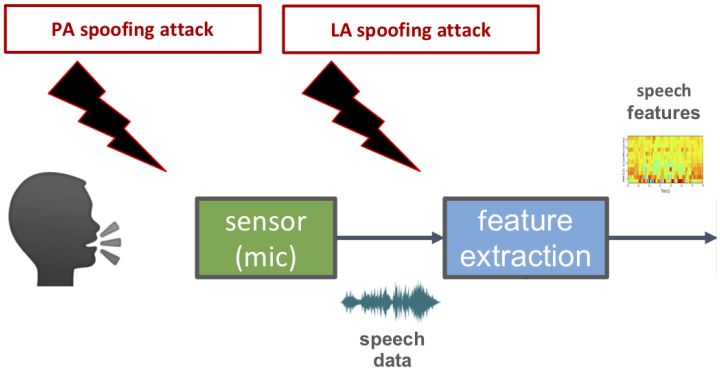


Figure 4.1: Difference between the LA and PA scenarios, from [5].

For this thesis we decided to focus only on countermeasures against

LA attacks, therefore we are going to describe in detail the portion of dataset designed for that task. In this case attacks are presented to the ASV system in a worst-case, post-sensor scenario. These attacks take the form of synthetic speech, referred to as TTS, or VC algorithms, proposed to the ASV system with no convolutive acoustic propagation nor microphone effects. As described in Chapter 2, TTS algorithms convert text input into speech, while VC ones transform input coming from a speech source into target speech. These two algorithms can be combined, too.

Furthermore, since the countermeasure systems should be able to detect also unseen attack strategies, the dataset's attack algorithms are categorized as *known* or *unknown*. Known attacks are the ones used to generate spoofed utterances in the training and validation partitions. On the other hand, the evaluation partition is composed by the mix of 2 known attacks and 11 unknown ones.

In Table 4.1 we show the exact partitions of the dataset, the number of utterances the mentioned attacks were applied on to create the spoofed section of each partition, and the different classes of algorithms corresponding to the attacks. All the three partitions of data (training, validation and evaluation) include also bonafide tracks, i.e., real and genuine speech signals, in different amounts. As reported in Table 4.1, the spoofed part of the database is divided as follows. The training data comprises spoofed utterances generated using four TTS and two VC algorithms, identified as classes of known attacks A01-A06. The validation's data spoofed partition is generated with the same algorithms used in the training set. The spoofed part of the evaluation dataset is built on spoofed utterances created using seven TTS algorithms and six VC ones; in this case we refer to the different algorithms as classes A07-A19.

Every class we display in Table 4.1 represents a different spoofing algorithm. Generally, all these algorithms are characterized by the same processing stages:

- an input processor, usually converting text into sequences of linguistic features such as phone and pitch-accent labels;
- a duration model, predicting the duration of context-dependent

	Training set	**Validation set**	**Evaluation set**
Bonafide samples	2580	2548	7355
Source for spoofed samples	4000	4000	13400
Spoofed samples	22800	22296	63882
Spoofing classes	A01, A02, A03, A04, A05, A06	A01, A02, A03, A04, A05, A06	A07, A08, A09, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19

Table 4.1: Partitioning of the LA database, with number of samples for each section and type of spoof algorithm.

phones;

- an acoustic model, that has the role of predicting a sequence of acoustic features;
- a waveform generator, that creates the audio waveform we can listen to.

We now give an overview on each algorithm:

A01 It is a NN-based TTS system. Its input processor is the Festival_Lite (Flite) [59]. It then uses a HMM-based duration model, a combination of a shallow autoregressive (SAR) mixture density network [60] and variational auto-encoder (VAE) [61] as acoustic model, and finally a WaveNet [15] vocoder as waveform generator.

A02 This is an NN-based TTS system similar to A01 except that the WORLD vocoder [14] is the one responsible for the waveforming.

A03 It is a NN-based TTS system similar to A02. The input processor chosen for this algorithm is the Festival and both the duration and

acoustic models are based on the HTS style features, [62, 63]. The waveform generator is the same as A02.

A04 This algorithm is a waveform concatenation TTS system based on the MaryTTS platform [13]. The acoustic model is performed by classification and regression trees (CARTs), that predict prosodic target features. The waveform generation is performed using standard diphone unit selection, with target and join cost coefficients.

A05 This is the first of the two VC spoofing algorithms in the known attacks. It is an NN-based VC system. It uses the WORLD vocoder both for input processing and waverform generator, there is no duration model, and a VAE [64] is used for the acoustic model.

A06 It is a transfer-function-based VC system [65]. The input processor of this algorithm is a source-filter model, the signal is then analyzed and the filters replaced in order to correspond to the target speaker. Following this modification, the signal is re-synthesized making use of the original residual signals and the new filters, via an overlap-add technique.

A07 This is a NN-based TTS. The first step of this algorithm is similar to A03. Both the duration and the acoustic models are RNN-based. The waverform generator used in this case is the WORLD vocoder, afterwards the WaveCycleGAN2 [66], a time-domain neural post-filter, is used to transform the output waveform into a natural-sounding waveform.

A08 It is a NN-based system similar to A01. However, in this implementation the VAE acoustic model is replaced by a one-hot speaker vector. Furthermore, a neural-source-filter waveform model [67] is used as waveform generator, instead of the WaveNet.

A09 This algorithm is a NN-based statistical parametric speech synthesis (SPSS) TTS system [68]. It uses RNNs for both the duration model and the acoustic one, that predicts the acoustic features that are then used by a Vocaine [69] vocoder to generate waveforms.

A010 It is an end-to-end NN-based TTS system [70], that applies transfer learning from speaker verification to a neural TTS system called Tacotron 2 [71]. For the waveform generator, WaveRNN [72] was used.

A011 This system is the same as A10, except for it using Griffn-Lim algorithm [73] to generate waveforms.

A012 This algorithm is a neural TTS system based on the AR WaveNet [15]. It is fairly similar to A09 except for the waveform generator, that is the aforementioned AR WaveNet.

A013 It is a combined NN-based VC and TTS system. It first makes use of a TTS system, and its outputs are then fed to the VC system. Direct waveform modification was used to generate the output waveform.

A014 This algorithm is a combination of TTS and VC as well, and it works in a way similar to the previous one. In the VC part, bottleneck features are first extracted from the waveforms of the source speaker via an automatic speech recognition (ASR) model, in order to acquire linguistic-related embedding vectors automatically. A RNN-based system is used as acoustic model. For waveform reconstruction, the STRAIGHT [74] vocoder is used.

A015 It is a combined TTS and VC system fairly similar to A14. The main difference lays in the waveform generator: in this case the algorithm uses speaker-dependent WaveNet vocoders.

A016 This is a waveform concatenation TTS system that uses the same algorithm as A04. The sole difference is that it was built using the dataset for the evaluation partition of the dataset, instead of the training and validation ones.

A017 It is a NN-based VC system that uses the same VAE-based framework as A05. Differently from A05, it uses a generalized direct waveform modification method [75, 76] for waveform generation.

A018 This is a non-parallel VC system [77], inspired by the standard i-vector framework [78, 79], used in text-independent ASV. The mechanism of the new vocoder used as waveform generator is similar to the transfer-function-based approach in A06.

A019 It is a transfer-function-based VC system using the same algorithm as A06. Differently from A06, the data fed to the system is taken from the evaluation partition of the database set for creating the spoofed utterances.

We summarized how the algorithms were constructed, to discuss the differences on performance in the following chapter; further details on every algorithm class can be found in [5].

Now that we explained the origin of our dataset, we proceed to explain how we used it to test the solutions proposed in the previous chapter.

4.1.2 Augmentation

We want to simulate a situation in which the recording is not obtained in optimal conditions, to see how our system performs in all the different scenarios described in Chapter 3.

In order to do so, we create four new datasets starting from the clean one described in the previous section: in three of these datasets we add Gaussian additive noise to every speech utterance, the last one presents a mixture of clean and noisy samples.

To create the aforementioned three datasets we use Python's library *Audiomentations* [80]. This library lets us select the amount of noise we want to add to the signal in order to lower its SNR. The noise signal is created by computing a Gaussian distribution of the same length of the clean sample. The width of this Gaussian distribution depends on the SNR we want for our final signal, and the standard deviation of the signal itself. Once the noise signal is computed, it is simply added to the clean one, creating the noisy signal, as described in Chapter 1.

In (1.6) we defined the SNR as:

$$\mathrm{SNR} = \frac{\sigma_x^2}{\sigma_\nu^2}, \tag{4.1}$$

where σ_x and σ_ν are respectively the standard deviation of the clean signal and of the noise signal's one. The SNR can be expressed in dB by applying:

$$\text{SNR}_{dB} = 10 \log_{10} \text{SNR}. \tag{4.2}$$

Each noise signal is obtained from the Gaussian distribution $\mathcal{N} \sim (0, \sigma_\nu^2)$, having standard deviation:

$$\sigma_\nu = \text{SNR}^{-\frac{1}{2}} \cdot \sigma_x = 10^{-\frac{\text{SNR}_{dB}}{20}} \cdot \sigma_x. \tag{4.3}$$

We empirically select three values for the SNR_{dB}, to construct the low, medium and high SNR datasets.

4.1.2.1 Low SNR

We refer to the *low* SNR dataset as the one with the lowest SNR among the rest. For this dataset, each noisy signal presents: $\text{SNR}_{dB} = 15\text{dB}$. In Figure 4.2 we can observe an original clean recording, while in Figure 4.3 we show the augmented version of the same excerpt. Comparing the two pictures,we can clearly note the addition of noise in the latter, evident especially in the silence portions of the signal.

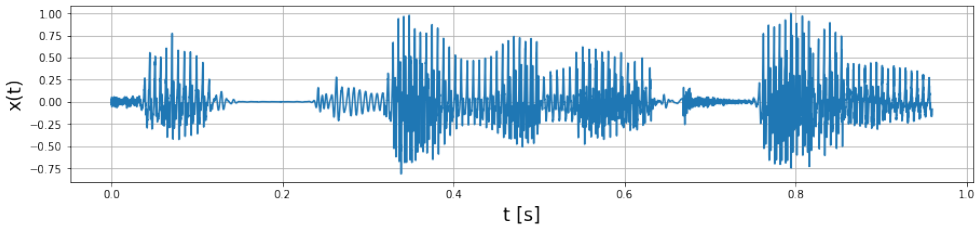


Figure 4.2: Clean audio track.

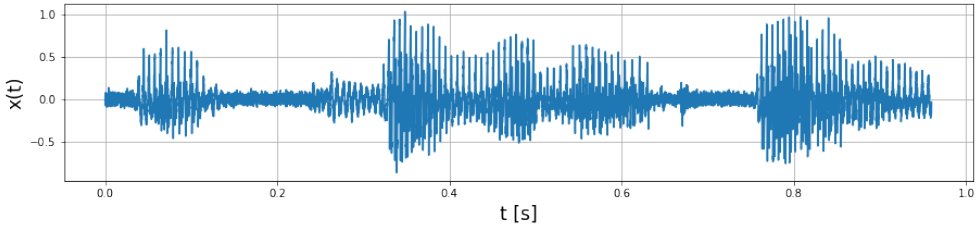


Figure 4.3: Low SNR audio track

.

4.1.2.2 Medium SNR

For the medium SNR, the chosen SNR_{dB} is equal to 20 dB and the resulting track is shown in Figure 4.4. Comparing this with Figure 4.3 we can see that the amplitude of the noise added to the recording is lower with respect to the latter mentioned track.

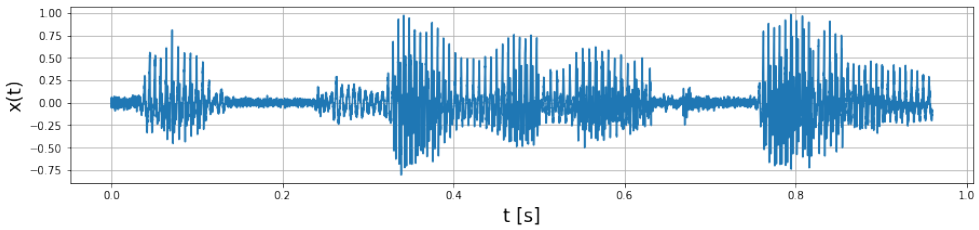


Figure 4.4: Medium SNR audio track.

4.1.2.3 High SNR

This dataset is the one who has the higher SNR among the noisy datasets. The chosen SNR_{dB} is 25 dB. An example with the representation of a noisy track with this SNR is displayed in Figure 4.5. As expected, this track demonstrates to be the closest one to the original one, presented in Figure 4.2.

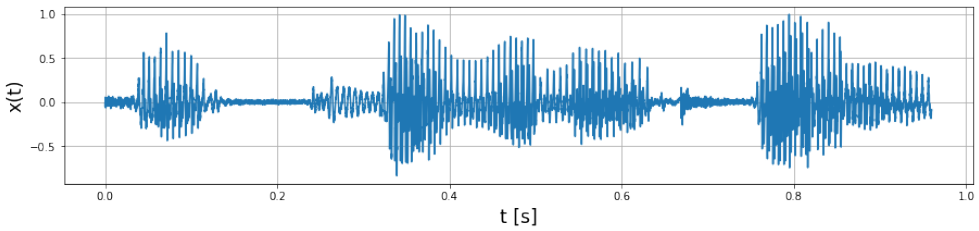


Figure 4.5: High SNR audio track.

In Figure 4.6 we also report an example of the pre-processed version of the same track analyzed in Figures 4.2, 4.3, 4.4, 4.5, in order to show the evident differences between the log-mel spectrograms belonging to the different datasets.

