SAPIENZA
UNIVERSITÀ DI ROMA

Parallel Convolutional Neural Networks for One-Dimensional Signal Denoising of Stimulated Raman Spectroscopy

Faculty of Information Engineering, Informatics, and Statistics
Master Course on Artificial Intelligence and Robotics

Luca Polenta
ID number 1794787

Advisor
Prof. Christian Napoli

Co-Advisor
Prof. Stefano Giagu

Academic Year 2021/2022

Thesis defended on 31 January 2023
in front of a Board of Examiners composed by:

Prof. Aristidis Anagnostopoulos (chairman)
Prof. Christian Napoli
Prof. Danilo Comminiello
Prof. Giorgio Grisetti
Prof. Fabrizio Silvestri
Prof. Andrea Vitaletti

:

Reviewer: Prof. Giuseppe Antonio Di Luna

Parallel Convolutional Neural Networks for One-Dimensional Signal Denoising of Stimulated Raman Spectroscopy
Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: polenta.1794787@studenti.uniroma1.it

Abstract

The goal of this thesis is to develop an algorithm based on neural networks to perform denoising of optical signals obtained by Raman spectroscopy in order to facilitate their interpretation and related applications. This is achieved through a parallel and residual neural network capable of isolating the Raman signal from spurious contributions and instrumental noise. The effort also focused on optimizing the neural network based on the study of different types and amounts of noise in the available datasets. This project is carried out through the collaboration with INFN (National Institute of Nuclear Physics) and the department of Physics at Sapienza, University of Rome.

Acknowledgments

I would like to thank my advisor Prof. Christian Napoli, my co-advisor Prof. Stefano Giagu and my tutor Giuseppe Fumero for their support during the development of this thesis.

I also thank my parents for always supporting me in my passion for computer science, which resulted in this wonderful master degree. Finally, I want to thank all my friends and family for believing in me. You are my energy.

Contents

Chapter 1

Introduction

Raman spectroscopy is a powerful analytical technique used to obtain information about the molecular structure of a sample. This technique is based on inelastic scattering of monochromatic light, by which the energy and characteristics of the molecules being analyzed can be determined.

The reason Raman spectroscopy is of great importance is that it has several advantages over traditional spectroscopy techniques, such as not requiring sample preparation or its destruction during analysis. It is also highly sensitive and can be used to analyze samples in the solid, liquid or gaseous state. This makes the technique suitable for a wide range of applications, such as the study of cells and tissues in medicine and biology, including cancerous ones, or to study the structure and molecular interactions of samples in chemistry. In any case, however, it is not exempt from several problems. First, it can be easily affected by environmental factors that can cause distortion of the spectrum line. In addition, the presence of other agents, such as water vapor, carbon dioxide and other gases, can interfere with accurate sample characterization. This results in the constant presence of a variable amount of noise. Such noise in the signal can be of a different nature and compromises the use of spectroscopic analysis.

Therefore, since the discovery of this technique, the research on Raman spectrum denoising has been an important topic over the past decades. It is important to

design effective denoising methods for the Raman spectrum which can help to improve the accuracy of Raman spectroscopy. Initially these techniques made use of noise reduction algorithms adapted to the specific case under consideration. In particular, they attempted to adapt the noise correction to the inherent characteristics of Raman spectrum. In general, most of these algorithms attempted to correct for baseline distortion, as the correction of additional types of noise requires very complex refinement and optimization to perform. However, since such noise is not the only one affecting the samples and the performance obtained was not always optimal or generalizable to different types of noise, studies have recently shifted to the use of artificial intelligence-based algorithms. In fact, they are able to learn autonomously the fundamental characteristics of the Raman spectrum and then remove the noise that distorts it.

Therefore, this thesis will address the creation of a new artificial intelligence algorithm for Raman spectrum denoising. This work will focus on two types of noise that are among the most common and influential in distorting sample analysis. The presence of different noise and at different levels of intensity results in networks that can become either overly complex or computationally heavy. To overcome this problem, the project does not involve the creation of a new type of neural network, but rather the innovative idea proposed is to aggregate different neural network components into a single model in order to achieve high performance at a low computational and time cost. In essence, a residual parallel convolutional neural network was proposed. The presence of parallel branches that are characteristically different from each other allow for improved learning of different fundamental aspects of the Raman spectrum. Subsequently, these parallel branches are processed to be finally subtracted with the network input. Such subtraction characterizes the residual nature of the network. This means that the parallel branches do not have to learn directly how to remove noise from the signal, thus returning the noise-free signal as output, but it must learn the characteristics of the noise to then allow subtraction with the noisy input signal in order to obtain the clean one. Furthermore, this characteristic will allow a

deeper and more complex network to be structured without affecting performance. This is necessary because this work focuses on two different types of noise, which are among the most common and influential in distorting the analysis of samples with Raman spectroscopy. Finally, the progress made with that neural network will be compared with some of the most efficient Raman spectrum denoising algorithms. This is made possible through the implementation of metrics for computing the error and similarity between the predicted signal and the corrected signal.
In summary, the key contribution made to the scientific community by this thesis is the creation of a new artificial intelligence algorithm applied to Raman spectrum denoising that has the following characteristics:

- Use of parallel branches to better study different signal features;
- Use of residual learning to perform Raman spectrum denoising with a deeper network that does not exhibit performance degradation due to its structure;
- Generic denoising since there is no focus on features peculiar to the origin of the signals;
- Attempting to train a unique generalizable model to the two different types of noise and their varying influences in the datasets.

Therefore, in the remainder of this thesis, after introducing the theory behind this project and related work, the proposed approach will be explained in detail and the experimental results obtained will be illustrated. Specifically, in Chapter 2 briefly presents Raman spectroscopy and its characteristics, including advantages and disadvantages. In addition, the specific technique from which the samples in the dataset are obtained will be explained, again delving into their characteristics. In Chapter 3 will be summarize the state of the art regarding Raman spectrum denoising. First, the most recent and famous noise reduction algorithms that do not make use of artificial intelligence will be discussed. Then the most relevant and interesting cases that make use of neural networks will be illustrated. Subsequently,

Chapter 4 will explain the datasets, their characteristics and the whole process of their pre-processing. In addition, it will explain both the metrics used to evaluate the performance obtained and the algorithms by which the results of the neural network will be compared. Chapter 5 will briefly describe the theoretical features that make up the different aspects of the neural network, concluding with a detailed description of its implementation. In Chapter 6, will discuss the experiments carried out to optimize and expand the capabilities of the neural network, evaluating and comparing the results obtained in order to determine what implies progress over the previous state of the art. Finally, in Chapter 7 the project results will be summarized and the consequences of the whole thesis will be discussed.

Chapter 2

Raman Spectroscopy

The field of interest underlying this thesis is Raman spectroscopy, an experimental technique in Physical Chemistry that is based on the Raman effect and involves the investigation of matter by scattering a monochromatic electromagnetic radiation. Its effects were first studied and observed by the Indian physicist, Chandrasekhara Venkata Raman in 1928 [1], and over time it has been the subject of considerable research by the scientific community due of the numerous benefits it brings. Its main characteristic is that it is a non-destructive and non-invasive material analysis technique that provides detailed information about the chemical structure and characteristics of an analyzed material. The core of this technique is that this information is unique and specific about the biochemical composition of the samples [2]. In addition, it is possible to analyze materials in the solid, liquid or gaseous state, and at different orders of magnitude, that is, macromolecularly or atomically. In order to understand the principles and characteristics behind the topic under consideration, this chapter will discuss: the physical principles of the general technique (Subsection 2.1); the advantages of using this technique (Subsection 2.2); the limitations to be overcome (Subsection 2.3); and finally, a new and sophisticated method based on Raman spectroscopy will be illustrated (Subsection 2.4), namely Femtosecond Stimulated Raman Spectroscopy (FSRS) which has been taken here as a model case to study denoising by means of of artificial intelligence algorithms.

