POLITECNICO
MILANO 1863

Dipartimento di Elettronica, Informazione e Bioingegneria

Master Degree in Music and Acoustic Engineering

Time-Scaling Detection in Audio Recordings

by:
Michele Pilia

matr.:
915389

Supervisor:
Prof. Paolo Bestagini

Co-supervisor:
Dr. Sara Mandelli

Academic Year
2019-2020

Abstract

The widespread diffusion of user friendly editing software for audio signals has made audio tampering extremely accessible to anyone. For this reason, it is increasingly necessary to develop forensic methodologies that enable to verify if a given audio content has been digitally manipulated or not.

Among the multiple available audio editing techniques, a very common one is time-scaling, i.e. the alteration of the temporal evolution of an audio signal not affecting any pitch component. For instance, this can be used to slow-down or speed-up speech recordings, thus enabling the creation of natural sounding fake speech compositions.

In this thesis we propose a valid methodology to blindly detect the application of time-scaling to an audio signal. Moreover, our method estimates the time-scaling factor, i.e. the ratio between the duration of the original signal and the time-scaled one.

Our solution is based on a data-driven approach. Specifically, we develop a Convolutional Neural Network that analyzes Log-Mel Spectrogram and phase information of the input audio signal to expose time-scaling.

The proposed technique is tested on a wide dataset of audio tracks that have been edited using three different time-scaling algorithms and several scaling factors. Results show that the Log-Mel Spectrogram is the preferred input to achieve accurate time-scaling detection. Moreover, experiments in a cross-dataset scenario show that the proposed method is able to detect time-scaling regardless of the specific implementation, thus proving the generalization capability of the proposed network.

Sommario

L'elevata diffusione di software di facile utilizzo per editing di segnali audio digitali, ha reso la manomissione dei suddetti segnali estremamente accessibile a chiunque. Per questo motivo, è sempre più necessario lo sviluppo di metodologie forensi che consentano di verificare se un determinato contenuto audio è stato manipolato digitalmente o meno.

Tra le molteplici tecniche di editing audio disponibili, una molto comune è il time-scaling, ossia l'alterazione dell'evoluzione temporale di un segnale audio senza influire su nessuna componente dell'intonazione. Ad esempio, il time-scaling può essere impiegato per rallentare o accelerare le registrazioni vocali, consentendo così la creazione di composizioni vocali false dal suono naturale.

In questa tesi proponiamo una valida metodologia per rilevare l'applicazione del time-scaling su un segnale audio. Inoltre, il nostro metodo stima il fattore di time-scaling, ovvero il rapporto tra la durata del segnale originale e di quello modificato.

La nostra soluzione si basa su un approccio basato sui dati. Nello specifico, sviluppiamo una rete neurale convoluzionale che analizza il Log-Mel Spectrogram e le informazioni sulla fase del segnale audio in ingresso su cui applicare il time-scaling.

La tecnica proposta viene testata su un ampio set di dati di tracce audio che sono state modificate utilizzando tre diversi algoritmi di time-scaling e diversi fattori di scaling. I risultati mostrano che il Log-Mel Spectrogram è l'input preferibile per ottenere un rilevamento accurato del time-scaling. Inoltre, gli esperimenti in uno scenario cross-dataset mostrano che il metodo proposto è in grado di rilevare il time-scaling indipendentemente dall'implementazione specifica, dimostrando così la capacità di generalizzazione della rete proposta.

Ringraziamenti

Un primo ringraziamento va alla mia famiglia, specialmente ai miei genitori, che mi hanno sostenuto in ogni modo sin dal primo giorno di questo percorso, nonostante la lontananza. Grazie di cuore a Chiara che mi è stata vicina e mi ha fatto sentire a casa in ogni momento, alleviando anche i momenti più impegnativi.

Vorrei ringraziare infinitamente il Prof. Paolo Bestagini e la Dott.ssa Sara Mandelli, che mi hanno guidato durante questo progetto con immensa disponibilità e competenza.

Infine, grazie a tutti i colleghi con cui ho trascorso questi anni al Politecnico. In particolare, grazie ad Antonino, con cui ho condiviso la maggior parte dei progetti, e da cui ho imparato tanto.

M.P.

Contents

List of Figures

Glossary

ACC Accuracy. 45

ACV Audio Cooccurrence Vector. 18

AdaGrad Adaptive Gradient Algorithm. 17

Adam Adaptive Moment Estimation. 16, 17

AE Absolute Error. 22, 23

CEL Cross-Entropy Loss. 39, 42, 43

CM Confusion Matrix. vii, 45–47

CNN Convolutional Neural Network. vii, xiii, xiv, 11–13, 15–20, 31, 34, 35, 37, 39, 41–45, 51, 53, 57, 58, 60–62, 64, 66–69

DFT Discrete Fourier Transform. 2, 3

DL Deep Learning. 12

DTFT Discrete-Time Fourier Transform. 2

ENF Electrical Network Frequency. 18

FFT Fast Fourier Transform. 3, 33

FM Feature Map. 12, 35, 36

FT Fourier Transform. 1–3, 9, 20, 25, 27–30

GAN Generative Adversarial Network. 19

GLCM Gray-Level Cooccurrence Matrix. 18

HPF-Ph High-Pass Filtered unwrapped Phase. x–xii, 32, 41, 45–49, 52, 54, 55, 58–64

IDFT Inverse Discrete Fourier Transform. 3

IFT Inverse Fourier Transform. 1, 2, 11

LMS Log-Mel Spectrogram. vii, x–xii, 20, 32, 34, 41, 44–47, 49–52, 54, 55, 57–59, 61–64

LMSC Log-Mel frequency Spectral Coefficients. 19

LPC Linear Prediction Coding. 19

LR learning rate. 16, 17, 43, 44

MAE Mean Absolute Error. 23

MBConv Mobile inverted Bottleneck Convolution. vii, x, 34, 36, 37

MedResPh Median Residual unwrapped Phase. vii, x–xii, 32, 41, 46–49, 52–57, 59–64

MFCC Mel-Frequency Cepstrum Coefficients. 19

ML Machine Learning. 13

MSE Mean Square Error. 23, 39, 42

NLL Negative Log-Likelihood. 42

NN Neural Network. 11, 19

PCC Pearson Correlation Coefficient. x–xii, 19, 48, 51, 53–55, 57–64, 68

PV-TSM Phase Vocoder Time-Scaling Modification. 10, 11, 41

RMSProp Root Mean Square Propagation. 17

SE Square Error. 22, 23

SGD Stochastic Gradient Descent. 16

STFT Short-Time Fourier Transform. vi, vii, xiii, xv, 3–5, 11, 24, 25, 27–30, 32–34, 66, 68

SVM Support Vector Machine. 19

List of Tables

Introduction

The widespread diffusion of electronic devices, such as smartphones, tablets, or portable PCs, together with the recent drastic technological evolution, resulted in a high ease of production of user-generated contents. These multimedia contents can be generated by means of professional, but increasingly easier to use, digital editing tools.

Because of the aforementioned simplicity to generate tampered contents, nowadays multimedia files spread at extreme speed, and it is challenging to detect if a digital content has been altered or not.

These phenomena have led to the study and research of multimedia forensic methods, in order to identify whether a given image, video, or audio content is the original one or a digitally-tampered version.

Up to now, several audio forensics techniques have been developed in order to detect tampering and forgeries of various kinds. As we will describe in the next chapters, the detection of tampering in audio signals can be complicated by the application of post-processing operations.

The main goal of this works concerns the identification of time-scaling operations on audio signals. We developed a solution to perform time-scaling detection and estimation, i.e. to predict if an input audio signal has been modified by means of a time-scaling algorithm, and possibly to estimate the correct value of the time-scaling factor. Typically, time-scaling modification algorithms leaves some interpolation traces on the Short-Time Fourier Transform (STFT) of the input signal. For this reason, we tried to blindly investigate audio signals in order to find such artifacts. The method we implemented strongly relies on a data-driven approach. Specifically, we developed a Convolutional Neural Network (CNN)-based solution, whose training process has been conducted with original audio signals and time-scaled ones.

The two fundamental goals of this thesis are:

- Classification, i.e. detecting if the input signal is original or if it has been processed by means of a time-scaling algorithm. Therefore, the CNN must return as output a discrete number indicating the presence or not of time-scaling operations. Namely, 1 if the system has encountered the application of a time-scaling algorithm, or 0 if the signal has been identified as original, i.e. with no time-scaling modifications.

- Regression, consisting of the estimation of the time-scaling factor α of the possibly applied time-scaling algorithm. In this case, the output value is a continuous number, indicating the prediction about the time-scaling factor.

As far as classification is concerned, the approach consists of analyzing a single temporal window composed by 64 frequency bins and 96 time samples extracted for each input audio signal, and making a prediction about the presence or not of time-scaling operations on that signal.

Regarding the task of regression, we identified two different approaches:

1. The "Single time-window" approach, consisting of making a prediction for a single temporal window of 64 frequency bins and 96 time samples for every input audio signal (i.e. likewise we proceeded for the classification task);

2. The "Multiple time-window" approach, implying that each input signal is divided into many subsequent temporal windows, then the overall estimation about the signal time-scaling factor is produced by computing the mean, the median value or the mode between all the predictions about the temporal windows of the same input signal.

For each audio signal, we feed our CNN with different 2D representations of the signal itself: the Log-Mel spectrogram, a high-pass filtered version of the unwrapped phase, a median filtered version of the unwrapped phase, and the three of them together.

The dataset needed for the training process of the employed CNN consists of original audio signals and time-scaled ones, obtained by means of three different algorithms: Audiotsm [5], Torchaudio [6] and Time-stretch-master [7].

We conducted all the experiments either by using the same dataset of the training stage and by adopting a cross-dataset approach, i.e. choosing two different datasets for training and test. This allowed us to measure the ability of the CNN to recognize general artifacts of time-scaling algorithms regardless of the specific algorithm used. The achieved results show that our solution is appropriate to solve the goals of this work. Specifically, the best outcomes have been obtained with the Log-Mel spectrogram as input, and in cross-dataset tests between Torchaudio and Time-stretch-master related datasets. On the contrary, we need further investigations to improve results for tests with unwrapped phases as input, and for cross-dataset configurations with the Audiotsm related dataset used during the testing stage.

This thesis is organized as follows.

Chapter 1 covers the basic concept which this work is based on, and that will be necessary to understand the rest of the thesis. Moreover, we present a brief overview about the state of the art in the multimedia forensics field. In Chapter 2 we formally define the problem that we want to solve with this work, relying upon a mathematical formulation. Chapter 3, instead, contains the description of all the preliminary analysis that we performed about the traces left by time-scaling algorithms on the audio signal's STFT and its phase. In Chapter 4 we describe the methodology that we devised to achieve the fundamental goal of this thesis. In particular, we will go into the details of the architecture of the chosen system. In Chapter 5, we analyze in details the achieved results for each task. Finally, in Chapter 6, we point out the possible future directions suggested by this work.

1
Theoretical Background

This chapter introduces the readers to the basic concepts that will be necessary to understand what we will discuss in the following chapters.

1.1 Time-Frequency Audio Representation

In this section we will focus on the standard concepts that are used to represent audio signals in the time and frequency domain, which a great part of this work is based on.

1.1.1 The Fourier Transform

The Fourier Transform (FT) has been theorized by Jean Baptiste Joseph Fourier in 1822, and it is an operator that allows to pass from the time domain to the frequency domain, and vice versa [8]. Indeed,if we express an analog signal $s(t)$ as a function of time t, we can describe it in the frequency domain with the following relationship:

$$S(f) = \int_{-\infty}^{+\infty} s(t)e^{-i2\pi ft}dt, f \in \mathbb{R}, \tag{1.1}$$

where f is the frequency and i the imaginary unit. The equation (1.1) is then the FT, while its counterpart, the Inverse Fourier Transform (IFT), is used to move from the frequency domain to the representation as a function of the time, and it is expressed as follows:

$$s(t) = \int_{-\infty}^{+\infty} S(f)e^{i2\pi ft}df, t \in \mathbb{R}. \tag{1.2}$$

Briefly, we can write $S(f) = \mathcal{F}[s|f]$ and $s(t) = \mathcal{F}^{-1}[S|t]$, where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are respectively the FT and IFT of the signal. Another way to express them is $s(t) \xrightarrow{\mathcal{F}} S(f)$, $S(f) \xrightarrow{\mathcal{F}^{-1}} s(t)$.

The above definitions are valid when we are in the analog domain, so when we talk about continuous signals. However, in our work, we are more interested in discrete-time signals, complex functions that we can define as:

$$s : \mathbb{Z}(T) \to \mathbb{C}, \tag{1.3}$$

with $\mathbb{Z}(T)$ being the set of the multiples of T:

$$\mathbb{Z}(T) = \{\cdots, -T, 0, T, 2T, \cdots\}, \tag{1.4}$$

and $T > 0$ is the *sampling period*, i.e. the time-spacing between the n instants where the signal is defined, so that $F_s = 1/T$ is the *sampling rate*, namely how many samples are aquired for each unitary time.

The signal (1.3) is usually denoted as $s(nT)$, $nT \in \mathbb{Z}(T)$. The FT of discrete-time signals is called Discrete-Time Fourier Transform (DTFT) and it is described by:

$$S(\Omega) = \sum_{n=-\infty}^{+\infty} s(nT)e^{-i\Omega n}, \tag{1.5}$$

where $\Omega = \omega/F_s = 2\pi fT$ is the *normalized frequency*, and the Inverse-DTFT is:

$$s(nT) = \frac{1}{2\pi}\int_{0}^{2\pi} S(\Omega)e^{i\Omega n}d\Omega. \tag{1.6}$$

In order to use electronic processors, we need to work with finite sequences of values, both in the time and frequency domain. For this reason, we will consider only a finite sequence of N time values:

$$s_n = \{s_0, s_1, \cdots, s_{N-1}\}, \tag{1.7}$$

and a finite sequence of N frequencies:

$$S_k = \{S_0, S_1, \cdots, S_{N-1}\}, \tag{1.8}$$

basically obtained by sampling $S(\Omega)$ at intervals of $\frac{2\pi}{N}$, where each frequency discrete sample is denoted as “bin”. Finally, this is achieved by means of the Discrete Fourier Transform (DFT):

$$S_k = \sum_{n=0}^{N-1} s_n e^{-i\frac{2\pi}{N}kn}, \tag{1.9}$$

while the Inverse Discrete Fourier Transform (IDFT) is expressed by:

$$s_n = \frac{1}{N} \sum_{k=0}^{N-1} S_k e^{i \frac{2\pi}{N} kn}. \tag{1.10}$$

As we mentioned before, the final goal of the DFT is to be used by eletronic computers, but, since it exhibits a computational complexity of $O(N^2)$, generally a more efficient version is implemented, that is the Fast Fourier Transform (FFT), with a complexity of $O(N \log N)$.

1.1.2 The Short-Time Fourier Transform

Previously, we described how the FT represents a signal as a linear combination of complex exponentials (1.1), but it does not give any information about time evolution, i.e. how the frequency behaves as the time changes. The algorithm that allows us to describe signals in a combined time-frequency representation is the Short-Time Fourier Transform (STFT), whose basic idea consists of dividing a signal into shorter segments by means of a window w, and performing the DFT over them [9]. From now on, for the sake of simplicity, we will simply denote a discrete signal as $s(n)$. Its STFT is defined as:

$$\text{STFT}\{s(n)\}(m,k) = S(m,k) = \sum_{n=-\infty}^{+\infty} s(n) w(n-m) e^{i \frac{2\pi}{M} kn}. \tag{1.11}$$

As we mentioned before, w is a *window function* localized at the time instant n that takes only a portion of the signal around n. The length M of w determines the behavior of the STFT in terms of time and frequency resolution, indeed, a narrow window provides good time resolution but poor frequency resolution, and vice versa for wide windows. Some popular window functions are, for example, the **Rectangular** window (Fig. 1.1), defined as:

$$w_R(n) \triangleq \begin{cases} 1, & -\frac{M-1}{2} \leq n \leq \frac{M-1}{2} \\ 0, & \text{otherwise} \end{cases}, \tag{1.12}$$

or the **Hanning** window (Fig. 1.2):

$$w_H(n) \triangleq \frac{1}{2} w_R(n) \left[1 + \cos(\frac{2\pi n}{M}) \right]. \tag{1.13}$$

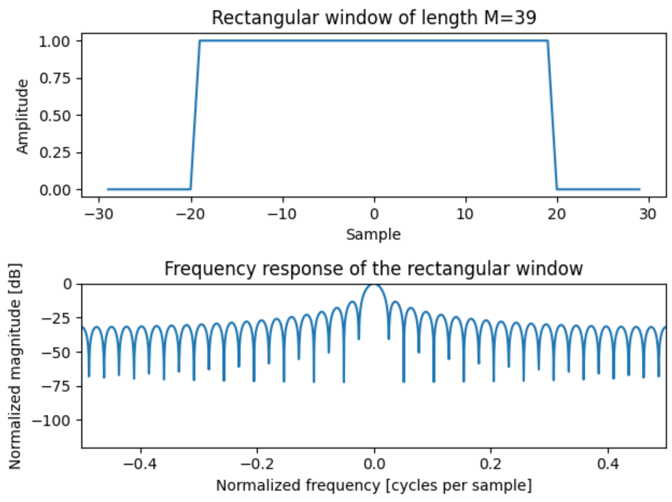


Figure 1.1: A Rectangular window and its frequency response.

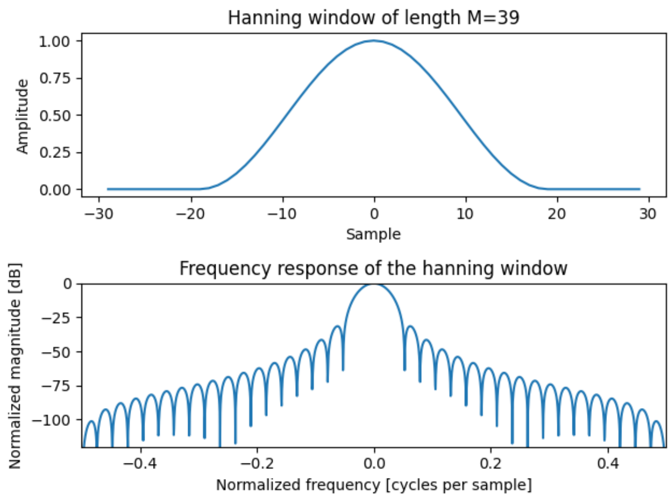


Figure 1.2: A Rectangular window and its frequency response.

1.1.3 Spectrogram

The STFT of a signal can be decomposed as follows [10]:

$$S(m,k) = |S(m,k)|\, e^{i\phi(m,k)}, \tag{1.14}$$

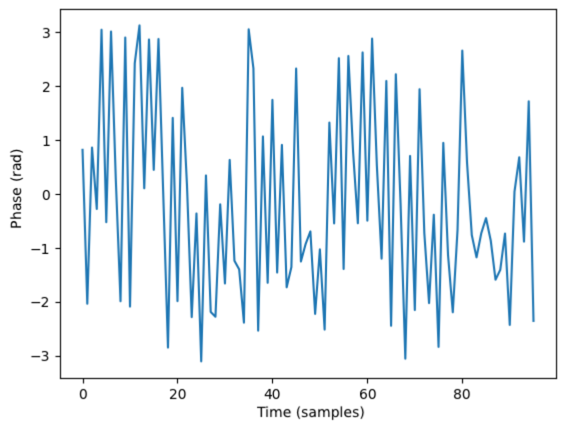
with

$$\phi(m,k) = \angle(S(m,k)), \tag{1.15}$$

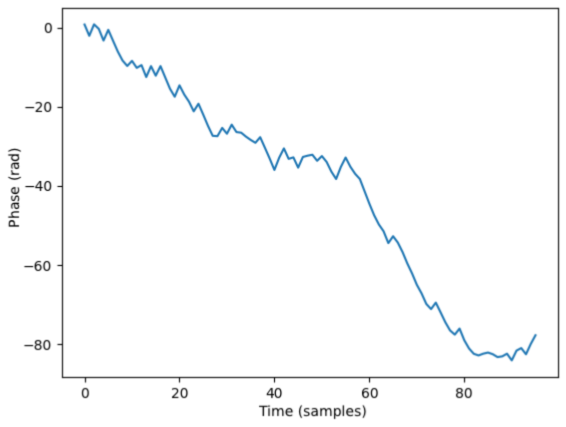
where $|S(m,k)|$ is the **spectrogram** of the signal (or magnitude of the STFT), and $\phi(m,k)$ is the **phase** spectrum of the STFT.