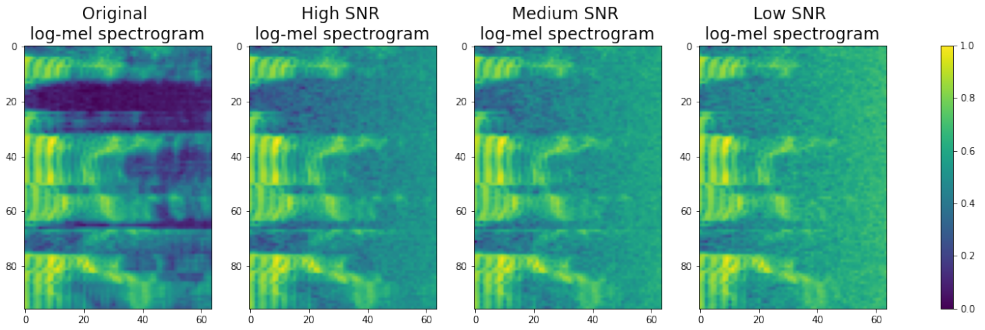


Figure 4.6: Example of pre-processed sample track for each type of dataset.

4.1.2.4 Augmented Dataset

We created this dataset to test the system and compare its performances, when trained with a database that contains a combination of both clean and noisy excerpts.

To create the augmented dataset, we copied an equal amount of samples randomly selected from all the prior explained datasets, i.e., the clean dataset and the three Gaussian noise augmented datasets. We want to ensure that each speech track is selected only once from the four datasets. Therefore, since the noisy datasets are created with the exact same samples from the clean one, to create this mixed dataset we follow these steps:

- We shuffle the rows of the table containing all the samples filenames, spoofing algorithms and binary labels.
- We separate the rows grouping them by the different classes of algorithms. We therefore obtain a number of rows subsets equal to 1 bonafide class plus 6 spoofed classes for training and validation partitions, and $1 + 12$ spoofed classes for the evaluation one.
- We assign to one fourth of each subset the path to the different datasets.
- We finally copy the samples into the new augmented dataset, getting the samples from the path written in the table.

The amount of samples for each algorithm is not always divisible by the number of type of datasets, 4, therefore the augmented dataset is

slightly smaller than the previous described ones. In Figure 4.7 we report an example of the way we select the samples to create the augmented dataset, in this case for training and validation datasets.

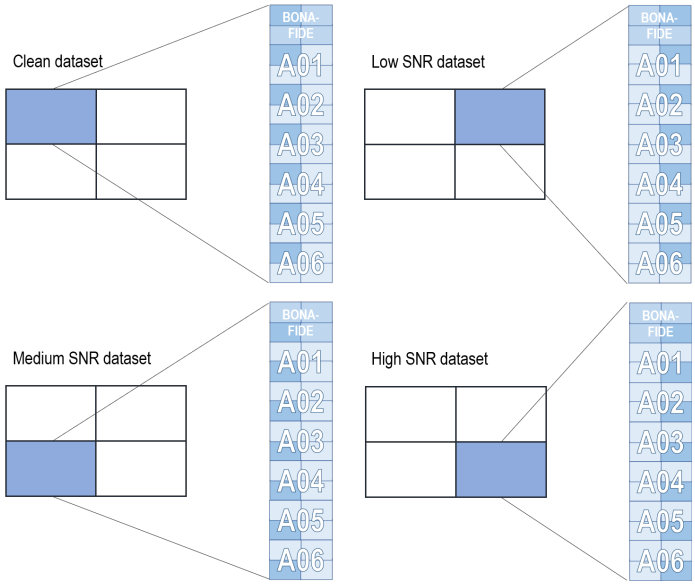


Figure 4.7: Example of the construction of the training and validation augmented datasets.

4.2 Training Setup

In this section we present in detail the setup used to perform the experiments, from the dataset balancing techniques, to the technical details such as the used learning rate for each different training.

4.2.1 Dataset Imbalance

The bonafide data section represents approximately the 10% of each partition of our datasets, and this may cause the CNNs to function incorrectly, e.g., selecting the spoofed class not because of its prediction ability, but because there is a 90% chance that the selected sample is in fact spoofed. To balance the ratio between bonafide and spoofed data we employed the oversampling technique. This means showing to the neural net a subsection of the chosen dataset at each epoch. Half of the data in a subset belongs to one class (i.e., bonafide) and the other half belongs to

the second one (i.e., spoofed). In doing so, some samples belonging to the class containing less observations are repeated, as shown in Figure 4.8.

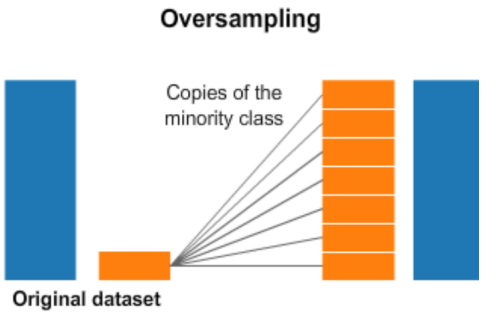


Figure 4.8: Visual representation of the oversampling technique, showing the repetition of parts of a subset of the dataset in order to pair it to the data belonging to the other class.

4.2.2 CNN Training Details

All our trainings have been performed on a server equipped with 12 Cores working at 2.6 GHz, a 126 GiB RAM and 2 24 GiB GPUs.

The platform we used for the experiments is TensorFlow, an end-to-end source platform intended for Machine Learning. In particular, we made great use of the Keras API in conjunction with TensorFlow.

As optimizer we used an Adam optimizer. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments [81]. We imposed the parameter of the learning rate of our training sequence equal to 0.001 for the VGGish classification and for the DnCNN denoising. We used a learning rate of 10^{-4} for the end-to-end pipeline.

While training the nets, we noticed the presence of overfitting between the validation and training loss. In attempt to mitigate this behavior, we used callback functions from TensorFlow. Hence, after verifying that the validation loss does not decrease during a period of a number of epochs, the callback function proceeds to lower the learning rate by a certain amount. We imposed as "patience" variable 3 epochs, and as dividing factor we chose to reduce by half the learning rate for each iteration of the callback.

We set a maximum amount of epochs different for each training pipeline. We use another callback function that stops the training once it witnesses no progress in the training loss after a chosen number of epochs. For the standalone VGGish classification we chose a maximum amount of epochs equal to 30 and we imposed a patience of 8 epochs for the early stopping callback. In the case of the DnCNN we chose 50 maximum epochs and 15 epochs for the callback function. For the end-to-end case we divided the training process in two stages: one where the DnCNN is used as non-trainable and the VGGish side is trainable, and the second and final one where both of the neural nets are trainable and train together. We set to 5 the maximum number of epochs for the first stage, with no early stopping. We chose 30 as the maximum amount of epochs and a patience of 15 epochs for the second stage.

For each training, except for the first stage of the combined classification, we use a callback function to select as best model the one providing minimum validation loss.

4.2.3 Pre-Processing and Denoising Parameters

In this section we delineate the parameters that were used for the pre-processing stages of the classifiers, both used alone and with the juxtaposed denoisers.

4.2.3.1 Pre-processing Parameters

In the pre-processing section in Chapter 3, we generally described how the original recording track is pre-processed to fulfill the requirements to be an input for the VGGish classifier. We hereby explain in detail each step, following the pipeline presented in the experimental framework of [30].

After loading the track as is, we check that the sample rate is equal to $f_s = 16$ kHz, otherwise we re-sample the recording. We then scale its dynamic, imposing it being between -1 and 1. Afterwards, we select the central window of each excerpt, choosing a length $l_{x,seconds} = 0.96$s, and discard recordings shorter than that. We choose to select the central window to be sure there is sound emitted, and not silence due to the

start o ending of the recording.

For the next step, the STFT is computed on the windowed signal using a Hanning, window having length $l_{w,seconds} = 0.025$s. Its length in samples is hence:

$$l_{w,samples} = f_s \cdot l_{w,seconds} = 16\text{kHz} \cdot 0.025\text{s} = 400. \tag{4.4}$$

The hop length corresponds to $l_{h,seconds} = 0.01$s, therefore, in samples:

$$l_{h,samples} = f_s \cdot l_{h,seconds} = 16\text{kHz} \cdot 0.01\text{s} = 160. \tag{4.5}$$

Finally, the length of the windowed signal after padding with zeros, n_{fft} is equal to $2^{\lceil \log_2(l_{w,samples}) \rceil}$.

The STFT is fed to a Mel filter-bank with sample rate equal to 16 kHz, the number of Fast Fourier Transform (FFT) components is equal the aforementioned n_{fft}, the number of Mel bins M is 64, the minimum Mel frequency is equal to 125 Hz and the maximum one is 7500 Hz.

The final number of frames of our log-mel spectrogram N is:

$$\text{N} = \frac{l_{x,seconds}}{l_{h,seconds}} = \frac{0.96\text{s}}{0.01\text{s}} = 96. \tag{4.6}$$

Pairing this with the number of Mel bins M equal to 64, we obtain the N×M shape of the log-mel spectrogram we use as input for our classification network: (96, 64).

4.2.3.2 Denoising Parameters

In this section we want to explain the parameters used for the signal processing denoisers we employ in our experiments. These denoisers are the Total Variation denoiser, the wavelet one, the bilateral and finally the Non-Local Means denoiser, and they are implemented following the documentation presented in [82].

In order to use the mentioned denoisers from [82], we firstly need to scale the pre-processed spectrograms between 0 and 1.

To broaden the spectrum of our experiments we input the denoisers both in linear and logarithmic scale.

The values chosen for the different functions were empirically selected after preliminary experimentation, trying to achieve an improvement in

the MSE between the denoised samples and the original ones, with respect to the MSE computed with original and noisy signals. We report them for the sake of making our simulation easier to reproduce, but we do not explain them in depth.

In Table 4.2 we display the parameters chosen for the Total Variation denoiser, where the *weight* represents the denoising weight. The greater this variable is, the more denoising is applied to the picture (at the expense of fidelity to input), [82].

	Linear	**Logarithmic**
weight	0.0005	0.0005
multichannel	True	True

Table 4.2: TV parameters for linear and logarithmic datasets.

As for the wavelet denoisers, presented in Table 4.3, we used the same values used in the examples shown in [83].

	Linear	**Logarithmic**
multichannel	False	False
restore_sigma	False	False
mode	Soft	Soft

Table 4.3: Wavelet parameters for linear and logarithmic datasets.

In Table 4.4 we report the bilateral denoisers' parameters. The variable *sigma_color* is the standard deviation for color distance, and its value is directly proportionate to the amount of difference the color of the pixels has to have, in order for the denoiser to average their value. Conversely, *sigma_spatial* is the standard deviation for range distance.

	Linear	**Logarithmic**
sigma_color	0.005	0.00005
sigma_spatial	5	5
multichannel	False	False

Table 4.4: Bilateral parameters for linear and logarithmic datasets.

A larger value results in averaging of pixels with larger spatial differences [82].

Finally, in Table 4.5 we show the values we chose for the Non-Local Means denoiser. Here, *sigma* represents the standard deviation of the noise, and h is the cut-off distance. A higher value of h results in a smoother image, at the expense of blurring features. Finally, *patch_size* is the size of the patch used for the Non-Local Means denoising, and *patch_distance* is the maximal distance in pixels where the patches used for denoising are to be searched for, [82].

	Linear	**Logarithmic**
sigma	0.01	0.1
h	$0.8 \cdot 0.01$	$0.8 \cdot 0.1$
fast_mode	False	False
patch_size	5	5
patch_distance	7	7

Table 4.5: Non-Local Means parameters for linear and logarithmic datasets.

4.3 Evaluation Metrics

In this section we explain the metrics we are going to use to evaluate the results. The outputs of our CNNs represent the predictions of our system for the inputted data. Let us denote, like we did in Chapter 3, $\hat{y}$ as the prediction of the system and y as the true value of the input's class. We evaluate the system's performances by comparing all the values of $\hat{y}$ to their correspondent y.

We mainly focus on the Balanced Accuracy matrices of the outputs of each pipeline. The Balanced Accuracy is a metric used to evaluate the performances of a binary classification, said to be useful when working with imbalanced datasets. It is computed by averaging the main diagonal of the Confusion Matrix.

The latter is a $N \times N$ matrix, where N is the number of classes predicted by the classifier. Each element $n_{i,j}$ of the matrix, displays the

number of samples with predicted class j and having true class i. We use 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 Results

In this chapter we explore the results obtained with our experimental campaign. This allows us to validate the proposed solution in a fair objective way.

Just as explained in the last section of Chapter 4, our main metrics to investigate the results are Balanced Accuracy values, computed starting from Confusion Matrices.

Moreover, to visually display the effects of the different denoising strategies we adopted, we show the spectrograms of one track chosen from the test subsection of the clean dataset.

We divide our results following the different pipelines prior explained in Chapter 3.

5.1 Baseline Classifier

In this section we address the results obtained by simply feeding the pre-processed samples to the VGGish classifier, with no denoising stage, as

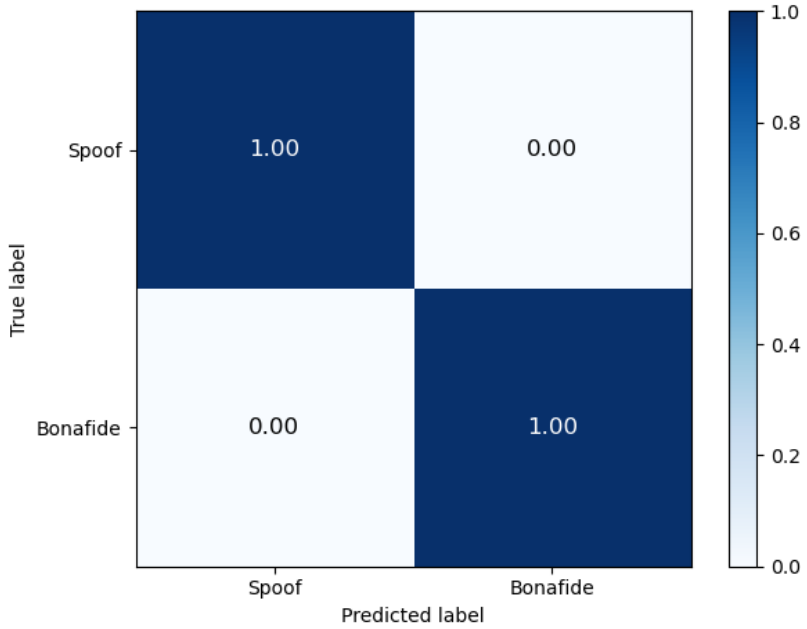
displayed in Figure 3.2a. The results shown in this section will act as baseline, to evaluate if our solutions provided any improvements.