2.1 Physical Principles

Raman spectroscopy is based on the study of photons passing through and interacting with matter when the sample is illuminated by a monochromatic source of electromagnetic radiation. Regarding energy exchange with the material, photons can exhibit three types of behavior: being absorbed, being scattered while conserving their energy (elastic or Rayleigh scattering) or finally being scattered while gaining or giving up energy (inelastic or Raman scattering) [3, 4]. These characteristics are illustrated in the following Figure 2.1:

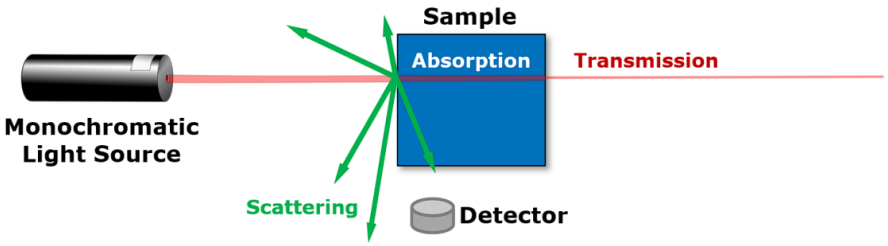


Figure 2.1. Raman spectroscopy acquisition process. Source: [4]

This technique without any expedient is called spontaneous Raman spectroscopy. In addition, the choice of laser source can be very wide, and depending on that choice, there are several advantages that mitigate some known problems. As specified in reference [5], generally, radiations that belong to the visible or infrared spectrum are used. Ultraviolet wavelength photons might also work, but they tend to cause photo-decomposition of the test sample.

In general, Raman scattering records changes in the electric field of the chemical molecule when it interacts with incident radiation. In this way, a measure of the deformability of a bond in an electric field is made. This factor depends essentially on the ease with which the electrons in the bond move, inducing a temporary dipole. A large concentration of free electrons in a bond leads to high polarizability, and the molecule will have an intense Raman signal.

Without going into the specifics of microscopic process description, which is beyond the scope of this paper, it is possible to denote that what is expected as a result of an

analysis of Raman spectroscopy is an emission at the same frequency as the incident radiation, so the outgoing wave will carry the same energy (Rayleigh Scattering), and then lower and higher frequencies there may be additional terms termed Stokes and Anti-Stokes, respectively. Usually, under standard conditions, Stokes peaks are significantly more intense than Anti-Stokes peaks. This effect results from the fact that the vibrational state of the atoms inside a molecule does not have a random energy, but has a well-defined and discrete one, called a level (quantization). Which levels of those are available depends on a probability distribution called the Boltzman distribution. When the temperature is equal to zero, only the lowest energy level called ground-state is accessible and sensitive to the Raman spectrum. As the temperature rises, other levels become accessible with increasing probability. Since an anti-stokes peak cannot occur unless there are higher energy levels than ground-state while stokes peaks do, at temperature equal to zero only stokes occur, while at increasing temperatures both occur, but the anti-stokes are less intense. These characteristics are shown in the following Figure 2.2:

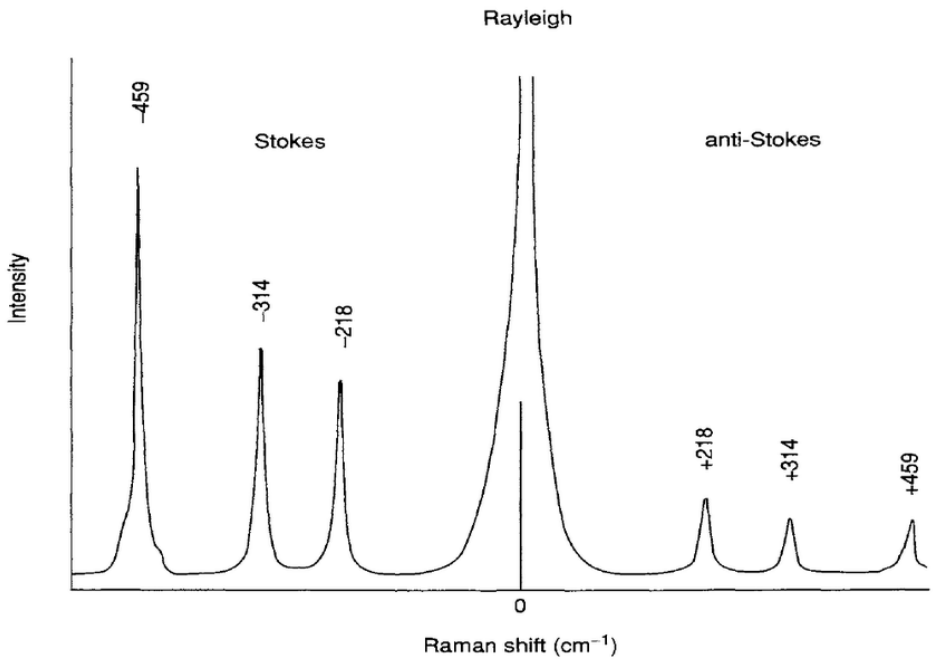


Figure 2.2. Raman spectrum. The spectrum includes a Rayleigh peak, Raman/Stokes scattering peaks and Raman/anti-Stokes peaks. Source: [6]

Finally, one last observation that can be reported concerns the fact that the graphs of the Raman spectra are always around zero in the x-axis. As can be seen from Figure 2.2 above, the x-axis is re-scaled so that it is centered at the Rayleigh peak. This is called "Raman Shift" and corresponds to the difference between the frequency observed in the Raman spectrum and that of the Rayleigh peak.

2.2 The Advantages of Raman Spectroscopy

The main advantage of this technique is that it can be applied to different types of materials. In each of these cases, the resulting signal is rich in features from which various useful information can be extracted. The features are available in a visual representation in the following Figure 2.3:

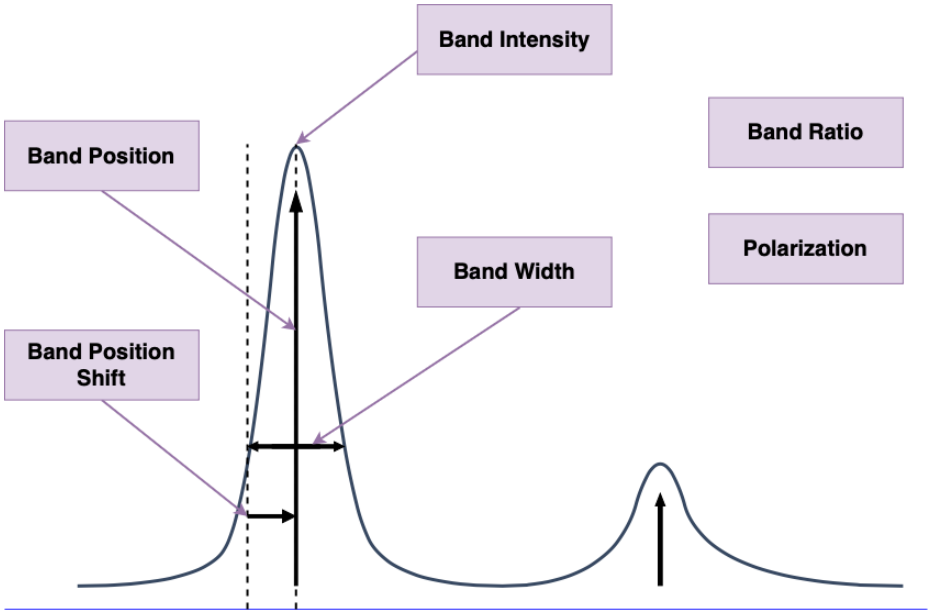


Figure 2.3. Analysis of the wave obtained by Raman spectroscopy

As illustrated in the reference [7], the information that can be extracted from each feature is as follows:

- **Band Intensity**: Reports information regarding the polarizability of chemical bonding, its orientation and detectability;

- **Band Position**: It is used to trace the chemical structure of the element;
- **Band Position Shift**: It is useful for studying the material response under deformation, pressure, tension and temperature changes;
- **Band Width**: In a solid, this measure quantifies the quality of the crystal structure (crystallinity), its defects, doping content and amorphism, that is the order of the atoms in its structure;
- **Band Ratio**: It measures the chemical concentration of the element;
- **Polarization**: An estimate of the molecular orientation can be derived from this measure.