In addition, we can define the unwrapped phase of the STFT [11] as:

$$\varphi(m,k) = \phi(m,k) + 2\pi p(m,k), \tag{1.16}$$



(a) Wrapped phase time evolution for a single frequency bin.



(b) Unwrapped phase time evolution for a single frequency bin.

Figure 1.3: The phase unwrapping process.

where $p(m,k)$ is an integer chosen appropriately so as to cancel any discontinuity between consequent frames such that:

$$|\varphi(m,k) - \varphi(m+1,k)| \geqslant \pi. \tag{1.17}$$

Starting from the concept of spectrogram we just defined, we end up with a really powerful visual representation tool that allows to see the spectral evolution of a signal over time. Typically a spectrogram is represented in a graph with two dimensions: on the horizontal axis we have the time evolution, while on the vertical one there are the frequencies, arranged on several possible scales. We usually represent the behaviour of the signal spectrogram (i.e. the STFT magnitude) as a function of time m and frequency k.

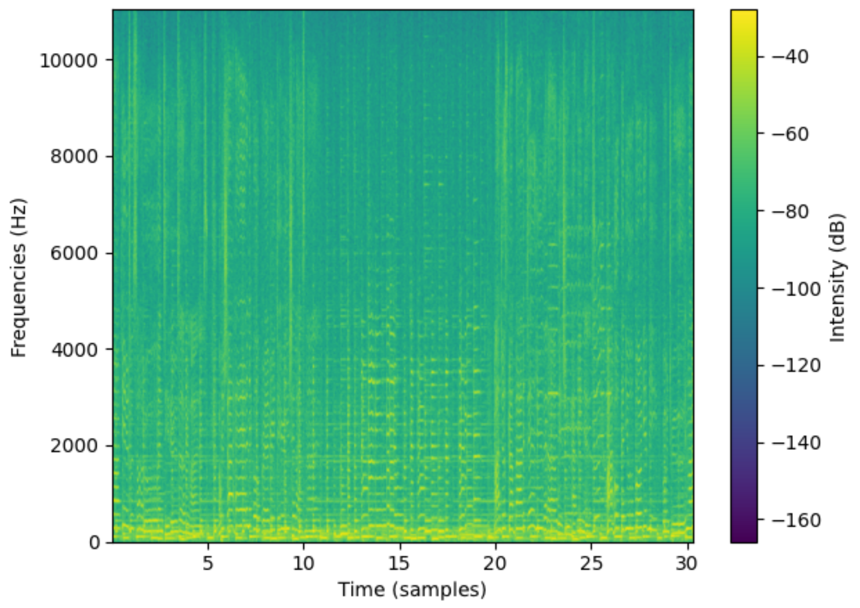


Figure 1.4: A spectrogram of a .wav audio file

1.1.4 Mel Scale

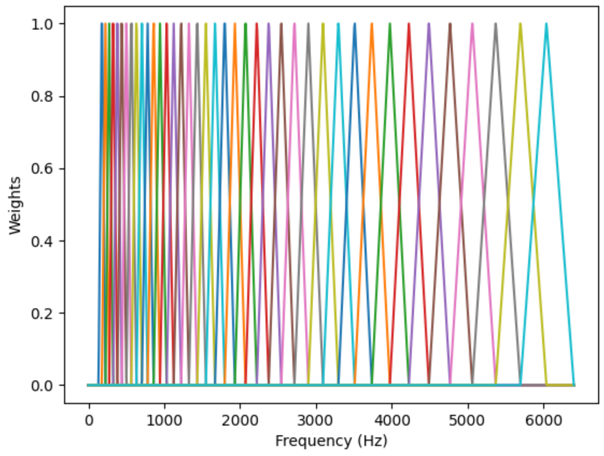
The **Mel-Scale** is the result of psychoacoustics research, relating real frequencies to the perceived pitch, starting from earlier studies lead by Stevens and Volkman [12]. From these studies emerged that human ear resolves frequencies in a non-linear fashion across the audio spectrum [13]. The human ear's behaviour is simulated with the so-called *Mel filterbank*, that is a set of triangular filters, usually 40, whose center frequencies are located on the Mel scale. The two extreme minima of each filter are located on the center frequency of the two adjacent ones, according to an equation that puts in relationship the index of the bin of the filter bank with its center frequency f[Hz]. In the literature there are two main formulas to define the frequency arrangement on the Mel scale. The *HTK* formula [14] maps the frequency range (Hz) to Mel values:

$$\mathrm{Mel}(f) = m = 2595 * \log_{10}\left(1 + \frac{f}{700}\right), \tag{1.18}$$

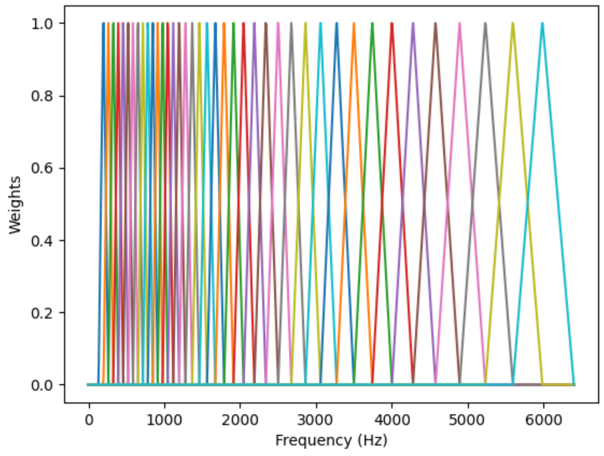
while the *Slaney* algorithm [15] maps the Hertz frequency range on a range from 0 to 40:

$$\mathrm{Mel}(f) = m = \begin{cases} (f - f_l)/66.67 & \text{if } f \leq 1000\text{Hz}, \\ 13 + \frac{\ln(f/1000)}{\ln(f_h/1000)} & \text{otherwise}, \end{cases} \tag{1.19}$$

where f_l is the lowest frequency of the filter bank (generally 133.33Hz) and f_h is the highest one (generally 6400Hz). The meaning of (1.19) is



(a) HTK Mel-scaled filterbank



(b) Slaney Mel-scaled filterbank

Figure 1.5: Two Mel filterbanks according to the different formulas

that, on 40 filters $H_m(k)$, where m is the filter index and k the discrete frequency, we have the first 13 linearly spaced by 66.67Hz followed by other 27 *log-spaced* ones.

We will refer to $\Omega(m)$ as the inverse function of $\mathrm{Mel}(f)$, by means of which we can compute the frequency in Hertz starting from the value on the Mel scale, refering to either the HTK or Slaney formula. The filterbanks described in Fig. 1.5 will be denoted from now on as $\mathrm{fb}_{\mathrm{mel}}$, whose triangular filters $H_m(k)$ are obtained from (1.18) and (1.19) with the following:

$$H_m(k) = \begin{cases} 0, & k < \Omega(m-1) \\ \frac{k-\Omega(m-1)}{\Omega(m)-\Omega(m-1)}, & \Omega(m-1) \leqslant k \leqslant \Omega(m) \\ \frac{\Omega(m+1)-k}{\Omega(m+1)-\Omega(m)}, & \Omega(m) \leqslant k \leqslant \Omega(m+1) \\ 0, & k > \Omega(m+1) \end{cases}. \tag{1.20}$$

1.1.5 Log-Mel Spectrogram

An important tool that is widely used for speech and audio processing is the Log-Mel spectrogram. Its working principle is analogous to the classic spectrogram described before, but it is computed in another way:

$$\text{LMS} = \ln(\text{fb}_{\text{mel}} \cdot |S(n,f)| + \varepsilon), \tag{1.21}$$

where fb_{mel} is the aforementioned mel-scaled filterbank matrix, ε is the logarithmic offset (an arbitrarily small number used to avoid the computation of the natural logarithm of zero) and $\cdot$ computes the matrix multiplication between the Mel filterbank fb_{mel} and the spectrogram $|S(n,f)|$. The obtained LMS is a 2-D matrix that will be displayed in the same way as in the classic case.

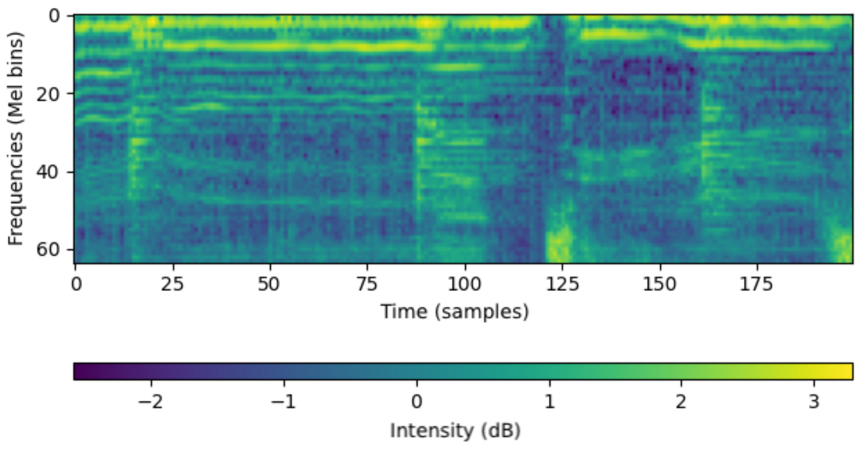


Figure 1.6: A Log-Mel Spectrogram of a .wav audio file.

1.2 Time and Pitch Scaling

In this section we will cover the fundamental principles at the basis of time and pitch scaling modification algorithms, including the main purposes and some formal definitions.

1.2.1 Main Idea

Time-scaling (or time-stretching) is used to control the time evolution (i.e. the duration) of a signal independently from its frequency behavior. The aim is to speed up or slow down the audio signal without changing its spectral content (in particular, the pitch in the case of a periodic signal) [16]. There are several applications in which time-scaling might be significantly useful, for instance:

- **Post-synchronization:** In cinema and music industry it often happens that different tracks of a song or images and dialogues of a movie have been prepared separately, so when they have to be put together there might be the need to adapt the speed of some audio signal;
- **Data compression:** Makhoul and El-Jaroudi in 1986 [17] theorized the use of time-scale modifications in order to compress data for storage or communications. Data can be compressed when stored or transmitted, by shrinking them, and expanded back once received or loaded. However, this method it is not used anymore since it can be applied only a limited number of times for each audio track, and fortunately now we have better data compression algorithms;
- **Reading by listening:** For blind people, listening is the only valid alternative to reading. However, since on average people can listen faster than they can read, time-scale modifications can increase the listening rate and the amount of usable information.

By contrast, *pitch-scaling* is referred to as the operation of altering the frequency content of an audio signal without changing its time evolution.

1.2.2 Time scaling definition

According to the FT, time and frequency evolutions are not completely independent, so it can be complex to determine a model to explain time-modification without referring to pitch-scaling. This goal can be accomplished by adopting a parametric model for audio signal. Maybe the simplest model we can use to start defining these algorithms is the quasi-stationary model introduced by [18, 19], in which the signal $s(t)$ is represented as a sum of $I(t)$ sinusoids with time-varying instantaneous frequency $\omega_i(t)$, phase $\phi_i(t)$ and amplitude $A_i(t)$:

$$s(t) = \sum_{i=1}^{I(t)} A_i(t) e^{j\phi_i(t)}, \tag{1.22}$$

with

$$\phi_i(t) = \int_{-\infty}^{t} \omega_i(\tau) d\tau. \tag{1.23}$$

From a mathematical point of view, the time-scale modification can be seen as a mapping between time in the original signal and time in the modified one. This mapping is described by the following equation:

$$t \to t' = T(t) = \int_0^\tau \frac{\tau}{\alpha} d\tau, \tag{1.24}$$

where $\alpha > 0$ is the so-called *time-modification rate*. When $0 < \alpha < 1$, the signal is slowed-down, whereas, for $\alpha > 1$, the signal is speeded-up by means of time-scale compression. Finally, if we refer to the sinusoidal model described in (1.22) and (1.23), the time-scaled signal can be described as:

$$s'(t') = \sum_{i=1}^{I(T^{-1}(t'))} A_i(T^{-1}(t')) e^{j\phi'_i(t')}, \tag{1.25}$$

with

$$\phi'_i(t') = \int_{-\infty}^{t'} \omega_i(T^{-1}(\tau)) d\tau. \tag{1.26}$$

The equation (1.25) expresses that the amplitude of the i_{th} sinusoid of the time-scaled signal at the instant t' is equal to the amplitude of the same sinusoid of the original signal at time $t = T^{-1}(t')$. In the same way, if we compute the derivative of $\phi'_i(t')$ with respect to t' we can notice that the instantaneous frequency at time t' in the original signal is the same of the original one at time $t = T^{-1}(t')$. This means that the time evolution has changed, while the frequency content has not.

1.2.3 Phase Vocoder Time-Scaling Modification

In this subsection we will describe the time-scaling algorithm we will refer to in the continuation of this work: the Phase Vocoder Time-Scaling

Modification (PV-TSM) [20]. The first step of the PV-TSM is to divide the input signal $s(n)$ into smaller analysis frames $s_m(n)$, where $m \in \mathbb{Z}$ identifies the analysis frame, spaced by an analysis hopsize H_a, in order to compute $S(m, f)$, the STFT of the input signal. The crucial step of the PV-TSM consists of computing a modified STFT of $s(n)$:

$$S^{\text{Mod}}(m, f) = |S(m, f)|\, e^{2\pi i \phi^{\text{Mod}}(m,k)}, \tag{1.27}$$

where $\phi^{\text{Mod}}(m, k)$ is the *adjusted phase*, computed to avoid artifacts, with the phase propagation iterative process as:

$$\phi^{\text{Mod}}(m+1, k) = \phi^{\text{Mod}}(m, k) + F^{\text{IF}}_{\text{coef}}(m, k)\frac{H_s}{F_s}. \tag{1.28}$$

$F^{\text{IF}}_{\text{coef}}(m, k)$ is the instantaneous-frequency coefficient at bin (m, k), H_s the synthesis hop size, i.e. the spacing between synthesis frames (1.29), and F_s the sampling rate. The modified analysis frames $s^{\text{Mod}}_m(n)$, $m \in \mathbb{Z}$, are obtained by means of the IFT of $S^{\text{Mod}}(m, f)$, then they are relocated on the time axis in order to perform the time-scale modification, obtaining the synthesis frames:

$$y_m(n) = \frac{w(n) s^{\text{Mod}}_m(n)}{\sum_{u\in\mathbb{Z}} w(n - uH_s)^2}. \tag{1.29}$$

Finally, the output time-scaled signal $y(n)$ is computed from the synthesis frames as:

$$y(n) = \sum_{m\in\mathbb{Z}} y_m(n - mH_s). \tag{1.30}$$

1.3 Convolutional Neural Networks

In this section we will define the basic concepts at the basis of the main architecture that we used for our project, i.e. the Convolutional Neural Network (CNN).

1.3.1 General Idea and Definitions

A CNN is an artificial feed-forward Neural Network (NN), whose connectivity pattern between neurons is based on the animal visual cortex organization [21].

CNNs are trainable architectures composed of multiple stages [22]. This concept was introduced in 1989 by Yann LeCun [23], who, following the concept of *Cognitron* (the very first NN theorized by Kunihiko

Fukushima in 1975 [24]), developed LeNet, a CNN with the aim of recognizing handwritten digits. CNNs addressed a niche market in the postal and banking sector until 2010s, when new algorithms and technology improvements, such as the possibility to use very large datasets and to perform computations on GPUs, brought forth better results. Thanks to these improvements, now CNNs are widely used for many purposes like image and auditory perception, language understanding and many others.

Each stage of a typical CNN has a set of Feature Maps (FMs) as input and another one as output, in such a way that each output FM represents a specific feature extracted at all locations of the input. Typically, a CNN is composed by *layers*, including, as the most used [22]:

- **Filtering (Kernel) Layer:** the filtering layer contains, as input, a 3D array, x_i, with n_1 2D FMs with size $n_2 \times n_3$, while the output is composed by a 3D matrix, called y_j, with a $m_1 \times m_2 \times m_3$ FM. The filter is a trainable convolution matrix (kernel) denoted as k with dimensions $l_1 \times l_2$ and the same depth of the input FM that connects x_i to y_j. The filter computes $y_j = b_j + \sum_i k_{ij} * x_i$ with b_j being a trainable bias parameter, and $*$ the convolution operator, meaning that each filter detects a particular feature at every input location. The filter moves with a certain step (*stride*) in both horizontal and vertical dimension, in order to cover the entire input FM. A convolutional layer, in order to be completely described, needs certain parameters to be described, such as the kernel with its size, the number of input/output channels, and some convolution hyperparameters like padding or stride.

- **Non-Linearity Layer:** In this layer a so-called *activation function* is applied to each point of the activation map (FM) of the previous layer. Some typical applied functions are the **Rectified Linear Unit** (ReLU: $f(x) = \max(0, x)$), the **Hyperbolic Tangent** ($f(x) = \tanh(x)$) and the **Sigmoid** ($\sigma(x) = (1 + e^{-x})^{-1}$). Sometimes, this function is followed by some sort of normalization, to enforce local competition between adjacent features, as suggested by visual neuroscience models [25, 26].

- **Feature Pooling Layer:** Generally, a pooling layer of a CNN consists of a function that replaces the output of the net at a certain location with a summary statistic of the nearby outputs [27]. Some popular pooling functions used in Deep Learning (DL) are the *max pooling* function, that takes the maximum output inside a rectangular neighborhood, the *average pooling*, the L^2 *norm* or the *weighted average*, based on the distance of the pixels from the center of the rectangular neighborhood. The pooling layer gives several benefits, such as the spatial size reduction, with a consequent decrease of the required computational power, noise reduction in the case of the *max pooling* function, and finally it is useful because it makes the representation substantially invariant to small variations. This property is important if we care more about *detecting* a certain feature rather than identifying exactly *where* it is.

The idea behind CNNs has its origin in the studies conducted by D. H. Hubel and T. N. Wiesel on the cat's and monkey's primary cortex [21, 28]. In these works they pointed out that in visual cortexes there are neurons that individually respond to certain portions of the visual field, called **receptive field** of the neuron. They identified *simple* cells with local receptive fields, whose task is analogous to the *ConvNets* filter bank layers, and *complex* cells, whose function is similar to the pooling layers.