5.1.1 Classification on Clean Dataset

In this section we report the results achieved in the noiseless scenario.

In Chapter 4, we divided the spoofing algorithms used to create the different sections of the clean dataset as *known* and *unknown*, following [5]. In order to check the performances in different scenarios, we ran experiments on all the partitions of the clean dataset (training set, validation set and evaluation set), keeping in mind that the training stage of the CNN is always performed on the training portion of the dataset, and the validation one on the validation dataset.

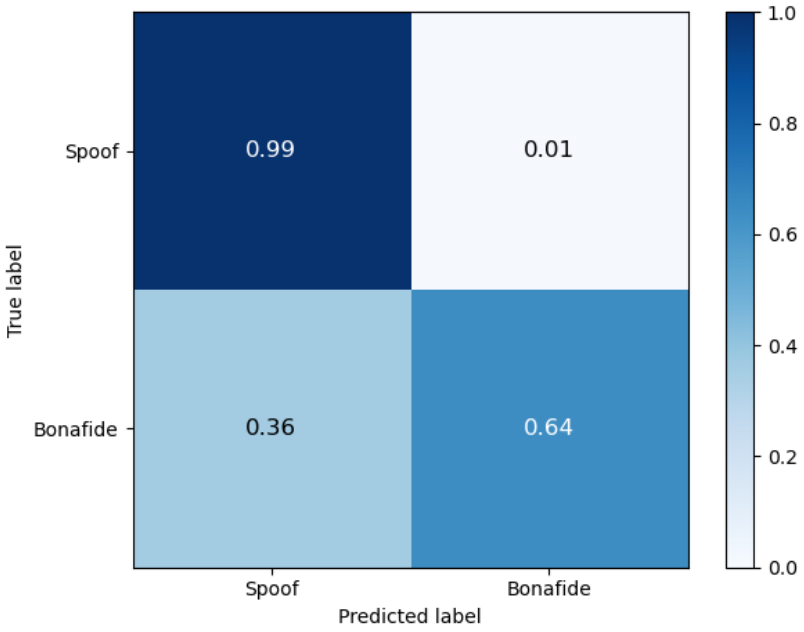
We display now the Confusion Matrices for training, validation and evaluation datasets in Figure 5.1. Taking into analysis the Confusion Matrix obtained using the training set as input of the CNN, Figure 5.1a, we can deduce that the Balanced Accuracy is equal to 1.00. So the 100% of the samples is correctly classified as spoofed or bonafide.



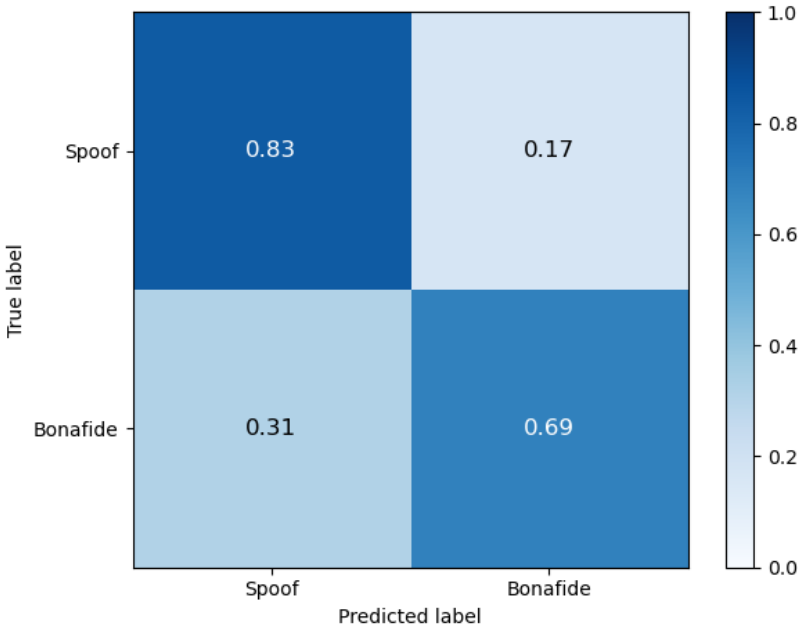
(a) Training set as input of the classification stage.

On the other hand, examining Figure 5.1b, we can observe that the Balanced Accuracy drops to 0.82, since the correct classification of the

bonafide samples decreases, reaching 0.64. This drop is explained by the fact that using the validation set as input of the classifier means exposing it to a new set of data, although still using *known* spoofing algorithms.



(b) Validation set as input of the classification stage.



(c) Evaluation set as input of the classification stage.

Figure 5.1: Confusion Matrices resulting from the classification of all the clean dataset partitions.

The last Confusion Matrix shown in Figure 5.1c reveals an additional worsening in the performances. The Balanced Accuracy is 0.76. This is the worst case scenario for using clean datasets both in training and testing stages. It was fairly predictable, given the new set of data inputting the CNN with *unknown* spoofing algorithms.

5.1.2 Classification on Noisy Datasets

In this section we explore how the baseline solution behaves when data augmented with noise serve as its inputs.

In Figure 5.2 we display the spectrograms obtained from the clean and noisy tracks. It reveals a clear difference between the clean and noisy samples. This Figure will serve as baseline for the analysis of the denoised samples in the next sections.

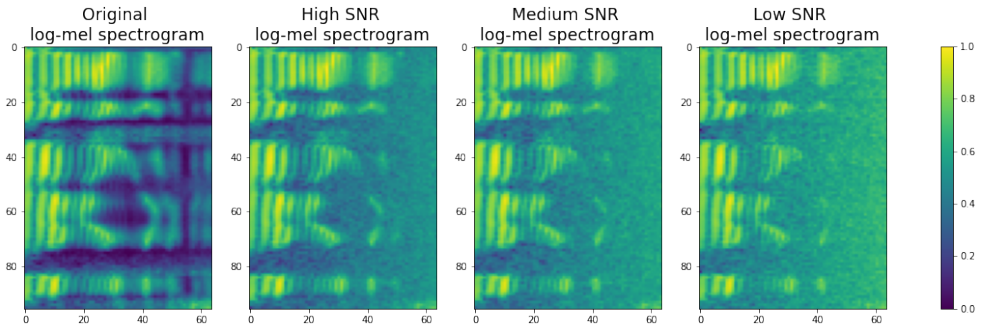
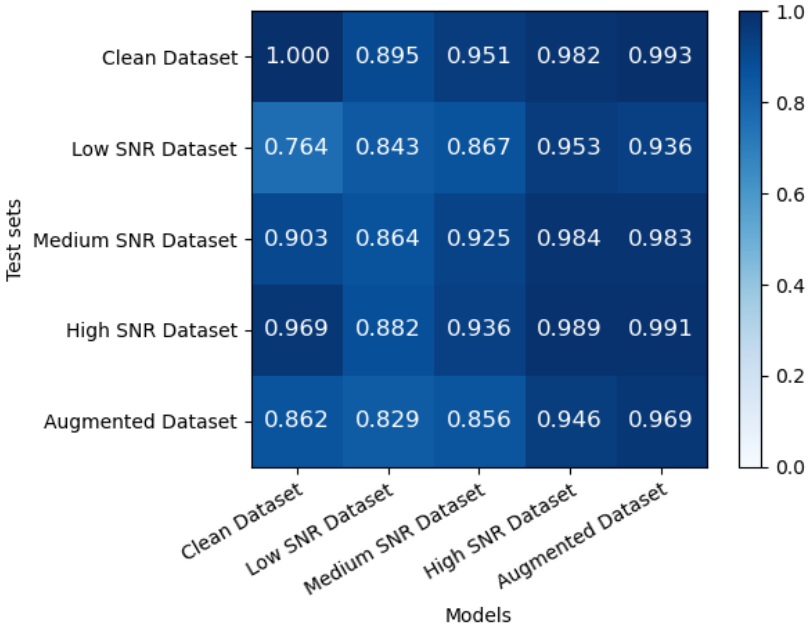


Figure 5.2: Spectrograms of a sample from the evaluation subset, displayed in its clean form and the noisy ones.

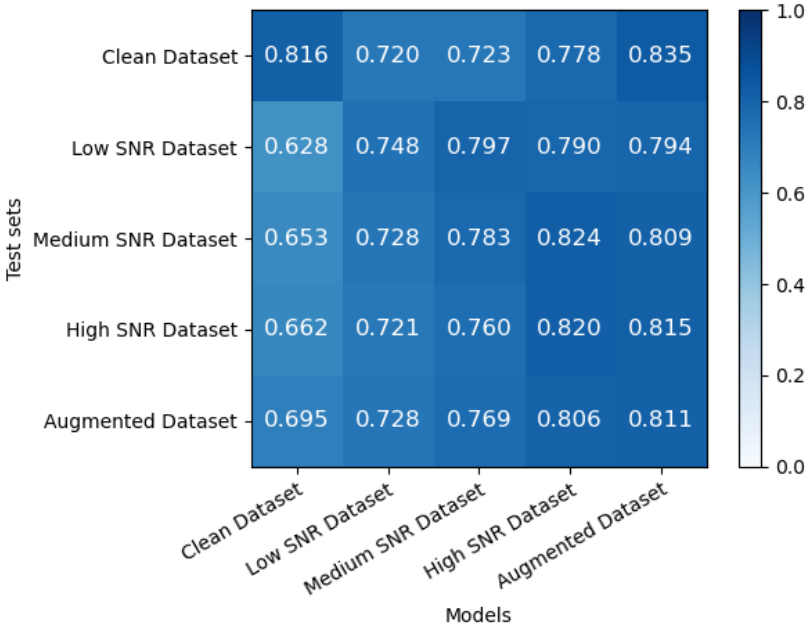
Analogously to the clean experiment, we display the results related to the classification stage using each noisy dataset as input. In order to supply a complete overview on the capabilities of the system, we report the results pertaining the application of all the sets of data to all the possible models. Consequently, we will see, both as training model and test set: the clean dataset, the high SNR dataset, the medium SNR set, the low SNR one and the augmented dataset, containing a mixture of all of the above ones.

From hereafter, instead of displaying each Confusion Matrix, we focus on the Balanced Accuracy values to compare the performances of our results. The Balanced Accuracy matrices have on the horizontal axis the

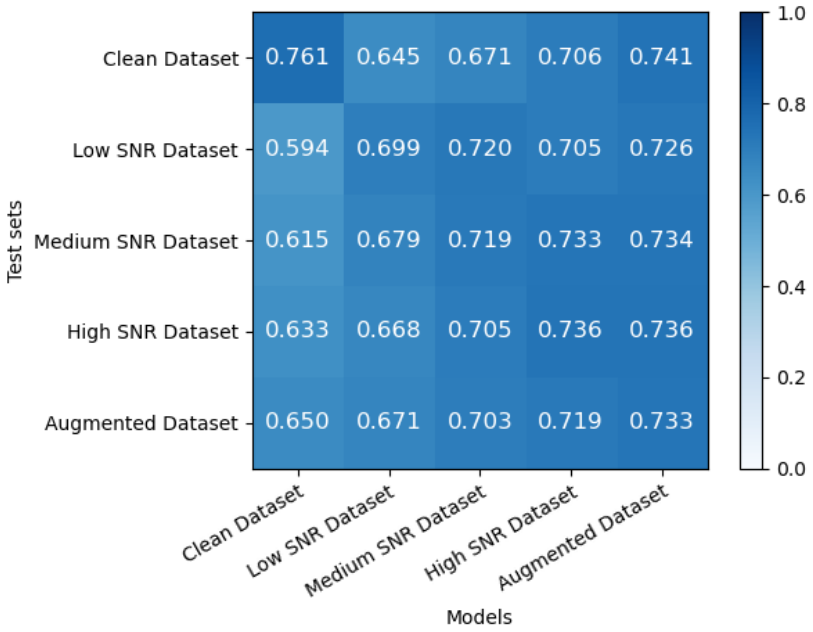
sub-sets used to train the CNN and on the vertical one the subsets on which the CNN has been tested on. With this matrix we want to highlight how the noisy environments influence the system's performances, reminding the reader that the baseline clean output corresponds to the cell related to clean dataset training model and clean dataset test set.



(a) Training clean and noisy sets used as input of the CNN.



(b) Validation clean and noisy sets used as input of the CNN.



(c) Evaluation clean and noisy sets used as input of the CNN.

Figure 5.3: Balanced Accuracy Matrices of the sole classification problem, using both clean and noisy datasets.

Once again, we present the results separately for training, validation and evaluation data, in the three graphs in Figure 5.3. It is important to note that the difference between them stands in the vertical axis, the one representing which set of data the CNN is tested on.

We can observe a ranking of performances similar to the ones of the previous experiment. The CNN fed with the training sets performs better than the other two scenarios, when fed with the evaluation sets it performs the worst.

We can also infer that, generally, as the datasets' SNR decreases the performances decrease too, in both rows and columns. Therefore, training the VGGish with sets of data with lower SNR concurs to lower Balanced Accuracies. A clear example of this can be seen in the columns corresponding to the Low SNR Dataset models in Figure 5.3a, 5.3b and 5.3c.

At the same time, when we feed the Low SNR Dataset to the CNNs created training the VGGish with any of the other datasets, we witness performances that are generally worse with respect to the other rows of

the matrices.

Finally, we can state that all the Balanced Accuracies where noise is involved are always lower than the ones observed in the prior section of this chapter (corresponding to the cell with clean dataset model and clean dataset test set).

We think it is interesting to make some additional considerations on the results obtained training the CNN with the augmented dataset. We can observe these results on the last column and last row of Figure 5.3. We can infer that this is a first step towards making the network more robust to additive noise, by noting that the Balanced Accuracy values related to augmented models are mainly closer to the baseline case with respect to the other results and show a more stable behavior as well. This can be observed especially in the examination of the evaluation set in Figure 5.3c.

