POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Convolutional Neural Network and Source Separation for bio-signals recognition and classification

Relatori
prof. Gabriella Olmo
firma dei relatori

.....

Candidato
Federico Barbiero
firma del candidato

.....

Anno Accademico 2019-2020

Summary

This is a study of a classification algorithm applied to bio-signals. Signals have been recorded from tomato plants by a multi-channel device, the aim is to be able to distinguish stress moments.
Vivent is company working on bio-signals and it developed a recorder that detects bio-responses of plant. It is proved that plants electrical activity is fundamental in regulating physiological processes and bio-defenses and this gets modified when the organism is subjected to stress situations. The idea is being able to detect the plants' status by inspecting bio-electrical activity, transforming signals into images and by using convolutional neural networks (CNN) to classify them.
At first to enhance classification performances and to better understand available data a study on source separation has been made. Independent Component Analysis is a statistical tool that allows to separate sources from an observed mixture, collected data have been processed in order to extract the essential structure of the data which should correspond to some physical causes that were involved in the process.
Later signals have been converted into images: the ratio is that CNNs work really well on images and, once trained, these networks are quick to compute outputs. Two different imaging techniques have been exploited: one using signals spectrograms and one combining Gramian Angular Fields and at Markov Transition Field. Images have been distinguished among stressed and relaxed moments and labeled.
At the end a CNN model has been trained, tested and classification performances have been measured in order to evaluate source separation and imaging technique effectiveness.

Acknowledgements

I would like to thank Vivent for letting me use their data and more in general for letting me be part of their project for this short period. Thanks to iPrint center for hosting me: it was a pleasure being one of you and thank for being so kind to me.
Vorrei ringraziare la prof. Olmo per la gentilezza e per la disponibilità, un grazie speciale va dato al prof. Marco Mazza per tutti i consigli, le idee e per il supporto ricevuto in questi mesi. La sua simpatia, la sua disponibilità e il suo entusiasmo sono stati fondamentali per portare a termine questo lavoro.
Forse si nota dalle prime righe ma non sono molto bravo a scrivere i ringraziamenti, non è nemmeno una cosa che amo particolarmente fare. Mi sembra sempre di scrivere frasi inflazionate e mielose, in più non credo di avere abbastanza talento comunicativo per riuscire ad esprimere efficacemente l'importanza che avete avuto voi in questi anni.
Sembrerà inelegante e un po' sbrigativo ma inizio subito con un GRAZIE che va a tutti, d'ora in poi consideratelo come implicito in ogni frase così evito di ripetermi; esistono troppo pochi sinonimi per una parola così semplice. Ho pensato di sfruttare le conoscenze maturate in questi cinque anni e mezzo tra numeri e formule per esprimere la ragione per cui vi sono grato associandovi a un'equazione che penso possa riassumere il motivo della mia gratitudine.
Agli amici "*torinesi*" collego lo sviluppo in serie di Fourier: $f(t) = \frac{a_0}{2} + \sum_{n=1}^{N}[a_n \cdot cos(nt) + b_n \cdot sin(nt)]$. Come ogni funzione periodica questi felici anni dell'università sono passati tra una ciclica alternanza di gioie, delusioni, momenti di allegria e di sconforto, orgoglio e sconsolazione. Se scomponessi questa esperienza vedrei che le componenti fondamentali siete stati voi.
Jean, Victor, Pietro e Dario voi siete le equazioni di Maxwell: $\nabla \cdot E = \frac{\rho}{\varepsilon}$, $\nabla \cdot B = 0$, $\nabla \times E = -\frac{\partial B}{\partial t}$, $\nabla \times B = \mu \cdot J + \mu\varepsilon\frac{\partial E}{\partial t}$. Innanzitutto perché siete 4 e almeno due di voi lavorano nel "settore", poi perché siamo stati bene sotto lo stesso tetto come questo sistema sta bene sotto lo stesso nome, anche se le equazioni hanno origini diverse. Ma soprattutto perché tra le 14 persone con cui ho condiviso casa voi siete stati quelli più duraturi e quelli che nei miei ricordi saranno sempre "*casa Colombo*". Queste equazioni descrivono come il campo elettrico accompagni il campo magnetico e viceversa, di come si influenzano e di come evolvano insieme; proprio come voi che siete sempre stati presenti nelle mie avventure e che avete contaminato la miei interessi e le mie idee.
Edoardo, Andrea e Lorenzo non ricordo nemmeno la vita prima della vostra amicizia. Negli ultimi anni sempre più kilometri ci separano ma questo non è sembrato essere un problema. È facile accumunarvi con la forza di gravità per come vincola le stelle tra loro

nonostante la lontananza: $F = -G\frac{m \cdot M}{r^2}$. Come ben sapete l'intensità della forza decresce con l'aumentare delle distanze ma se con voi questo non sembra accadere deve essere dovuto alla vostra massa o alla costante che ci lega, ben più grande di G.
Sofia, sei all'estero con mille progetti e sogni pronti per essere realizzati nonostante tutto quello che è successo e dopo tutto quello che hai passato: significa che in te ci deve essere una forza speciale. Se ci pensi quanto scritto non è altro che una rilettura della seconda legge della dinamica di Newton: $F = ma$. Mi piacerebbe avere metà della tua incredibile determinazione e della tua volontà, continua a farmi da esempio.
A mio papà Nicola, che non vede l'ora di vedere quello che comincia una volta finita questa prova (tranquillo, il CV non l'ho ancora aggiornato), associo il teorema di Bayes: $P(A|B) = \frac{P(B|A)P(B)}{P(A)}$. La formula lega la probabilità di un evento rispetto a una causa nota. Ecco sicuramente il mio successo è condizionato a tutti gli sforzi, il supporto e il fatto che non hai mai fatto mancare niente per farmi arrivare qui.
Infine mamma Luisa spero tu apprezzi il secondo principio della termodinamica: "*In un sistema fisico chiuso l'entropia è una funzione non decrescente con il tempo*", $\Delta S \geq 0$. Il risvolto che più mi piace di questo principio è che un sistema fisico chiuso è destinato a morire termicamente, per mantenerlo "*vivo*" è necessario continuare ad immettere calore e energia. Lasciando perdere metafore drammatiche, è soprattutto grazie alla tua continua profusione di fatiche, di sforzi e alla tua tenacia se oggi possiamo festeggiare questo successo. Se questo percorso, non proprio lineare e facile, è giunto alla fine in gran parte è merito tuo.

Contents

If you're not failing every now and again, it's a sign you're not doing anything very innovative.

[Woody Allen]

Part I

Part one

Chapter 1

Problem Description

1.1 Introduction

Vivent is a Swiss-based company working on bio-signals and it developed an autonomous multichannel recorder to detect bio-responses of plant. This device is an electrophysiological sensor that allows recording electric potentials variations: it is composed of an amplifier that measures voltage potential differences between plant tissues and a reference site (Daniel et al.).
Plants' cells, tissues and organs communicate in order to best adapt to environment, signals can be generated by biotic or abiotic stimuli and translated into chemical or electrical messages. It is proved that electrical signals are fundamental in order to regulate physiological processes such as growth, gas exchange, respiration, transpiration and that bio-electrical activity is modified in response to stress conditions or biological cycles. The idea is to determine the plants' status by inspecting bio-electrical activity (Daniel et al.). The autonomous multichannel device has been used to record signals from tomatoes in soilless culture in growing conditions similar to those used by commercial growers and Vivent aims to determine the plant status by using supervised machine learning.

1.2 Data

Plants have been monitored for days (see fig.1.1) and results have been stored in a database which consists of approximately 1TB of data coming from different experiments. During experiments plants have been exposed to singular stress condition: water and/or nutrient deficit, climate shocks or infestation. This work will focus just on data collected during a parasite infestation (Spider Mites, family: Tetranychidae) but the procedure can be generalized to other type of experiment.

The recording device uses eight differential pairs whose output is recorded on eight different channels labeled (from channel0 to channel7) at an initial sampling frequency of $f_s = 16kHz$. Each channel consists in a pair of two prongs: one inserted into the plant, the other into soil (reference ground point). After acquiring samples, every 32

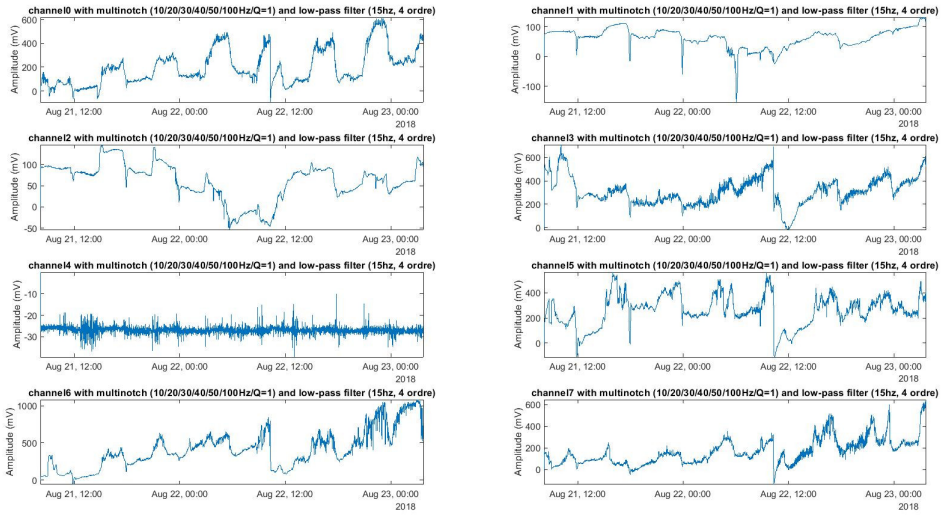

Figure 1.1. Example of the 8 channels recording during a long time window.

measures were averaged and so the resulting sampling frequency is $f_s = 500Hz$. This relative high sampling frequency applied to weeks-long signals leads to high dimensional data. 8-channels recordings can reach easily gigabyte size and the length of this signals can cause dimensionality and efficiency problems.
In addition to dimensional problem signal are characterized by noise:

- $50Hz$ noise: supply line frequency effects data quality.
- Spikes: A software problem during data acquisition can occur: latency can cause a call to software interruption routine and the result is the reading and saving of an abnormal value. It is fundamental in the processing phase to remove this noise component, luckily their amplitude is much higher than the normal signals values making them easy to find (see fig. 1.2).
- Random Noise: Even if in controlled environment, plants are complicated biological systems intrinsically affected by noise and random fluctuations.

To reduce the effects of $50Hz$ noise a software filter is applied to input data. To reduce or delete spikes, signals have been filtered using a median filter (see fig. 1.2).

1.3 Previous Works and Results

Signals classification is an important branch of machine learning and artificial intelligence: a lot of works have been done in recognition of time series, audio signals or biological behaviors, luckily, many tools are available and easy to implement. Vivent idea was to

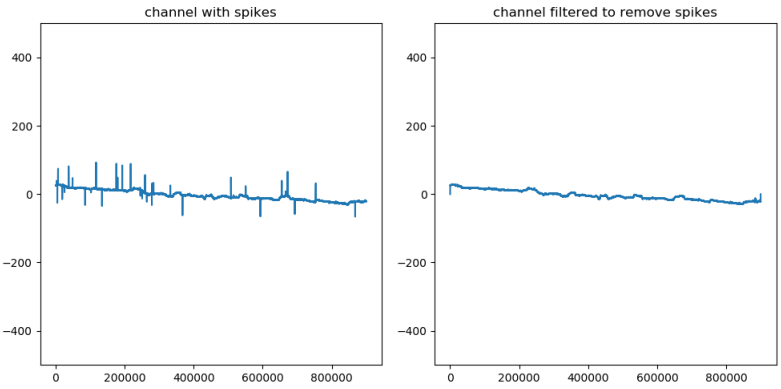

Figure 1.2. It is clear that applying a median filter remove almost all the spikes preserving signal characteristics.

convert the 8-channels recordings into images, label the stressed/infested data and then build a Convolutional Neural Network (CNN) to classify object. The ratio behind this choice is to rely on the high performances that CNN can reach at image classification.
In order to preserve as much information as possible signals have been converted into spectrograms: a spectrogram is the representation of signal frequency spectrum (Fourier transformation) and its evolution along time, an example is proposed in fig. 1.3). The y-axis represents signal frequencies (from 0 up to $250Hz$, maximum readable frequency), the intensity of the component is described by pixels' darkness (the darker is pixel the higher is the intensity of the relative frequency component). To esteem Fourier, transform smaller windows along the sample have been used, at the end by reading the image from left to right it is possible to understand the spectrum evolution along time.

Data related to infestation and relaxed moments have been labeled, collected in a dataset and used to train a CNN. Unfortunately, this kind of model seems not to have reliable and accurate results.
The first idea to improve performances is to separate noisy components and interesting behaviors. $50Hz$ noise is a common component of each channel, it forms a data bias and remove it may help, in addition a better inspection of data space may lead to a deeper understanding of the problem and to a detection of helpful information.

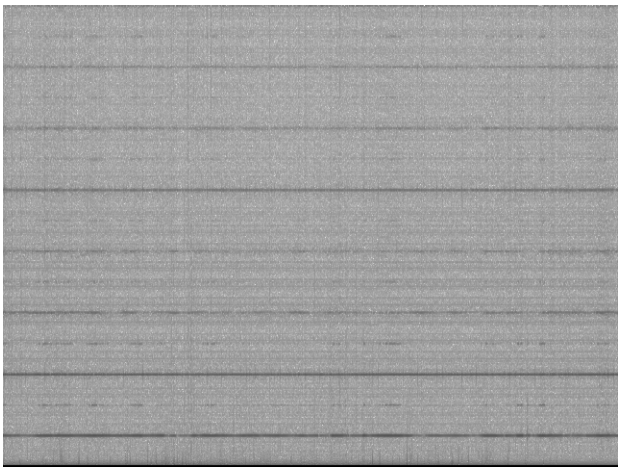

Figure 1.3. Example of single channel observation of 30 minutes converted into a spectrogram. These are gray-scale images because it is easier to convert them into 2D matrices while preprocessing them for classification tasks. The dark visible lines are related to the $0Hz$ components and to $50Hz$ noise and its relative multiples.

Chapter 2

Source Separation

2.1 Cocktail Party Problem

Source separation consists of processing an observed mixture of signals in order to extract the elementary components. To better describe the problem, it is often used a conceptual experiment called *"Cocktail Party Problem"*: it consists in distinguishing different conversations in a noisy room by using some microphones. Each recorder receives a linear mixture of different conversations and by processing these combinations it is possible to get the original sources (see fig. 2.1).

