Università Degli Studi di Milano Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Data Science

Decoding Raman Spectroscopy Towards the Diagnosis of SARS-COV-2 Infection and Other Diseases.

Relatore: Prof.ssa Vincenzina Messina

Correlatore: Dott. Cristiano Carlomagno

Dario Bertazioli

Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano-Bicocca,
Viale Sarca 336, 20126 Milano, Italia
Matricola n° 847761

A.A. 2019/2020

Ringraziamenti

Desidero dedicare questo spazio per un ringraziamento a tutte le persone che mi hanno permesso di portare a termine questo lavoro di tesi, e con esso il percorso di formazione universitaria.

Per prima cosa, grazie alla relatrice di questa tesi, prof.ssa Vincenzina Messina, sia per aver creato l'opportunità di partecipare a questo interessante e importante progetto di ricerca, sia per la sua enorme disponibilità al confronto.

Un grazie anche alla dott.ssa Marzia Bedoni, al dott. Cristiano Carlomagno, e a tutti i membri del Laboratorio di Nanomedicina e Biofotonica Clinica, per aver ideato e posto le solide fondamenta di questo progetto, e per la loro costante disponibilità.

Ringrazio inoltre il mio collega Michele Andrico, che mi ha affiancato in una fase cruciale del progetto.

A huge thanks to my dear friend Anders, for sharing its incredible knowledge of the Deep Learning field in our conversations, as well as for many of his life-saving tips.

Come non ringraziare tutti i colleghi con i quali ho condiviso questi anni universitari, molti dei quali sono diventati cari amici. Penso ad Alessandro, Andrea, Fabrizio, Massimilano, Mirko, con i quali ho condiviso parecchi progetti, e che hanno rallegrato la quotidianità in università.

Un ringraziamento speciale ai miei genitori per il loro impegno nel motivarmi e nel sostenermi, ma anche ai miei parenti, in particolare i miei zii Sergio e Loredana, per essermi sempre vicini.

Decoding Raman Spectroscopy Towards the Diagnosis of SARS-COV-2 Infection and Other Diseases.

Dario Bertazioli

Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano-Bicocca,
Viale Sarca 336, 20126 Milano, Italia

November 30, 2020

Abstract

Raman Spectroscopy promises the ability to encode in spectral data the significant differences between saliva samples belonging to patients affected by a disease and samples of healthy patients (controls). Based on this assumption, an efficient and non-invasive method for pathology screening could be developed. However, the decoding and interpretation of the Raman spectral fingerprint is a difficult and time-consuming procedure even for domain experts. The presented work tests an end-to-end machine learning pipeline able to classify spectral data according to their provenance in terms of disease-positive or controls patients. The pipeline has been tested against the SARS-COV-2 Infection, and for the screening of neurodegenerative diseases such as Amyotrophic Lateral Sclerosis, Alzheimer, and Parkinson. The challenging character of this work is the attempt to deal with a small amount of data, at the same time trying to reduce the human delegated task, i.e. by avoiding the typical preprocessing required for interpreting the spectral data in an ordinary approach. The results are relevant, with the system being able to significantly discriminate positive (COVID, Alzheimer or Parkinson-affected) patients with satisfying performances, although a generalization study on a larger amount of data could be required for proper clinical settings.

Advisor: *Prof.ssa Vincenzina Messina*
Co-Advisor: *Dott. Cristiano Carlomagno*

Data and information confidentiality

All the data, analytical and computational processes, protocols, parameters, methodologies and results are confidential. All the presented material, when not already published, is aimed at the scientific publication on online journals. In particular all the procedures and results obtained in the "Parkinson's Disease" Sec. 4.2.2 and "COVID-19" Sec. 4.2.3 have been submitted to online scientific journals with peer-review processes [1, 2]. The authors deny the permission of the material reproduction and use without written explicit consent. For further information contact the author.

Contents

1 Introduction **1**

1.1 #AIForGood . . . 1
1.2 General Overview . . . 1
1.3 Data Collection Methods: Problem's Physics . . . 2
1.3.1 Spectroscopy Techniques . . . 2
1.3.2 Raman Spectroscopy . . . 3
1.3.3 RS on Saliva Samples . . . 6
1.4 Spectral Data Analysis and Computational Techniques Overview . 7
1.5 Medical Overview and Current Diagnosis . . . 8
1.5.1 Amyotrophic Lateral Sclerosis . . . 8
1.5.2 Parkinson's and Alzheimer's Disease . . . 8
1.5.3 COVID-19 . . . 9

2 Literature Review and State of the Art **11**

2.1 ML Applied to Raman Spectroscopy . . . 11
2.2 DL Applied to Raman Spectroscopy . . . 14
2.2.1 Our Contribution . . . 14

3 Elements of Machine learning, Deep Learning, and Optimization **16**

3.1 Intro to Machine Learning Basics . . . 16
3.1.1 The Learning Algorithm . . . 16
3.1.1.1 The Task . . . 17
3.1.1.2 The Performance Measure . . . 17
3.1.1.3 The Experience . . . 19
3.1.2 Generalization in Data Fitting . . . 20
3.1.2.1 Generalization . . . 20
3.1.2.2 Validation Techniques . . . 20
3.1.2.3 Theoretical Hints . . . 20
3.1.2.4 Capacity . . . 21
3.1.2.5 No Free Lunch and Regularization . . . 21
3.1.3 Parameters and Hyper-parameters . . . 23
3.1.4 Common Machine Learning Algorithms: Implementation Hints . . . 23
3.1.4.1 Support Vector Machines . . . 23

3.1.4.2 Decision Trees and Random Forests 25
3.1.4.3 Boosting Machines and XGB 25
3.2 Deep Neural Networks 27
3.2.1 Introduction and Historical Notes 28
3.2.2 Modern Deep Feed Forward Neural Networks 32
3.2.2.1 Following the Gradients to Learn 32
3.2.2.2 NN Architectures: General Design and Components 33
3.2.3 Optimization for Deep Neural Networks 36
3.2.3.1 Gradient Descent and Its Variants 36
3.2.3.2 Back-Propagation 39
3.2.3.3 Optimizers 42
3.2.4 Regularizing Deep Neural Networks 43
3.2.4.1 Training Behaviour in the Needs of Regularization 43
3.2.4.2 Explicit Regularizers 45
3.2.4.3 Implicit Regularizers 47
3.2.4.4 Self Regularization 49
3.2.5 Architectures 50
3.2.5.1 Convolutional Neural Networks 50
3.2.5.2 Others 54
3.2.6 Hyperparameter optimization 56
3.2.6.1 Automatic Hyperparameter Optimization 56
3.2.7 Transfer-Learning 61
3.2.8 Applications 63

4 Experiments **65**
4.1 The General Pipeline 65
4.1.1 Exploration 67
4.1.2 Data Processing 68
4.1.3 Data Synthesis 74
4.1.3.1 Plain Data Augmentation 75
4.1.3.2 Conditional Convolutional Variational AutoEncoder (CCVAE) 75
4.1.4 Performance Evaluation 78
4.1.5 Classification Models 78
4.1.6 Optimization 80
4.1.7 Transfer Learning 83

4.2 Diagnostic Applications 84
4.2.1 Raman fingerprint for the diagnosis of Amyotrophic Lateral Sclerosis (ALS) 84
4.2.1.1 Data 84
4.2.1.2 Results 87
4.2.2 Salivary Parkinson's disease (PD) fingerprint 87
4.2.2.1 Data 87
4.2.2.2 Results 89
4.2.2.3 Error Analysis 91
4.2.3 The Raman COVID-19 salivary fingerprint 92
4.2.3.1 Data 92
4.2.3.2 Results 94

5 Discussions and Conclusions **97**
5.1 ALS 97
5.2 PD 98
5.3 COVID-19 100

6 Future Work **103**

A Appendix A: Details for the ALS Task **116**

B Appendix B: Details for the PD/AD Task. **116**

C Appendic C: Details for the COVID-19 Task **118**

1 Introduction

1.1 #AIForGood

Technology, when applied to the extent of improving the quality of many people's life, finds its best scope.

In the field I am most concerned about, *Data Science*, there are, of course and luckily, various examples of projects aiming in the direction well described by the recent form of a hashtag: "#AIForGood". Personally, I have easily identified the correspondence of the previous statement in the application of data-analysis techniques to the medical and biomedical area. Therefore, the eventual and ideal objective of this thesis is clear: contributing to some example of technological applications for the good of our society.

1.2 General Overview

Concretely, this work deals with the technical implementation of a diagnostic procedure that could be applied to several pathologies. In particular, to demonstrate the power and flexibility of the general pipeline, these methods have been tested on three different neurodegenerative pathologies, Amyothrofic Lateral Sclerosis (ALS), Alzheimer's Disease (AD) and Parkinson's Disease (PD), and on the recent and widespread infection caused by the epidemic SARS-COV2 coronavirus.

A high-level overview of the methodology could be the following: when an individual is suspected to be affected by one of the aforementioned diseases, a saliva sample is collected and analysed by means of a Raman Spectrometer, encoding the information contained in the biofluid in the form of spectral data. Given the low-frequency (thus, low-energetic) characteristics of the Raman method, the acquired spectrum is difficult to interpret. At this point, the application of some machine learning (ML) and deep learning (DL) algorithms enable a fast and automatic data-analysis which results in a classification of the involved patients indicating the presence or absence of the suspected disease.

Of course, the full procedure is not fairly described by the previous statements, which can be taken as a fundamental summary containing all the main steps of the work.

To have clearer insights in the underlying mechanisms that allow the pipeline to produce useful results, it is worth to introduce some concepts relative to the low-level "problem physics" (Sec. 1.3), to the characteristics and current diagnostic procedure of the investigated disease (Sec. 1.5) and to the overall inner working of ML and DL algorithms (Sec. 3)

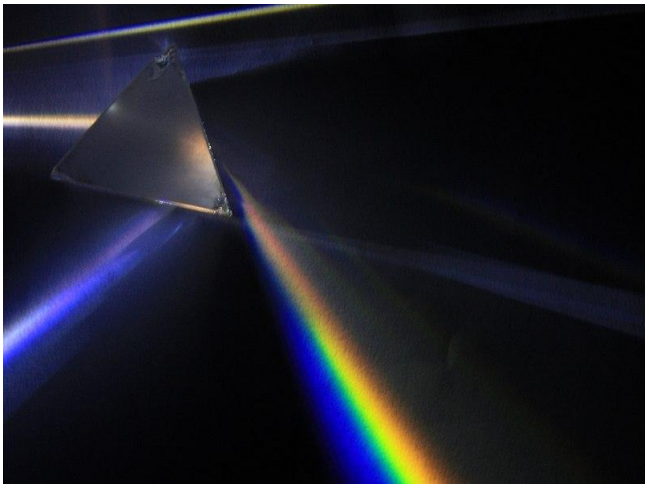

Figure 1.1: Analysis of white light by dispersing it with a prism is an example of spectroscopy. Figure taken from [5].

1.3 Data Collection Methods: Problem's Physics

In order to fully understand how the data required for the diagnostic pipeline is collected by the Laboratory of Nanomedicine and Clinical Biophotonics (LABION) research group, it is necessary to be introduced to Spectroscopy, a series of techniques mainly developed by physicists to better investigate the structure of matter, and, in particular, to Raman Spectroscopy, the peculiar spectroscopic technique exploited in this work.

1.3.1 Spectroscopy Techniques

Starting from the late 19th century, physicists, chemists, and bio-technologists have developed and employed all sort of techniques to investigate the intimate excitations of matter. One of the most famous and successful set of techniques is known as **spectroscopy**. Spectroscopy consists of the investigation of the interaction between matter and electromagnetic radiation [3, 4]. From a historical perspective, spectroscopy emerges through the study of curious phenomena related to the famous prism experiment represented in Fig.1.1: gas phase matter of visible light is absorbed and dispersed by a crystal prism in a characteristic way. Years after years, Spectroscopy, in particular in the electromagnetic regime, has improved becoming a fundamental exploratory tool in many fields of science, allowing to characterize the composition, the physical structure and more in depth the electronic structure of matter. Beside Raman Spectroscopy, that will be discussed later in details, the physics method of this thesis, other important applications revealed biomedical spectroscopy to be a fundamental technique in

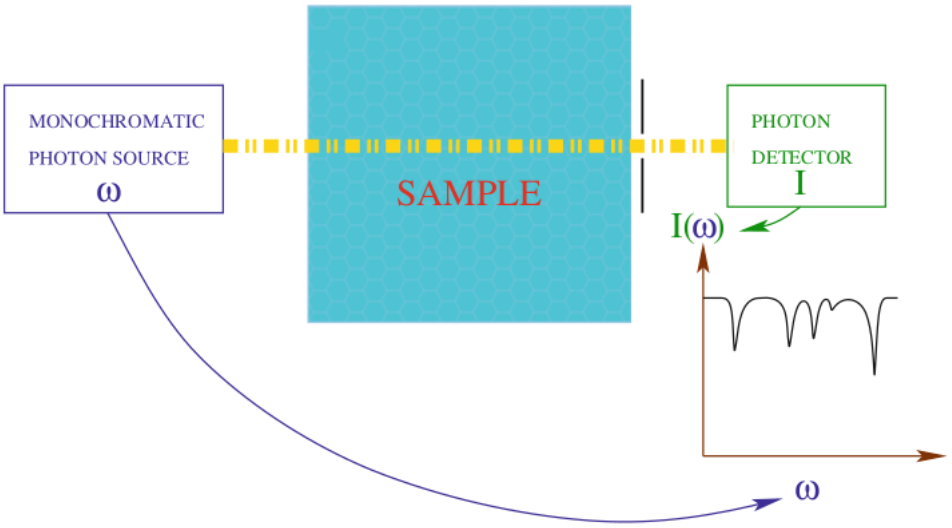

Figure 1.2: A conceptual scheme of the setup for absorption spectroscopy. Figure taken from [6].

the areas of tissue analysis and medical imaging. A rich body of evidence has been collected through two broad classes of spectroscopies:

1. **Absorption**: a collimated beam of coherent (monochromatic) light is directed through the sample. If the beam frequency happens to match a characteristic sample transition (informally, a particular energy threshold that corresponds to some structural excitement in the sample), the photons are absorbed. While the absorbed energy is transformed, e.g. re-emitted in random directions, the intensity loss of the beam is recorded as a function of frequency, producing a characteristic spectrum, as sketched in Fig.1.2,

2. **Emission**: the sample is excited (by means of high temperature, or energized through an electron beam) and the lights that it subsequently emits (during the de-excitation transition) is again recorded are a function of frequency (see Fig.1.3).

Scientists routinely acquire and investigate both kind of spectra in the IR, visible, UV and X-ray spectral ranges, to characterize the structural form of the target samples.

1.3.2 Raman Spectroscopy

Raman Spectroscopy (RS) is a spectroscopic technique that, given a target sample to be investigated, typically a molecular system, allow to study and observe low-frequency modes, also called rotational, vibrational and rotovibrational modes.

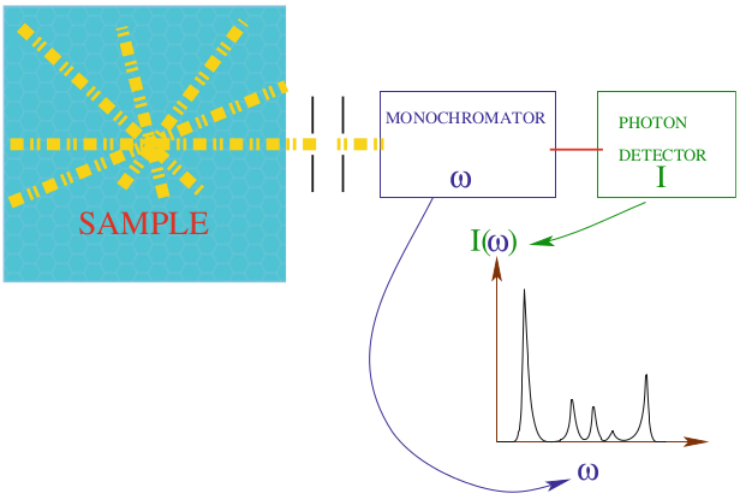

Figure 1.3: A conceptual scheme of the setup for emission spectroscopy. Figure taken from [6].

The "modes" of a molecular aggregate could be seen as a collective behaviour of its components in response to a perturbation of the system (in terms of energy): intuitively, the components of the molecules (i.e. atoms), excited by a laser beam, vibrates and rotates collectively until a phonon, a quantum of rotovibrational energy, is released after the de-excitation.

The physics of RS, also represented in Fig. 1.5, is rather simple: when an incident photon, part of a laser beam, is scattered by the target, its energy is modified with respect to the initial conditions. Filtering out other components of the scattering process, such as the Rayleigh scattering or the infrared absorption components, this energy shift between the incident and the scattered photon can be associated with a rotational or vibrational mode of the target, as previously mentioned. This process is referred to as Raman scattering, a particular type of inelastic scattering. Therefore, by varying the frequency of the incident laser beam, scientists are able to map the presence and the characteristics of the low-frequency modes at given energy by observing the energy shifts at which the light is emitted after being Raman Scattered.

The obtained mapping allows determining a sort of "fingerprint" (also referred to as *Raman Fingerprint*) directly associated with the molecular composition of the target sample. An example of Raman Spectrum is reported in Fig.1.4.

Since the RS exploits laser excitements, it is non-invasive and non-destructive, i.e the involved energy quantity is small and no radiations are produced. Due to those characteristics, in addition to being sensitive, rapid and fully automatable, the "Raman Fingerprints", besides their wide use in chemometrics applications, have shown their potential in the biomedical field: RS can, indeed, describe the

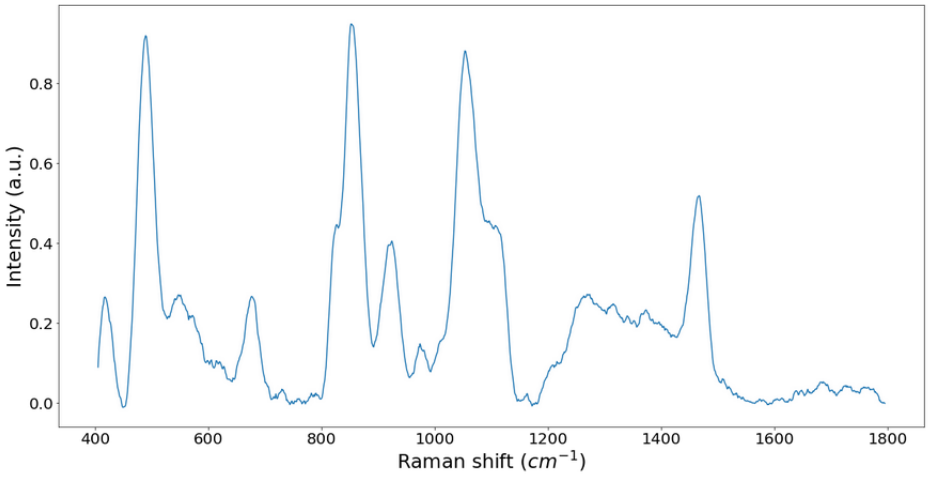

Figure 1.4: An example of Raman Spectrum. It is worth to notice that the energy shifts are sometimes reported as "Raman Intensities" along the y-axis of a typical Raman spectral plot in relation to the change in wavenumber of the incident photon. The wavenumber is fundamentally a measure of the frequency in space of a wave, and it is measured in terms of cycles per unit of distance. It can be related to the energy of the photon by $k = E/\bar{h}c$, where k is the incident photon wavenumber, E is its associated energy, c is the speed of light, and $\bar{h}$ is a famous physics constant, governing the phenomena of the quantum world.

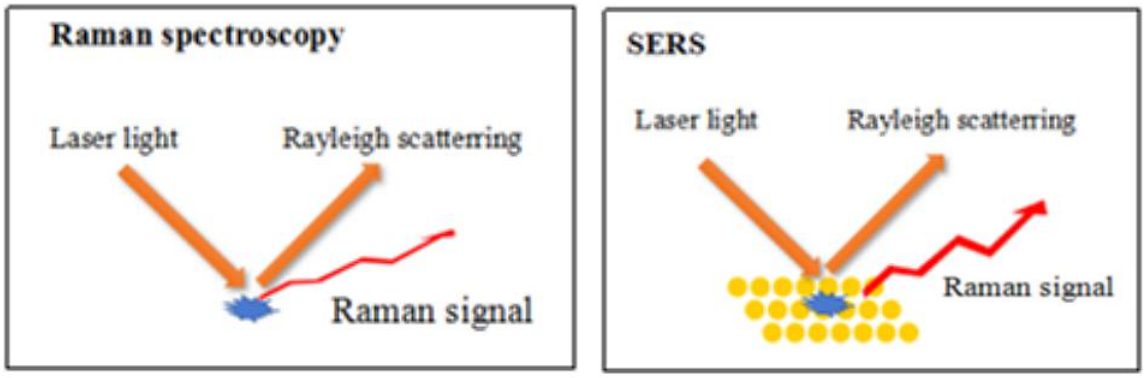

Figure 1.5: A conceptual scheme of the setup for RS. The Surface-enhanced Raman Spectroscopy (SERS) version of RS is characterized by the addition of a particular nano-structure that can enhance the Raman signal.

chemical composition of a sample, producing an overview regarding the presence, concentrations and interactions of molecules contained in a molecular system . In addition, the intrinsic safety and non-destructiveness of RS techniques allows the creation of highly sensitive portable spectrometers, such as the one in Fig.1.6, which can be easily employed as biosensing point of care in clinical areas.

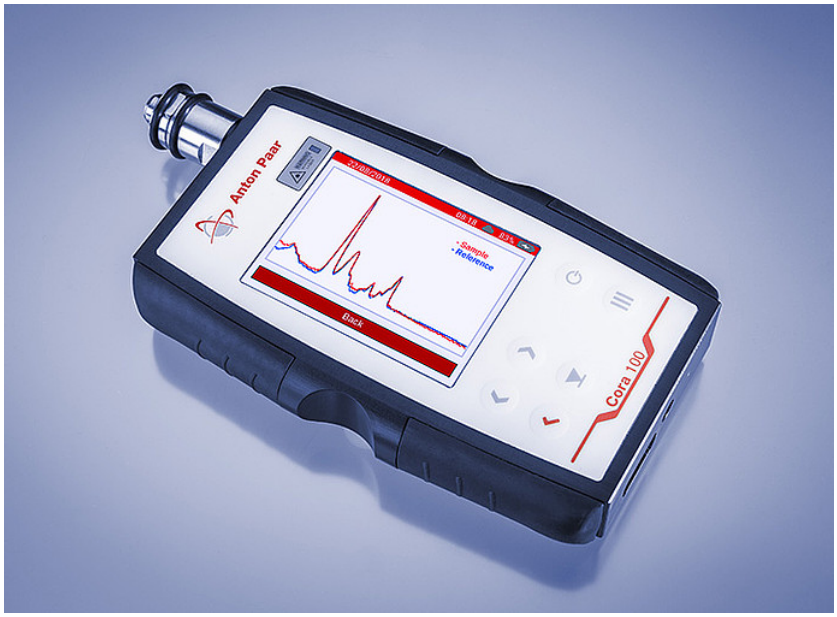

Figure 1.6: An example of a portable Raman Spectrometer available on the market.

1.3.3 RS on Saliva Samples

Biomedical applications of RS involve a variety of target samples, typically biofluids such as cerebrospinal or blood-based (mostly serum) fluids. However, the search for meaningful biomarkers, which could potentially unveil the presence or insurgence of a disease in a patient, is hindered by the invasiveness of their collection procedure, which is not even always feasible in cases of severe degenerations and therapy monitoring in late-stage patients. On contrary, in recent research, the analysis of saliva samples has demonstrated the potential of such a biofluid for the identification of the presence and relative concentrations of relevant biomarkers, making saliva one of the most promising analysis-targets, especially considered its extremely easy accessibility and the non-invasiveness of its collection.

Saliva is a complex biofluid, being composed by several different molecules, among which proteins, metabolites, carbohydrates, nucleic acids, and hormones, in an overall aqueous solution. These molecules represent potential bio-markers, since they are involved in active and passive processes of transport from oral cavity cells, salivary glands, and plasma to saliva. In addition, not only their presence can lead to potential hints for the identification of a pathological state, but also their concentrations and variations in concentration, as well as their modifications, interactions and environments, can give insights for the disease progression and response to specific pharmacological and rehabilitation therapies. Moreover, these molecules are represented in pathological state in specific

patterns that involve several immunological and physiological biomarkers, with the clear difficulty of the overall detection of all the associated levels. Hereby, the application of RS to saliva sample to obtain easy-accessible and meaningful information regarding the health state of the sample-donor has been purposed and recently tested, representing the prelude of the development of Raman-based biosensing Point of Care for the rapid and non-invasive diagnosis of several diseases.

1.4 Spectral Data Analysis and Computational Techniques Overview

RS, given its low-energetic characterization of the target, provides results which are difficult to interpret, especially when the investigated sample is a complex molecular system, such as common biofluids. Data analysis and computational techniques could, therefore, be applied to the collected spectral data to unveil new insights in the sample characterization and/or to enable the discrimination of different classes and the identification of patterns and structures in the analyzed targets.

In particular, this work focuses on the application of machine learning and deep learning techniques for the characterization and classification of spectral data generated via RS on human salivary samples. A literature review is presented in Sec. 2, and an introduction to ML and DL concepts is provided in Sec. 3. Several approaches for various tasks are implemented and tested, from the application of standard machine learning methods, such as Support Vector Machine (SVM) and Random Forest (RF), to the more sophisticated Deep Learning (DL) techniques, such as Fully-Connected Neural Networks (FCNNs) and Convolutional Neural Networks (CNNs). From a technical perspective, the learning problem is complex, given the statistically non-significant difference between the overall distribution of data from the various classes to be classified, and it is further complicated by the low volume of the available data. Exhaustive details about the experiments can be found in Sec. 4, where 1D CNNs prove to be the most suitable method for dealing with RS data. Under the data-analysis perspective, those techniques enable a rapid and accurate diagnosis of several pathologies in an automated manner, saving time and efforts both to biomedical scientists and potentially to the involved patients looking for their diagnosis results.

1.5 Medical Overview and Current Diagnosis

In this section, a medical introduction to the mentioned pathologies will be given, together with an overview of existing diagnosis techniques and a brief summary regarding the employment of RS and the data collection.

1.5.1 Amyotrophic Lateral Sclerosis

Amyotrophic Lateral Sclerosis is a neurodegenerative disease that leads to progressive and irreversible muscle atrophy. Currently, the diagnosis of ALS is time-consuming, complex, and articulated in several steps, with the clinical and neurophysiological evaluation that is often accompanied by monitoring of progression and a long procedure for the exclusion of similar neurodegenerative diseases. The delayed diagnosis strongly slows the potential development of adequate therapies and the time frame for a prompt intervention. The discovery of new biomarkers could improve the disease diagnosis, as well as the therapeutic and rehabilitative effectiveness and monitoring of the pathological progression. For this application, saliva samples had been collected from 19 patients with ALS and 10 healthy subjects, and analysed using RS. The proposed approach could result in a fast and sensitive procedure that can make more efficient the diagnostic procedure and the monitoring of therapeutic and rehabilitative processes in ALS.

1.5.2 Parkinson's and Alzheimer's Disease

Parkinson's disease (PD) is one of the most common neurodegenerative disorders occurring in elderly, associated with underlying and unrelenting neurological tissue degeneration including the inactivation of dopaminergic neurons in the substantia nigra and with the appearance of Lewy bodies made of abnormal α-Synuclein. Epidemiologically, PD is considered to be the second most relevant neurodegenerative disorder after Alzheimer's Disease (AD), with an increasing burden in aging society, genetic susceptibility, protein aggregation and by a long preclinical period in which the neuroprotective therapy should start. Usually, the PD diagnosis mainly relies on clinical motor symptoms, which occur generally decades after the pathology onset. This lag time hampers the detection of the earliest phases of the disease and the time at which the treatment with neuroprotectors drugs could have the greatest effort. The preclinical phase events have been associated to the loss of catecholinergic and cholinergic neurons and to other modification in the substantia nigra pars reticulata, suggesting an alteration in the physiological tissue environment and in biomolecules distribution several years before the diagnosis. Furthermore, an obstacle to the early PD diagnosis

is represented by the association between Mild Cognitive Impairment (MCI) and PD. Generally, only the 75% of PD clinical diagnosis are confirmed at autopsy, in most cases because the clinical signs can also occurs in other neuropathological conditions called extrapyramidal signs, parkinsonian features and parkinsonism. In this frame, researchers are focused on the definition and the characterization of a defined and measurable PD biomarker, easily collectable and able to highlight the early pathology onset, to monitor its progression and the therapeutic and motor- neuronal-rehabilitation efficacy. Due to the tissue specific pathology, most of the potential circulating biomarkers have been identified in cerebrospinal fluid and blood (serum, plasma), ranging from signal and structural protein, enzymes, fatty acids, nucleic acids and molecules of the inflammatory system. The main limitations related to many biomarkers are the invasiveness of the sample collection procedure, which limit the multiple sample collection for monitoring purposes, to the sharing of specific biomarkers with other similar pathologies (PD and AD) and the techniques used for their characterization. Several studies reported the presence of part of the listed biomarkers also in other biofluids such as urine and saliva, demonstrating the possibility to reduce the invasiveness of the sample collection. The most used diagnostic techniques are ELISA, electrochemiluminescence, Western blotting, xMAP technology, Protein Misfolding Cyclic Amplification (PMCA), mass spectroscopy, cytofluorimetry, enzymatic assays and Real-Time Quaking-Induced Conversion (RT-QuIC), besides the neuroimaging procedures used for the diagnosis (computed tomography, single photon emission computed tomography and magnetic resonance imaging). However, these techniques are time-consuming and expensive, involving complex procedures for sample or patient preparation. Therefore, a fast methodology able to determine a global biomarker specific for the neurodegenerative pathology and collectable with a minimal invasive procedure could be of crucial importance for the PD diagnosis and monitoring of therapeutic and rehabilitative efficacy. Saliva is a complex biofluid, currently considered as a non-invasive and ideal diagnostic material collectable without any discomfort in comparison with blood. Currently, different salivary biomarkers have been proposed for the diagnosis of neurodegenerative disease including AD, PD, Amyotrophic Lateral Sclerosis and Huntington disease, with correlation of the biomarker concentration in saliva with the blood levels.

1.5.3 COVID-19

SARS-CoV-2 is a coronavirus belonging to betacoronavirus genus lineage B, related to SARS-like viruses with identified differences in genetic material and surface proteins. At present, COVID-19 diagnosis is mainly based on epidemiological

history, clinical manifestations, Computer Tomography and nasopharingeal swab assay for the detection of viral nucleic acid test results. Many studies on the well characterized SARS-CoV, and theoretically translatable on the SARS-CoV-2, demonstrated the early infection of epithelial cells of the major and minor salivary gland ducts with a primary accumulation of virus bodies into saliva. With the infection progression, the viral load into saliva grows exponentially, converging into saliva different secretion coming from other districts including upper and lower respiratory tracts, nasopharingeal and blood vessels through crevicular fluid. The range of biological molecules simultaneously detectable using RS vary from proteins to lipids, nucleic acids, hormones, metabolites, cells, bacteria and viruses, obtaining information regarding their presence, concentrations, interactions, environment and mutations. The identification of a SARS-CoV-2 Raman signature carry information about the molecular characterization of the analyzed sample (saliva in this study) and when using a portable Raman easily could represent an extremely useful, fast and low cost point-of-care (POC) for the viral infection diagnosis.

2 Literature Review and State of the Art

2.1 ML Applied to Raman Spectroscopy

In this section, we will present widely diffused and emerging applications that are based on the combination of machine learning, deep learning and artificial intelligence methods together with RS or SERS. The literature review will be focused on four of the most common types of application: food and beverage related applications, the forensics domain, the study of bacteria and viruses, and, last but not least, medical diagnostics applications. For a complete review of the literature regarding ML/DL and RS, the reader can refer to [7].

Food and beverage ML has been used in the food industry in combination with RS to identify the origin of a food product, such as meat, in order to potentially avoid fraud events such as the one happened in 2013, when some meat products labelled as containing beef in parts of Europe were found to contain or made entirely of horse meat. [8] demonstrated that the leg and breast meat of turkeys and chickens can be distinguished from each other, but the same procedure can be applied to other common foods, such as margarine in butter [9], and luxurious food items, in particular caviar [10]. Furthermore, the origin of foods may also influence the nutritional values: [11] showed that milk obtained from humans, cows, buffalo, and goats can be distinguished from each other using Raman spectroscopy.

On the other hand, vibrational spectroscopy can be also employed for the detection of molecules that have been added to food. Indeed, the use of synthetic dyes, often added to foodstuffs to enhance the colour, is regulated, or even banned, in several countries. Many studies related to dyes detection use SERS coupled with multivariate analysis: the known carcinogen Sudan-1 [12]; carmine [13]; food blue, tartrazine, sunset yellow, and acid red [14].

Another possible application that exploits RS and ML is in the agriculture: the detection of molecules related to the use of pesticides and fungicides in agriculture is extremely important to the health of the consumer: [15] is an example of those tasks.

Forensics Another field that benefits from the combination of RS and ML is the forensics one. First of all, it is possible to characterize the properties of pharmaceutical drugs [16], typically finalized to the detection of illicit drugs. An example is [17], where the authors showed that oxycodone, heroin, tetrahydrocannabinol, and cocaine could be readily distinguished by a fairly tuned SVM

model, combined with PCA. Furthermore, the detection can happen also in complex biofluids: the detection of methamphetamine and heroin in saliva has been carried out in [18], and the same goes for tetrahydrocannabinol in saliva by the authors of [19], or for morphine and different fentanyls in urine.

Beyond illicit drugs detection, another areas of forensic sciences that can exploit RS is Crime scene analysis [20]: in particular, different types of information that can be obtained by analyzing a bloodstain in such a scene. Discriminating the source of the blood (animal vs. human) is an important application, presented in [21], where blood samples from 16 species of animals were compared to that of human blood through a PLS-DA model with consistent results. On the other hand, gender detection and age (range) detection is made possible by analyzing bloodstains through RS and ML, as shown in [22] .

Another branch of forensics is the Chemical forensics: it is responsible to determine the origin of compounds found at a crime scene. RS helped this domain, for example, through the determination of the synthetic route used to prepare mustard gas, a chemical warfare agent [23].

Another interesting application is reported in [24]: when dealing with explosives identification, a strategy, in contrast to the usual putting a physical barrier (glass vial) between the sample and the instrument (and operator), can involve using distance, developing a hyper-spectral Raman imaging system through the use of a telescope, which made it possible to safely acquire Raman spectra of explosives at a distance of 15m, to be then classified using a random forest.

Bacteria and viruses The analysis of individual bacteria and viruses is enabled by a variant of RS, called Raman micro-spectroscopy. In this technique, the diameter of the laser beam is typically in the range of 1 mm, comparable to the size of many bacteria. Therefore, spectral databases of bacteria can be built, keeping in mind that a large number of spectra per type of bacterium in needed considered the large degree of variability that can occur. For example, the authors of [25] compared 11 bacterial species that can potentially cause urinary tract infections, acquiring an average of 268 spectra for each species to then exploit an SVM model in a 10-fold cross-validation, reaching an accuracy rate of 92%.

When detecting bacteria, a possible way to do that consists on looking for small molecules that are secreted by the bacteria. The authors of [26] presented one possibility of ascertaining the pathogenicity of toxigenic *Clostridium difficile* in stool samples is to determine the presence of toxin A (TcdA) or toxin B (TcdB). A total of 36 stool samples and 360 Raman spectra were used to prepare the ml models, among which stochastic gradient boosting machine, SVM (linear kernel),

and PCA-LDA.

RS can be applied as well to the study of Viral infections, such as influenza with recent studies on identifying new emerging influenza viruses [27]. From a high-level perspective, blood samples acquired from patients that are positive or negative for a bacterial or viral infection can be used to generate predictive models for future patients. We only cite a few examples related to Dengue fever or Hepatitis C [28].