From now on we show solely the results obtained by testing on the evaluation partition of the datasets (and not for training and development sub-sets), since this is the most general case, that could be used in further analysis of this task.

5.2 Classifier with Denoising Stage

In this section we report the results obtained considering also the denoising step of our proposed pipeline. We first report results relative to standard denoising algorithms. We then illustrate our findings in the situation in which the denoiser is linked to the classifier, thus being tailored to the problem at hand.

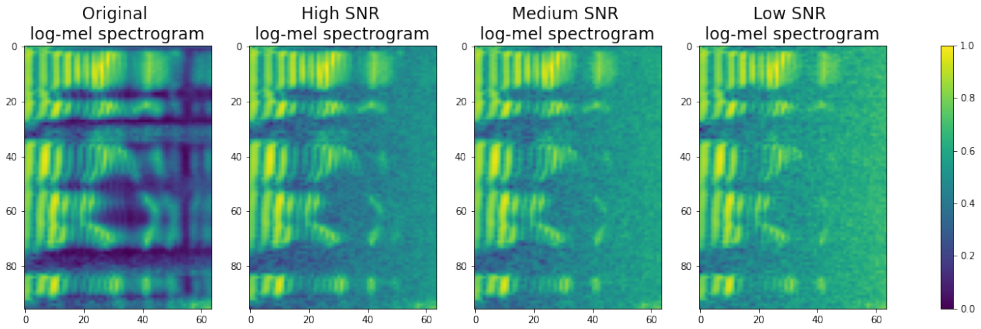
As described in Chapter 4, for this proposed solution we pre-process the input spectrogram through different denoisers before feeding it to the CNNs.

We proceed now in illustrating and commenting the results obtained employing as denoising stage each algorithm implemented with the parameters reported in Section 4.2.3.2, and the one presented in 3.2.1.6.

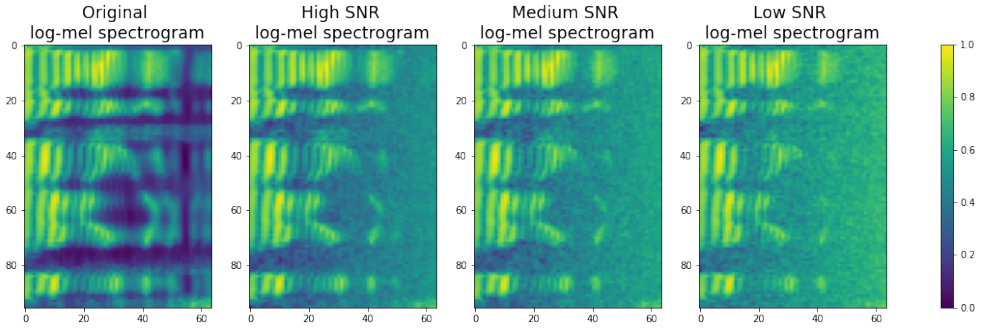
5.2.1 Total Variation

In this section we describe the performances observed with Total Variation denoising on our datasets.

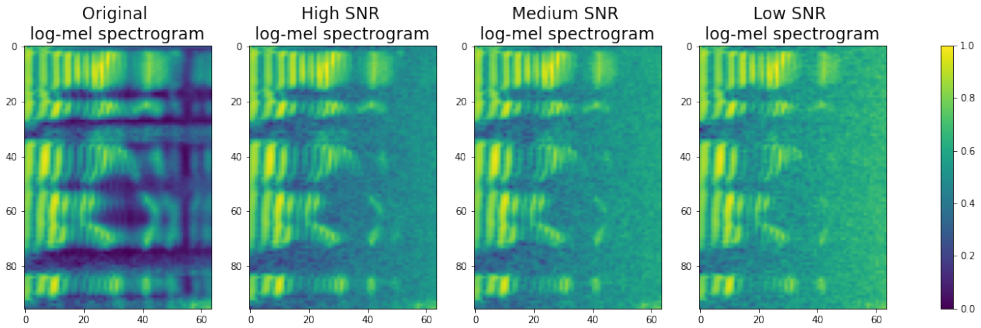
In Figure 5.4 we are not able to point at any defining differences between 5.4b and 5.4c, nor in comparison with Figure 5.4a.



(a) Baseline spectrograms from Figure 5.2.



(b) TV denoising with linear input.



(c) TV denoising with logarithmic input.

Figure 5.4: Spectrograms before and after TV denoising.

After creating the new datasets as previously described, we use these as input of our differently trained CNNs in order to compute the metrics needed to check on the system's performances. As for the Balanced Accuracy values shown in Figure 5.5, we can actually denote some slight

increase in performance, in particular in the row corresponding to the clean linear dataset test sets and in the column related to the clean dataset model. The rest of the results end up being really close or worse to the ones in Figure 5.3c.

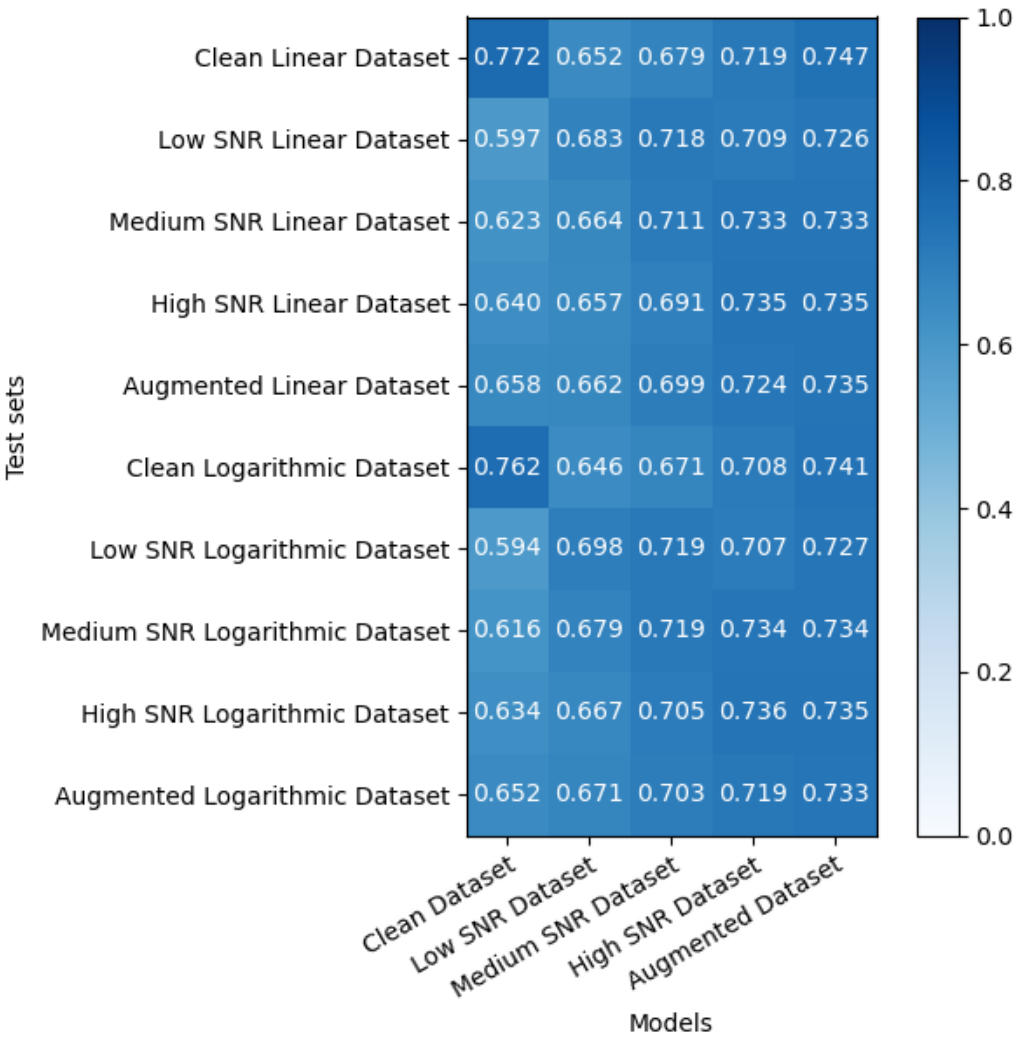


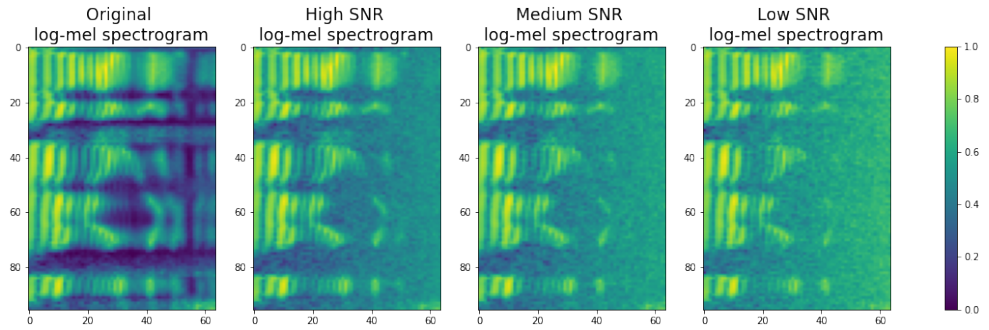
Figure 5.5: Balanced Accuracy matrix computed with linear and logarithmic datasets created with TV denoising.

5.2.2 Wavelet

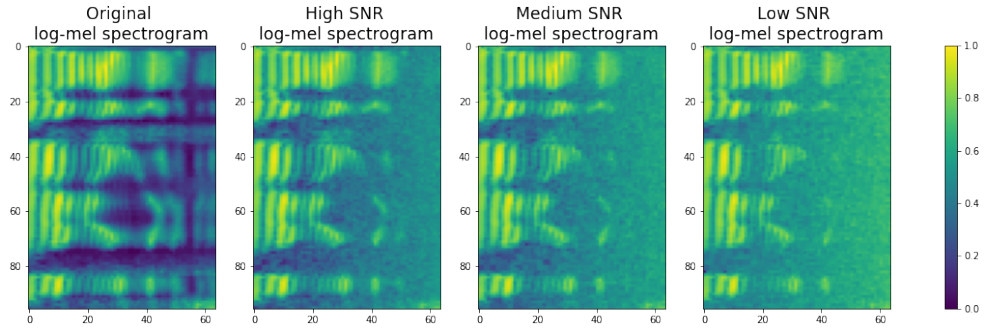
This section is about the evaluation of the performances of the CNNs with wavelet transform and thresholding denoising.

As in the just described section, observing the spectrograms in Figure 5.4, 5.6b and 5.6c can be considered almost identical. Furthermore,

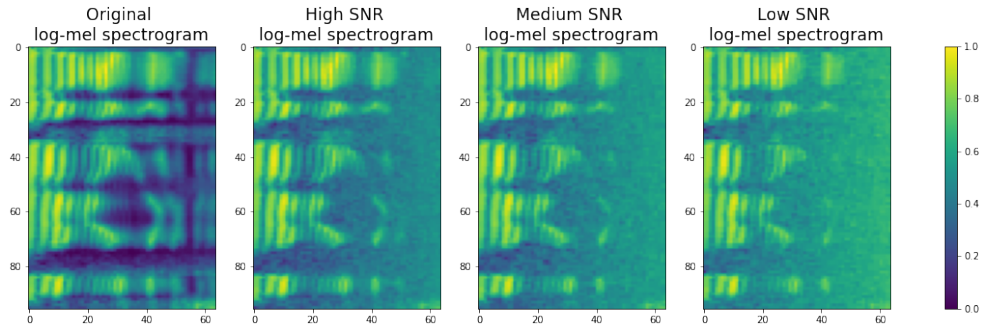
there is no evidence of improvement with respect to Figure 5.6a, as well.



(a) Baseline spectrograms from Figure 5.2.



(b) Wavelet denoising with linear input.



(c) Wavelet denoising with logarithmic input.

Figure 5.6: Spectrograms before and after wavelet denoising.

Let's focus now on the Balanced Accuracies values obtained using the Wavelet algorithm for the denoising stage. In Figure 5.7 we can observe that, as for the linear side of the results, there is a small increasing in values, but the numbers are overall fairly equal to the ones in Figure 5.3c. Conversely, the logarithmic inputs cause a drop in the performances, with respect to 5.3c as well.