All these features make this technique of fundamental importance for the study of carbon-based and inorganic materials. In addition to biological investigations, however, it is used in a wide variety of fields, including planetary exploration, industrial process control, and even in the medical field as, in combination with artificial intelligence algorithms and neural networks, it is used to study cancer cells, melanomas, and similar diseases for diagnostic purposes. Moreover, as reported in the reference [2] that analyzes advantages and disadvantages of this method, it also enables the analysis of materials with high specificity without the need for labels and by simultaneously detecting several macro-molecules.
However, this technique is not exempt from disadvantages and the next subsection will discuss its main limitations.

2.3 Limits and Challenges of Raman Spectroscopy

Similar to any scientific method, Raman Spectroscopy has limitations [2] that may affect the quality of the data collected. These limitations are:

- A low sensitivity;
- Weakness of Raman signals results in long acquisition times;

- The long exposure times and low scattering efficiency make imaging by point scanning challenging, that is, it is very complex to analyze the biological circuits of microorganisms, plants and animals;
- Sophisticated data analysis is often required;

The last issue is due to various reasons, such as the inability to always perform material analysis in ideal situations, but mostly it is due to the fact that this method is very prone to noise. However, new techniques have been developed over time that have mitigated some of these problems, although they have also introduced new problems to be addressed.

2.4 Femtoseconds Stimulated Raman Spectroscopy

As explained in reference [8], there are basically two main problems related to spontaneous Raman spectroscopy: first, Raman scattering is a weak process, since spontaneous scattering is induced by the weak isotropic radiation field and is easily overwhelmed by fluorescence. In fact, the latter can be orders of magnitude more intense and is commonly present as background in Raman spectra. Second, Raman experiments are limited to the picosecond time domain, making it difficult to study structural details in faster reactions. This is due to the fact that there are limitations caused by the Fourier transform that relates the shape in time and frequency of the laser impulse used. If the laser impulse is short in the time domain, it will have a wide signal in the frequency domain (and vice versa). In Raman spectroscopy, an attempt is made to use a laser with a narrower frequency than the Raman peaks, otherwise the true peaks would not be perceived and just a convolution with the spectrum of the laser pulse would be detected. The problem occurs when the variation in Raman spectra over time is to be studied. In this case it is necessary to have a laser impulse that is short in the time domain but produces a corresponding one in the frequency domain that is too broad. These two requirements, namely to have a laser that is short in time and narrow in frequency, are not satisfiable

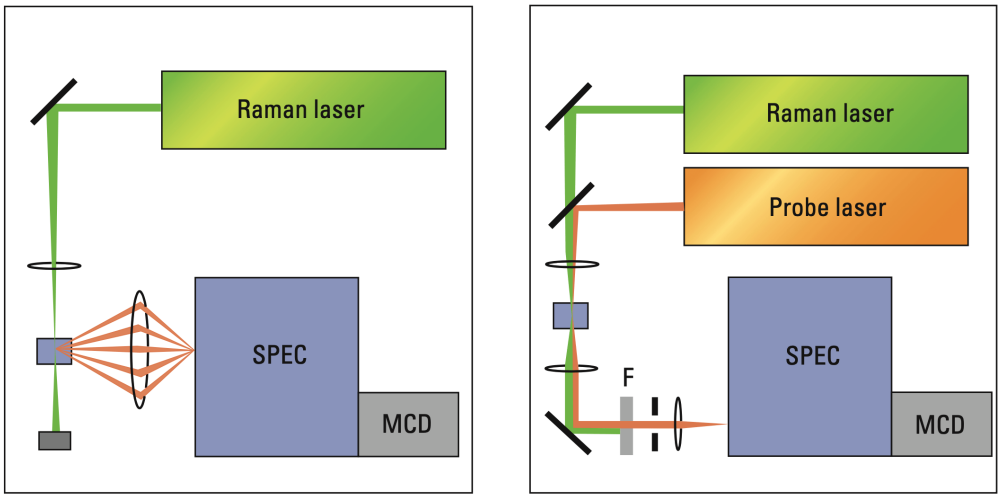
except within a certain margin that remains limited to picoseconds at most. To address this, as well as other issues, several variations of this technique have been developed over time. One of the most interesting techniques that seeks to solve these problems is Femtosecond Stimulated Raman spectroscopy (FSRS). It allows switching from a picosecond time scale ($1ps = 10^{-12}s$) to a femtosecond (fs) time scale ($1fs = 10^{-15}s$). This change is of fundamental importance for the study of extremely rapid reactions, such as chemical and physical reactions occurring at the atomic level. The basic idea is simple: instead of using a single light source to observe spontaneous Raman scattering, the FSRS uses two lasers. The second laser simply drives and stimulates the Raman transitions. It is in fact called a "probe" laser. It induces and stimulates Raman scattering because, when the classical Raman laser and the probe laser simultaneously illuminate the sample, a transition of photons from the Raman pulse to the probe pulse occurs, resulting in an increase and gain in probe intensity. Stimulated Raman scattering is in contrast to the randomly oriented scattering of spontaneous Raman, but achieves the same result with the advantage of a more intense and direct photon flux. The difference between the two techniques can be seen in Figure 2.4.

This new approach has numerous advantages, such as the fact that the technique becomes insensitive to spontaneous background fluorescence, provides an extremely high data acquisition rate, and it is thus possible to produce spectra with a high signal-to-noise ratio. Therefore, much of the previously mentioned problems such as weakness, long exposure times and sensitivity to fluorescence are limited. However, even this technique is not exempt from having problems. It is always a very sensitive technique. This means that it can be easily damaged by environmental factors such as heat, light and humidity, or by noise from the instruments and sensors used. In particular, it can manifest itself mainly in two ways:

- All along the discretized signal as a noise that creates more or less intense random variation from one point to the next one;

- As baseline distortion which tends to increase or decrease rather than be constant all along the discretized signal.

As reported in the reference [9], even in the off-resonance regime, additional non-linear spectroscopic contributions occur that are superimposed on the collected signal and form an unwanted background. They are even due to physical phenomena related to the action of the Raman pulses and the probe and can interfere with the time-resolved Raman spectrum. Additionally, line-shape distortions can occur due to sample dynamics.



(a) **(b)**

Figure 2.4. Comparison of Spontaneous Raman Spectroscopy (a) and Femtoseconds Stimulated Raman Spectroscopy (b). Source [8]

Consequently, given also the importance of many fields of application of this method, it is possible to understand the importance of having a signal as clean and correct as possible to enable a realistic and reliable study of the material. In fact, this work focuses mainly on improving the study of the sample by performing post-processing of the data in order to remove most of the noise that afflicts them.

Chapter 3

State-Of-The-Art for Raman Spectrum Denoising

The research on Raman signal denoising has been an important topic over the past decades. Initially, the methods did not involve artificial intelligences, but relied exclusively on sophisticated signal correction algorithms that attempted to remove the noise while keeping signal characteristics intact. Typically, these algorithms were derived from generic techniques with proven efficiency in similar fields, to which modifications were made in order to consider the features present in the specific case under consideration. In this chapter, some of most efficient algorithms will be discussed (Subsection 3.1). Despite their wide applicability, such techniques were not always able to fully unearth the spectral properties of the Raman signal. For example, some of these techniques do not consider the spectral correlations between complex signal lineshapes and the baseline, while, for others, the optimization with respect to the case under consideration is very complex and specific, thus also not generalizable to the variability of the possibly occurring noise. This has led to the development and implementation of new and better performing techniques. Therefore, with the emerging and growing interest in the field of artificial intelligence, it has been applied to the denoising of Raman spectroscopy. In particular, machine learning is the specific type of artificial intelligence relevant to this field. The name

machine learning was coined in 1959 by Arthur Samuel and refers to algorithms that use statistical techniques to give computer systems the ability to progressively improve performance on a specific task by learning from data. Therefore, the greatest advantage of such algorithms concerns the fact that they do not need to be adapted to all specific signal characteristics. They automatically try to understand all features of the signal, and only certain hyperparameters need to be set. Initially, simpler networks were structured to perform this task and only some of their internal aspects, such as loss functions, were adapted to better suit the task at hand. Subsequently, as artificial intelligences grew exponentially in complexity, machine learning methods were created which were increasingly efficient, but also more expensive in terms of power and execution time or overall more difficult to optimize. In the last part of this chapter, the most successful machine learning techniques applied to the field under consideration will be discussed (Subsection 3.2).