Medical Diagnostics As mentioned in the previous application, several studies have been performed using serum samples from patients of known infections in order to build predictive models for a possible future diagnoses. The same approach can be applied to medical disorders and pathologies: in the following, some highlights of a few proof-of-concept studies focused on identifying or classifying aspects of medical disorders are presented. The first example relates to the screening of diabetes mellitus, which typically requires that a blood sample to be taken from the patient, procedure that is often considered as an invasive approach. Therefore, a screening method using a non-invasive approach yielding rapid results has ideal characteristics: Raman spectra acquired at different sites on the body (earlobe, inner arm, thumbnail, and median cubital vein) instead of blood samples have been used to this extent in [29], where artificial neural networks were exploited resulting in a classification accuracy rate of 96% for spectra acquired at the inner arm. However, it is worth to notice that, from my perspective, a higher number of patients is required to determine how reliable this approach would be in a proper clinical setting.

SERS has also proven its potential as a point-of-care test for eye disorders: the study in [30] deals with cataracts, and oxidative stress-induced age-related macular degeneration and diabetic macular edema.

One of the most interesting applications of Raman,SERS and ML is the medical condition given by cancer. In recent years, cancers including liver [31] or thyroid have been investigated through the use of Raman/SERS and multivariate analysis. The identification of the presence of cancer markers within serum is one of the most important diagnostic tools: examples of small molecules that diagnostic approaches can focus on are metabolites (i.e. tryptophan,phenylalanine, proline, valine, adenine, thymine) in serum.

Exosomes, larger bio-materials w.r.t. serum biofluid, can also be taken in consideration for investigating cancers. Being nanoscale membrane vesicles that are released by a variety of cell, they can found throughout various biofluids, and their chemical and biochemical are strictly related to their cell-type of origin: This characteristic has the potential for distinguishing not only based on cancerous vs.

noncancerous origins, but also on the type of cancer, as discussed, for example, in [32].

2.2 DL Applied to Raman Spectroscopy

DL architecture has been recently applied to RS. Up to now, the main works dealt with classifying spectral databases of minerals, as in [33, 34], using CNNs. Other examples of DL applications to Raman can be seen in [35], where the authors built a convolutional system able to classify artificial mixture of chemical compounds. The most important and inspiring work for this thesis is that in [36], where a 1D ResNet was applied to spectral data of bacteria to perform a classification in proper clinical settings.

2.2.1 Our Contribution

In comparison with the related literature described above, our main contributions are three-fold:

- We integrate the various steps needed in the application of ML/DL techniques to spectral data (spectra filtering, preprocessing, augmentation, preparation for training) in a unified pipeline where most of its components are easily automated.
- We approach a patient-wise classification in clinical settings, where almost (almost is the key-word) no statistically-significant differences are observed among data belonging to different classes, resulting in a harder problem with respect to a typical Raman application such as those presented above.
- In contrary with all the literature we are aware of, we apply a Hyperparameter optimization step to further refine the results of our DL models. The typical choice in the literature is, instead, to provide a DL architecture whose parameters configuration is either standard, driven by domain knowledge and best practices, or the result of a trial-and-error approach.
- We test the transfer learning approach for spectral data in a low-data-volume situation, with interesting and promising results that are expected to improve the classification quality once data availability is increased (especially in the pretraining phase, condition which is reasonably achievable).
- The most similar and challenging work with respect to ours is undoubtedly patient-level Bacteria classification performed in [36]. However, it is worth noting that the authors had tons of data available (thousands of patients),

when compared to our case. This enabled them to exploit more structured DL architectures (such as Resnets) to achieve their impressive results. However, their approach is not suitable for our case, where data scarcity tends to be dominant.

3 Elements of Machine learning, Deep Learning, and Optimization

This section is intended as a theoretical introduction to the ML and DL concepts, to provide the reader that might be new to ML concepts with a better understanding of the investigation practices and results of this work. A reader that is expert in ML could easily skip the current section, but whether she/he is convinced to read this introductory, some "expert corner" notes will be provided to alleviate possible boredom.

The Deep Learning technology applied in this thesis is derived from the Machine Learning class of algorithms. To understand how DL works[1], let me introduce the basic working principles of ML. For a wider and complete perspective on ML, the reader can refer to [40, 41].

3.1 Intro to Machine Learning Basics

In the following, I will describe what a (machine) learning algorithm is, how it can be applied to a learning problem through a dataset and what are the main challenges related to the entire process. We will briefly discuss the components of a typical ML algorithm, starting from its settings, introducing the role of hyperparameters, and of the optimization process which, under the hoods, allows the whole system functioning. An overview of algorithm categories will be presented, distinguishing between supervised and unsupervised learning. Finally, I will give a brief theoretical introduction to the specific ML algorithms exploited for the experiments carried out during the thesis.

3.1.1 The Learning Algorithm

A learning algorithm is mainly characterized by its capability of learning from "past" experiences, elaborating examples provided in the form of "encoded" data. Fig.3.1 illustrates an example of a typical learning algorithm structure.

A formal definition, in the vein of [42] is the following.

Definition 1 (Learning Algorithm)**.** *An algorithm is said capable of learning from a set of Experiences E with respect to a Task T and performance measure P, if P improves with E.*

[1]Note: understanding WHY Deep Learning works is still a super-hard challenge in the research area! [37, 38, 39]

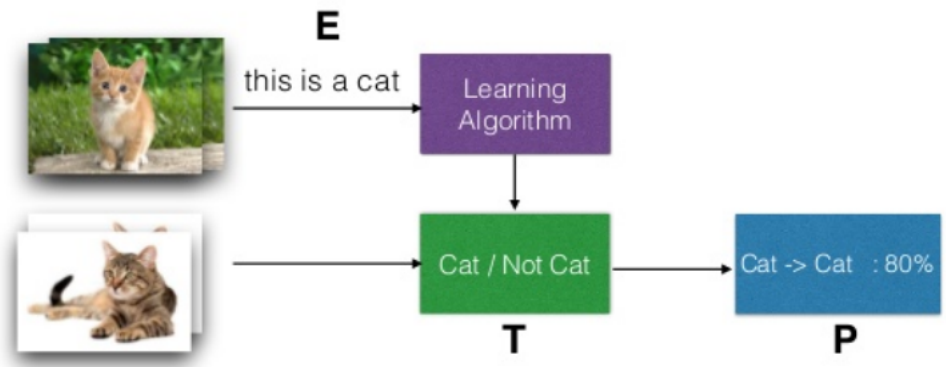

Figure 3.1: An example of a learning algorithm that learns from experience (example of cats) to accomplish a task based on distinguishing between what is a cat and what is not.

3.1.1.1 The Task It is worth to notice that the learning process is not itself a task, as meant in Def.1, whereas an example of a proper ML task is the implementation of an A.I assistant to interpret human speech.

There are different classes of tasks T: classification problems, regressions (function approximations), transcriptions, translations, structured output processes, anomaly detection, data synthesis, missing values-related approaches, audio/image denoising or super-resolution, probability density function estimation, and many others.

For the current work, the main task will be classification. To accomplish such a task, the learning system is asked to specify which of k categories some input data belong to, or more formally:

Definition 2 (Classification Task)**.** *During a classification task a computer program is required to (learn and) produce a mapping function* $f : \mathbb{R}^n \rightarrow \{1, ..., \}$ *such that* $y = f(\vec{X})$*, where an input vector* $\vec{X} \in \mathbb{R}^n$ *is mapped to a category identified by the label* $y \in \{1, ..., k\}$*.*

We usually will refer to **Binary Classification** when $k = 2$, or **Multi-class Classification** when $k > 2$.

3.1.1.2 The Performance Measure The capability of a program to perform T is assessed by a performance evaluation: a performance measure P is, therefore, required to quantitatively measure the output quality. P is very closely related to the specific T class. For classification tasks, the most frequently encountered P is the **Accuracy** ($acc \in [0, 1]$), measuring the proportion of correctly classified outputs. Conversely, the **Error Rate** ($e \in [0, 1]$) measures the percentage of wrongly predicted outputs.

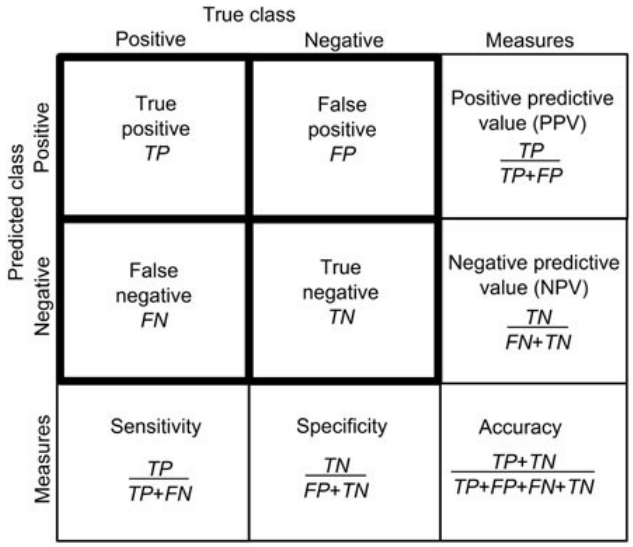

Figure 3.2: A general Confusion Matrix. Note that the performance measures such as Precision, Recall, Sensitivity, Specificity are derived directly from the matrix.

The choice of P is sometimes not straightforward, since it is strictly related to a case-by-case desired scenario for a system. In biomedical applications, such as in a diagnostic system like our proposed methodology, accuracy can be still taken in consideration, but only along with other quantities, such as the coupled **Precision-Recall** and **Sensitivity-Specificity**, that gives wider information about the process results.

In a binary classification task, Precision is the fraction of correctly positively classified samples among the entire set of positively-classified examples (which could also contain misclassified samples from the negative class). Recall is the fraction of correctly positively classified samples among the set of truly positive samples.

On the other hand, Sensitivity, the extent to which true positives are not overlooked, and Specificity, the extent to which true negatives are correctly predicted, could be derived from a Confusion Matrix. A confusion matrix is a matrix collecting the results of the classification in terms of True Positives (the examples from the positive class correctly classified) , True Negatives (the examples from the negative class correctly classified), False Positives (examples from the negative class wrongly labelled as positive ones) and False Negatives (examples from the positive class wrongly labelled as negative ones. An example of Confusion Matrix and the interpretation of its values can be found in Fig. 3.2.

In medical diagnosis, a sensitive test is such that it rarely overlooks a true positive, whereas a specific test rarely assigns a wrong label to a negative example.

Depending on the desired results, one could often choose whether to privilege either Precision or Recall (as well as either Sensitivity or Specificity). Often, an ideal system has performance metrics that represent a trade-off between the aforementioned cases, depending on the desired target.

3.1.1.3 The Experience To fully understand the Def.1, a brief discussion of what Experience is can be helpful: it is the raw matters that fuels the learning process. A collection of examples is said to form a dataset, the basic substance of Experience. By means of data, a ML algorithm is allowed to make an ideally complete experience of a certain domain (if the dataset is itself complete, that is it contains enough information to guarantee a sustainable accomplishment of a task).

ML algorithms can be divided into two main classes, the Supervised or Unsupervised ones[2], depending on what kind of experience they are allowed to have during the training process. Supervised learning is characterized by the association of the input features with a set of annotations that provide examples of the results of correct learning. For instance, labels are the supervised components of a classification task, providing the information of which class an input belongs to, that is the ground-truth to be compared with the results of this specific learning problem. When no such annotations are provided together with the dataset, the problem is said to be Unsupervised, and a learning algorithm can only exploit the set of features to accomplish a certain task, without access to further information. An example of Unsupervised Learning is clustering.

It is worth noticing that the same problem (say, classification) can be both tackled in a supervised (i.e. considering labels) or in an unsupervised (just by ignoring labels) manner. Even from a formal perspective, the joint probability for a vector $\vec{X} \in \mathbb{R}^n$ to belong to a certain class based on its feature set $\{X_1, ..., X_n\}$ can be decomposed as

$$p(\vec{X}) = \Pi_{i=1}^{n} p(X_i | X_1, ..., X_{i-1}). \tag{1}$$

It indicates how the unsupervised problem of finding $p(\vec{X})$ can be solved by explicitly modelling it into n supervised learning problems $p(X_i|...)$ or, conversely, the supervised problem of inferring $p(y|\vec{x})$ can be modelled into the unsupervised learning of the joint probability distribution $p(\vec{X}, y)$, thus obtaining

$$p(y|\vec{X}) = \frac{p(\vec{X}, y)}{\Sigma_{y'} p(\vec{X}, y')}. \tag{2}$$

[2]A particular case of a crossover between the two traditional classes is embodied by the recently introduced Semi-Supervised learning algorithms.

Therefore, no net distinction between the two classes can be formally stated, being the difference only a matter of the chosen approach.

3.1.2 Generalization in Data Fitting

3.1.2.1 Generalization One of the main challenges in developing a ML model is the generalization. During the training phase, the algorithm modifies its inner structure (its parameters) to better fit the input data, in order to have good performances on the task with respect to a performance measure P.

However, a good fit of the input data during training is not sufficient, and the model must be able to perform well on new, previously unseen data. This is often referred to as **generalization**.

3.1.2.2 Validation Techniques To assess this, a common ML practice is to divide the dataset into a training set (typically composed of 70% of the dataset instances), which is accessed by the ML algorithm during the training, and a test set (usually the remaining 30%), whose examples are used to test the model "out-of-the-box" performances. Such a splitting technique is called **Hold-Out**. To provide a confidence interval on a performance measure P, one can iterate the Hold-Out procedure with different (random) splittings, obtaining an **Iterated Hold-Out**.

A more sophisticated validation technique is the **K-Fold Cross Validation**: the dataset is partitioned in K disjoint subsets (folds), and the training is repeated K times using at each iteration a different fold as the test set and the remaining ones as training set. Performances are then averaged (and the std. dev. is calculated) across the K distinct results, providing a more reliable confidence interval for the metrics.

3.1.2.3 Theoretical Hints Those procedures are enforced by some **Statistical Learning Theory** concepts: assuming that the data are generated by a process (data-generating process) and that the examples in the dataset are $i.i.d.$, then such a process can be described as a probability distribution over a single sample, p_{data}. This guarantees that there is a relationship between the training and the test set: they share the same p_{data}. Therefore, if $\mathbb{E}[e]$ is the expected error, then:

1. for a randomly selected model $\mathbb{E}[e_{training}] = \mathbb{E}[e_{test}]$,
2. for an already trained model $\mathbb{E}[e_{training}] \leq \mathbb{E}[e_{test}]$.

Such reasoning gives us two clear signs of what a good algorithm is expected to do:

1. make $e_{training}$ small,
2. keep $|e_{test} - e_{training}|$ small.

3.1.2.4 Capacity When these two principles are not respected, problems can arise. These problems are usually referred to as, respectively, **Underfitting**, when the model is not able to fit the training data well enough, and **Overfitting**, when it poorly generalize over the test set. Underfitting and Overfitting are closely related to the **model capacity**. From a high-level perspective, capacity is a measure of the fitting ability of a model with respect to a wide variety of functions (**representational capacity**). Overfitting problems can sometimes be handled by limiting the capacity of a model, therefore allowing it to represent only simpler (smoother) functions. An example of limited capacity is the linear regression model, where a bound on the **hypothesis space** restricts the possible solutions (in terms of functions) to the class of linear functions. It's worth noticing that training and test errors vary along with the dataset size: we can often (ideally always) reduce the train-test generalization gap by gathering more training examples.

Expert Corner For an expert reader, it is interesting to notice that Deep Neural Networks (DNNs) (introduced in 3.2) are believed to do this capacity limitation autonomously [43, 44], by learning across the spectrum of the possible solution functions the low-frequency ones first (thus providing smoother functions as solution candidates at first). However, as the training goes on, higher and higher functions are represented, thus leading again to overfitting when the training phase is too long.

Another note for a knowledgeable reader: the relation between Overfitting and model capacity is closely related to the long-standing statistical problem of the **Bias-Variance Trade-off**, where, informally, a more complex model (low bias) is said to be prone to overfitting (high variance). However, super-recently this principle, which is believed to be valid for most of the ML models developed so far, has been shown to be in contradiction with some results obtained with Neural Networks, where high complex Deep Neural Network models still greatly perform on the test set [39, 45, 46].

3.1.2.5 No Free Lunch and Regularization In principle, different ML algorithms have different generalization properties when compared against the same task. In particular, there is no universally better algorithm for every task. This statement can be formalized in the following theorem, see [47] for its proof.

Theorem 1 (No Free Lunch Theorem)**.** *Given a pair of algorithms* a_1*,* a_2*,*

$$\sum_f p(d_m^y|f, m, a_1) = \sum_f p(d_m^y|f, m, a_2) \tag{3}$$

where d_m^y is an ordered set of size m of the cost function values y associated with the input vector $x \in X \subseteq \mathbb{R}^n$, $f : X \to Y$ is the function to be optimized and $p(d_m^y|f, m, a)$ is the conditional probability of obtaining a given sequence of cost function values from m runs of the algorithm a over f. The same could be rephrased as:

Theorem 2 (No Free Lunch Teorem Rephrasing)**.** *Given finite sets* $X \subseteq \mathbb{R}^n$ *and* $Y \subseteq \mathbb{R}^m$*, assume that* $\tilde{f} : X \to Y$ *is chosen randomly according to a uniform distribution of the set* S^V *of all possible function* $f : X \to Y$*. For the problem of optimizing over the set* V*, no algorithms perform better than blindsearch.*

The No Free Lunch Theorem suggests that ML models should be designed to perform well on a specific task. To do so, many tricks could be exploited. In general, one could specify preferences on how the learned function should be (i.e. which "class" should it belongs to). Informally, those preferences correspond to introducing some **regularization**. In general, regularization can be defined as in [48]:

Definition 3 (Regularization)**.** *Regularization is any modification to a learning process that is intended to reduce its generalization error but not (directly) affecting its training error.*

The control of capacity, mentioned in the previous paragraph, is an example of an implicit regularization. Explicit regularizers directly influence the cost function to be optimized, by typically adding and extra term together with the loss component. An example of explicit regularization is the **weight decay**, where in addition to the standard loss function $y(x; \Theta)$ (i.e. maximum likelihood for linear regression) an extra term $\Omega(w)$ is added (for linear regression, $\Omega(w) = w^T w$), expressing the preference towards smaller squared L^2 norm for the weights.

Expert Corner For an interested reader, although in Deep Neural Networks weight decay has been largely used, recently some hints suggested that Neural Networks autonomously tend to prefer a solution with small (sometimes *nuclear*) norms [39, 49].

Another interesting example demonstrates that the difference between implicit and explicit regularization can be subtle and not net: Dropout [50], a largely employed regularization trick for DNNs, has been proved to induce an explicit regularization over the loss function in Deep Linear Networks [51, 52, 53].

3.1.3 Parameters and Hyper-parameters

From an over-simplified, high-level perspective, during the training phase a machine learning algorithms adapt its inner structure to better fit the input data producing the desired results for a certain task. In practice, this happens by optimizing a loss function $f(X, \Theta)$ with respect to the parameters Θ: eventually, a set $\tilde{\Theta}$ of parameters are selected as a good (enough) solution, and the optimization ends. However, the proper Θ are not the only parameters in a ML model: some others exist, which are not directly tuned during the learning process. Those values which can not be learned straightforwardly are called **hyper-parameters**. In general, hyperparameters represent settings that allows one to control some behaviours of the algorithm. Examples of hyperparameters are the degree of polynomial fitting in a polynomial regression, or the λ value in the lasso [54] or ridge [55] regression. Since their values are not selected by the learning process, their tuning (that is, the identification of suitable values) is often a manual, time-consuming challenge. In general, different choices of hyperparameters during the training lead to different performances on the test set. Therefore, a strategy is required to carefully and suitably make a *good* choice: a validation set (a smaller partition of the training set) is usually exploited to evaluate the effects of hyperparameter changes. In the last years, various algorithms for automatic **hyperparameter optimization** (**HPO**) have been proposed: the result is a nested learning problem where the hyperparameters are the learning objectives. Examples of HPO algorithms will be presented later in Sec. 3.2.6.

3.1.4 Common Machine Learning Algorithms: Implementation Hints

In this section, I'd like to introduce some high-level concepts regarding the working principles of some ML algorithms. Of course, we will focus our attention over the algorithms used later in the experimental section of this thesis: the Support Vector Machine (SVM), the Random Forest (RF) and the Extreme Gradient Boosting (XGB). In particular, we will refer to their implementation for a classification task.

3.1.4.1 Support Vector Machines The general idea of an SVM binary classifier is based on feature "spatial" separation: the goal is to separate the two classes by making the marginal gap among them as large as possible. A graphical example is reported in Fig.3.3. In the linear version of SVM, the predictions are driven by the linear function $f(x) = W^T X + b$, that can be re-written in terms of the dot product between single input features:

$$f(x) = b + \sum^{\mathrm{m}}{}_{i=1} \alpha_i X^T X^{(i)} \tag{4}$$

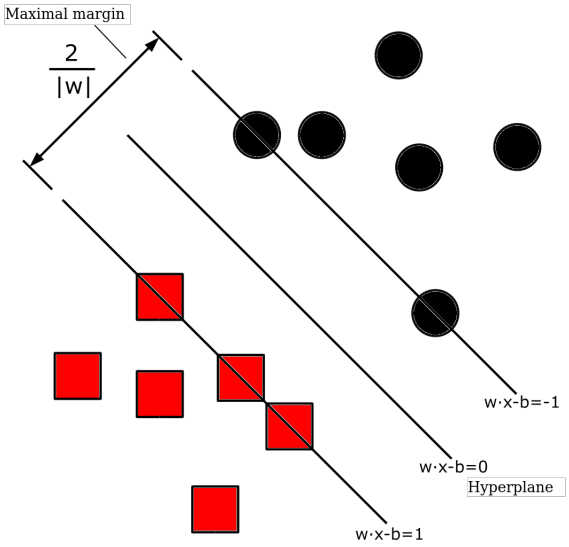

Figure 3.3: 2D SVM Classification diagram: the SVM identifies the optimal hyperplane that not only divides well the two classes, but also maximises the marginal gap among them, through each class' support vectors. Figure taken from [5].

where α, the vector of coefficient, is learned along with the optimization process. SVM can efficiently perform non-linear classification by means of the **Kernel Trick**: features are mapped to a typically higher-dimensional space (where they become linearly separable!) through a mapping function ϕ associated with a special function, called **kernel function** $k(X_1, X_2) : X \times X \to \mathbb{R}$ that maps dot products (such as those in Eq.4) to a multiplication:

$$k(X_1, X_2) = \phi(X_1)\phi(X_2). \tag{5}$$

Furthermore, many kernels are efficiently implemented from a computational perspective. A well-known kernel function is the Gaussian Kernel, also known as **Radial Basis Function** (**RBF**). An intuition of kernel mapping is visible in Fig.3.4.

Expert Corner For the interested reader, it is worth to notice that SVM training algorithms tend to learn a sparse α vector, mostly composed by zeroes. In this way, when a new example has to be classified, the kernel function is evaluated only for the training examples $X^{(i)}$ that are associated with a non-zero α_i. These features are known as **Support Vectors**.

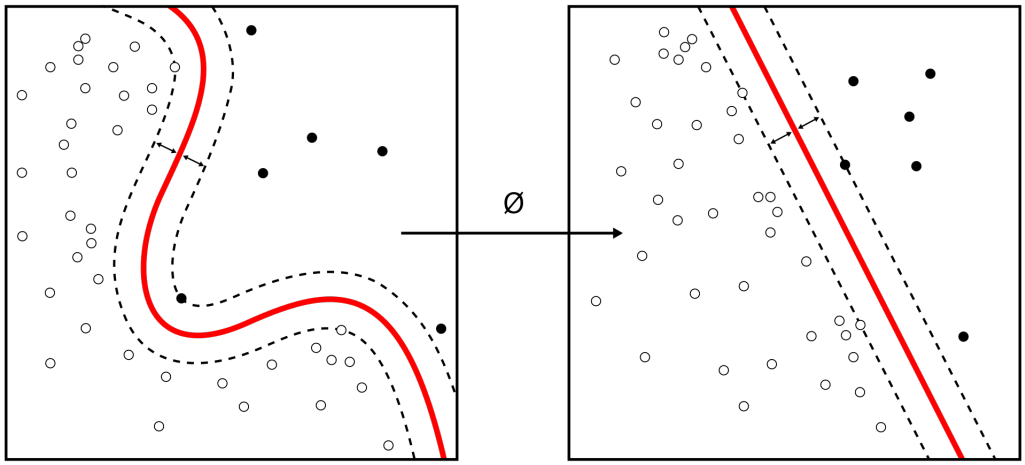

Figure 3.4: The kernel-related feature map Φ mapping a non-linearly separable low dimensional problem (right) into the 2D representation of a linearly separable higher-dimensional problem. Figure taken from [5].

3.1.4.2 Decision Trees and Random Forests Another famous ML algorithm is the **Decision Tree** (DT) classifier [56]. The general idea of DT is to partition the feature space into non-overlapping sets that are further processed and split up along the learning process following a tree-like structure to create a mapping between points in certain input regions and outputs. The detailed mechanism is described in Fig.3.5. Random Forest (RF) is an ensemble ML method consisting of the aggregation of many Decision Trees (i.e. via bagging), reducing and controlling the overfitting over the training set which is typical of a plain Decision Tree.

3.1.4.3 Boosting Machines and XGB Boosting Machines were born from the ideas of Kearns and Valiant [57, 58], who were trying to answer with their research a specific question: "Can a set of weak learners create a single strong learner?". Informally, a weak learner can be seen as a ML algorithm whose performances are barely better than random decisions. Conversely, a strong learner can be seen as a fully trained ML algorithm with good overall performances.

A boosting algorithm is a meta-algorithm (that is, an algorithm that sort of organizes other algorithms in a common structure) that turned out to be the (affirmative) answer to the mentioned research question. From a high-level perspective, a boosting algorithm builds up a strong learner by iteratively adding weak learners to an ensemble model. At each step, a single weak learner l_n is added to the "ensemble function" L_n after being trained to predict the variation of the previous L_{n-1} predictions from the ground truth. Fig.3.7 informally shows the ensemble learning curve during the training. Therefore, the predictions of the

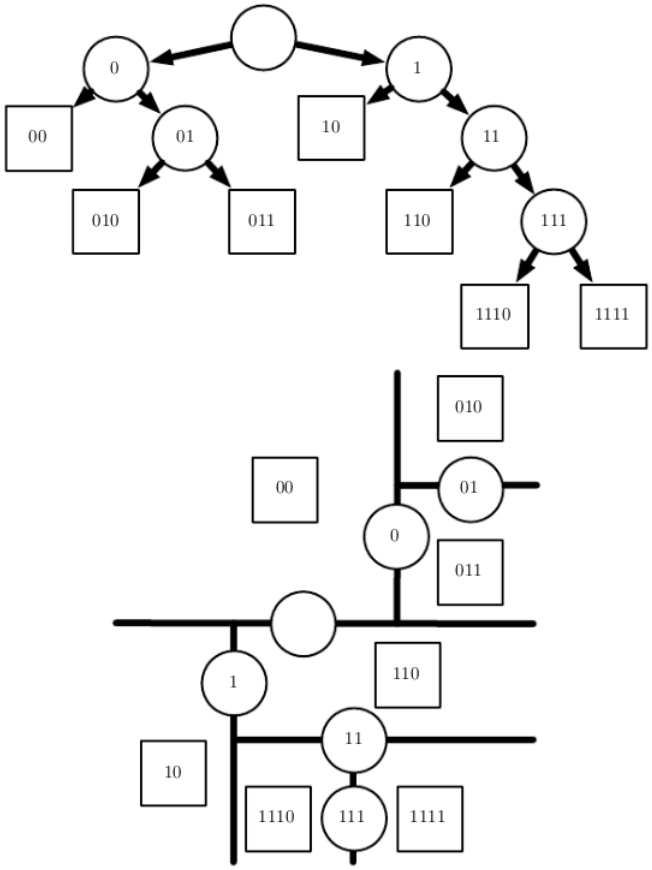

Figure 3.5: Diagram of the splitting mechanism of DT. In the top, each input node of the tree decides whether to send the input example to the node (0) or (1). The next node decides a new split either towards a new node (the classification goes on) or to a leaf (the classification ends), and so on. Nodes are represented as circles, leaves as squares. In the bottom diagram, a graphical representation of the 2-D real plane $\mathbb{R}^2$ with decision boundaries drawn from a DT, partitioning the feature space according to a certain mapping between the input (feature regions) and the output (classification leaves). Figure taken from [48].

ensemble model

$$L_n(X) = \sum_{i=0}^{n} l_i(X) \tag{6}$$

are ameliorated (less biased) step-by-step, and, at each step n, its performances tend to improve: $P(L_n) \geq P(L_{n-1})$.

In principle, the boosted ensemble can be constructed from any type of classifier. The most common implementation exploits DTs as weak learners. In particular, the Gradient Boosting Machine (GMB) is a decision-tree-based boosted model that optimizes the ensemble loss function via an optimization method called "Gradient Descent" (GD). GD is a famous method widely used in

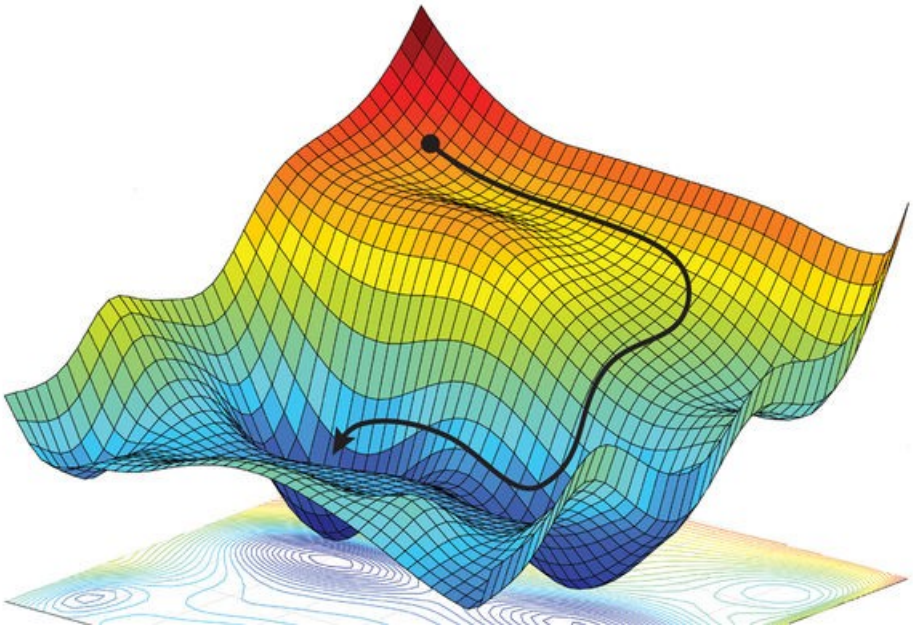

Figure 3.6: Example of a typical Gradient Descent optimization: from the starting point, the algorithm "follows" the gradients to reach a minimum.

ML and even more in DL. GD will be presented in details in Sec. 3.2.3. Fig.3.6 illustrates the high-level principle of GD.

Intuitively, GBM tends to iteratively lower its bias during the training process. However, according to the well-known "Bias-Variance tradeoff" principle, this should theoretically bring it to suffer from high variance and overfitting. Some regularization has therefore been introduced, such as the incremental shrinkage, where a multiplication coefficient λ, usually called Learning Rate, modulates the role of the newly added weak learner to the overall predictions. Eq.6 becomes then

$$L_n(X) = L_{n-1}(X) + \lambda l_n(X), \tag{7}$$

where $\lambda \in (0, 1]$.

Another form of implicit regularization can be added by implementing stochasticity in the GD algorithm. Stochastic Gradient Descend will be mentioned later in Sec. 3.2.3.1. One of the most famous implementations of GBMs is the Extreme Boosting Machine (XGB), with APIs covering many programming languages such as Python, R, Scala etc.

3.2 Deep Neural Networks

In this section, we will go through an extensive exploration of the basis behind the rise of Deep Learning, both from an introductory and historical perspective

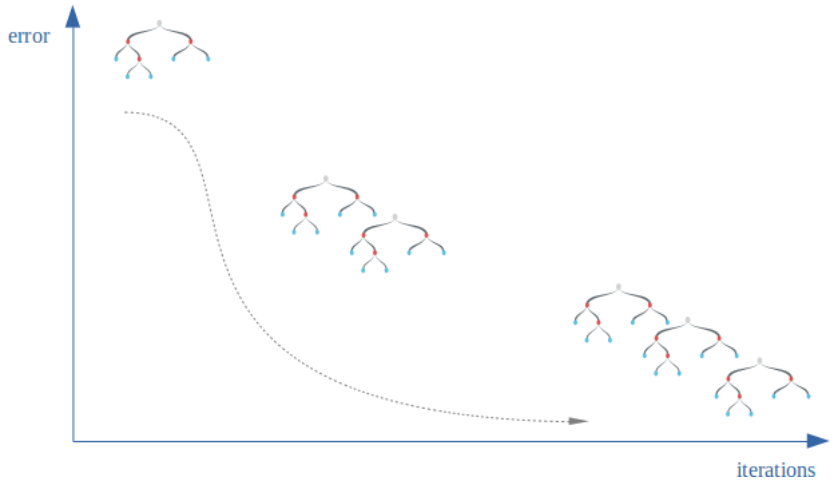

Figure 3.7: GBM Error representation as a function of the iteration step n. Adding weaker learners trained to correct the wrong predictions at the previous step increases the performances diminishing the ensemble classifier bias.

(Sec. 3.2.1), through a formal presentation (Sec. 3.2.2), and considering the scenario under the lens of optimization (Sec. 3.2.3). After clarifying the main working principles of DL, we will touch some more advanced topics, such as the key points related to Transfer Learning (Sec. 3.2.7) and some of the main application challenges of this research area (Sec. 3.2.8).

3.2.1 Introduction and Historical Notes

DL has its basis on the work of Alexey Ivakhnenko and Lapa (1967) [59]: they were able to conceptualize a supervised, deep, feedforward, working learning algorithm known as multilayer perceptrons. A few years later (1971), a deep network with eight hidden layers was trained successfully trained [60]. Similarly, Kunihiko Fukushima (1980) introduced the neocognitron [61], a hierarchical, multilayered artificial neural network applied to Computer Vision (handwritten character recognition and other pattern recognition tasks).

However, the term Deep Learning was introduced by Rina Dechter only a few years later, in 1986 [63]. While more historical notes are reported in Fig.3.8, let us start our introduction by analyzing this terminology and the meaning of *Deep Learning*. We already encountered the *learning* concept starting from Sec. 3.1 and 3.1.1: what DL learns? DL algorithms can be thought as function approximators, learning correlations patterns between the input features and the desired output by approximating a certain mapping function. On the other hand, the term *deep* is relatively new and demands explanations. It is not by chance that it appears also in the terminology *Deep Feed-Forward Neural Network*. Let us, therefore,

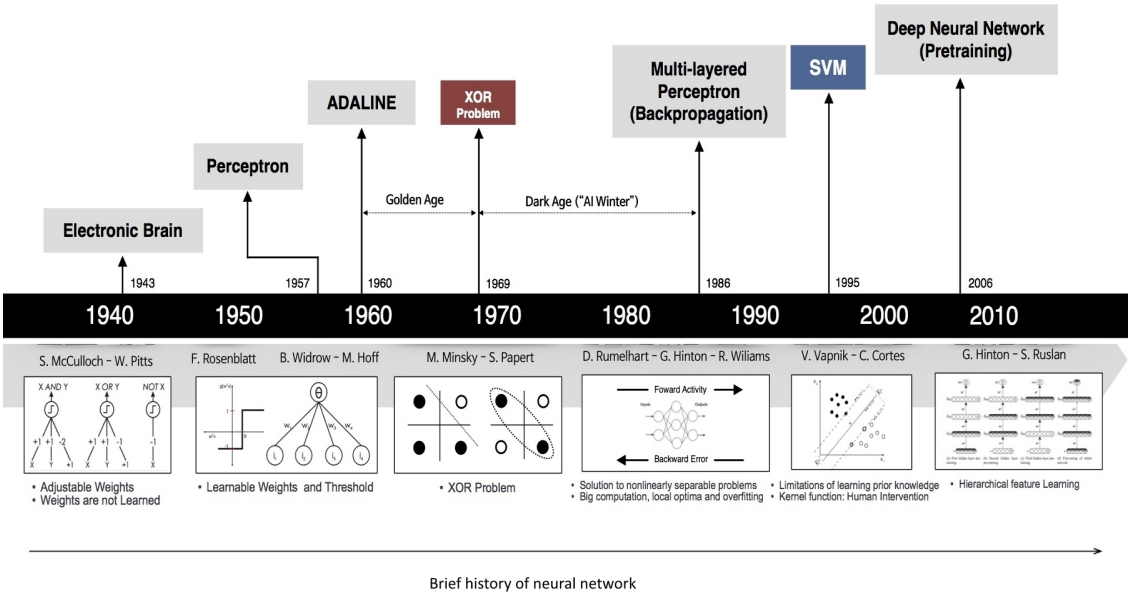

Figure 3.8: Timeline of DL development. Figure taken from [62].

focus on those terms, gradually:

1. **Network**: Deep Neural Networks are *networks* since they can be represented as the composition of several functions along with a directed acyclic graph. Fig.3.9 gives a high-level idea of this concept. Each node (circle) in the graph indicates a variable. Operations are introduced to have a formal view of the graph: a computational graph is associated with a set of allowable operations, and from the association of those operations more complicated functions can be represented.