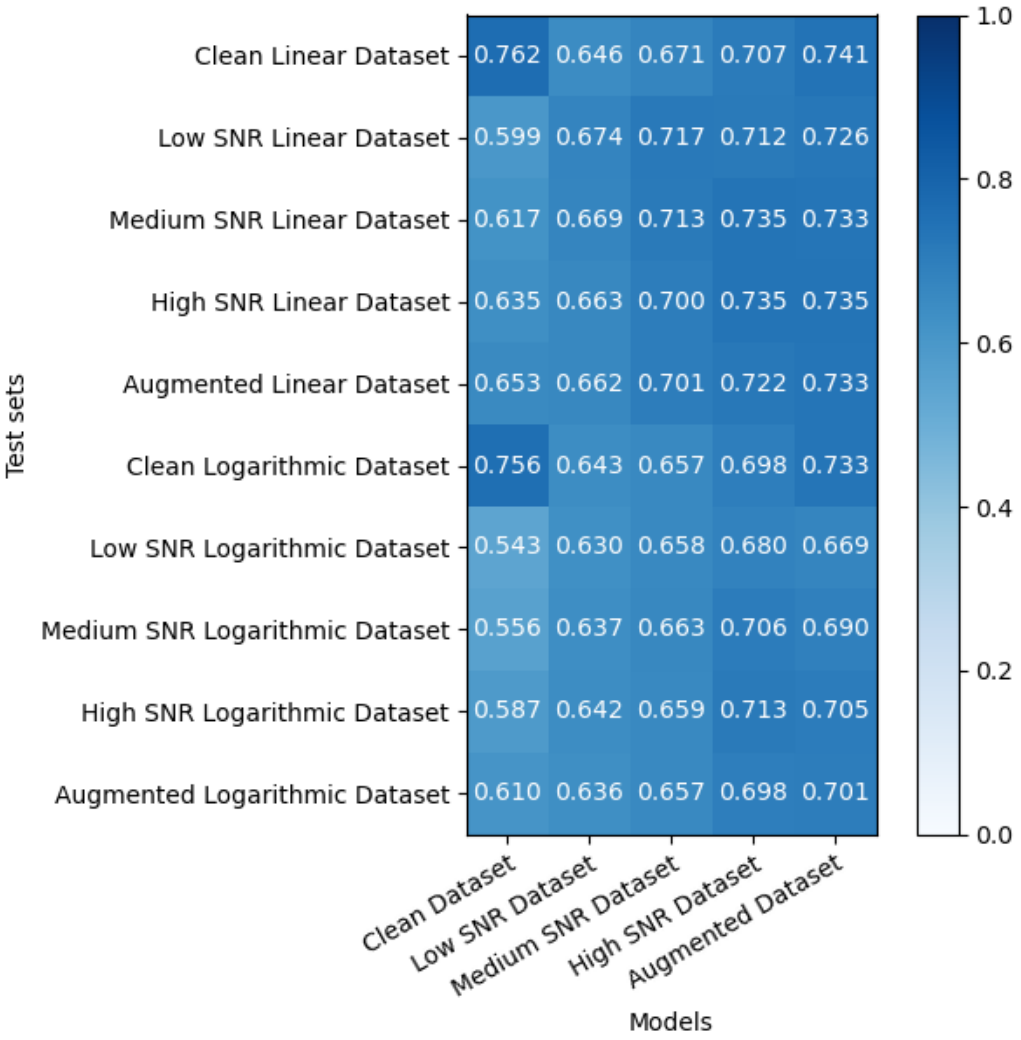
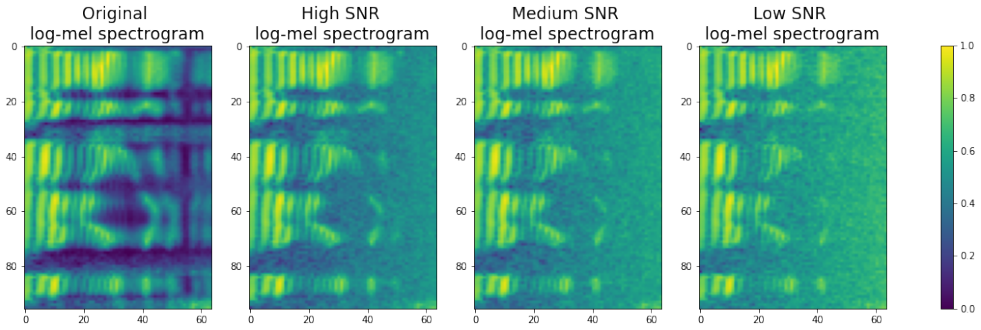


Figure 5.7: Balanced Accuracy matrix computed with linear and logarithmic datasets created with wavelet denoising.

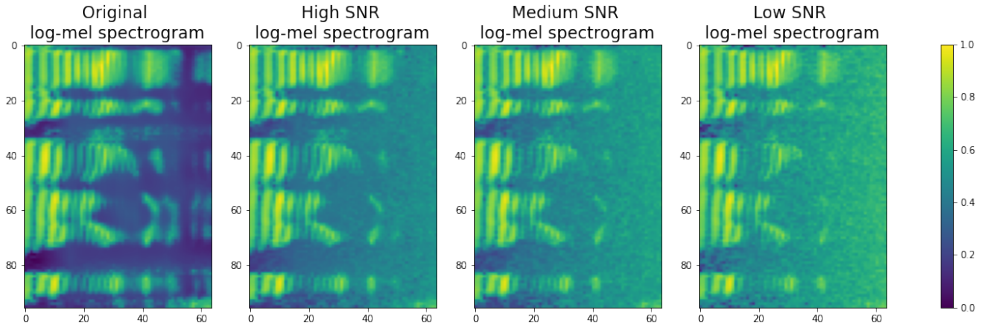
5.2.3 Bilateral

In this section we take in exam the results of the denoiser and CNNs pipeline, when the selected denoiser is the bilateral one.

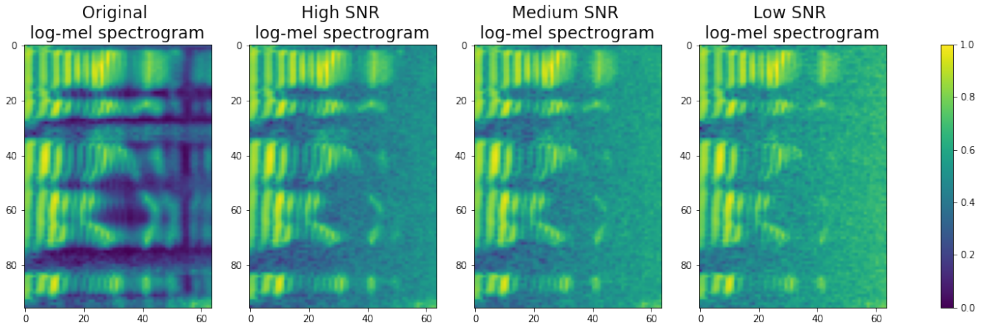
The spectrograms in Figure 5.8b and 5.8c show evident differences between one another. In particular, the linear input 5.8b displays smoother spectrograms with respect to 5.8c, and most of all with respect to 5.8a.



(a) Baseline spectrograms from Figure 5.2.



(b) Bilateral denoising with linear input.



(c) Bilateral denoising with logarithmic input.

Figure 5.8: Spectrograms before and after bilateral denoising.

In Figure 5.9 we show the matrix with the Balanced Accuracy values obtained using the chosen bilateral denoiser as pre-processing denoising step, applied to the evaluation dataset. While the logarithmic inputs manage to output data with performances comparable with the ones belonging to the baseline matrix, Figure 5.3c, the linear ones have the consequence of decreasing the quality of the classification. The smoothing observed in Figure 5.8b may be the cause of this deteriorating of performances.

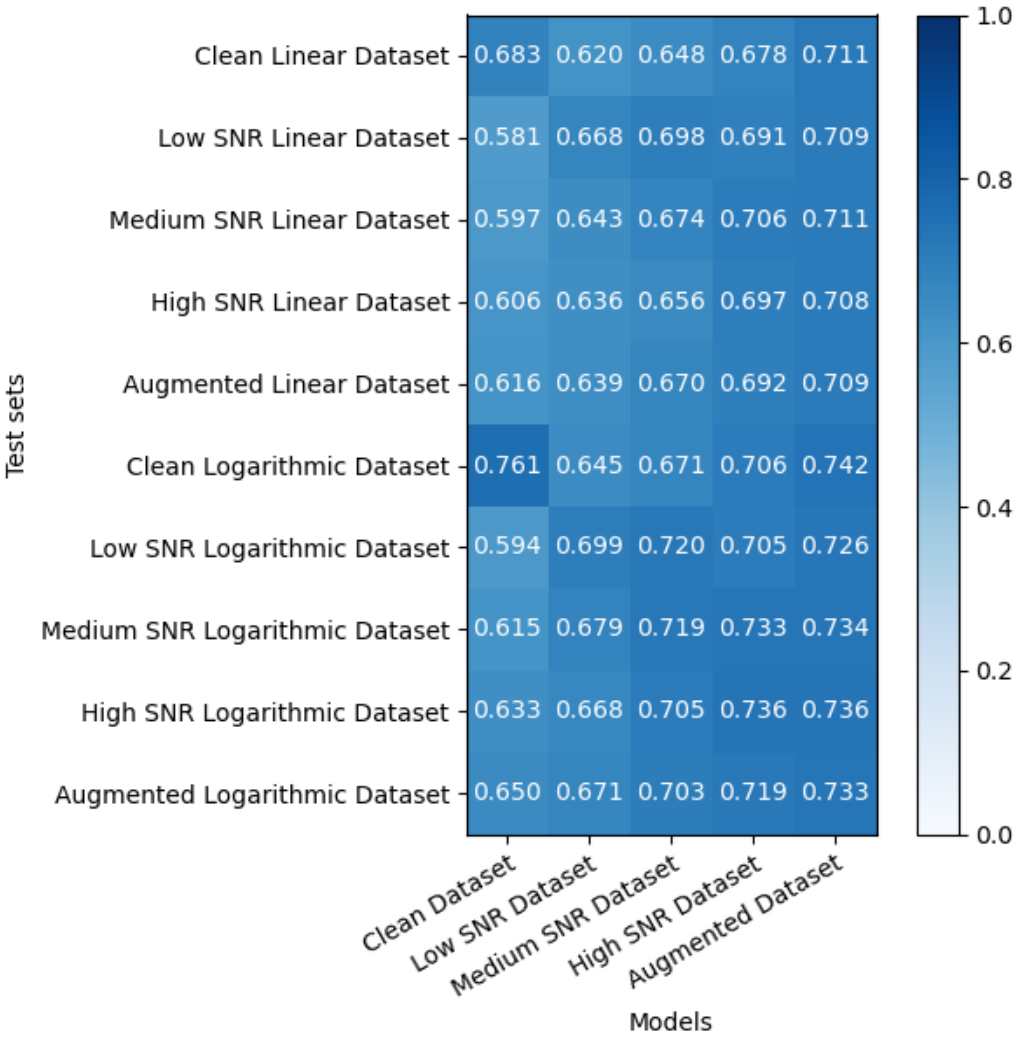
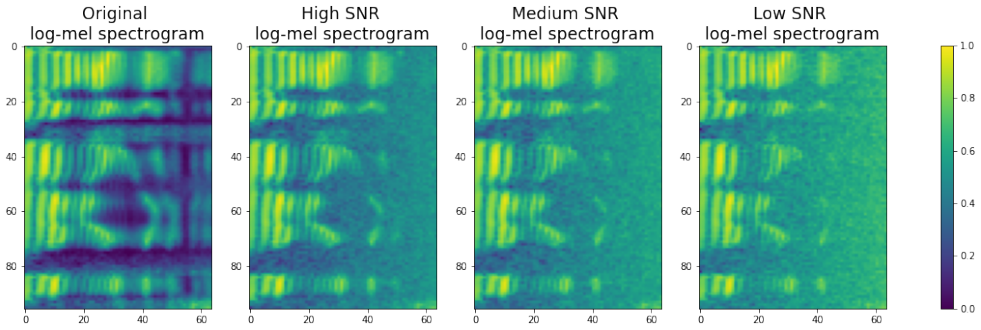


Figure 5.9: Balanced Accuracy matrix computed with linear and logarithmic datasets created with bilateral denoising.

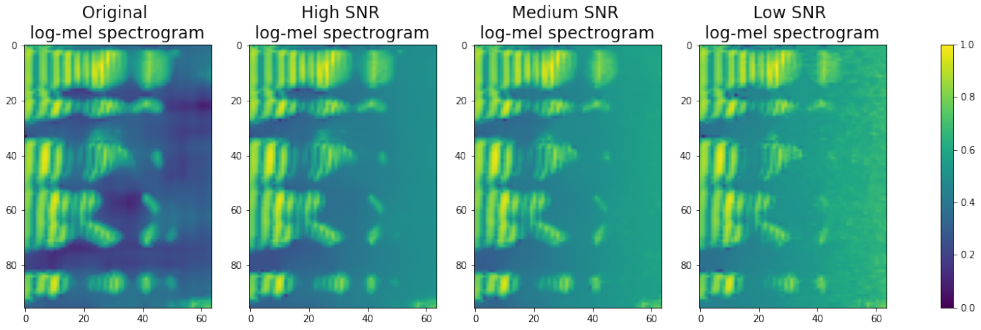
5.2.4 Non-Local Means

We now proceed to show the results achieved with Non-Local Means denoising pre-processing of input spectrograms fed to the CNNs.

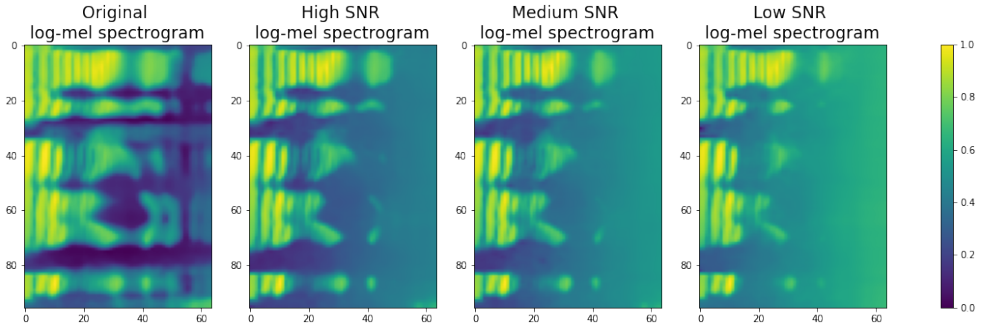
Both Figure 5.10b and 5.10c show spectrograms that, compared to 5.10a, exhibit a highly smoother behavior. We can also identify differences between Figure 5.10b and 5.10c: the denoised version of the linear original spectrogram, 5.10b, has higher values than the logarithmic one, 5.10c, therefore lighter tones, and the noisy representations of the latter display smoother spectrograms with respect to the above ones.



(a) Baseline spectrograms from Figure 5.2.



(b) Non-Local Means denoising with linear input.



(c) Non-Local Means denoising with logarithmic input.

Figure 5.10: Spectrograms before and after Non-Local Means denoising.

The Balanced Accuracies reported in Figure 5.11 show that the just described smoothing in the spectrograms does not coincide with an increase in performances for the CNNs. Conversely, neither the linear nor the logarithmic test sets provide any rise in the Balanced Accuracy values. This could be appointed, same as in the previous section, to the over-smoothing of the spectrograms. We can deduce that, together with the smoothing of the noisy images, the denoiser has the consequence of altering features important for the rightful classification of the samples.