3.1 Signal Denoising Algorithms

Raman spectroscopy is a technique that dates back to the first half of the twentieth century, and so several studies on signal denoising have been conducted since then. Without going through the entire history of the techniques used, it is more coherent to discuss these techniques by grouping them into categories and delving into some of the work that has demonstrated comparable results with more recent neural networks.

A first approach to carry out denoising of the Raman spectrum is to decompose the signal into its constituent parts and then reconstruct it later using only the coefficients related to the desired features. This set of constituent parts forms a complete basis that tries to be as orthogonal to the original signal as possible. In this way it is possible to use only the coefficients related to the signal of interest and eliminate unwanted noise. One way to perform signal decomposition is to use wavelets. They are created by taking the Fourier transform of the signal and

using a window function to constrain the frequency range of the resulting wavelet. The process is then repeated for different times, scales and frequencies, resulting in a series of wavelets that can be used to represent the signal. These wavelets are then combined in a hierarchical structure, starting from the lowest frequency and highest scale wavelet to the highest frequency and lowest scale wavelet. This combination of wavelets is known as wavelet expansion and results in a set of coefficients representing the signal. These coefficients are finally used to reconstruct the original signal. One method that makes use of wavelet decomposition is Wiener estimation. This technique is particularly useful for removing noise caused by sensors or electrical interference that can afflict the Raman spectrum, and a case of its use in this field can be found in paper [10]. This procedure offers several advantages: for example, it is possible to obtain a solution that does not take into account the unique features generated by certain types of materials by Raman spectroscopy and is able to perform the denoising of the signal in a more universal way. In fact, the results obtained from this project with three types of samples (including a phantom sample, a human fingernail, and leukemic cells) has shown huge improvements in performance over two commonly used denoising methods, namely moving average filtering and Savitzky-Golay filtering. This method provides comparable or even better denoising performance in cases with low signal-to-noise ratios. In addition, the performance of the proposed method is significantly less sensitive to parameter choice. However, the problem with this methodology is that each wavelet family is more or less suitable for certain functional forms of baseline and noise. In principle, wavelet decomposition can also be used for all types of noise in the Raman spectrum. Nevertheless, it is not easy to construct a denoiser that works well with different spectra. In fact, it would be necessary to manually optimize many aspects of the wavelet family to best fit the spectrum under consideration, and this is neither simple nor easily generalized when you have different types of noise.

Another interesting method of denoising the Raman spectrum by signal decomposition is achieved through a technique called EMD (Empirical Mode Decomposi-

tion) [11–13]. This method, like the previous one, can also be categorized as a wavelet noise reduction methods. In this case, decomposition of the original signal occurs without leaving the time domain and is used to analyze non-linear and non-stationary signals. Moreover, it is not necessary to analyze or pre-process the signal before denoising it. Through an iterative process, a signal $x(t)$ is decomposed into its intrinsic mode functions (IMFs). In mathematics, IMFs of a signal are functions used to represent the signal regardless of the coordinate system used to describe it. This set of functions forms a complete basis that attempts to be as orthogonal as possible to the original signal. However, this technique can present two issues: first, the algorithm works optimally if the number of zero crossing points (i.e., the number of points at which the function changes sign) and the number of peaks are equal or at most differ by one unit, and this characteristic is not necessarily present in a signal obtained by Raman spectroscopy. Therefore, the algorithm cannot iterate more than once resulting in it being less effective. The second problem is related to a mathematical property, namely, the envelope of a function computed on Raman spectra. The envelope of a function is the curve tangent to each point of the signal. Generally, the average value of the upper envelope and the lower envelope is zero in the case of the Raman spectrum. This means that the upper envelope lines and the lower envelope lines are locally symmetrical with respect to the x-axis of frequencies. This characteristic compromises the proper functioning of the technique. Therefore, this technique is useful only in special cases where the Raman spectroscopy signals exhibit all the characteristics necessary for the algorithm to work properly.

Another very interesting category of methods for spectrum denoising is those based on asymmetric least squares (ALS). This approach is essentially based on the ordinary least squares method. It defines a function as the sum of the squared difference between the noisy Raman signal and the baseline to be estimated. Then, iteratively, this function is minimized. The asymmetric version was first proposed by Eilers and Boelen [14] in 2005, and has been taken up by other works as well [15]. In essence, an additional term is introduced into the weight in front of the reconstruction term (the

MSE). It needs to be estimated and minimized to ensure that the estimated baseline is as smooth as possible. This ensures that the most important features of the Raman spectrum or signals similar to it are not corrupted. The ALS algorithm adapts poorly to very intense peaks, such as those often found in the Raman spectrum. This is due to the way the weights are used in the penalization: if they have a small value, the residual peak might be large enough to significantly affect the baseline fit, whereas if too small values are used, the baseline will fit poorly and will not remove the noise well properly. Therefore a variant of the ALS was proposed in [16] and it is called PSALSA (Peaked Signal's Asymmetric Least Squares Algorithm). It introduces an adaptive value for the weights that is based on an exponential trend. As a result, regions with the most intense peaks will exhibit large residuals and smaller weights, while regions of peak-free noise will have small residuals and weights close to the generic parameter p, which is no longer necessary to choose excessively small. The reported results show that it was also possible to adapt this algorithm to the case of the Raman spectrum, although not all types of noise that may affect the signal were removed. In fact, a frequently occurring noise is that which afflicts the entire signal through point-by-point fluctuations. In this work, as in the other works cited in this subsection, it was not always well removed because an additional step is often needed to remove it completely. In addition, it is also necessary to consider the fact that the ALS and PSALSA assume that peaks are always defined as positive, but in the FSRS there are dispersive peaks that are not always defined that way. This makes this method not applicable to the case under consideration in this paper.

Finally, a last category of techniques is Polynomial Fitting-based methods. For example, this is the case in paper [17], which proposes a method called "Adaptation of Cubic P-Spline Based on Morphology". It is based on fitting penalized cubic splines and mathematical morphology. Mathematical morphology is a nonlinear technique consisting mainly of two basic operations, called erosion and dilation, and the combination of these operations provides two other operators, namely closure and opening. Without going into too much detail, it is possible to use such operations

to generate sequences of nodes for use in fitting cubic p-splines in order to denoise the signal. For example, through closure one can provide a rough estimate of the Raman spectrum. The use of polynomials to model regression functions and perform curve fitting of various types is a technique already widely used in the past. The novelty of this method, however, lies in the use of P-splines. In general, splines are mathematical functions that describe a set of polynomials interconnected at specific points called spline nodes. Compared with more classical fitting methods, the main advantage of splines is that the polynomial equation is not the same for the entire set of points, so the fitting function can change from one point interval to another, allowing for better fitting and interpolation of much more complicated point distributions. In addition, P-splines are regression splines fitted to least squares with a high roughness penalty that avoids overfitting. The reported results are interesting in that they report both better values in the metrics used to compare with other algorithms and qualitative improvements when graphically analyzing the spectra. However, even in this case, not all of the two main types of noise that can plague the Raman spectrum are properly removed.

In conclusion, the results obtained from these denoising algorithms are certainly very interesting, but they still have some very complicated issues to deal with, such as having to adapt them to each feature of the signal to be denoised or the fact that some types of noise are not completely removed. This shows how it is necessary to take a step forward methodologically and technologically by introducing new approaches based on neural networks.

3.2 Deep Learning Neural Networks

As mentioned earlier, a new methodology for approaching signal denoising makes use of neural networks. In this subsection and the next one, artificial intelligences adapted to spontaneous Raman spectroscopy will be illustrated. Although it is different from the stimulated Raman spectroscopy that is covered by this paper,

they are nevertheless striking examples of how such techniques are applicable in this field.