2. **Neural**: Deep Neural Network models are biologically inspired by the structure that the human brain is believed to have. In this correspondence, we call neurons the main computational units in the aforementioned graph. However, note that the biological correspondence holds only from a high-level, informal perspective: the computational mechanism of Neural Networks has different characteristics with respect to the much more complex human brain.

3. **Feed-Forward**: following the biological view, each neuron is typically connected with other neurons. This makes it possible for the information to flow through the many components of the graph. As the term *feed-forward* suggests, in the basic, standard implementation of a Deep Neural Network the flow has a unique direction, from the input X up to the mapped results

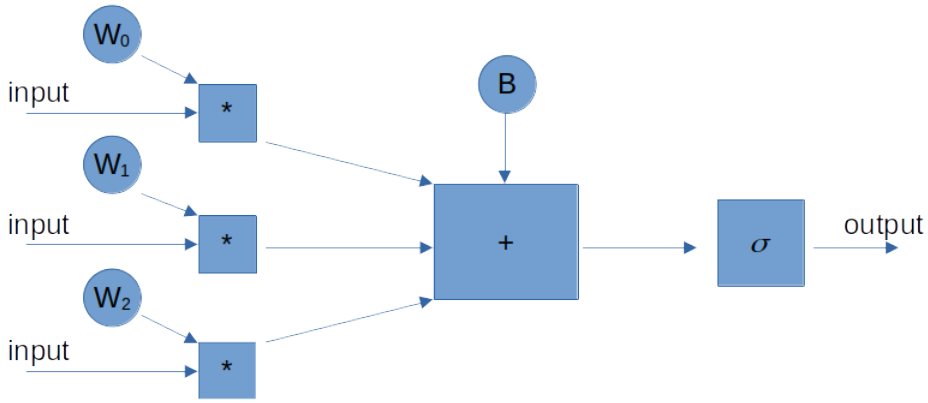

Figure 3.9: Representation of a neural network as a computational graph, where parameters (circles) are combined according to certain operation (squares) to provide the output.

$\phi(X)$ [3].

4. **Deep**: eventually, this term refers once again to the typical model structure: as reported in Fig.3.10, the neurons are disposed among layers, and each layer makes a sort of "approximation step" to form the final mapping

$$f(X) = f^{(n)}(f^{(n-1)}(...f^{(2)}(f^{(1)}(X))...)), \tag{8}$$

where n represents the depth of the network (that is, the number of layers). In modern NNs, n tends to be large, therefore we talk about Deep NNs.

Summing up, from a high-level perspective a Deep Neural Network can be seen as a structured graph where the main computational units (neurons) are disposed in connected layers which performs multiple approximation steps mapping the input X to the desired output $f(X)$ in the forward phase of the information flow.

Deep Learning vs Machine Learning What really distinguishes DL algorithms from ML ones is the ability of **Feature Learning**. In common ML models, features (that is, suitable representations of the information contained in the input data) are manually engineered: for example, before applying an SVM algorithm to a high-dimensional dataset, a dimensionality reduction step, i.e. applying Principal Component Analysis (PCA), is usually performed: this not only enables a more efficient training by reducing the dimensionality, but sort of highlights important information in the data by reducing redundancy.

[3]However, feedback mechanisms can be implemented, such as those in Recurrent Neural Networks.

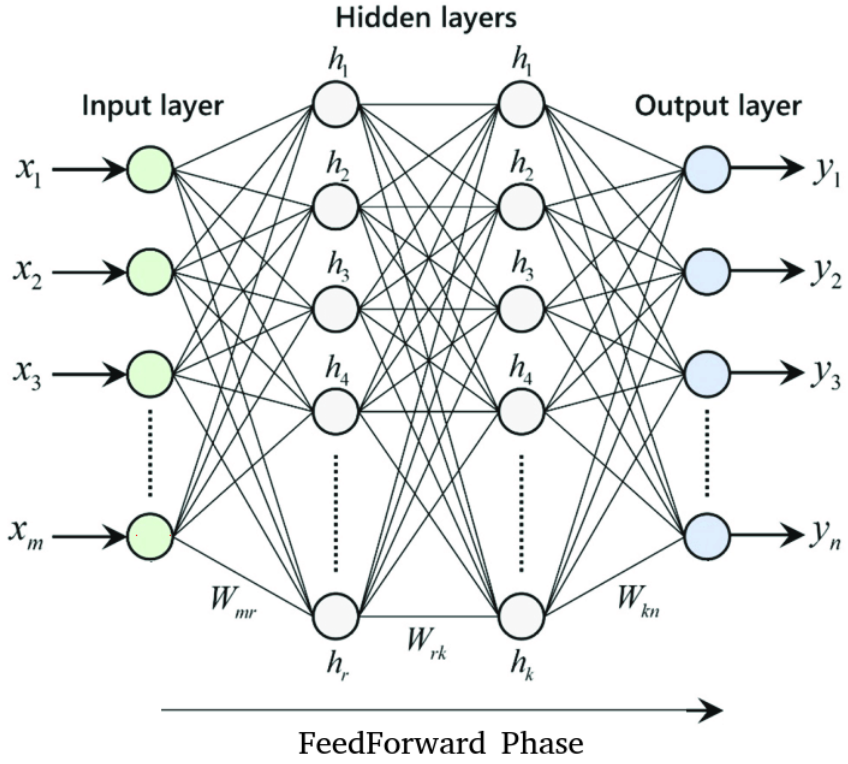

Figure 3.10: Diagram of a neural network: the input layer is responsible to linearly map the input in a predefined shape to prepare the proper input for the first hidden layers. Hidden layers then process the information in a non-linear way, producing the hidden representations h_i. Eventually, the Output Layer produces the outputs in a format suitable for the given task.

As a result, this operation can be seen as a feature engineering step. Feature engineering is also very widely applied in Computer Vision and Signal Processing in general [64].

A formal example of feature engineering is the following. Considering a linear model, it is obviously affected by the incapability of fitting non-linear function well enough. However, linear models can be extended to represent non-linear functions $f(X)$ by transforming the input X via a suitable map ϕ

$$y = f(X) = \phi(X)^T W, \tag{9}$$

and the problem is solved.

To obtain ϕ, several options are usually available:

1. **Kernel Trick**: this option uses a very generic map ϕ, such as RBF kernels, and it relies on the fact that ϕ maps into a higher-dimensional space, where the problem is much more likely to be linearly separable. However, while

this allows a good enough fitting capacity, the generalization over the test set is not guaranteed to improve automatically.

2. **Hand-crafted Features**: as mentioned before, the feature engineering step is often an alternative. However, this requires expertise, domain knowledge and can be time-consuming, rarely automatable.

3. **Feature Learning**: this option refers to the alternative introduced by DL. During the training process, a DL algorithm not only aim to accomplish the given task, but also to learn a suitable feature representation ϕ. In particular, when considering the initial example, the non-linear function $f(X, \Theta, W)$ can be modelled as

$$f(X, \Theta, W) = \phi(X, \Theta)^T W, \tag{10}$$

where the parameters Θ of ϕ can be learned during the training. It is worth to note that prior domain-related knowledge can still be incorporated in the learning process by restricting the class of possible ϕ mappings to a specific family $\tilde{\phi}$ that is known to be suitable.

3.2.2 Modern Deep Feed Forward Neural Networks

This section represents a more formal explanation of the working principles of DNNs: the general concepts of a Gradient-based learning algorithm (3.2.2.1) and the description of a typical Deep Neural Network architecture (3.2.2.2).

3.2.2.1 Following the Gradients to Learn In this paragraph, we will introduce Gradient Descent (GD), the optimization process that allows the under-the-hood actual functioning of a Deep Neural Network (as well as of some other GD-based ML algorithm, such as the GBM introduced in Sec. 3.1.4.3).

First of all, let us recall the general structure of a ML model. Almost every learning algorithm is based on the minimization of a specific loss function, which is usually directly correlated with the performance of the model on a given task (the lower the loss, the higher the performances). While in many machine learning algorithm the loss minimization is a convex problem (i.e. SVMs and linear regression) where solutions are found in closed forms or by approximately solving some (possibly vast) linear equation systems, the Deep Neural Network structure introduces non-linearity in the learning process, giving up to the property of convexity.

In general, loss minimization, also called optimization, can be done in many ways: for example using heuristics or meta-heuristics, evolutionary algorithms,

or, in the worst case, by brute force methods. However, some losses have an important mathematical property: they can be expressed analytically and are differentiable.

Indeed, the most common training algorithm for Deep Neural Networks is based on **differentiability**: gradients (vector fields indicating the steepest directions in the functional landscape) drive the optimization procedure towards low loss values. Although they have no guarantees of convergence (in contrast with the closed-form solutions or optimization methods on convex problems), the powerful method is still benefiting from many important features. However, the optimization algorithm should be smart enough not to get stuck in saddle points or particularly bad local minima. Optimizers-related details will be discussed in Sec. 3.2.3.

It is worth to notice that there typically is no need to reach the global minimum of a loss function for a DL problem: considering that the data we have are only (few) examples sampled from the real case (true) distribution, reaching exactly the deepest minimum would with good probability correspond to overfit, for many problems. In other words, we do not need to find the best solution for our problem representation, since we are aware that such a representation of the learning problem is often not the best one (i.e. it can always be ameliorated by gathering more data), and therefore we should be fine with a slightly more general solution that is expected to generalize well outside the training set.

In practice, Gradient Descent algorithm let us follow the gradient of the loss function towards the minimum direction, with some tricks that allow the exploration of the functional landscape instead of a pure exploitation approach, as sketched in Fig. 3.6.

3.2.2.2 NN Architectures: General Design and Components In this section, the reader will be introduced to the formal structure of DNNs, including its terminology and an overview of the main components.

First of all, the scheme of a standard (rather basic, a DL researcher would say) is presented in Fig.3.10. The most fundamental component of the network is the unit (or neuron): as anticipated in Sec. 3.2.1, each circle of the figure is a unit. As Fig.3.9 shows, a single unit receives an input and processes it in a similar way of linear regression. The neuron has two parameters associated: a **weight** (w) and a **bias** (b), through which it calculates the results of $x' = wx + b$. As the last single-unit step, an activation function is fed with x', producing $\sigma(x')$ as the final output. Notice that here x and x' are generic single-unit inputs, and they depend on where in the network the specific unit is: the input will be a proper input feature (i.e. raw data representation) x if the unit is at the very beginning of

the network, otherwise x will be the result of the previous single-unit calculation $\tilde{x_n} = \sigma(x'_{n_1})$. When a neuron is placed neither at the beginning (input) neither in the end (output) of the network, it is called hidden unit (since it is hidden in between the network).

Many types of activation function for hidden units exist, but the most common are:

1. **Sigmoid**: Prior to the introduction of ReLU functions, a very common activation function was the Logistic Sigmoid function, defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{11}$$

However, the characteristics of Sigmoids, as can be seen in Fig.3.11(a), is that they saturate across most of their domain: when z is considerably positive ($z >> 0$) then $\sigma(z) \to 1$, while on the other hand $\sigma(z) \to 0$ when $z << 0$. Due to this property, the gradient-based learning is slowed down because no clear indication for the learning direction is provided (the first derivative tends to be constant when σ saturates). Only sensible values near $z \approx 0$ are good gradient-related indicators.

An activation function similar to σ is the **Hyperbolic Tangent** (see Fig.3.11(b), closely related to σ by

$$tanh(z) = 2\sigma(2z) - 1. \tag{12}$$

Tanh has some slightly better performances then sigmoid, but still suffers from the same problems [65].

2. **ReLU**: **Rectified Linear Units** activation function [66] is a piece-wise linear function that overcomes the problems introduced by the sigmoidal function, especially regarding its saturation. It is defined as

$$g(z) = \max(0, z), \tag{13}$$

and its representation can be observed in Fig.3.11(c). Its introduction comes with several advantages:

   (a) a piece-wise linear function is easy and very efficient to be optimized,
   (b) its first derivative remains large when the unit is active ($z > 0$)
   (c) the second-order derivative g'' is piece-wise constant, making the gradient-learning more feasible by not introducing second order effects.

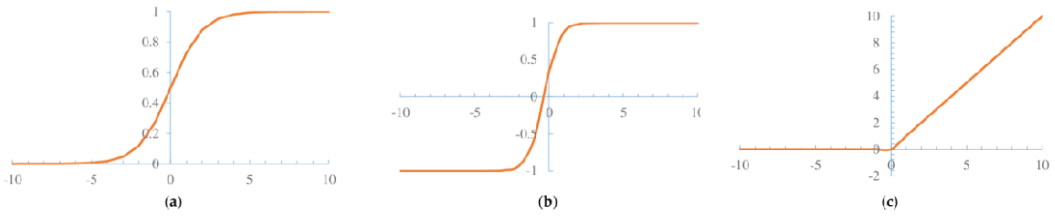

Figure 3.11: Most common Hidden Units Activations: Sigmoid (a), Tanh(b), and ReLU (c).

There are many generalizations of ReLU: Absolute Value Rectification, Leaky ReLU (LReLU) [67], Parametric ReLU (PReLU) [68], Maxout [69] are some examples. For an extensive, detailed and complete review of activation functions for DNNs the reader can refer to [70].

3. Others: other types of activation function, although rarely used, are:
   (a) Radial Basis Function (RBF): $h_i = e^{\frac{1}{\sigma_i^2}||\mathbf{W}_{:,i} - X_i||^2}$,
   (b) Softplus: the smooth version of ReLU defined as $g(z) = log(1 + e^z)$ [71],
   (c) HardTanh: similar to tanh and ReLU, but its bounded. $g(z) = \max(-1, \min(1, z))$ [72].

Units are grouped in organized structures called layers, with respect to a property: each unit of a layer is connected with (all, in the most basic case) units of the previous and subsequent layer (whereas units from the same layer are not interconnected). The first layer, called input layer, has the role of linearly mapping the input data X_0 to the first hidden layer. The subsequent layers are called hidden layers: they are responsible for the most important step, the information processing, which often exploits the non-linearity (given by the non-linear activation function) to learn suitable features. The output layer, at the very end of the network, provides the results in a form that is suitable for the given task. To this extent, the activation function for the last output units can be different from those of the hidden layers. In particular, some possibilities are:

1. Linear Unit: the simplest output unit. It outputs an affine transformation of the input $\mathbf{h}$: $\hat{y} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$. It can be used to generate a conditional Gaussian distribution
$$P(y) = N(y; \hat{y}, I). \tag{14}$$
Furthermore, it can be used in a wide variety of problems such as Regressions, since formally maximizing the log-likelihood with linear output units corresponds to minimizing the mean squared error (MSE).

2. Sigmoid: the same activation described previously. It provides a suitable output for a binary classification task: a binary (conditioned Bernoulli) distribution probability.

3. Softmax: it is used to output a probability distribution over a discrete variable with n possible modalities. In other words, the softmax, defined as

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{15}$$

outputs a Multinoulli distribution. The softmax is the preferred choice when dealing with a multi-class classification problem.

3.2.3 Optimization for Deep Neural Networks

What was described in so far represents the feed-forward phase of the network: the input flows through the input layer and the various hidden layers produce hidden representation (the aforementioned *learned features*) by processing information in a non-linear way, up to the output layer that provides the results in the desired shape.

But: *How is the network trained?*

This question is equivalent to: *for each hidden unit, how the parameters* (w, b) *are chosen?*

The short answer is: they are learnt through gradient descend on the loss function iteratively updating the parameter values according to the step-wise gradient calculated exploiting the chain-rule-based backpropagation technique.

The long answer is represented by this section, where the Gradient Descent (GD) (Sec. 3.2.3.1) and Back-Propagation algorithms (Sec. 3.2.3.2) will be introduced, together with an overview of the main optimizers available for training Neural Networks via GD in Sec. 3.2.3.3. Furthermore, some alternative examples of training algorithms for Neural Networks will be mentioned in Sec. 3.2.3.1, together with the most important challenges for Neural Networks from the optimization perspective in Sec. 3.2.3.

3.2.3.1 Gradient Descent and Its Variants As we have already understood, most of ML (and DL, of course) algorithms require some kind of optimization process, to minimize an objective function, also called cost function, loss function, or simply loss. The Gradient Descent (GD) method, introduced by Cauchy in 1847 [73], is based on the observing that the gradients (the derivatives, in scalar terms) and the first-order Taylor expansion specifies how a small change

in the input influences the output:

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x). \tag{16}$$

In other words, the gradients tell us how to move through x in order to minimize or maximize $y = f(x)$. In particular, they describe the direction in which f changes the fastest. Indeed, from the directional derivative of a generic differentiable $f : \mathbb{R}^n \to \mathbb{R}$, one have:

$$\lim_{\alpha \to 0} \frac{\partial}{\partial \alpha} f(\mathbf{X} + \alpha \mathbf{u}) = \mathbf{u}^T \nabla_x f(\mathbf{X}), \tag{17}$$

where $||\mathbf{u}|| = 1$. Therefore, to minimize f:

$$\min_{\mathbf{u}, \mathbf{u}^T\mathbf{u}=1} \mathbf{u}^T \nabla_x f(\mathbf{X}) = \min_{\mathbf{u}, \mathbf{u}^T\mathbf{u}=1} \|\mathbf{u}\|_2 \|\nabla_x f(\mathbf{X})\|_2 \cos(\theta) = \min \cos(\theta) \tag{18}$$

so that f decreases (increases) by moving in the direction of negative (positive) gradient. Starting from a (typically random) point, the GD method proposes a new point

$$X' = X - \epsilon \nabla_x f(x), \tag{19}$$

calculates the gradients in X' and recursively iterates. In Eq.19, ϵ, a positive real number known as **Learning Rate**, is a hyper-parameter controlling the size of the new point update. The GD procedure converges when the gradients tend to zero, i.e. X' would be updated by such a small quantity that the changes over f are no longer relevant.

Stochastic Gradient Descent GD can be implemented in several variants: one of the most famous is the Stochastic Gradient Descent (SGD) [74]. Starting from the observation that computing the exact expectation of the gradients for a DNN cost function, as

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{x,y \sim p_{data}} \nabla_\theta \log_{p_{model}}(X, Y; \theta) \tag{20}$$

can be very expensive, since it would require the evaluation over the entire training set, it is possible to obtain a fair (unbiased) estimation of the gradient by taking a sort of average over the gradients on multiple mini-batches of m examples sampled from the training set. This results in the SGD method, formally described as:

Algorithm 1 Stochastic Gradient Descent procedure at the k-th iteration.

Be ϵ the Learning Rate,
Be θ the initialized parameters,
while *! Stopping criterion* **do**
  Sample a mnibatch of m examples from the training set $\{\mathbf{X_1}, ..., \mathbf{X_m}\}$ and the associated labels $\mathbf{y^{(i)}}$,
  Compute the gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{X_i}, \theta), \mathbf{y_i})$,
  Apply the update: $\theta' \leftarrow \theta - \epsilon \hat{\mathbf{g}}$.
end

In SGD, the learning rate ϵ becomes crucial: it can no longer stay fixed, but needs to be scheduled to descrease during the training, to compensate the source of noise introduced by the random sampling of mini-batches, that would otherwise prevent the algorithm to converge.

Momentum Another improvement of GD-based algorithms is the **Momentum**, introduced by Polyak in 1964 [75]. Its scope is to accelerate the learning process, since the SGD convergence can sometimes be particularly slow. The Momentum procedure keeps track of the past gradient direction (through an exponentially decay moving average) and continues to move towards that direction. Formally:

Algorithm 2 Stochastic Gradient Descent procedure with Momentum at the k-th iteration.

Be ϵ the Learning Rate,
Be θ the initialized parameters,
Be α the momentum parameter,

while *! Stopping criterion* **do**
  Sample a mini-batch of m examples from the training set $\{\mathbf{X_1}, ..., \mathbf{X_m}\}$ and the associated labels $\mathbf{y^{(i)}}$,
  Compute the gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{X_i}, \theta), \mathbf{y_i})$,
  Compute the velocity update: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$,
  Apply the update: $\mathbf{theta'} \leftarrow \theta + \mathbf{v}$.
end

It is worth to notice how both the name "Momentum" and its content derives from the physics concept of momentum, which, informally, describes the inertia amount of a moving body. The momentum method can be further improved by

its Nesterov [76] variant, based on the Nesterov's accelerated gradient algorithm. In this case, the update rule is given by

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\nabla_\theta[\frac{1}{m}\sum_{i=1}^{m} L(f(\mathbf{X}_i; \theta + \alpha\mathbf{v}), \mathbf{y}_i)], \tag{21}$$

which further speeds up the convergence.

Alternatives to Gradient Descent Many Alternatives has been proposed exploiting different mechanism from GD for optimizing DNNs, some of them reaching competitive performances. We only cite a few examples, underlining that GD-based algorithms are still one of the best options available nowadays. Examples of alternatives are:

1. **Difference Target Propagation**: the main idea is to compute targets rather than gradients, at each layer. The procedure consists of formulating targets, assigning a proper target to each layer, applying Difference Target Propagation, and training an auto-encoder with difference target propagation. A detailed description is available in [77].

2. **Hilbert-Schmidt Independence Criterion (HSIC) Bottleneck**: in short, a sort of mutual information-based objective function between hidden representations and labels is found and maximized. Furthermore, by doing this one minimizes the mutual dependency between hidden representations and the inputs. A detailed description can be found in [78].

3. **Others**: genetic algorithms can be exploited to directly search for optimal parameter configurations [79]. Other alternatives are Simulated Annealing Procedures , Markov random fields methods. Another Curious alternative is represented by the **Extreme Learning Machine**, essentially neural networks where the weights connecting the inputs to the hidden nodes are assigned randomly and never get updated. The weights between the hidden nodes and the outputs are learned in a single step by solving linear equations [80].

3.2.3.2 Back-Propagation We already introduced the forward propagation phase for a Deep Neural Network: information flows through the input layer, is elaborated in a non-linear way by the hidden layers and produces, in the output layer, a result whose quality corresponds to a scalar cost $J(\theta)$, where J is the loss function of the NN. The *dual* backward step is represented by the **Back-Propagation** process [81]: the information now flows from the cost function

back through the network allowing to compute the gradients that will be essential for the iterative weigh update made during Gradient Descent Optimization. BackPropagation, also simply known as **BackProp**, exploits simple mathematical tricks, such as the recursive application of the chain rule, to perform efficient and inexpensive gradient calculation, in contrast to the computationally expensive numerical evaluation of analytical gradient expressions. While BackProp has become famously known due to its application to DNNs, the idea of obtaining derivates by propagating the information through a function composition (in our case, a multi-layer NN) is very general, and can be employed in other ML algorithms as well as wherever there is the need to compute a generic gradient $\nabla_X f(X, Y)$ for an arbitrary function f, $X \in \mathbb{R}^n$, $Y \in \mathbb{R}^m$. In the case of Neural Networks, we are interested to compute $\nabla_\theta J(\theta)$, where J is the loss function and θ the network parameters. As we mentioned, BackProp is derived from an elemental rule of calculus: let us then recall the concept of Chain Rule

Chain Rule When a function is a composition of other simpler functions, $f(y) = f(g(x))$, the chain rule allows computing its derivatives based on the derivative of the composed functions:

$$\frac{df}{dx} = \frac{df}{dy}\frac{dy}{dx}. \tag{22}$$

Beyond the scalar case, we equivalently have:

$$\nabla_X f = (\frac{\partial g}{\partial x})^T \nabla_y f, \tag{23}$$

where $\frac{\partial y}{\partial x}$ is the Jacobian Matrix of g, $X \in \mathbb{R}^n$, $Y \in \mathbb{R}^m$. Lastly, the procedures generalizes to a tensor X:

$$\nabla_x f = \sum_j \nabla_x g(X)_j \frac{\partial F(Y)}{\partial Y_j} \tag{24}$$

where $(\nabla_x f)_i = \frac{\partial f}{\partial X_i}$. The method, of course, generalizes also to the composition of more than two functions by iteratively applying itself through the function chain (from where the name).

BackProp for Fully Connected Neural Networks We formally present the BackProp algorithm when applied to a Neural Network. The feed-forward phase can be algorithmically described as:

Algorithm 3 Forward propagation for a standard DNN. The loss is $\mathcal{L}(\hat{y}, y)$, a regularizer $\Omega(\theta)$ is considered, and the input is a single example $\mathbf{x}$

l network depth
$W^{(i)}$, $i \in \{1, ..., l\}$ the weight matrices of the model
$b^{(i)}$, $i \in \{1, ..., l\}$ the biases of the model
$\mathbf{x}$ the input, $\mathbf{y}$ the target

$\mathbf{h}^{(0)} = \mathbf{x}$
for $k = 1, ..., l$ **do**
  $\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}$
  $\mathbf{h}^{(k)} = f(\mathbf{a}^{(k)})$
end
$\hat{\mathbf{y}} = \mathbf{h}^{(l)}$
$J = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) + \lambda\Omega(\theta)$

Starting from those concepts, the BackProp pseudocode can be represented as:

Algorithm 4 Backward propagation for a standard DNN. We assume the feed-forward phase described in Algorithm.3 is already done. For each layer k, the gradients on the activations $\mathbf{a}$ are computed. From those gradients, the gradient of the parameters can be obtained, and either used immediately via a stochastic gradient update or used with other gradient-based optimization methods.

$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
for $k = 1, ..., l$ **do**
  Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation:
  $\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot \mathbf{f}'(\mathbf{a}^{(k)})$
  Compute gradients on $\mathbf{W}$ and $\mathbf{b}$:
  $\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta)$
  $\nabla_{\mathbf{W}^{(k)}} J = \mathbf{g}\mathbf{h}^{(k-1)T} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)$
  Propagate the gradients w.r.t. the next lower level hidden layer's activations:
  $\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)T}\mathbf{g}$
end

As we can notice, BackProp is a simple algorithm that, if efficiently implemented, can provide fast computation for the derivatives. The gradients are then exploited by the GD procedure to train the network.

Challenges and Alternatives Several challenges occur when optimizing the BackProp method for Neural Networks represented as computational graphs. Symbol-to-Symbol differentiation [82] can help when dealing with memory-consumption problems that arise with large graphs (large/particularly deep NNs). The algorithm would also need improvements to handle multi-tensor cases, or multi data-type inputs (i.e. 32-bit /64-bit floating points). Undefined gradients can sometimes emerge, and a real implementation of BackProp should track and handle those events. Few alternatives to backpropagation exist. We mention them in the following, but we highlight how for Deep Neural Network applications the BackProp still remains the most common algorithm. Early examples of Backprop-free neural networks alternatives are the Elman Neural Network or Jordan Neural Networks, whereas some newer approach can be seen in the work of Gaier and Ha [83] with their "Weight Agnostic Neural Networks".

3.2.3.3 Optimizers Optimization algorithms for Deep Neural Networks are also known as optimizers. Gradient Descent, Stochastic Gradient Descent, and Stochastic Gradient Descent with Momentum can be taken as examples of optimizers. However, Modern Optimizers are characterized by having the adaptive feature for their hyper-parameter: in particular, they adapt the individual learning rates for model parameters during the training phase. The most famous and widely used SGD-based adaptive optimizers are:

1. **AdaGrad**: it adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the accumulation of all the historical squared values of the gradients [84].

2. **RMSProp**: modifies the AdaGrad implementation by changing the gradient accumulation to an exponentially weighted moving average. This results in a better convergence property when dealing with highly non-convex functions, since only the last tracks of gradients are considered relevant to RMSProp, conversely to the Adagrad implementation where the gradients relevance is equal through the past history. RMSProp can be further improved with the addition of a Nesterov Momentum term.

3. **Adam**: the name is suggested by its "adaptive moments" component. With respect to RMSProp, Adam adds the momentum term which is incorporated directly as an estimate of the first-order moment with exponential weighting of the gradient. Furthermore, Adam implements bias corrections to the estimate of first-order and second-order moments to take in consideration their initialization. As a result, Adam is generally considered quite robust

to its hyperparameter choice, and it became one of the most used optimizers for NNs. A detailed description of its working principles can be observed in Algorithm.5, whereas the original paper is [85].

Algorithm 5 Implementation of the Adam Algorithm.

Parameters:
ϵ, the learning rate,
ρ_1, ρ_2, the exponential decay rates for the moment accumulation,
δ, a numerical constant to avoid numerical overflow and instability,
θ, the initial network parameters,
$\mathbf{s}, \mathbf{r}$ moment variables initialized to zero,
t, time step initialized to zero,
while *! stopping criterion* **do**
  Sample a mini-batch from the training set,
  compute the gradients,
  update the time step: $t \leftarrow t + 1$,
  update biased first moment estimate: $\mathbf{s} \leftarrow \rho_\mathbf{1}\mathbf{s} + (\mathbf{1} - \rho_\mathbf{1})\mathbf{g}$,
  update biased second moment estimate $\mathbf{r} \leftarrow \rho_\mathbf{2}\mathbf{r} + (\mathbf{1} - \rho_\mathbf{2})\mathbf{g} \odot \mathbf{g}$,
  calculate bias corrections: $\hat{\mathbf{s}} = \frac{\mathbf{s}}{\mathbf{1}-\rho_\mathbf{1}^\mathbf{t}}$, $\hat{\mathbf{r}} = \frac{\mathbf{r}}{\mathbf{1}-\rho_\mathbf{2}^\mathbf{t}}$,
  update parameters: $\theta = \theta - \epsilon\frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\delta}$
end

3.2.4 Regularizing Deep Neural Networks

We've already introduced the role of regularization for general ML purposes in Sec. 3.1. In this paragraph, the focus will be on regularization techniques applied to DL models. We will understand what are the signs of overfitting, and, therefore, of the need for regularizing a Neural Network training. After that, an overview of regularization techniques for Deep Neural Networks will be presented. Lastly, we will provide the readers with some hints on self-regularization theories that could potentially give (at leaf a part of) the answer to the question *"Why Deep Neural Networks works so well?"*.

3.2.4.1 Training Behaviour in the Needs of Regularization Question: *While training a DNN, when there is a need for regularizers?* Short Answer: *When Deep Neural Network models show to overfit training data at the cost of poor generalization performances.* The question then becomes: *What are the most common signs of overfitting?*

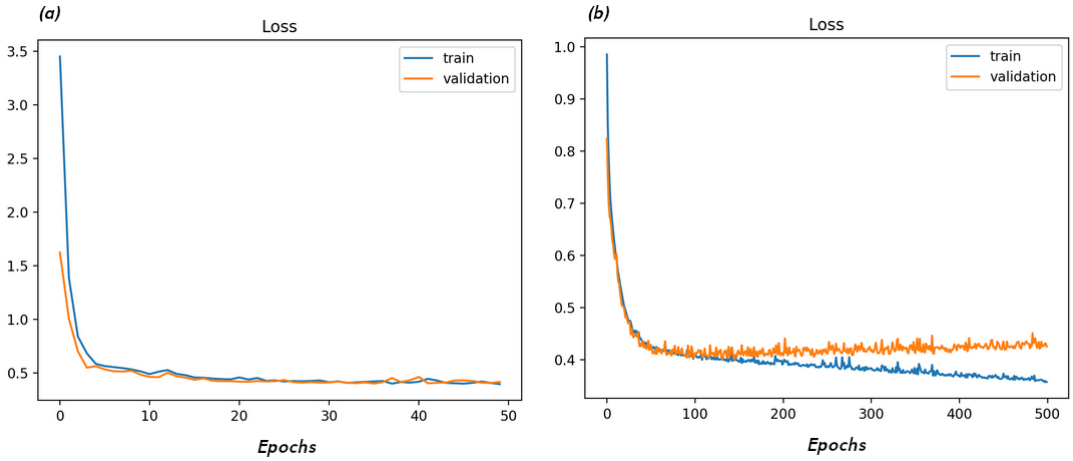

Figure 3.12: Example of a training and validation loss curves of a DNN. In panel (a), a good behaviour of the model with respect to generalization capability. In panel (b), a typical sign of overfitting occurs after epoch 100.

One of the clearest indications of overfitting is obtained by monitoring the model performances on a validation set, a subset of the training set, during the training. Usually, the most commonly monitored quantities are the performance measure(s) (i.e. accuracy for a balanced classification task) and the loss values. Practitioners love to keep track of those quantities epoch-wise, with their values optionally averaged over the mini-batches. The result is a graph like that in Fig. 3.12. The key point of such a graph is that the training curves (both accuracy and loss ones) are typically not sufficient to assess model performances and specifically generalization capabilities. One always need to observe them in combination with the validation curves: when a training loss decrease constantly and the validation one has the same overall behaviour (such as in Fig. 3.12 panel a), the model is probably performing well enough. When, instead, the validation curve tends to change its behaviour, i.e. the validation loss suddenly starts to increase (see Fig. 3.12 panel b), it could be a clear sign that the model is focusing too much on the training data, causing the system to overfit.

Other more complex procedures to detect overfitting and learning problems rely on feature analysis and interpretation, to understand what the network is focusing on for the labelling decisions.

In general, Deep Neural Networks are complex and powerful tools, and they are very prone to overfit when no source of regularization is exploited. A good habit when training Neural Networks is then to always use some sort of regularizers, sometimes even more than one, combining together different regularizing effects.

Expert Corner An extreme case of overfitting and issues in learning can happen when training a particularly large or deep NN. It has been shown that such a network is capable of fully memorizing training data, being able to perform well in the training phase even on random labels. In these cases, the capacity of the model is so high that the Neural Network is believed to encode and memorize the entire training dataset in its parameter space. The result is that no proper learning takes place.

3.2.4.2 Explicit Regularizers In Sec. 3.1.2.5 we introduced the differences between the implicit and explicit families of regularizers. Explicit regularizers impose constraints over the learning problems by directly influencing the loss function $\mathcal{L}$, typically with the addition of an extra term $\mathcal{L}_{reg}$, to form the final loss function

$$\mathcal{L} = \mathcal{L} + \mathcal{L}_{reg}, \tag{25}$$

whose minimization integrates the effect of the regularizer in the optimization problem. Typically, the regularizing term of the loss has the form:

$$\mathcal{L}_{reg} = \lambda\Omega(\theta), \tag{26}$$

where λ is the hyperparameter controlling the strength of the regularization effect, and $\Omega(\theta)$ is a function of the network parameters θ. It is worth to notice that Ω is typically a function of the weights only, in the sense that it affects only the weights of a NN, leaving the biases unconstrained. This is because fitting the biases typically requires less data (and efforts) than fitting the weights. Therefore, leaving unregularized biases should not introduce too much variance in the learning problem.