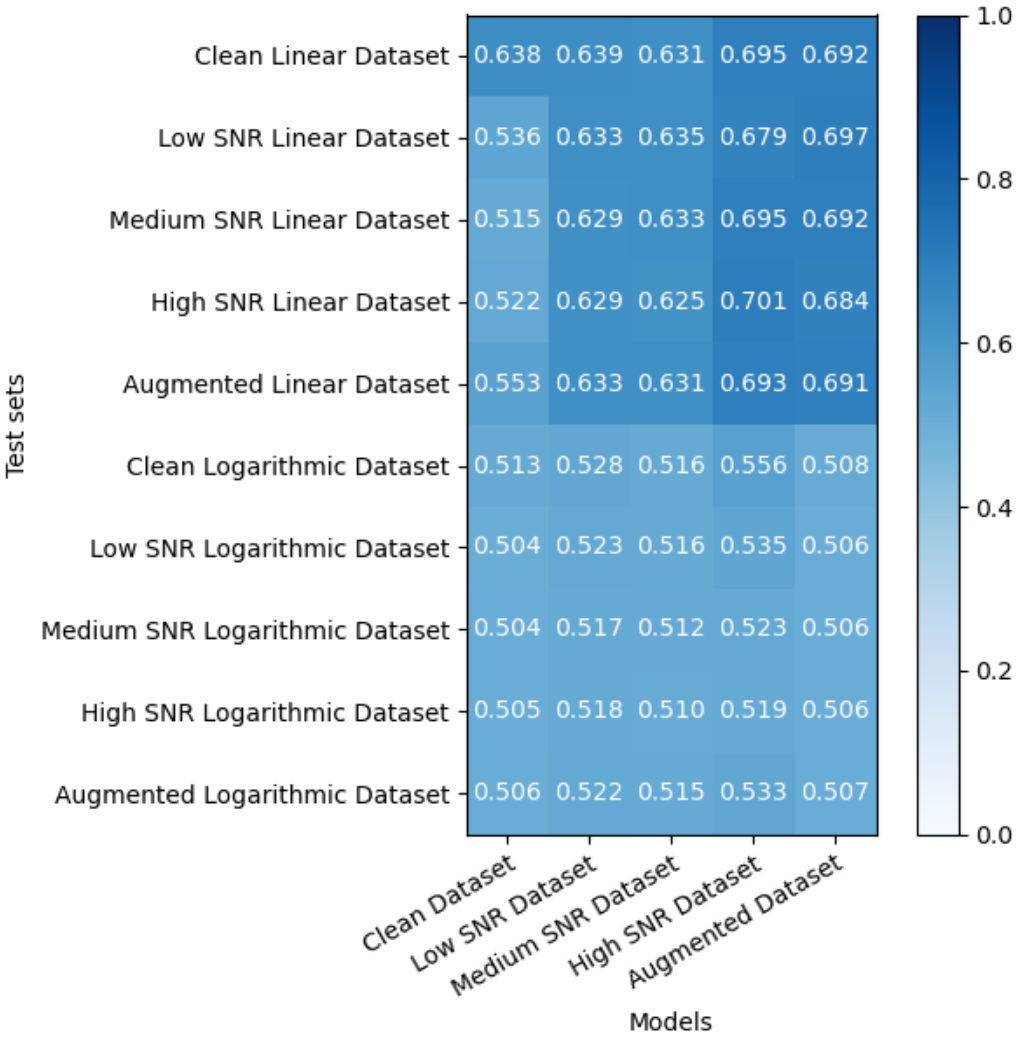


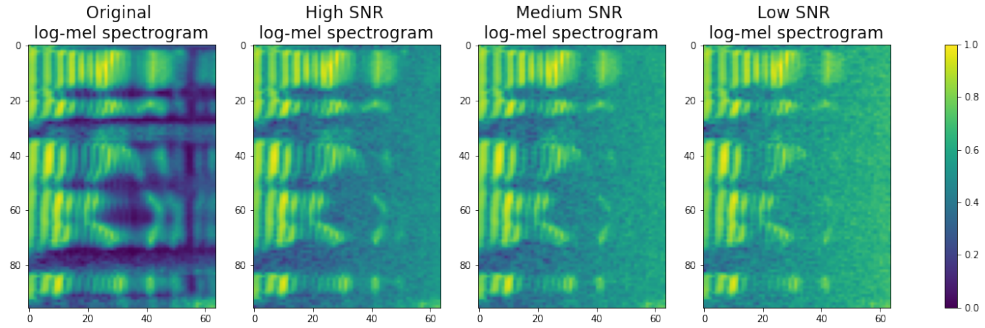
Figure 5.11: Balanced Accuracy matrix computed with linear and logarithmic datasets created with Non-Local Means denoising.

5.2.5 DnCNN

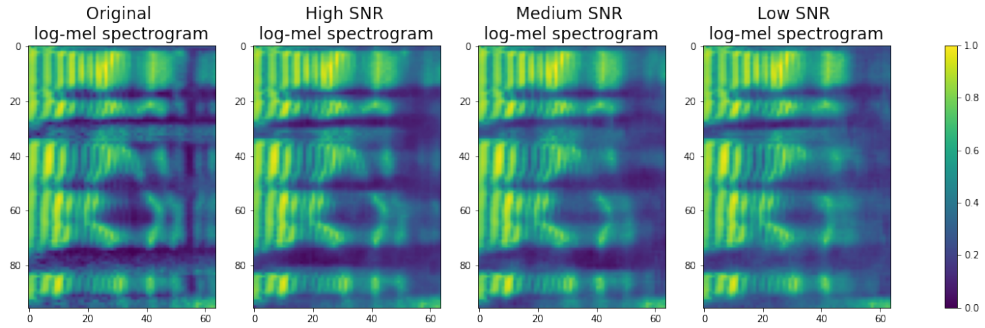
In this section we illustrate the results pertaining the pipeline composed by the DnCNN denoiser followed by the classifier.

In Figure 5.12b we can clearly observe an improvement in the spectrograms with respect to Figure 5.12a. The first representation to the left in 5.12b does not accentuate any change comparing it to the corresponding one in 5.12a, whilst the spectrograms related to the noisy environments show great improvements with respect to the not processed ones. Furthermore, the noisy spectrograms in 5.12b connote a smoothed behavior

compared to the original one in the same figure.



(a) Baseline spectrograms from Figure 5.2.



(b) Spectrograms after DnCNN denoising.

Figure 5.12: Comparison between baseline spectrograms and DnCNN denoised ones.

Despite showing great improvement in the spectrogram representation, the Balanced Accuracy matrix in Figure 5.13 shows that the overall performances of the pipeline have not increased with respect to the ones in Figure 5.3c. An exception can be made for the column using the model created with the clean dataset, where we can notice better values in comparison to the ones in 5.3c. Besides this exception, performances are on average at best equivalent to the ones in 5.3c.

5.3 End-To-End Results

Finally, we hereby analyze the results provided by the data driven denoising joined to the classification.

As we explained in depth in Chapter 4, the end-to-end pipeline was realized by concatenating pre-trained models of both the DnCNN and VGGish. After some preliminary experimentation we conveyed that the

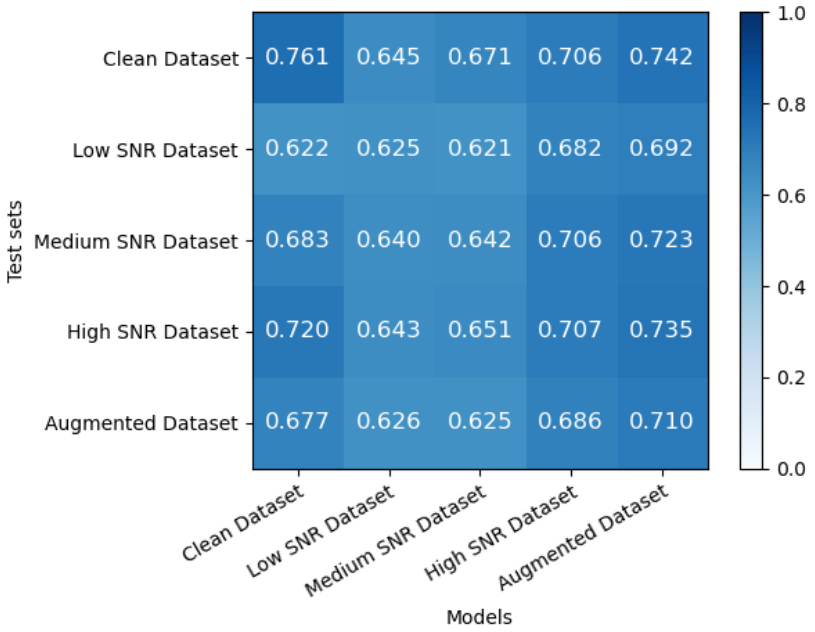


Figure 5.13: Balanced Accuracy matrix of the CNNs having as input the datasets created denoising the test sets with the DnCNN.

most effective combination of DnCNN and VGGish models was obtained using the models pre-trained on the high SNR dataset for both nets.

Reminding the architecture of the end-to-end pipeline, displayed in Figure 3.15, in Figure 5.14 we show one of the two outputs of the net, *dncnn_output*, the one corresponding to the cleaned spectrograms. Here

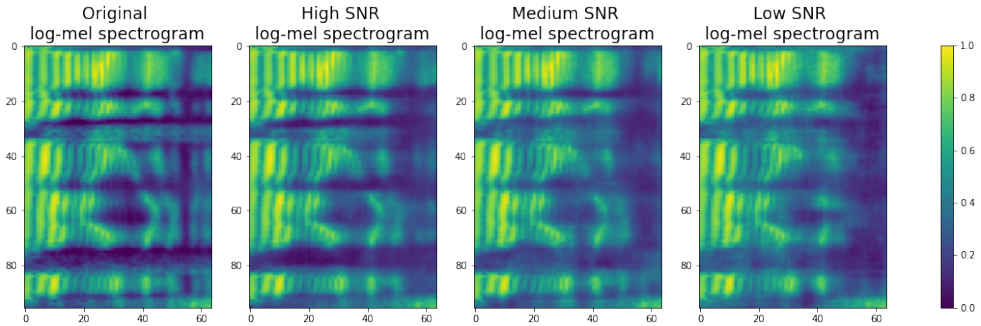


Figure 5.14: Spectrograms of the track in 5.2, one of the two outputs of the end-to-end pipeline.

we can observe big improvements with respect to the baseline spectrograms and a much more similarity with the original clean representation shown in the same picture. We can furthermore compare Figure 5.14 to

Figure 5.12b, and we can conclude that the two are fairly similar, except for the latter mentioned one having lower values, therefore darker shades, in the noisy spectrogram representations.

Comparing Figure 5.15 with 5.3c, we can also note that on average the first reaches higher values of Balanced Accuracy, despite presenting a lower highest value. Looking at Figure 5.15 in its entirety, we can conclude that we were able to render our net robust to noise. As matter of fact, using end-to-end training, metrics obtained on evaluation dataset are generally good, regardless of the presence of noise in the tracks.

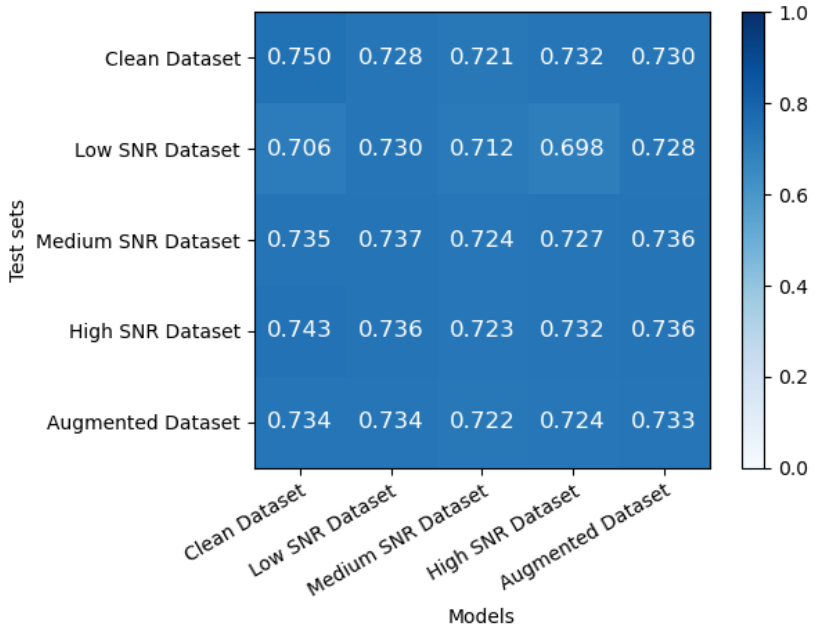


Figure 5.15: Balanced Accuracy matrix of the end-to-end pipeline.

5.4 Conclusive Remarks

In this chapter we provided a rundown of the results we deemed more meaningful for the purposes of this thesis. We now report a brief comparison of the achieved results to clarify pros and cons of the discussed methods.

Our goal was to increase the performances of the CNNs, trained and tested on both clean and noisy datasets. We purposely chose to use four "classic" signal processing denoisers in order to verify if their perfor-

mances could be comparable to data driven ones used in juxtaposition to a data driven classification. Our results show that we achieve some little or no improvement using signal processing based denoisers. Using a data driven based denoiser prior to the CNN also has the consequence of performances pretty similar to the baseline ones. The best results are achieved with the end-to-end pipeline.

The results also suggest that obtaining denoised spectrograms very similar to clean ones does note ensure actual improvements in the Balanced Accuracy values.

Amongst the experimental computation, not mentioned for the sake of readability of the results, we want to add the following simulations and considerations. We divided the datasets by their individual spoofing algorithms (presented in Section 4.1.1) and we used these subsections, paired with the bonafide section of each partition, as input of all the CNNs. Since each spoofing algorithm is different from the others belonging to the same dataset partition, it makes sense to deduce that the CNNs will perform differently depending on the spoofing class they are fed with.