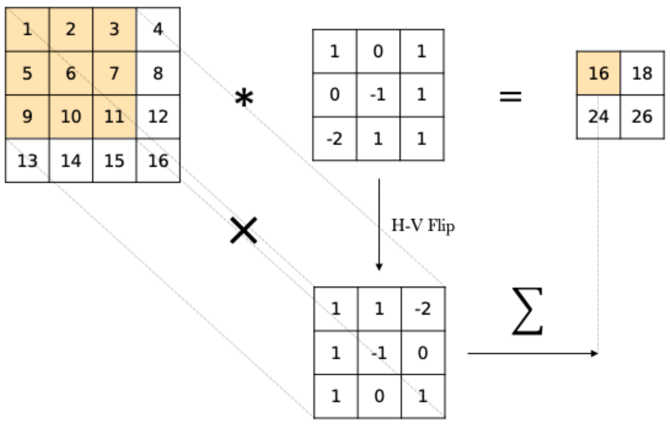
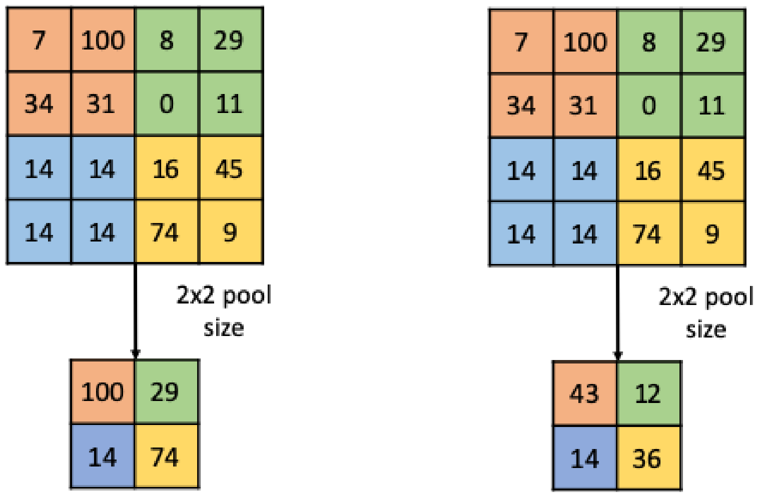


Figure 1.7: An example of convolution between a matrix and a Kernel filter.

1.3.2 Classification and Regression

Machine Learning (ML) problems can be split into two main categories: **supervised** and **unsupervised** learning. In the former we utilize known datasets, by means of *labels*, to make predictions, while in the latter we *infere* structures about data without any prior information about them.



(a) Max pooling (b) Average pooling

Figure 1.8: Two examples of pooling.

Supervised Learning is the category we are interested in most, and it can be defined through the following attributes:

- **Data:** dataset $\mathcal{D}$ is composed of n samples, such that $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ and $D_i = (x_i, y_i)$, where $x_i : (x_{i,1}, x_{i,2}, ..., x_{i,d})$ is the input vector of size d and y_i is the desired output (target or label);
- **Goal: learning** the mapping between input and output $f : X \to Y$ such that $y_i \approx f(x_i), \forall i = 1, ..., n$.

Supervised learning is itself divided into two main groups: **Classification** and **Regression**, whose substantial difference is that in classification the output Y is **discrete**, while in regression Y is **continuous**. For example, a typical regression problem can be to guess the price of a house starting from some of its features, such as the size or the age (Fig. 1.10), while an example of a classification problem can be the identification of the animal in a picture (Fig. 1.11).

After that, both tasks require a *loss computation* stage, that performs the calculation of a so-called loss (or cost) function. The loss function

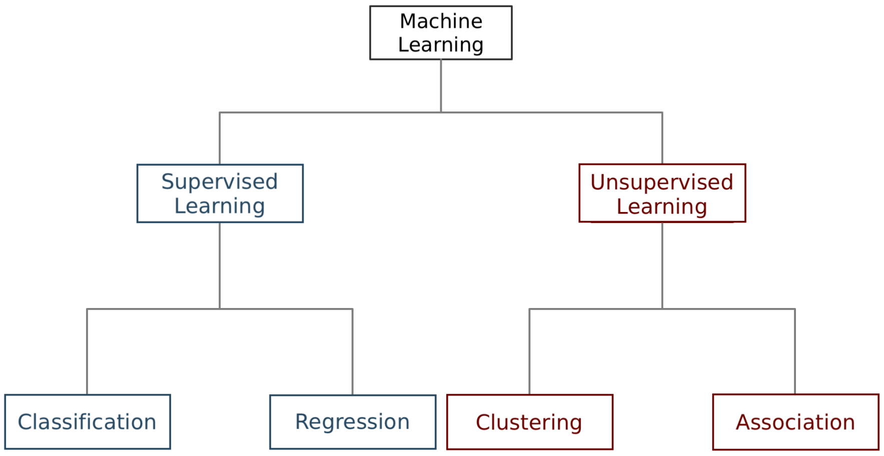


Figure 1.9: A summary diagram of the main goals of Machine Learning

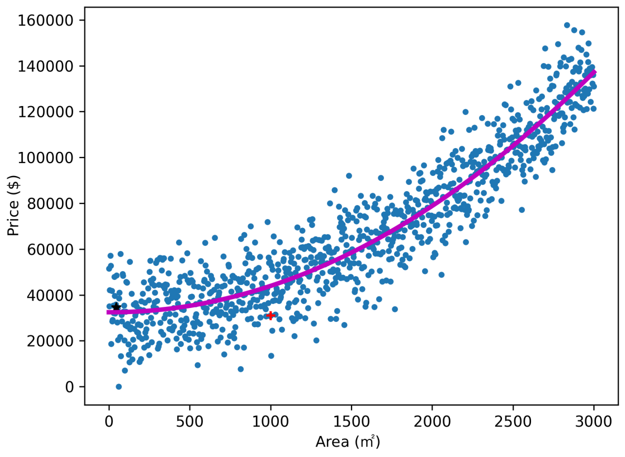


Figure 1.10: An example of Regression

maps an event of the system to a real number that represents the behavior of the CNN, so that all the complexity can be reduced to a unique single scalar value. The importance of such reduction lies on the fact that it makes the system comparable with others and, specially, allows us to measure its quality. Concretely, a loss function measures how much our system's results $f(x_i)$ differ from the expected ones y_i [29].

1.3.3 How to train a CNN

Once the loss is computed, the problem of training a CNN is equivalent to the problem of minimizing the loss function. At the very first step of the training process the trainable filter weights k_{ij} are randomly initialized, hence the loss is quite high, but, as we said before, the goal is to minimize

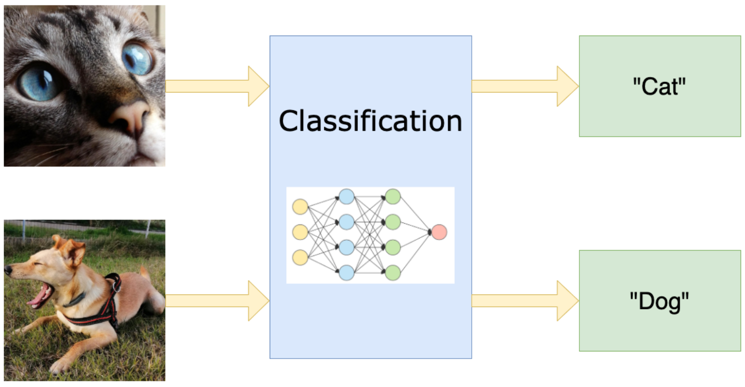


Figure 1.11: An example of Classification

it by adjusting such weights. The procedure by which we try to minimize the loss of a CNN is called **back-propagation**, that has the goal of computing the *gradient* (1.31) of the loss with respect to the filter weights, so that it can be set to zero by optimizing the Kernel weights.

$$\nabla f(p) = \begin{cases} \frac{\partial f}{\partial k_1}(p) \\ \vdots \\ \frac{\partial f}{\partial k_n}(p) \end{cases}, \tag{1.31}$$

where

$$p = (k_1, \cdots, k_n). \tag{1.32}$$

As we described in (1.31) and (1.32), the gradient of a function is the vector field containing at the point p all the partial derivatives of that function at p, and it is also a measure to understand the direction of the fastest increase of the function $f(p)$.

After the gradient is computed with back-propagation, we need to update the weights by means of an optimization algorithm. One of the most popular ones is the Stochastic Gradient Descent (SGD), whose update rule is $k_{ij} := k_{ij} - \lambda \frac{\partial L}{\partial k_{ij}}$, where L is the loss function (or *cost function*) and λ is the so-called learning rate (LR). Another important optimization algorithm is the Adaptive Moment Estimation (Adam), that computes adaptive LRs for each parameter, instead of keeping the same LR for all the trainable weights, and, following the Momentum optimization technique, takes into account the gradient of past steps to guide the optimization process [30]. Adam algorithm combines the advantages of

Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) [31, 32], resulting in the key features we described before. The update rule for the $(ij)^{\text{th}}$ weight at time step $t+1$ according to the Adam optimization algorithm is the following [33]:

$$k_{ij,t+1} := k_{ij,t} - \frac{\lambda \times \hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon}, \tag{1.33}$$

where

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t}, \tag{1.34}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t}, \tag{1.35}$$

and where

$$m_t = (1 - \beta_1) g_t + \beta_1 m(t-1), \tag{1.36}$$

$$v_t = (1 - \beta_2) g_t^2 + \beta_2 v(t-1). \tag{1.37}$$

Finally, there are only a few parameters remained to be defined, in order to complete the definition of the Adam algorithm: ϵ is an arbitrarily small number chosen in order to avoid the division by zero, g is the gradient defined as $\nabla J(k_{ij,t+1})$, and β_1 and β_2 are the so-called the exponential decay rates, usually chosen as $\beta_1 = 0.9$, $\beta_2 = 0.999$.

LR is one of the most crucial hyperparameters for CNNs, since it is very important to find a good compromise on its value, so as not to choose it too big, avoiding to find an optimal solution, nor too small, making this way the computation very slow and inefficient.

A useful set of tools generally used in the training process of CNNs are the so-called callback functions, i.e. a set of functions that allow to keep track of the situation of training in real-time, and eventually to early-stop the process if no improvements are brought about the learning point of view. Callbacks include some functions to dynamically modify the LR, such as the “Reduce LR on plateau”, that automatically reduces the LR when it reaches the so-called *plateau*, i.e. a quasi-constant value (hopefully a minimum).

Another important concept is the notion of *epoch*: the number of epochs is a parameter that denotes how many times the learning algorithm will pass through the entire training subset. During a single epoch each sample in the training subset has had the possibility to adjust its trainable weights. To understand the relationship between epochs and batches, an epoch consists of one or more batches.

Once a good approximation about the optimal filter weights has been found, we can say that the training process has ended, namely our CNN has *learned* how to answer at its best to our inputs, meaning that the filter weights are set so that the error (i.e. the loss) between the network's output and the desired one is minimized.

1.4 Audio Forensics: State of the Art

The diffusion of electronic devices and multimedia platforms to share digital user-generated contents has been growing exponentially in the last decades. Moreover, there are more and more softwares on the market that allow users to edit multimedia content in a very simple way with professional results. Finally, multimedia files are usually employed in law courts as evidence for trials. For these reasons, the evolution and development of multimedia forensics solutions have been crucial in the recent period.

1.4.1 Related Methods

Several research fields have been developed on the topic of audio forensics, starting from the most classic studies on the Electrical Network Frequency (ENF) for validating audio signals, concerning the extraction, enhancing [34] and analysis of the small frequency variations in the noise generated by the electrical current, and the consequent comparison with timestamps in the provider's database in order to guarantee the authenticity of the audio signal. Audio signals can be modified by means of tampering techniques such as splicing, deletion or replacement, and the detection of such modifications is a pursued goal in many research studies.

To avoid the detection of the aforementioned artifacts, some kind of post-processing techniques, like artificial reverberation or noise application, might be applied after the tampering, in order to mask it. Luo, in [35] developed a method to detect post-processing application by means of Audio Cooccurrence Vector (ACV), a feature based on the Gray-Level Cooccurrence Matrix (GLCM), used for image processing, that is the joint probability distribution of two pixels' cooccurrence values at a given offset, nemely, how often pairs of pixels with a specific value occur in the image at each offset. The offset is a position operator denoted as $(\Delta x, \Delta y)$ that can be applied to any pixel in the image, e.g., $(3, 5)$ means "three down, five right".However, the cooccurrence matrix, the offset can be parametrized also in terms of a distance, d, and an angle, θ, instead of the offset $(\Delta x, \Delta y)$. An image with p different pixel values will produce a $p \times p$ co-occurrence matrix, for each given offset.

Yan, in [36], did a similar thing, indeed, his algorithm had the goal to detect audio post-processing, but in this case, it was done by tracking the pitch, and measuring the similarity by means of the Pearson Correlation Coefficient (PCC) and the average difference. Zhang in [37] reached the same goal using a Support Vector Machine (SVM) classifier using Mel-Frequency Cepstrum Coefficientss (MFCCs) and Linear Prediction Coding (LPC) as features, while, in [38], Korycki described how to detect tampering exploiting the traces leaved by lossy compression algorithms. Speech resampling detection has been achieved in [39] by exploiting the Nyquist-Shannon fundamental theorem of sampling, and so the inconsistency between the bandwidth energy and the sampling rate. However, this method works well with resampling factors higher than one (oversampling), but it performs worse with undersampling. The detection of the application of the sustain piano pedal has been accomplished in [40], recurring to a CNN, with a dataset composed by the same tracks *with* and *without* the piano pedal, using MIDI messages as the ground-truth information.

In [41], Canclini et al. described a method to detect if an audio signal has been recorded indoor or outdoor, either by thresholding the RT60 parameter (outdoor environments are supposed to have a smaller reverberation time than indoor ones), or by using a SVM with MFCCs and Log-Mel frequency Spectral Coefficientss (LMSCs) as input features. Moreover, they tried to estimate the reverberating signal, and then to subtract it from the original one, in order to test in a stronger way the algorithm, as an anti-forensics technique.

Other anti-forensics techniques have been described, as in [42], for audio source identification purposes, and in [43, 44], for image forensics. In all of these three cases, the tool used to achieve the desired result was a Generative Adversarial Network (GAN) [45], a system composed by two NNs, in which one, the generator, has to produce real and fake data, which the other, the discriminator, has to recognize. The training is achieved so that the generator learns how to *fool* the generator, and vice versa.

1.5 Conclusive Remarks

In this chapter we defined the basic concepts that we will need in the next chapters. In particular, we illustrated the theory behind Time-Frequency audio representation, overviewing the Fourier Transform (FT), the Mel scale, and some different kinds of spectrograms (e.g. the Log-Mel Spectrogram (LMS)).

Moreover, we summarized the salient features about Deep Learning that we exploited during our project, such as Convolutional Neural Networks (CNNs) and basic Deep Learning tasks (e.g. Classification and Regression).

Finally, we depicted the State of the Art related to audio forensics methodologies, trying to correlate them with our project.

2

Problem Statement and Formulation

In this chapter we will describe the goals of this project, with all the different facets. The main target is to blindly identify the application of some sort of time-scale modification on an audio track, hence, without any prior information about it. The purpose is to investigate the signal and try to find-out eventual traces left by the algorithm itself.

In order to state the quality of our system we need to define an appropriate cost function. It is important to choose a suitable one that sufficiently represents our problem, in order to be able to optimize our results at their best.

2.1 Problem Formulation

Formally, let us consider an audio recording $x(n)$, $n = 0, 1, \cdots, N$. Given a generic audio track, our goal is to detect whether it is the original version or a time-scaled one, namely to produce a discrete output $y \in [0, 1]$, where 0 means "original", and 1 means "time-scaled". If this is the case, we also aim at estimating the time-scaling factor α, i.e. an approximation $\hat{\alpha} \in \mathbb{R}^+$. From this point, the identification of time-scaling can be split into two specific purposes: *detection* and *estimation*.

2.2 Time-scaling Detection

The time-scaling detection is essentially a binary classification, namely assigning the audio track a discrete label such as “time-scaled” (i.e. positive, *affected* by time-scaling) or “original” (i.e. negative), so the system should tell if the signal is the original one or if it has been altered in its time evolution.

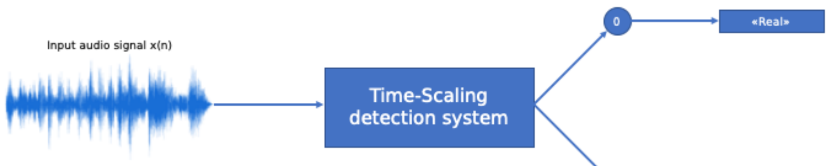


Figure 2.1: Formulation of time-scaling classification.

2.3 Time-scaling Estimation

In time-scaling estimation, instead, we are talking about linear regression, then the goal is to produce a number in the continuous domain, as an hypothesis about the value of the eventual time-scaling factor α (1.24), representing the rate between the duration of the original signal d_{or} and the duration of the time-scaled one d_{ts}:

$$\alpha = \frac{d_{\text{or}}}{d_{\text{ts}}}. \tag{2.1}$$

Obviously, in the time-scaling estimation scope, if no time-scaling has been applied, the returned value should be 1. In this case, being the output a number in the continous domain instead of a discrete class, the error should be computed in a different way. Some standard cost functions for regression tasks are, for example, the Absolute Error (AE), referred to as:

$$\text{AE} = |\alpha - \hat{\alpha}|\,, \tag{2.2}$$

or the Square Error (SE), expressed by:

$$\text{SE} = \left|(\alpha - \hat{\alpha})^2\right|, \tag{2.3}$$

where α_i is the true time-scaling factor of the signal, and $\hat{\alpha}$ the resulting value from the system. To reduce the computational effort of the training process, it is often used to split the dataset, that could be very large, into smaller portions, called batches. For the reasons we just mentioned, we need to resort to cost functions at batch-level, denoted as *batch losses*, such as the Mean Absolute Error (MAE), and the Mean Square Error (MSE), obtained by computing the mean of respectively the AE and the SE between all the samples of the batch.

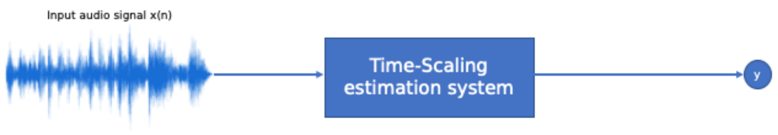


Figure 2.2: Formulation of time-scaling estimation.

2.4 Conclusive Remarks

In this chapter we described the main goals we wanted to achieve in this thesis. In addition, a mathematical formulation of the problem and of the results to be obtained has been provided.

3 Preliminary Analysis on Phase Re-sampling Traces

Before digging into the proposed method for audio time-scaling detection, in this chapter we present some preliminary studies we have conducted on the phase of time-scaled audio signals. In particular, we show that it is reasonable to expect re-sampling traces on the phase of time-scaled audio signals. This motivates the use of the audio phase as additional input to our detector.

3.1 Hints about Phase Re-sampling Traces

In Chapter 1 we have described the working principles of time-scaling algorithms in the Short-Time Fourier Transform (STFT) domain. We showed that those algorithms have to perform some sort of re-sampling operations on the STFT during the time-scaling synthesis stage. This is necessary due to the mapping between the original and time-scaled temporal axis.

In the light of this, STFT magnitude and phase re-sampling traces could be exploited to detect time-scaling operations. However, special care must be used if phase is considered due to the way phase gets processed and interpolated by time-scaling algorithms.

While STFT magnitude interpolation is a trivial task, interpolating the phase is not as straightforward. Indeed, a correct phase interpolation must ensure phase continuation depending on the instantaneous frequency of each tone present in the audio track. Moreover, due to its 2π periodicity, phase directly computed from an STFT is wrapped (i.e. it lies in the interval $[-\pi, \pi]$), whereas phase continuation is performed in an unwrapped domain (i.e. not considering the 2π periodicity). All of these considerations must be taken into account in order to possibly expose re-sampling traces on the phase. As an example, a wrapped phase signal shows so many discontinuities that would make re-sampling detection a challenging task. It is therefore necessary to understand in which domain it is better to perform phase analysis to expose re-sampling, thus time-scaling.

3.2 Empirical Phase Re-sampling Detection Analysis

In order to understand which is a good domain for phase re-sampling detection, we decided to conduct some experiments with the image re-sampling detector proposed by Kirchner in [1]. Indeed, the STFT phase can be considered a 2D image with no loss of generality.

The considered detector [1] is a powerful forensic tool that exploits the frequency representation of a signal in order to possibly expose re-sampling traces. In a nutshell, the method is based on the computation of the so-called p-map, that is a high-pass version of the signal under analysis. Kirchner [1], Popescu and Farid [46] showed that the polished and filtered Fourier Transform (FT) of the p-map exhibits strong prominent peaks if some sort of re-sampling has been applied to the signal under analysis (Fig. 3.1).

In the following, we show the output of the re-sampling detector in a series of controlled tests with the goal of understanding whether it is possible to detect re-sampling on the STFT phase of time-scaled audio signals in a specific domain.