The use of neural networks begins with the implementation of initially simpler but still very efficient models. In the article [18], a DNN (Deep Neural Network) model obtained by joining two CNN in parallel was developed. The two CNNs are structured quite simply: they are a series of convolutional layers alternating with MaxPooling layers and using "ReLU" as the activation function. Eventually, the results of the two parallel networks are concatenated by a fully connected layer before being passed to the regression output layer. The structure of the network is shown in Figure 3.1.

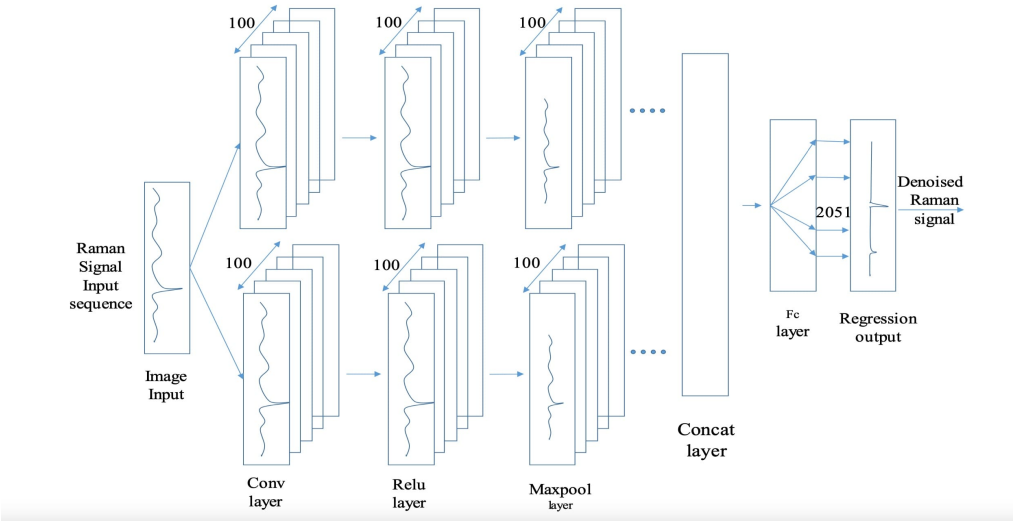


Figure 3.1. Structure of the parallel network in [18]

According to their results, the use of two CNNs in parallel makes it possible to better study the signal characteristics and thus to recognize it in the presence of much noise. The results obtained from this procedure are much better than previous denoising methods that did not use neural networks, and also performing better than simpler neural networks such as a single CNN. However, these are only the first glimpses of the potential that can be achieved through neural networks. In fact, subsequent work has achieved better results using the same signal analysis metrics. For example, other works such as the one presented in the reference [19], start

from to use basic architectures that are already widely used (like CNNs) and focus efforts on refining the parameters used by the network to train. In fact the proposed architecture is very simple. It is a fully convolutional neural network for denoising. The model consists of five convolutional layers and the activation functions are ReLU. Generally, MSE loss is very common in this type of architecture, but it has the problem that it can give higher priority to denoising areas with limited spectral information (regions of low intensity variation) than to regions of higher intensity variation. The latter, which correspond to signal peaks, are a priority because their shape, intensity, duration and the timing in which they occur are key features that we want to study, denoise and preserve. Therefore, in order to give more priority to these regions, a variant of the loss MSE was developed and it is computed as follows:

$$\mathrm{LOSS}(\mathrm{x^e}, \mathrm{x^{ref}}) = MSE(x^e, x^{ref}) + \alpha * MSE(x^e_{peak}, x^{ref}_{peak}) \tag{3.1}$$

where MSE is computed as:

$$\mathrm{MSE}(\mathrm{x^e}, \mathrm{x^{ref}}) = \frac{1}{N} * \sqrt{\sum_{i=1}^{N} (x_i^e - x_i^{ref})^2} \tag{3.2}$$

where x^e and x^{ref} are the samples located in the region around the single most prominent peak in the spectrum (easily identified by the maximum value in the spectrum), and α is a weighting factor that determines the priority the algorithm gives to smoothing over peak preservation.

In addition, a modified version of the SNR was also developed to evaluate the results reported by the network. Without going into detail, this version, like the loss variant used, focuses on regions where there are peaks to compute the SNR.

The results obtained by this network demonstrate how noise can be removed from a spectrum by a convolutional neural network. Moreover, the SNR value is four times better, which corresponds to an experimental acquisition that would take four times less long to mitigate shot noise by doing statistics, that is, averaging the results

of several acquisitions. At first glance this might suggest that the network works excellently, but in reality there is a big limitation: the network focuses on noise in the peak region, but it may not necessarily occur only there. It could also occur along the entire signal or it could occur as a baseline distortion. However, by taking into greater consideration only portions of the signal, a complete cleaning of the noise is not done and neither is an adequate cleaning for each type of noise.

There are also several papers that make use of more complex neural networks to perform Raman Spectrum denoising. For example, an unsupervised deep learning neural network capable of performing both denoising and segmentation of the Raman signal was developed in [20]. The network, called "UHRED" (Unsupervised Hyperspectral Resolution Enhancement and Denoising), is able to perform the denoising with a "one-shot" technique that learns all the most relevant features from a single sample. Furthermore, a really interesting feature of this network is that it allows the sample to be devoid of requirements that other types of networks might require, such as labels. However, to perform denoising of the one given sample, the network applies a k-means clustering algorithm to the processed data and segments it. The results obtained show that UHRED unequivocally improves hyper-spectral contrast in low SNR images. The implementation of the k-means clustering algorithm in latent space also led to excellent results in the case of unsupervised segmentation. This leads to a more intuitive map of chemical species, but still has limitations. In fact, multi-modal signals such as harmonic generation, fluorescence, thermal lensing and others are not included. The researchers of this project believe that as computing power continues to improve, the UHRED scaffold can be further expanded and generalized to include more cases.

Furthermore there are additional works that attempt the denoising of results obtained by Raman Spectrum through convolutional networks dedicated to image denoising. In particular, the U-Net was used in [21]. U-Net is a popular FCNN (Fully Convolutional Neural Networks) architecture. It was created as a network for biomedical image segmentation, and this type of network is trained to learn the segmentation map

from a single corresponding input, using only pixel-level supervision. This is possible because FCNNs are constructed as an encoder-decoder network. The encoding part of the network is typically a standard CNN, which is used to extract the most representative features of the input. Instead the decoding part of the network is designed to over-sample the feature map and produce a segmentation map. The architecture of the network can be seen in the following Figure 3.2 that was taken from the original paper [22] in which the network was first presented:

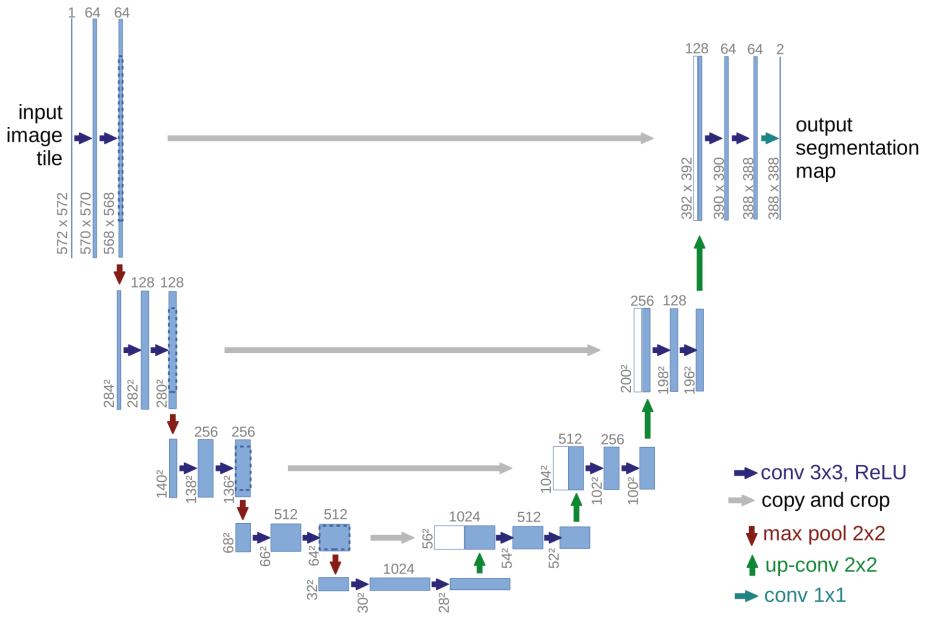


Figure 3.2. U-Net network architecture. The blue blocks correspond to multichannel feature maps. The number of channels is indicated at the top of the block, while the x-y dimension is indicated at the lower left edge of the block. White boxes represent copied feature maps. Instead, the arrows have different meanings indicated by the legend in the image itself. Source [22]