We thus now present the worst and best results obtained using the spoof algorithms partitions as inputs of the CNNs, trained joined with the DnCNN. We display once again the Balanced Accuracy matrix presented in Figure 5.3 in Figure 5.16, for a visual comparison with the other matrices presented in this final analysis.

In Figure 5.17 we can clearly see that both 5.17a and 5.17b show worse performances with respect to the baseline one, 5.16. Conversely, regarding the comparison between 5.17a and 5.17b we can denote better performances on average in the latter, confirming the improvements introduced with the end-to-end pipeline. Furthermore, it shows much more averaged values, just as in Figure 5.15.

On the other hand, Figure 5.18 reports better results for every cell with respect to 5.16. As for the comparison between 5.18a and 5.18b, we can infer increased values in the Balanced Accuracies, especially in the column corresponding to the results obtained using the model trained on the low SNR dataset.

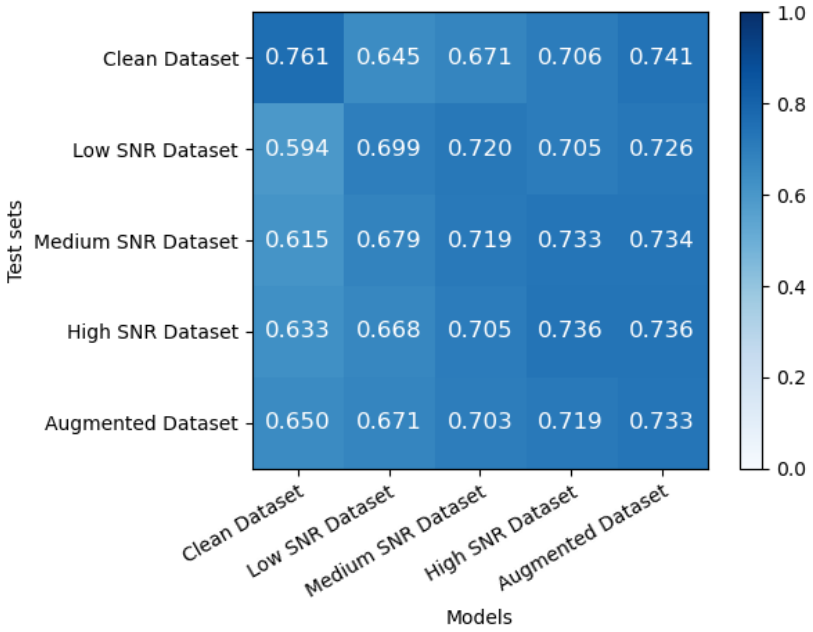
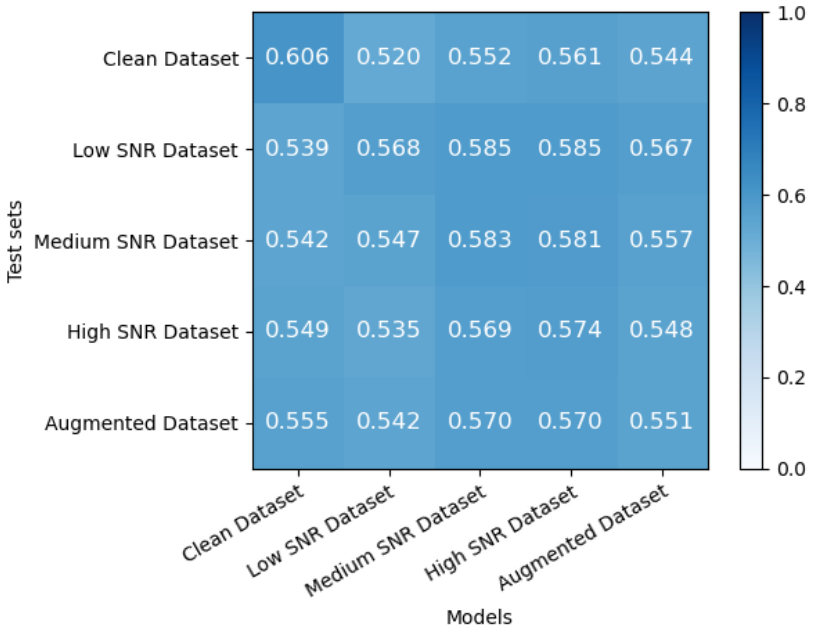


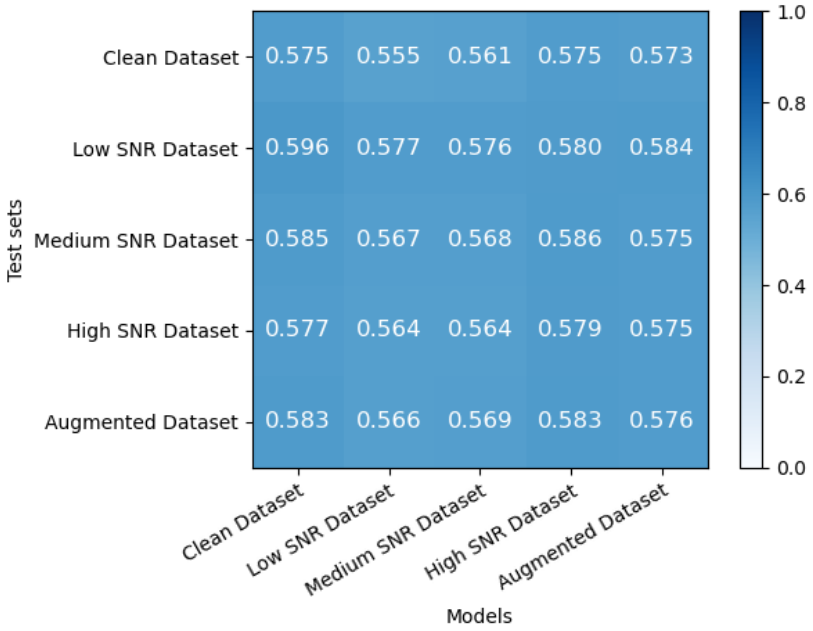
Figure 5.16: Baseline Balanced Accuracy matrix obtained with the usual inputs.

This result also shows a much more averaged behavior, once again corroborating the forecast of a better performance using the end-to-end pipeline.

These last considerations bring to light the difference in functioning caused by different spoofing algorithms. In particular, the ones we chose to display, as stated in Chapter 4, are both implemented by employing NN-based systems. Focusing on the implementation of each algorithm may concur to future developments in this kind of studies.

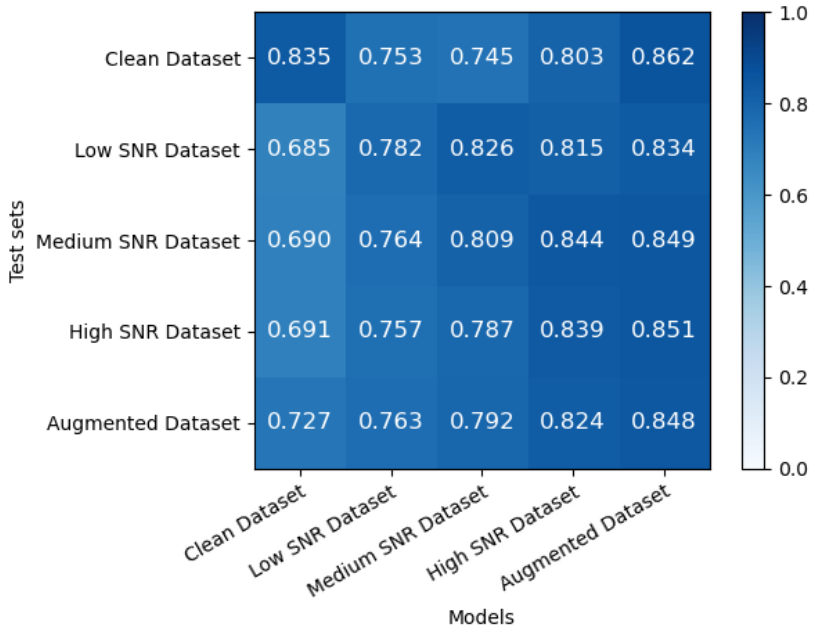


(a) Balanced Accuracy matrix obtained feeding the inputs to the baseline CNN.

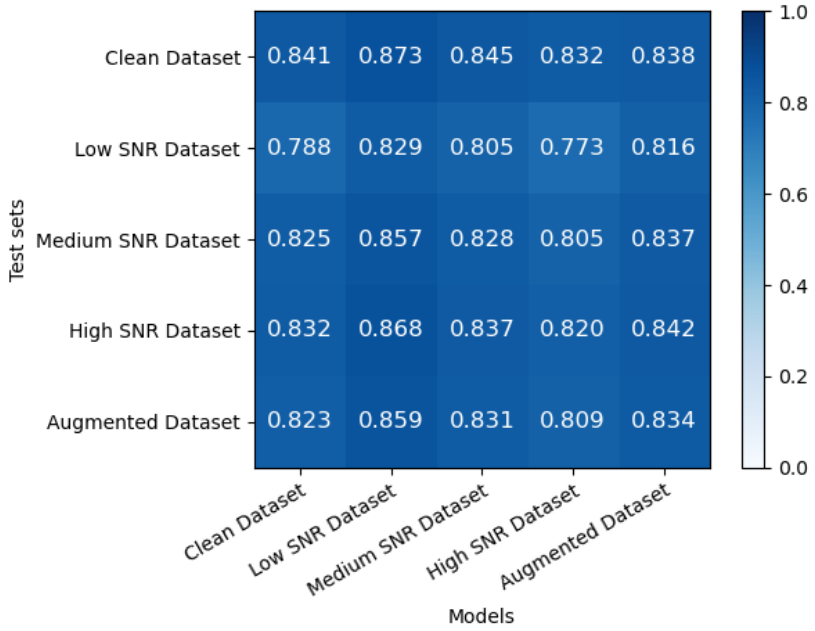


(b) Balanced Accuracy matrix obtained feeding the inputs to the end-to-end system.

Figure 5.17: Balanced Accuracy matrices obtained with different pipelines having the same untouched pristine and noisy inputs. The inputs are generated with subsets of the evaluation partition comprising of bonafide and A10 class spoofed data.



(a) Balanced Accuracy matrix obtained feeding the inputs to the baseline CNN.



(b) Balanced Accuracy matrix obtained feeding the inputs to the end-to-end system.

Figure 5.18: Balanced Accuracy matrices obtained with different pipelines having the same untouched pristine and noisy inputs. The inputs are generated with subsets of the evaluation partition comprising of bonafide and A17 class spoofed data.

6

Conclusions and Future Works

The increasing curiosity surrounding deepfake media results in always changing and refining algorithms that are usually aimed to be used as entertainment. What the common public may not perceive is that new techniques to mimic people's features (being speech recordings, images or recorded videos) also lead to more impersonation crimes. Therefore the job of forensic analysts is enriched with day to day updates on synthetic models they will have to figure out a detecting technique for.

In this thesis we proposed many countermeasures to execute synthetic speech detection in presence of noisy inputs. We tackled the problem of deepfake speech detection with the use of a Convolutional Neural Network called VGGish, previously trained on the YouTube 100-M database. In the hope of improving the performance of the classification process on noisy signals, we employed multiple signal processing denoisers and a data driven one juxtaposed to the CNN model. Furthermore, we set out an end-to-end system composed of a data driven denoiser, a DnCNN residual denoiser, and the VGGish being trained together.

The dataset on which we based our clean set of data originates from the ASVSpoof 2019 Challenges event, where the contestants had to confront themselves with new Text-To-Speech (TTS) and Voice Conversion (VC) algorithms for the creation of improved synthetic speech excerpts. We created our noisy environments by adding Gaussian noise to the pristine signals, lowering the original tracks' Signal to Noise Ratio (SNR) in different amounts. Moreover, we created an augmented dataset, where we collected data from each type of noisy and clean dataset in equal quantity.

We then tested the VGGish network on all the datasets, verifying the great decrease in performances we assumed it was going to happen feeding corrupted signals to the sole classifier. We proceeded by adding a denoising step to our pipeline: we therefore firstly separately fed the inputs to Total Variation, wavelet, bilateral and non-local means denoisers and then to the VGGish. Furthermore, we replicated the last pipeline, using a data driven approach for the denoising step. We selected a DnCNN, a net used for residual prediction for images. Finally, we tested an end-to-end system composed of pretrained versions of the DnCNN and the VGGish.

The results reported in Chapter 5 confirm our initial thesis on the performances of the different pipelines. The best performances are actually achieved by the end-to-end system we proposed, followed by the ones obtained with the configuration of DnCNN used separately from VGGish, and then there is a last group where all the other denoising techniques fall into.

We reckon it is important to point out that the best performances do not coincide with the best performance achieved for each pair of modeling and test set, but identify as such based on averaged scale. Furthermore, the most important goal achieved with the end-to-end pipeline is the leveling of Balanced Accuracy values, meaning that this solution can be an aid in the making of systems insensitive to noise.

Despite the good performance achieved with the proposed method, we hereby lay down some future research lines that we think may be a continuation for the systems we developed for this thesis.

In Chapter 4 we explained the difference between logical and physical access attacks. For our study we chose to use the LA scenario data alone, therefore a first proposal for future developments is to measure the system performances using the physical access spoofing attack as well.

Moreover, it could be interesting to observe the functioning of the system when different noise models are considered. An idea could be to use different kinds of degradation to corrupt the recordings, e.g. using Room Impulse Responses to introduce reverberation in the system, or making use of different compressing techniques.

Another suggestion could be adding a regularizing *Dropout* layer at the end of each convolutional block of the VGGish (VGGish architecture in Figure 3.14), in the attempt of further lowering the overfitting phenomenon affecting the training of the network.

As for the training of the VGGish classifier, instead of being applied on the whole training dataset, the system could be trained on portions of the dataset comprising the bonafide data and each spoofing algorithm alone, to see how performances are affected.