A well-known and most used class of explicit regularizers $\Omega(\theta)$ imposes constrains on the norm of the network weights. Famous norm penalties are listed below:

1. L^2 **norm penalty**: this regularization strategy is driven by the regularization term $\Omega(\theta) = \frac{1}{2}\|\mathbf{w}_2^2\|$. It is also known under the name of *Weight Decay*, *Ridge Regression*, *Tikhonov regularization*. To understand how this term influence a loss function, let's consider the linear regression problem, where $\mathcal{L} = (\mathbf{Xw} - \mathbf{y})^{\mathbf{T}}(\mathbf{Xw} - \mathbf{y})$, and therefore

$$\mathcal{L}_{tot} = \mathcal{L} + \mathcal{L}_{reg} = (\mathbf{Xw} - \mathbf{y})^{\mathbf{T}}(\mathbf{Xw} - \mathbf{y}) + \frac{1}{2}\,\alpha\mathbf{w}^{\mathbf{T}}\mathbf{w} \tag{27}$$

The normal solution (without regularization) would be $\mathbf{w} = (\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-\mathbf{1}}\mathbf{X}^{\mathbf{T}}\mathbf{y}$. The regularized solution is

$$\mathbf{W} = (\mathbf{X}^{\mathbf{T}}\mathbf{X} + \alpha\mathbf{I})^{-\mathbf{1}}\mathbf{X}^{\mathbf{T}}\mathbf{y}. \tag{28}$$

One can notice that the regularization alters the "correlation" matrix $\mathbf{X^T X}$ by making its diagonal term larger, therefore causing the learning algorithm to see a higher variance in the input X, making it shrink the weights on those features whose covariance with the output is little compared to this increased variance. This observation could be considered as a general effect of weight decay.

For neural network in particular, one can observe how the regularization term influences a single step of the weight updates. When we calculate the gradients of Eq. 25, it becomes:

$$\nabla_{\mathbf{w}} \mathcal{L}_{\mathbf{tot}}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \alpha \mathbf{w} + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}). \tag{29}$$

Then, the single gradient step is:

$$\mathbf{w} \leftarrow (1 - \epsilon\alpha)\mathbf{w} - \epsilon \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}). \tag{30}$$

Therefore, the addition of the weight decay term modifies the update rule by shrinking the weight vector at each step.

2. L^1 **norm penalty**: similarly to weight decay, this regularization is driven by the term $\Omega(\theta) = \frac{1}{2}\|\mathbf{w}_1\| = \sum_i |w_i|$. In comparison to the weight decay, that favours small norms of the weights, the L^1, also called *Lasso Regression*, can be shown to favour small absolute values (with a similar approach showed for the L^2). Implicitly, L^1 also favour sparsity in the weight matrix, the reason why Ridge regression is often used as a feature selection mechanism.

3. **Expert Corner**: **Nuclear Norm Penalty and Weights Products**. L^1 and L^2 regularizations can be seen as special cases of a general class of regularizers driven by the nuclear norm of the weights:

$$\Omega_{\phi,\theta} = \min_{\{W^k\},r} \sum_{i=1}^{r} \theta(W_i^1, ..., W_i^k) \;\; s.t. \;\; \phi(W_i^1, ..., W_i^K) = Z, \tag{31}$$

where θ is positively homogeneous of the same degree of ϕ, the function representing a particular network structure. The nuclear norm regularizers can be seen as much more modern regularizers with respect to the other well-known norm penalties, because they have some interesting properties. In particular, It can be shown that when the problem to be optimized (the loss function) is homogeneous of degree n, and therefore convex, adding a special family of nuclear norm penalties in their variational form [86, 87], the weight product norm (product of the decomposition elements of the weight

matrix) [88, 89], of the same degree n does not introduce non-convexity, that is, the problem remains convex [90, 91].

For non convex function, the behaviour is not always clear, but in general those kinds of norms seem to perform as well as the commonly used, when not outperforming their regularization capability.

3.2.4.3 Implicit Regularizers In contrast to explicit regularization techniques, implicit ones does not directly affect the loss function, nor introduce extra terms in the optimization objective. Instead, they indirectly influence the training phase, in different manners depending on the specific regularizers. From an optimization perspective, those procedures are not properly regularizers, but they can be seen more as training tricks that acts as regularizers because they induce some effects that improve generalization. Given this fact, they can be fully considered regularizers when using the definition given in Sec. 3.1.2.5. Furthermore, sometimes explicit regularizers are shown to be equivalent, in term of their induced effects, with respect to the introduction of particular terms (explicit regularizers) in the loss function. In those cases, we can see how the distinction between implicit and explicit regularization techniques is not always net. Given the vast possibilities of influencing the training phase, a large number of implicit regularizers exists. In the following, we present some of the most famous ones:

1. **Dropout** The dropout technique [50] is a very powerful and computationally inexpensive regularizer. During the training of a Neural Network with dropout, for each mini-batch, a binary mask is randomly sampled and applied to all the input and hidden units of the network. In other words, with dropout each unit has a certain probability to be active (when the sampled mask has the non-zero value) or to be inactive (the output is multiplied by the zero value of the mask). These probabilities are independent for each unit, since the masks are sample independently. A graphical representation of Dropout is shown in Fig. 3.13. It is worth to notice that this procedure leads to the creation of several sub-networks during the training, and the overall results can almost be compared with a bagging procedure. Nowadays,Dropout is commonly used in a wide variety of neural architectures. Furthermore, this technique is also an example of implicit regularizers that can allow an explicit formulation for certain Neural Networks [51, 52, 53].

2. **Early Stopping**: this is a simple regularization trick. It consists of monitoring the validation and training curves, and to stop the training when some sign of overfitting occurs. For example, referring to Fig. 3.12 panel b, the early stopping would be applied by restoring the weights of the final

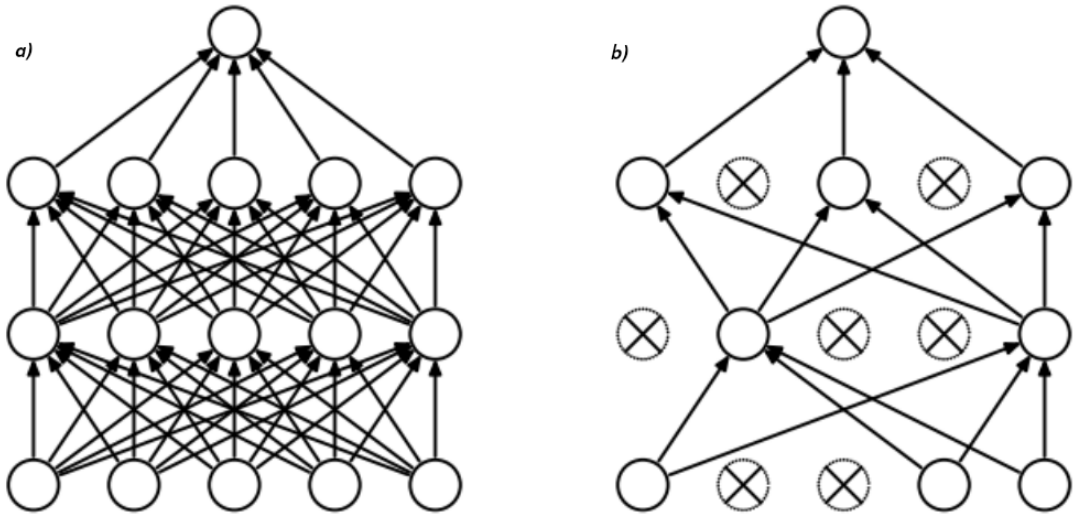

Figure 3.13: Dropout. In panel a), a standard NN. In panel b), the same Neural Network after the application of the Dropout masks.

model to those learnt about epoch 100, where the overfitting, indicated by the rising of the validation curve, shows up. **Expert Corner:** It is worth to notice how the effect of early stopping can be so easily explained when one considers the frequency analysis approach presented in [37, 43], where it is clear that training a NN for a long time can allow fitting higher and higher frequencies, therefore to produce overfitting on the problem. Conversely, applying Early Stopping means to impose a sort of constrain on the higher-bound for frequency learning, resulting in the learning of only low (suitable) frequency that allows a better generalization.

3. **Batch Normalization**: it is a normalization technique, typically applied for hidden features, that consists in the normalization of the feature set according to a standard distribution. In addition to its regularization effect, somewhat comparable with those of norm penalties, Batch Normalization [92] has been proved to speed up the learning process and to make it more robust to different initializations and learning rates.

4. **Noise Injection**: a possible regularization strategy is to inject random noise in the input features. This increase the input variances, and the regularization effect is similar to the norm penalty one [93]. Furthermore, noise can be added to the hidden features (the research in this direction leads to the introduction of the previously presented Dropout Technique), and also to the weights themselves [94]. This last option is characterized by introducing the uncertainty factor in the weights representation: it would be equivalent to a stochastic approach for Bayesian inference over the weights.

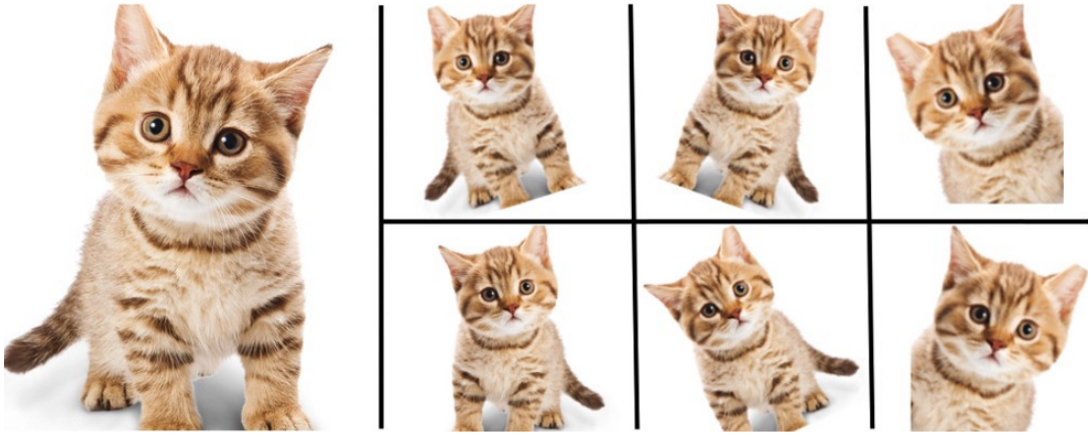

Figure 3.14: Example of Data Augmentation: the real image (left) is used to produce new data which represents the same subject in slightly different "situations" by applying random rotations, cropping, symmetric mirroring etc. Figure taken from [95].

5. **Data Augmentation**: it is a well-known, almost obvious fact that to increase generalization performances of ML and DL models gathering more data is always an option. However, sometimes it is just enough to further exploit the already available data. Data Augmentation techniques are developed to maximally exploit the information content of a dataset. A famous example where Data Augmentation techniques significantly increase performances is the computer vision domain. For images, data augmentation can be applied by introducing copies of the available data modified by exploiting rotations of random angles, cropping, zooming, resolution changes, and through the application of several filters, as shown in Fig.3.14. More formally, a data augmentation procedure exploits as much as possible the information content of a dataset my making explicit the invariances of the data itself. For example, a cat is still a cat, whether it is horizontally oriented or rotated of some random angles. Depending on the context, many other types of data augmentation can be applied. Data Augmentation applied to spectral data will be further discussed in Sec. 4.1.3.

3.2.4.4 Self Regularization

Expert Corner : in recent years, Deep Neural Networks have shown incredible performances over many benchmarks in the domain of Computer Vision, Signal Processing and several other fields. Many researchers have then asked a question to the DL community, that sounds like: *We know and understand* ***How***

DL works, but ***Why*** *it actually works so well?*. A piece of the (still unclear and not closed) answer can be found in terms of hidden, almost unknown inner mechanism of Deep Neural Network trained with SGD, that is assimilable to a sort of self-regularization (i.e., a regularization effect that is intrinsic in the working behaviour of a model and not externally imposed). Some evidence of such a mechanism has been found in the following researches: [37, 38, 39, 43].

3.2.5 Architectures

The standard basic Neural Network model we have described so far is known as **Fully-Connected Neural Network**. It consists of many layers, where each unit in each layer s connected to all the units of the sub-sequential layer. In this section, I will introduce other possible Neural Network model types and architectures, more elaborated, complicated and/or sophisticated than the basic Fully-Connected networks. First of all, the CNN architecture will be described in details, presenting its various components, such as the convolution operations (which describe *what* a CNN is) and the pooling layer, its application and the various interpretation of its learning process, namely describing the biological view related to hierarchical feature learning.

3.2.5.1 Convolutional Neural Networks The most straightforward method to introduce CNNs is to present what the convolution operation is. The convolution operation is usually exploited in various fields, such as engineering, physics, pure and applied mathematics, etc. The variant defined in the NN domain is similar, although slightly diverse in its formal definition.

Convolution Operation In a general form, the following is the definition of convolution:

Definition 4 (Convolution Operation)**.** *A convolution is an operation defined over two function f, g (typically, of real-valued argument), that produces another function according to:*

$$\gamma(t) = (f * g)(t) = \int f(x)g(t-x)dx, \tag{32}$$

where the operation convolution is indicated by the symbol $*$.

Therefore, the convolution combines two functions to produce another function, with some interesting properties: in particular, the generated function can be seen as describing how the shape of the first function is modified by the second one. In the NN domain, the first term of the convolution (in the case of Def. 4,

f) is referred to as *Input*, whereas the second is known as the *Kernel*. The output is called *Feature Map*. Formally, for NNs we define:

Definition 5 (Convolution for NNs)**.** *Be the input* $\mathbf{X}$ *a multidimensional array of data, and the kernel* $\mathbf{K}$ *a multidimensional array of parameters that are learnt by the network. The convolution operation outputs:*

$$\mathbf{f} = (\mathbf{X} * \mathbf{K})(\mathbf{t}) = \sum_{\mathrm{x}=-\infty}^{+\infty} \mathbf{X}(\mathrm{x})\mathbf{K}(\mathrm{t}-\mathrm{x})\mathrm{dx}, \tag{33}$$

where $\mathbf{f}$ *is the produced Feature Map.*

We note that the Def. 5 is a sort of discrete version of Def. 4, since data, when represented on a computer, are typically discrete: for instance, think at the pixel of an image. For 1D data, such as audio signals or time series, Def. 5 is enough clear. For 2D data, such as images, the definition generalizes through the 2D formula:

$$\mathbf{f}(i,j) = (\mathbf{X} * \mathbf{K})(\mathrm{i},\mathrm{j}) = \sum_{\mathrm{m}} \sum_{\mathrm{n}} \mathbf{X}(\mathrm{m},\mathrm{n})\mathbf{K}(\mathrm{i}-\mathrm{m},\mathrm{j}-\mathrm{n}). \tag{34}$$

Practically, CNN uses the convolution operation to produce (learn) suitable feature maps through the learning of the kernel functions (i.e. network parameters). After a few convolutional layers, where convolutional operations are implemented, some Fully-Connected layers use those feature maps to perform a given task (i.e. to classify the images). Fig.3.15 gives a high-level idea of the convolution operation applied to the image domain.

Pooling Operation Common CNN models are composed, such as the one represented in Fig. 3.15 by a series of blocks given by a CNN layer and a Pooling Layer (plus a non-linear activation step in the middle). The CNN layer performs the convolutional mapping and then apply a non-linear activation function, in the same way as it was for the Fully-Connected layers. A Pooling layer implements the pooling operation: it modifies the output of the previous layer by condensing certain output values with a summary statistics of the nearby output region. Examples of pooling operator are the Max-Pooling [97], where a certain output region is replaced by the maximum value found in that area, or the Average-Pooling, where the replacement happens based on the average of the values in the considered region. An example of Pooling operations is represented in Fig. 3.16.

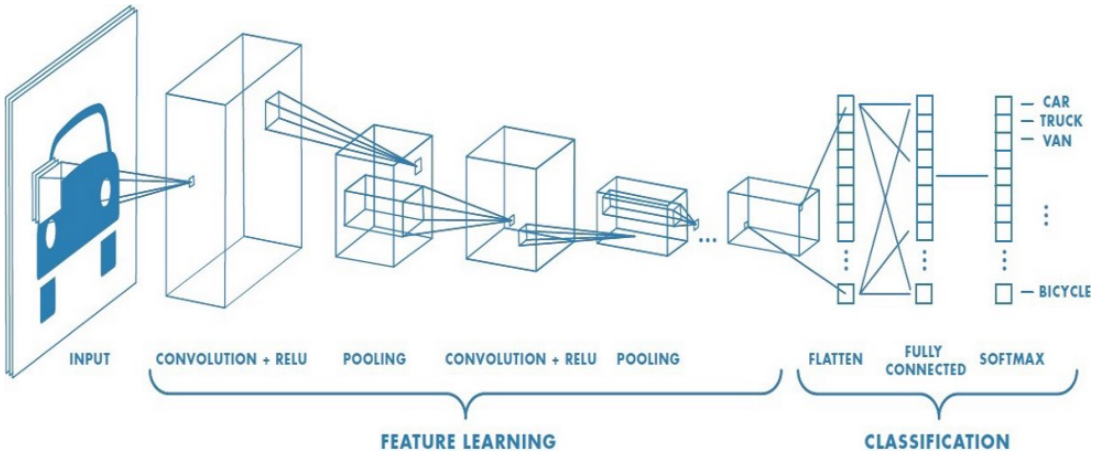

Figure 3.15: An example of CNN: the input image is processed by the Convolutional part, which performs feature extraction, and then further elaborated by the Fully connected components to accomplish the classification task. Figure taken from [96].

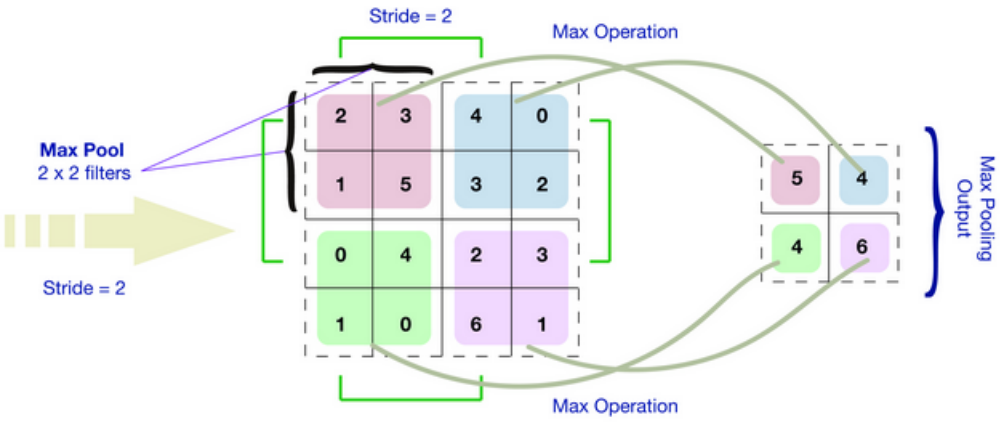

Figure 3.16: The Max-Pooling operation, condensing values from 2x2 squared regions.

Advantages of CNNs By definition, the units (parameter sets) in a CNN layer are no longer fully-connected, but they have **sparse connectivity**, since kernels are typically smaller than the size of the input. Fig. 3.17 describes sparsity in contrast to the fully-connected representation. This characteristic reduces the total number of parameters to be learnt, saving memory and making the learning process more efficient and faster. Furthermore, parameters can be **shared**: more than one function in a model can use the same set of parameters. Indeed, in CNNs each component of a kernel map is used at (almost) every position of the input: rather than having a separate and independent set of parameters for each location, only one set has to be learnt. The parameter sharing has multiple effects: it further reduces memory usage, increases the efficiency of the algorithm, and

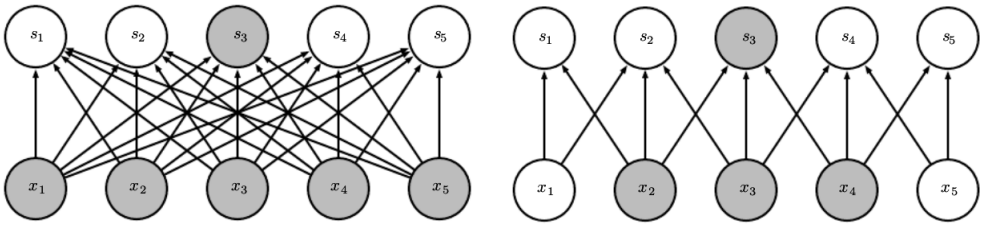

Figure 3.17: An illustrative representation of CNN model sparsity, where a unit x is connected only to a few units of the next layer (right side), in contrast to Fully-Connected models (left). Figure taken from [48].

.

makes the convolutional layers equivariant to translations[4]. In addition, another advantage of convolutional blocks is given by the pooling operation: it increases the efficiency by reducing the number of parameters (condensing information) and it also induces a weak translational invariance[5].

Historical and Biological Perspective From a historical perspective, Convolutional networks have a fundamental role in the history of deep learning. They represent one of the first successes derived from the study of human brains applied to machine learning applications. Convolutional networks are considered one of the first commercial successes of Deep Learning, since they have been used to solve important commercial applications (especially in the domain of Computer vision) and they still remain at the forefront of almost any DL-based commercial applications of deep learning today. Examples of early commercial applications are the neural networks obtained at ATT for reading checks [98], or several OCR and handwriting recognition systems(i.e. by Microsoft [99]. A complete historical review of CNNs can be found in [100].

From a biological perspective, it is worth to highlight how the convolutional structure is obtained through the study of the brain. Indeed, Convolutional networks are probably the greatest success of biologically inspired artificial intelligence. The key design principles of neural networks were borrowed from neuroscience. Neurophysiologists David Hubel and Torsten Wiesel collaborated for several years to study how the mammalian vision system works [101, 102]. Their research won a Nobel prize. Their findings were based on recording the activity of individual neurons in cats. They observed how neurons in the cat's brain responded to images projected in precise locations on a screen in front of

[4]That is, when the input changes, the output changes accordingly, in the same way.

[5]If the input is translated by a small enough value, the value of the output after pooling is likely to remain unchanged.

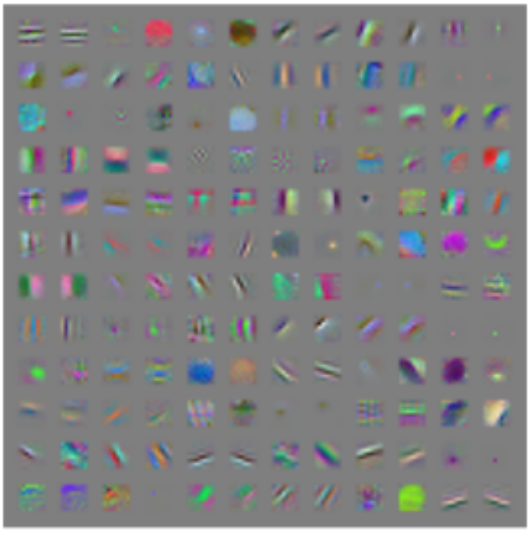

Figure 3.18: Convolution kernels learned by the first layer of a supervised convolutional network. Note that these features remind of specific colors of edges typical of natural images.

the cat.

In particular, the most interesting discovery was that neurons in the early visual system responded most strongly to very specific patterns of light, such as basic geometric shapes (i.e. lines). This kind of pattern matching is retrieved in CNN, for example see Fig.3.18.

Furthermore, it has been shown that while learning a complex problem, such as the classification of the Imagenet benchmark in Computer Vision, the CNN is capable of learning hierarchical features: the lower layers learn basic kernel features, matching with basic geometrical shapes. Deeper the layer, higher the abstraction level is, and kernels are learnt according to the combination of simpler features provided by the previous layers. An example of hierarchical feature learning is reported in Fig. 3.19

3.2.5.2 Others Shortly, other examples of famous Deep Learning architectures are:

1. **Recurrent Neural Networks** (RNNs): in contrast to Convolutional networks, which provide a way to specialize neural networks to work with grid-structured data such as images, recurrent neural networks are thought to process one-dimensional, sequential data. A recurrent neural network has characteristics connections between nodes that forms directed graphs along a temporal sequence. This property allows dealing with temporal

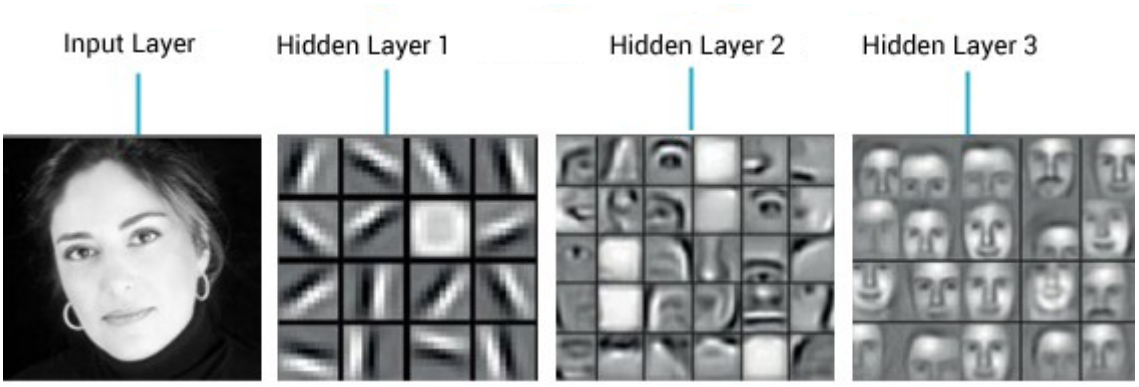

Figure 3.19: An example of hierarchical feature learning: from the input image, the first layer produces kernels related to edges and basic shapes of the figure, then the second layer produces combinations of edges, and the third provides some sort of object models.

dynamic data. RNNs have one or more internal states (memories) to process temporal input, making great progress in tasks such as unsegmented, connected handwriting recognition [103] or speech recognition [104, 105]. These internal memories act as storage capacity under direct control by the neural network. The storage unit can also be substituted by other networks or graphs, to deal with time delays or feedback loops. Those states are typically called **gated state** or **gates**, and many successful recurrent models, such as LSTMs [106] are based on them.

2. **Generative Models: AE, VAE, GAN.** Other interesting variants of deep learning models are the Generative Models. They are a powerful way to model any sort of data distribution using an unsupervised learning approach. Generative models have achieved tremendous success in recent years. Two of the most commonly used and efficient approaches are Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN). In general, Autoencoders (AE) are particular model structures whose task is to reproduce the input they are feed with, after several manipulations. In particular, they typically compress it in the hidden layers, the reason why they can be seen as a compression algorithm when taking as outputs the hidden representations. Variational Autoencoders still reproduce the input, but applying a reparametrization trick in the central layer, that re-sample the features from the encoded latent distribution to generate an output that is similar, but not exactly identical, to the input. From this starting point, Generative Adversarial Networks (GANs) has been created. They generate images by jointly training two subsystems called Generator and Discriminator: the generator generates an output based on some examples, and the

Discriminator establishes whether the output is coherent with the (latent) training data distribution.

3.2.6 Hyperparameter optimization

Most deep learning algorithms have a complex structure, characterized by several hyperparameters. We already introduced the concept of hyperparameters in Sec. 3.1.3, and underlined the difference from the regular (learnt) parameters. In general, dealing with hyperparameters is hard, since they control the behaviour of the overall system under various perspectives: some hyperparameters control the running time and memory consumption of the model (i.e. the number of epochs or the batch size of a DNN), some others deal with the quality of the results (i.e. the λ regularization strength parameter). Therefore, when learning and deploying a NN model, selecting a suitable set of hyperparameters is a must, and it is often a hard choice.

How to properly select hyperparameters? There are typically two approaches: the manual tuning and the automatic strategy. Manual tuning hyperparameters requires expertise and domain knowledge, in addition to many trial and error attempts and a large amount of (human) time. **Automatic Hyperparameter Optimization** (HPO) reduces those requirements, saving human time, at the obvious expense of high computational cost.

3.2.6.1 Automatic Hyperparameter Optimization Although NNs can still perform well with only a small set of tuned hyperparameters, they can often benefit from the tuning of fifty or more parameters. In those situations, manual hyperparameter selection tends to become unfeasible. An option is then to wrap the hyperparameter selection as another learning problem, where hyperparameters can be seen as the parameters to be optimized. This is the concept of Automatic Hyperparameter Optimization. Notice, however, that those wrappers, the HPO algorithms, have their own hyperparameters to tune (such as the searching space for the hyperparameter of the model). Fortunately, those secondary hyperparameters are easier to tune and can be standardized without too much effort. In the following, I'll present several different HPO algorithms, sorted by computational complexity: from lightweight methods such as grid or random searches, up to some more sophisticated model-based techniques. In the experimental section (Sec. 4.1.6), we will further describe these techniques from a more practical perspective.

Grid and Random Search When dealing with few hyperparameters, a good option is to perform grid search. In the grid search, the user manually provide a small set of values to be explored. The grid search algorithm fits a model for every combination of the specified hyperparameter values. The combination that provides the best validation set error corresponds to the best hyperparameters. The problems of grid search are:

- it is not always clear what is a suitable search space to be specified by the user. Knowing it requires expertise.
- the computational cost grows exponentially with the number of hyperparameters.

To partially solve those problems, the Random Search approach has been introduced by [107].

Random search is simple yet convenient to use. It has been proved to converge much faster to good values of the hyperparameters when compared to grid search. Firstly, the user defines a marginal distribution for each hyperparameter, (a Bernoulli or Multinoulli for binary or discrete hyperparameters, a uniform distribution for continuous real-valued hyperparameters etc.). Then the algorithm proceeds as the gridsearch, it fits several models by sampling random values for the hyperparameters from their specified distributions, and the best validation error is associated with the best set of hyperparameters. When compared to grid search, the advantages of random search are [107]:

1. no need to discretize or bin the values of the hyperparameters,
2. more exploration,
3. exponentially more efficient,
4. converges faster and better.

An illustrative example of the difference between grid search and random search is presented in Fig. 3.20.

Model-Based HPO As we mentioned before, the search for suitable hyperparameters can be wrapped into an optimization problem. In those cases, the learnt values are the hyperparameters, while the loss function to be optimized is the validation set error. Sometimes, it can be feasible to compute the gradients of some differentiable error measure on the validation set with respect to the hyperparameters: then, the algorithm can simply follow this gradient to learn [108]. However, in most cases, there is no way to calculate gradients, typically due to hyperparameters interactions with the validation error being intrinsically non-differentiable (for example, when the hyperparameter values are discrete). When

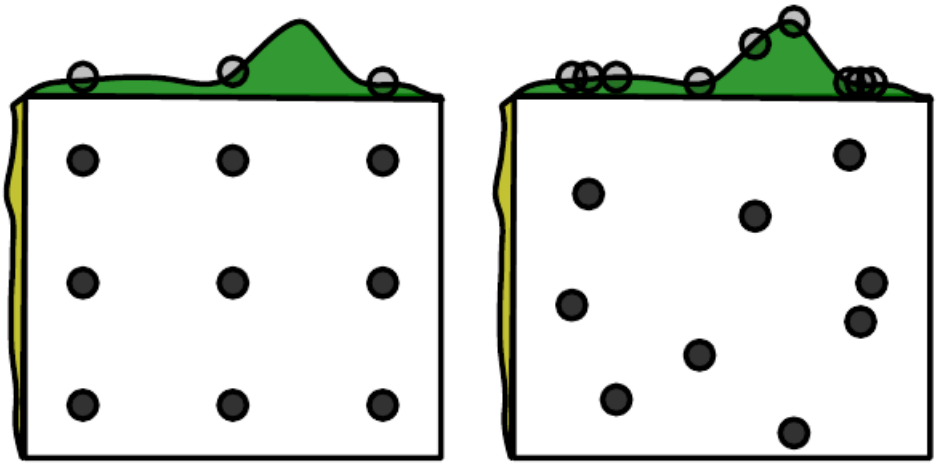

Figure 3.20: A Comparison of grid search and random search methods. In a typical case, only some hyperparameters have a significant influence on the result. In this example, only the hyperparameter on the horizontal axis has a significant role. Therefore, grid search wastes an amount of computation that is exponential in the number of non-influential hyperparameters, while random search tests a unique value of every influential hyperparameter on nearly every trial. Figure taken from [107].

no gradients information is available, one can build a model of the validation error and apply other optimization methods within this model.

A common drawback for model-based HPO algorithms is that they require higher computational resources, and the need for a training experiment to properly end before they are able to extract information from it. It's worth to notice that this last drawback is particularly less efficient compared to manual search, where one can usually realize early on if some set of hyperparameters is completely pathological.

Many model-based algorithms use a Bayesian approach to estimate the expected value of the validation set error for each hyperparameter and the related uncertainty. Some of those approaches include Spearmint [109],TPE [110] and SMAC [111].

What's the idea behind Sequential Model-Based optimization (SMBO) algorithms? In practice, those SMBO algorithms minimise the validation loss by sequentially selecting different hyperparameter sets through bayesian reasoning (dependently on the previous runs). In other words, those optimization algorithms sort of look back at the result of last runs to focus future searches on regions of the search space which look more promising.

Formally, the optimization problem in general can be written as the problem of finding:

$$\mathbf{x}' = \underset{\mathbf{x}\in\mathbf{X}}{\operatorname{argmin}}\, \mathbf{f}(\mathbf{x}), \tag{35}$$

where $\mathbf{x}$ is a generic set of hyperparameters in the hyperparameter space (search space) $\mathbf{X}$ and $\mathbf{f}$ is the loss function over the validation set. The reason to utilize SMBO is that often a proper fitness function $\mathbf{f}$ is too expensive to be precisely evaluated. Therefore, SMBO models an approximate function (called surrogate model), that is cheaper to compute. In practice, SMBO numerically optimizes this surrogate model, and the optimal point obtained is then used to evaluate the real loss function.

The optimization procedure in its bayesian form is based on Bayes' Theorem: in short, given a model Y and an input X (here, a set of hyperparameters) the posterior probability of Y given X ($P(Y|X)$) can be modelled as

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}, \tag{36}$$

where $P(X|Y)$ is the likelihood of X given Y, $P(Y)$ is the prior probability of Y and $P(X)$ is the marginal probability of X.

How the search space is explored? The tradeoff between exploration and exploitation is a well-known characteristic of most of the optimization algorithm in general. For SMBO, new points are proposed by means of an **Acquisition Function**. It is used to define a balance between exploring new regions and exploiting those who are already known to be promising. Many acquisition functions can be used for SMBO algorithms: the most common are probably the Expected Improvement (EI), i.e. for a Gaussian Process given by

$$\begin{aligned} &\mathbb{E}\,\mathbb{I}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^*))\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases}, \\ &Z = \tfrac{\mu(\mathbf{x}) - f(\mathbf{x}^*)}{\sigma(\mathbf{x})}, \\ &x^* = \operatorname{argmin}_{x \in X} f(x), \end{aligned} \tag{37}$$

and the Lower Confidence Bound (LCB), simply given by

$$\text{LCB} = \mu(x) - \xi\sigma(x), \tag{38}$$

where Φ the cumulative distribution function and ϕ the probability density function for the normal distribution given by $N(\mu, \sigma)$.

SMBO algorithms mainly differ from each other due to their optimization approach and to how the surrogate model is implemented. In the following, some different Model-based algorithmic flavours will be described: Tree-structured Parzen Estimators and Gaussian processes.