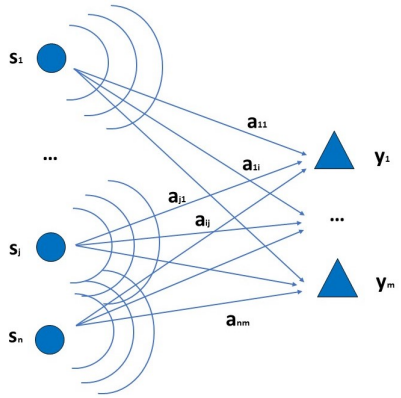

Figure 2.1. This is an easy way to visualize the cocktail party problem: there are n sources (s_i) and m observed signal (y_j) that can be seen as microphones.

The easiest way to model this problem is to consider n sources (the conversations) and m observed signals (the microphones), naturally each source reaches each microphone summed up to the other into a mixture that it is possible to model through a mixing matrix A (equation 2.1).

$$y_i(t) = \sum_{j=1}^{n} a_{ij} \cdot s_j(t) \qquad i = 1, ..., m$$
$$y(t) = A \cdot s(t) \qquad y \in \mathbb{R}^m,\ s \in \mathbb{R}^n,\ A \in \mathbb{R}^{m \times n}$$

Some assumptions need to be made:

- Sources need to be statistically independent: meaning that signals arise from different physical sources.
- Signals are stationary: they do not change in short periods.
- Mixing is considered linear and instantaneous: A is a $n \times m$ matrix and the operation is a dot product and not a convolution.
- Matrix is not time dependent: $a_{ij}(t) = a_{ij} \forall t$. The coefficients do not change along time.

The goal is to esteem an un-mixing matrix $W \in \mathbb{R}^{n \times m}$ that, once applied to the observed signal $y(t)$, is able to reconstruct the sources $s(t)$ (eq 2.1).

$$s = W \cdot y \qquad W \in \mathbb{R}^{n \times m} \tag{2.1}$$

In order to be able to find a solution the system must satisfy some conditions:

- A must not be singular.
- $m \geq n$, the number of observed signals must be greater or equal to the number of sources. If $m < n$ the problem is under-determined and not solvable.

2.2 First Approach

The first tried approach has been proposed by Christian and Herault [1988]. Their solution consists in building a recurrent neural network that minimizes the mean squared error of the observed signals (see fig 2.2).

As $y(t) = A \cdot s(t)$, it is possible to write equation 2.2, where C is a matrix applied to observed signal which is updated at each iteration.

$$y(t) = A \cdot s(t) - Cy(t) \tag{2.2}$$

$$y(t) = (\mathbb{I} + C)^{-1} \cdot A \cdot s \tag{2.3}$$

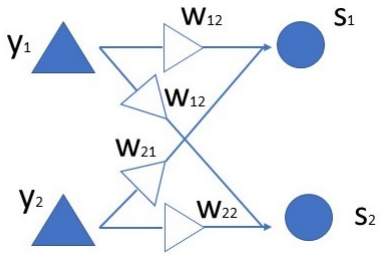

Figure 2.2. Network scheme: the problem consists in finding W's weights that applied to observed signals returns separated sources.

$(\mathbb{I}+C)^{-1}$ is the un-mixing matrix W. Minimizing the error and reaching a the optimal solution is possible by deriving the coefficients.

$$\frac{\partial c_{ij}}{\partial t} = a \cdot f(y_i(t)) \cdot g(y_j(t)) \tag{2.4}$$

From equation 2.4 discretizing and combining things together it is possible to get an iterative algorithm that at each instant updates matrix weights and is able to distinguish sources.

$$y^{t+1} = (\mathbb{I} + C^t)^{-1} y^t \tag{2.5}$$

$$c_{ij}^{t+1} = a \cdot f(y_i^t) \cdot g(y_j^t) \cdot \Delta t \tag{2.6}$$

c_{ij}^t, y_i^t,y_j^t are matrix weights and observed signals at t-iteration, a is a learning rate parameter, f and g are non-linear functions which guarantees no-symmetry (otherwise the result would be C symmetric and this may not correspond to the truth, most used functions are $tanh(x)$ or x^3).

Even if this algorithm seems not difficult to understand and easy to implement it was not used because it has some criticisms:

- At every iteration a matrix inversion is done, this is a computational costly operation that does not permit its application on long signal data.
- By a first and raw study on this algorithm seems not to well separate sources with big frequencies differences (signal must have same order of magnitude).
- Part of data are used to train model and to update weights loosing their information.
- A important tuning phase must be made in order to guarantee convergence: a, f, g must be accurately chosen.
- The number of sources must be equal to the number of observed signals ($m = n$). It is possible to face a problem where $m \geq n$ but m must be a priori known.

2.3 Independent Component Analysis

Independent Component Analysis (ICA) is a different approach to source separation. Problem is treated in a statistical way; signals are not seen as time-dependent measures but as random variables with T realizations along time. ICA defines a generative model where sources are assumed non-Gaussian and independent, these should underline informative details and correspond to some physical phenomena(Aapo Hyv¨arinen and Oja [2001]).
Starting from 2.1 it is possible to determine the matrix W by analyzing some statistical properties of the mixed components y_i. A solution strategy is considering statistical independence and it will be shown that it is possible to determine coefficients w_{ij} just by making this assumption.
The key idea is to take advantage of central limit theorem which stands that a sum of non-Gaussian random variables is closer to a Gaussian distribution than original ones. Consider a linear combination $x_j = \sum_{i=1} w_i \cdot y_i$ of the observed mixture (which is itself a linear combination of the independent components), this will be maximally non-Gaussian if it is equal to one of the independent components (so Gaussianity of $x_j = \sum_{i=1} w_i \cdot y_i$ is the same of eq. 2.1). At the end result will not be the exact value of W (there is no a priori knowledge of matrix A) but it will be a good estimator approximation.

2.3.1 Data Whitening

Independence is the key in order to determine W: the value of any components must not give any information on the values of other components. Independence is a much stronger request than uncorrelatedness: *correlation* $\rightarrow$ *dependence* but it is not always valid that *correlation* $\leftarrow$ *dependence*, in fact it is possible to find dependent and uncorrelated random variables (see fig. 2.3). Uncorrelatedness in itself is not enough to separate the components (Aapo Hyv¨arinen and Oja [2001]).

By using decorrelation methods, it is possible to transform any linear mixture of independent components into uncorrelated ones. ICA consists in estimating the orthogonal transformation after decorrelation by locating square's edges in right figure 2.3 and computing the matrix rotation that gives the original components.
It is convenient to deal with whitened or sphered mixture vectors: a zero-mean random vector $z = (z_1, ..., z_n)$ is said to be white if its elements z_i are uncorrelated and have unit variances (Aapo Hyv¨arinen and Oja [2001]): $E[z_i \cdot z_j] = \delta_{ij}$ and $\Sigma = \mathbb{I}$ (whiteness is a stronger property than uncorrelatedness). Whitening can always be done with a linear operation: it is sufficient to find a linear transformation V that transform a given a random vector y with m elements into another vector z such that $z = V \cdot x$ is white (or sphered).
One popular method for whitening is to use the eigenvalue decomposition of the covariance matrix $\Sigma = EDE^T$ where $E = (e_1, ..., e_n)$ is the matrix whose columns are the unit norm eigenvectors and $D = diag(d_1, ..., d_n)$ is the diagonal matrix of the eigenvalues of Σ. Then a linear whitening transform is given by $V = ED^{-1/2} \cdot E^T$, it can be proved that this transformation satisfies the whitening conditions (Aapo Hyv¨arinen and Oja [2001])

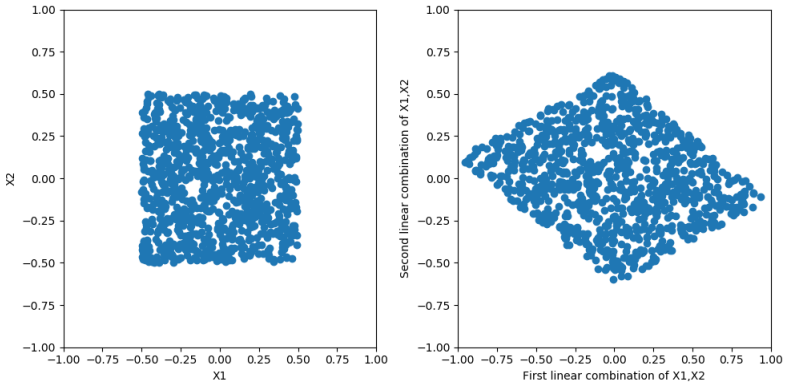

Figure 2.3. Left picture shows two independent components (X_1 and X_2) with uniform distributions. In the right plot are shown two uncorrelated mixtures of the same independent components (applying a mixing matrix the result is a rotation of the plane). Even if mixtures are uncorrelated, components are not independent: if the one component has value near the corner on the horizontal axis this clearly an information that can be used to infer the possible values that the components has on the vertical axis.

2.3.2 Gaussianity Measure

To create the generative model, it is necessary to find a way to measure Gaussianity. A possible choice can be kurtosis which is the fourth-order cumulant of a random variable.

$$kurt(y) = E[y^4] - 3E[y^2]^2 \tag{2.7}$$

Kurtosis can be both positive or negative and zero for a Gaussian variable (there are non-Gaussian random variables that have zero kurtosis but they are rare), non-Gaussianity is measured by the absolute value of kurtosis and it can be used as object function in a maximization optimization problem.
It is possible to design a working algorithm from kurtosis but there are also some drawbacks: it can be very sensitive to outliers and other measures might better perform.
Another possible measure choice is negentropy. The entropy H of a random vector y with density $p_y(\varepsilon)$ is:

$$H(y) = -\int p_y(\varepsilon) \cdot log(p_y(\varepsilon))d\varepsilon \tag{2.8}$$

The idea of using entropy is related to a fundamental result of information theory saying that Gaussian variables have the largest entropy among all random variables of equal variance (Aapo Hyv¨arinen and Oja [2001]). To build an optimization problem the entropy measure must be zero for Gaussian variables and always non-negative. It is possible to write and define negentropy J as:

$$J(y) = H(y_{gauss}) - H(y) \tag{2.9}$$

Where y_{gauss} is a Gaussian random variable with the same correlation (and covariance) matrix as y.

Computing random variables' distributions can be a long and expensive task, the best way to face the problem is to use a negentropy approximation. The classic method consists in writing J as:

$$J(y) \approx [E[G(y)] - E[G(y_{gauss})]]^2 \tag{2.10}$$

Where G is a non-quadratic function, by choosing it a G that does not grow too fast, it is possible to obtain more robust estimators. It is demonstrated that good functions are $G = log(cosh(y))$ and $G = -e^{-\frac{y^2}{2}}$.
Equation 2.10 can be used as objective function to find the local maximum of non-Gaussianity under the constraint of constant y variance. Each local maximum gives one independent component. Non-Gaussianity in the n-dimensional space has $2n$-local maxima (two for each independent component) corresponding to s_i and $-s_i$ (independent components can be estimated only up to a multiplicative sign).

2.3.3 FastICA

Simple gradient algorithm can be derived by maximizing negentropy approximation, deriving respect to W, and normalizing $E[(W^T \cdot z)]^2 = ||W||^2 = 1$. The normalization is necessary to project W the unit sphere to keep the variance of $(W^T \cdot z)$ constant.
A much faster method for maximizing negentropy can be found using a fixed point algorithm called FastICA: it finds a direction (a unit vector w), such that the projection $(W^T \cdot z)$ maximizes non-Gaussianity approximation (eq 2.10). Recall that the variance of $(W^T \cdot z)$ must be constrained to unity, it is equivalent to constraining the norm of $|W| = 1$. All this can be derived as an approximative Newton iteration.

1. Center the data to make its mean zero.
2. Whiten the data.
3. Choose an initial random vector W such that $|W| = 1$.
4. $W \leftarrow E[z\, g(W^T \cdot z)] - E[g'(W^T \cdot z)] \cdot W$, where g is the derivative of the non-quadratic function G used in 2.10.
5. $W \leftarrow \frac{W}{||W||}$
6. if not converged go back to 4.

FastICA combines the good properties of the fixed-point algorithm with the robustness and easy computations of negentropy approximation (Aapo Hyv¨arinen and Oja [2001]):

- Fast convergence: it can be proved that the convergence of this algorithm is cubic.
- There are no parameters to tune: there is no learning rate or other parameters in the algorithm. It easy to use, more robust and reliable.
- Using negentropy approximations instead of kurtosis enhances the statistical properties of the estimator.

The algorithm just presented estimates only one independent component: to get more independent components different kinds of decorrelation must be used.

2.3.4 ICA Problems

Even if ICA and FastICA are reliable, robust, fast and easy solutions to source separation they have some criticisms that must be exploited

- First there is no a unique solution. The optimization problem has $2n$-local maxima, two for each independent component. The result will be one of the possible approximations of the results and by starting from a random vector two runs of the same algorithm may converge to different equivalent solutions.
- When dealing with signals the problem of having more local maxima can mean that the resulting source may be flipped with a phase shift of 180^o.
- Zero averaging and signals whitening is fundamental but it means loosing some information linked to mean value o or to signal amplitude.
- The algorithm does not give any information about the most important component.
- The number of independent components must be a priori known.
- Algorithms need sample data to esteem expected values or cumulants: this makes difficult to implement an on-line or a self-adaptive algorithm.

Part II

Part two

Chapter 3

Algorithm Development

To deal with data and to perform all the following tasks Python 3.7 has been used. Python is an easy object-oriented programming language and it relies on many open source libraries and packages helpful for the scope of this thesis: numpy, scipy, sklearn, matplotlyb, tensorflow and PIL are some of the used packages. Another reason to use Python instead of other languages (like Matlab) is that, later, to train and test the neural networks Google Colab was used. This is a helpful tool based on Python Jupiter Notebook which allows to train models on dedicated, very powerful remote platforms. Training an easy network on a normal laptop would request too long time, on Colab this type of operations takes minutes.
To easily manage and process data (once downloaded and saved on a hard disk) a Python package was written, inside this a Python file is dedicated to recordings opening and management (`dataManeger`). This operation, implemented in a function named `dataOpen()`, can take long because files are really heavy and the system requires sufficient RAM and time to process the request. Described functions' core is the numpy function `np.fromfile()` which allows to open and read binary files.
Another function was implemented (`dataManege()`): it splits given data lists into a given subsection. This is helpful to do not deal with days long recordings but with a requested time-window: it asks the desired starting and ending instants and returns the data relative to that time interval. It is possible to specify if the desired output is as a numpy array (with shape $8 \times T$, where T is time interval length) and if spikes must be removed by applying a median filter (using `scipy.singals.medfilt(data, kernel=3)`, see fig 1.2). Other functions have been written to perform source separations and to plot results and saved in dedicated packages (`FastIcaOnData` and `PlottingFunctions`). The `main()`, using the desired imported functions, splits recordings into sections, performs source separation, plots and saves the results in the given folders.