Finally, the binary classification implemented by our system could be replaced by a multiclass classification, identifying one class for the bonafide data and for each of the spoofing algorithms taken into account.

Bibliography

[1] "Understanding the mel spectrogram." `https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53`. Accessed: 2021-04-09.

[2] S. S. Stevens and J. Volkmann, "The relation of pitch to frequency: A revised scale," *The American Journal of Psychology*, vol. 53, no. 3, pp. 329–353, 1940.

[3] B. Sisman, J. Yamagishi, S. King, and H. Li, "An overview of voice conversion and its challenges: From statistical modeling to deep learning," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 132–157, 2021.

[4] 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, no. 7, pp. 3142–3155, 2017.

[5] X. Wang, J. Yamagishi, M. Todisco, H. Delgado, A. Nautsch, N. Evans, M. Sahidullah, V. Vestman, T. Kinnunen, K. A. Lee, L. Juvela, P. Alku, Y.-H. Peng, H.-T. Hwang, Y. Tsao, H.-M. Wang, S. L. Maguer, M. Becker, F. Henderson, R. Clark, Y. Zhang, Q. Wang, Y. Jia, K. Onuma, K. Mushika, T. Kaneda, Y. Jiang, L.-J. Liu, Y.-C. Wu, W.-C. Huang, T. Toda, K. Tanaka, H. Kameoka, I. Steiner, D. Matrouf, J.-F. Bonastre, A. Govender, S. Ronanki, J.-X. Zhang, and Z.-H. Ling, "Asvspoof 2019: A large-scale public database of synthesized, converted and replayed speech," 2020.

[6] "FaceSwap." https://github.com/MarekKowalski/FaceSwap.

[7] J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. NieSSner, "Face2face: Real-time face capture and reenactment of rgb videos," 2020.

[8] S. Suwajanakorn, S. M. Seitz, and I. Kemelmacher-Shlizerman, "Synthesizing obama: learning lip sync from audio," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, pp. 1–13, 2017.

[9] "Deepfakes Software For All: FaceSwap." https://github.com/deepfakes/faceswap.

[10] J. Thies, M. Zollhöfer, and M. NieSSner, "Deferred neural rendering: Image synthesis using neural textures," 2019.

[11] "Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone." https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html.

[12] "Like a phone call: XiaoIce, Microsofts social chatbot in China, makes breakthrough in natural conversation." https://blogs.microsoft.com/ai/xiaoice-full-duplex/.

[13] M. Schröder, M. Charfuelan, S. Pammi, and I. Steiner, "Open source voice creation toolkit for the MARY TTS Platform," in *12th Annual Conference of the International Speech Communication Association - Interspeech 2011*, (Florence, Italy), pp. 3253–3256, ISCA, Aug. 2011.

[14] M. Morise, F. Yokomori, and K. Ozawa, "World: A vocoder-based high-quality speech synthesis system for real-time applications," *IEICE Trans. Inf. Syst.*, vol. 99-D, pp. 1877–1884, 2016.

[15] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[16] K. Tokuda, Y. Nankaku, T. Toda, H. Zen, J. Yamagishi, and K. Oura, "Speech synthesis based on hidden markov models," *Proceedings of the IEEE*, vol. 101, pp. 1234–1252, May 2013.

[17] M. Zakariah, M. K. Khan, and H. Malik, "Digital multimedia audio forensics: past, present and future," *Multimedia Tools and Applications*, vol. 77, pp. 1009–1040, Jan 2018.

[18] R. C. Maher, *Introduction to Forensic Audio Analysis: Authenticity, Enhancement, and Interpretation*, pp. 1–2. Cham: Springer International Publishing, 2018.

[19] N. Bonettini, E. D. Cannas, S. Mandelli, L. Bondi, P. Bestagini, and S. Tubaro, "Video face manipulation detection through ensemble of CNNs," in *International Conference on Pattern Recognition (ICPR)*, 2020.

[20] Y. Li, M.-C. Chang, and S. Lyu, "In ictu oculi: Exposing ai generated fake face videos by detecting eye blinking," 2018.

[21] D. Güera and E. J. Delp, "Deepfake video detection using recurrent neural networks," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, 2018.

[22] F. Matern, C. Riess, and M. Stamminger, "Exploiting visual artifacts to expose deepfakes and face manipulations," *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pp. 83–92, 2019.

[23] A. Lieto, D. Moro, F. Devoti, C. Parera, V. Lipari, P. Bestagini, and S. Tubaro, "'"Hello? Who Am I Talking to?" A Shallow CNN Approach for Human vs. Bot Speech Classification," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[24] E. A. AlBadawy, S. Lyu, and H. Farid, "Detecting ai-synthesized speech using bispectral analysis," in *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2019.

[25] B. Clara, B. Paolo, A. Fabio, S. Augusto, and T. Stefano, "Synthetic speech detection through short-term and long-term prediction traces," *EURASIP Journal on Information Security*, vol. 2021, p. 2, Apr 2021.

[26] M. Todisco, H. Delgado, and N. Evans, "Constant q cepstral coefficients: A spoofing countermeasure for automatic speaker verification," *Computer Speech & Language*, vol. 45, pp. 516–535, 2017.

[27] M. Sahidullah, T. Kinnunen, and C. Hanilçi, "A comparison of features for synthetic speech detection," 2015.

[28] C. Zhang, C. Yu, and J. H. Hansen, "An investigation of deep-learning frameworks for speaker verification antispoofing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 4, pp. 684–694, 2017.

[29] H. Dinkel, N. Chen, Y. Qian, and K. Yu, "End-to-end spoofing detection with raw waveform cldnns," *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar 2017.

[30] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. Wilson, "Cnn architectures for large-scale audio classification," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 131–135, 2017.

[31] J. O. Smith, *Spectral Audio Signal Processing*. `http://-ccrma.stanford.edu/~jos/sasp/`, Accessed: 2021-09-06. online book, 2011 edition.

[32] K. Gurney, *An Introduction to Neural Networks*. An Introduction to Neural Networks, Taylor & Francis, 1997.

[33] 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, IEEE, 2017.

[34] I. Tabian, H. Fu, and Z. Sharif Khodaei, "A convolutional neural network for impact detection and characterization of complex composite structures," *Sensors*, vol. 19, no. 22, 2019.

[35] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[36] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 326–327, 1995.

[37] A. E. Rosenberg, "Automatic speaker verification: A review," *Proceedings of the IEEE*, vol. 64, no. 4, pp. 475–487, 1976.

[38] M. Sahidullah, H. Delgado, M. Todisco, T. Kinnunen, N. Evans, J. Yamagishi, and K.-A. Lee, "Introduction to voice presentation attack detection and recent advances," 2019.

[39] T. Dutoit, *An Introduction to Text-to-Speech Synthesis*. Text, Speech and Language Technology 3, Springer Netherlands, 1 ed., 1997.

[40] M. Rashad, H. M. El-Bakry, I. R. Ismail, and N. Mastorakis, "An overview of text-to-speech synthesis techniques," *Latest trends on communications and information technology*, pp. 84–89, 2010.

[41] P. Taylor, *Text-to-Speech Synthesis*. USA: Cambridge University Press, 1st ed., 2009.

[42] A. I. Mecwan, V. G. Savani, S. Rajvi, and P. Vaya, "Voice conversion algorithm," in *Proceedings of the International Conference on Advances in Computing, Communication and Control*, ICAC3 '09, (New York, NY, USA), p. 615619, Association for Computing Machinery, 2009.

[43] T. Hayashi, A. Tamamori, K. Kobayashi, K. Takeda, and T. Toda, "An investigation of multi-speaker training for wavenet vocoder," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 712–718, 2017.

[44] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. van den Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," 2018.

[45] E. Helander, H. Silen, T. Virtanen, and M. Gabbouj, "Voice conversion using dynamic kernel partial least squares regression," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 3, pp. 806–817, 2012.

[46] M. R. Kamble, H. B. Sailor, H. A. Patil, and H. Li, "Advances in anti-spoofing: from the perspective of asvspoof challenges," *APSIPA Transactions on Signal and Information Processing*, vol. 9, p. e2, 2020.

[47] A. Janicki, "Spoofing countermeasure based on analysis of linear prediction error," in *Proc. Interspeech 2015*, pp. 2077–2081, 2015.

[48] B. Balamurali, K. E. Lin, S. Lui, J.-M. Chen, and D. Herremans, "Toward robust audio spoofing detection: A detailed comparison of traditional and learned features," *IEEE Access*, vol. 7, pp. 84229–84241, 2019.

[49] G. Pinheiro, M. Cirne, P. Bestagini, S. Tubaro, and A. Rocha, "Detection and synchronization of video sequences for event reconstruction," in *IEEE International Conference on Image Processing (ICIP)*, 2019.

[50] H. Xu, Z.-H. Tan, P. Dalsgaard, and B. Lindberg, "Robust speech recognition by nonlocal means denoising processing," *IEEE Signal Processing Letters*, vol. 15, pp. 701–704, 2008.

[51] A. Chambolle, "An algorithm for total variation minimization and applications," *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1, pp. 89–97, 2004.

[52] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1, pp. 259–268, 1992.

[53] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 60–65 vol. 2, 2005.

[54] S. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on Image Processing*, vol. 9, no. 9, pp. 1532–1546, 2000.

[55] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, pp. 425–455, 09 1994.

[56] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pp. 839–846, 1998.

[57] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[58] K. Murphy, *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning series, MIT Press, 2012.

[59] "HTSWorking Group, the english TTS system Flite+HTS engine." `http://hts-engine.sourceforge.net/`.

[60] X. Wang, S. Takaki, and J. Yamagishi, "An autoregressive recurrent mixture density network for parametric speech synthesis," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4895–4899, 2017.

[61] C. Doersch, "Tutorial on Variational Autoencoders," 2016.

[62] Z. Wu, O. Watts, and S. King, "Merlin: An open source neural network speech synthesis system," in *9th ISCA Speech Synthesis Workshop (2016)*, pp. 202–207, Sept. 2016. 9th ISCA Speech Synthesis

Workshop , ISCA 2016 ; Conference date: 13-09-2016 Through 15-09-2016.

[63] "HTS Working Group, An example of context-dependent label format for HMM-based speech synthesis in Japanese (2015)." `http://hts.sp.nitech.ac.jp/archives/2.3/HTS-demo_NIT-ATR503-M001.tar.bz2`.

[64] W.-C. Huang, H.-T. Hwang, Y.-H. Peng, Y. Tsao, and H.-M. Wang, "Voice conversion based on cross-domain features using variational auto encoders," *2018 11th International Symposium on Chinese Spoken Language Processing (ISCSLP)*, Nov 2018.

[65] M. Driss, J.-F. Bonastre, and C. Fredouille, "Effect of speech transformation on impostor acceptance," vol. 1, pp. I – I, 06 2006.

[66] K. Tanaka, H. Kameoka, T. Kaneko, and N. Hojo, "Wavecyclegan2: Time-domain neural post-filter for speech waveform generation," 2019.

[67] X. Wang, S. Takaki, and J. Yamagishi, "Neural Source-filter-based Waveform Model for Statistical Parametric Speech Synthesis," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5916–5920, 2019.

[68] H. Zen, Y. Agiomyrgiannakis, N. Egberts, F. Henderson, and P. Szczepaniak, "Fast, Compact, and High Quality LSTM-RNN Based Statistical Parametric Speech Synthesizers for Mobile Devices," pp. 2273–2277, 09 2016.

[69] Y. Agiomyrgiannakis, "Vocaine the vocoder and applications in speech synthesis," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4230–4234, 2015.

[70] Y. Jia, Y. Zhang, R. J. Weiss, Q. Wang, J. Shen, F. Ren, Z. Chen, P. Nguyen, R. Pang, I. L. Moreno, and Y. Wu, "Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis," 2019.

[71] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan, R. A. Saurous, Y. Agiomvrgiannakis, and Y. Wu, "Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4779–4783, 2018.

[72] L. Wan, Q. Wang, A. Papir, and I. L. Moreno, "Generalized End-to-End Loss for Speaker Verification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4879–4883, 2018.

[73] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. van den Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 2410–2419, PMLR, 10–15 Jul 2018.

[74] H. Kawahara, I. Masuda-Katsuse, and A. de Cheveigné, "Restructuring speech representations using a pitch-adaptive timefrequency smoothing and an instantaneous-frequency-based f0 extraction: Possible role of a repetitive structure in sounds1speech files available. see http://www.elsevier.nl/locate/specom1," *Speech Communication*, vol. 27, no. 3, pp. 187–207, 1999.

[75] K. Kobayashi, T. Toda, and S. Nakamura, "Intra-gender statistical singing voice conversion with direct waveform modification using log-spectral differential," *Speech Communication*, vol. 99, pp. 211–220, 2018.

[76] W.-C. Huang, Y.-C. Wu, K. Kobayashi, Y.-H. Peng, H.-T. Hwang, P. L. Tobing, Y. Tsao, H.-M. Wang, and T. Toda, "Generalization of spectrum differential based direct waveform modification for voice conversion," 2019.

[77] T. Kinnunen, L. Juvela, P. Alku, and J. Yamagishi, "Non-parallel voice conversion using i-vector PLDA: towards unifying speaker ver-

ification and transformation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5535–5539, 2017.

[78] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-End Factor Analysis for Speaker Verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, 2011.

[79] P. Kenny, "A small footprint i-vector extractor," in *Odyssey 2012-The Speaker and Language Recognition Workshop*, 2012.

[80] "Audiomentations - a python library for audio data augmentation." `https://github.com/iver56/audiomentations`. Accessed: 2021-04-15.

[81] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[82] "scikit-image: image restoration module." `https://scikit-image.org/docs/stable/api/skimage.restoration.html#skimage.restoration`. Accessed: 2021-09-01.

[83] "Examples of denoising a picture using total variation, wavelet and bilateral filters." `https://scikit-image.org/docs/stable/auto_examples/filters/plot_denoise.html#sphx-glr-auto-examples-filters-plot-denoise-py`. Accessed: 2021-09-01.