Tree-structured Parzen Estimator For the optimization settings, the Eq. 36 can be rephrased as

$$P(f|D_t) \propto P(D_t|f)P(f), \tag{39}$$

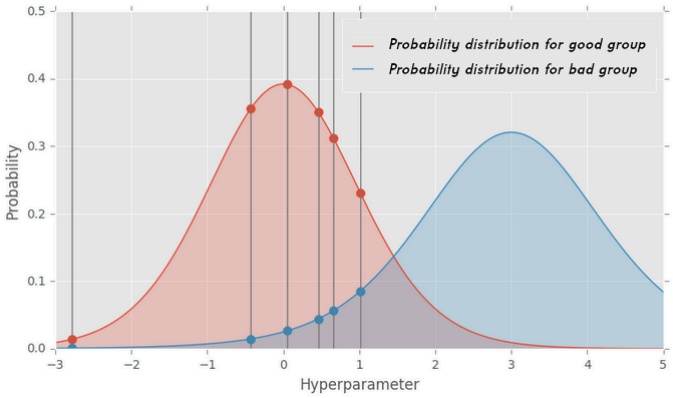

Figure 3.21: An illustrative examples of the probability distributions $h(x)$, associated with high-quality sampling, and $l(x)$, associated with low-quality solutions.

where f is the objective function, and $D_t = \{(\mathbf{x_i}, \mathbf{f}(\mathbf{x_i})) : \mathrm{i} = 1, ..., \mathrm{t}\}$ denotes the set of accumulated observations. Starting from this observation, the Tree-structured Parzen Estimator (also called TPE) algorithm uses a non-conventional Bayesian optimization approach to surrogate-based optimization. Indeed, TPE models the likelihood $P(D_t|f)$ and the prior $P(f)$ by performing Parzen window density estimation (also known as kernel density estimation) [112], generating two separate distributions that specify high and low-quality regions of the search space. The algorithm ranks the observations in D_t based on its f values, and splits it into two sets,D_l and D_h, such that D_h contains a selected percentile of the highest quality observations, and D_l contains the rest. Exploiting non-parametric adaptive parzen windows, two probability distributions $h(x)$ and $l(x)$ are estimated from those subsets respectively. In other words, $h(x)$ represents the probability of a region in the input space to be associated with a high-quality observation, and $l(x)$ represents low-quality regions. These probability distributions are represented in Fig. 3.21

Finally, the surrogate model can be obtained by considering the likelihood of $\mathbf{x_{t+1}}$ belonging to the distribution h and l:

$$P(\mathbf{x}_{t+1}|f(\mathbf{x}_{t+1}), D_t) = \begin{cases} h(x) & \text{if } f(\mathbf{x}_{t+1}) < f^{\gamma} \\ l(x) & \text{if } f(\mathbf{x}_{t+1}) \geq f^{\gamma} \end{cases}, \tag{40}$$

where f^{γ} is the lowest function value associated with the γ-percentile according to which the split of D_t was made.

Lastly, the next points are typically sampled from $h(x)$, and the sample associated with the highest value of the acquisition function (i.e. the highest expected improvement) is then used to evaluate the objective function.

Another possible choice for the surrogate model is to take a Random Forest

regression model, where the ideas are similar to the other SMBO algorithms, but will not be presented for brevity sake's.

Gaussian Process As mentioned, Gaussian Processes-based Bayesian Optimization algorithms rely on constructing a probabilistic surrogate for the objective function after having accumulated some observations (i.e. few evaluation of the loss function), after which the next point to evaluate is purposed through the optimization of a surrogate function.

A Gaussian Process (GP) is an extension of the multivariate Gaussian distribution to an infinite stochastic process. This construction can be utilized as a surrogate model for the prior on f in Eq. 39. The GP is generated with a mean μ and covariance k for a normal distribution over possible values of $f()$

$$f \approx GP(\mu(\mathbf{x}), k(x, x')), \tag{41}$$

, where μ is often taken to 0, and the covariance function can be taken as a squared exponential function

$$k(\mathbf{x}_i, \mathbf{x'}_j) = \exp(-\frac{1}{2\lambda}\|\mathbf{x}_i\mathbf{x'}_j\|^2), \tag{42}$$

where is a parameter related to the smoothness.

When considering a set of priorly made observation D_t, the algorithm is expected to provide where to make the next observation. In particular, given the next observation (to be made!) $f_{t+1} = f(\mathbf{x_{t+1}})$, it is gaussian jointly with $\mathbf{f_{1:t}} = [\mathbf{f}(\mathbf{x_1}), ..., \mathbf{f}(\mathbf{x_t})]$. From this observation, the predictive distribution $P(f_{t+1}|D_t, \mathbf{x}_{t+1})$ can be derived [113]. Its expression constitutes the probabilistic surrogate model that replaces the true objective function.

A representation of the surrogate model with its variances is presented in Fig. 3.22.

The acquisition function then exploits the point distribution information to guide the search towards promising regions, driven by a high-quality mean, high variance, or both. After selecting an acquisition function, for example $\mathbb{E}\,\mathbb{I}$ introduced in Eq. 37, it remains to maximize $\mathbb{E}\,\mathbb{I}(\mathbf{x})$: a much easier task than to optimize the true loss function.

3.2.7 Transfer-Learning

Sometimes is possible to exploit previously learned knowledge in other situations or problems. Often, in new problems there's some kind of lack of data: transfer learning can help those settings, allowing to re-use similar datasets to guide the

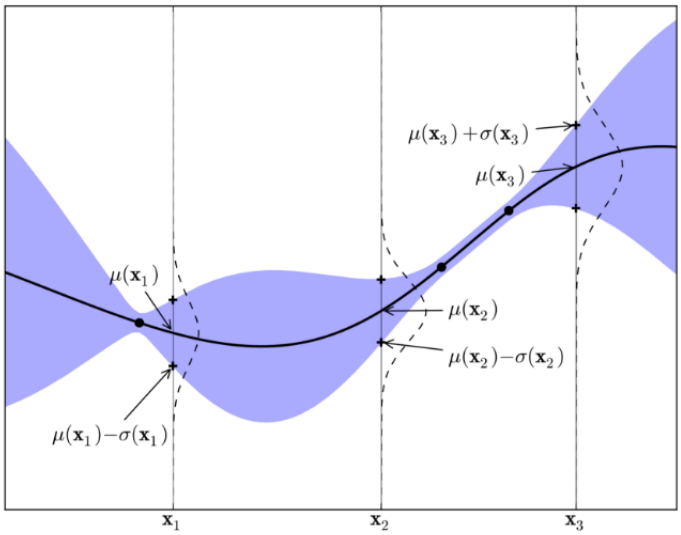

Figure 3.22: An illustrative example of surrogate function modelled as a Gaussian Process. Figure taken from [113].

learning problem in the right direction. Where what has been learned in one problem (from the data distribution P_1) is reused to improve generalization in another setting (P_2), we refer to **Transfer Learning** (and/or Domain Adaptation). The basic assumption behind these ideas is that many of the factors that explain the variations in P_1 are in the same way able to provide insights into the patterns that need to be captured for the learning problem P_2. This concept is easy to visualize when dealing with a classification task, where the input data can be very similar but the classification target may be of a different nature. Imagine we've learnt about one set categories, such as cats and dogs, in the first problem, and we would like to learn about a different set of visual categories, such as ants and wasps, in the second setting. Obviously, there is significantly more data in the first problem, and the first learning problem could help to learn suitable representations that allow to better generalize from only a few examples taken from P_2.

In summary, often classification problems (especially in the visual domain) share low-level notions of edges and visual shapes, the effects of geometric changes, changes in lighting, etc.

Ofen, transfer learning can be achieved exploiting the representation learning process that happens in the shared lower layers and the new data (from P_2) is used to refine the task-dependent upper layers.

Domain Adaptation, can be seen as a particular case of Transfer Learning: the task remains substantially the same between each setting, and the input

distribution is slightly changed. For example, consider a speech recognition system: when the system is fine-tuned over the new proprietary of a device, we are performing Domain Adaptation.

3.2.8 Applications

This section has the scope of making the reader understand why machine learning practitioners are so enthusiast of the deep learning technology, and why, especially in some domains such as Computer Vision or Natural Language Processing, it is considered incredibly successful. To do that, let's simply present a series of famous, curious and/or interesting applications:

1. **Self-Driving Cars**: Together with the latest hardware technologies, Deep Learning is literally bringing autonomous driving to life. For instance, the Uber Artificial Intelligence Labs at Pittsburg is not only working on making autonomous driverless cars, but they are also pushing towards the integration of several "smart features" such as food delivery options with the use of self-driving cars. Many data sources are processed by deep learning algorithm: data from cameras, sensors, geo-mapping etc. All this data flows is used to navigate through traffic, identify paths, signage, detect pedestrian, and more tools. Many other companies, such as Google, Tesla are working on similar researches.

2. **Natural Language Processing** (NLP): Natural language processing deals with the complexities associated with language and its comprehension and interpretation. When considering the syntax, semantics, tonal nuances, expressions, or even sarcasm, undoubtedly NLP can be considered one of the hardest tasks. Applications in this area relate to answering questions, language modelling, classifying text, twitter analysis, or sentiment analysis. Nowadays, distributed representations, convolutional neural networks, recurrent and recursive neural networks, reinforcement learning are pushing DL beyond its limits in NLP.

3. **Virtual Assistants**: a very popular application of deep learning is virtual assistants: consider for example Alexa, Siri, or Google Assistant. Those applications exploit deep learning to know more about their subjects and to implement recommender systems. Furthermore, They understand your commands which is a speech recognition task, still through deep learning. They can be viewed as one of the closest forms of "artificial intelligence" that exists, and it is mostly powered by deep learning.

4. **Visual Recognition**: Deep Learning can be used applied to images to perform an incredible amount of tasks: first of all, facial and identity recognition, but also sorting operations, grouping, labeling and annotations, image searching and many others. Large-scale visual recognition through deep neural networks is nowadays incredibly boosting the development of many practical applications in this domain of digital media management.

5. **Fraud Detection**: Deep Learning can be also used in the banking and financial sector, to detect frauds in digital transactions. Autoencoders are one of the most common techniques for anomaly detection. Deep Learning can identify patterns in customer transactions and credit scores to understand whether the current behaviour is anomalous.

6. **Healthcare**

   In the healthcare domain, in addition to several other successful applications, Deep Learning is often used to produce early, accurate and speedy diagnosis of several diseases, such as the methodology proposed in this thesis. Furthermore, it can help clinicians in their work with the development of specific assistants, or to better understand genetics patterns and predict future risk of diseases.

4 Experiments

This section is dedicated to the experimental methodology implemented in my thesis work. The objective is to provide a reusable pipeline that could allow the diagnosis of several diseases that could be identified by RS on salivary (or even serum) samples. The pipeline is semi-automated, having in mind a trade-off between human control, essential in any biomedical application, and autonomous processing, that could allow scientist to save precious time from boring, tedious data processing, model selection and model tuning.

The complete and general pipeline is presented in Sec. 4.1.

Its application to several pathologies, namely ALS, PD, AD, and the recently widespread infection due to COV-SARs-2, is tested and presented in Sec. 4.2.

4.1 The General Pipeline

As previously mentioned, a Machine Learning and Deep Learning approach is pursued to automate the analysis of the acquired RS data for unveiling hidden patterns that correlate with the pathologies object of this study.

An exploratory analysis of the available data is reported in Sec. 4.1.1.

For an efficient application of ML techniques to Raman spectral data, a series of steps must be integrated in an analytical pipeline aimed at reducing manual intervention and enabling an automatic classification task. The computational pipeline, characterized by a modular structure providing a flexible and systematic experimental framework, is depicted in Fig.4.1. The proposed pipeline includes four different phases: data preprocessing and augmentation, model selection, model tuning, model evaluation.

In particular, in order to build a classification model able to correctly classify patients through the analysis of their RS samples, the huge amount of information contained in a single Raman spectrum must be processed in order to identify similarity or distance between spectra and characteristic peaks. Spectral pre-processing is, therefore, an essential step for successfully applying both MultiVariate Analysis (MVA) and Machine Learning and can strongly affect the classification performances. Preprocessing details are reported in Sec. 4.1.2. By applying DL techniques it is possible to obtain two crucial objectives: reduce the dimensionality of the acquired data and train a classification model to recognize and classify the input spectra collected from the healthy person or from the patient affected by a specific pathology. In fact, Deep Learning based methods are well adapted to highlight the complex connections within high dimensional data provided by RS, being trained to recognize features in each spectrum that can be

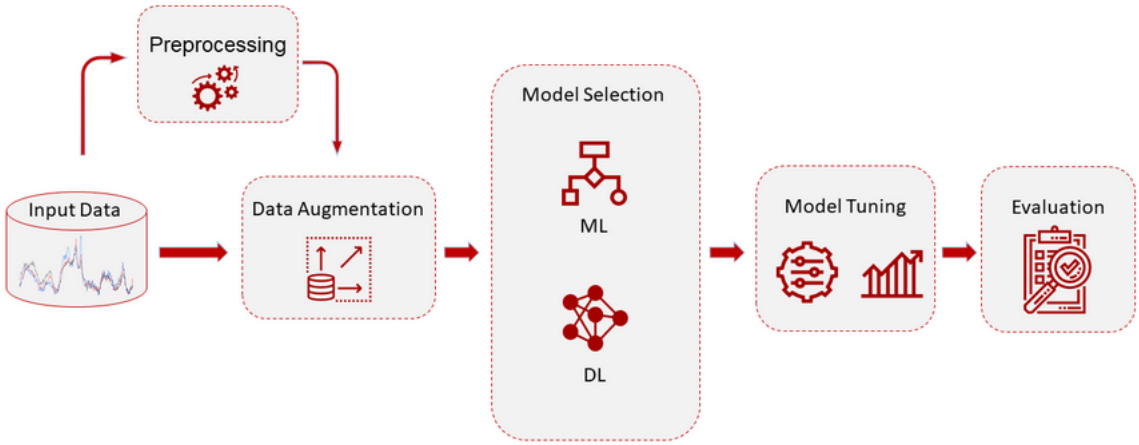

Figure 4.1: Diagram of the proposed pipeline. An optional preprocessing phase, suitable for ML models, is followed by a data augmentation procedure, after which a model selection step is performed by comparing several ML and DL models. The most promising models undergo hyperparameter optimization. The performances of the optimized algorithms are then evaluated in a Leave-One-Patient-Out Cross-Validation.

assigned to the proper label or experimental group.

Convolutional neural network (CNN) has been presented in Sec. 3.2.5.1 as an important branch of deep learning technology inspired by the biological visual cognition mechanism. In recent years, various methods have been developed for the effective training of deep CNN and CNN-based models have been proved to outperform the classical ML algorithms (such as Decision Trees, Support Vector Machine and Random Forest) in classifying Raman spectra [33]. In this work, we trained CNN models to classify noisy spectra of saliva samples collected from patients affected by several pathologies. Both the Machine Learning Baseline and the Deep Learning experiments are reported in Sec. 4.1.5.

There are a few challenges in the application of DL algorithms to Raman (or, more generally, biomedical) data. First of all, a large amount of data is required to learn classification models able to generalize avoiding the overfitting problem. Furthermore, the depth of such models leads to a complex structure characterized by hyper-parameters, such as the number of layers or the number of training epochs, whose tuning is extremely hard to be done manually. The first problem has been addressed by designing a data augmentation strategy aimed at generating new synthetic examples applying variations and distortions to the original ones, to maximally exploit intrinsic invariances in the data itself. An extensive review of this approach is presented in Sec. 4.1.3.

On the other hand, Automatic Hyperparameter Optimization, exploiting

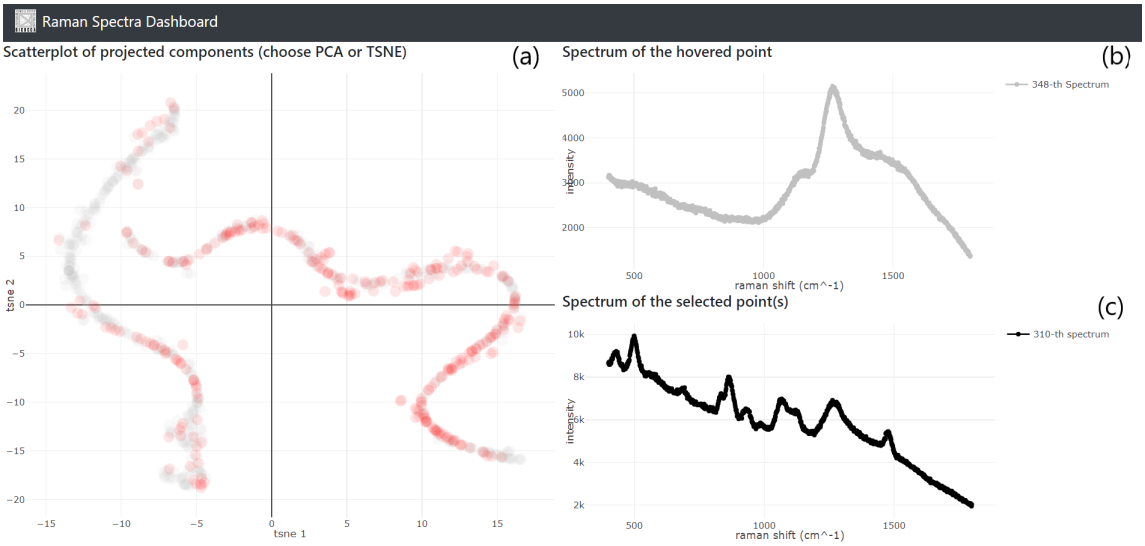

Figure 4.2: Screenshot of the interactive dashboard developed in order to facilitate the identification and removal of uninformative spectra. Panel (a): data represented using t-SNE dimensionality reduction. Panel (b): example of not well formed spectra. Panel (c): example of well formed spectra.

Sequential Model-Based Optimization and other techniques, has been employed to tackle the latter challenge with the intensive computation of (at least sub-)optimal hyperparameter configurations, as described in Sec. 4.1.6.

As the last step of this pipeline, a rigid performance evaluation scheme has been designed, as depicted in Sec. 4.1.4.

4.1.1 Exploration

This paragraph introduces and describes the datasets used for the different tasks described in the Introduction (see Sec. 1.2), their composition, and the processing treatment to which they have been subjected. In particular, we remind that our tasks include the classification of:

- ALS affected patients vs Healthy subjects (CTRLs).
  We will often refer to this task as "the ALS problem".
- PD affected patients vs AD affected ones (pathological controls) vs Healthy subjects (CTRLs).
  We will often refer to this task as "the PD problem".
- COVID-19 affected patients (COV+) vs patients who overcome the Sars-Cov-2 infection (COV-) vs Healthy subjects (CTRLs).
  We will often refer to this task as "the COVID problem".

The first case study can be seen as a sort of playground, because the available

data comes from only 19 ALS patients and 10 CTRLs. Therefore, our objective on this problem was to assess the capability of our models and techniques to deal with such a kind of data (spectral data), without focusing too much on the obtained performances. On the other hand, the other problems involved a higher number of subjects (67 and 101 total patients for the PD and COVID problem, respectively), that allowed to replicate the whole procedure applied to the playground problem to better suited case studies. In particular, the available datasets for those problems are described in details in Sec. 4.2.1, 4.2.2, 4.2.3 for the ALS, PD and COVID problem respectively, together with the data collection methods.

For an initial data exploration, a interactive dashboard has been developed (one for each problem). An example of such a dashboard applied to the ALS case is reported in Fig. 4.2. Such a tool can be used (and has been used) also for the semi-manual detection of outliers and non-informative spectra, as further described in the next paragraph.

4.1.2 Data Processing

Raw Raman spectra, obtained after the measuring process through a Raman Spectrometer, are not particularly suitable to be used directly for a classification task, since various noises sources are included in the measured signal, among which those intrinsic in the measuring process itself, linked to the frequency calibration of the spectrometer, or noise arising from the molecular fluorescence.

The preprocessing steps revealed to be useful in order to remove such unwanted contributions and to recover the pure Raman signal, which carries the most important information encoded in the spectra. This preprocessing can typically lead to a performance increase in many Machine-Learning-based models. It is worth to notice, however, that Deep Learning models are believed to be capable to handle directly raw data. Therefore, in the next steps of the pipeline, the ML baseline has been applied to preprocessed data, and DL models have been tested on both preprocessed and unpreprocessed, raw data.

In order to enable ML algorithm and to obtain more informative spectra for the purposes of classification we applied the following preprocessing steps according to [114]:

Removal of the *outlier* spectra that were not informative for the purposes of classification.

In our case studies, the spectral data reflects the presence and the concentration of certain substances in the saliva samples. A spectrum is therefore con-

sidered as an "outlier" when during the extraction phase it undergoes some issues including saturation, laser misdirection, cosmic rays interference, and others: the resulting datum is not informative since the signal-to-noise ratio is typically too low. Due to the nature of these data, the data cleansing operation is often manually performed by a domain expert and therefore it is highly time-consuming.

Our attempt was to define a pipeline that includes, among the other objectives, also the attempt to automate the pre-processing phases about outlier treatment using unsupervised techniques such as clustering. In particular, two different dimensionality reduction techniques have been applied before clustering the raw data:

- Principal Component Analysis (PCA) is an orthogonal transformation converting a set of observations of possibly correlated variables into a set of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance. This technique is one of the most frequently exploited in the literature to treat high dimensional data;
- T-distributed Stochastic Neighbor Embedding (t-SNE) [115] is a non-linear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions.

Once applied one of those techniques (let us take the t-SNE for instance), a DBSCAN clustering algorithm is tested to obtain two separate clusters. Although we found that the obtained clusters were not significant to discriminate uninformative and informative spectra, we noticed an interesting phenomenon, particularly remarkable for the ALS and the PD cases: after the clustering, moving along one of the identified clusters (reported in fig. 4.2 panel a), there seems to be a continuous transition between informative spectra (panel c), found in the higher part of each cluster (high values of the second t-SNE component), and less informative ones (panel b) moving towards low values of the second t-SNE component.

Therefore, a semi-automatic outlier removal method is obtained by manually observing the spectra using the interactive dashboard[6] developed exclusively for this purpose, identifying suitable threshold value for the second t-SNE component and retaining (for each clusters) only the spectra whose correspondent t-SNE second component projections have values above the established threshold[7].

[6]The dashboard is available at `https://ramandash.herokuapp.com/`

[7]This transition is particularly clear for the t-SNE projections. Thus, many different runs of the t-SNE algorithm were executed to make sure the phenomenon could not be caused by a fortunate run due to the intrinsic stochasticity of the procedure. The transitions were consistently found over every single run.

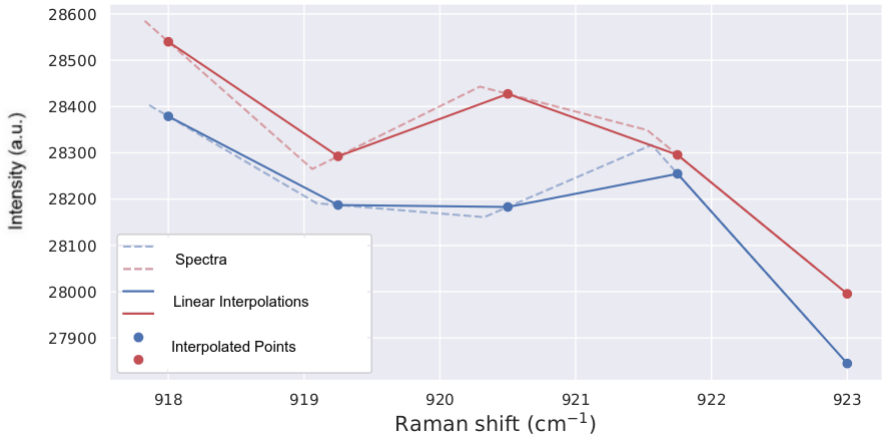

Figure 4.3: Two sample spectra are interpolated on a common grid of x-axis points.

However, this method was found to be effective only for the ALS case, and partially for the PD case. When the dimension of the dataset increases, the t-SNE embedding seems to be no-longer useful for discriminating non-informative spectra.

Therefore, a standard (and therefore less sensible) outlier preprocessing has been applied for the PD and the COVID cases. We considered outliers those spectra having an intensity which was either more than 10% null or completely saturated. This led to the removal of 50 spectra from each of the considered dataset.

Realignment of the Raman shift axis Raman Spectra acquisition can happen in various moments and conditions: therefore, some acquisition errors and noise due to the calibration of the laser frequencies of the spectrometer can arise, resulting in a misalignment of the Raman shift axis for distinct samples. For this reason, when acquiring a series of spectra, it is possible that a slight wavenumber shift occurs leading to a misalignment of the Raman peaks between the various spectra.

To the realignment scope, we applied a linear interpolation to resample each spectrum on a given grid of 900 points between 400 and 1600 cm^{-1}. The procedure is illustrated in Fig. 4.3. It's worth noticing that, although the misalignment is not a huge source of noise, its processing and removal can typically favour classification techniques.

Subtraction of the background signal given by the intensity of the aluminum substrate used for the SERS. Since the signal coming from aluminum, shown in Fig. 4.4, although characterized by a rather important band between

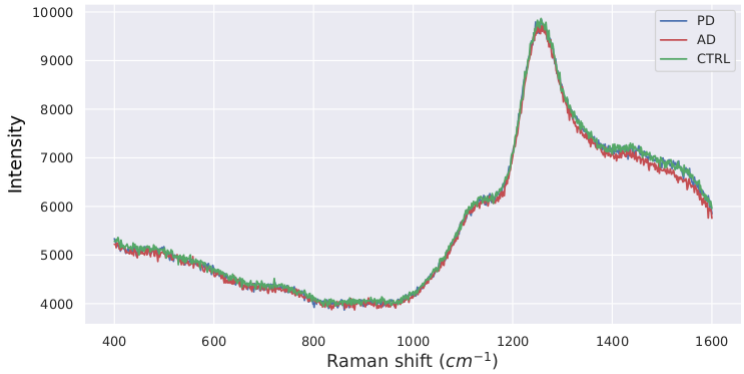

Figure 4.4: Example of background signal given by the aluminium substrates for the classes of the PD-AD problem.

1200 and 1300 cm^{-1}, is almost identical for each spectrum regardless of the class it belongs to, it has been simply subtracted from the original signal.

Removal of the background noise When acquiring a Raman spectrum, a background noise, given by the fluorescence of the molecules that are centered by the laser beam, can negatively influence the Signal-To-Noise Ratio that characterize the signal. The fluorescence is greater at higher energies, and therefore the laser diods of the spectrometers tend to be calibrated to emit lights in the Near-Infra-Red region, not to completely saturate the acquired spectra through sample fluorescence.

To remove this noise source, of of the most popular technique is based on Polynomial Fitting: a polynomial function is fitted through a grid of points sampled from the spectrum, and the baseline effect is removed. The higher the complexity in sample composition, the most probable to obtain an important fluorescence noise. Therefore, for highly complex samples, such as biofluids, a high-order polynomial fitting is required, in contrast to low-order (i.e. linear) fitting that might be enough for simple molecular systems.

In this work, the fluorescence of the samples, which causes a considerable increase in intensity, in handled through a baseline fitting with a sixth-degree polynomial. No signal smoothing was performed to preserve the original signal and avoid information loss. The baseline fitting procedure is summarized in Fig. 4.5.

Despiking Sometimes, it is possible that the measuring process is influenced by cosmic rays: cosmic particles, such as highly energetic Muons, can interact with the instrument, producing anomalous peaks in a spectrum, such as

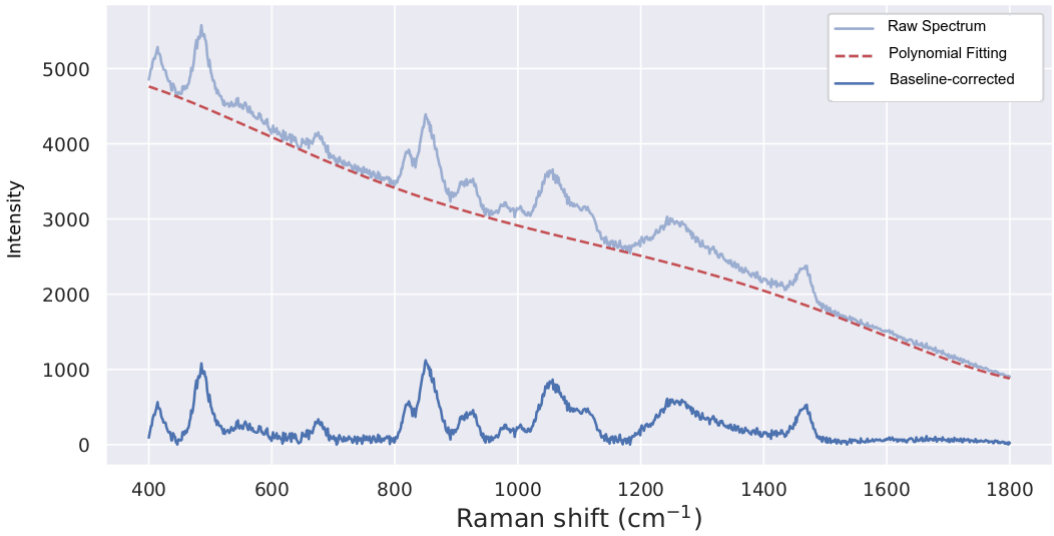

Figure 4.5: The baseline fitting procedure.

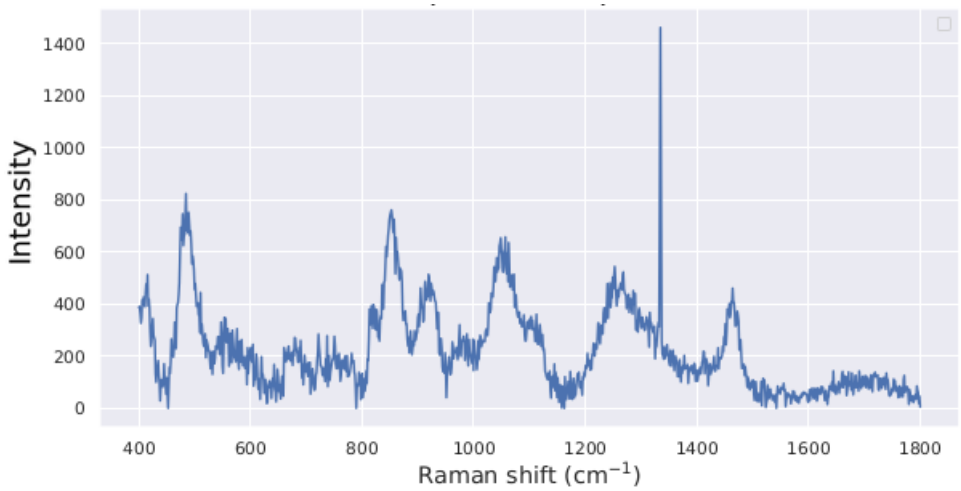

Figure 4.6: An example of spectrum with a spike in the spectral band in between 1200 and 1300 cm^{-1}: an anomalous peak caused by the interaction of a cosmic particle with the instrumental set-up.

the one reported in Fig. 4.6. Since these anomalies could potentially negatively influence the subsequent data analysis, it is important to identify cosmic rays interactions in the spectra and to remove them.

Therefore, any spike given by cosmic rays was removed by applying the Whitaker-Hayes algorithm [116], where the series of subsequent differences in a spectrum is taken into account in order to highlight peak anomalies. The threshold used to identify these spikes has been set to 3.5, as recommended by the authors, while the size of the neighborhood through which to correct the noise

peak has been set to 11.

Intensity Normalization Normalization techniques can be applied to Raman Data, for two main reasons: to enable a homogeneous comparison between spectra acquired in different conditions and possibly with different instruments, and to prevent numerical instability in the data analysis phase. Many intensity normalization techniques exist, but the most used for RS are:

- **Standard Normal Variate Normalization**, where a set of spectra is normalized according to the mean and the standard deviation of its intensities,
- **Unit Vector Normalization** (l_2), where each spectrum is normalized according to its l_2 norm,
- **Min-Max Normalization**, where each spectrum is normalized according to the minimum and maximum intensity peak,
- **Max Normalization**, where each spectrum is normalized according to its maximum intensity peak.

In order to select the best normalization method in terms of classification accuracy we compared the mentioned techniques against the performances of ML algorithms on the PD-AD classification task.

Nevertheless, the obtained results showed substantial robustness of the classification models with respect to these normalization methods. The l_2 normalization technique was then taken for the next experiments, given its wide diffusion in RS analysis.

Dimensionality Reduction When applying ML methods, usually practitioners tend to avoid the application on raw data, when those data have high-dimensionality: this is mainly to avoid performance issues, often caused by the curse of dimensionality [117, 118]. Therefore, a dimensionality reduction step is usually performed when processing spectral data, before to feed them to a ML model.

In this work, we tested ML models both on the original-dimensionality of the data, and after having applied the Principal Component Analysis (PCA). PCA reduces the dimensionality of the data by extracting a set of orthogonal components that explain the highest possible variance in the original dataset. For the whole study, we always select the first 100 PCA components, when applying dimensionality reduction.

4.1.3 Data Synthesis

As we mentioned in Sec. 3.1, the amount of available data largely influence the performances of any Machine Learning (or even more Deep Learning) model. In particular, when dealing with a complex problem, such as the one we are studying, where mostly non-statistically-significant differences are observed among features belonging to the various classes, the volume of the available data plays an important role, especially for data-hungry algorithms such as Neural Networks. In the cases depicted in this introduction, a Data Augmentation Step should be implemented.

Data Augmentation (DA), introducted in Sec. 3.2.4.3, is a procedure that generates new synthetic data samples by applying variations and distortions to the original data by maximally exploiting their intrinsic invariances. These techniques are largely used in Computer Vision but can be easily extended to other fields. As a result of DA procedures, the generalization performances of models are expected to improve while partially overcoming data scarcity. When the considered dataset appears to be unbalanced (in the PD-AD case, in favor of the CTRL class), we applied a data augmentation procedure making variations both in the offset and in the (global and local) slope of the original spectra and by introducing a Gaussian noise to the signals where new patients' data have been generated according to a different multiplication factor based on the degree of imbalance. It is worth noticing that data augmentation has been applied only to the training set, not to the test set. Since the performances have been measured in a Leave-One-Patient-Out Cross Validation fashion, as will be described in Sec. 4.1.4, the data augmentation has been performed online for each training folder.

Our Data augmentation strategy practically demonstrated to increase the performances of both Machine Learning and Deep Learning model in a significant proportion. The data augmentation strategy has been applied to each of the classification tasks. However, a comparative study has been performed only for the ALS and the PD/AD cases. Given the important results obtained via this study, we were able to demonstrate the importance of the data augmentation mechanism in alleviating the data scarcity while improving generalization power of the tested models. Therefore, when approaching the COVID-related task, we directly applied data augmentation without the need for a preliminary comparative study.

An overview of the results for the comparison of models with and without data augmentation for the ALS and PD/AD cases can be seen, respectively, in Tab 4.1 and in Appendix B, in relation to the classification models presented in paragraph 4.1.5.

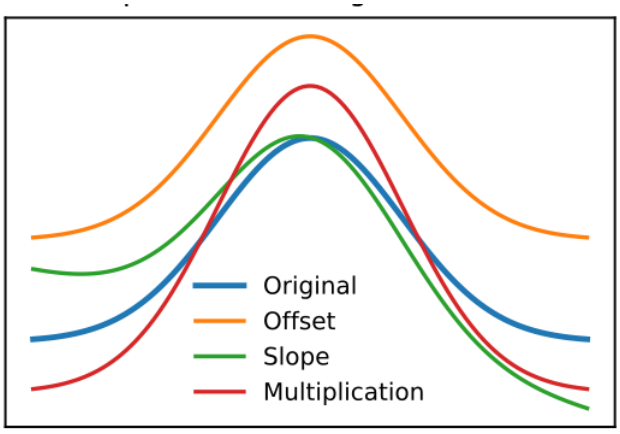

Figure 4.7: The original Raman band (blue line) and its clones after the augmentation procedure. Figure taken from [119].

4.1.3.1 Plain Data Augmentation First of all, in RS it is possible to simulate the spectral imperfections and variations characteristics of the measuring process by injecting a small contribution of Gaussian noise to the original spectra (at random wavenumbers). Beside direct noise injection, other augmentation components could be the stochastic modulation of the offset and the slope, as described in [119]. This has the following effects, also depicted in Fig.4.7:

- it slightly shifts the intensities, making the system invariant to weak translation along with the intensity axis it reshapes the Raman peaks according to a certain multiplicative factor
- it alters the spectrum slope, from a local (close to a peak) or global perspective.

An example of the augmentation effects on an entire test spectrum can be observed in Fig. 4.8.

4.1.3.2 Conditional Convolutional Variational AutoEncoder (CCVAE) Another option that was tested for the data augmentation was a Conditional Convolutional Variational Autoencoder. To the best of our knowledge, no such an approach has been pursued yet in the Raman-related literature.