This architecture, however, has been modified to work with a smaller and less populated dataset. The most significant modification involves the replacement of pooling operations between convolutional up-sampling operators. Consequently, the output is more resolute and thereby allows the subsequent convolutional layer to learn more detail and with greater accuracy. In essence, the idea behind the whole network is that some convolutional layers are structured with filters that can

perceive macroscopic details (such as optical aberration, uneven illumination, object shape and size, and others), while other layers with different types of filters focus on microscopic details (such as shot noise, pixel-to-pixel variations, fine structural features, and others). The results obtained in this work are certainly better than those obtained in other work using less powerful techniques, especially since it focuses on different types of noise. In any case, however, it still has drawbacks as well. As the authors themselves point out, the main limitation is that this method requires an appropriate dataset, which, however, compromises the generalizability of the process. In fact, they trained two different models for the two types of denoising they wanted to perform. As much as this leads to more accurate results in their respective fields, it does not help to develop a universal and generalized system.

In the end, another interesting work is the one proposed in [23]. In it, an encoder-decoder architecture is adopted. The network involved three encoding layers and three symmetrical decoding layers, as well as a final convolutional layer. Generally, the encoding part learns a reduced-size representation of the dataset and then the decoding part tries to reconstruct from that representation the original data, ignoring insignificant components such as noise. In addition, in this work a custom loss was created to prioritize regions where peaks occur. This is necessary because such regions are often few and small compared to the totality of the signal, and if they are not properly considered, they may happen to be underestimated. In addition, this loss makes sure to recognize a peak by comparison with the baseline: if the peak height exceeds a certain threshold, the MSE controller parameter is activated to consider that region. The architecture can be seen in Figure 3.3.

The results reported that the peaks are cleaned of noise without compromising their fundamental characteristics, so high-fidelity work seems to be done, although there may be imperfections underlying them. The work, however, again shows how they concentrate more on the peak regions, thus ignoring the rest of the signal. Since noise can occur anywhere and in different ways, this approach does not perform complete noise denoising for the Raman Spectrum.

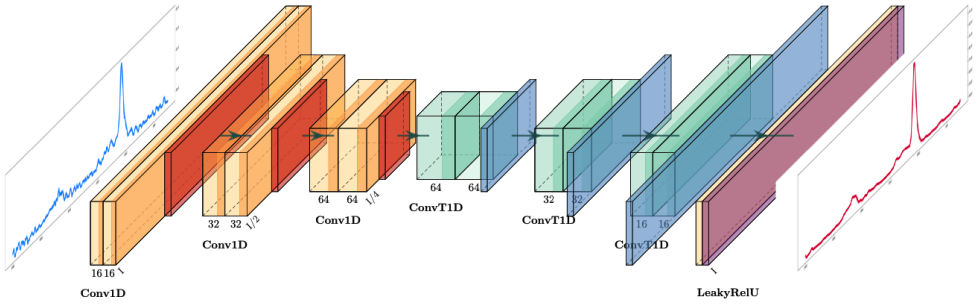


Figure 3.3. The network architecture of paper [23]. The blocks in yellow and orange represent the encoding part, while green and blue is represented the decoding part. Finally, in violet is the final 1D convolutional layer. Source [23]

Chapter 4

Problem Definition

As mentioned earlier, the intent of this work is to achieve the ability to cope with the two types of noise present in the stimulated Raman spectra and to create a neural network that works best with different types of noise at different intensity levels. For this purpose, different datasets with various features among them were used (Section 4.1). In addition, to evaluate the work quantitatively, several metrics were implemented and used. Finally, the results reported are compared with the methods already recognized as the most efficient. The explanation of each will be carried out (Section 4.2).

4.1 Datasets and Pre-Processing

There are three datasets used in this work. Each of them consists of 5500 samples, 5000 with different types of peaks and 500 without peaks, which during the training process are divided between an 80% training and a 20% test set. In addition, 15% of the training set is used as validation set during the training, so overall a 68% of the dataset is used as training set, a 12% as validation set and the remaining 20% as test set. Each sample consists of 801 frequency steps. Furthermore, each dataset is pre-processed before being passed to the network. In fact, the training set is normalized to the standard deviation computed on the set itself, and then

each sample is shifted by the mean of the samples. Also, the same normalization is done for the test set, but the scaling is performed with its own mean and standard deviation. Finally the whole set is normalized to the absolute value of the maximum of the training set. This pre-processing should help improve training performance. The main difference between one dataset and another lies in the two types of noise contained in it. These data sets are generated through a simulation model that reproduces the experimental results, and during their creation it is possible to establish two parameters that define the types of noise:

- The relative intensity between baseline and signal peaks denoted by the parameter **mu_TA**. It is computed as the ratio of the first to the second value. The smaller the ratio, the less the peak towers over the signal. This factor is not externally controllable, but depends on the physics of the sample being measured.

- The shot noise, which causes the noisy point-to-point fluctuations due to the inherent firing noise in photon measuring devices. It is controlled by **"number of acquisitions"** parameter. The larger this value is, more spectra can be averaged to decrease the relative influence of the shot noise. Finally, this factor is a parameter that can be established in the laboratory, thus controllable to a certain amount.

The names and characteristics of the three datasets are as follows:

Dataset	Number of Acquisitions	mu_TA
Mixed Noise	[150 150]	[1 100]
Low Noise	[100 100]	[1 50]
High Noise	[1 1]	[1 50]

Table 4.1. Characteristics of the three datasets

While graphic examples of some samples from each dataset are in the following Table 4.2:

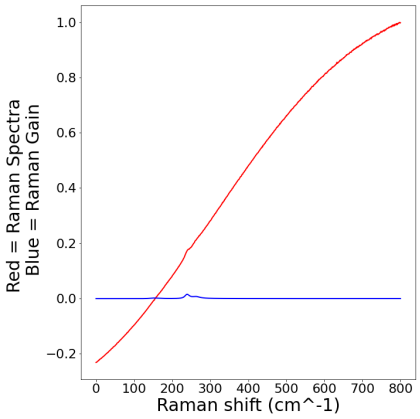
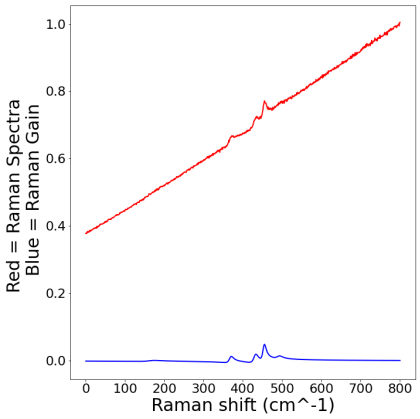
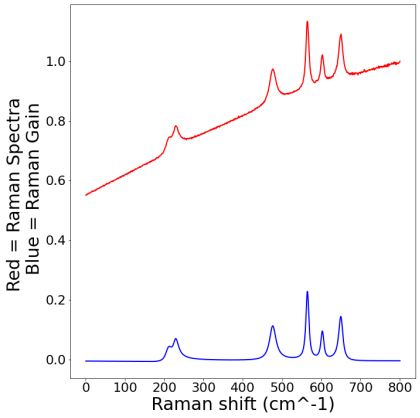
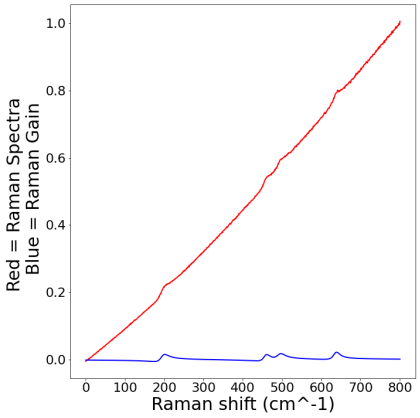
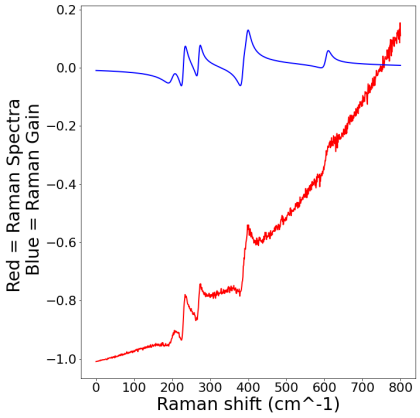
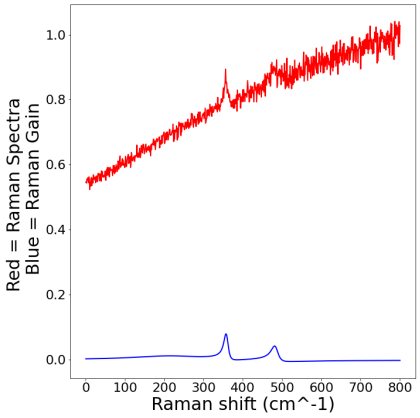
MIXED NOISE		
LOW NOISE		
HIGH NOISE		