3.2.1 Controlled Experiments: Single Tone

In this experiment, we consider the analysis of an audio signal composed by a single tone (i.e. a single sinusoid at a given frequency), which is time-scaled using different scaling factors. As we have full control over the experiment, we can predict which is the theoretically expected phase

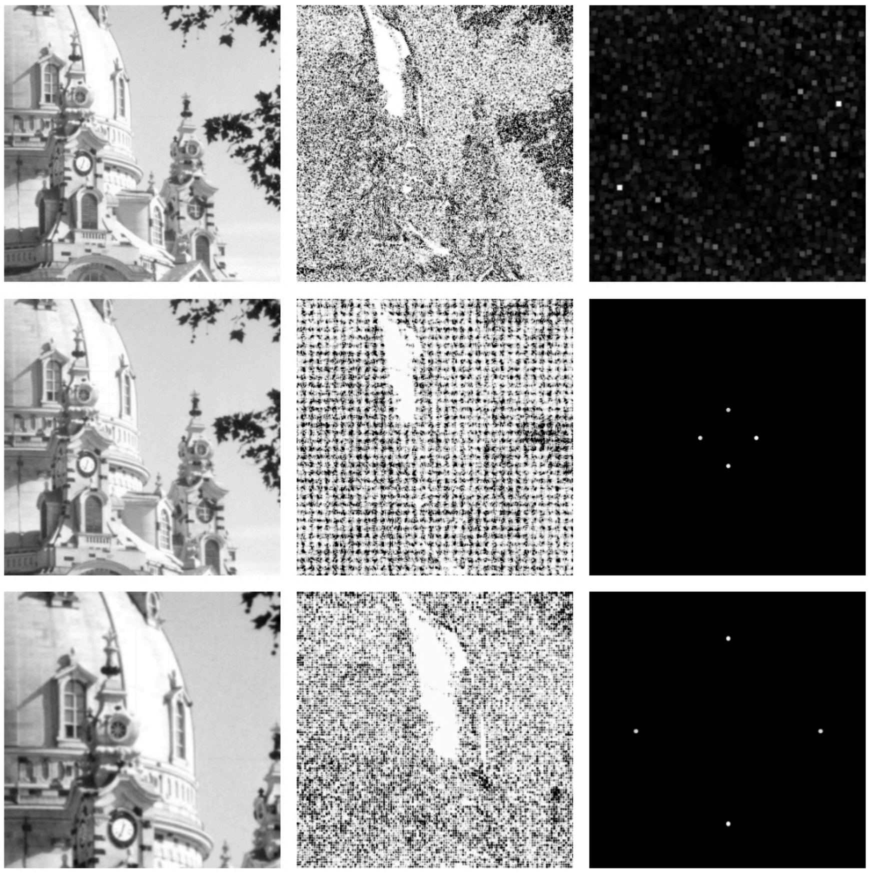
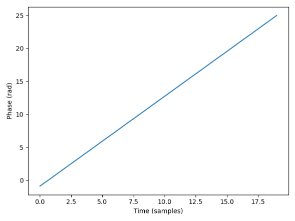


Figure 3.1: Results of Kirchner's re-sampling detection for original image (top row), 111% up-sampling (middle row) and 150% (bottom row) [1]. Left column: images; center column: p-maps; right column: p-map spectra.

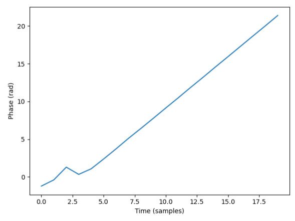
of the time-scaled signal. We therefore compare the obtained unwrapped phase after time-scaling with the theoretical one.

Considering a pure tone at 440 Hz, Fig. 3.2 shows the theoretical phase of its time-scaled version with factor 0.1 (a), the phase computed from the actual time-scaled version (b), and the difference between these phases (c). Fig. 3.3 and 3.4 report the result of the same experiment for two additional time-scaling factors (i.e. 0.3 and 1.5, respectively).

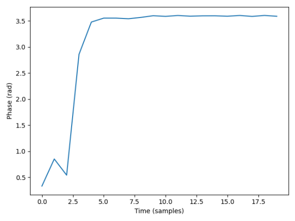
By analyzing these figures, it is possible to notice that the phase difference oscillates around a constant value (after an initial overshooting due to the way time-scaling is implemented on the first time windows of a signal). These oscillations must come from the computed phase, as the theoretical one is linear. This suggested us to inspect the time-scaled phase and also, if available, the difference between the expected phase and the resulting one from the time-scaling algorithm, in order to find an eventual correlation between its periodicity and the time-scaling modification factor.



(a) Expected phase of 440 Hz sinusoid's STFT for time-scaling factor = 0.1

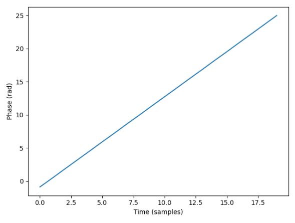


(b) Time-scaled phase of 440 Hz sinusoid's STFT for time-scaling factor = 0.1

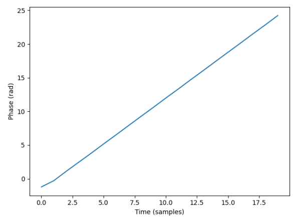


(c) Difference between expected and resulting phase, at 440 Hz, time-scaling factor = 0.1

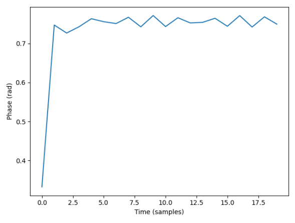
Figure 3.2: Phase error analysis for a 440 Hz time-scaled sinusoid (time-scaling factor = 0.1).



(a) Expected phase of 440 Hz sinusoid's STFT for time-scaling factor = 0.3

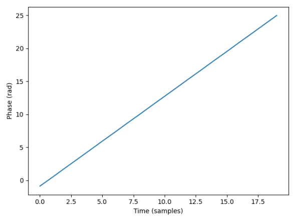


(b) Time-scaled phase of 440 Hz sinusoid's STFT for time-scaling factor = 0.3

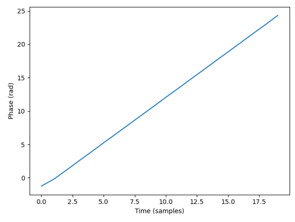


(c) Difference between expected and resulting phase, at 440 Hz, time-scaling factor = 0.3

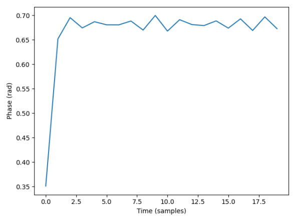
Figure 3.3: Phase error analysis for a 440 Hz time-scaled sinusoid (time-scaling factor = 0.3).



(a) Expected phase of 440 Hz sinusoid's STFT for time-scaling factor = 1.5



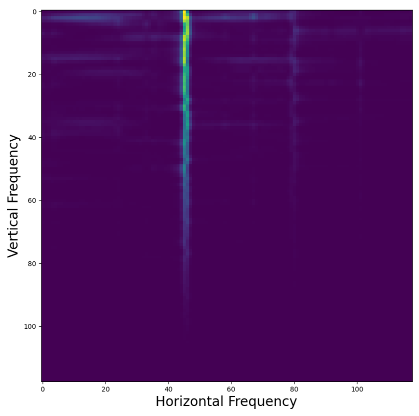
(b) Time-scaled phase of 440 Hz sinusoid's STFT for time-scaling factor = 1.5



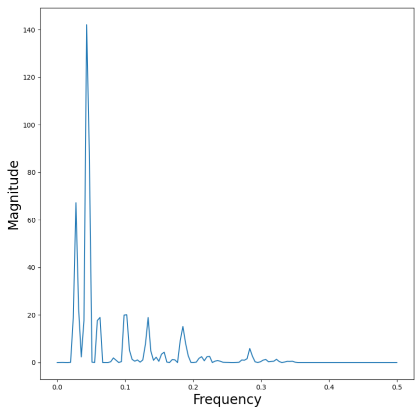
(c) Difference between expected and resulting phase, at 440 Hz, time-scaling factor = 1.5

Figure 3.4: Phase error analysis for a 440 Hz time-scaled sinusoid (time-scaling factor = 1.5).

Fig. 3.5 reports the results of the re-sampling detector applied to the unwrapped phase of the STFT of a time-scaled sinusoid with fundamental frequency of 440Hz and time-scaling factor of 0.1. Specifically, Fig. 3.5a shows the FT of the p-map, while Fig. 3.5b reports a linear version of

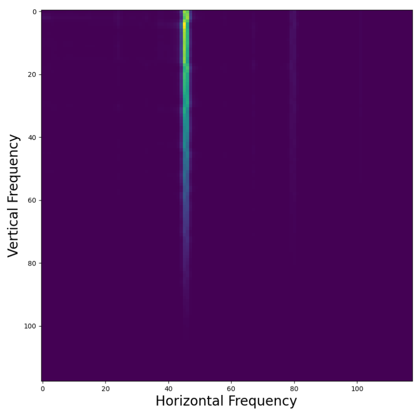


(a) The p-map FT of the STFT phase for a 440 Hz sinusoid with time-scaling factor = 0.1

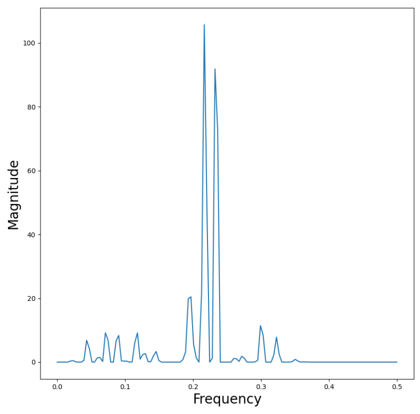


(b) The p-map FT (linear version) of the STFT phase for a 440 Hz sinusoid with time-scaling factor = 0.1

Figure 3.5: Detector results on the unwrapped phase of the STFT for a 440 Hz time-scaled sinusoid (time-scaling factor = 0.1).



(a) The p-map FT of the STFT phase for a 440 Hz sinusoid with time-scaling factor = 0.3

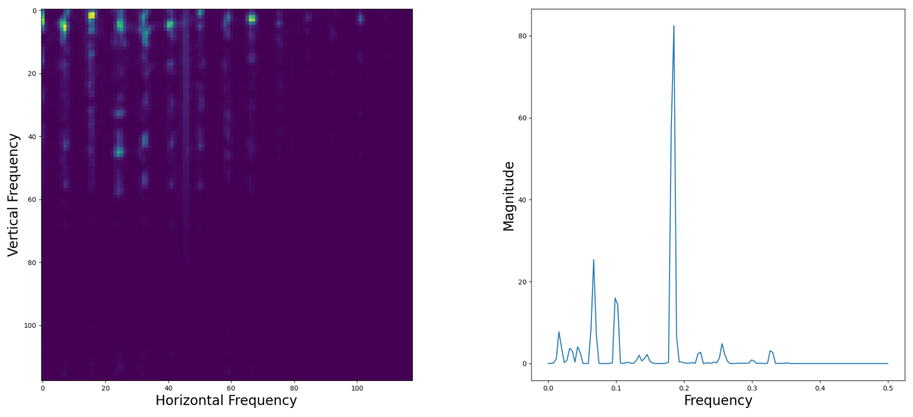


(b) The p-map FT (linear version) of the STFT phase for a 440 Hz sinusoid with time-scaling factor = 0.3

Figure 3.6: Detector results on the unwrapped phase of the STFT for a 440 Hz time-scaled sinusoid (time-scaling factor = 0.3).

the same signal, obtained by integrating the p-map spectrum over the angles in polar coordinates. In addition, in Fig. 3.6 and Fig. 3.7 we can see the same detector results on the unwrapped phases of two time-scaled sinusoids with a fundamental frequency of 440Hz and time-scaling factors of 0.3 and 1.5 respectively.

The results just mentioned exhibit distinguishable periodicities in the



(a) The p-map FT of the STFT phase for a 440 Hz sinusoid with time-scaling factor = 1.5

(b) The p-map FT (linear version) of the STFT phase for a 440 Hz sinusoid with time-scaling factor = 1.5

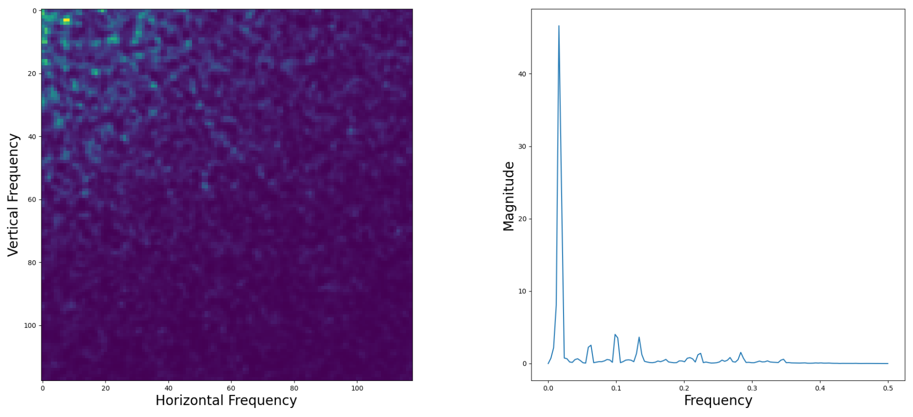
Figure 3.7: Detector results on the unwrapped phase of the STFT for a 440 Hz time-scaled sinusoid (time-scaling factor = 1.5).

FT of the p-map, as we can clearly notice in Fig. 3.5a, Fig. 3.6a and Fig. 3.7a, and manifest peaks in their respective linear versions (Fig. 3.5b, Fig. 3.6b and Fig. 3.7b).

3.2.2 Experiment in the wild: complex audio track

Moving to a more generic and uncontrolled situation, the experiment consists of the analysis of a complex audio track representing a music recording of multiple instruments together. This track is time-scaled using the same scaling factors of the previous scenario (i.e. 0.1, 0.3 and 1.5). In this case, since we are not able to exactly produce the actual expected audio track with a different time evolution, we cannot compare the obtained unwrapped phase after time-scaling with the theoretical one.

In Fig. 3.8 we can analyze the results of the detector on the unwrapped phase of the STFT of a time-scaled complex track with a scaling factor of 0.3. In this case, the intrinsic complexity of the track itself led to the lack of clear periodicities in the FT of the p-map (Fig. 3.8a), and to weaker peaks in the linear version (Fig. 3.8b).



(a) The p-map FT of the STFT phase for a complex track with time-scaling factor = 0.3

(b) The p-map FT (linear version) of the STFT phase for a complex track with time-scaling factor = 0.3

Figure 3.8: Detector results on the unwrapped phase of the STFT for a complex time-scaled track (time-scaling factor = 0.3).

3.3 Conclusive Remarks

The preliminary analysis described before on the phases of the STFT of time-scaled audio signals gave us some qualitative hints about the correlation between the application of some time-scaling modification algorithm and the presence of some recurrent pattern on the phase. As we have seen, also the Kirchner's detector shows some prominent peak in the case of strictly controlled test cases (e.g. synthetically-generated sinusoids). This is not valid anymore with more complex tracks, where the intrinsic complexity of the signal masks the eventual traces left by the time-scaling algorithm. Real test cases of complex audio tracks also make practically impossible to pursue the approach of comparing the resulting time-scaled signal with the expected one of the same duration, since in this situation it would be very hard to produce a faithful estimation of the expected phase.

Finally, despite a simple re-sampling detector turned out to be not suitable to provide quantitatively reliable results about time-scaling detection and/or estimation, it has been a very good starting point to find a correlation between time-scaling and phase artifacts. As a matter of fact, this detector inability to fully expose phase re-sampling motivated us in investigating data-driven approaches that we will described in the following chapters.

4
Proposed Methodology

In this chapter we will present the strategy we have used to achieve the goals that we described up to this point. The adopted methods are based on *EfficientNet*, a Convolutional Neural Network (CNN) developed for image classification purposes [2]. Since we identified two different tasks, we needed to define two different CNNs, with the fundamental components in common, but with some slightly different architecture elements, according to the task to be completed.

A summary representation of the proposed system is shown in Fig. 4.1. We will analyze in details the architecture of such system in the following sections.

4.1 Audio Pre-Processing

As we said before, the implemented CNN is based on a model that has been developed to work with color images. Consequently, we need to represent the signal as a 3D matrix with three (color) channels. We moved from audio to three-channel images representations by producing different kinds of 64×96 matrices from the signal, having so 64 frequency bins to represent the spectral content, and 96 frames to describe temporal evolution. These channel matrices are combined then in groups of three with different arrangements.

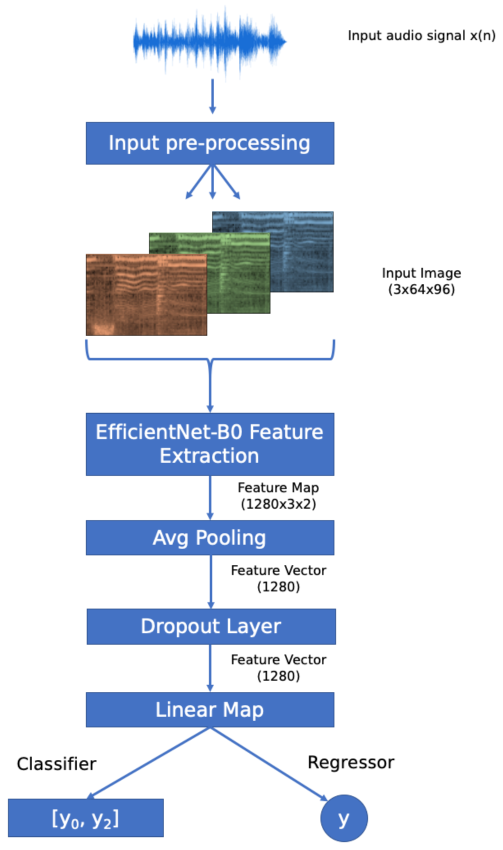


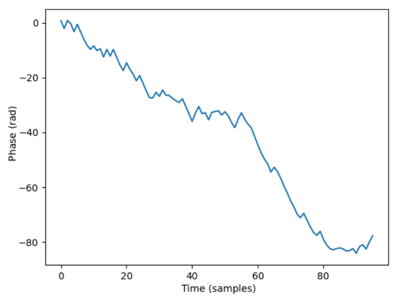
Figure 4.1: A summary representation of the proposed time-scaling detection system

The channel 2D representations of the signal that we have decided to use are:

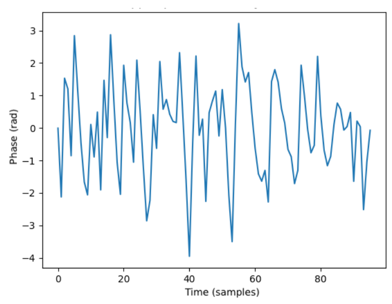
- Log-Mel Spectrogram (LMS): This combined time-frequency visual description of the audio signals has been computed according to the Mel-scale conversion by means of the HTK formula (1.18) [14];
- High-Pass Filtered unwrapped Phase (HPF-Ph): This representation is achieved by high-pass filtering the unwrapped phase $\varphi(m, k)$ of the Short-Time Fourier Transform (STFT) (1.16);
- Median Residual unwrapped Phase (MedResPh): This maytrix is computed by subtracting the median-filtered version of $\varphi(m, k)$ from the original one.