3.1 Source Separation Algorithm

It is easy to perform source separation using sklearn function `FastICA()`, an own algorithm has not been developed because build-in functions are optimized in order to be robust

and efficient. The requested inputs are: the number of independent components (the only parameter that will be specified), the maximum number of iterations and a tolerance to determine convergence conditions, the option to whiten or not your input data (inputs get whitened by default, see 2.3.1), the non-quadratic function to esteem negentropy (G in 2.10) and a possible weights matrix to initialize the iterative procedure (instead of a random guess). Parameters that are not specified as input are left to their default values. Once the model has been generated, it must be fitted to data which must be presented in a column-vector matrix, so channels recordings must be transposed. The output of `.fit_transform()` are the unmixed sources transposed.

```
from sklearn.decomposition import FastICA


ica = FastICA(n_components=n_components)
reconstructed_signal = ica.fit_transform(data.T)
reconstructed_signal=reconstructed_signal.T
```

The algorithm is really simple and easy to use but it has a problem: the number of independent components must be previously known.

3.1.1 Determining Number of Independent Components

It is fundamental to find a way to esteem the number of Independent Components (ICs) in the system because performing the separation algorithm with erroneous parameters can lead to not satisfying results. The main problem is that there is no objective measure to compute the ideal number of sources in a given mixture, but it is possible to make a reliable esteem based on Ganesh R. and Dinesh, 2009 idea.
Ganesh R. and Dinesh [2009] method consists in inspecting the determinant of the normalized ICA solution matrices (W and A) and use it as a measure to get the number of ICs in a linear model (eq. 2.1) with $m > n$.
The idea carried out on this thesis work is to compute the determinant of the mixing matrix esteem A (obtained by FastICA's method `.mixing_`). If the number of sources is equal to the number of observed signals A will be a square matrix $m \times m$, but if $m > n$ at least one of the columns is linear combination of the other making the determinant $|A| \approx 0$. Then, assuming that a selection of k among m recordings should be able to separate sources (linearity consequence), it is possible to use this selection to esteem $|A|$, if $|A| \approx 1$ (after normalizing A using Frobenius norm) then all columns are linearly independent and their number consists in the number of ICs.
This simple observation can be used to write a python function (`n_ICs()`) that returns an esteem of independent components in the system: it starts from data and compute the mixing matrix using just 2 independent components (assuming that at least one component is present) and get its determinant. If determinant is over a given threshold value (after some experiment on synthetic data a value of 10^{-2} has been set) an independent component is added up and the procedure repeated until the requested is satisfied. At that point the cycle breaks and the number of fitted independent components is given as

output.

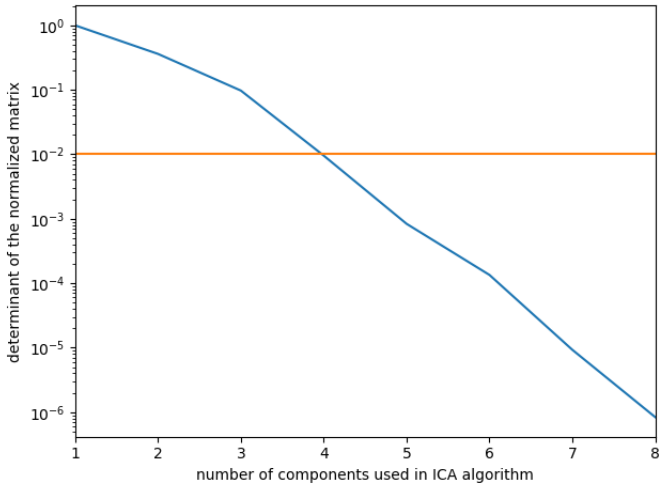

Figure 3.1. To plot this graph FastICA has been run with an increasing number of ICs and the normalized matrix determinant has been computed at each iteration. It can be shown that after few components it drastically decreases. Once passed the threshold (the orange line) the algorithm breaks and returns the number of ICs, in this case it would have been 3.

This is a simple method to write and perform but it has some important drawbacks:

- It slows down computations: for every data time window it computes ICA model several times. Luckily FastICA is really quick and this does not compromise the code execution too much.
- It uses just selection of observation to determine the number of ICs: under the assumption of linearity in an ideal set up this should not be a problem but in a real-world environment some distortion or approximations are introduced.
- It depends on a tolerance value and there is no way to tune it and to evaluate it (a part from making tries on synthetic data).
- There is no way to measure split goodness
- This seems to work under strong assumption that may limit the model robustness.

3.1.2 FastICA performance

By running the described code, signals and sources have been plotted and saved with their relative Fourier transform. Indicators of the algorithm goodness are sources catching

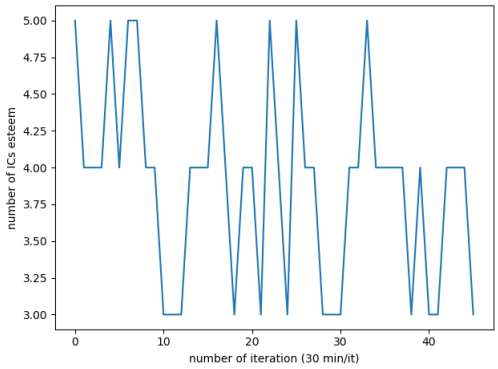

Figure 3.2. A 30 minutes time window has been taken and at every iteration the number of ICs has been estimated. This is the results: they changed from 5 to 3 and by a better looking of the all data it seems that the higher the number of ICs is the more stressed looks the plant.

channels main behaviors and at least one catching the $50Hz$ noise signals.
An example of data on 30 minutes and their relative spectrum are shown (see fig 3.3).

By looking at this data it is difficult to say where information is a part from low frequency components. In addition, it is clear that noise effects data quality. Once computed, sources are plotted and saved (see fig. 3.4)

The illustrated behavior is pretty satisfying: being able to capture a filtered component (50 Hz) and the fact that an interference noise can be detected is a symptom that the algorithm is really sensitive. It is possible to separate sources, detect noisy components and save the meaningful ones for classification (and discard useless data).

3.2 Spectrograms Generation

Once FastICA algorithm has been implemented, the next step is to convert channels and sources signals into images and create the datasets. Used imaging technique was spectrograms-generation, this choice (as previously explained in 1.3) is made because it is easy and immediate to generate these images. In matplotlyb package the function `specgram()` is available: its inputs are data, the sampling frequency, number of points used in each block for the transform, the number of overlap points between blocks, the scale and the image color map, function outputs are: the periodogram (estimate of the spectral density of a signal), the frequency vector, the centers of the time bins and the image object representing the data in the plot (the only required output).
To plot this type of data the inputs were set as:

```
_, _, _, im = plt.specgram(data, NFFT=1024, Fs=500, noverlap=128, scale='
    dB', cmap='binary')
im.show()
```

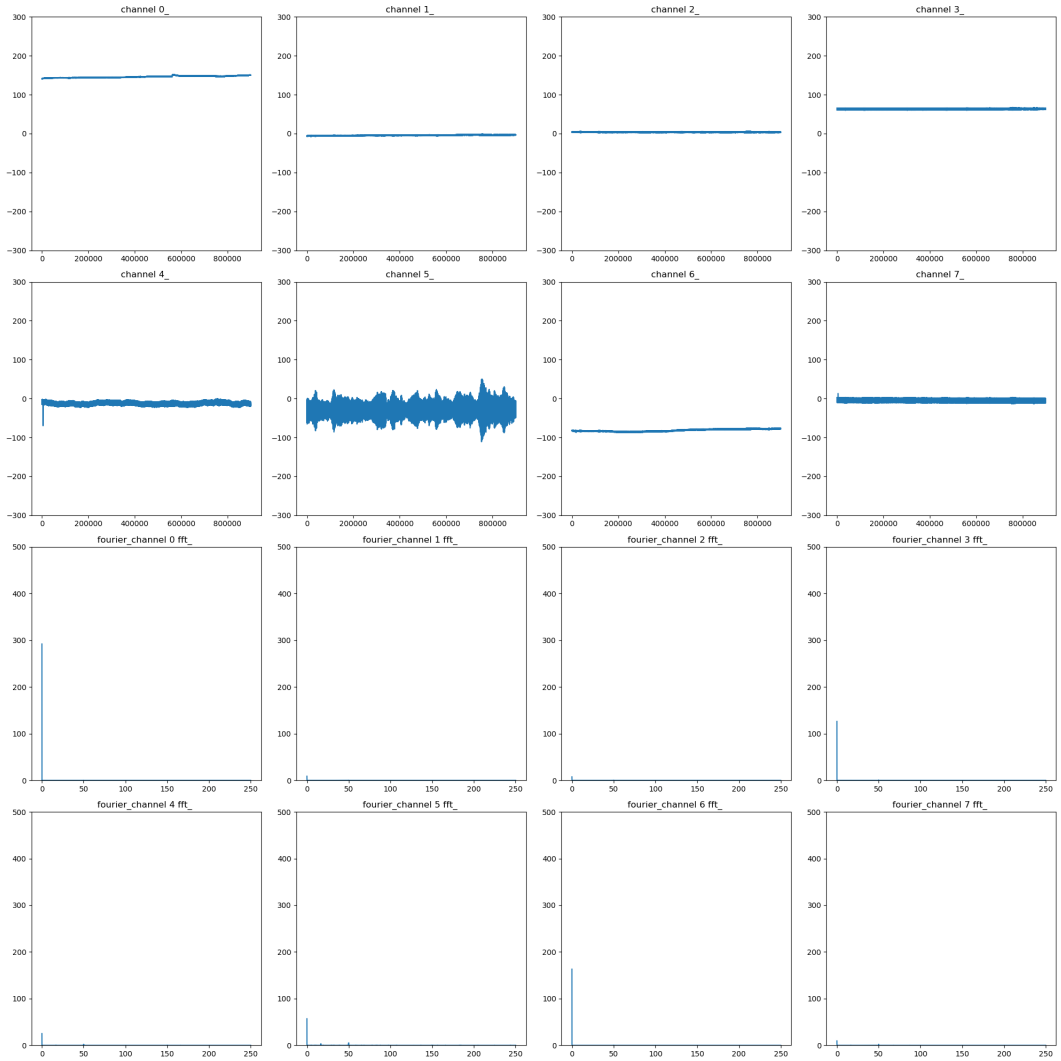

Figure 3.3. Looking at these data is helpful to understand if the plant is infested or not. This looks a relative quiet moment with the 5th channel affected by a kind of undetermined stress.

Code has been run for every channel and source time window and a result example is shown in fig 1.3, all these images were titled using the recording moment and the channel/source number and, at the end, they were saved in the respective folders.

3.3 Time Window Length

By reading Bensoussan et al. [2016] and talking with domain expert it has been confirmed that parasite activity is pretty slow. Parasites damage plants cells during their feeding

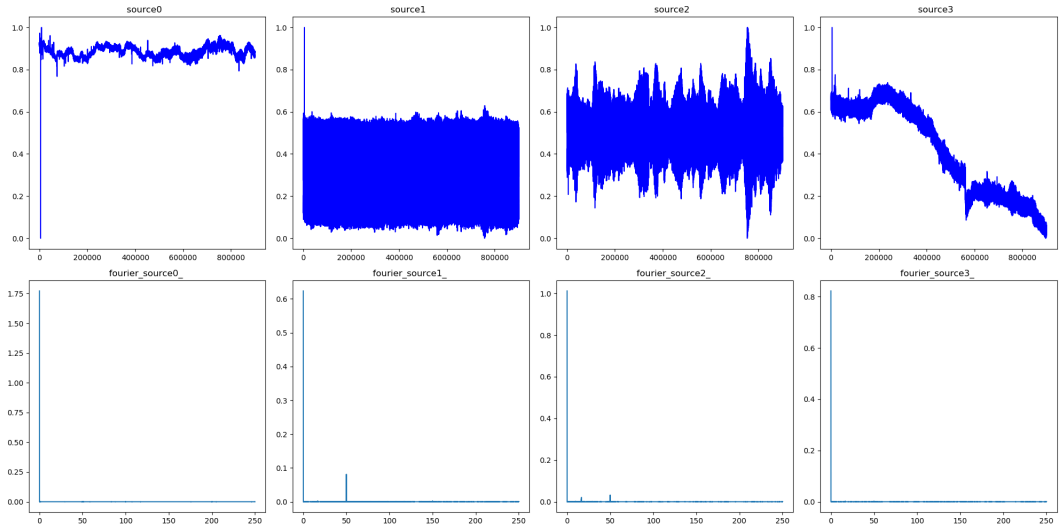

Figure 3.4. 4 independent components were estimated and fitted. Results are interesting because the second and the third sources seem to capture the 50 Hz noise (which has being filtered too) and a small 16.7 Hz signal (maybe due to some interference).

and this may take minutes, while observing data some really particular fluctuations has been noticed: these patterns are frequent when plant is infested by spider mites and rare in other type of experiments. The hypothesis behind this behavior is that parasites suck and inject some molecules that may trigger some comunication cascade inducing chemical signals and voltage variations in the plant. This cascade may activate some defenses responses within the plant. An example of high intensity infestation is shown in figure 3.5.

Sometimes it is clear that an infestation will happen just by noticing few fluctuations that slowly increase their frequency and images tends to look like those in fig. 3.6. This is an average behavior: signals can change amplitude, frequency, shape... due to this variability it is difficult to automatically distinct these patterns without machine learning. To better deal with data and to detect infestation moments a time window of 30 minutes will be used (which consists in building signals vectors with $30 \times 60 \times 500 = 900000$ observations).

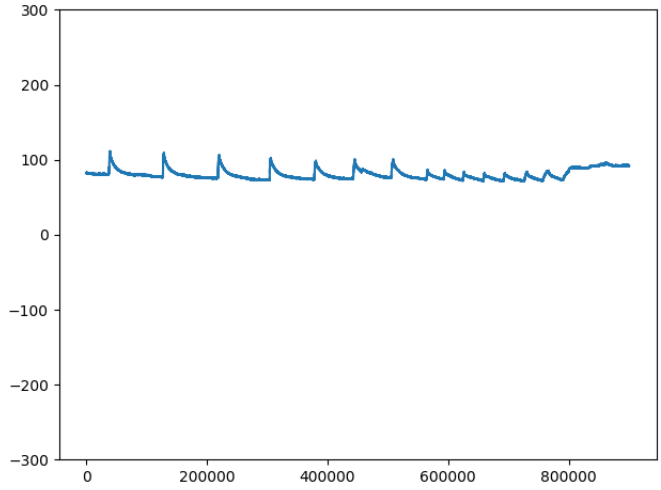

Figure 3.5. This is an example of a channel during an intense infestation moment. In this particular case it is interesting to notice that fluctuations tend to decrease their amplitude and then, at the end, almost disappear.