Such an architecture is aimed at reproducing the input with slightly variations, that are caused by an intermediate layer in the middle of the network (in the information bottleneck), called reparametrization layer. In this layer, the encoded featured are re-sampled from the latent space from a Log-variate Gaussian Distribution: in this way, the output is re-constructed not from the original features, but from features that have been slightly "shifted" in the latent space. This resampling operation is usually known as the *Reparametrization Trick*.

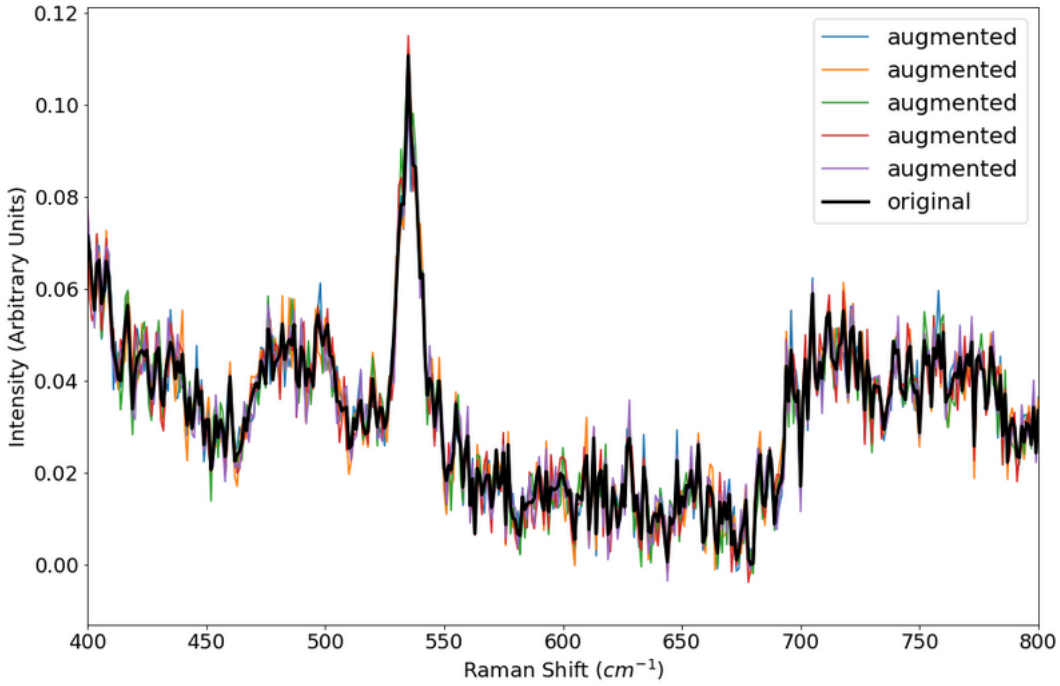

Figure 4.8: An example of the augmented results for a test spectrum.

The convolutional character of this architecture has been chosen in the attempt to match the design of the final model, and in addition due to the suitability of convolution to deal with spectral data, as assessed along the entire work. The "Conditional" term in the name of the architecture is given by the fact that this augmentation system is trained in a non-standard way by conditioning the output to the class label of the input: in this way, we assure that the results are as close as possible (in terms of overall information content) to the input, but still varying enough to allow a proper augmentation mechanism.

A high-level scheme of the CCVAE autoencoder is sketched in Fig.4.9, and an example of data augmentation through CCVAE is reported in Fig.4.10.

Unfortunately, the proper deployment of this augmentation system requires a lot of both computational and human time, given the numerous hyperparameters to be tuned (the most important being the dimension of the information bottleneck, and the parameters of the log-gaussian distribution from where the reparametrization trick is applied, and that determines the perturbation strength in the latent space).

On the other hand, using a trial and error architecture, the augmentation performances resulted to be very similar to those provided by the "standard" augmentation method: for this reason, the first proposed method has been pre-

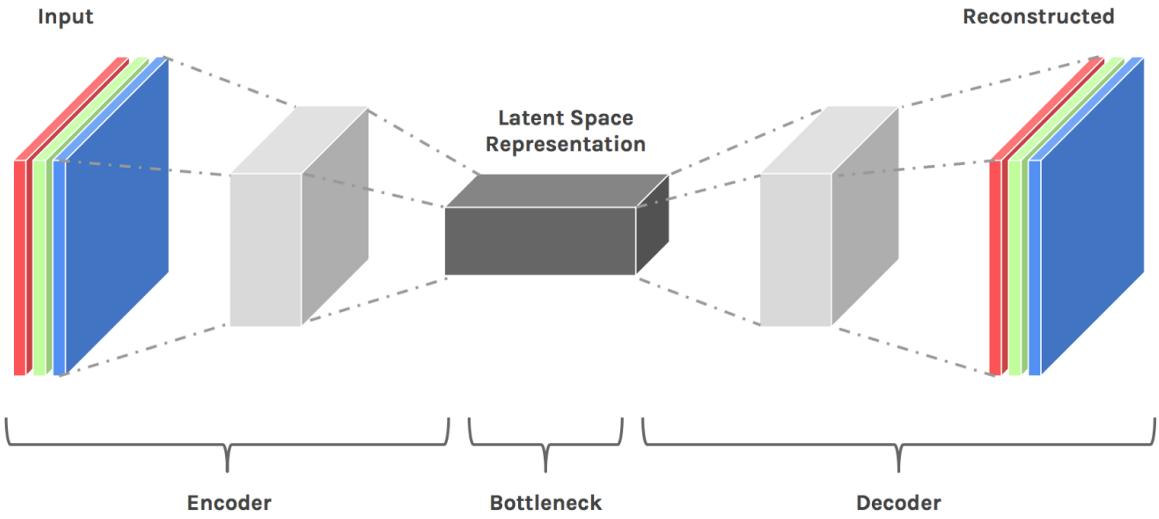

Figure 4.9: A diagram of a Convolutional Autoencoder, with the encoder section that compresses information, the reparametrization layer acting in the latent space and the decoder that reconstructs a slightly modified version of the original input. Figure taken from [120].

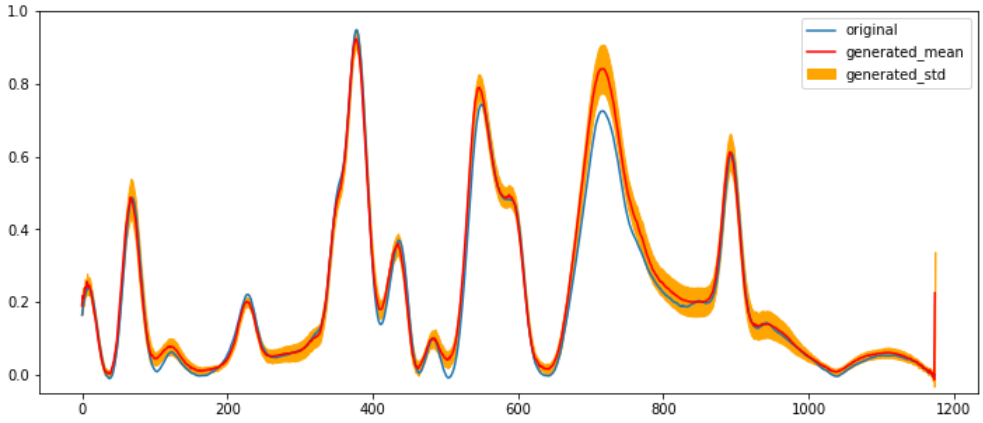

Figure 4.10: The original spectrum is 100x augmented through the CCVAE, and the result is reported in terms of a distribution about the original input. Units in the figure are arbitrary both for the X-axis and the Y-axis.

ferred as the main augmentation system, both because it is less complex and more interpretable and controllable, relegating the CCVAE approach to a future-work study.

4.1.4 Performance Evaluation

In order to evaluate and compare the performances of the models, we considered their ability in discriminating between disease-affected and CTRL patients. In particular, we considered the most common performance evaluation metrics, such as accuracy, precision, recall, f-measure. Furthermore, Sensitivity and Specificity were also taken into consideration, since most of the medical diagnosis procedures are evaluated in terms of those metrics.

It is worth to recall that for each saliva sample analyzed, an average quantity of 25 spectra are obtained. The developed models give a classification in term of the single spectrum. Therefore, since we have multiple spectra associated with each patient, each patient is classified according to the majority label assigned to his/her spectra.

In summary, we build a single spectra classification model and then we aggregate the classification result at the patient level.

However, the number of patients in the dataset is sometime still limited and this could compromise the reliability of the evaluation performance indices. For this reason, to overcome any classification bias given by an arbitrary test-set choice, we apply the Leave-One-Patient-Out Cross-Validation, a robust and stable procedure where each test-fold is composed of the entire set of spectra from a single patient (and the training-fold is obviously composed by the spectra of the remaining patients), that guarantees a most accurate estimate of the model performances.

This strategy has also the advantage that, for each fold, spectra belonging to the same patient are not mixed through the training and the test set. If this happened, a classification bias would arise, leading the performance of the model to a more optimistic estimate, because spectra belonging to the same patients tend to be more similar to each other, with respect to spectra coming from different individuals.

4.1.5 Classification Models

As mentioned in Sec. 2, in the last years, ML techniques, such as Support Vector Machine, Decision Trees, Random Forest, and Gradient Boosting, described in details in Sec. 3.1.4, have been increasingly applied for the classification of spectral signals. In particular, the method that is frequently reported to outperform other algorithms is SVM, e.g. in [33]. Random forests have also been used as a viable alternative to SVM for high-dimensional data with a large number of training examples.

In our study, we implemented SVM, RF and XGB as baseline ML algo-

rithms and compared them with different DL models, namely Fully Connected Neural Networks (FCNNs) and Convolutional Neural Networks (CNNs), whose theoretical background is presented along with Sec. 3.2.1 and Sec. 3.2.5.1.

For each problem, namely the ALS classification, the AD/PD problem and the COVID diagnosis, an initial comparison of the main algorithms has been carried out after an initial model configuration resulting from a fast gridsearch on the ML models and a trial and error procedure on the DL ones. In particular, the ML algorithms were initialized with their standard implementation provided by the framework we used (Sklearn), and a grid search optimization was applied to select the most important hyperparameters.

A more "complex" model, the FCNN, is implemented in Keras with a quick manual search of most suitable hyper-parameters. The network is therefore initially composed of three dense layers. The chosen optimizer is Adam with a default learning rate (lr= 0.001), while the batch size is set to 64.

With a perspective of increasing complexity, the last type of model tested is a CNN model, consisting of three convolutional layers and two dense layers. Both the DL models employ LeakyRelu activation functions, and makes use of a intense application of dropout in the fully connected layers, which is the only form of regularization applied (no norm-based regularizers are applied, since a smoothing-based result in the overall weight distribution is not desired, being the spectral data composed and characterized by several intensity peaks).

In good agreement with [33, 34], from this initial comparison,the convolutional models applied to each of the problems we faced seem to outperform the others (the extensive comparison is reported in Appendix). This intermediate result makes perfectly sense since CNN models have characteristic properties (such as local connectivity and translational equivariance, in addition to a weak translational invariance induced by the series of pooling operation) well suited for dealing with spectral data, where a "vertical" translational invariance (invariance to small changes in intensity, due to the error relative to the measurement process) and horizontal invariance (invariance to small shift along the x-axis, due to the laser calibration error during the spectral acquisition) are found to be relevant.

The combination of those characteristics seems to allow a CNN model to better treat raw un-preprocessed spectral data. This feature results to be useful since it permits to avoid a manual preprocessing of the data, resulting in saving a consistent amount of time to scientists.

The details of the tested models are reported in the Appendices. After this preliminary comparison, a hyperparameter optimization of the most promising models is performed, as described in the next paragraph (4.1.6).

The entire ML pipeline is coded in Python, and while ML algorithms have been implemented through the Sci-kit Learn Library [121], for the DL ones we exploited Keras, the Tensorflow high-level API [122].

4.1.6 Optimization

As we explained in details in Sec. 3.1.3, Performance of ML and DL models can be strongly affected by the value of their hyperparameters. Let's recall that examples of hyperparameters are kernels for SVM, the number of trees in a random forest, the number of layers in a DL architecture. The manual tuning of these hyper parameters is often impractical, in particular when dealing with DL approaches that require the selection of a wide variety of hyperparameters.

As extensively described in Sec. 3.2.6, many approaches to automatize HPO are available: rather simple grid-search or random-search, characterized by cheap computational requirements at the cost of modest performances, Tree Parzen Estimator (TPE), a lightweight Bayesian method whose cost/performance trade-off is competitive, and proper SMBO algorithms, whose performances tend to be superior given their obviously computationally intensive characteristics.

While for the optimization of ML models gridsearch proved to be enough in finding a suitable hyperparameters configuration, for the most promising DL model, the CNN architecture, SMBO (both GP- and RF- based) has been used to optimize its hyperparameters [123].

In this work, the HPO of the most promising model has undergone two main phases: the overall DL model structure optimization (selecting the number of Convolutional or Dense layers etc.) and the tuning of the "fine" hyperparameters (such as the number of convolutional filters for each convolutional layer, the filter's width etc.). In addition, during the optimization task, the hyperparameters of the data augmentation generator, controlling the strength of the perturbation applied to the original spectra, are tuned.

The cited tasks are performed starting from a baseline model with fixed hyper-parameters and exploiting the Optuna framework [124] in combination with the Scikit-Optimize [125] library.

The choice of Optuna is guided by the need to use a performing and efficient Bayesian optimization framework. The performances are given by the possibility of integrating, in addition to the native Tree-structured Parzen Estimator (TPE) sampler able to achieve good results in reasonable computational time, a Sequential Model-Based Optimization (SMBO) algorithm provided by Scikit-Optimize, allowing one to choose base estimators such as Gaussian Process (GP) or Random Forest regression and acquisition functions (e.g. Lower

Confidence Bound LCB). The efficiency is given by the native implementation of a pruning mechanism, that allows to discard unpromising evaluation and save computational time. In addition to this, Optuna also provides a set of features for displaying optimization processes that allow one to properly visualize how the hyper-parameter search space has been explored; examples of these visualizations are reported in Figs. 4.12, 4.13, 4.14.

First of all, starting from the baseline architecture described in the previous paragraph 4.1.5 and with fixed hyper-parameters as reported in the Appendix, an architecture search is performed with 500 evaluations. This search consists in determining how many convolutional layers are needed (from one up to three), whether each convolutional layer should be followed by a MaxPooling and Batch-Normalization layer or not, the magnitude of the dropout rate placed between the convolutional and dense block of the network, how many dense layers to use (always between one and three), which optimizer to choose between Adam, Nadam and RMSprop and what learning rate value to use. Considering that the majority of the sampled values are True/False boolean values, i.e. do include a layer into the architecture or not, the RF has been chosen as the base estimator with LCB acquisition function.

Following the initial architecture search, the data augmentation hyper-parameters optimization is performed. The generator has three main parameters : *slopeshift*, *betashift*, and *multishift*, as introduced in paragraph 4.1.3 and extensively described in [119]. Aiming at obtaining sensible values for such parameters, an optimization procedure is applied using fixed hyper-parameters for the neural net and the architecture resulting from the previous phase, running 100 evaluations. Since the sampled values are continuous, GP is exploited as a base estimator in combination with the LCB acquisition function.

As a further check on the network architecture, using the optimized parameters for the data augmentation mechanism, the architecture search is performed once again with 50 evaluations (RF and LCB have been used respectively as base estimator and acquisition function), but with the training data augmented; in such a way the model should generalize better and be more robust, since it includes the contribution of the data augmentation procedure.

In conclusion, the neural network hyper-parameters including, for example, the number of filters for the convolutional layers, the kernel size, etc. are optimized performing $\tilde{5}00$ evaluations (although the pruning mechanism makes it difficult to properly count the number of *effective* iterations) , with the overall fixed structure resulting from the previous steps. Just as it was for the data augmentation optimization phase, in this case the sampled values are continuous: for this reason, GP and LCB have been used.

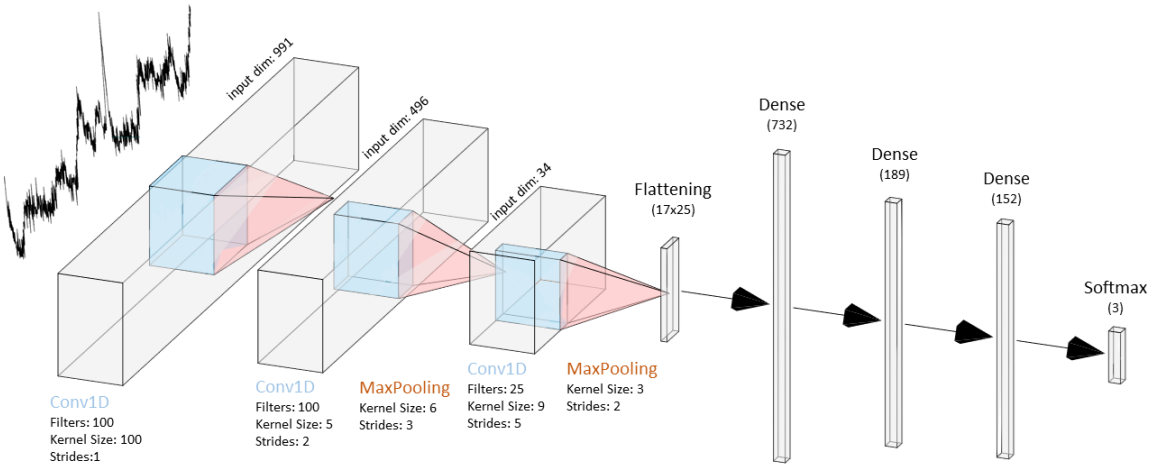

Figure 4.11: Diagram of the Convolutional Neural Network model with the hyper-parameters obtained through the optimization procedure for the COVID dataset.

It is worth noting that all the optimizations are performed in 10-folds cross-validation (with the folds always grouped by the patient-id, not to mix spectra belonging to the same patient in the training and the test phase) and the objective function is minimized on the average classification error (i.e. $1 - \text{accuracy}$).

The CNN model resulting from this optimization phase is distinct according to which problem is considered. For brevity, we report only the final model obtained when optimizing in the context of the COVID problem, whereas the other obtained models can be found in Appendix A and B for, respectively, the ALS and the PD task.

Fig. 4.11 reports the final CNN model obtained for the COVID dataset: it is composed by three 1D convolutional layers, the last two followed by a maxpooling operation and batch normalization, and 3 dense layers, with an intense application of dropout, being it the only form of regularization besides data augmentation.

The optimization history for the mentioned problem is summarized in Fig. 4.12, whereas Fig. 4.13 illustrates the exploration of the hyper-parameters search space performed by the SMBO algorithm during the optimization task. To the same extent, Fig. 4.14 reports the contour plots related to a few selected hyperparameters.

The classification results obtained thought the presented models are reported in paragraph 4.2.3.

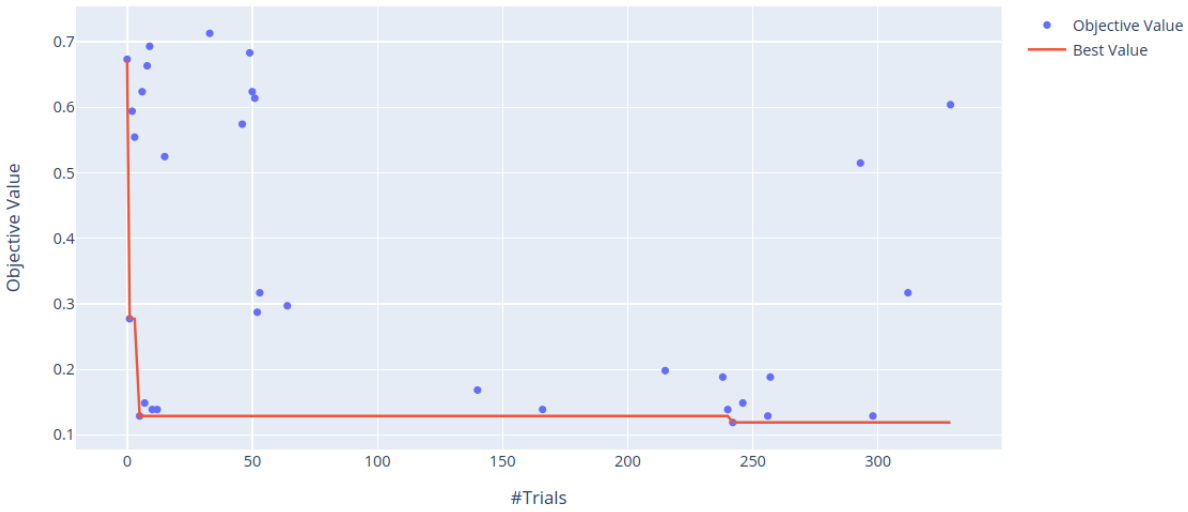

Figure 4.12: Optimization history on the COVID dataset: each point represents the output from an optimization epoch, the red line represents the best seen path. Figure taken from [2].

4.1.7 Transfer Learning

In the vein of Sec. 3.2.7, an attempt was made to implement a transfer learning and domain adaptation approach for spectral data. T o this extent, the LABION group provided us with a spectral data collection composed of more than 1000 spectral instances belonging other not only from human saliva samples but mostly from mouse exosomes and serum from patients with Alzheimer's disease. The transfer learning workflow is initially organized starting from the final model (optimized following the procedure explained in Sec. 4.1.6). An auto-encoder is implemented using the convolutional block of the final model (thus discarding the dense block) as an encoder, generating the decoder as its symmetrical structure. Such an auto-encoder is then trained on various spectral data (saliva samples, mouse exosomes, and AD serum). After the training procedure, the encoder component is extracted and integrated into a new neural network with the same overall structure of the optimized model from the optimization pipeline.

The diagram in fig. 4.15 summarizes the procedure .

During the training of the transfer learning model, in order not to perturb too much the weights obtained by the auto-encoder, the learning rate is halved for all the epochs, with respect to the learning rate obtained at the end of Sec. 4.1.6. It is eventually possible to define a model with the same architecture of the one obtained after the optimization pipeline but with pre-initialized weights thanks to the auto-encoder procedure. The performances of the transfer learning model are

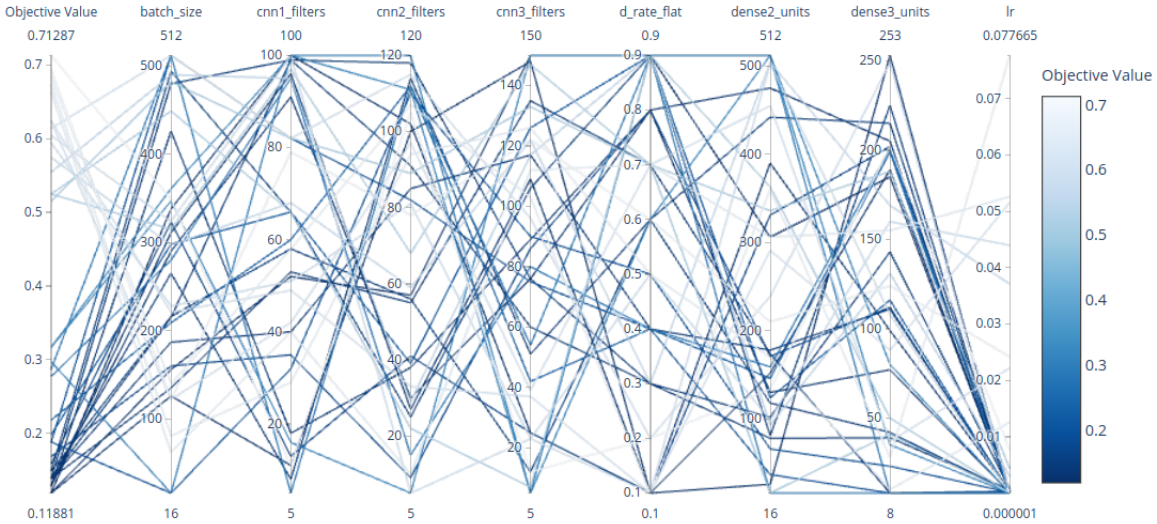

Figure 4.13: Hyper-parameters search space exploration during the CNN optimization task on the COVID dataset: each line represents the values used for each hyper-parameter and the objective value reached from those values. From the visualization it is possible to observe how the optimization mechanism has improved exploration during the epochs.

then compared to those of the plain model (with no pre-training) with a unified training schedule (excluding the learning rate).

This approach has been experimented in particular for the ALS/CTRL classification problem described in Sec. 4.2.1, where the data scarcity was particularly outstanding, and the final result of this attempt is reported in the last row of Tab. 4.1.

However, we realized that a larger volume of pretraining data was needed to guarantee the usefulness of such an approach, and we omitted this part of the analysis for the remaining classification problems.

4.2 Diagnostic Applications

4.2.1 Raman fingerprint for the diagnosis of Amyotrophic Lateral Sclerosis (ALS)

4.2.1.1 Data ALS Patients (pALS) were recruited if they had previously received a diagnosis and they were male between 50 and 85 years old at entry. Exclusion criteria for pALS were represented by concomitant obstructive respiratory diseases; renal failure; cardiovascular, oncological, immune, hematological and psychiatric diseases; bacterial or fungal infections in progress (e.g. oral candidiasis). CTRL were considered if they did not constantly and continuously

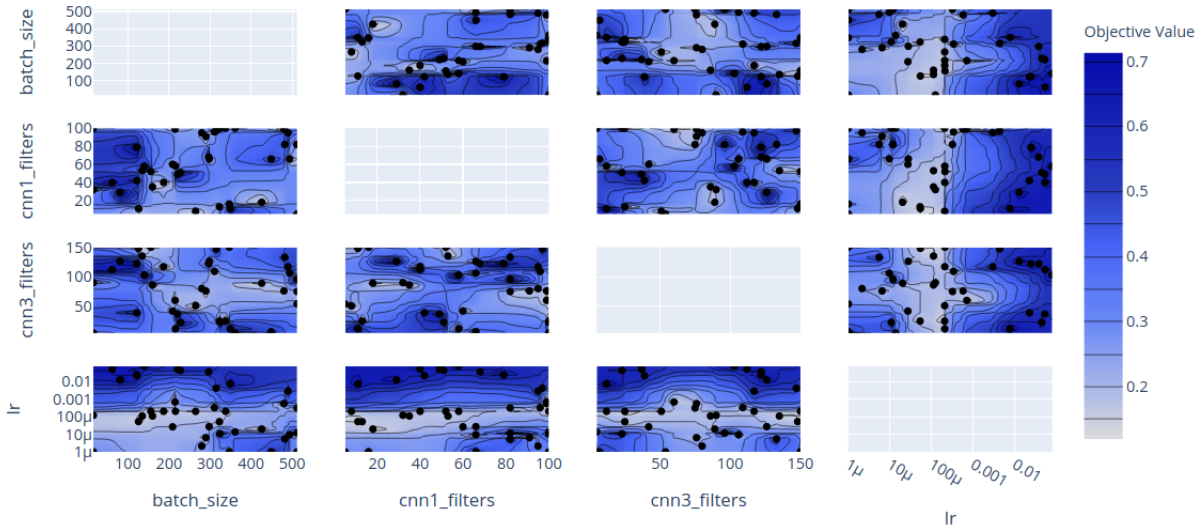

Figure 4.14: Contour plot for the optimization over the COVID dataset: at the end of the optimization phase, this visualization shows a contour that gives an intuitive spatial view of the search space.

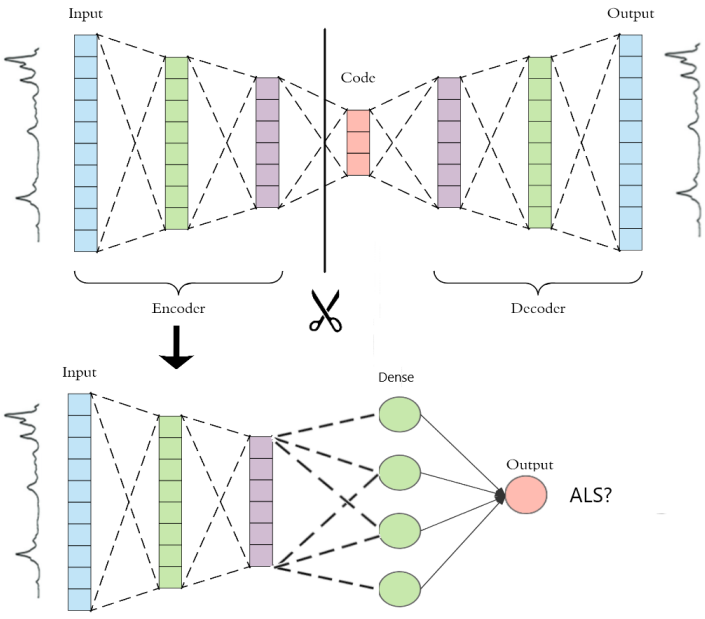

Figure 4.15: Transfer learning workflow: after the first phase which includes an auto-encoder training using spectral data from ALS patients, mouse exosomes, and AD serum, the encoder component is employed to create a new model adding fully connected layers at the end.

take drugs (e.g. anti-hypertensive and anti-diabetic drugs) and did not report chronic and inflammatory dis-eases, particularly the oral cavity. Furthermore, the pathological and not pathological control groups were strictly age matched to pALS and only male subjects were recruited to limit sex hormone variability in saliva. It is well known that the chemical composition of saliva is influenced by the presence of hormones and how this affects a different Raman signature of the saliva of both men and women.

pALS and CTRL who consecutively accessed to the IRCCS Fondazione Don Carlo Gnocchi, in Milan (Italy) between 18th October 2017 and 22nd December 2020, were recruited.

19 ALS patients (n = 19) and 10 CTRL (n = 10) met the relevant eligibility criteria for being included. Demographic information as well as clinical data (age and comorbidities) were collected for all the participants.

To limit variability in salivary content not related to ALS, saliva was obtained from all subjects at a fixed time, after an appropriate lag time from feeding and teeth brushing. Pre-analytical parameters (i.e. storage temperature and time between collection and processing), dietary and smoking habit (when provided) were properly recorded. The swab for the salivary collection was removed, placed in the mouth and chewed for 60 seconds to stimulate salivation. Then the swab was centrifuged for 2 minutes at 1,000 g. Collected samples were stored at 80 °C. Before the Raman acquisition, saliva samples were filtered with different cut-off ranges, collecting and analysing by RS the eluted sample and discarding the concentrated counterpart. Raman and SERS spectra were acquired using an Aramis Raman microscope (Horiba Jobin-Yvon, France) equipped with a laser light source operating at 785 nm with laser power ranging from 25–100%. Acquisition times between 10-30 seconds were evaluated. The instrument was calibrated before each analysis using the reference band of silicon at 520.7 cm^{-1}. A drop of saliva (3 μL) was dropped on the designed substrate (Glass, Calcium Fluoride or Aluminium foil) and dried at room temperature. Raman spectra were collected from at least 10 points following a line-map from the edge to the centre of the drop. Spectra were acquired in the region between 400 and 1800 cm^{-1} using a 50x objective (Olympus, Japan). Spectra resolution is about 1.2 cm^{-1}.

The software package LabSpec 6 (Horiba Jobin-Yvon, France) was used for map design and the acquisition of spectra. SERS measurements were performed mixing saliva samples with AuNPs and AgNPs with a variable ratio (1:9 and 5:5) and incubated for 30 minutes before casting.

The data collection methods have been created and refined by the LABION research group, and the presented summary is partially extracted from [126].

4.2.1.2 Results To resume and complete the methodology discussion in Sec. 4 with its results, a series of summary tables are presented. In particular, a comparison between a machine learning model (SVM), a fully connected neural network model and a convolutional model, reported in Appendix A, leads to the selection of the CNN approach as the most suitable (to our judgment) for the objective task.

Starting from Tab. A.1 in Appendix A, in particular referring to the "Value" column, it is possible to observe the final neural network architecture reached during the optimization procedure.

Tab. A.2 and Tab. A.3 report the optimal values obtained respectively for the parameters of the data augmentation mechanism and the hyper-parameters of the CNN model.

As a further summary representation of the optimized model, Fig. A.4 is presented in Appendix A.

Moreover, the classification results obtained during the development of the project are shown in Tab. 4.1, which includes both an analysis of the data augmentation procedure presented in Sec. 4.1.3 and the transfer learning attempt described in Sec. 4.1.7. The results are obtained using a fixed seed of the random number generator permitting to evaluate the models' performance on identical training, validation and test sets.

For the ALS/CTRL classification problem, given the low amount of patient-level available data, we were not able to retain a train/test split that takes into account the variability given by the group structure of patients. Therefore, we simply analyse the results in terms of average accuracy from a regular 10-folds cross-validation (therefore, not stratified over patients, as described in Sec. 4.1.4 and as applied to the remaining classification problems, where more data were available) and its relative standard deviation. We will further remark what are the implications of this evaluation choice, together with a discussion regarding the classification results, in Sec. 5.1.

4.2.2 Salivary Parkinson's disease (PD) fingerprint

4.2.2.1 Data AD patients were recruited according to the diagnosis of dementia due to AD following the McKhann et al guidelines and diagnosis of MCI due to Alzheimer's Disease following the clinical criteria described by Albert et al, excluding patients with neurological or major psychiatric comorbidities. PD patients were diagnosed according to the Movement Disorder Society Clinical Diagnostic Criteria for PD, excluding patients affected by vascular parkinsonism (with cerebrovascular disease, as indicated by brain imaging computed tomog-

Table 4.1: Summary of the results reached using the realized neural networks and measured in 10-folds cross-validation for the ALS dataset: CNN baseline is the first neural net realized and it is mentioned in Sec. 4.1.5, CNN OPT and CNN OPT-Aug refers to the neural nets obtained from the optimization procedure respectively without and with data augmentation. CNN TL, discussed in Sec. 4.1.7, refers to the model trained following the transfer learning workflow.

Results	Mean (10-folds CV)	Std
CNN baseline	86.25%	(±5.90%)
CNN OPT (no Aug.)	89.29%	(±5.92%)
CNN OPT (Aug.)	90.91%	(±3.24%)
CNN TL (no Aug.)	89.41%	(±3.67%)

raphy or magnetic resonance imaging, or by the presence of symptoms that are consistent with stroke). CTRL were recruited and selected according to the age and sex of the AD and PD patients in order to limit sex hormone variability in saliva, that can affect the Raman signature of saliva of both men and women. Exclusion criteria for CTRL were the constant and continuous drug administration (e.g. anti-hypertensive), the presence of chronic or/and inflammatory oral or systemic diseases.

PD patients and CTRL were recruited after the access to IRCCS Fondazione Don Carlo Gnocchi, in Milan (Italy) between October 2017 and June 2020. AD participants were recruited at Istituto Auxologico Italiano, IRCCS Department of Neurology and Laboratory of Neuroscience.

For this study, a total number of 24 PD (n=24), 10 AD (n=10) and 33 CTRL (n=33) were selected for the saliva collection. For all the participants, demographic, personal and clinical data were recorded.

For the Saliva collection procedure, the swab was placed in the mouth of the subject and chewed for 1 minute in order to stimulate salivation. To limit the results variability, the salivary collection time was fixed at an appropriate period from the last meal (3h) and from teeth brushing (2h). Storage time and temperature, participants smoking and particular dietary habits, time between the collection and the Raman analysis were recorded. The swab was consecutively centrifuged at 1000g for 2 minutes in order to recover saliva. Collected samples were stored at -80° C when the immediate processing was not possible. Raman spectra were acquired using a Raman microscope Aramis (Horiba Jobin-Yvon, France), equipped with a laser source at 785 nm and adjustable laser power emission.

The analysis were performed after instrument calibration with the reference

band of silicon at 520.7 cm^{-1}, using 30 seconds acquisition time. 3 μL of saliva was cast on an aluminium foil in order to achieve the Surface Enhanced Raman Scattering (SERS) effect and dried at room temperature. Raman analysis was performed using a square-map (80μm x 60μm) close to the centre of the drop, with the acquisition of at least 30 points for each subject. The acquisition range was set between 400 and 1600 cm^{-1} for the analysis on aluminum, while the range was restricted to 400 - 1500 cm^{-1} for spectra collected on calcium floride disks. All the analysis were performed using a 50x objective (Olympus, Japan) and with a spectral resolution of 0.8 cm^{-1}/step. The laser grating was set at 600 while the hole was kept at 400. All the methods described in this work were performed in accordance with the relevant guidelines and regulations.