The reason why we adopted the two aforementioned filtering-based approaches was that, initially, the unwrapped phases of time-scaled STFTs $\varphi(m, k)$ exhibited for each frequency bin k a time evolution such as small high-frequency variations over an important low-frequency signal

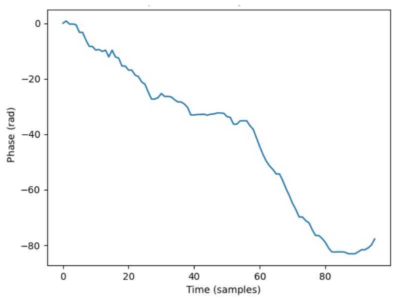
(Fig. 4.2). We wanted to filter-out this low-frequency information to enhance the high-frequency variations. To do so, we adopted the two strategies described above: the first one, the HP-filtered $\varphi(m, k)$, has been reached by means of a 2$^{\text{nd}}$-order Butterworth filter [47] with a cut frequency set at 1kHz. The second method we used to keep the “noisy” variations is to subtract the median-filtered version of $\varphi(m, k)$ from the original one, since the median filter is a filtering technique used to remove noise by simply replacing each sample with the median value of the neighbors [48].



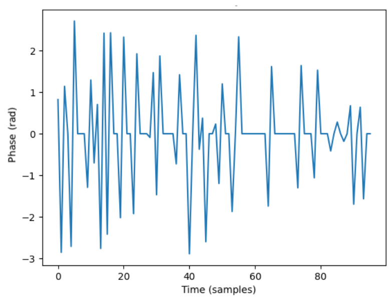
(a) Unwrapped phase for a single bin.



(b) HP-filt. phase for a single bin.



(c) Median filt. phase for a single bin



(d) Median residual for a single bin

Figure 4.2: The filtering process of the unwrapped phase of the STFT

All of these representations are based on the computation of the STFT of the input audio signal, that, in order to guarantee consistency, we have implemented in all the cases with the same parameters:

- F_s: 16kHz;
- Window type: Hanning;
- Window length: 25 ms (400 samples);
- Hop size: 10 ms (160);
- Fast Fourier Transform (FFT) length: 512 samples;

Table 4.1: A summary of the baseline network of the EfficientNet [4].

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	# Channels $\hat{C}_i$	# Layers $\hat{L}_i$
1	Conv 3×3	224×224	32	1
2	MBConv1, k3×3	112×112	16	1
3	MBConv6, k3×3	112×112	24	2
4	MBConv6, k5×5	56×56	40	2
5	MBConv6, k3×3	28×28	80	3
6	MBConv6, k5×5	14×14	112	3
7	MBConv6, k5×5	7×7	192	4
8	MBConv6, k3×3	7×7	320	1
9	Conv 1×1 & Pooling & FC	7×7	1280	1

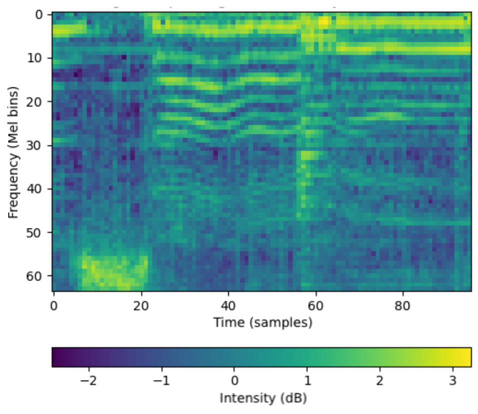
- f_l (for LMS): 125Hz;
- f_h (for LMS): 7500Hz.

First, since we wanted to be coherent with the 64×96 shape of the channel matrices, we needed to reduce the dimensionality of the STFT from the resulting one to the standard size. In order to do so, we cut the matrices on the time axis, and kept only the frequency bins associated to the center frequencies of the HTK Mel filterbank.

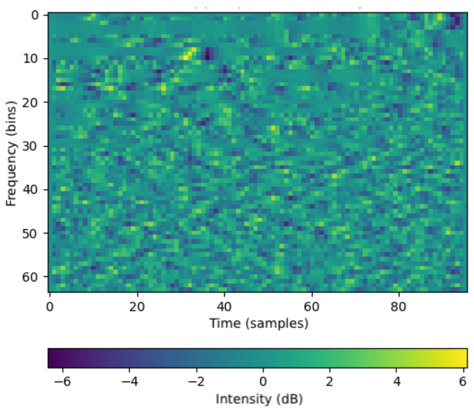
As we said before, our CNN requires 3-channel images as input, therefore, we had to combine the channel matrices that we just described in groups of three, and we decided to arrange them either by replicating the same matrix three times, or by combining the three of them together, as summarized in Fig. 4.3.

4.2 Backbone Architecture

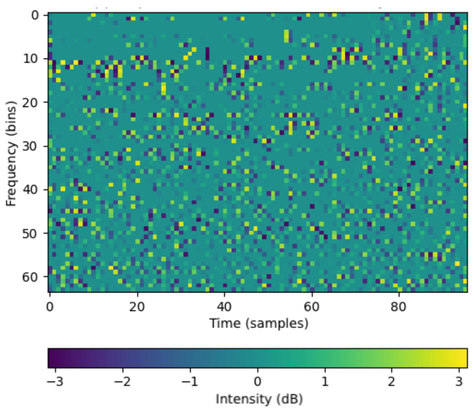
Our CNN is based on EfficientNet, a model described by Tan et al. in [2], whose main peculiarity is to uniformly scale its dimensions (depth, width and resolution) by means of a compound scaling factor. This scaling process starts from the baseline architecture, called EfficientNet-B0 [4]. As depicted in Table 4.1, the EfficientNet-B0 is built on the Mobile inverted Bottleneck Convolution (MBConv), also called Inverted Residual Bottleneck, a layer that was first introduced in [4] as fundamental component of the presented MobileNetV2 Neural Network. MBConv has been introduced on the convinction that:



(a) Log-Mel Spectrogram



(b) HP-filtered phase.



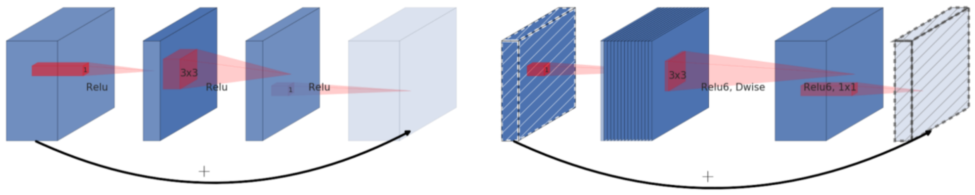
(c) Phase – median filter

Figure 4.3: The three possible channel matrices of the CNN.

- Feature Maps (FMs) can be encoded in lower-dimensionality subspaces;

- Non-linearities in the narrow layers of residual structures may result in loss of information, without any increase of the ability in terms of representational complexity.

The MBConv is based on an inverted residual structure (Fig. 4.4(b)), where shortcuts are introduced between thinner layers, in order to compensate possible degradation problems [49]. As it is described in Table 4.2, a MBConv$t$ block, takes as input a tensor made of k channels, and $(h \times w)$ resolution. It is composed by three convolution sublayers:



(a) The residual bottleneck block. (b) The inverted residual bottleneck.

Figure 4.4: Residual bottleneck block (a) vs. MBConv (b)[2].

Table 4.2: A summary of the MBConv block[4].

Input	Operator	Output
$h \times w \times k$	1×1 conv 2d, ReLU6	$h \times w \times (tk)$
$h \times w \times (tk)$	3×3 dwise $s = s$, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times (tk)$	linear 1×1 conv 2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

1. A point-wise (1×1) convolution, in order to expand the low-dimension input FM to a t-times deeper one, i.e. with shape $(h \times w \times tk)$, followed by a ReLU6 activation function, where the ReLU6 is defined as $y = \min(\max(0, x), 6)$ (Fig. 4.5);

2. A depth-wise convolution by means of a (3×3) kernel filter with stride s, also followed by the ReLU6 activation function, in order to obtain spatial filtering, resulting into a FM with shape $\left(\frac{h}{s} \times \frac{w}{s} \times tk\right)$;

3. Finally, the spatially-filtered FM is projected back to a low-dimension subspace with a point-wise convolution, followed this time by a linear activation function.

As we can see in Table. 4.1, the MBConvs in EfficientNetB0 are defined at each stage i with the expansion factor t, the dimensions of the kernel filter at the intermediate sub-layer, the input Resolution $\left(\hat{H}_i \times \hat{W}_i\right)$, the

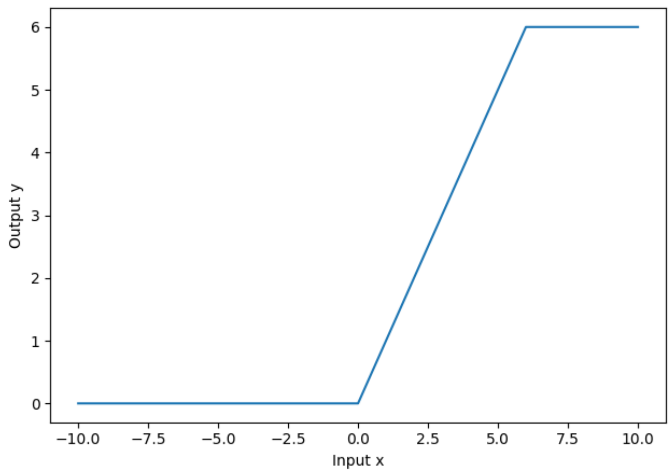


Figure 4.5: The ReLU6 activation function

output depth $\hat{C}_i$, and the number of layers $\hat{L}_i$, meaning that the same MBConv indicated at stage i is repeated $\hat{L}_i$ times, having $\hat{C}_i$ output channels at each repetition.

The EfficientNet-B0 is the first fundamental block of our CNN that we have chosen to extract a feature vector of shape $(1280, 2, 3)$ from the input tensor of size $(3, 64, 96)$. The feature extraction block (Fig. 4.6) is followed by an average-pooling layer, that further reduces the dimensions to a 1D vector of 1280 scalar features. Then, the feature vector is fed to a dropout layer, whose main goal is to set to zero some random feature, according to a Bernoulli probability distribution [50], in order to avoid co-adaptation and overfitting issues [51]. Finally, a linear transformation is applied to the feature vector, such that:

$$y = x \cdot W + b, \tag{4.1}$$

where y is the output of the CNN, with shape (n_{out}), x is the feature vector of 1280 elements, W is the trainable weight matrix, whose shape is $(1280, n_{\text{out}})$ and b is a trainable bias parameter. The parameter n_{out} is the number of output classes, and is the parameter that, as well as the loss function, differentiates the classification CNN from the estimation one.

4.2.1 Classifier

In the classification task, $n_{\text{out}} = 2$, hence the output is a vector of two numbers $[y_0, y_1]$, such that:

- If $y_1 > y_0$, the CNN has detected the precence of time-scaling;

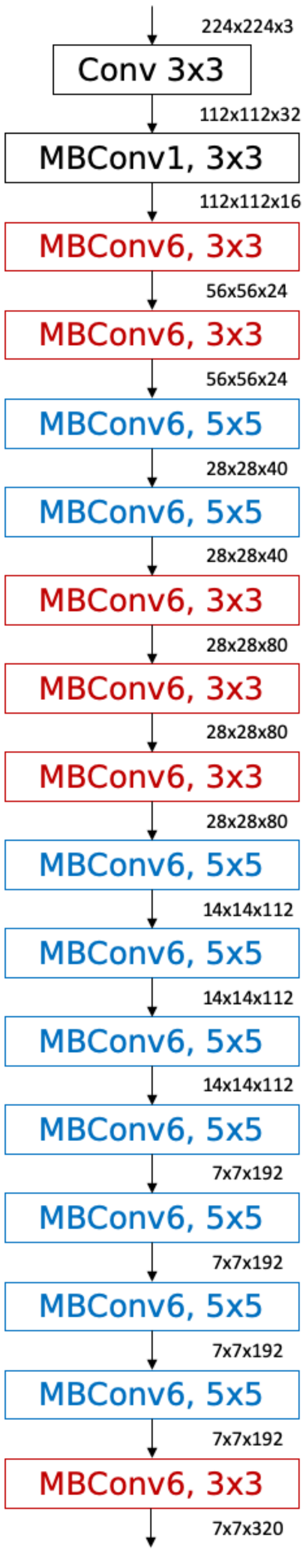


Figure 4.6: The extended representation of the EfficientNet-B0 [3].

- If $y_0 > y_1$, the output prediction is “original”.

After the linear transformation, we find a nonlinear layer, whose goal is to compute the *softmax* function. The softmax function, or normalized exponential function, is a common choice as a last activation function of a CNN, with the aim to normalize the output of the network to a probability distribution over predicted output classes. The softmax function ς takes the output vector of the previous layer (i.e. the linear transformation) as input, that is a vector x of $C = n_{\text{out}}$ real numbers, and normalizes it into a probability distribution y, made by C probability values proportional to the exponentials of the input elements. The softmax function is defined as follows [27]:

$$y_c = \varsigma(x_c) = \frac{e^{x_c}}{\sum_{j=1}^{C} e^{x_j}}, \tag{4.2}$$

meaning that the softmax applies the exponential function to each element x_c of the input vector x and normalize them by computing the ratio between the sum of all these exponentials; this normalization ensures the sum of the components of the y to be 1.

In this case, as a loss measure, we chose the Cross-Entropy Loss (CEL), whose specific implementation will be defined in the following chapter.

4.2.2 Regressor

In the case of time-scaling estimation, instead, $n_{\text{out}} = 1$, so the output is simply a number that is the prediction of the CNN about the time-scaling factor α. For the regressor, the error function we chose to optimize the prediction results is the Mean Square Error (MSE) (2.3).

4.3 Conclusive Remarks

In this chapter we presented the methodology that we developed to solve the central issues of this work. Specifically, we defined the fundamental architecture of the employed CNN, pointing out the distinct features and building blocks, according to the particular task to achieve.

5 Experiments and Results

In this chapter we present the practical implementation choices of the the methodology we described before. After that, all the obtained results, referring to each test that we conducted, will be analyzed in details.

5.1 Dataset

In this section we will define how the dataset used for the test experiments has been built.

5.1.1 Generation

First, we created a dataset of the original audio tracks that we used to get only the original (not time-scaled) samples. We select audio tracks from two state-of-the-art datasets:

- GTZAN genre recognition dataset: the first dataset we took data from is the one created by George Tzanetakis for the research described in [52]. This dataset is composed by 1000 mono tracks, each of which 30 seconds long, sampled at 22050Hz, 16 bit depth. The dataset is equally divided into 10 genres (blues, classical, pop, country, rock, disco, hip-hop, metal, reggae, jazz).

- IRMAS Testing Dataset: it is the testing subset of the dataset compiled by Ferdinand Fuhrmann in 2012, for Instrument Recognition in Musical Audio Signals [53]. It is composed by 2874 excerpts with a length between 5 and 20 seconds, sampled at 44.1 kHz, 16 bit.

The time-scaling modified tracks have been generated using three different Phase Vocoder Time-Scaling Modification (PV-TSM) algorithms:

- Audiotsm [5];
- Torchaudio TimeStretch [6];
- Time-Stretch-Master [7].

Several time-scaled tracks have been generated using the three aforementioned algorithms on the original present tracks. The amount of time-scaled tracks for each original one, i.e. the balancing principles of the datasets, were chosen in a different way for classification and regression tasks. Regarding each task, the time-scaled tracks are generated in a mirrored way (i.e. with the same randomly-chosen time-scaling factor for the same original track and the three algorithms), so that, both for classification and regression, we end up into three instance of the same dataset, with all the same parameters except for the time-scaling algorithm. Specifically, the datasets are balanced in the following way:

- Classification: the balancing is achieved when we have the same number of original and time-scaling tracks;
- Regression: in this case, we generated an amount of time-scaled tracks such that we had approximately the same number of files for each time-scaling factor from 0.1 to 2.0, using a step of 0.2 for training and validation subsets, while using a step of 0.1 for the test subset, including in all the three cases the files with $\alpha = 1$, i.e. the original files.

After the audio datasets have been built, for each audio track we have computed the Log-Mel Spectrogram (LMS), the High-Pass Filtered unwrapped Phase (HPF-Ph) and the Median Residual unwrapped Phase (MedResPh), i.e. the three kinds of input to the proposed Convolutional Neural Network (CNN). Let us recall that the shape of the input temporal windows of the CNN must be (64×96), but the computation of the aforementioned windows leads to more than 96 time frames. We decided to divide the resulting spectrograms into several (64×96) temporal windows and to keep only from the 3rd to the 6th following the temporal evolution order, having then a dataset consisting of four temporal windows for each audio track.

5.1.2 Dataset split

Audio files contained in our dataset have been grouped into three subsets for training, validation and test stages, according to the following proportions:

- Training: 68%;
- Validation: 17%;
- Testing: 15%.

To summarize, the final dataset “seen” by the CNN is composed by two versions (i.e. for time-scaling detection and estimation), each of which is composed by three mirrored instances (one for each time-scaling algorithm) divided into three balanced subsets for training, validation and testing stages. Hence, the final datasets are divided in this way:

- Classification:
  - Training: 20727 samples;
  - Validation: 5184 samples;
  - Test: 4568 samples
- Regression:
  - Training: 52140 samples;
  - Validation: 7850 samples;
  - Test: 6910 samples.

5.2 Training Setup

As we described in the previous chapters, we need to define a cost function to minimize in order to get the best results by adapting the trainable parameters of the CNN. For the regressor we chose the Mean Square Error (MSE) (2.3), while for the classifier we adopted the Cross-Entropy Loss (CEL), in this case, with the implementation provided by Pytorch, combining the softmax nonlinear activation function (4.2), planned on the last layer of the CNN, and the Negative Log-Likelihood (NLL) loss function, defined as:

$$\mathrm{NLL}(y) = -\log y. \tag{5.1}$$

This implementation results into a loss function equivalent to the CEL, defined as:

$$\mathrm{CEL}_{\mathrm{pt}}(x_c) = -\log\left(\frac{e^{x_c}}{\sum_{j=1}^{C} e^{x_j}}\right). \tag{5.2}$$

Following a benchmark procedure in CNN training, we divide training and validation datasets in several smaller portions, called batches. In our case, we used a batch-size of 20 samples. The Optimization algorithm we chose for our system is the Adam optimizer, presented for the first time by Kingma and Ba in [30].

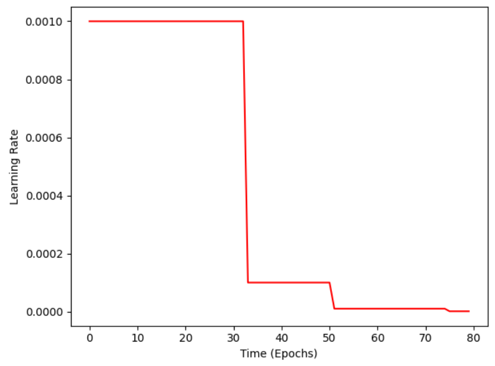
In the implementation of our system, we decided not to keep a constant value for the learning rate (LR), but to adapt it dynamically by automatically reducing the LR when the validation loss reaches the so-called *plateau*, i.e. a quasi-constant value. In simple words, when the validation loss does not exhibit any improvement after a certain number of epochs, the LR is reduced by a factor 10 in order to tune the trainable model parameters in a finer way. This allowed us to start with a relatively high LR value (i.e. 10^{-3}), making more important the earlier results, and to reduce it then after 10 epochs with no improvements on the related loss, in order to help achieving boosts to the learning process.

We decided to monitor the progress of the learning process, with the aid of the so-called callbacks, i.e. a set of functions that allowed us to keep track of the situation of training in real-time, and eventually to early-stop the process if no improvements are brought about the learning point of view. In particular, the training process can stop in two cases:

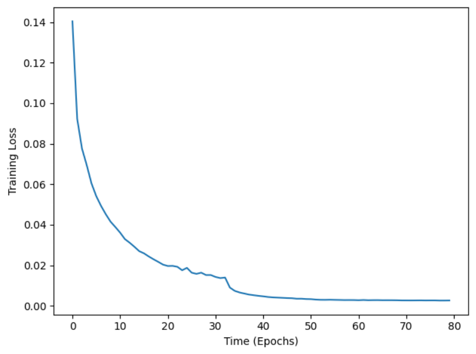
1. Early patience reaching: prior to learning process, a hyperparameter called “patience” is set to identify the number of epochs after which, if no improvement about the validation loss function is obtained, the training must be stopped to avoid any useless additional computation. In our case, we set it to 20 epochs;

2. Reaching of the maximum number of epochs: a threshold about the maximum amount of epochs is set, so that, after this number of epochs, the training is stopped in any case, without considering any information about the improvements of the learning curves. In our implementation, the maximum number of epochs was se to 80.