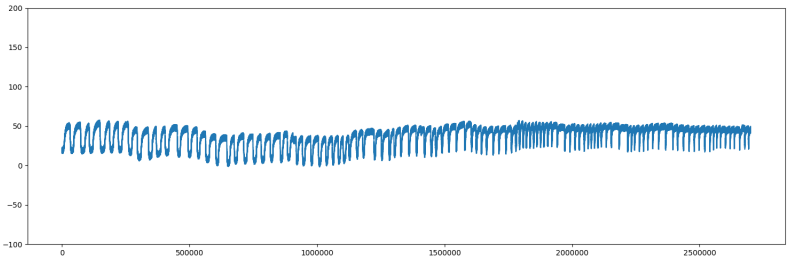

Figure 3.6. This is a longer observation (1 hour and 30 minutes) to show how infestation activity can change along time, increasing the frequency of fluctuations.

Chapter 4

Signal Classification

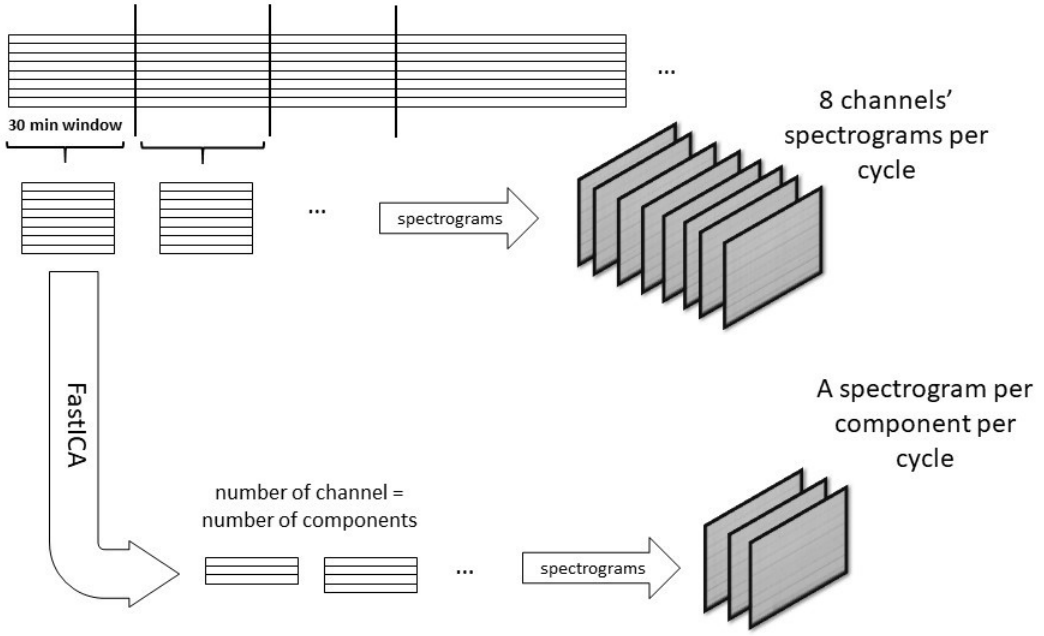

Figure 4.1. This is a schematic representation of the image-generating algorithm. It has to generate and save more than 10 images per iteration and to compute ICs several time so it may take some minutes (around 3-4 cycles per minute) to process a days-long file.

4.1 Labeling Process

Once all images have been generated, stressed and relaxed datasets are created. The idea is to exploit channels, select the stressed moments and among the 8 images, save the spectrograms related to the infestation in a folder named `channels_stressed_specgram`

and repeat the operation with sources images (selecting the same time instant but saved in another folder `sources_stressed_specgram`). The goal is to understand if source separation produces better images by taking away noise and unwanted signals. Obviously, some control data have been collected in order to create a balanced dataset but there was a problem: used control data come from the same observation of spider mites data (just by picking relaxed moments) and there are no guarantees that these do not contain any information about infestation. Data recorded on a completely healthy plant with no stress were not available and even using data from other experiments is not a good idea because plants should manifest other type of signals that can affect the analysis.
Finally, two datasets were created: one containing spectrograms from channels (stressed and relaxed) and one with sources images. Naturally these are related to the same instants but cannot contains the same amount of data (sources images are less). To maintain proportions among label some relaxed moments have been added to sources images.
At this point folders have been uploaded into Google Colab and stressed data have been associated to label 1 and relaxed to 0.

4.2 Neural Network

Artificial neural networks (NNs) are machine learning models inspired by the human's neurons: these models can learn to perform tasks by processing a list of positive and negative examples. The designer goal is to make networks able to solve relative complex problems in a robust way, as a human being would, without directly coding it (usually these are tasks where a traditional programming approach is infeasible). In last years, due to an increase of computing power, it has been easier and more efficient to build these network structures and this brought to an increasing interest and to new applications in many different fields: computer vision, speech recognition, data mining, playing board and video games, self-driven vehicles, medical diagnosis...
A NN is consists in many connected computational units called neurons linked together by edges (the equivalent of brain synapses): every neuron receives an input (a number), processes it (through some functions and methods that will be described later) and transmits it to other units. Each neuron link is associated to a weight that is modified and adjusted during the learning phase. Neurons are grouped into layers and the network depth consists in the number of contained layers. In traditional and common networks neurons are connected to following and preceding neurons (see a network structure example in 4.2). Layers can be various and performs different tasks, most popular examples are: fully connected layer (with every neuron in one layer connecting to all neurons in the next layer), pooling layer (grouping neurons values into one, reducing inputs dimension), convolutional layer (particularly indicated to extract images features by applying convolution filters.
As said, NNs can be used in different tasks that are above classifications (such as forecasting, regression or unsupervised learning) but this project will focus supervised machine learning: in this case the algorithm learns from an available list of object value and the desired output (label) adjusting the weights to improve the accuracy performances and minimizing the observed error measure. The way errors are measured is done by defining

a loss function: if its values keep decreasing learning continues and stop when until convergence.
The way weights are updated is called Backpropagation: it consists in trying to compensate each error during learning. Backpropagation calculates the gradient of the loss function respect to the weights and updates them.

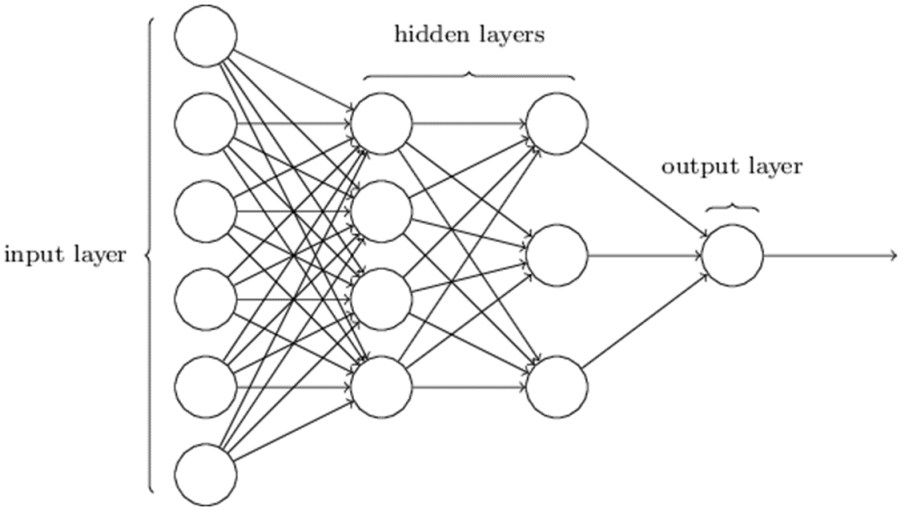

Figure 4.2. Input data are firstly passed through the input layer, then they are processed sequentially by hidden layers and, at the end, the outcome is given by the last output layer. Each layer is characterized by the number of neurons in it (layer size).

4.2.1 Convolutional Neural Networks

A convolutional neural network (CNN) is a machine learning tool that have been shown to have great performance on images processing. These network structures are similar to fully connected neural networks (fig. 4.2) that, instead of normal matrix operations, uses convolution: the network hidden layers convolve data as shown in fig. 4.3.

At classifying images CNNs perform better than fully connected neural networks because they can handle more easily tensor-like-data (RGB images are 3D matrices, the previously computed spectrograms are 2D) requiring a smaller number of neurons and parameters (and so, with the same input size, less computational power and memory).
Usually CNN structure consists in some convolutional layers that extract features followed to few fully connect layers to map data into label space and to perform classification (see fig. 4.4)

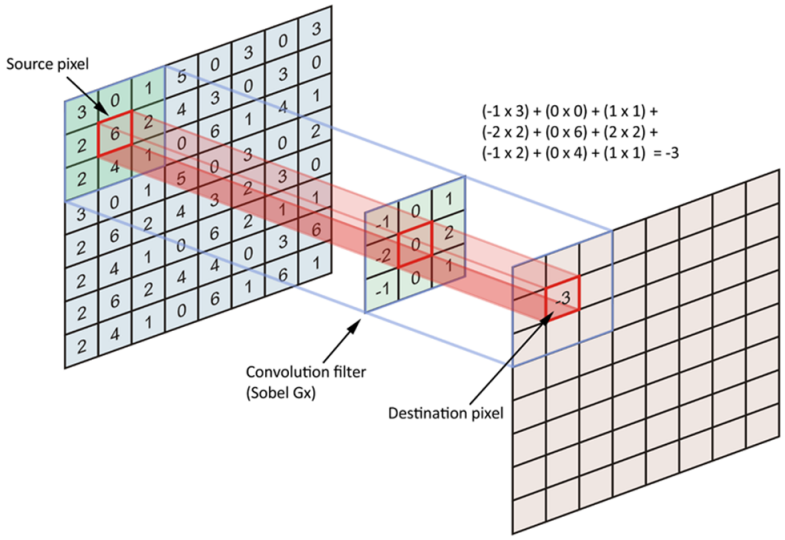

Figure 4.3. Data are passed through a sliding window that combining its values with the data ones produces the output, convolutional layers are characterized by their filter mask (or kernel).

4.2.2 Convolutional Layer

As already said, CNNs rely on convolutional layer whose parameters are of a set of small filters (kernels), passed on the full input data shape. At every iteration each filter processes data as shown in fig. 4.3, by adjusting filters' weights. The result is that network learns how to efficiently filter the images by detecting that some features proper of the dataset.
In this way input images are transformed to abstract feature maps and passed to following layers, miming the brain behavior. This technique resolves some problems related to gradient computing when training traditional neural networks with many layers by using backpropagation.

4.2.3 Pooling Layer

Pooling is a way to reduce data size with a sort of non-linear down-sampling: there are several functions to implement this (max pooling is the most used one see figure 4.5), but they are all translation-invariant.

Pooling layers are useful to progressively reduce data size, to reduce the number of parameters, and so to reduce the chances of overfitting. Polling layers let the algorithm use less memory and computational power, a common way to use these tools is to insert them between successive convolutional layers.
Pooling units can use many functions or metrics (average pooling, l_2-norm, ...).

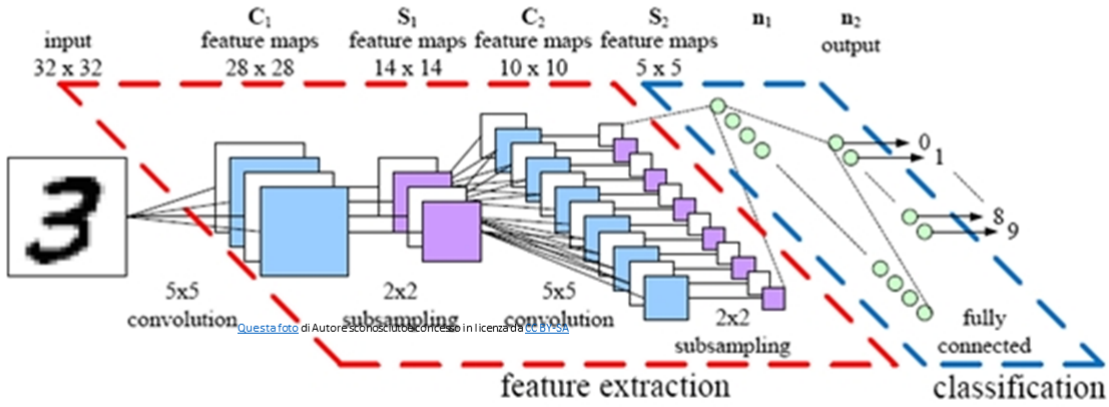

Figure 4.4. This is an example of CNN structure performing hand written digits classification.

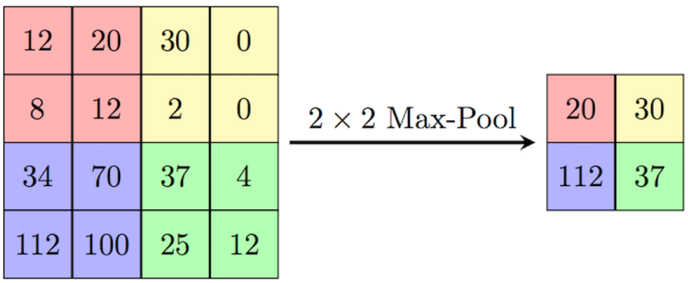

Figure 4.5. Data are split into non-overlapping partitions and, for each, output is the maximum value inside.

4.2.4 Activation Function

The activation function is the neurons characteristic: every input is processed and the output is obtained. A neuron can be seen as *on* or *off* depending on the activation functions value computed from the input. This is similar to the behavior neurons and axons in human brain. Particularly non-linear activation functions allow networks to compute solutions using a smaller number of nodes.
The simplest activation function is the Heaviside one: neurons are on if inputs are above a threshold otherwise off. Other type of more sophisticated functions can model better neural behaviors, the most popular is ReLU (Rectified Linear Unit), which is defined as $f(x) = max(0, x)$ and it is shown in fig. 4.6, ReLU is particularly popular because it is easy to compute, removes negative values and increases the nonlinear properties of the overall decision function.