The data collection methods have been created and refined by the LABION research group, and the presented summary is partially extracted from [1]

4.2.2.2 Results Briefly summarizing the methodology described in Sec. 4 and adapted for the PD/AD classification task, we remind the reader that the application of DL techniques to spectral data involves a series of challenges.

As a result of the high complexity (and capacity) of such DL models, the available data volume should be large enough to create a uniform and coherent dataset, enabling a successful training phase. In our case, given our intention to perform a patient-level classification, in the vein of Sec. 4.1.4, the data distribution strictly depends on the number of individuals: the higher the number of patients, the most uniform and coherent the data distribution will be, and, viceversa, the fewer the patients the harder the training. Such a consideration led us to the design of a DA protocol, resulting in the generation of synthetic spectral examples to favour the network training and to boost the classification performances.

Furthermore, higher complexity (with respect to ML models) leads to a large number of possible configurations for the components of a DL model. A suitable architecture must be selected by composing various layers and, especially for CNNs, the configuration design requires a great effort in selecting the hyperparameters. Usually, this design is obtained by a trial and error method, which could be time-consuming and require particular expertise, without any guarantee of convergence to an optimal configuration. As described in Sec. 4.1.6, we optimized the CNN architecture and fine-tuned its hyperparameters through SMBO: in Appendix B, Fig. B.4 shows the final configuration of the model.

In particular, our CNN architecture consists of three 1-D convolutional layers for the feature extraction and three fully connected layers for the classification. The details for the optimization process related to the PD/AD task are reported

Table 4.2: Accuracy, sensitivity and specificity at Patient and Spectra-level obtained by the proposed CNN in Leave-One-(Patient)-Out Cross-Validation for the PD task.

	Accuracy	Sensitivity	Specificity
Patient-level	0.881	0.873	0.923
Spectra-level	0.798	0.793	0.883

in Appendix B.

We empirically show (in Appendix B, Tab. B.1) that the optimization strategy results in increased model performances.

Given the difficulty of the learning problem, characterized by a low-volume high-dimensional dataset, both the ML and DL models have been trained on preprocessed data. An initial evaluation of the possible normalization methods led us to the choice of $l2$ as the most suitable normalization technique for our experiments.

In addition, we performed classifications on raw unprocessed data, and we verified, in good agreement with Liu et al., that DL models, in particular CNNs, are capable of gaining competitive performances without the need for any preprocessing on the data. Therefore the "raw-CNN" performances will be reported along with the others.

The deep learning architecture is trained on the 3-class classification problem outputting a probability distribution associated with the class choices, where the highest probability is used to predict the label. The model is tested via a Leave-One-Patient-Out Cross Validation method, and performance breakdowns are reported in the followings, both under a single-spectra perspective (spectra-level) and under the patient-classification perspective (patient-level).

The average spectral-level accuracy for the CNN model is 0.798 on normalized data (0.73 on raw data). Complete Metrics, including Sensitivity and Specificity, are reported in Tab. 4.2.

In comparison, for our baseline methods, SVM and RF achieve accuracies of 0.77 and 0.746, respectively, while the rather simple DL model, the FCC network architecture, scores an accuracy rate of 0.77.

Tackling the patient-level classification, it is worth to remind that the labelling decision for a patient can be taken based on its entire spectral set (since numerous spectral samples are acquired from the same individual). Furthermore, the percentage of spectra belonging to the same patient and correctly classified is itself a confidence indicator.

Classification metrics can thus be condensed into a confusion matrix grouped

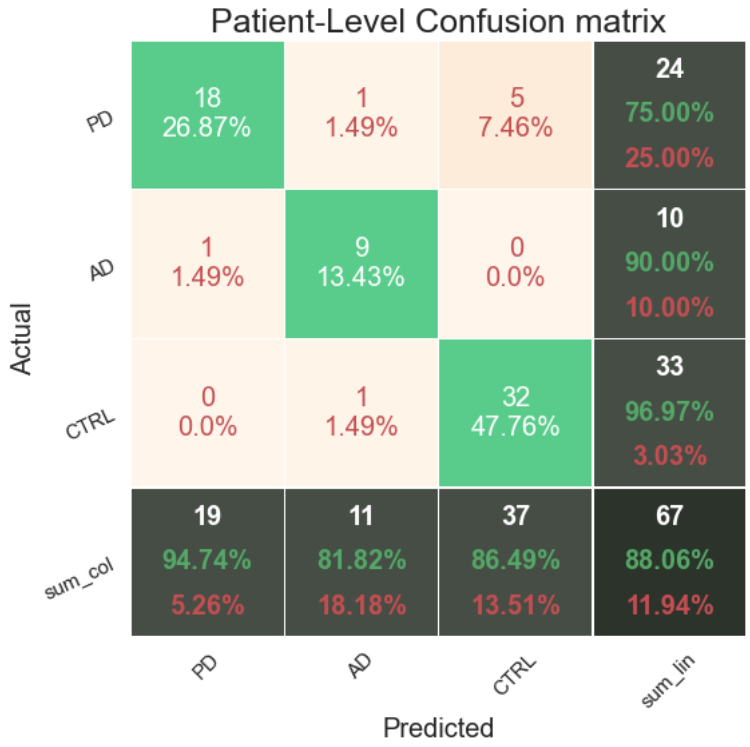

Figure 4.16: Confusion Matrix obtained from the classification of 67 patients of the PD classification task through the proposed CNN model via Leave-One-(Patient)-Out Cross-Validation.

by patients, reported in Fig. 4.16, where the average accuracy score of the CNN model is 0.881 on normalized data (0.875 on raw data). In comparison, RF and SVM have both accuracy rates of 0.836, while the FCNN score is 0.85. Complete Metrics for the patient-level classification are reported in Tab. 4.2.

The details of the extensive comparison at patient-level of the ML baseline and DL models, also tested against the use of data augmentation, are reported in Tab. B.1 in the Appendix. The classification results are extensively discussed in Sec. 5.2.

4.2.2.3 Error Analysis Observing the classification results, It is possible to notice that PD patients are, at a first glance, more prone to be misclassified as CTRL cases. On the other hand, AD and CTRLs individuals are usually correctly classified (no confusion between those classes). Indeed, during the testing of our ML/DL pipeline, we found that AD and CTRLs patients are usually correctly classified by all our models. Conversely, patients belonging to PD are more prone to be misclassified.

When carefully investigating over a number of patients that were misclas-

sified by every model we tested on them, a closer exploration of their spectra revealed some possible anomalies that could lead to the misclassification. Graphical analysis is shown in Fig. 4.17, where statistically significant differences can be observed between the average spectra of the selected patient compared against the mean spectra of all the other patients belonging to the same class. In particular, similar patterns are found along with the other frequently misclassified spectra.

The patients PD_11 e PD_24 are misclassified as CTRLs, whereas the patient PD_07 is constantly labelled as Alzheimer-affected. Their spectra show similarities with the average spectra of the misclassified classes. For example, Fig.4.17 panel a) shows the comparison between the average spectra of PD_07, of the PD class (excluded PD_07) and of the wrongly labelled class, AD. It's worth noting that at 850 cm^{-1} the PD_07 spectra lacks of a characteristics PD peak, following more the general enveloping of an AD spectrum. The same observation holds for the spectral bands at 1000 and 1050 cm^{-1} Fig.4.17 panel b) shows the same analysis for the PD_11 patient: in this case, an anomalous band is reported at 900-1000 cm^{-1}, with no correspondence neither in the PD class nor in the CTRL one.

Based on those anomalous observations, the suspected patients have been crosschecked with respect to their clinical information, in order to identify any correlations that could possibly explain the negative results in the classification task. It turned out that PD_11 e PD_24 suffers from a particularly early-stage Parkinson, which could easily explain the overall similarity of their spectra with those typical of the CTRL class. On the other hand, PD_07 suffers from a concomitant rare form of neurodegenerative cerebellar atrophy, which is rather similar to Alzheimer under a clinical perspective. This observation could explain the similarity between the Raman fingerprint of PD_07 with respect to the typical spectra of the AD class.

It's important to notice that this information can be positively taken as a chance to further refine the data collection protocols and data processing methods (such as filterings) in the future.

4.2.3 The Raman COVID-19 salivary fingerprint

4.2.3.1 Data Saliva samples, health records and clinical data were acquired at IRCCS Fondazione Don Carlo gnocchi ONLUS, Santa Maria Nascente Hospital, Milano, Italy and Centro Spalenza Hospital, Rovato, Italy between 16th April 2020 to July 2020. The COVID-19 diagnosis was conducted following the World of Health Organization (WHO) guidelines, declaring a positive case after

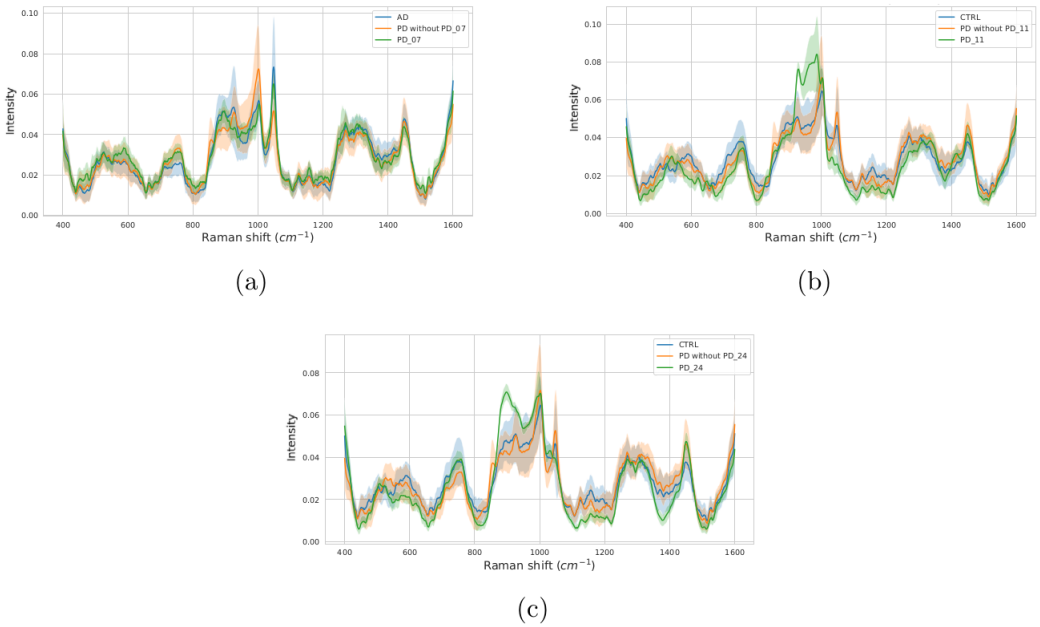

Figure 4.17: Average and std. dev. of the spectra belonging to: a PD patient (green), patients belonging to the PD class (orange), and patients belonging to the wrongly predicted class (blue).

the positive result of sequencing or Real-Time reverse-transcription Polymerase Chain Reaction (RT-PCR) assay of SARS-CoV-2 for nasopharyngeal swabs.

The patients were considered COVID-19 negativized after two consecutive tests with negative results, recording the time of negativization as the time between the last SARS-CoV-2 positive test and first negativization assessment (two consecutive negative SARS-CoV-2 tests).

Demographic data of patients and healthy subjects were recorded, as well as eventual comorbidities (e.g. diabetes, cardiovascular diseases, arterial hypertension, neurological disorders, respiratory obstructive diseases and cancer or concomitant viral or bacterial infections) and pharmacological and/or invasive or non-invasive respiratory therapies.

The total number of subjects involved in the study was: 30 patients affected by COVID-19 (COV+; n=30), 37 subjects negative to the SARS-CoV-2 test with an ascertained episode of COVID-19 (COV-; n=37) and 34 age and sex correlated Healthy Subjects (CTRL; n=34). The data acquisition was performed with the same protocol described in 4.2.2.1.

The data collection methods have been created and refined by the LABION research group, and the presented summary is partially extracted from [2].

4.2.3.2 Results After the computational experiments performed through the described pipeline, we selected the final CNN architecture and fine-tuned its hyperparameters through SMBO, empirically noting a performance increase with respect to the trial-and-error-based models: in an analogous way with respect to the other cases, the resulting model consists of three 1-D convolutional layers for the feature extraction and three fully connected layers for the classification, with the final configuration represented in Fig. 4.11.

For the architectural part of the optimization process, the results are equivalent to those obtained in the PD case, and detailed in Tabs. B.2 in Appendix B. For the fine-tuning optimization task, the obtained hyperparameters are reported in Tab. 4.3, whereas visualization of the obtained results are reported in Figs. 4.12,4.13,4.14.

While the ML baseline models are trained on normalized data, the DL architecture has been evaluated both on preprocessed and on raw data, given the challenging learning problem, characterized by a low-volume high-dimensional dataset: we obtain that, in good agreement with Liu et al. [33], CNNs can reach competitive performances on raw data, while avoiding several time-consuming preprocessing steps.

The CNN architecture has been trained on the 3-class classification problem to output, through a soft-max, a probability distribution associated with the class choices, where the highest probability is taken as a prediction of the corresponding label. Performances are evaluated via a Leave-One-Patient-Out Cross Validation method, and scoring breakdowns are reported in the confusion matrix in Fig. 4.18.

For the CNN model, the average spectral-level accuracy is 0.83 on raw data (the same score 0.83 on normalized data). Complete Metrics, including Sensitivity and Specificity, for the spectra-level evaluation are reported in Tab. 4.4. In comparison, our baseline methods, the more common ML techniques of SVM, RF and XGB achieve accuracy of 0.73, 0.72, and 0.81, respectively.

Considering classification metrics at a patient-level, the average accuracy score of the CNN model is 0.89 on raw data (0.84 on normalized data). In comparison, SVM, RF, and XGB have, respectively, accuracy rates of 0.76, 0.79 and 0.86. Complete Metrics for the patient-level classification are reported in Tab. 4.4 (Patient-level). COV patients are, at a first glance, more prone to be misclassified as COV-. On the other hand, CTRL and COV- individuals are usually correctly classified. We noted that, among the misclassified patients, four were affected by severe comorbidities (but still matching the inclusion criteria). By excluding these patients, the DL model accuracy increased to 0.92 (Tab. 4.4, patient-level COV+=26).

Table 4.3: Summary of the optimized parameters for the COVID dataset for the final neural network hyper-parameters optimization phase with their relative search space and the final values obtained: int. refers to integer search space while logunif. and discrete unif. respectively to log-uniform and discrete uniform search space.

Parameter	Search Space	Final value
CNN1 filters	int. [l = 5, h = 150]	100
CNN1 kernel size	int. [l = 5, h = 100]	100
CNN1 strides	int. [l = 1, h = 5]	1
CNN2 filters	int. [l = 5, h = 120]	100
CNN2 kernel size	int. [l = 5, h = 30]	5
CNN2 strides	int. [l = 1, h = 5]	2
MaxPool2 size	int. [l = 2, h = 10]	6
MaxPool2 strides	int. [l = 2, h = 6]	3
CNN3 filters	int. [l = 5, h = 150]	25
CNN3 kernel size	int. [l = 5, h = 20]	9
CNN3 strides	int. [l = 1, h = 5]	5
MaxPool3 size	int. [l = 1, h = 10]	3
MaxPool3 strides	int. [l = 1, h = 4]	2
Dropout rate flat	discrete unif. [l = 0.1, h = 0.95, q = 0.05]	0.1
Dense units 1	int. [l = 32, h = 1024]	732
Dropout rate	discrete unif. [l = 0.1, h = 0.95, q = 0.05]	0.7
Dense units 2	int. [l = 32, h = 1024]	189
Dense units 3	int. [l = 32, h = 1024]	152
Learning rate	logunif. [l = 1e-6, h = 1e-1]	0.0002

More insights in the classification results are discussed in Sec. 5.3. To gain further insight into the learning problem, and to easily enable the comparison between our method and other existing tests able to discriminate COV patients, we investigate binary classifications cases, in particular the COV vs CTRL and CTRL vs COVNEG cases, training specific models only on the data belonging to the mentioned classes (pair-wise). The performance breakdowns for the binary classification problems are reported in Tab. 4.4. It is worth noticing that the binary classification problems are further penalized by the reduced total amount of data (with respect to the ternary classification). However, the resulting performaces are comparable, when not even higher, with respect to the ternary case.

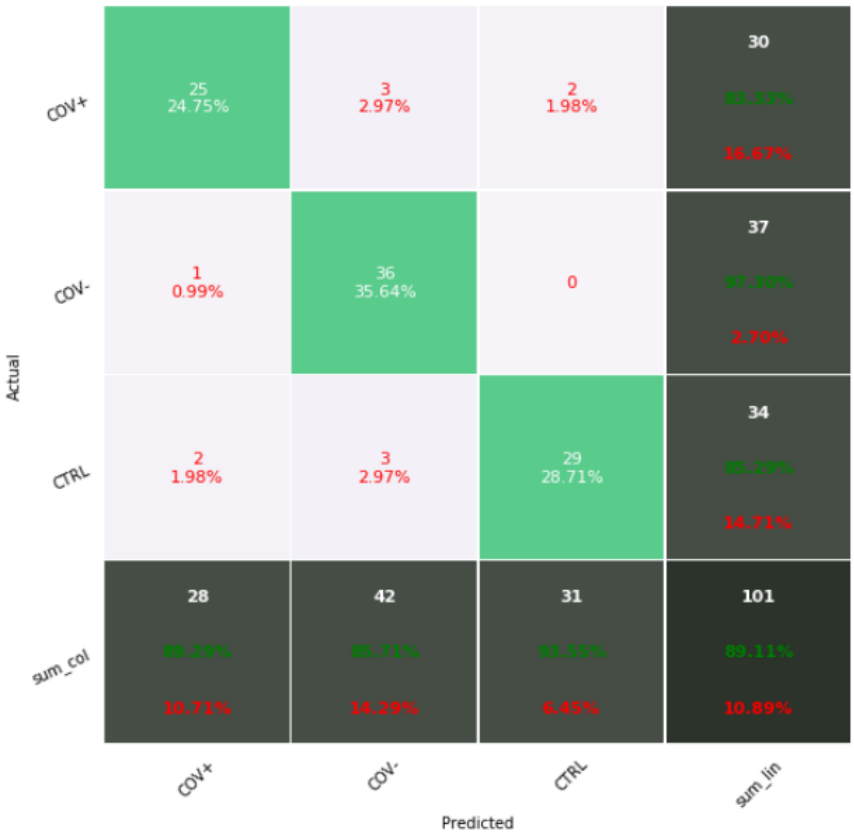

Figure 4.18: Confusion Matrix obtained from the classification of 101 patients of the COVID classification task through the proposed CNN model trained on raw data via Leave-One-(Patient)-Out Cross-Validation.

Table 4.4: Main results for the COVID dataset. **Ternary)** CNN accuracy, sensitivity and specificity at the patient level of the ternary (COV+ vs COV- vs CTRL) classification model using leave-one patient out cross-validation, with the related COV+ sensitivity and specificity percentages. **Binary)** CNNs accuracy, sensitivity and specificity at the patient level for the binary (COV+ vs CTRL, COV+ vs COV- and COV- vs CTRL) classification model.

	Accuracy	Sensitivity	Specificity	COV+ Sensitivity	COV+ Specificity
Ternary					
Patient-level	89%	89%	95%	83.3%	96%
Patient-level (COV+ = 26)	92%	91%	96%	85%	96%
Binary					
COV+ vs CTRL	89%	83%	94%	-	-
COV+ vs COV-	89.6%	83%	93%	-	-
COV- vs CTRL	91.5%	89%	94%	-	-

5 Discussions and Conclusions

5.1 ALS

A comparison between a machine learning model (SVM), a fully connected neural network model and a convolutional model, reported in Appendix A, leads to the selection of the CNN approach as the most suitable for the objective task. The resulting model undergoes a process of hyper-parameter optimization composed of various stages, during which different kinds of parameters are optimized through TPE and Gaussian Process algorithms. The optimized convolutional model, whose optimization details are described in Tab. A.3 and visualized in Fig. A.4, has a structure analogous to the one developed in [33], indicating that the model type seems to be suitable for the treatment of such a particular kind of data (spectral data).

Performance measures for the optimized system reported in Tab. 4.1 show the constant improvement obtained along with our pipeline, demonstrating the validity of our methodology for developing a system that suitably deals with spectral data.

The data augmentation procedure seems to help the model in the classification task, having optimized the parameters of the augmentation in order not to introduce major distortions of the (limited in amount) original data: this approach not only improves the final accuracy but also reduces the standard deviation associated with the cross-validation stage.

The Transfer Learning approach gives its contribution by slightly ameliorating the final model, mainly by giving a better initialization of the convolutional features: the main effect, in addition to a small increase in the accuracy, is to diminish the standard deviation, as a symptom of more stable training phases

along with the different cross-validation folds.

In summary, the overall performances indicate the significant ability of CNN to distinguish between target (ALS-Positive or Controls) spectra. However, it is worth to notice that a larger data collection is needed to effectively assess stable and reliable performances.

The major weakness of the available data is the impossibility[8] of maintaining a net separation of different patient spectra during the training (due to the restricted number of patients) while having multiple spectra extracted for each patient (but from different samples and positions in the sample). Having more data would allow to properly simulate a screening process, where a fixed number of patients is retained completely separated and independently used as a test set, verifying the reliability of the approach.

The current evaluation performed on the available data allows determining the suitability of the exploited model for this particular task, and for the specific data type. As mentioned in Sec. 4.1.5 and Sec. 3.2.5.1, the CNN model is endowed with some important properties that well conciliate with spectral data.

The optimization pipeline allows both to extract a suitable range of parameters to perform meaningful distortion of the original data, in order to produce a consistent augmentation both in conceptual terms (the kind of exploited invariance in the augmentation) and under the perspective of the system physics (the intensity of the perturbations should be coherent with the experimental spectroscope error measure and with the physical changes in concentrations of the various compounds in the samples).

The Transfer Learning attempt is an interesting experiment, in a field of applications where the available amount of data is terribly limited (as it happens to be our case). Although it does not lead to a net increase in performance in the current case, it's worth to notice that the amount of data included in the transfer learning procedure might be greatly increased as well, with much more ease with respect to the ALS-related data, possibly resulting in a much more incisive behaviour.

5.2 PD

As we mentioned in the introductory section, specific biomarkers associated to several pathologies open the doors to the possibility to assess the disease onset, to

[8] Attempts were made against this possibility, resulting in a too-large dependency of the obtained results along with the particular choice of patients selected as the test set. An extrapolation of those attempts is visible through the training curves collected at `https://app.wandb.ai/dbertazioli/raw_saliva_patient_id_deeplearning_cv_various_K`.

shorten the time for the differential diagnosis, to evaluate the disease progression and to perform a continuous monitoring of both therapies and rehabilitation efficiency.

In this work, we exploited a RS-based analysis for the spectroscopic characterization of saliva collected from PD patients. Such a procedure allowed the collection of a highly informative signal from saliva using a fast, cheap, and minimally invasive procedure.

Our DL-based approach was refined along with the pipeline described in Sec. 4 and performed in order to automatize the signal pre-processing and to create a classification model able to discriminate subjects considering the entire Raman spectral set associated to the subject. This approach can be seen as a preliminary contribution towards the creation of a fast and sensitive Raman based diagnostic and monitoring tool close to the clinical application.

In particular, as described in Sec. 4.2.2, the most performing method identified is a deep convolutional model, which showed to have the capability to reliably deal with spectral data. This can be explained by the fact that CNN models have properties (e.g. local connectivity and a weak translational invariance induced by the multiple pooling operations) well suited for dealing with spectral data, where a vertical translational invariance (invariance to small changes in intensity, due to the error of the measurement process) and horizontal invariance (invariance to small shift along the x-axis, due to the laser calibration error during the spectral acquisition) play an important role.

These properties, along with the Representation Learning capability, allow a CNN model also to better treat raw spectral data, when compared to the ML baseline. This is an important result because it means that the manual preprocessing of the data is not required with this model, saving a consistent amount of time and avoiding the introduction of possible errors and bias during the manual data manipulation. Our innovative ML/DL approach at the Patient-level reached an accuracy of 88.1%, which makes it competitive in the diagnostic field and even surpasses previously proposed assays for PD.

As it can be observed from Fig. 4.16, the ML/DL pipeline resulted in the proper classification of almost all of the considered CTRL and AD subjects, whereas PD subjects were observed to be more prone to misclassification. A possible explanation for the erroneously PD attributions in CTRL group can be found when looking at the clinical data related to the patients. In fact, two of the PD subjects revealed an initial PD stage: probably, due to this reason, the classification model wrongly assigned the major part of the Raman spectra to the CTRL group. On the other hand, a PD patient wrongly assigned to the AD group was diagnosed with cerebellum atrophy in the neurodegeneration process,

which occurs principally in AD with sporadic cases also in PD. In general, we advance the hypothesis that in some cases, the erroneous attributions could be due to external factors, such as pathological progression or comorbidities, rather than pure mistakes of the model.

These findings suggest that the created model must be further trained with data belonging to new experimental groups and sub-groups related to the same pathology (e.g. different pathological stages and comorbidities) to further refine the classification mechanism. This is one of the greatest advantages of this approach: it is always possible to easily train/retrain the model with new and different data related to the target pathologies, leading to a refinement in the discriminatory power.

In conclusion, our study demonstrated the potential application of the Raman analysis for the simultaneous identification of a large range of molecules present in saliva. We are strongly convinced that this method could be easily transferred to the clinical setting, thus, the Raman analysis of saliva could provide clinicians and researchers with a powerful instrument for the PD managing.

5.3 COVID-19

In the COVID-related task, a Raman-based approach was presented for the discrimination of current or past infection by SARS-CoV-2 from a simply collectable biofluid such as saliva. As we already mentioned, the sample processing and analytical procedure, described in Sec. 4.2.3, is simple, automatable, highly informative and fast: once properly deployed, the whole pipeline could be used to perform analysis obtaining results in a few minutes.

The discrimination ability of the RS relies on the wide range of information that can be obtained from a single analysis: the information regarding the analysed biofluid gives insights from all the biochemical species present on the base of their concentration, environment, chemical nature, modifications, mutations, alterations and interactions. Furthermore, it is possible to identify and track slight changes in the expression pattern of these molecules, typically resulting in a deformation of the normal trend in curve shape, which represents the basis for a classification model. With the constant increase of confirmed and severe cases of COVID-19, easy identification of infected subjects through a fast and minimally invasive procedure such as RS on saliva would be ideal. In addition to that, the identification of subjects with a past infection (with or without symptoms) could play a fundamental in epidemiological studies relative to the infection confinement.

In practice, we trained a series of classification models intended for the

definition of the subject condition. A baseline of Ml models was compared against DL techniques, with the selection of CNNs as the most-performing algorithm.

This result is in good agreement with what previously found for the other tasks, and the explanation is the same provided in Sec. 5.2. The CNN model has undergone a HPO phase, whose results are reported in Sec. 4.2.3.

The resulting network shows a high accuracy rate in the attribution of the data in the exact experimental group, with variations in the discrimination of single categories.

CTRL and COV- were attributed with higher discrimination power respect to the COV+ that present slightly lower percentages of accuracy, sensitivity and specificity. This difference is partly due to the attribution of misclassified patients to the COV- class, indicating a close relationship between the two groups.

The slight increase in the discrimination power observed removing from the training the four patients with relevant comorbidities is encouraging from a scientific perspective, because it underlines the sensitivity of the Raman fingerprint also in complex clinical pictures.

Obviously, an increase in subject numbers and the introduction of new experimental classes is expected to improve the discrimination power of the model and its generalization capability, learning to attribute less weight to potential comorbidities when dealing with a specific classification task. Probably, also the lower number of COV+ patients plays a crucial role in this attribution and, for this reason, the recruitment of a higher number of COVID-19 infected subjects is in progress.

In conclusion, our results demonstrated the existence of a characteristic salivary COVID-19 Raman fingerprint, which is used to train and test a classification model able to discriminate between COV+, CTRL and COV- patients.

Exploiting portable Raman combined with the implementation of the proposed classification model with more COVID-19 cases could lead to the creation of a point of care able to reliably assess the positivity to the SARS-CoV-2 infection and to allow longitudinal monitoring by predicting the severity of the disease in terms of respiratory symptoms and manifestations.

It is worth to notice that, once assessed the discrimination capability of our models, the performances are expected to drastically increase together with the input of new Raman and clinical data of more patients. In fact, introducing more data about the Raman signal collected from saliva of different patients belonging to specific experimental groups (e.g. longitudinal clinical course, comorbidities or asymptomatic), the classification model will be potentially able to discriminate more accurately the infection onset and also to predict the severity stage of the

respiratory tract.

The optimal scenario includes different Raman workstations connected through an online learning network, able to analyse and to furnish a huge amount of information for the system training.

6 Future Work

An incredible amount of future work can be established from the current results:

1. First of all, When the patient recruitment will provide us with a higher number of patients involved in the study, we will be able to further investigate the generalization capability of the proposed methods.

2. Furthermore, a series of technical experiments could be run, investigating new architectures (i.e. ensemble models, or more complex DL models such as Resnets). An interesting experiment in this field would be to hybridize the CNN model with a self-attention-based approach, to learn both local (via the CNN) and non-local (via the attention mechanism) features from the spectral data.

3. From a data scarcity perspective, the data augmentation procedure could be further ameliorated: first of all, an extensive study on data generation techniques is required (to the best of our knowledge, there are no such studies in the related literature), involving regular Autoencoders and more complex VAE, CVAE, CCVAE models. GANs could be also exploited for the augmentation task, potentially achieving next-level performances.

4. The GAN strategy can be also thought in combination with the transfer learning approach, possibly refined with a (at least 1000x) larger amount of training data. In general, transfer learning can have the potential to overcome some limitations due to data scarcity.

5. The whole procedure should be performed and tested on data acquired with portable RS, typically having lower resolutions. For this task, if the resolution is too low to allow a reliable detection and diagnosis, a super-resolution approach could be designed, to maximally exploit the high-resolution data already acquired in those years.

6. Another important future step is the interpretation of the model predictions: this task would be powerful when dealing with unknown/unattributed spectral bands, where insights on the spectral bands on which the model focuses on could results in a more precise and scientific attribution and interpretation of the whole method.

References

[1] Carlomagno, Picciolini, Gualerzi, Meloni, Banfi, Aux, Bertazioli, Andrico, Messina, and Bedoni. Characterization of a salivary parkinson's disease fingerprint through raman spectroscopy and machine learning. *Submitted to "Progress in Neurobiology"*.

[2] Carlomagno, Gualerzi, Picciolini, Banfi, Lax, Bertazioli, Messina, Navarro, Clerici, Arienti, Bianco, Caronni, and Bedoni. The raman covid-19 salivary fingerprint: a fast and sensitive approach for the characterization of covid-19 onset from saliva. *Submitted to "Nature Communications"*.

[3] Philip Bunker, William Harter, and Eric Heller. Molecular rotation spectra. *Physics Today*, 38:13, 01 1985.

[4] Philip Bunker and Per Jensen. *Molecular Symmetry and Spectroscopy*, volume 32. 01 1998.

[5] Wikipedia. `https://en.wikipedia.org/wiki`.

[6] N. Manini. *Introduction to the Physics of Matter: Basic atomic, molecular, and solid-state physics*. Undergraduate Lecture Notes in Physics. Springer International Publishing, 2015.

[7] Félix Lussier, Vincent Thibault, Benjamin Charron, Gregory Q. Wallace, and Jean-Francois Masson. Deep learning and artificial intelligence methods for Raman and surface-enhanced Raman scattering. *TrAC Trends in Analytical Chemistry*, 124:115796, mar 2020.

[8] David I Ellis, David Broadhurst, Sarah J Clarke, and Royston Goodacre. Rapid identification of closely related muscle foods by vibrational spectroscopy and machine learning. *Analyst*, 130(12):1648–1654, 2005.

[9] Reyhan Selin Uysal, Ismail Hakki Boyaci, Hüseyin Efe Genis, and Ugur Tamer. Determination of butter adulteration with margarine using Raman spectroscopy. *Food Chemistry*, 141(4):4397–4403, 2013.

[10] H Mohamadi Monavar, N K Afseth, J Lozano, R Alimardani, M Omid, and J P Wold. Determining quality of caviar from Caspian Sea based on Raman spectroscopy and using artificial neural networks. *Talanta*, 111:98–104, 2013.

[11] Arslan Amjad, Rahat Ullah, Saranjam Khan, Muhammad Bilal, and Asifullah Khan. Raman spectroscopy based analysis of milk using random forest classification. *Vibrational Spectroscopy*, 99:124–129, 2018.

[12] William Cheung, Iqbal T Shadi, Yun Xu, and Royston Goodacre. Quantitative Analysis of the Banned Food Dye Sudan-1 Using Surface Enhanced Raman Scattering with Multivariate Chemometrics. *The Journal of Physical Chemistry C*, 114(16):7285–7290, 2010.

[13] Yan-xiong Wu, Pei Liang, Qian-min Dong, Yang Bai, Zhi Yu, Jie Huang, Yuan Zhong, Yu-Chan Dai, Dejiang Ni, Hai-bo Shu, and Charles U Pittman. Design of a silver nanoparticle for sensitive surface enhanced Raman spectroscopy detection of carmine dye. *Food Chemistry*, 237:974–980, 2017.

[14] Yu-jie Ai, Pei Liang, Yan-xiong Wu, Qian-min Dong, Jing-bin Li, Yang Bai, Bi-Jie Xu, Zhi Yu, and Dejiang Ni. Rapid qualitative and quantitative determination of food colorants by both Raman spectra and Surface-enhanced Raman Scattering (SERS). *Food Chemistry*, 241:427–433, 2018.

[15] Shizhuang Weng, Shuan Yu, Ronglu Dong, Jinling Zhao, and Dong Liang. molecules Detection of Pirimiphos-Methyl in Wheat Using Surface-Enhanced Raman Spectroscopy and Chemometric Methods.

[16] Yves Roggo, Klara Degardin, and Pierre Margot. Identification of pharmaceutical tablets by Raman spectroscopy and chemometrics. *Talanta*, 81(3):988–995, 2010.

[17] Hannah Dies, Joshua Raveendran, Carlos Escobedo, and Aristides Docoslis. Rapid identification and quantification of illicit drugs on nanodendritic surface-enhanced Raman scattering substrates. *Sensors and Actuators B: Chemical*, 257:382–388, 2018.

[18] Reza Salemmilani, Brian D Piorek, Rustin Y Mirsafavi, Augustus W Fountain, Martin Moskovits, and Carl D Meinhart. Dielectrophoretic Nanoparticle Aggregation for On-Demand Surface Enhanced Raman Spectroscopy Analysis. *Analytical Chemistry*, 90(13):7930–7936, 2018.

[19] Kundan Sivashanmugan, Kenneth Squire, Ailing Tan, Yong Zhao, Joseph Abraham Kraai, Gregory L Rorrer, and Alan X Wang. Trace Detection of Tetrahydrocannabinol in Body Fluid via Surface-Enhanced Raman Scattering and Principal Component Analysis. *ACS Sensors*, 4(4):1109–1117, 2019.

[20] Shelby R Khandasammy, Marisia A Fikiet, Ewelina Mistek, Yasmine Ahmed, Lenka Halámková, Justin Bueno, and Igor K Lednev. Bloodstains, paintings, and drugs: Raman spectroscopy applications in forensic science. *Forensic Chemistry*, 8:111–133, 2018.

[21] Kyle C Doty and Igor K Lednev. Differentiation of human blood from animal blood using Raman spectroscopy: A survey of forensically relevant species. *Forensic Science International*, 282:204–210, 2018.

[22] Aliaksandra Sikirzhytskaya, Vitali Sikirzhytski, and Igor K Lednev. Determining Gender by Raman Spectroscopy of a Bloodstain. *Analytical Chemistry*, 89(3):1486–1492, feb 2017.