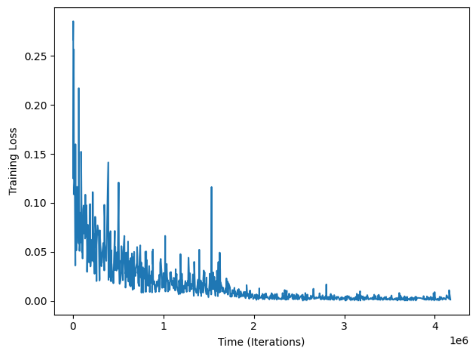
The training process has been conducted for time-scaling detection and estimation with the parameters we described above, then the now we will go in the details of how the results have been collected. In Fig. 5.1 we can see the monitoring logs refering to the training-validation process



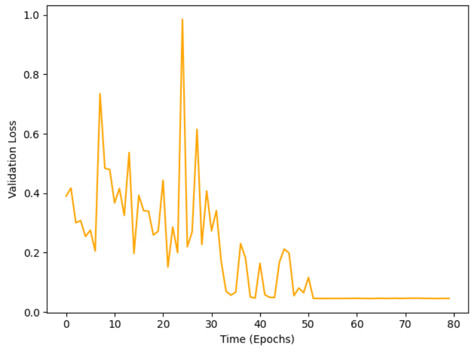
(a) The LR evolution.



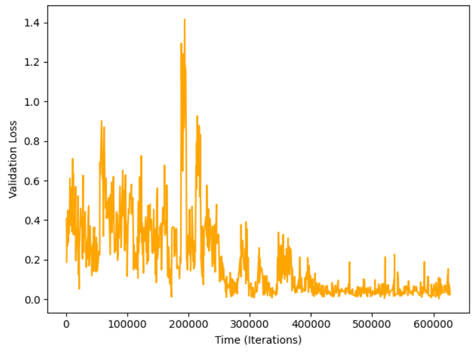
(b) The training epoch losss evolution.



(c) The training batch losss evolution.



(d) The validation epoch losss evolution.



(e) The validation batch losss evolution.

Figure 5.1: The logs of the training process for the regression task with LMS as input, Audiotsm time-scaling algorithm.

of the CNN with LMSs as input and the Audiotsm algorithm to obtain the time-scaling modified tracks.

Fig. 5.1a shows how the LR evolves with the passing of the epochs. We can see that the LR value decreases with a factor of 10 when the validation epoch loss does not reach any minima within 10 epochs. For instance, we can see that at the 20th epoch the validation loss reaches a local minimum, and then it keeps oscillating at higher values. Hence, around the 30th epoch, the LR value changes from 10^{-3} to 10^{-4}. This allowed to fine the the trainable parameters with a better precision, indeed we can see that the loss starts to decrease again. We can see that this behavior is repeated later on, even between the 70th and the 80th, indeed, since smaller improvements are recorded also in the latest epochs, no patience of 20 epochs is reached, then the training process is stopped at the 80th epoch.

In Fig. 5.1b we can see the training epoch loss evolution, while in Fig. 5.1c and Fig. 5.1e there are respectively the training and validation losses for each sample in the two subsets.

5.3 Classifier Results

We decided to measure the performance of the classification prediction by means of the Accuracy (ACC), defined as:

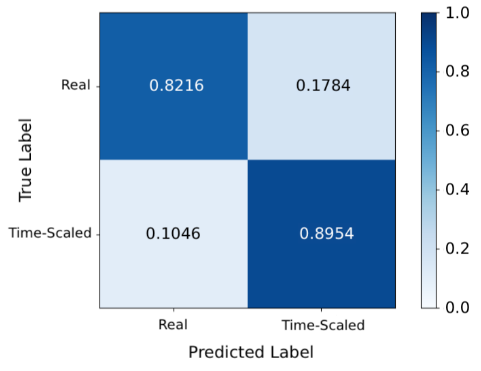
$$\text{ACC} = \frac{T_{\text{R}} + T_{\text{TS}}}{T_{\text{R}} + T_{\text{TS}} + F_{\text{R}} + F_{\text{TS}}} = \frac{T_{\text{R}} + T_{\text{TS}}}{\text{R} + \text{TS}}, \tag{5.3}$$

where:

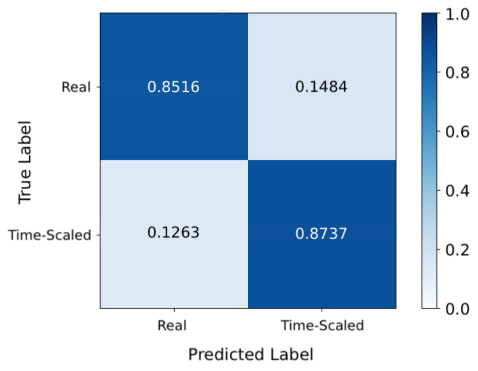
- T_{R} and T_{TS} are the number of true (correct) predictions of respectively original and time-scaled signals;
- F_{R} and F_{TS} the false (wrong) predictions;
- R and TS are the number of the overall original and time-scaled audio tracks in the dataset.

As we can see in Fig. 5.2, for detection (i.e. classification) task, we have obtained satisfactory results with LMS as input, either with the same dataset for both training and testing stages (Fig. 5.2a, ACC = 0.8585), and with cross-dataset tests (Fig. 5.2b, ACC = 0.8626), proving that the CNN has learned to recognize those typical features of original and time-scaled signal, despite of the specific implementation details.

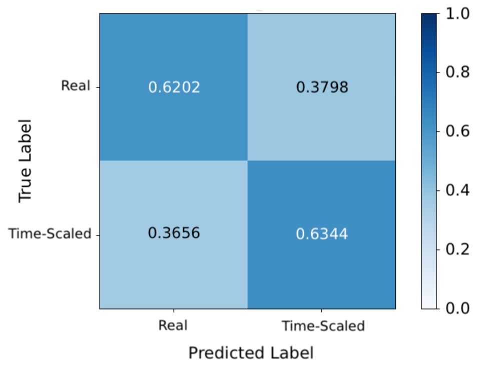
Nevertheless, how we can see in the other CMs, the system is not as accurate with other input combinations. For instance, in Fig. 5.2c, we can see the outcome of the CNN with the HPF-Phs as input and the time-scaled tracks obtained by means of the Audiotsm algorithm. The resulting accuracy is ACC = 0.6273, that is slightly better of a purely-random decision. If we consider instead the same input (HPF-Ph) but in a cross-dataset test, with the Torchaudio algorithm in training stage and the Time-stretch-master for test phase, results get an improvement: indeed the accuracy is ACC = 0.6760, because of the increased ability



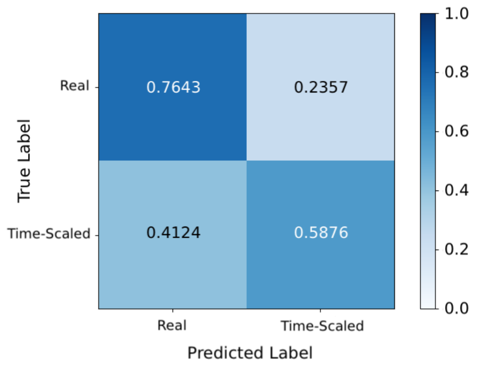
(a) The Confusion Matrix of classification test with LMS and Audiotsm algorithm



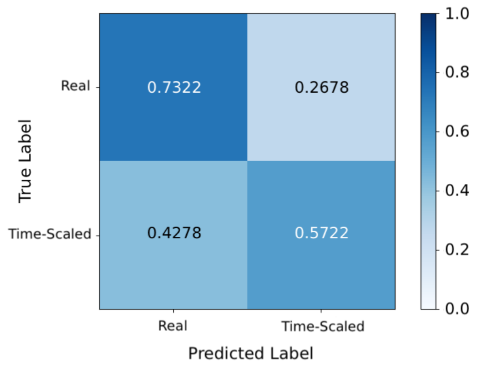
(b) The CM of the test with LMS and Torchaudio (training), tsm (test)



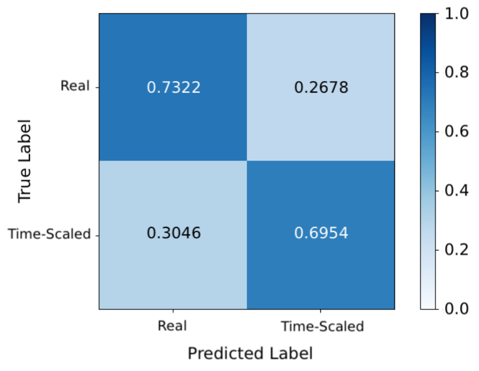
(c) The Confusion Matrix of classification test with HPF-Ph and Audiotsm algorithm.



(d) The Confusion Matrix of classification test with HPF-Ph, Torchaudio (training), and Time-stretch-master (testing).



(e) The Confusion Matrix of classification test with MedResPh and Time-stretch-master algorithm.



(f) The Confusion Matrix of the test with all the three inputs, Time-stretch-master (training), and Torchaudio (testing).

Figure 5.2: Some examples of Confusion Matrix (CM) for the classification task.

of recognizing the original tracks after the training with the Torchaudio dataset. In Fig. 5.2e, we can see again the good ability of recognizing the original tracks, in spite of a poor quality on the individuation of time-scaled samples, resulting into an accuracy of ACC = 0.6522. Finally, in Fig. 5.2f we can see the resulting CM from a cross-dataset test, with Time-stretch-master algorithm for training and Torchaudio for test, using this time the three kinds of input together, i.e. the LMS, the HPF-Ph and the MedResPh. With this input set-up, the result are slightly better than HPF-Ph and MedResPh, indeed, in this case the accuracy is ACC = 0.7138.

In Table 5.1, we can see all the accuracy results for classification tests conducted with the LMS as input. The results are pretty similar to each other, since they are located in a range between 0.8516 (Torchaudio for training and Audiotsm for test), and 0.8690 (Time-stretch-master for training and Torchaudio for test).

Table 5.1: Classifier Results with Log-Mel Spectrogram as input. In bold, the maximum and the minimum achieved accuracy values.

Training	Test Algorithm	Accuracy
Audiotsm	Audiotsm	0.8585
	Torchaudio	0.8569
	Time-stretch-master	0.8558
Torchaudio	Torchaudio	0.8631
	Audiotsm	**0.8516**
	Time-stretch-master	0.8526
Time-stretch-master	Time-stretch-master	0.8670
	Audiotsm	0.8542
	Torchaudio	**0.8690**

In Tables 5.2 and 5.3 there are the results relating to tests with HPF-Ph and MedResPh as input. The values here are decisely worse, being between 0.5979 (Table 5.3, Time-stretch-master for training and Audiotsm for test), and 0.6801 (Table 5.2, Time-stretch-master algorithm for both training and test stages).

To conclude, in Table 5.4 there are the results for tests with all the three kinds of input together, i.e. the LMS, the HPF-Ph and the MedResPh. The best outcome is achieved with the model trained using the Torchaudio algorithm (ACC = 0.7259), both in a cross-dataset combination, with Time-stretch-master algorithm for test, and with Torchaudio algorithm itself in the testing stage.

The worst results, instead, have been obtained with both the cross-dataset situations with the Audiotsm algorithm used for test.

Table 5.2: Classifier Results with HPF-Ph as input. In bold, the maximum and the minimum achieved accuracy values

Training	Test Algorithm	Accuracy
Audiotsm	Audiotsm	0.6273
	Torchaudio	0.6593
	Time-stretch-master	0.6642
Torchaudio	Torchaudio	0.6755
	Audiotsm	0.6274
	Time-stretch-master	0.6760
Time-stretch-master	Time-stretch-master	**0.6801**
	Audiotsm	**0.6256**
	Torchaudio	0.6759

Table 5.3: Classifier Results with MedResPh as input. In bold, the maximum and the minimum achieved accuracy values.

Training	Test Algorithm	Accuracy
Audiotsm	Audiotsm	0.5998
	Torchaudio	0.6406
	Time-stretch-master	0.6393
Torchaudio	Torchaudio	**0.6617**
	Audiotsm	0.6074
	Time-stretch-master	**0.6617**
Time-stretch-master	Time-stretch-master	0.6522
	Audiotsm	**0.5979**
	Torchaudio	0.6524

5.4 Regressor Results

For the estimation task, the accuracy criterion is the Pearson Correlation Coefficient (PCC), defined as:

$$\mathrm{PCC} = \frac{\mathrm{cov}(\hat{Y}, Y)}{\sigma_{\hat{Y}} \sigma_Y}, \tag{5.4}$$

where:

- $\hat{Y}$ is the array containing the outputs resulting from all the input testing data, namely $\hat{Y} = \{\hat{\alpha_1}, \hat{\alpha_2}, \cdots, \hat{\alpha_N}\}$, where N is the amount of samples in the testing subset;

Table 5.4: Classifier Results with all the three kinds of input together, i.e. the LMS, the HPF-Ph and the MedResPh. In bold, the maximum and the minimum achieved accuracy values

Training	Test Algorithm	Accuracy
Audiotsm	Audiotsm	0.7162
	Time-stretch-master	0.7118
	Torchaudio	0.7083
Torchaudio	Torchaudio	0.7259
	Audiotsm	0.6961
	Time-stretch-master	**0.7259**
Time-stretch-master	Time-stretch-master	0.7136
	Audiotsm	**0.6906**
	Torchaudio	0.7138

- Y contains the relative ground truths, i.e. $Y = \{\alpha_1, \alpha_2, \cdots, \alpha_N\}$;
- $\sigma_{\hat{Y}}$ and σ_Y are the standard deviations of $\hat{Y}$ and Y;
- $\text{cov}(Y, \hat{Y})$ the covariance of $\hat{Y}$ and Y.

For the regression task, we developed two approaches to get the results after the training stage: one is based on the same working principle of the classifier, i.e. making a prediction by looking to only 64 frequency bins and 96 temporal samples of the query signal. We noticed that this method suffered a bit from the impossibility to gather some information by the temporal evolution of the signal, considering just one temporal window too much "poor" to represent the entire signal.

For this reason, we decided to make also a prediction about the entire signal, and not only about a single portion of it. Specifically, we apply the previously-trained model on all the temporal windows of each audio track, i.e. all the subsequent temporal windows of 64 frequency bins and 96 time samples (Fig. 5.3). Then, we reduced the M predictions related to the M temporal windows to a single value.

In particular, we computed the prediction $\hat{\alpha}_n$ about the n^{th} audio signal by means of three different operations:

1. In the first case, we computed $\hat{\alpha}_n$ as the arithmetic mean between the M output values;
2. as an alternative scenario, we computed $\hat{\alpha}_n$ as their median value;

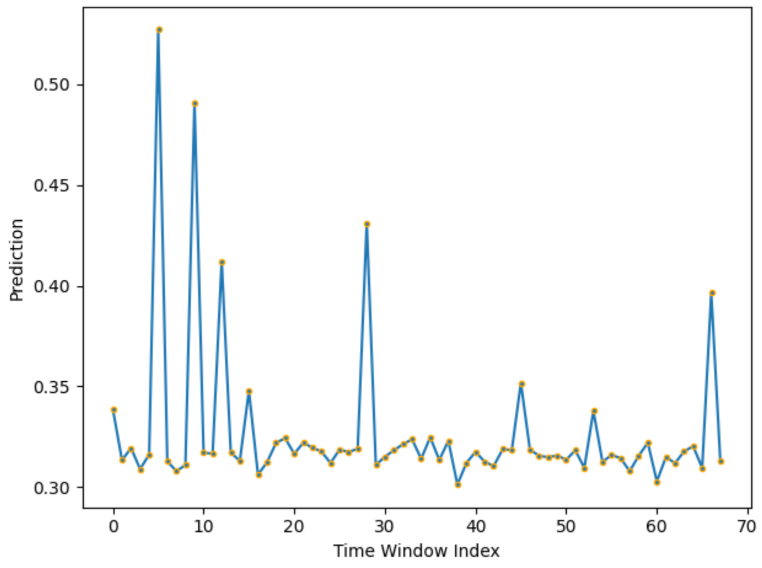


Figure 5.3: The temporal evolution of the predictions for each time window of a time-scaled audio signal with a factor $\alpha = 0.3$ by means of Torchaudio algorithm. In this case the model received a LMS as input, and it has been trained with the Time-stretch-master algorithm.

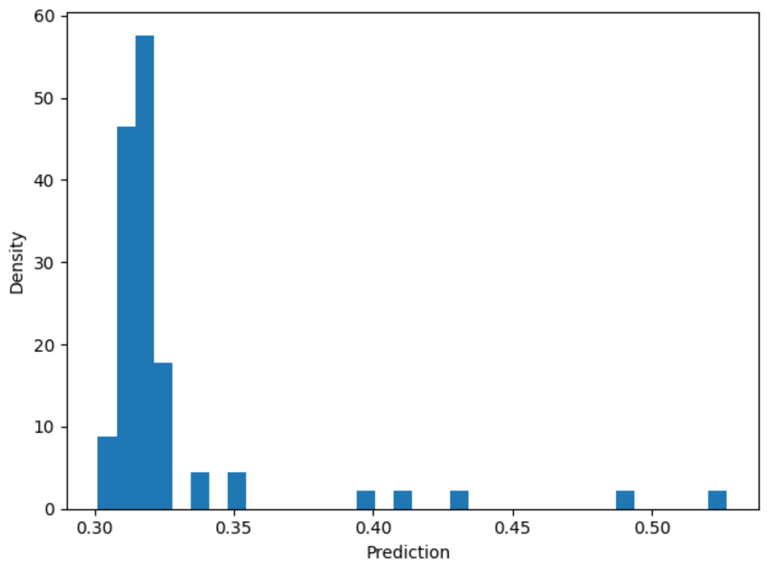


Figure 5.4: The histogram of the predictions about all the time windows of a time-scaled audio signal with a factor $\alpha = 0.3$ by means of Torchaudio algorithm. In this case the model received a LMS as input, and it has been trained with the Time-stretch-master algorithm.

3. finally, we computed $\hat{\alpha}_n$ as the mode between the M values. Since the regressor's output values are continuous values, the practice to compute the mode is to discretize the data by making a histogram,

i.e. replacing the values by the midpoints of the intervals they are assigned to. The mode is then the value where the histogram reaches its peak (Fig. 5.4).

Regarding both approaches, every regression task has been accomplished by using in the training stage only 10 time-scaling factors (i.e. from 0.1 to 1.9 with a step of 0.2), plus the original tracks. In the testing stage, we decided to use three different sets of time-scaling factors:

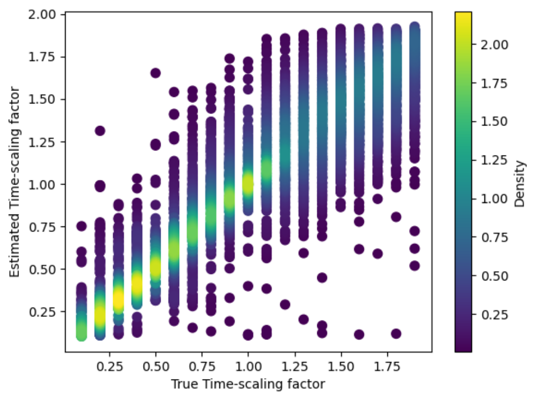
- The time-scaling factors used for at training time ("PCC Tr"), i.e. from 0.1 to 1.9 with a step of 0.2, plus the original tracks with $\alpha = 1$;
- The factors not used in the training stage ("PCC NT'), i.e. from 0.2 to 2.0 with a step of 0.2, in addition to the original tracks;
- The union of the time-scaling factor presented in the two previous points ("PCC All"), i.e. from 0.1 to 2.0 with a step of 0.1.