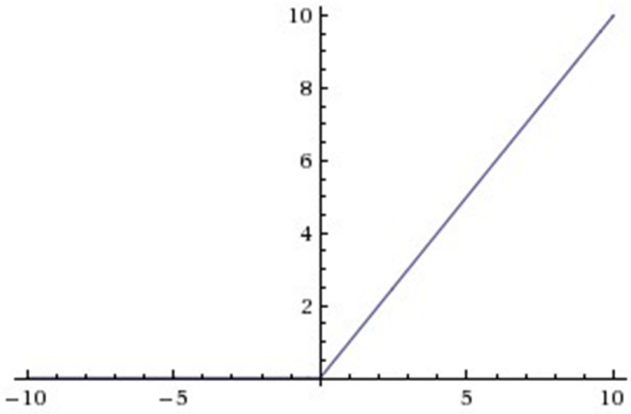

Figure 4.6. ReLU function graph.

Other functions are widely used such as $f(x) = tanh(x)$ or the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

4.2.5 Regularization

Neural Networks are models with a huge number of parameters and a lot of non-linearities are highly complex models which tends to overfit data. This effect can be attenuated by using regularization. It is possible to reduce the number of errors and increase algorithm robustness by using some techniques such as l_1 or l_2 regularization: these tries to limit the capacity by adding norm penalty parameter $\Omega(w)$ to the model (where w indicates weights vector). Now to solve the optimization problem algorithm must minimize the original loss function with the addition of regularization term. Adding this constraint leads to have a better control on weights growth (or decrease) and model complexity is reduced.

4.2.6 Dropout

Another technique to reduce overfitting is dropout. It consists in dropping out some nodes and their edges during training in order to reduced network size. The removed nodes are then reinserted into the network with their original weights. While designing a dropout stage it is possible to set the probability that a node will be dropped.
Dropout seems to reduce node interactions and leads to a more robust and efficient learning; model better generalizes when used on new unseen data and training speed is improved too.

4.3 Model

4.3.1 Preprocessing

Images folders from sources and channels have been uploaded into Colab. Datasets have similar sizes: 827 images for sources one and 690 for channel one (remember that some images have been added to `sources_no_stressed_specgram` in order to maintain the same label distribution). Images have been converted into numpy arrays and label vector was created, dataset label distributions are around 50%, naturally in the observed data this is not the real proportion but at first it is important to see if the algorithm is able to distinguish between stressed and relaxed behavior.
CNN was implemented with `tensorflow` and particularly using `keras`. It is extremely easy to build and fit neural networks using this package, networks inputs must 4D normalized tensors (*number of samples* $\times$ *image size*, remember that images are 3D matrix with values from 0 to 255) so the 2D spectrograms are reshaped in order to appear 3D matrices with just one depth level. In this operation pixels have been divided by 255 in order to map them in [0,1] range.
At this point data were shuffled and split into training and test sets (with test size proportion of 0.25).

4.3.2 Batch

Before starting to putting things together into a model few other parameters must be exploited.
The first one is batch size: it defines the number of observations that will be propagated through the network at each epoch (see 4.3.3). Algorithm may require a lot of memory and, to reduce this request, instead of passing all data a small selection is used: algorithm firstly takes the first selected number of samples and train the network. Then, it takes the second selection and repeat what just said. This procedure is iterated until all samples have been passed through of the network.
Another important advantage of using smaller batches is that it speeds up computations. The paid price to use less memory and for a quicker training is less accuracy when computing loss function gradient respect to weights.

4.3.3 Epochs

Another important hyperparameter is the number of epochs: it consists in number of iterations in which the network will learn from the dataset. At every epoch each batch of samples is passed through the net. The number of epochs is traditionally large in order to let the network to learn until loss has been sufficiently minimized. A good way to evaluate learning process is looking at learning curves: these graphs exploit loss and accuracy evolution along the epochs (see 4.7). A good behavior is characterized by a loss decrease and a respective accuracy metrics increase until they stabilize around some convergence values. These plots are really helpful to inspect and to understand if the model learned enough or if it overfitted data (usually when training accuracy is high but validation is much lower).

4.3.4 Optimizer

Optimizer is the chosen algorithms that minimizes the objective function (the model loss). The neural network weights are the parameters that define the learning and they must be updated in order to make outputs decision rule as much coincidence as possible with input rules.
The easiest way updating rule is Gradient Descent: it consists in computing loss gradient respect to w and update $w = w + \eta\ J(w)$ where J is the loss gradient respect to w and η is a learning rate. This technique calculates the gradient of the whole training dataset just to perform one update, it can be very slow and hard to use it with large datasets. Using this optimizer guarantees the convergence to a local minimum (global minimum if error surfaces is convex).
Another widely used optimizer is Stochastic Gradient Descent (SGD) which is faster than normal gradient descent. Updating rule is similar as before ($w = w + \eta\ J(w, x_i, y_i)$ where x_i and y_i are the examples of training data and label) but parameters updates have higher variance and this causes more fluctuations of loss function. This effect can easily be seen in learning curves and it is symptom that the SGD helps to better explore the error surface lowering the risk of getting stacked in local minimum. The drawback is that SGD has more difficulties to converge even if it has been shown that using a slowly decrease the learning rate (η), helps.
Another way to boost SGD performances is to make use of Mini Batch Gradient Descent: it performs an update for every batch (4.3.2), this should reduce the variance of updates leading to a better and more stable convergence. Nowadays mini-batch gradient descent is one of the most popular choice to training a neural network and it is used by default when SGD is requested.
To overcome the difficulties that SGD has to converge Momentum technique was proposed: it accelerates SGD by exploring the error surface along relevant direction and softening irrelevant ones. It adds a parameter γ and learning rule becomes $w = w - V(t)$, with $V(t) = \gamma\ V(t-1) + \eta\ \nabla J(w)$. γ represent the fraction of the past step update vector in the the current vector, usually set to 0.9. The momentum term increases when moving through increasing gradient direction and reduces in the opposite case and this led to faster and more stable convergence.

There are many other more sophisticated algorithms to solve this optimization problems and mostly they rely on different way to updating rule and compute momentum. The aim is always to have a better exploration of the solution space and to reduce the risk of getting stacked in local minima. All have specific pros and cons: in later network design and tuning Adam optimizer will be used. In fact, it works well, it is efficient and it is able to fast converge at good solutions.

4.3.5 Model Structure

Dataset images coming from sources or channel data are really similar so it is possible to take advantage of this (and of their label distribution both close to 50%) to train and test similar models. Based on previous works first model has been coded: using the following structure:

1. A convolutional layer characterized by 256 filters with kernel size 3×3 and ReLU activation function.
2. A 3×3 max pooling layer (see fig. 4.5).
3. A convolutional layer characterized with 128 filters, kernel size 3×3 and ReLU activation function.
4. A 2×2 max pooling layer.
5. A convolutional layer characterized with 64 filters, kernel size 3×3 and ReLU activation function.
6. A 2×2 max pooling layer.
7. A fully connected layer with 64 neurons using ReLU function (data must be flattened before).
8. A dropout with p=0.3 (see 4.2.6).
9. A final fully connected layer with just 2 neurons (as many as the labels) with softmax activation function. Softmax is defined as $f(x) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 0,1,2,...K$. It can take values just in [0,1] and so it can be used to have a probability measure of the object having the predicted label (in few words: network confidence).

The chosen loss function is sparse the *sparse categorical cross-entropy*: categorical cross-entropy compares the predicted label and true label as in eq. 4.1.

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{4.1}$$

w is the model parameters vector, y_i is the true label and $\hat{y}_i$ is the predicted label. Sparse categorical cross-entropy and simple categorical cross-entropy use the same loss equation and have the same outcomes. The only difference is how they code label (by

one-hot encoded or by integers) and using sparse categorical cross entropy should be faster and require less memory.
From training set a fifth of data are saved in a validation set in order to evaluate algorithm performance on unseen data at each epoch (so it is possible to detect overfitting). At the end the training sets are composed of 496 images for sources folders and of 413 for channel one. Batches and epochs are set to 20.

```
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=(img_shape)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64))
model.add(Dropout(0.3))
model.add(Activation('relu'))

model.add(Dense(2))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])


model.fit(X_train, y_train, epochs=20, batch_size=20, validation_split
    =0.2)
```

Firstly, model was trained using sources data. The results are shown in fig. 4.7. It is clear that results are quite disappointing: a part from the accuracy scores around 50% (that on a balanced dataset is like random guessing) even the loss seems to decrease just at the first iteration and then probably get stacked into a local minimum value. Model has been also tested on a test set (coming fro the first data split, see 4.3.1), even here result is extremely poor: 44%. It is clear that model has learn nothing.

4.3.6 Tuning

To increase the performances a first try was to add convolutional layer (64 filters, kernel size 3×3 and ReLU activation function) has been added between the last polling and the first fully connected layer (between 6 and 7 in model described in 9). Results are shown in

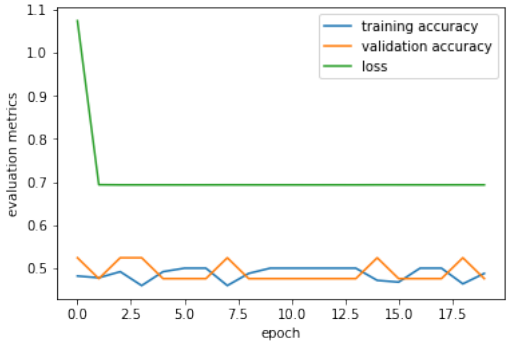

Figure 4.7. This is the first obtained learning curve. At first loss seems to decrease but accuracy scores keep almost constant. This behavior is not the ideal learning curve one, symptom that the network did not learn enough.

figure 4.8, accuracy score on the test set was 51% (probably due to its label distribution change).

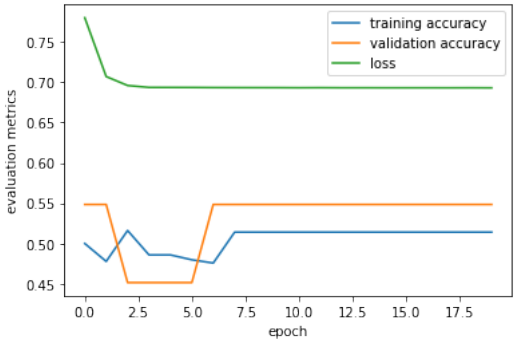

Figure 4.8. Adding layers seems no to bring any improving.

It is clear that the model still does not work. Another possibility to try to increase performances is to enhance the number of parameters: at last convolutional layer and in the fully connected one the number of parameters was increased to 128 (5 and 7 in model described in 9). Results are shown in figure 4.9 accuracy score on the test set was 49%.

Other trials have been made but the situation does not get better.

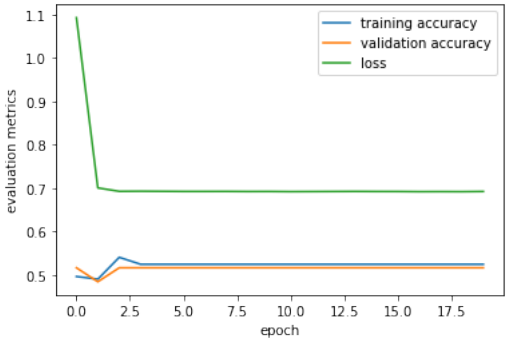

Figure 4.9. Even increasing the number of neurons in layers does not increase performances.

Chapter 5

Results

By applying the same model on channel data the situation is the same: using the same model with similar data leads to same results 5.1.

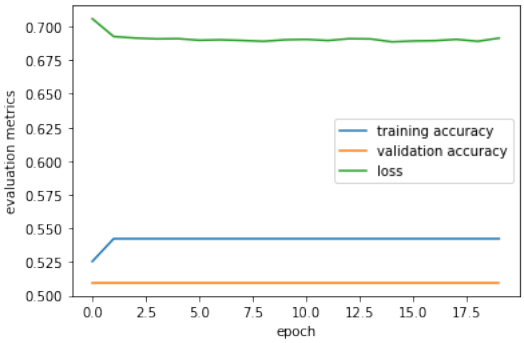

Figure 5.1. Performing the last described network on channels data brought none significant change. Accuracy score over the test set was 53%.

The small accuracy changes are more related to the small changes in dataset distribution than to network learning.
This is poor results are not surprising: in fact, it is impossible to distinguish two different labeled images (see fig. 5.2). Supervised machine learning is able to reproduce the human ability to distinguish images but if the "supervisor" has difficulties to perform classification it is unrealistic that the machine can learn it.

After these considerations it is clear that the chosen signal imaging technique does not fit data. It loses important information and does not underline any interesting behavior. As explained in 3.3 biological signals are slows, with minutes time periods. When setting spectrograms parameters, the number of points were set to 1024, so the smallest sampled frequency is the one related to a signal that runs a whole period on 1024 samples: $T_{max} = \frac{1}{500}\ s \cdot 1024 \approx 2\ s$ which correspond to $f_{min} \approx 0.5\ Hz$. Any information related

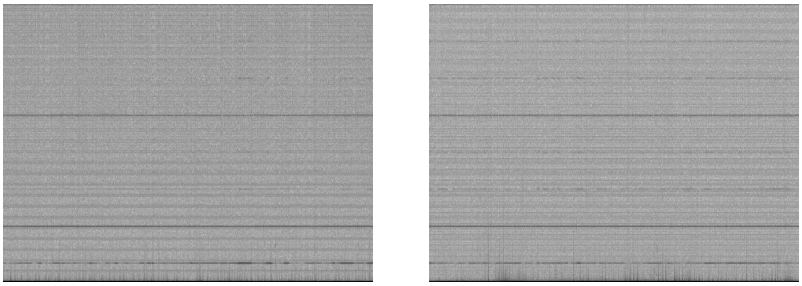

Figure 5.2. Left picture is related to a relaxed moment or to a noisy component, right one is related to an infestation moment. It is impossible to inspect the labels just by looking at these pictures.

to signals that changes more slowly than 0.5 times per second is lost in the continuous components. Using longer spectrograms with longer time windows may not be efficient leading to huge images (longer time window means more rows in the image) difficult to manage for the algorithm. In addition to that by looking at sources' spectrum plots no information seems to come at higher frequencies, valuable information stays in signals that do not change more than 20-30 times in 30 minutes (see fig. 3.4 and 5.3).