[23] Daniel Wiktelius, Linnea Ahlinder, Andreas Larsson, Karin Höjer Holmgren, Rikard Norlin, and Per Ola Andersson. On the use of spectra from portable Raman and ATR-IR instruments in synthesis route attribution of a chemical warfare agent by multivariate modeling. *Talanta*, 186:622–627, 2018.

[24] Christoph Gasser, Michael Göschl, Johannes Ofner, and Bernhard Lendl. Stand-off Hyperspectral Raman Imaging and Random Decision Forest Classification: A Potent Duo for the Fast, Remote Identification of Explosives. *Analytical Chemistry*, 91(12):7712–7718, jun 2019.

[25] Sandra Kloß, Bernd Kampe, Svea Sachse, Petra Rösch, Eberhard Straube, Wolfgang Pfister, Michael Kiehntopf, and Jürgen Popp. Culture Independent Raman Spectroscopic Identification of Urinary Tract Infection Pathogens: A Proof of Principle Study. *Analytical Chemistry*, 85(20):9610–9616, oct 2013.

[26] Satya Kiran Koya, Sally Yurgelevic, Michelle Brusatori, Changhe Huang, Lawrence N Diebel, and Gregory W Auner. Rapid Detection of Clostridium difficile Toxins in Stool by Raman Spectroscopy. *Journal of Surgical Research*, 244:111–116, 2019.

[27] Jae-young Lim, Jung-soo Nam, Hyunku Shin, Jaena Park, Hye-in Song, Minsung Kang, Kwang-il Lim, and Yeonho Choi. Identification of Newly Emerging Influenza Viruses by Detecting the Virally Infected Cells Based on Surface Enhanced Raman Spectroscopy and Principal Component Analysis. *Analytical Chemistry*, 91(9):5677–5684, may 2019.

[28] Anabia Sohail, Saranjam Khan, Rahat Ullah, Shahzad Ahmad Qureshi, Muhammad Bilal, and Asifullah Khan. Analysis of hepatitis c infection

using raman spectroscopy and proximity based classification in the transformed domain. *Biomed. Opt. Express*, 9(5):2041–2055, May 2018.

[29] Edgar Guevara, Juan Carlos Torres-Galván, Miguel G. Ramírez-Elías, Claudia Luevano-Contreras, and Francisco Javier González. Use of raman spectroscopy to screen diabetes mellitus with machine learning tools. *Biomed. Opt. Express*, 9(10):4998–5010, Oct 2018.

[30] Wansun Kim, Soo Hyun Lee, Sang Hun Kim, Jae-Chul Lee, Sang Woong Moon, Jae Su Yu, and Samjin Choi. Highly Reproducible Au-Decorated ZnO Nanorod Array on a Graphite Sensor for Classification of Human Aqueous Humors. *ACS Applied Materials & Interfaces*, 9(7):5891–5899, feb 2017.

[31] Xiaozhou Li, Tianyue Yang, Siqi Li, Lili Jin, Deli Wang, Dagang Guan, and Jianhua Ding. Noninvasive liver diseases detection based on serum surface enhanced raman spectroscopy and statistical analysis. *Opt. Express*, 23(14):18361–18372, Jul 2015.

[32] Wooje Lee, Afroditi Nanou, Linda Rikkert, Frank A W Coumans, Cees Otto, Leon W M M Terstappen, and Herman L Offerhaus. Label-Free Prostate Cancer Detection by Characterization of Extracellular Vesicles Using Raman Spectroscopy. *Analytical Chemistry*, 90(19):11290–11296, oct 2018.

[33] Jinchao Liu, Margarita Osadchy, Lorna Ashton, Michael Foster, Christopher J. Solomon, and Stuart J. Gibson. Deep Convolutional Neural Networks for Raman Spectrum Recognition: A Unified Solution. *Analyst*, 142(21):4067–4074, aug 2017.

[34] Jinchao Liu, Stuart J Gibson, James Mills, and Margarita Osadchy. Dynamic Spectrum Matching with One-shot Learning. Technical report.

[35] Xiaqiong Fan, Wen Ming, Huitao Zeng, Zhimin Zhang, and Hongmei Lu. Deep learning-based component identification for the raman spectra of mixtures. *The Analyst*, 144, 03 2019.

[36] Chi Sing Ho, Neal Jean, Catherine A. Hogan, Lena Blackmon, Stefanie S. Jeffrey, Mark Holodniy, Niaz Banaei, Amr A.E. Saleh, Stefano Ermon, and Jennifer Dionne. Rapid identification of pathogenic bacteria using Raman spectroscopy and deep learning. *Nature Communications*, 10(1):1–8, dec 2019.

[37] Zhi-Qin J. Xu and Hanxu Zhou. Deep frequency principle towards understanding why deeper learning is faster. *ArXiv*, abs/2007.14313, 2020.

[38] Charles H Martin and Michael W Mahoney. Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning. Technical report, 2018.

[39] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences of the United States of America*, 116(32):15849–15854, 2019.

[40] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[41] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[42] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[43] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks, 2019.

[44] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxlcr, Min Lin, Fred A Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *36th International Conference on Machine Learning, ICML 2019*, volume 2019-June, pages 9230–9239, jun 2019.

[45] Peizhong Ju, Xiaojun Lin, and Jia Liu. Overfitting Can Be Harmless for Basis Pursuit: Only to a Degree. 2020.

[46] Ziheng Jiang, Chiyuan Zhang, Kunal Talwar, and Michael C Mozer. Exploring the Memorization-Generalization Continuum in Deep Learning.

[47] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.

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

[49] Janice Lan, Uber Ai, Rosanne Liu, Hattie Zhou Uber, and Jason Yosinski. LCA: Loss Change Allocation for Neural Network Training. Technical report.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[51] Jacopo Cavazza, Benjamin D Haeffele, Connor Lane, Pietro Morerio, Vittorio Murino, and René Vidal. Dropout as a Low-Rank Regularizer for Matrix Factorization. Technical report, 2018.

[52] Jacopo Cavazza, Pietro Morerio, Benjamin Haeffele, Connor Lane, Vittorio Murino, and Rene Vidal. Dropout as a low-rank regularizer for matrix factorization. volume 84 of *Proceedings of Machine Learning Research*, pages 435–444, Playa Blanca, Lanzarote, Canary Islands, 09–11 Apr 2018. PMLR.

[53] Poorya Mianjy, Raman Arora, and Rene Vidal. On the implicit bias of dropout. volume 80 of *Proceedings of Machine Learning Research*, pages 3540–3548, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[54] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.

[55] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

[56] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[57] M. Kearns and L. G. Valiant. Learning boolean formulae or finite automata is as hard as factoring. Technical Report TR 14-88, Harvard University Aiken Computation Laboratory, 1988.

[58] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, January 1994.

[59] C M BERNERS-LEE. Cybernetics and Forecasting. *Nature*, 219(5150):202–203, 1968.

[60] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Trans. Syst. Man Cybern.*, 1:364–378, 1971.

[61] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[62] Wei Di, Anurag Bhardwaj, and Jianing Wei. *Deep Learning Essentials: Your Hands-on Guide to the Fundamentals of Deep Learning and Neural Network Modeling*. Packt Publishing, 2018.

[63] Rina Dechter. Learning while searching in constraint-satisfaction-problems. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, page 178–183. AAAI Press, 1986.

[64] A. Zheng and A. Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly, 2018.

[65] A Olgac and Bekir Karlik. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence And Expert Systems*, 1:111–122, 02 2011.

[66] V. Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[67] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[68] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[69] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. volume 28 of *Proceedings of Machine Learning Research*, pages 1319–1327, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[70] Chigozie Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *ArXiv*, abs/1811.03378, 2018.

[71] Charles Dugas, Y. Bengio, François Bélisle, Claude Nadeau, and Rene Garcia. Incorporating second-order functional knowledge for better option pricing. pages 472–478, 01 2000.

[72] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Computing Research Repository - CORR*, 12, 03 2011.

[73] Claude Lemaréchal. Cauchy and the gradient method, 2012.

[74] H. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[75] Boris Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4:1–17, 12 1964.

[76] Y. Nesterov. A method for solving the convex programming problem with convergence rate o(1/k^2). 1983.

[77] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Y. Bengio. Difference target propagation. pages 498–515, 08 2015.

[78] Kurt Wan-Duo Ma, J. Lewis, and W. Kleijn. The hsic bottleneck: Deep learning without back-propagation. *ArXiv*, abs/1908.01580, 2020.

[79] Gregory Morse and Kenneth Stanley. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. pages 477–484, 07 2016.

[80] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, volume 2, pages 985–990 vol.2, 2004.

[81] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[82] Brian Guenter. Efficient symbolic differentiation for graphics applications. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, page 108–es, New York, NY, USA, 2007. Association for Computing Machinery.

[83] Adam Gaier and David Ha. Weight Agnostic Neural Networks. In H Wallach, H Larochelle, A Beygelzimer, F d\textquotesingle Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5364–5378. Curran Associates, Inc., 2019.

[84] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[85] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[86] Samuel Burer and Renato D. C. Monteiro. Local minima and convergence in low-rank semidefinite programming. *Math. Program.*, 103(3):427–444, July 2005.

[87] Ricardo Cabral, Fernando De la Torre, Joao Costeira, and Alexandre Bernardino. Unifying nuclear norm and bilinear factorization approaches for low-rank matrix decomposition. pages 2488–2495, 12 2013.

[88] Francis R. Bach, Julien Mairal, and Jean Ponce. Convex sparse matrix factorizations. *CoRR*, abs/0812.1869, 2008.

[89] Francis R. Bach. Convex relaxations of structured matrix factorizations. *CoRR*, abs/1309.3117, 2013.

[90] Benjamin Haeffele, Eric Young, and Rene Vidal. Structured low-rank matrix factorization: Optimality, algorithm, and applications to image processing. volume 32 of *Proceedings of Machine Learning Research*, pages 2007–2015, Bejing, China, 22–24 Jun 2014. PMLR.

[91] Benjamin D. Haeffele and René Vidal. Global optimality in tensor factorization, deep learning, and beyond. *CoRR*, abs/1506.07540, 2015.

[92] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[93] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Comput.*, 7(1):108–116, January 1995.

[94] Kam Jim, Bill G. Horne, and C. Lee Giles. Effects of noise on convergence and generalization in recurrent networks. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, page 649–656, Cambridge, MA, USA, 1994. MIT Press.

[95] kdnuggets. `https://www.kdnuggets.com/`.

[96] Towardsdatascience. `https://towardsdatascience.com`.

[97] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*, ICANN'10, page 92–101, Berlin, Heidelberg, 2010. Springer-Verlag.

[98] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[99] Patrice Simard, David Steinkraus, and John Platt. Best practices for convolutional neural networks applied to visual document analysis. pages 958–962, 01 2003.

[100] Yann Lecun, Koray Kavukcuoglu, and Clement Farabet. Convolutional networks and applications in vision. pages 253–256, 05 2010.

[101] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[102] D. Hubel and T. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[103] Alex Graves, Marcus Liwicki, Santiago Fernandez, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition, 2008.

[104] H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, 2014.

[105] Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. *CoRR*, abs/1410.4281, 2014.

[106] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[107] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

[108] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Comput.*, 12(8):1889–1900, August 2000.

[109] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.

[110] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[111] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag.

[112] X. Wang, P. Tino, M. A. Fardal, S. Raychaudhury, and A. Babul. Fast parzen window density estimator. In *2009 International Joint Conference on Neural Networks*, pages 3267–3274, 2009.

[113] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010.

[114] Rekha Gautam, Sandeep Vanga, Freek Ariese, and Siva Umapathy. Review of multidimensional data processing approaches for Raman and infrared spectroscopy. *EPJ Techniques and Instrumentation*, 2(1):8, 2015.

[115] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[116] Darren A Whitaker and Kevin Hayes. A simple algorithm for despiking Raman spectra. *Chemometrics and Intelligent Laboratory Systems*, 179:82–84, 2018.

[117] Richard Bellman. Dynamic Programming. *Science*, 153(3731):34–37, 1966.

[118] Eamonn Keogh and Abdullah Mueen. *Curse of Dimensionality*, pages 314–315. Springer US, Boston, MA, 2017.

[119] Esben Jannik Bjerrum, Mads Glahder, and Thomas Skov. Data augmentation of spectral data for convolutional neural network (CNN) based deep chemometrics. *CoRR*, abs/1710.01927, 2017.

[120] amunategui.github.io/cvae-in-finance. `amunategui.github.io/cvae-in-finance`.

[121] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, November 2011.

[122] Chollet, f. (2015) keras, github. `https://github.com/fchollet/keras`.

[123] F. Archetti and A. Candelieri. *Bayesian Optimization and Data Science*. SpringerBriefs in Optimization. Springer International Publishing, 2019.

[124] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery.

[125] Scikit-optimize. `https://zenodo.org/record/1157320`.

[126] C Carlomagno, P I Banfi, A Gualerzi, S Picciolini, E Volpato, M Meloni, A Lax, E Colombo, N Ticozzi, F Verde, V Silani, and M Bedoni. Human salivary Raman fingerprint as biomarker for the diagnosis of Amyotrophic Lateral Sclerosis. *Scientific Reports*, 10(1):10175, 2020.

A Appendix A: Details for the ALS Task

Model Comparison :

The detailed parameters for each model considered during the model selection are listed below (the SVM is a SKlearn implementation, whereas the FCNN and the CNN model are implemented in Keras):

1. SVM:
   (a) params: C: 5, gamma: 'auto', kernel: rbf
   (b) valid: 0.853 (±0.11)
   (c) test: 0.845 (±0.116)
2. FCNN:
   (a) params: 3 layer FCNN 256/128/64,LeakyRelu activations, BatchNorm+Dropout, EarlyStopping, Adam Optimizer with default learning rate (lr=0.001), batch size: 64
   (b) valid: 0.88 (±0.21)
   (c) test: 0.85 (±0.04)
3. CNN:
   (a) params: 3 CNN layers,Relu/LeakyRelu activations, 16/32/64 filters (respectively for each layers), kernel size: 32 , kernel strides: 1 , max pooling, pool size: 2, pool strides:2, 2 dense layers (256/128 neurons respectively) dropout (rate= 0.4, 0.3 respectively for each layer), batch size: 32, ES + LR scheduler
   (b) valid: 0.89 (±0.05)
   (c) test: 0.86 (±0.06)

Optimization The optimization process results are detailed in Tabs. A.1,A.2,A.3, whereas visualization of the obtained results are reported in Figs. A.1,A.2,A.3.

The final model obtained through the HPO process is represented in Fig. A.4.

B Appendix B: Details for the PD/AD Task.

Model Comparison The comparison of ML and DL models is reported in Tab. B.1.

Table A.1: Summary of the optimized parameters for the architecture search phase in the optimization pipeline with their relative search space and final value obtained for the ALS problem: where categorical and Boolean values the choice is about to include or not the corresponding layer, where discrete unif. the values are sampled into the interval determined by the low (l) and high (h) limits with a certain step (q), where logunif. log-uniform sample strategy is applied into the specified interval.

Parameter	Search Space	Final value
MaxPool1	categorical [True, False]	True
BatchNorm1	categorical [True, False]	False
CNN2	categorical [True, False]	True
MaxPool2	categorical [True, False]	True
BatchNorm2	categorical [True, False]	True
CNN3	categorical [True, False]	True
MaxPool3	categorical [True, False]	True
BatchNorm3	categorical [True, False]	True
Dropout_rate_flat	discrete unif. [l = 0, h = 1, q = 0.1]	0.3
Dense2	categorical [True, False]	False
Dense3	categorical [True, False]	False
Optimizer	categorical [Adam, Nadam, RMSprop]	Adam
learning rate	logunif. [l = 1e-6, h = 1e-1]	0.00037

Table A.2: Summary of the optimized parameters for the data augmentation optimization phase with their relative search space and final value obtained for the ALS task: logunif. refers to log-uniform sample strategy, the search interval is available in square brackets with low (l) and high (h) limits.

Parameter	Search Space	Final value
Beta shift	logunif. [l = 1e-4, h = 1e-1]	0.099
Slope shift	logunif. [l = 1e-3, h = 1e-1]	0.001
Multi shift	logunif. [l = 1e-3, h = 1e-1]	0.001

Optimization The optimization process results are detailed in Tabs. B.2, B.3, whereas visualization of the obtained results are reported in Figs. B.1,B.2,B.3.

The final model obtained through the HPO process is represented in Fig. B.4.

Table A.3: Summary of the optimized parameters for the final neural network hyper-parameters optimization phase with their relative search space and the final values obtained for the ALS task: int. refers to integer search space while logunif. and discrete unif. respectively to log-uniform and discrete uniform search space. It is worth to notice how the "Learning rate" and the "Dropout rate flat" hyper-parameters are re-optimized compared to Tab. A.1: in fact in this case the data augmentation procedure is integrated.

Parameter	Search Space	Final value
CNN1 filters	int. [l = 5, h = 100]	39
CNN1 kernel size	int. [l = 5, h = 60]	48
CNN1 strides	int. [l = 1, h = 5]	1
MaxPool1 size	int. [l = 2, h = 10]	4
MaxPool1 strides	int. [l = 2, h = 10]	9
CNN2 filters	int. [l = 5, h = 100]	27
CNN2 kernel size	int. [l = 5, h = 60]	60
CNN2 strides	int. [l = 1, h = 5]	3
MaxPool2 size	int. [l = 2, h = 10]	10
MaxPool2 strides	int. [l = 2, h = 10]	2
CNN3 filters	int. [l = 5, h = 100]	10
CNN3 kernel size	int. [l = 5, h = 60]	5
CNN3 strides	int. [l = 1, h = 5]	1
MaxPool3 size	int. [l = 2, h = 10]	7
MaxPool3 strides	int. [l = 2, h = 10]	2
Dropout rate flat	discrete unif. [l = 0, h = 1, q = 0.1]	0.0
Dense units	int. [l = 32, h = 1024]	868
Dropout rate	discrete unif. [l = 0, h = 1, q = 0.1]	0.2
Learning rate	logunif. [l = 1e-6, h = 1e-1]	0.0008

C Appendic C: Details for the COVID-19 Task

Model Comparison The comparison for the ML baseline models is reported in Tab. C.1, with a focus on both raw and preprocessed data, and a comparison of normalization techniques.

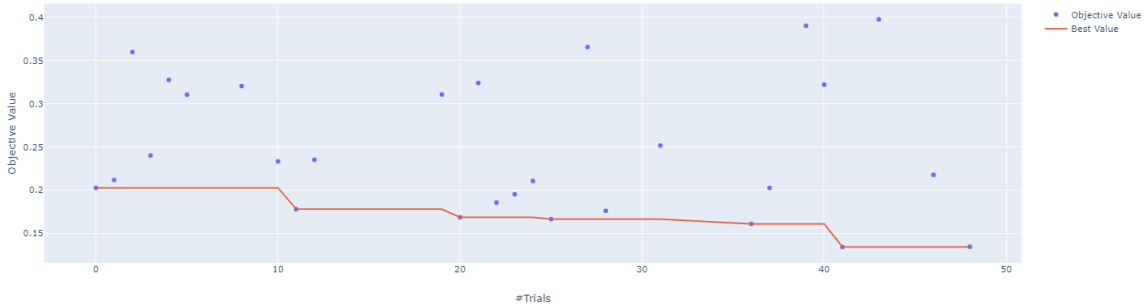

Figure A.1: Optimization history on the ALS dataset: each point represents the output from an optimization epoch, the red line represents the best seen path.

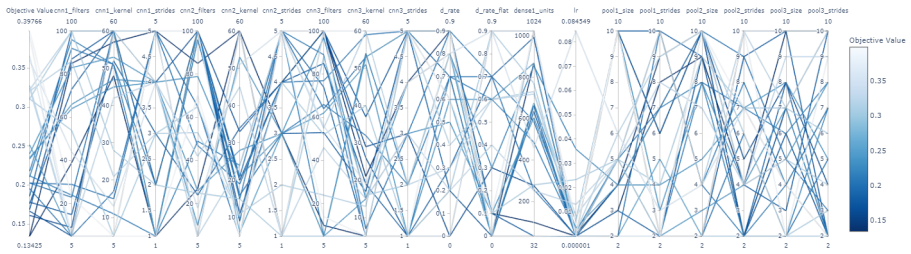

Figure A.2: Hyper-parameters search space exploration during the CNN optimization task on the ALS dataset: each line represents the values used for each hyper-parameter and the objective value reached from those values. From the visualization it is possible to observe how the optimization mechanism has improved exploration during the epochs.

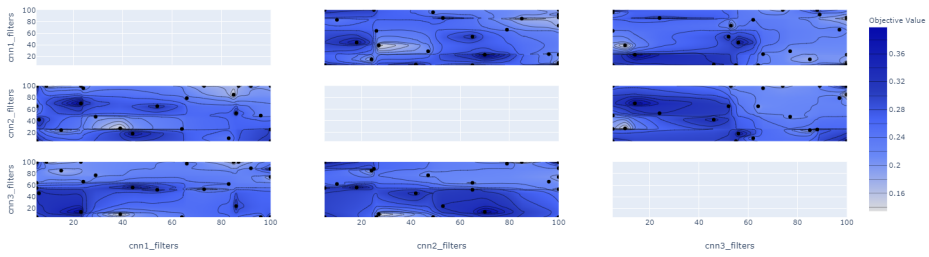

Figure A.3: Contour plot for the optimization over the ALS dataset: at the end of the optimization phase, this visualization shows a contour that gives an intuitive spatial view of the search space.

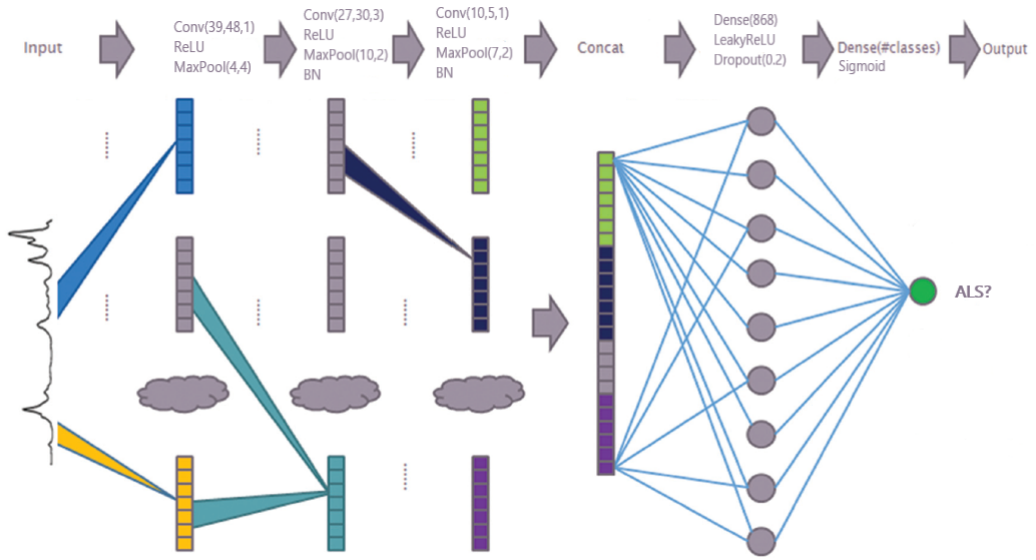

Figure A.4: Diagram of the neural networks proposed for the ALS task, inspired by and adapted from [33]. The shown hyper-parameters are those found at the end of the optimization procedure.

Table B.1: Extensive result comparison, showing performances of ML models and DL ones at patient-level on preprocessed data (l2 normalization) with and without Data Augmentation (DA) for the PD dataset. The last row refers to the final CNN model applied on raw unprocessed data, still gaining competitive performances.

Model	Accuracy	Precision	Recall	F-Measure
SVM	0.836	0.817 (±0.11)	0.812 (±0.12)	0.803 (±0.07)
SVM + DA	0.851	0.825 (±0.12)	0.830 (±0.08)	0.821 (±0.08)
RF	0.821	0.844 (±0.05)	0.759 (±0.15)	0.787 (±0.06)
RF + DA	0.836	0.824 (±0.06)	0.773 (±0.15)	0.791 (±0.08)
FCNN	0.836	0.813 (±0.06)	0.816 (±0.09)	0.811 (±0.06)
FCNN + DA	0.851	0.834 (± 0.04)	0.81 (±0.08)	0.82 (±0.04)
CNN	0.851	0.824 (±0.09)	0.849 (±0.1)	0.828 (±0.06)
CNN Opt + DA	0.881	0.877 (±0.05)	0.873 (±0.09)	0.87 (±0.03)
CNN Opt + DA (RAW)	0.866	0.832 (±0.09)	0.87 (±0.01)	0.85 (±0.05)

Table B.2: Summary of the optimized parameters for the PD dataset for the architecture search phase in the optimization pipeline with their relative search space and final value obtained for the ALS problem: where categorical and Boolean values the choice is about to include or not the corresponding layer, where discrete unif. the values are sampled into the interval determined by the low (l) and high (h) limits with a certain step (q), where logunif. log-uniform sample strategy is applied into the specified interval.

Parameter	Search Space	Final value
MaxPool1	categorical [True, False]	False
BatchNorm1	categorical [True, False]	True
CNN2	categorical [True, False]	True
MaxPool2	categorical [True, False]	True
BatchNorm2	categorical [True, False]	True
CNN3	categorical [True, False]	True
MaxPool3	categorical [True, False]	True
BatchNorm3	categorical [True, False]	False
Dense2	categorical [True, False]	True
Dense3	categorical [True, False]	True
Optimizer	categorical [Adam, Nadam, RMSprop]	Adam

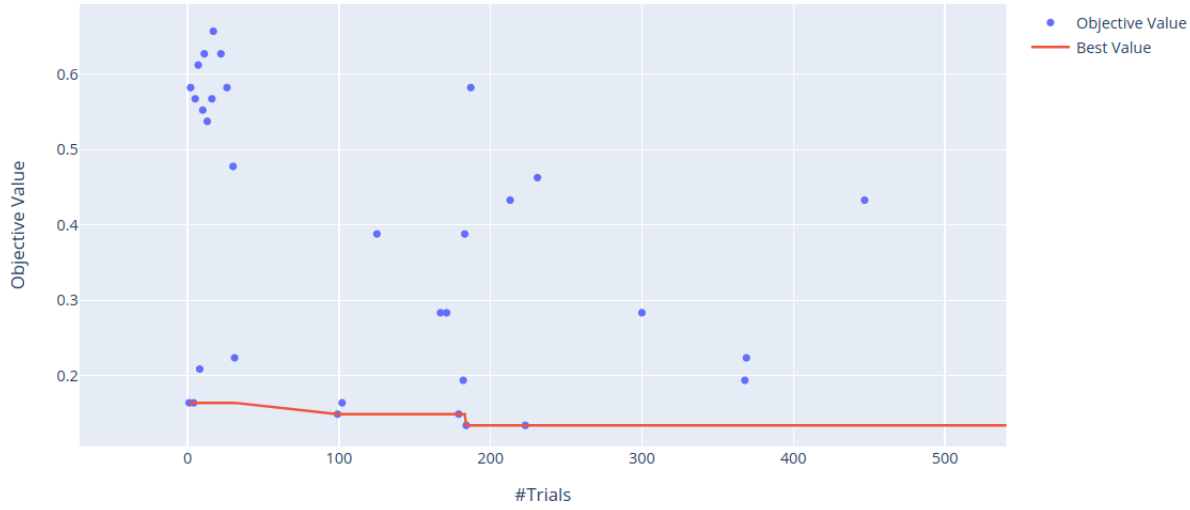

Figure B.1: Optimization history on the PD dataset: each point represents the output from an optimization epoch, the red line represents the best seen path.

Table B.3: Summary of the optimized parameters for the PD dataset for the final neural network hyper-parameters optimization phase with their relative search space and the final values obtained: int. refers to integer search space while logunif. and discrete unif. respectively to log-uniform and discrete uniform search space.

Parameter	Search Space	Final value
CNN1 filters	int. [l = 5, h = 150]	60
CNN1 kernel size	int. [l = 5, h = 100]	60
CNN1 strides	int. [l = 1, h = 5]	1
MaxPool1 size	int. [l = 2, h = 10]	5
MaxPool1 strides	int. [l = 2, h = 10]	5
CNN2 filters	int. [l = 5, h = 120]	70
CNN2 kernel size	int. [l = 5, h = 30]	11
CNN2 strides	int. [l = 1, h = 5]	4
MaxPool2 size	int. [l = 2, h = 10]	4
MaxPool2 strides	int. [l = 2, h = 6]	4
CNN3 filters	int. [l = 5, h = 150]	93
CNN3 kernel size	int. [l = 5, h = 20]	45
CNN3 strides	int. [l = 1, h = 5]	1
MaxPool3 size	int. [l = 1, h = 10]	2
MaxPool3 strides	int. [l = 1, h = 4]	2
Dropout rate flat	discrete unif. [l = 0.1, h = 0.95, q = 0.05]	0.1
Dense units 1	int. [l = 32, h = 1024]	360
Dropout rate	discrete unif. [l = 0.1, h = 0.95, q = 0.05]	0.5
Dense units 2	int. [l = 32, h = 1024]	224
Dense units 3	int. [l = 32, h = 1024]	122
Learning rate	logunif. [l = 1e-6, h = 1e-1]	0.0002

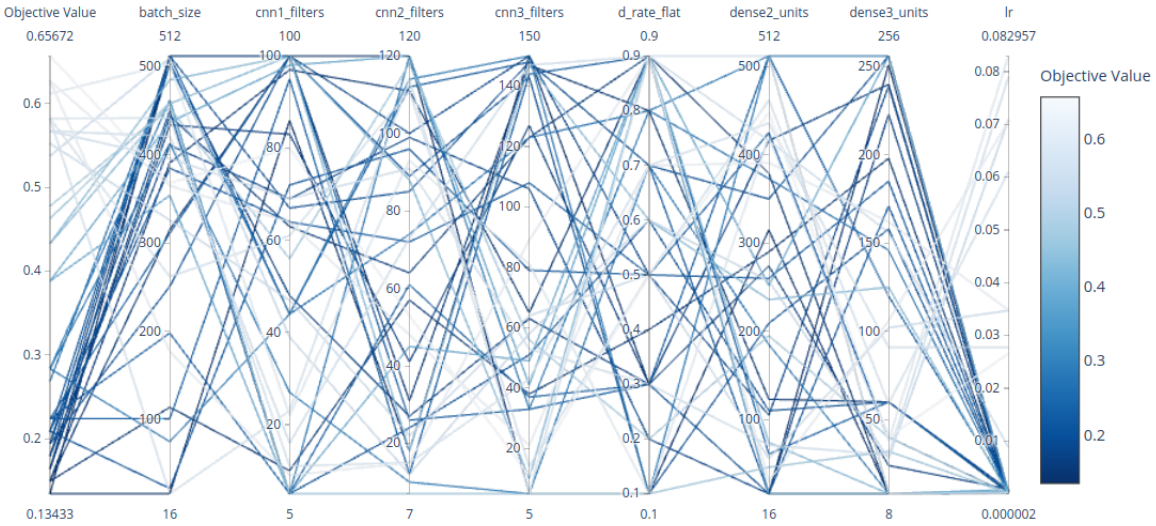

Figure B.2: Hyper-parameters search space exploration during the CNN optimization task on the PD dataset: each line represents the values used for each hyper-parameter and the objective value reached from those values. From the visualization it is possible to observe how the optimization mechanism has improved exploration during the epochs.

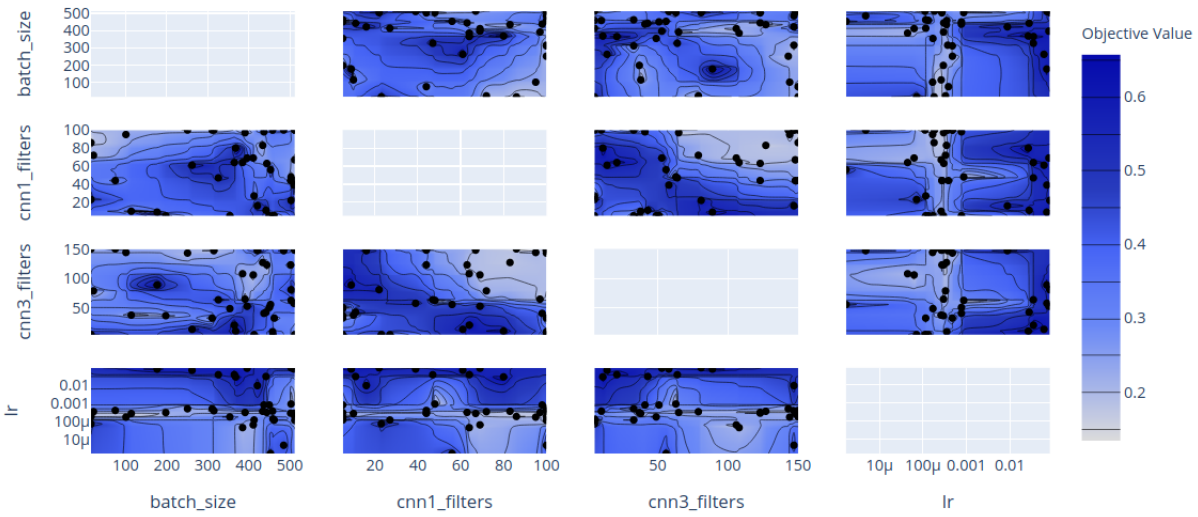

Figure B.3: Contour plot for the optimization over the PD dataset: at the end of the optimization phase, this visualization shows a contour that gives an intuitive spatial view of the search space.

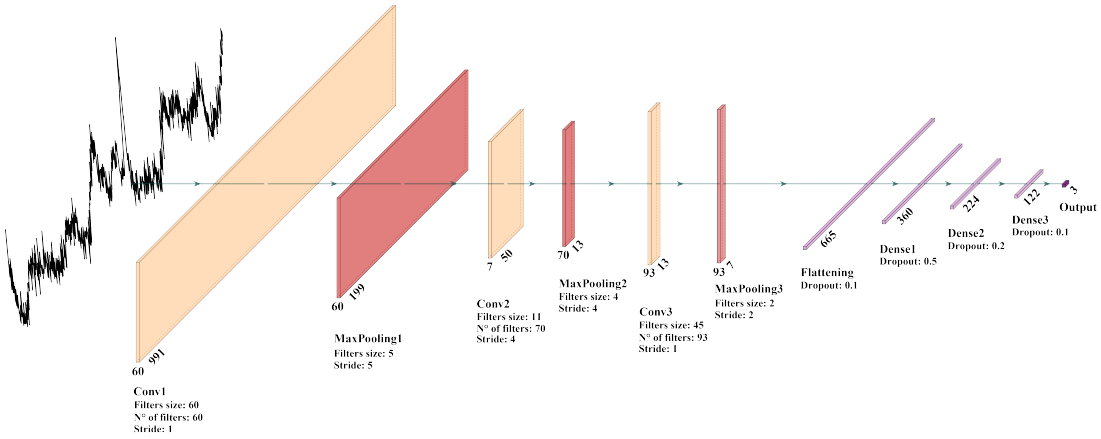

Figure B.4: A graphic representation of the best 1D-CNN model configuration obtained through the HPO process for the PD task. Figure taken from [1].

Table C.1: Comparison of ML models with respect to different preprocessing type, both for the spectra-level and patient-level perspective for the COVID task.

Spectra-level LeaveOneOut CV	Preproc type							
Model	raw	raw pca	l2	l2 pca	snv	snv pca	minmax	minmax pca
svm	0,689	0,697	0,712	0,692	0,709	0,704	0,7269	0,705
rf	0.516	0.726	0.663	0.721	0.676	0.706	0.648	0.663
xgb	0.604	0.812	0.7361	0.776	0.736	0.753	0.729	0.751
Patient-level LeaveOneOut CV	Preproc type							
Model	raw	raw pca	l2	l2 pca	snv	snv pca	minmax	minmax pca
svm	0.713	0.730	0.75	0.717	0.742	0.731	0.758	0.733
rf	0.52	0.782	0.51	0.772	0.56	0.792	0.55	0.762
xgb	0.75	0.861	0.78	0.851	0.782	0.861	0.777	0.861