5.4.1 Single Time Window Results

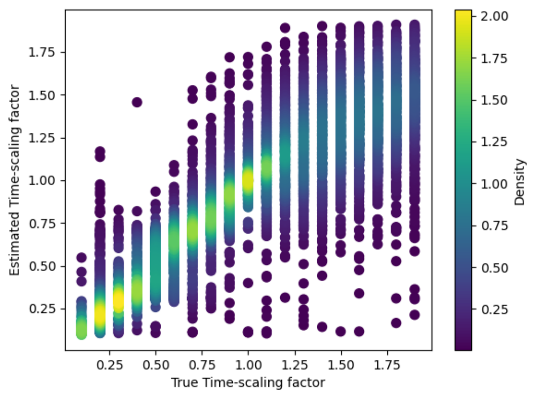
Here we will see and comment the results according to the first method we have depicted, i.e. the analysis of a single temporal window for each audio track. In Fig. 5.5 and Fig. 5.6 we can see two examples of the visual representation we chose to describe the test results as a whole. From these scatter plots it emerges that the predictions for $\alpha > 1$ are worse, since the concentration of the results is weaker, with more vertical dispersion than predictions for lower factors, where the results are more concentrated near the correct value. The cause of this behavior is likely to be that time-scaling with $\alpha < 1$ implies the application of some interpolation, with the introduction of artificial samples, that are easier to detect. By contrast, time-scaling with $\alpha > 1$ involves the rejection of some samples, making harder the recognition of artifacts in the audio signal.

From Table 5.5 to Table 5.8 we have the results referring to the regression task, considering separately each prediction about every single input temporal frame.

In Table 5.5 we can see the PCC values resulting from the regression tests with LMS as input to the CNN. The best outcomes have been achieved when we used the same algorithm for training and testing stages, ora with cross-dataset setups between Torchaudio and Time-stretch-master. The best result is 0.9391, obtained for both the models trained with Audiotsm and Torchaudio. By contrast, we have got



(a) The results of tests with LMS and Audiotsm algorithm.



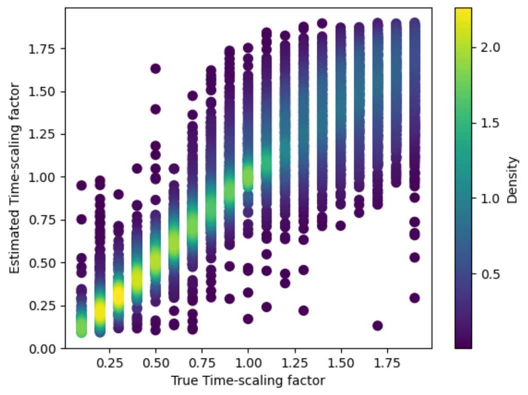
(b) The results of tests with LMS and Audiotsm algorithm (training), Time-stretch-master (testing).

Figure 5.5: Some examples of regression results (single time-window approach).

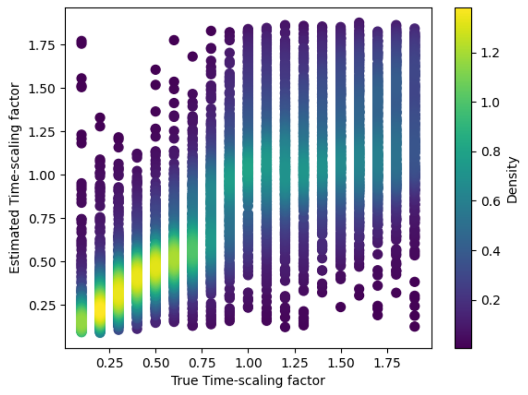
the worst results when we tested the Audiotsm algorithm with a model trained with Torchaudio or Time-stretch-master. The worst case is represented by the model trained with Torchaudio and tested with Audiotsm, with PCC = 0.7728.

In Tables 5.6 and 5.7 we have shown the results related to HPF-Ph and MedResPh. We were able to achieve moderately good results, such as 0.8699 (Table 5.6), or 0.8415 (Table 5.7), in the cross-dataset case with Time-stretch-master for training ans Torchaudio for test. The worst results, as we have seen in other cases, are represented by the models trained with Torchaudio or Time-stretch-master, and tested with Audiotsm ($\simeq 0.6$).

Finally, in Table 5.8 we show the results of the tests conducted



(a) The results of tests with all the three kinds of input together and Torchaudio algorithm (training), Time-stretch-master (testing).



(b) The results of tests with MedResPh and Audiotsm algorithm (training), Torchaudio (testing).

Figure 5.6: Some examples regression results (single time-window approach).

with all the three possible kinds of input together. In this case we were able to achieve PCC values such as 0.9172, with the model trained with Torchaudio and tested with Time-stretch-master. However, the worst results are quite satisfactory, such as PCC = 0.7745, in the case of the model trained with Torchaudio and tested with Audiotsm, using the time-scaling factors not present during the training process.

As we mentioned before, the “PCC Tr” column reports the results regarding the tests conducted with the same time-scaling factors used to train the CNN. For this reason, we expected to find the best results with this configuration. On the contrary, in the “PCC NT” column we have the results for the time-scaling factors which the CNN has never encountered during training. Hence, we assumed to achieve the worst results with this configuration. Finally, the “PCC All” column repre-

Table 5.5: Regressor Results with LMS as input (single time-window approach). In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	**0.9391**	0.9164	0.9299
	Torchaudio	0.9015	0.8784	0.8926
	Tstretch	0.8998	0.8785	0.8915
Torchaudio	Torchaudio	**0.9391**	0.9275	0.9340
	Audiotsm	**0.7728**	0.7965	0.7814
	Tstretch	0.9385	0.9275	0.9337
Tstretch	Tstretch	0.9375	0.9222	0.9313
	Audiotsm	0.7809	0.7846	0.7822
	Torchaudio	0.9376	0.9221	0.9313

Table 5.6: Regressor Results with HP-Ph as input (single time-window approach). In bold, the maximum and the minimum achieved PCC values

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.8120	0.8222	0.8159
	Torchaudio	0.7797	0.8185	0.7943
	Tstretch	0.7792	0.8203	0.7947
Torchaudio	Torchaudio	0.8460	0.8652	0.8534
	Audiotsm	0.6453	**0.6247**	0.6364
	Tstretch	0.8475	0.8665	0.8548
Tstretch	Tstretch	0.8365	0.8681	0.8486
	Audiotsm	0.6759	0.6534	0.6666
	Torchaudio	0.8346	**0.8699**	0.8482

sents the most general case, having both the factors present during the training and those never encountered before. Therefore, with this setup, we supposed to obtain results in between the first two columns.

This forecast was generally respected in Tables 5.5 and 5.8, namely in the tests with respectively LMS and all the three kinds of input together (i.e. LMS, HPF-Ph and MedResPh) as input. By contrast, we can see the exactly opposite outcome in Tables 5.6 and 5.7, i.e. in the tests with HPF-Ph and MedResPh as input. We think this unexpected behaviour surely needs further investigations and will be tackled in our future research.

Table 5.7: Regressor Results with MedResPh as input (single time-window approach). In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.7455	0.7670	0.7538
	Torchaudio	0.7380	0.7670	0.7488
	Tstretch	0.7383	0.7663	0.7487
Torchaudio	Torchaudio	0.8098	0.8412	0.8219
	Audiotsm	0.6280	**0.6030**	0.6181
	Tstretch	0.8127	**0.8415**	0.8238
Tstretch	Tstretch	0.8248	0.8409	0.8309
	Audiotsm	0.6362	0.6117	0.6262
	Torchaudio	0.8252	**0.8415**	0.8314

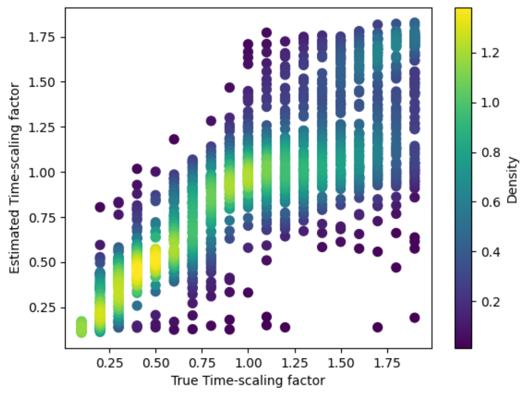
Table 5.8: Regressor Results with the three kinds of input together, i.e. the LMS, the HPF-Ph and the MedResPh (single time-window approach). In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.9006	0.8835	0.8937
	Torchaudio	0.8844	0.8654	0.8771
	Tstretch	0.8853	0.8658	0.8778
Torchaudio	Torchaudio	0.9167	0.9132	0.9150
	Audiotsm	0.7972	**0.7745**	0.7870
	Tstretch	**0.9172**	0.9125	0.9150
Tstretch	Tstretch	0.9146	0.9144	0.9141
	Audiotsm	0.7933	0.7792	0.7870
	Torchaudio	0.9148	0.9150	0.9144

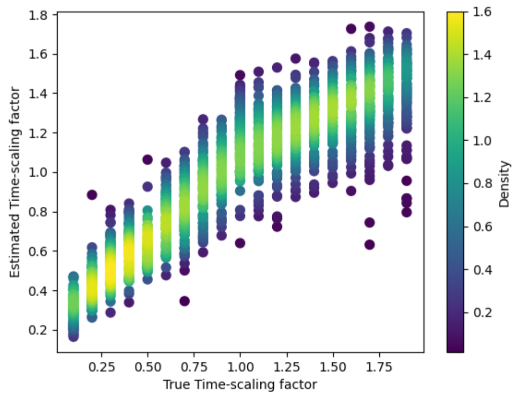
5.4.2 Multiple-Window Results

In this subsection we will go into the details of the approach consisting of doing the test on the entire audio signal, then taking a final value according to the arithmetic mean, the median value or the mode.

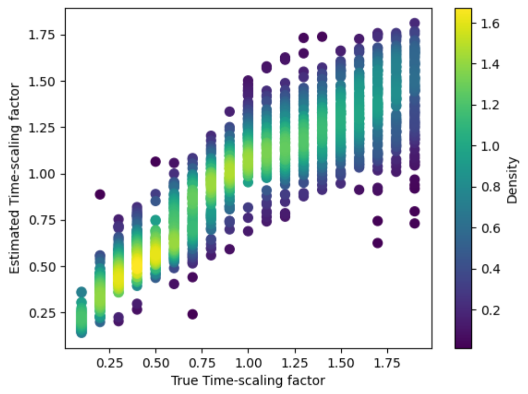
In Fig. 5.7 we can see two examples of the scatter plots that we used to represent the test results, as depicted in the previous section.



(a) The results of the "multiple time-window" approach for regression task (max value).



(b) The results of the "multiple time-window" approach for regression task (average value).



(c) The results of the "multiple time-window" approach for regression task (median value).

Figure 5.7: Some examples of results for the regression task with the "multiple time-window" approach, MedResPh as input, and Audiotsm time-scaling algorithm.

In this case, the results have been obtained with MedResPh in input and the dataset built with the Audiotsm time-scaling algorithm. For this combination, the training process has been conducted on 11 time-scaling factors, from 0.1 to 1.9, with a step of 0.2, including the original signals, with $\alpha = 1$. The testing stage, instead, as we can see from the scatter plots is conducted on 20 time-scaling factors, i.e. from 0.1 to 2.0, with a step of 0.1.

For instance, in Fig. 5.7a, the results have been computing by taking the Mode value between all the predictions about the same track. In a similar way, Fig. 5.7b and Fig. 5.7c show respectively the results obtained by means of the arithmetic mean and the median value between all the predictions about a track. As reported before in Section 5.4.1, we can see better results for low scaling factors (i.e. $\alpha < 1$), while, for higher factors (i.e. $\alpha > 1$), we can see how in all the cases the results exhibit a more prominent vertical dispersion. This is probably be due to the fact that time-scaling with $\alpha < 1$ can be seen as downsampling, implying that some interpolation is applied, hence new artificial samples are introduced into the signal. On the contrary, time-scaling with $\alpha > 1$ can be seen as a type of up-sampling, then it is a process in which we are "discarding" some information from the original signal. That is why it can be more difficult to detect time-scaling modification artifacts in those situations.

As we already described, each test case is summarized by the PCC, as shown from Table 5.9 to Table 5.20. In Table 5.9 we can see the results from the tests with LMS in input, with the Mode approach, while in Table 5.10 we refer to the Mean approach and in Table 5.11 to the Median value. Given that the results are quite good in all the cases that we just mentioned (worst result: 0.8201), we can notice that this system better performs with the Mean and Median approach, since they exhibit comparable results. The CNN has shown excellent results also with cross-dataset test, especially between Time-stretch-master and Torchaudio algorithms, while it seems to provide slightly worst results when we test with Audiotsm the model trained with the other two algorithms. The results show also that they are quite good even if we test the network with time-scaling factors "never seen" during training (e.g. PCC = 0.9670 in Table 5.10). At last, the best result is obtained with a cross-dataset set-up, using Time-stretch-master algorithm for training and Torchaudio for test, with the mean approach, and keeping only the factors used for training (PCC = 0.9738, Table 5.10).

As we could expect, the tests with LMS as input exhibit the best results if we consider only the time-scaling factors employed during training

Table 5.9: The summary of the tests for regression (multiple time-window approach), Mode value, LMS as input. In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	**0.9600**	0.9425	0.9532
	Torchaudio	0.9260	0.9162	0.9219
	Tstretch	0.9261	0.9038	0.9170
Torchaudio	Torchaudio	0.9566	0.9401	0.9499
	Audiotsm	**0.8201**	0.8423	0.8282
	Tstretch	0.9562	0.9380	0.9489
Tstretch	Tstretch	0.9513	0.9459	0.9492
	Audiotsm	0.8270	0.8396	0.8315
	Torchaudio	0.9498	0.9478	0.9490

Table 5.10: The summary of the tests for regression (multiple time-window approach), Mean value, LMS as input. In bold, the maximum and the minimum achieved PCC values, and two results for time-scaling factors not present during training.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.9737	0.9623	0.9691
	Torchaudio	0.9506	0.9397	0.9463
	Tstretch	0.9507	0.9397	0.9464
Torchaudio	Torchaudio	0.9722	0.9664	0.9694
	Audiotsm	**0.8593**	0.8831	0.8678
	Tstretch	0.9720	0.9664	0.9693
Tstretch	Tstretch	0.9737	**0.9670**	0.9708
	Audiotsm	0.8662	0.8682	0.8664
	Torchaudio	**0.9738**	**0.9670**	0.9708

(i.e. “PCC tr” column), while we have the lowest values for the values never “seen” by the CNN and intermediate ones if we consider a combination of the previous two. The only exception about this behavior is represented by the cross-dataset configurations in which we did the tests with the Audiotsm algorithm, in which we can see the exactly opposite outcomes.

The results related to the tests with HPF-Ph as input are shown in

Table 5.11: The summary of the tests for regression (multiple time-window approach), Median value, LMS as input. In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	**0.9754**	0.9578	0.9684
	Torchaudio	0.9503	0.9386	0.9458
	Tstretch	0.9506	0.9387	0.9460
Torchaudio	Torchaudio	0.9726	0.9656	0.9693
	Audiotsm	**0.8508**	0.8788	0.8601
	Tstretch	0.9727	0.9656	0.9693
Tstretch	Tstretch	0.9726	0.9652	0.9694
	Audiotsm	0.8581	0.8666	0.8603
	Torchaudio	0.9726	0.9652	0.9695

Tables 5.12, 5.13 and 5.14 with respectively the Mode, Mean and Median approach to extract the result from the predictions of each track. Here the results are slightly worse than before, but we still reach values such as PCC = 0.9421 in Table 5.13 with Mean approach and Time-stretch-master algorithm for training and Torchaudio for test, but, moreover, with time-scaling factors not used during training. As in the previous case with LMS, also with HPF-Ph the best cross-dataset results are achieved between Time-stretch-master and Torchaudio algorithms, while Audiotsm performs better if used both for training and test. Also with this combination the results get worse with cross-dataset tests where the test dataset is built with the Audiotsm algorith: indeed, in Table 5.12 we get the lowest result (PCC = 0.6552) with Torchaudio algorithm for the training stage, and time-scaling factors not present in the training process.

If we consider the tests with HPF-Ph as input, the outcome related to the set of time scaling factors usied during tests confirmed our expectations only in Table 5.9, i.e. with the “Mode” approach. For the other two combinations, instead, the outcome has been substantially the opposite, with best results for the “PCC NT” column and worst for the “PCC Tr” one.

The situation related to tests with MedResPh as input is depicted in Tables 5.15, 5.16 and 5.17. The best results have been obtained by taking the mean of all the predictions for each track (Table 5.16, PCC = 0.9344), and we still can see the excellent results with cross-

Table 5.12: The summary of the tests for regression (multiple time-window approach), Mode value, HPF-Ph as input. In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.8752	0.8917	0.8816
	Torchaudio	0.8358	0.8703	0.8484
	Tstretch	0.8258	0.8671	0.8420
Torchaudio	Torchaudio	0.8945	0.8930	0.8939
	Audiotsm	0.7450	**0.6552**	0.7082
	Tstretch	**0.8977**	0.8891	0.8943
Tstretch	Tstretch	0.8808	0.8961	0.8868
	Audiotsm	0.7589	0.7082	0.7370
	Torchaudio	0.8771	0.8955	0.8843

Table 5.13: The summary of the tests for regression (multiple time-window approach), Mean value, HPF-Ph as input. In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.9188	0.9416	0.9272
	Torchaudio	0.8820	0.9227	0.8969
	Tstretch	0.8816	0.9235	0.8970
Torchaudio	Torchaudio	0.9207	0.9365	0.9267
	Audiotsm	0.8010	**0.7964**	0.7967
	Tstretch	0.9214	0.9376	0.9275
Tstretch	Tstretch	0.9197	0.9419	0.9283
	Audiotsm	0.8336	0.8320	0.8308
	Torchaudio	0.9197	**0.9421**	0.9283

dataset tests between Torchaudio and Time-stretch-master, in addition to good results for other combinations. Also in this case, the lowest PCC is obtained with cross-dataset tests, with Time-stretch-master for training and Audiotsm for test (Table 5.15, PCC = 0.6707).

Unexpectedly, the results referring to tests with MedResPh as input, generally exhibit the best results if we consider the tests with only the factors not present during the training stage. On the contrary, the worst results have been achieved when we tested the CNN with the factors we

Table 5.14: The summary of the tests for regression (multiple time-window approach), Median value, HPF-Ph as input.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.9199	0.9384	0.9265
	Torchaudio	0.8842	0.9210	0.8979
	Tstretch	0.8827	0.9196	0.8965
Torchaudio	Torchaudio	0.9230	0.9349	0.9273
	Audiotsm	0.7888	0.7725	0.7787
	Tstretch	0.9232	0.9361	0.9279
Tstretch	Tstretch	0.9202	0.9391	0.9275
	Audiotsm	0.8213	0.8050	0.8118
	Torchaudio	0.9195	0.9390	0.9270

Table 5.15: The summary of the tests for regression (multiple time-window approach), Mode value, MedResPh as input. In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.8513	0.8423	0.8478
	Torchaudio	0.8221	0.8450	0.8308
	Tstretch	0.8077	0.8316	0.8166
Torchaudio	Torchaudio	0.8576	0.8817	0.8670
	Audiotsm	0.7427	0.6778	0.7174
	Tstretch	0.8673	0.8612	0.8648
Tstretch	Tstretch	0.8718	0.8729	0.8721
	Audiotsm	0.7447	**0.6707**	0.7154
	Torchaudio	0.8807	**0.8842**	0.8817

used to train it. This unexpected behavior will certainly be subject of future investigations.