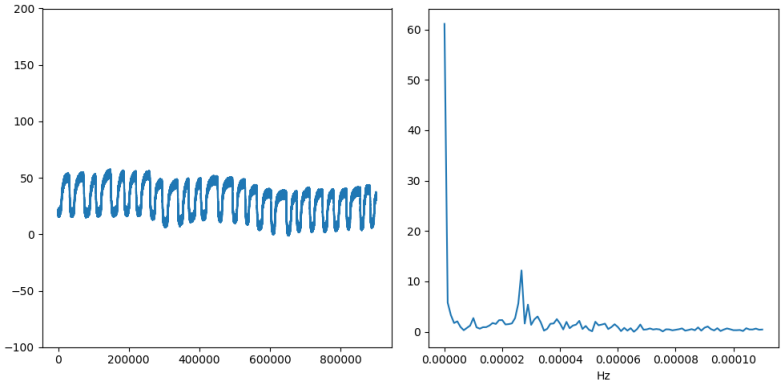

Figure 5.3. This is the previously shown infested signals (fig 3.6) and its spectrum just considering the first 100 frequency components. In this way every signal faster than $\frac{100}{30\times60\times500} \approx 0{,}0001\ Hz$ is discarded like in a low pass filter. This signal seems to vary between 20-30 times in 30 minutes and, in fact, there is a peak on a low frequency.

In order to achieve better results signals must be treated in a different way and a new imaging technique must be designed.

Part III

Part Three

Chapter 6

New Approach

Spectrograms' criticisms and weaknesses have been explained. It is necessary to find a new way to treat data. First of all, it is fundamental to highlight low frequency behavior: it has been shown that there is no information in high frequency components and so signals can be filtered using a low-pass filter. Then conversion to images should take into account signal shape, it should be computationally fast, easy to implement and it should generate easy-to-distinguish images with a manageable size.

6.1 New Imaging Technique

Based on Wang and Oates [2015] work signals are converted into RGB pictures through 3 different imaging techniques: Gramian Angular Fields (sum and differences, GAFS and GAFD) and Markov Transition Field (MTF). Each of these transformations returns a matrix that will form the final RGB image (a matrix for each color channel).

6.1.1 GAFS and GAFD

Gramian Angular Fields come from Gram Matrix and inner product: Gram matrix is defined as the dot-product of every couple of vectors in a considered set (see eq. 6.1). In the contest of time series Gram Matrix might be helpful because it should allow to encode time into matrix geometry (if vectors are time dependent).

$$\begin{bmatrix} ||v_1|| & v_1 \cdot v_2 & v_1 \cdot v_3 & ... & v_1 \cdot v_n \\ v_2 \cdot v_3 & ||v_2|| & v_2 \cdot v_3 & ... & v_2 \cdot v_n \\ v_3 \cdot v_1 & v_3 \cdot v_2 & ||v_3|| & ... & v_3 \cdot v_n \\ \vdots & \vdots & \vdots & \ddots & \\ v_n \cdot v_1 & ... & & & ||v_n|| \end{bmatrix} \tag{6.1}$$

The idea is to treat signals as time series S and transform them by a two-step procedure: normalizing and mapping them into polar coordinates $(s_i, t_i) \rightarrow (\phi_i, r_i)$, then

exploiting the angular field by considering trigonometric operations between each observation point. In this way it is possible to identify the temporal correlation within different time intervals.
Given a time series $S = s_1, s_2, ..., s_T$ of T real-value observations, it is possible to map it in the interval $[0, 1]$ (normalizing it) and represent it into polar coordinates by using:

$$\begin{array}{cc} \phi_i = arcos(s_i) & s_i \in [-1,1] \\ r_i = \frac{t_i}{T} & t_i \in \mathbb{N} \end{array} \tag{6.2}$$

Where t_i is the time instant and T is the vector length; r_i and ϕ_i are respectively the radius and the angle in the Cartesian coordinate system. This mapping is bijective: for every time series there is one and only one possible polar-coordinates mapping and vice versa (due to *arccos* bijective characteristic for values in [0,1]).
To build Gramian matrix the first idea would be to compute the inner product in the 2D polar space but this approach has a limitation: vectors' norm is time dependent and it increases with time, inner product will be distorted highlighting most recent observations. An alternative to the simple inner product is to consider the angle operation:

$$x + y = cos(\theta_x + \theta_y) \tag{6.3}$$

This is not an inner product (semi-positive and linearity are not always satisfied) but it is possible to build a sort of Gram matrix defined as:

$$\begin{bmatrix} cos(\theta_1 + \theta_1) & cos(\theta_1 + \theta_2) & cos(\theta_1 + \theta_3) & ... & cos(\theta_1 + \theta_T) \\ cos(\theta_2 + \theta_1) & cos(\theta_2 + \theta_2) & cos(\theta_2 + \theta_3) & ... & cos(\theta_2 + \theta_T) \\ cos(\theta_3 + \theta_1) & cos(\theta_3 + \theta_2) & cos(\theta_3 + \theta_3) & ... & cos(\theta_3 + \theta_T) \\ \vdots & \vdots & \vdots & \ddots & \\ cos(\theta_T + \theta_1) & ... & & & cos(\theta_T + \theta_T) \end{bmatrix} \tag{6.4}$$

As pointed out before this is not an inner product but a penalized version of that. Converting back polar coordinates into Cartesian space gives:

$$\begin{gathered} cos(\theta_1 + \theta_2) = cos(arccos(x) + arccos(y)) = \\ cos(arccos(x)) \cdot cos(arccos(y)) - sin(arccos(x)) \cdot sin(arccos(y)) = \\ x \cdot y - \sqrt{1 - cos(arccos(x))^2} \cdot \sqrt{1 - cos(arccos(y))^2} = \\ x \cdot y - \sqrt{1 - x^2} \cdot \sqrt{1 - y^2} \end{gathered}$$

Using this operation, it is possible to build GAFS (Gramian Angular Field Summation) matrix G as:

$$\begin{gathered} cos(\theta_i + \theta_j) = \\ S' \cdot S - \sqrt{1 - arccos(cos(s_i))^2} \cdot \sqrt{1 - arccos(cos(s_j))^2} = \\ S' \cdot S - (\sqrt{\mathbb{I} - S^2})' \cdot \sqrt{\mathbb{I} - S^2} \end{gathered}$$

Where S is the time series row vector and $\mathbb{I}$ is the unity vector. By considering the *sin* of the angles difference it is possible to define and compute GAFD (Gramian Angular Field Difference) matrix analogously. The result G is:

$$sin(\theta_i - \theta_j) = \\ (\sqrt{\mathbb{I} - S^2})' \cdot S - S' \cdot \sqrt{\mathbb{I} - S^2}$$

GAFs contain temporal relations because $G_{i,j|\ |j-i|=k}$ represents the observations correlation with respect to time interval k. To sum up, the strength points of using these matrices are: diagonal values are rescaled original values of the time series, and temporal relations are taken in account. The main drawback is that data size rose significantly ($T \to T^2$) but this is not surprising as the goal is to get a 2D matrix from a 1D vector.

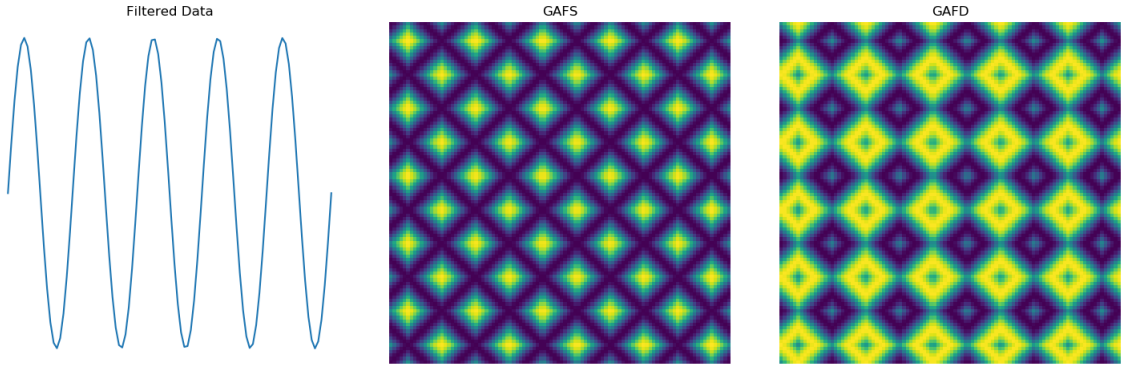

Figure 6.1. This is an example of GAFS and GAFD building by taking a simple *sin*-wave. Obviously, this not the final RGB color image but just two color channels.

6.1.2 MTF

To build the third color channel, Markov Transition Field will be used (MTF). Given a time series, data are divided into quantile bins q_j ($j \in [1, Q]$, default value $Q = 5$) according to their magnitude. These bins will be considered as the possible states of a Markov process and the matrix $M \in \mathbb{R}^{Q \times Q}$ will be estimated by computing (and normalizing) transitions frequencies between quantile bins: m_{ij} will be the transition probability $q_i \to q_j$. To create the final matrix bins that contain the i-th and j-th observation are defined as q_i and q_j and each observation s_i is associated to the corresponding bin, G will be defined as:

$$\begin{bmatrix} m_{i,j\ |x_1 \in q_i; x_2 \in q_j} & m_{i,j\ |x_1 \in q_i; x_3 \in q_j} & \cdots & m_{i,j\ |x_1 \in q_i; x_n \in q_j} \\ m_{i,j\ |x_2 \in q_i; x_2 \in q_j} & m_{i,j\ |x_2 \in q_i; x_3 \in q_j} & \cdots & m_{i,j\ |x_2 \in q_i; x_n \in q_j} \\ \vdots & \vdots & \ddots & \vdots \\ m_{i,j\ |x_n \in q_i; x_2 \in q_j} & \cdots & & m_{i,j\ |x_n \in q_i; x_n \in q_j} \end{bmatrix} \tag{6.5}$$

$G_{i,j|\ |j-i|=k}$ stands for the transition probability between the points in time interval long k. The main diagonal G_{ii} will represent the self-transition probability at time step i. Result is shown in fig. 6.2.

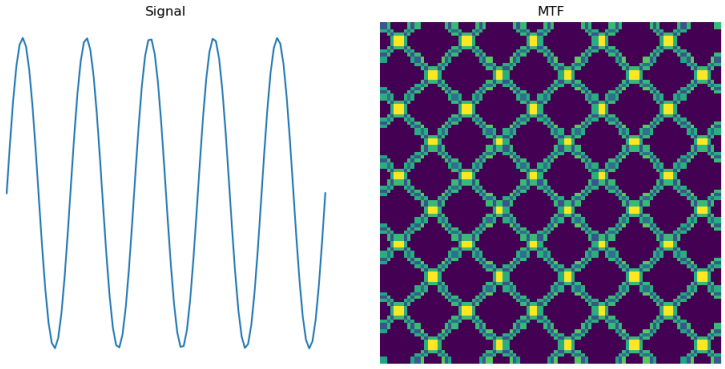

Figure 6.2. This is a MTF example computed by considering the same simple *sin*-wave of fig. 6.1. This is a color representation of 2D matrix (do not confuse with a 3D tensor or a RGB image)

Chapter 7

Algorithm Development

These new described techniques are interesting but there are problems when dealing big vectors (channels recording have 900000 elements). A first requirement is to reduce data size and the chosen way is to massively filter them: data have been mapped into frequency space using Fourier transform, just the first 100 components were taken and signals were reconstructed by anti-transforming. Essentially a low pass filter was applied. This is a heavy operation because at the end vector length is 100, but this is made by taking into account convenience (managing smaller data) and data significance (it has been shown in previous chapters that high frequency components were almost useless).

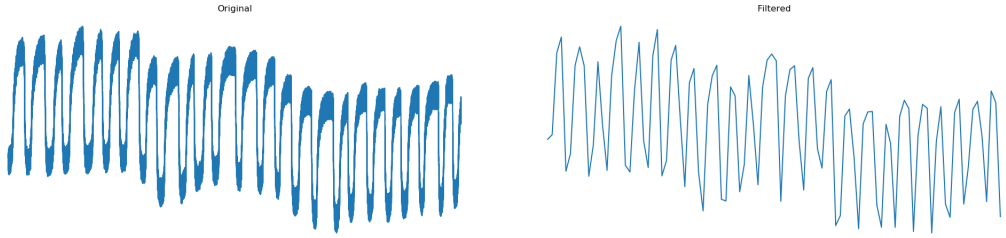

Figure 7.1. This picture shows the way signals get modified by anti-transforming just the first 100 components. Signals' main shape and frequency are preserved. Obviously the second graph is less detailed but this is the price that must be paid for dimensionality reduction.

At this point, to compute the GAFs and MTF matrices, the respective built in functions were used. Luckily these were implemented in `pyts`, a python package for time series classification: `GramianAngularField()` and `MarkovTransitionField()` are easy to use, fast and reliable. These functions receive the time series and then transform into matrix by `fit_transform()` command. Each method will return images with 100×100 pixels, so at the end dataset will be a collection of $100 \times 100 \times 3$ tensors.
GAFs function main input parameters are: the considered percentage of the time series, the desired range to map input data ([0,1]), the method ("*summation*" or "*difference*" to specify if GAFS and GAFD) and the possibility to use overlapping windows. Examples

are shown in figures 7.3

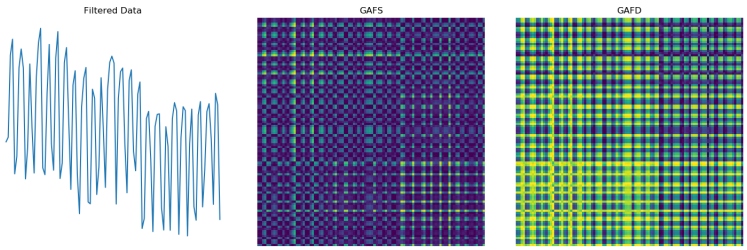

Figure 7.2. By applying the described procedure on stressed data this is the result. Images look squared and seem to reflect signal frequency and behavior.

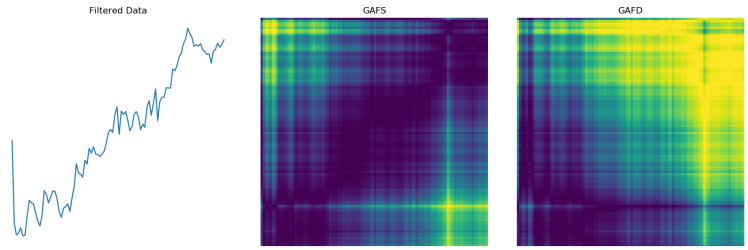

Figure 7.3. This is the result of procedure applied to some relaxed data (no spider-mites activity). Images look more uniform and color gradient is more regular. A CNN should be able to catch this kind of features.

MTF inputs are: the considered percentage of the time series, the number of bins (it will be left to default value 5), the strategy to esteem bins' width (if the bins must have the same width or the same amount of points) and if an overlapping window should be used. Examples are shown in figures 7.5.

In order to code and easily manage these tools a function in `plottingFunctions()` Python file was written: `creation_matrix()`. Input are just the filtered signal and the requested type of transformation; it is possible to specify to perform them all: GAFs and MTF are computed and piled in a 3D tensor forming the final RGB image (red channel is associated to GAFD, green to GAFS and blue to MTF, see fig.7.7).

7.1 Dataset Generation and Data Labeling

The `main` has been run again over all data used before in spectrograms-generation phase but with new imaging algorithm. A resume of this procedure is shown in figure 7.8. At the end of all cycles the number of files is the same of before (827 images for sources one and 690 for channel one), the only difference is the way images are produced. The goals of classification algorithm now are two: evaluate the goodness of this new way to convert signals into images and to see if source separation helps to generate better scores.

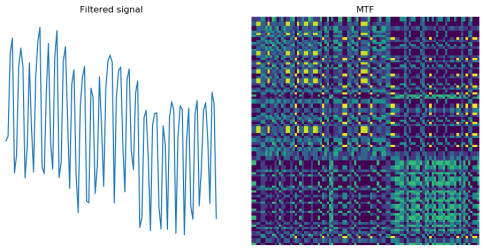

Figure 7.4. MTF computing function applied to stressed data, this is the result.

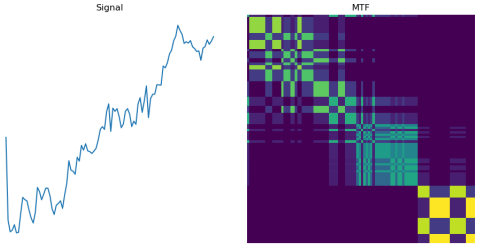

Figure 7.5. Relaxed data MTF, even at looking at this transformation images are different and that a CNN should be able to distinguish them.

In order to create the datasets related to stressed and relaxed moments used during classification, the procedure explained in 4.1 was used: the exact same channels and the same interesting moments were picked, labeled and saved into `channels_stressed_images`, `channels_relaxed_images`. This operation has been repeated with sources data related to the same moments. As before some relaxed data were added in `sources_relaxed_images` in order to maintain labels distribution close to 50%.

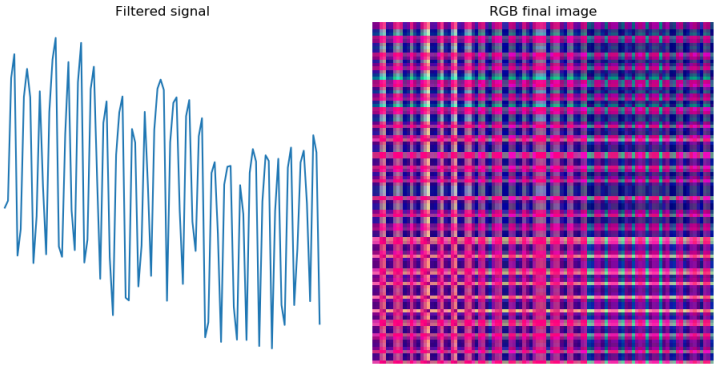

Figure 7.6. An example of stressed data final image. It is clear that the "squareness" is preserved and it seem to characterize infested moments.

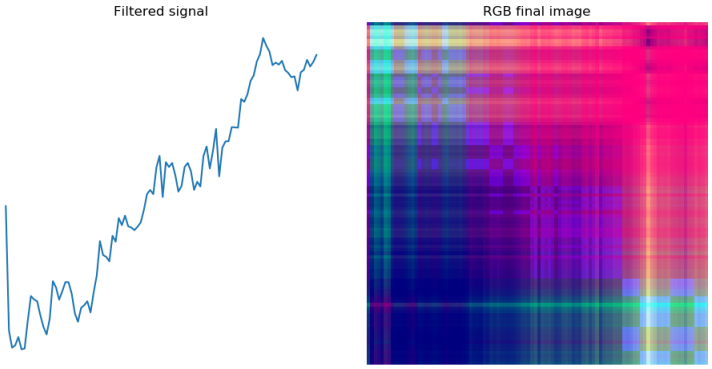

Figure 7.7. As before colors' gradients are smoother and regular and there are fewer squares or irregular lines.

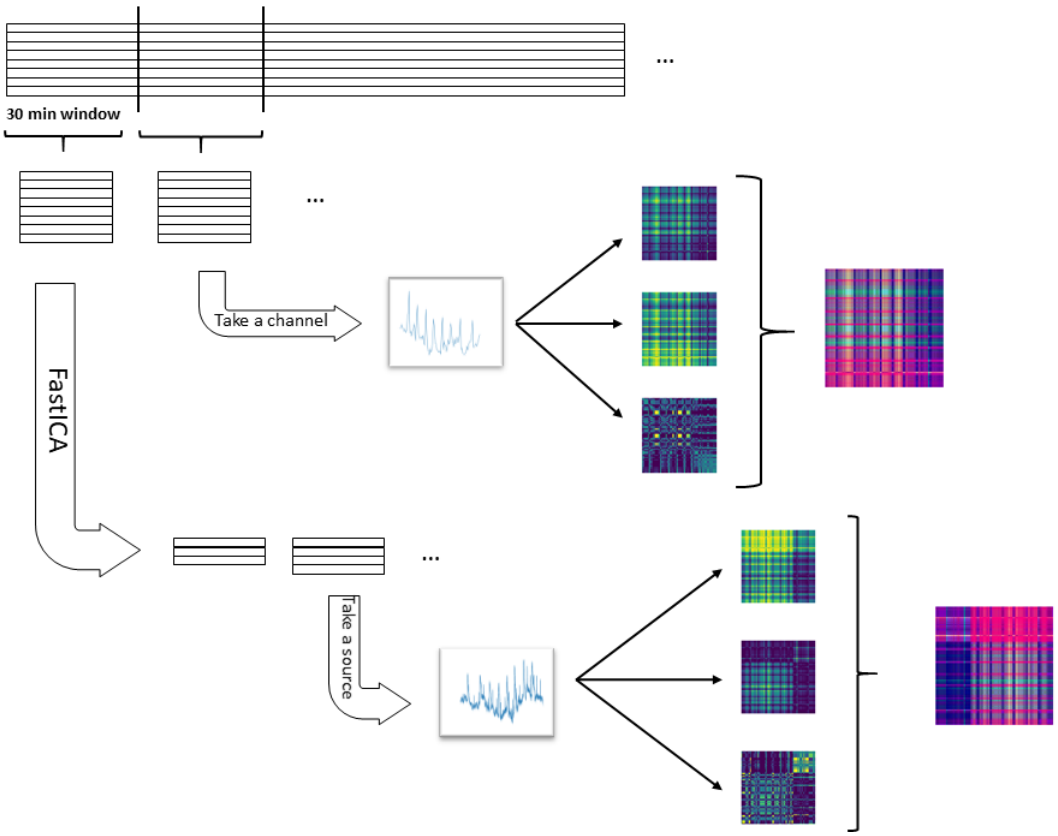

Figure 7.8. This is a scheme to explain how images are generated. Consider that every 30 minutes all 8 channels and all the sources are transformed into images.

Chapter 8

Signal Classification

Dataset were uploaded on Google Colab and a new CNN (see 4.2) has been built. Obviously before starting images are opened, saved in a numpy array and normalized, label vector has been created too. The whole set is shuffled and split into training and test set (with test set size 0.25 of the total, to be used for final evaluation).
To start, channels images have been used and a first model has been coded with the following structure:

1. A convolutional layer characterized by 256 filters with kernel size 3×3 and ReLU activation function.
2. A 2×2 max pooling layer.
3. A convolutional layer with 128 filters with size 3×3 and ReLU activation function.
4. A 2×2 max pooling layer.
5. A fully connected layer with 64 neurons using ReLU function (data must be flatten before).
6. A dropout stage with p=0.3
7. A final fully connected layer with just 2 neurons with softmax activation function.

The chosen loss function is sparse categorical cross-entropy, the chosen optimizer is Adam and the evaluation metric is accuracy. Batch size and epochs are set to 20 both and validation set to check overfitting is 0.2 of training set. This model is just a starting point and it will be enhanced and refined in the followings.

```
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=(100,100,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3)))
```

```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64))
model.add(Dropout(0.3))

model.add(Dense(2))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])

model.fit(X_train, y_train, epochs=20, batch_size=20, validation_split
    =0.2)
```

8.1 Model Performance

The first model performance is summed up the graph shown in figure 8.1. First results are satisfying, by looking at the learning curves it is possible to see the desired behaviors: loss curve decreases while accuracy scores increase along the epochs until they both reach a stable trend. The accuracy score over the test set is 84% (a little lower than the scores reached over the other sets), demonstrating that learning from these images is possible.

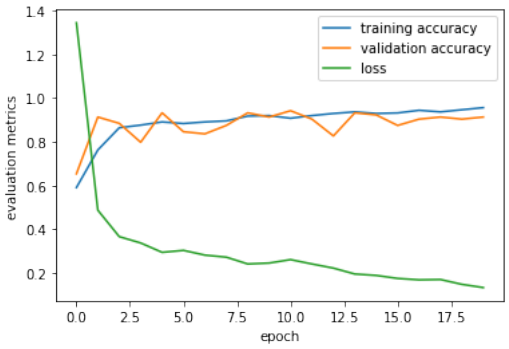

Figure 8.1. Loss curve decreases and accuracy scores increases, validation accuracy has more variability than training accuracy but on average they are similar, meaning that this network is able to generalize what learned from training set on other data (overfitting did not occur).

Even training time-performance were better than previous models used on spectrograms (see 4.3.5) but this is probably due to simplicity and capacity of this network (less layers and parameters).

8.2 Tuning

In order to see if it is possible to boost performances a convolutional layer (with 64 filters, kernel size 3×3 and ReLU activation function) and a following max pooling layer (2×2) have been added between points 4 and 5 in model described before (see chapter 8).

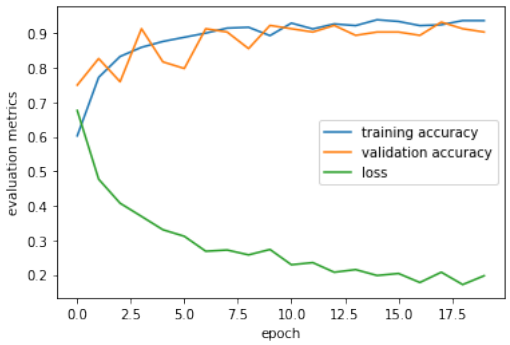

Figure 8.2. In this picture the ideal learning curve behavior is still present. Accuracy scores are now closer.

The addition of one layer leads to a small improve: loss and accuracy scores seem to converge earlier and they reach slightly better values. Even the score on the unseen test set increased: 87% (similar to the learning accuracy scores). Another exploited way to increase performances is to enhance the number of neurons at each layer and to add another fully connected layer followed by a dropout stage. Model now consists in:

1. A convolutional layer characterized by 512 filters with kernel size 3×3 and ReLU activation function.
2. A 2×2 max pooling layer.
3. A convolutional layer with 256 filters with size 3×3 and ReLU activation function.
4. A 2×2 max pooling layer.
5. A convolutional layer with 128 filters with size 3×3 and ReLU activation function.
6. A 2×2 max pooling layer.
7. A fully connected layer with 128 neurons using ReLU function (data must be flatten before).
8. A dropout stage with p=0.3
9. A fully connected layer with 64 neurons using ReLU function.
10. A dropout stage with p=0.3

11. A final fully connected layer with just 2 neurons (as many as the labels) with softmax activation function.

Due to the increase of the number of weights that need to be estimated this model gets slower training phase.

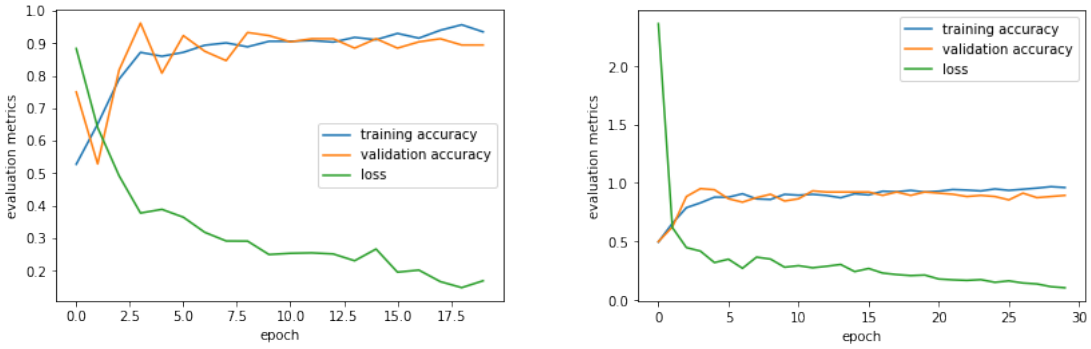

Figure 8.3. Even in this case the learning curves have and ideal behavior and similar performances as before. Increasing the number of layer and neurons did not boost performance. To check that performances do not improve anymore all neurons were doubled again (first convolutional layer had 1024 filters) and results are shown in right graph. This last network training was really slow.

Figures in 8.3 show last model learning curves. These CNNs have satisfying scores and on test data they both get 87%.
The good news are that CNNs work well on these new images, they do not overfit and are easy to implement and train (a simpler model can be use without any particular drop of performances). Adding layers or increasing model complexity seems not to help and this may be caused by some problems that will be discussed later.

8.3 Model on Sources Images

The model described in chapter 8 has been applied to sources images to evaluate if source separation produces better images that can lead to higher accuracy scores. All hyperparameter are left to their values set in the previous analysis. Results are shown in figure 8.4.

The score on final test set used for evaluation is: 78%. Results are good but not as good as before when channel data were used: learning seems more difficult and slower, validation and training accuracy start ranging between two different values (over 90% on training, around 80% on validation set, maybe due to some generalizing difficulties or due to a start of overfitting).
Sources data were used to train a more complex model (see model described at 8.2) to see if it is possible to increases accuracy and performance in order to reach values comparable with those obtained when channels data were used.
Results were similar and network seems not able to learn more than that (the test set

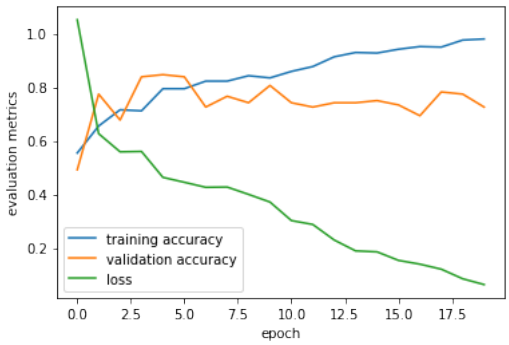

Figure 8.4. Loss and accuracy are still good (loss decreases while the accuracy metrics increase) but learning looks slower and with lower results.

accuracy is 82%) and learning curves are shown in see fig. 8.5). This means that source separation do not provide better images to classify stressed or relaxed moments and increasing model complexity does not bring significant improvements.