Finally, in Tables 5.18, 5.19 and 5.20 we have the values of the results for the tests with all the three kinds of input (i.e. the LMS, HPF-Ph and MedResPh) for the CNN. Here the results about the PCC are notably good, since they are all located in the range between 0.8153 (Table 5.18), in the case of the cross-dataset test with Torchaudio algorithm for training and Audiotsm for test, with time-scaling factors not present in the training stage, and 0.9631 (Table 5.20), in the cross-dataset test with

Table 5.16: The summary of the tests for regression (multiple time-window approach), Mean value, MedResPh as input. In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.8972	0.9288	0.9091
	Torchaudio	0.8693	0.9112	0.8844
	Tstretch	0.8697	0.9115	0.8847
Torchaudio	Torchaudio	0.9074	0.9306	0.9162
	Audiotsm	**0.8079**	0.8220	0.8116
	Tstretch	0.9071	0.9311	0.9162
Tstretch	Tstretch	0.9145	**0.9344**	0.9219
	Audiotsm	**0.8079**	0.8157	0.8092
	Torchaudio	0.9145	0.9341	0.9218

Table 5.17: The summary of the tests for regression (multiple time-window approach), Median value, MedResPh as input.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.8976	0.9234	0.9070
	Torchaudio	0.8701	0.9017	0.8813
	Tstretch	0.8703	0.9027	0.8819
Torchaudio	Torchaudio	0.9093	0.9270	0.9159
	Audiotsm	0.8049	0.8069	0.8034
	Tstretch	0.9101	0.9260	0.9160
Tstretch	Tstretch	0.9129	0.9327	0.9200
	Audiotsm	0.7978	0.7997	0.7957
	Torchaudio	0.9124	0.9324	0.9196

Time-stretch-master algorithm and the same time-scaling factors present in the training stage. Differently from other combinations, we can notice quite good results also referring to cross-dataset tests with the Audiotsm algorithm used in the testing stage (e.g. PCC = 0.8828, Table 5.19, and PCC = 0.8760, Table 5.20).

As we could imagine, if we consider the results with all the three kinds of input together (i.e. the LMS, HPF-Ph and MedResPh), the best results have been achieved most of the times if we tested the CNN with only the factors present, during training. The worst results, instead,

Table 5.18: The summary of the tests for regression (multiple time-window approach), Mode value, three kinds of input together, i.e. the LMS, the HPF-Ph and the MedResPh. In bold, the maximum and the minimum achieved PCC values.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.9319	0.9100	0.9234
	Torchaudio	0.9017	0.8910	0.8974
	Tstretch	0.9023	0.8974	0.9003
Torchaudio	Torchaudio	0.9338	**0.9381**	0.9352
	Audiotsm	0.8471	**0.8153**	0.8332
	Tstretch	0.9354	0.9290	0.9328
Tstretch	Tstretch	0.9363	0.9347	0.9356
	Audiotsm	0.8484	0.8380	0.8426
	Torchaudio	0.9344	0.9353	0.9347

Table 5.19: The summary of the tests for regression (multiple time-window approach), Mean value, three kinds of input together, i.e. the LMS, the HPF-Ph and the MedResPh. In bold, the cross-dataset test result for the model with Torchaudio algorithm used for training and Audiotsm in the testing stage.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.9534	0.9494	0.9516
	Torchaudio	0.9371	0.9320	0.9350
	Tstretch	0.9369	0.9320	0.9348
Torchaudio	Torchaudio	0.9600	0.9598	0.9596
	Audiotsm	0.8887	0.8763	**0.8828**
	Tstretch	0.9600	0.9599	0.9596
Tstretch	Tstretch	0.9614	0.9610	0.9609
	Audiotsm	0.8860	0.8765	0.8814
	Torchaudio	0.9601	0.9607	0.9599

are shown in the “PCC NT” column, and, finally, the tests with all the time-scaling factors brought to intermediate PCC values.

Table 5.20: The summary of the tests for regression (multiple time-window approach), Median value, three kinds of input together, i.e. the LMS, the HPF-Ph and the MedResPh. In bold, the maximum and the minimum achieved PCC values, and the cross-dataset test result for the model with Time-stretch-master algorithm used for training and Audiotsm in the testing stage.

Train Alg.	Test Alg.	PCC Tr	PCC NT	PCC All
Audiotsm	Audiotsm	0.9551	0.9455	0.9511
	Torchaudio	0.9359	0.9362	0.9359
	Tstretch	0.9360	0.9355	0.9357
Torchaudio	Torchaudio	0.9600	0.9600	0.9596
	Audiotsm	0.8810	**0.8690**	0.8743
	Tstretch	0.9602	0.9603	0.9598
Tstretch	Tstretch	**0.9631**	0.9605	0.9615
	Audiotsm	0.8781	**0.8760**	0.8754
	Torchaudio	0.9617	0.9599	0.9605

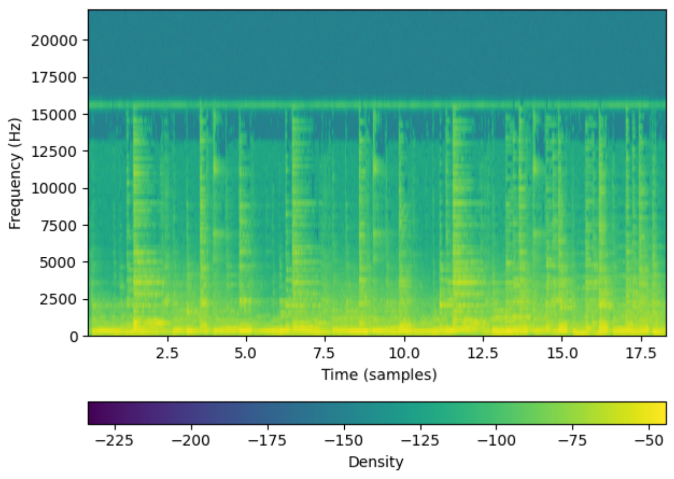
5.5 Conclusive Remarks

Let us summarize finally the results we got from all the tests on original and time-scaled signal. For example, we can see in Fig. 5.8 the spectrograms of a time-scaled signal (Fig. 5.8a) and of the respective original one (Fig. 5.8b).

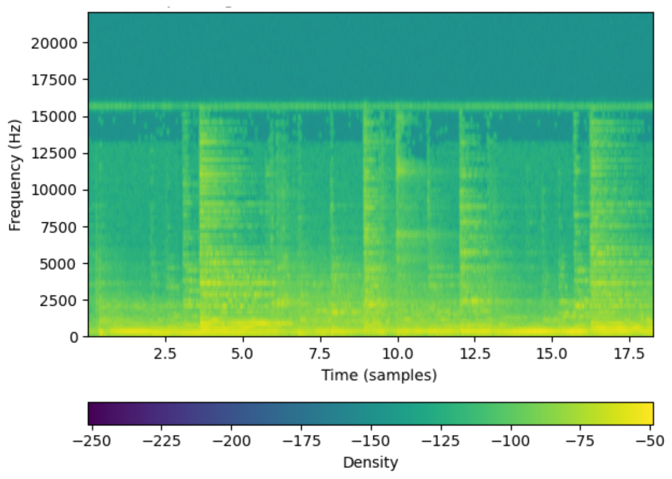
Starting from the results we showed and analyzed in 5.3 and 5.4, we can derive some final considerations about the obtained results, referring to cross-dataset configurations and the chosen input for the CNN.

Specifically, we noticed excellent results for cross-dataset tests when we put together Torchaudio and Time-stretch master algorithms. On the contrary, the outcomes worsened considerably in all the cross-dataset setups with Audiotsm algorithm in testing stage.

Concerning the input of the CNN, for all cases the best results have been achieved with the LMS, despite we were able in some situations to get satisfying results also with the HPF-Ph and the MedResPh.



(a) Spectrogram of original signal.



(b) Spectrogram of time-scaled signal.

Figure 5.8: Spectrograms of original (a) and time-scaled (b) signals.

6 Conclusions and Future Works

This thesis proposed a methodology in the audio forensics field for automatic time-scaling detection and estimation, i.e. respectively to find if some sort of time-scaling modification has been applied to an audio signal and eventually to find the right value of the time-scaling factor.

Typically, time-scaling modification algorithms leave some interpolation traces on the Short-Time Fourier Transform (STFT) of the input signal. For this reason, we tried to blindly investigate audio signals in order to find such artifacts. The proposed methodology is based on a data-driven approach, using a Convolutional Neural Network (CNN) trained on original and time-scaled audio signals.

In this thesis we have set two main tasks regarding time-scaling identification:

- Classification, i.e. indicating if the input signal is the original one or a time-scaled modified version. In this case, the CNN goal is to produce an output discrete number, such that 0 means that the track is the original one and 1 indicates that the input was a time-scaled signal;

- Regression, i.e. predicting the time-scaling factor α of an audio signal. For this task, the CNN should give as output a continuous value, indicating the estimation of the eventual factor of the applied time-scaling modification algorithm (if no modification has been applied, the output should be 1).

The regression task has been accomplished by proposing two different approaches:

1. The first consisted of making a prediction for a single temporal window composed by 64 frequency bins and 96 time samples that we have extracted from the input audio signal;

2. The second one involved that each input signal has been divided into many subsequent temporal windows, then a prediction has been made about each one of them. Finally, we computed the overall estimation for the input signal by computing the mean, the median value or the mode between all the predictions.

The aforementioned CNN has been trained with different kinds of 2D representations of the audio signal itself, i.e. the Log-Mel spectrogram, a high-pass filtered version of the unwrapped phase, a median filtered version of the unwrapped phase, and the three of them together.

We built the dataset to train and test our CNN by combining original audio signals with time-scaled ones, obtained by means of three different algorithms: Audiotsm [5], Torchaudio [6] and Time-stretch-master [7]. Our tests have been conducted either by using the same dataset of the training stage and by adopting a cross-dataset approach, i.e. choosing two different datasets for training and test. This allowed us to measure the ability of the CNN to recognize general artifacts of time-scaling algorithms regardless of the specific training dataset used.

The test results for classification showed good accuracy values when the Log-Mel spectrogram has been used as input (e.g. accuracy is always greater than 0.85), either with cross-dataset configurations and when we used the same dataset during training and test stages. The situation regarding the classification task worsened noticeably when we considered the two filtered versions of the unwrapped phase as input, since in these cases we achieved accuracy values between 0.5979 and 0.6801, i.e. a slightly better result than a random approach. When we considered the three inputs together (i.e., the Log-Mel spectrogram and the two filtered versions of the unwrapped phase), we were able to achieve accuracies such as 0.7259 with the model trained using the dataset built from Torchaudio algorithm.

If we referred to the regression task with the analysis of a single time window for each audio signal, the best results have been achieved with the Log-Mel spectrogram as input (e.g. we can achieve a Pearson Correlation Coefficient (PCC) = 0.9391), considering either the Torchaudio or Audiotsm datasets with the time-scaling factors present during training. Tests with a more general configuration (i.e. with time-scaling factors both present and not during training) exhibited good result as well, such as PCC = 0.9337 in cross-dataset setups. PCC values became lower when we used the unwrapped phases as input of the CNN, reaching results such as PCC = 0.6030 in the worst case. If we used instead all the three kind of inputs together, we were able to achieve PCC = 0.9050 when we tested the model trained on the Torchaudio dataset with all the time-scaling factors.

Finally, considering the multiple time window approach, we found out the same qualitative behavior of the single-window procedure, that is, the best results have been achieved with Log-Mel spectrogram as input, while the worst came out from tests with the unwrapped phase as input. The real salient feature of this second approach was that we can gather more information from the temporal evolution of the track. This allowed us to achieve excellent results such as PCC = 0.9754 in the case of the Audiotsm algorithm in both training and test, and the mode approach to compute the final prediction about the signal. Furthermore, we obtained comparable results such as PCC = 0.9708 in cross-dataset configuration and all the time-scaling factors during testing, denoting a good ability to perform in a reliable way also in real scenarios.

6.1 Future Works

As we mentioned before, the tests that we conducted allowed us to point out the strongest features of our system. However, there are still a few aspects that are worthy of investigations.

For instance, we would like to improve the results of the classifier, maybe by finding the right compromise between the dataset size, the input pre-processing and the CNN parameters. To overcome this issue, we will investigate the parameters by means of which we compute the STFT, its phase and the related filters, as well as the mel-scale conversion to adapt the frequency scale.

For both the classification and regression tasks, we clearly noticed some configurations that lead to worse results, namely the two filtered versions of the unwrapped phase as input of the CNN, and the cross-dataset tests with the Audiotsm dataset employed during the testing stage. We will spend some more effort to train the CNN for these specific scenarios.

Once solved these issues, a possible future research area will be the detection of the specific algorithm used to modify the signal (among a hopefully larger selection of time-scaling implementations). Once we identify the time-scaling factor and the exact algorithm used, another task we are interested in is to restore the audio track with its original time evolution. To achieve this, we would like to train a CNN by minimizing the error between the resulting restored signal (i.e. the prediction) and the desired original one (i.e. the ground truth).

Bibliography

[1] M. Kirchner and T. Gloe, “On resampling detection in re-compressed images,” in *2009 First IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 21–25, 2009.

[2] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” 2019.

[3] “Efficientnet: Improving accuracy and efficiency through automl and model scaling.” `https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html`. Accessed: 2021-03-07.

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2019.

[5] Muges, “Audiotsm phase vocoder.” `https://github.com/Muges/audiotsm`, Oct. 2017. Accessed: 2021-02-21.

[6] Pytorch, “Time stretch.” `https://pytorch.org/audio/stable/_modules/torchaudio/transforms.html#TimeStretch`. Accessed: 2021-02-21.

[7] “Time stretch master.” `https://github.com/gaganbahga/time_stretch`, Dec. 2020. Accessed: 2021-02-22.

[8] G. Cariolaro, “Unified signal theory,” 2011.

[9] M. Portnoff, “Magnitude-phase relationships for short-time fourier transforms based on gaussian analysis windows,” in *ICASSP ’79. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, (Washington, DC, USA), pp. 186–189, IEEE, 1979.

[10] K. K. Paliwal and L. D. Alsteris, “On the usefulness of stft phase spectrum in human listening tests,” *Speech Communication*, vol. 45, no. 2, pp. 153–170, 2005.

[11] J. Martinez-Carranza, K. Falaggis, and T. Kozacki, “Fast and accurate phase-unwrapping algorithm based on the transport of intensity equation,” *Applied Optics*, vol. 56, p. 7079, 09 2017.

[12] 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.

[13] S. Umesh, L. Cohen, and D. Nelson, “Fitting the mel scale,” 1999.

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

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

[16] M. Kahrs and K. Brandenburg, *Applications of Digital Signal Processing to Audio and Acoustics.* USA: Kluwer Academic Publishers, 1998.

[17] J. Makhoul and A. El-Jaroudi, “Time-scale modification in medium to low rate speech coding,” in *ICASSP ’86. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 11, pp. 1705–1708, 1986.

[18] L. Almeida and F. Silva, “Variable-frequency synthesis: An improved harmonic coding scheme,” in *ICASSP ’84. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 9, pp. 437–440, 1984.

[19] R. McAulay and T. Quatieri, “Speech analysis/synthesis based on a sinusoidal representation,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 4, pp. 744–754, 1986.

[20] J. Driedger and M. Müller, “A review of time-scale modification of music signals,” *Applied Sciences*, vol. 6, no. 2, 2016.

[21] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurons in the cat’s striate cortex,” *Journal of Physiology*, vol. 148, pp. 574–591, 1959.

[22] Y. LeCun, K. Kavukcuoglu, and C. Farabet, “Convolutional networks and applications in vision,” 2010.

[23] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[24] K. Fukushima, "Cognitron: A self-organizing multilayer neural network," *Biological Cybernetics*, vol. 20, pp. 121–136, 1975.

[25] Siwei Lyu and E. P. Simoncelli, "Nonlinear image representation using divisive normalization," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[26] N. Pinto, D. D. Cox, and J. J. DiCarlo, "Why is real-world visual object recognition hard?," *PLoS Comput Biol*, vol. 4, p. e27, 2008.

[27] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*. Adaptive computation and machine learning, MIT Press, 2016.

[28] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *Journal of Physiology (London)*, vol. 195, pp. 215–243, 1968.

[29] Y. Lemoigne and A. Caner, *Molecular Imaging: Computer Reconstruction and Practice*. NATO Science for Peace and Security Series B: Physics and Biophysics, Springer Netherlands, 2008.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[31] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 07 2011.

[32] M. C. Mukkamala and M. Hein, "Variants of RMSProp and Adagrad with logarithmic regret bounds," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 2545–2553, PMLR, 06–11 Aug 2017.

[33] "MS Windows NT kernel description." `https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/`. Accessed: 2021-03-17.

[34] G. Hua and H. Zhang, "Enf signal enhancement in audio recordings," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1868–1878, 2020.

[35] D. Luo, M. Sun, and J. Huang, "Audio postprocessing detection based on amplitude cooccurrence vector feature," *IEEE Signal Processing Letters*, vol. 23, pp. 688–692, 2016.

[36] Q. Yan, R. Yang, and J. Huang, "Copy-move detection of audio recording with pitch similarity," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1782–1786, 2015.

[37] Y. Zhan and X. Yuan, "Audio post-processing detection and identification based on audio features," in *2017 International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR)*, pp. 154–158, 2017.

[38] R. Korycki, "Detection of tampering in lossy compressed digital audio recordings," in *2012 Joint Conference New Trends In Audio & Video And Signal Processing: Algorithms, Architectures, Arrangements And Applications (NTAV/SPA)*, (Lodz), pp. 97–101, IEEE, 2012.

[39] Z. W. D. Y. R. Wang, L. Xiang, and T. Wu, "Speech resampling detection based on inconsistency of band energy," *Computers, Materials & Continua*, vol. 56, no. 2, pp. 247–259, 2018.

[40] B. Liang, G. Fazekas, and M. Sandler, "Piano sustain-pedal detection using convolutional neural networks," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 241–245, 2019.

[41] M. Mascia, A. Canclini, F. Antonacci, M. Tagliasacchi, A. Sarti, and S. Tubaro, "Forensic and anti-forensic analysis of indoor/outdoor classifiers based on acoustic clues," in *2015 23rd European Signal Processing Conference (EUSIPCO)*, pp. 2072–2076, 2015.

[42] X. Li, D. Yan, L. Dong, and R. Wang, "Anti-forensics of audio source identification using generative adversarial network," *IEEE Access*, vol. 7, pp. 184332–184339, 2019.

[43] C. Chen, X. Zhao, and M. C. Stamm, "Mislgan: An anti-forensic camera model falsification framework using a generative adversarial

network," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, pp. 535–539, 2018.

[44] Y. Luo, H. Zi, Q. Zhang, and X. Kang, "Anti-forensics of jpeg compression using generative adversarial networks," in *2018 26th European Signal Processing Conference (EUSIPCO)*, pp. 952–956, 2018.

[45] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[46] A. C. Popescu and H. Farid, "Exposing digital forgeries by detecting traces of resampling," *IEEE Trans. Signal Process.*, vol. 53, no. 2-2, pp. 758–767, 2005.

[47] "Scipy butterworth filter." `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html`. Accessed: 2021-02-27.

[48] W. K. PRATT, "Median filtering," *Semiannual Report, Univ. of Southern California*, 1975.

[49] T. Liu, M. Chen, M. Zhou, S. S. Du, E. Zhou, and T. Zhao, "Towards understanding the importance of shortcut connections in residual networks," *CoRR*, vol. abs/1909.04653, 2019.

[50] J. Uspensky, *Introduction to Mathematical Probability.* No. v. 10 in Ballard CREOL collection, McGraw-Hill book Company, Incorporated, 1937.

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

[52] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, pp. 293–302, 2002.

[53] J. J. Bosch, J. Janer, F. Fuhrmann, and P. Herrera, "A comparison of sound segregation techniques for predominant instrument recognition in musical audio signals," in *Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR*

2012, Mosteiro S.Bento Da Vitória, Porto, Portugal, October 8-12, 2012 (F. Gouyon, P. Herrera, L. G. Martins, and M. Müller, eds.), pp. 559–564, FEUP Edições, 2012.