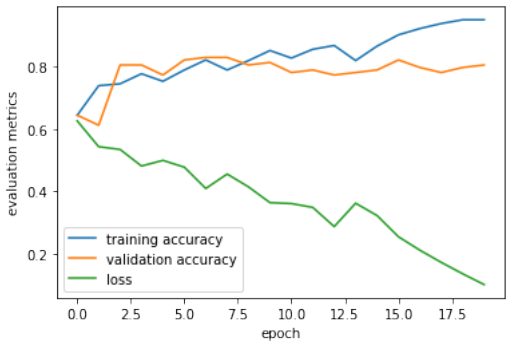

Figure 8.5. Loss and accuracy have a more linear behavior than before: it is still good (loss decreases while the accuracy metrics increase) but learning looks slower.

Chapter 9

Results

By this first analysis it is possible to say that encoding signals into this type of RGB images is a good choices. The transformation seems to capture stressed characteristics and map them into features that convolutional neural networks are able to learn.
This technique seems to easily reach 85-87% accuracy scores, but it does not reach higher values probably due to datasets composition (labeling mistakes, wrong data selection, as will be discussed later).
Source Separation seems not to help in producing better images: it must be pointed out that the main reason for their use (filter noise and the 50 Hz supplier signal) became useless when low pass filter was used. Still they allowed a better inspection of data: due to their sensitivity it was possible to state that high frequency behavior was just noise. The reason of their worse performances is probably due to the way sources are computed from observed signals: in fact, they are a linear mix of channels recordings and they may mix relevant information with some useless one. In addition to that channels images and sources ones were labeled by looking at recordings behavior but sometimes by inspecting the dataset it is clear that source separation detects infested signals from channel data where it would be difficult to say if plant is infested or not (see fig. 9.2).

This type of interesting technique ability (due its sensitivity) has not been measured and it is impossible to evaluate it just by using CNNs on a dataset built from a selection of images.
Probably the type of stress is not the ideal to evaluate source separations too: parasites stress looks well located (usually it is present in just one of the eight channel) and it is not a global status of the plant. Source separation would be able to capture common components in a mix but if the interesting signal is present in just one observation, source separation may be useless (and it may loose the information about what channel is infested). Probably it should be evaluated on other type of stress experiments (like water and nutrient deficits).

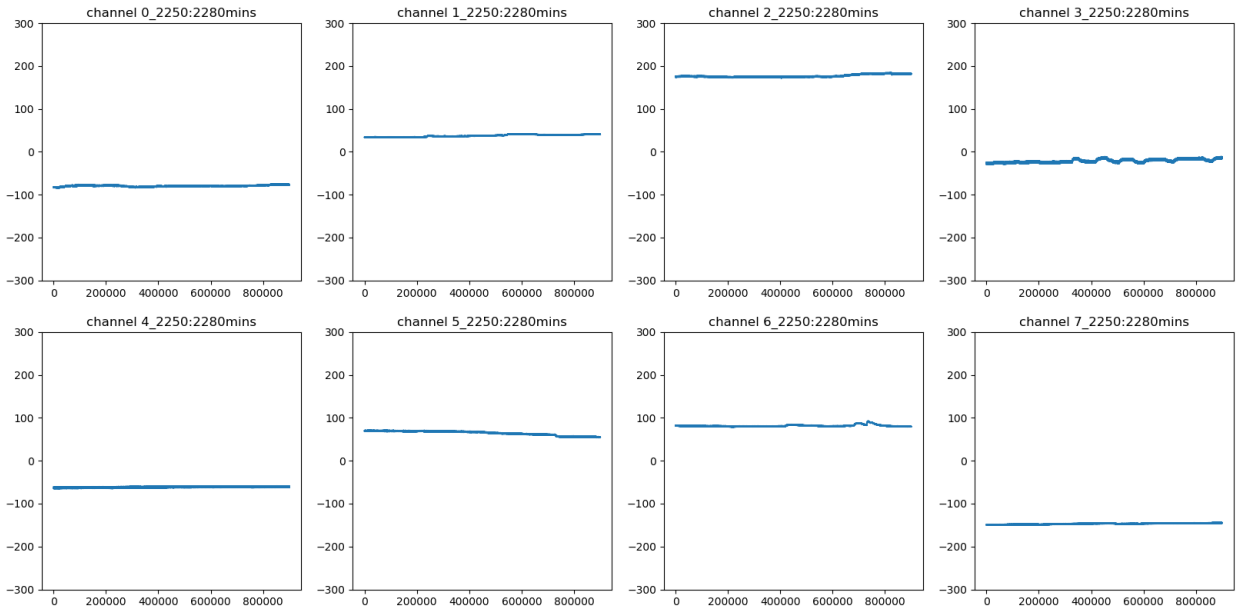

Figure 9.1. Watching at these channels signals it would be difficult to see if there is any stress factor.

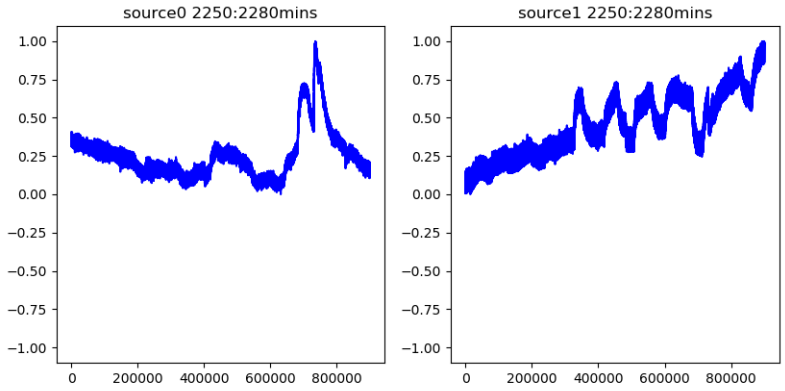

Figure 9.2. These sources computed by previous channels data show some kind of infested behavior that it would not be catched by a normal approach showing some hidden potential in using source separation algorithms.

Part IV

Part four

Chapter 10

Conclusions

Results reached using the imaging technique based on Wang and Oates [2015] work are pretty satisfying, on the other hand using spectrograms leads to a relevant information lost. It has been proved that it is possible to distinguish between stressed and relaxed moments using supervised machine learning with high accuracy.
Although the presented models must be refined prior to be used on a real-world environment yet: datasets were balanced and this characteristic has been helpful to evaluate if the recognition is possible (using a dataset with a small value of stressed data and a lot of relaxed ones can bring to erroneous high accuracy scores) but it leads to over-estimation of the phenomena which does not occur along 50% of the observations. Different data selection and evaluation metrics should be exploited.
The procedure can be generalized to other type of experiments and a more robust routine should de defined, this may help to deal with the other experiment data and signals which may have different frequency, shape or characteristics than spider-mites infestation ones.

10.1 Criticisms

Source separation seems not to bring better signals or images but it is also true that this experiment was not the best way to evaluate it. Source separation should be tested with more stress factors combined together (i.e. a dehydrated and infested plant, or a nutrient-need organism under thermal shock) and it should be able to separate this type of factors even if overlapped or covered.
Obviously, the separation algorithm works under the strong assumptions described in chapter 2.1. There are no guarantees that signals are independent and stationary, biosignals such plant ones seem to vary a lot and they change along time, components can switch off and on, they may have big ranges of frequencies and many other problems or features can impede the algorithm to get successful results. Esteeming the number of components in a mixture is still a problem: the proposed technique seems to work but there is no way to evaluate it, a part from using synthetic data or empirical explanations. During the analysis the number of sources has always been considered smaller than the number of channels, in real-world environment there is the possibility that this is not

always true (number of channels can be enhanced without big difficulties but this may lead to computational costs increase).
Another weak point of this approach is dataset creation: data were not labeled from a domain expert and mistakes may be introduced. A deeper dialogue with biologists and agronomists and a better understanding of the underlying phenomena would boost performances for sure. Some control data recorded on a perfectly healthy plant over some days are necessary to analyze better the to better understand the real causes that generate signals and to have better data for classification. Some misclassification errors are due to wrong and not uniform labeling: to label data it is fundamental to have a unique and sure judging criterion but this is possible just when the spider-mites feeding process and the bio-signal generation process is completely understood. Otherwise some unsure data may lead to wrong labeling and lowering the recognition model performances (an example is shown figure 10.1). Probably due to this type of data some mistakes have been introduced along the dataset and this may be one of the reasons why neural networks accuracy do not increase too much during tuning phase.

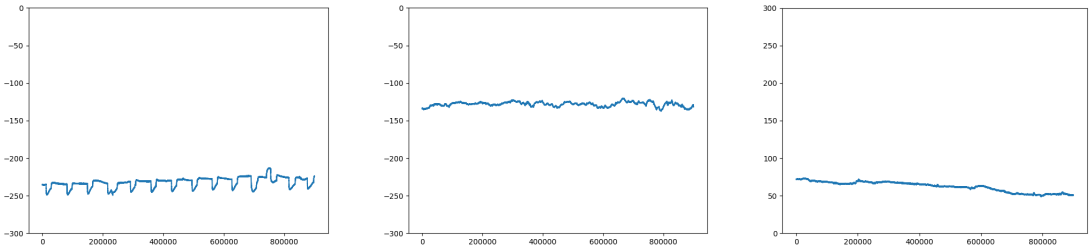

Figure 10.1. Left image is a stressed channel, right one is a relaxed. In the middle the situation is quite ambiguous because the signal has some small irregular fluctuations but the overall trend looks quite and relaxed. How this data should be classified? Considering that this type of situation is not rare.

The used convolutional neural networks were just simple models using few layers because the problem is binary (just two possible labels), dataset was balanced and its size was big but not so much to justify the use of more complex networks. This simplicity is enough if the goal is to evaluate a procedure or to check if a dataset is suitable with the network task but it is not enough if the model must be used in a real-world scenario, with more variables and possible outcomes (multi-label problems).

10.2 Improvements

All weakness points described in the previous section can be overcome by some corrections or by adopting different approaches and strategies.
First of all a deeper study of the domain will improve the classification algorithm and the source separation system: a statistical study about signal correspondence along the different experiments will make easier to associate a particular signal behavior to the relative stress source and it would be fundamental to build a sort of "*plant dictionary*". Then by

better understanding the effect-cause relationship it would be easier to evaluate sources separation quality and to improve labeling phase reducing misclassification errors.
Another possible improvement is to exploit different source separation techniques: FastICA is really helpful because it is quick, easy and it is able to treat big amount of data without problems but its drawbacks are explained in 2.3.4. Recurrent neural networks (RNN) may be exploited: these may be helpful to deal with flow data but they require more tuning, training phase is slow and they need some data as to compare and learn.
To better deal with ambiguous data (fig. 10.1) a solution would be to label images with 3 possible values: one for high stress moments, one for relaxed and one for data that seem halfway, this approach would return an esteem about the level of stress that the plant is suffering but it will require different, more sophisticated networks.
Using multi-label dataset will be necessary when the solution will be implemented in a more realistic scenario: in a real-world environment plants can suffer from dehydration, from infestations, from climate conditions, they can need food, nutrients or special treatments. Possible outputs are many and the goal is to build an intelligent system that is able to tell what plant needs. To do so the only way is to set a multi-label problem. Obviously having more possible outcomes increases model complexity and deeper neural networks will be the necessary. Pre-trained deep neural networks can be a possible solution: there are many structures of pre-trained models with many layers that are suited for image classification with a big number of labels and images. Just first deeper layers have been already trained and last ones must be trained on the user dataset in order to adjust them to the required problem.

10.3 Future Work

My work is just a starting point and, in my opinion, next steps should be focused on implementing a simple model that can be used on a real-world scenario to evaluate this work potential.
Along this thesis a working procedure has been presented: data must be split into suitable time windows, relevant information must be detected and signals needs to be filtered, converted into RGB images, label vectors need to be created and at the end the machine learning model must be trained and tested. This routine must be generalized and adapted to other type of experiments, the aim is to find a suitable time window to capture all type of stresses and to build a model which is able to distinguish among images coming from different experiments.
Another important future project will be adjusting the described routine to process flows of data. A solution strategy may be to apply it on a self-updating time window: at every time step data are updated with more recent recordings, the oldest ones are discarded, then the whole window gets processed to create images (see 7.8) which are finally classified by a already trained model (using Google Colab makes possible to download and save remote trained models and use them on local processors).
For sure a better CNN model should be fitted and implemented, machine learning models must be correctly trained in order to give the right importance to the stress signals and correctly esteem the class probabilities (this information can be used to measure the level

of plant stress too).
One of the first results will be a device which is able to record plants' signals, process and classify them in order to tell what the plants need: in this way growing and monitoring would be easier, more effective and resources waste almost zeroed (think about water saving or insecticide use).
Without any doubt this research field has a great potential: there is nothing in literature dealing with plants bio-signal, there is a lot of work to do and a lot of research needs to be done. I like to think at my work just as starting point to future developments. The topic is really interesting and it is not difficult to see future applications and why this kind of tools will be the future of agricultural and agri-food sectors.

Bibliography

Juha Karhunen Aapo Hyv¨arinen and Erkki Oja. *Independent Component Analysis*. John Wiley, 2001.

Nicolas Bensoussan, M. Estrella Santamaria, Vladimir Zhurov, Isabel Diaz, Miodrag Grbić, and Vojislava Grbić. Plant-herbivore interaction: Dissection of the cellular pattern of tetranychus urticae feeding on the host plant. *Frontiers in Plant Science*, 2016.

Jutten Christian and Jeanny Herault. Une solution neuromimétique au problème de séparation de sources. *Machines Numériques*, 1988.

Tran Daniel, Fabien Dutoit, Elena Najdenovska, Nigel Wallbridge, Carrol Plummer, Marco Mazza, Laura Elena Raileanu, and Cédric Camps. Electrophysiological assessment of plant status outside a faraday cage using supervised machine learning. *Nature*.

Naik Ganesh R. and K. Kumar Dinesh. Determining number of independent sources in undercomplete mixture. *EURASIP Journal on Advances in Signal Processing*, 2009.

Zhiguang Wang and Tim Oates. Imaging time-series to improve classification and imputation. 2015.