Table 4.2. Samples from all dataset types

The samples are extracted randomly from a flat distribution between the extremes specified in the Table 4.1. The high value of the number of acquisitions of the Mixed Noise and Low Noise datasets indicates that there will be little noise along the entire signal and will be slightly higher in Low Noise as it is calculated with slightly smaller values than Mixed Noise. High Noise, on the other hand, presents a large amount of this type of noise. In addition, the Mixed Noise has less soaring peaks than the High Noise and Low Noise datasets because it has more variability due to the fact that mu_TA is randomly extracted between 1 and 1/100, as well as having a lower minimum value.

4.2 Evaluation

The complexity of the proposed task, i.e., computation by regression of the one-dimensional signal obtained from Raman spectroscopy, requires the definition of objectively comprehensive criteria for evaluating the performance of the implemented models. In fact, given the high variability of the cases, a qualitative and graphical analysis of the samples obtained in output is not sufficient. To do so, several metrics were developed, which will be made explicit in Subsection 4.2.1, and compared with different methods, illustrated in Subsection 4.2.2.

4.2.1 Metrics

As a first metric, a manually written custom peak finder is used. It is based on the study of ground truth and predicted signal. First, all peaks of the ground-truth are found by computing the second derivative of the signal and verifying that there is a sign change around the peak. Once the indices of the peaks are found, as well as saved for later comparison with those found in the predicted signal, a minimum threshold of the amplitude is established to consider a peak in the variations in the predicted signal. This was necessary because there are constant micro-variations in the predicted signal between one point and the next in the signal, and it was

necessary to establish the minimum variation to be considered a peak, although not necessarily corresponding to an actual peak in the ground-truth. In addition, again through the study of the second derivative and the prominence of peaks in relation to their lowest point in the region in which they occur, variations falsely marked as peaks are eliminated. Once this step is completed, each peak detected in the predicted signal is compared with that detected in the ground-truth to verify that they occur in the same region. Once the peaks are then matched or mismatched, several basic metrics are computed to validate training performance. Specifically, they are as follows: TP (True Positive), i.e., the number of spikes correctly predicted; FP (False Positive), i.e., the number of spikes incorrectly predicted as such; FN (False Negative), i.e., the number of spikes not predicted. Through them it is possible to calculate additional metrics such as:

- **Precision:** this metric determines the percentage of reliability of positive responses. It is the ratio of correctly predicted positive predictions to total predicted positive predictions. It can often lead to a misleading analysis in that high accuracy values can be obtained even when the network is not optimal, so in general it is a useful tool to compare with the others that will be performed below. It is computed as follows:

$$\text{Precision} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Positive (FP)}} \tag{4.1}$$

- **Recall:** is the ratio of correct positive predictions to all predictions that should have been predicted. Therefore, it measures the proportion of relevant results that are correctly identified by the algorithm. The formula is the following:

$$\text{Recall} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Negative (FN)}} \tag{4.2}$$

- **F1-Score:** it is an overall measure of a model's performance that combines precision and recall. Having a good F1-Score means having few false positives

and few false negatives, thus predicting correctly. An F1-Score is perfect when it equals 1, while 0 corresponds to incorrect predictions. The formula is:

$$\text{F1-Score} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{\text{FN} + \text{FP}}{2}} \quad (4.3)$$

Obviously, such metrics are not sufficient for objective validation of the results because they are based exclusively on the peaks and not on the general trend of denoising throughout the whole signal. Therefore, several other metrics were integrated to study the predicted signal in its entirety. The first of them is **NMSE** (Normalized Mean Square Error). The value of the NMSE will be between 0 and 1. The smaller the value, the more similar the two data sets are. However, it is important to denote that it is only a measure of similarity, not of accuracy. NMSE is defined as follows:

$$MSE(x, y) = \frac{1}{N} \sum_{i=1}^{N} (x_i - y_i)^2 \quad (4.4)$$

$$NMSE(x, y) = \frac{MSE(x, y)}{MSE(x, 0)} = \frac{\|x - y\|_2^2}{\|x\|_2^2} \quad (4.5)$$

where y the value predicted by the model and x is the groundtruth GT. This gives a simple relation between NMSE and relative ℓ^2 error.

In addition, the **SSIM** (Structural Similarity Index Measure) metric was also calculated [24]. It determines the structural similarity index between two data, which is a value between -1 and +1. A value of +1 indicates that the two data are the same, while a value of -1 indicates that the data are very different. It is usually possible to report a value between 0 and 1 without changing the meaning of the metric, and this change is applied in the following paper. This metric was originally developed for two-dimensional (2D) images, but it can be applied to 1D time series by considering the signal as an $N_t * 1$ pixel image and using a 1D convolution window. The formulas that characterize it are as follows: The formulas that characterize it are as follows:

$$SSIM(x, y) = l(x, y) * c(x, y) * s(x, y) \quad (4.6)$$

where:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4.7}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{4.8}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{4.9}$$

$$C_1 = (K_1 L)^2 \tag{4.10}$$

$$C_2 = (K_2 L)^2 \tag{4.11}$$

$$C_3 = C_2/2 \tag{4.12}$$

C_1, C_2 and C_3 are used to avoid singularities when the denominators are too small; K_1 and K_2 are arbitrary constants that by default have a value of 0.01 and 0.03, respectively; finally L is the length of the signal. In any case, however, there is a more efficient and commonly used version of SSIM. It is called MSSIM and takes into account the mean and variance of the input data. Thus it is better able to capture smaller structural details of the data and is more robust to degrading factors such as noise or compression. Therefore, this makes it more reliable in the current case. Its formulation is as follows:

$$MSSIM = \frac{1}{N}\sum_{j=1}^{N} SSIM(x_j, y_j) \tag{4.13}$$

These formulas are not computed manually, but the optimized scikit-learn [25] library is called. In addition, the default values of hyperparameters K_1 and K_2 were changed to 0.001 and 0.003, respectively, to better fit the case under consideration. Furthermore, another metric computed is the **SNR** (Signal to Noise Ratio). Signal to noise ratio (S/N or SNR) is a measure used in science that compares the level of a desired signal to the level of background noise. SNR is usually expressed as a ratio, either in dB or as a percentage. A higher SNR means there is a stronger

signal and less background noise. The formula used is as follows:

$$SNR = \frac{\sum |(y_{pred} - y_{true})|}{\sum |y_{pred}|} \tag{4.14}$$

where y_true is the signal without noise and y_pred is the predicted signal. In addition, a variant of this formula has been developed that considers the possible presence of peak areas near a sinkhole. In this **custom SNR**, the indices found in the peak area of the signal and the indices found in the remaining spectrum are calculated. Then the maximum value of the signal in the peak area is extracted and the SNR is computed using the formula:

$$SNR = \frac{I - N_{mean}}{N_{std}} \tag{4.15}$$

where N_mean is the mean of the peak-free zone, N_std is the standard deviation of the peak-free zone, and finally I is maximum of the absolute value of the signal in the peak zone. In this way, if there is a peak in the area under consideration, I is its maximum value, otherwise if it is a peak area where there is a large depression in the neighborhood, then it is the maximum in absolute value of the area (thus also considering the negative peaks). Furthermore, an additional variance of SNR was computed. It is the **PSNR** (Peak Signal to Noise Ratio) and it is calculated using the scikit-learn library [25]. It is a measure of the quality of a reconstructed signal that checks the amount of error introduced. The higher the value of PSNR, the lower the error and thus the higher the quality of the signal. Generally, it is defined as the ratio of the maximum power of a signal to the noise power along the entire signal, but in this case it is easier to describe using the MSE with the following formula:

$$PSNR(y_{pred}, y_{true}) = 20 * \log_{10} \frac{MAX(y_{pred})}{\sqrt{MSE(y_{pred}, y_{true})}} \tag{4.16}$$

where $MAX(y_{pred})$ is the maximum value of the signal and MSE is computed as in the previous Formula 4.4.

Finally, the last metric that is computed is the **NMAE** (Normalized Mean Absolute Error). In statistics, the mean absolute error (MAE) is a measure of how close forecasts or predictions are to the eventual outcomes. The MAE is a linear score which means that all the individual differences are weighted equally in the average.It is also referred to as L1 loss. The MAE can be normalized by dividing it by the mean of the actual values, thus obtaining the NMAE. The NMAE is a good metric to use when comparing models because it is scaled to the range of the data. A model with a low NMAE is a good model. The NMAE is also robust to outliers because it is based on the absolute value of the error. In this case, it is computed using the SNR and the formula is as follows:

$$NMAE = \frac{1}{SNR} \tag{4.17}$$

where SNR is the Formula 4.14 previously defined.

4.2.2 Methods in Comparison

To corroborate the results that will be obtained, it is necessary to make a comparison with other algorithms that are already validated by the scientific community. To do this, five different algorithms were implemented and the same dataset that is passed to the neural network implemented in this work is submitted to them. Finally, all the results obtained in the test are exposed in the Chapter 6. Unfortunately, none of the following algorithms are based on neural networks. In fact, the neural networks exhibited in Chapter 3 were either not designed to work with all the features that are instead addressed in this paper or they fit datasets that do not match the currently available one. Therefore, it would have been complicated to fit very complex models to the current dataset. In any case, the most valid algorithms were chosen in terms of baseline correction. The following subsection will then discuss their features.

The first one is called **PSALSA** (Peaked Signal's Asymmetric Least Squares Algorithm) [16]. This is the same algorithm mentioned in the Section 3.1 of Chapther 3 on state-of-the-art analysis. Briefly recalling its characteristic, it is an adaptation of the ALS (Asymmetric Least Square) algorithm [26] designed specifically to work with signals filled with intense peaks, as in the case of the Raman spectrum.

The second algorithm is called **"Adaptation of Cubic P-Spline Based on Morphology"** [17]. Also this algorithm is the same algorithm mentioned in the Section 3.1 of the Chapther 3 on state-of-the-art analysis. Again briefly recalling its functioning, it aims to adopt curve-fitting algorithms already known, as in the case of algorithms making use of polynomials, to the case of the Raman Spectrum.

The third algorithm is called **IModPoly** (Improved Modified Polynomial (IModPoly) Baseline Algorithm). The method was developed by Zhao [27] and is similar to ModPoly [28]. Both are based on the method of Thresholding [29]. It is an iterative process that first fits the data using traditional least squares and then sets the next iteration's data fit as the minimum between the current data and the current fit. The classical ModPoly version uses least-squares fitting of a polynomial background with adaptive removal of peak regions from the fit, while the IModPoly version improves the peak removal scheme. To do this, IModPoly attempts to improve the algorithm for noisy data by including the standard deviation of the residual when Thresholding is performed.

The forth algorithm is based on the **Savitzky-Golay filter** [30]. It is a type of finite impulse response filter that is commonly used for data smoothing and noise reduction. The Savitzky-Golay filter is a linear filter and operates through a convolution process that fits successive subsets of points with a low-degree polynomial by the method of linear least squares. When the data points are equidistant, an analytical solution to the least squares equations can be found in the form of a single set of "convolution coefficients". Eventually, these coefficients can be applied to all subsets of data to provide estimates of the smoothed signal (or its derivatives).

A final comparison metric is the application of the **UnivariateSpline** [31]. It is a

function of Scipy library that allows fitting a spline $y = spl(x)$ of degree k to the given x, y data. As mentioned earlier during the explanation of the algorithm that makes use of P-splines in Subsection 3.1, splines are mathematical functions that are very suitable for adapting a set of points. An advantage of using univariate splines is that they can be used to interpolate data that are not necessarily uniformly spaced. This is possible because there are several interpolating polynomials, each best interpolating a small region around focal points. In addition, another advantage is that they can be easily evaluated at any point, even if the data points are not equidistant. The parameter "k" indicating the degree of univariate splines refers to the degree of polynomials. A high degree produces a better fit to more complicated functions, although this may introduce artifacts. In the current case it has been set to 5, the maximum degree. In addition, no smoothing condition has been specified. Finally, to iteratively improve the fitting process, the function is called recursively for a finite number of times.

The implementation of the first four algorithms was not written from scratch, but is available at the following GitHub link [32] which refers to the paper [15]. The last one, however, was implemented from scratch.

Chapter 5

Selected Methods And Their Implementation

The goal of this work is not to create a new method from scratch, but aims to implement and combine several already pre-existing excellent methods, whose effectiveness has been demonstrated countless times in different fields. This merging and adaptation of different methods was necessary to work optimally in the case of the Raman spectrum as well.

This paper is mainly based on two aspects: parallel convolutional neural networks (Subsection 5.2) and Residual Neural Networks (Subsection 5.3). In any case, however, both are based on the general concept of convolutional neural networks, which will be discussed for completeness (Subsection 5.1).

5.1 Convolutional Neural Networks

At the core of the entire project is one of the most classic, powerful and widely used neural networks, namely Convolutional Neural Networks. CNNs are neural networks that are designed to work with data that have a spatial structure. They are able to take as input different types of data, assign importance (learnable weights) to various aspects of the input, and then finally learn aspects of that input which are relevant

to the task's purpose. The pre-processing required in a ConvNet is much less than in other algorithms. Whereas in pre-CNN methods, filters were created from scratch and had to be designed to study all aspects of interest, ConvNets now have the ability to learn these features on their own with properly set up training. A ConvNet can optimally capture the spatial and temporal dependencies of n-dimensional input data through the application of relevant filters. This is achieved by inspiring the network of CNNs to the computational process of the human brain [33]: each node in the network (like each neuron) receives a stimulus as input and processes it, producing an output. It is then propagated to the nodes in the next layer. This process continues until the network termination is reached and an overall evaluation of the initial stimulus is carried out, extrapolating what is important or not. The first studies to create a computer process capable of implementing this concept date back to the 1980s and 1990s [34,35]. Since then, with the growing interest and use of such algorithms in many aspects surrounding everyday life, a number of more complex and powerful CNN architectures have emerged, such as: AlexNet [36], VGGNet [37], GoogLeNet [38], ResNet [39], and so on. In addition, ResNet architecture features are also included in this paper and will be discussed in Subsection 5.3. In any case, however, all these architectures are based on the same functioning, which will be explained in the following subsection.

5.1.1 How CNNs work

CNNs are mainly composed of an initial convolutional layer, which may be followed by other convolutional layers [40], and pooling layers. By increasing the number of convolutional layers, CNNs increase their complexity and, tendentially, their efficiency. Obviously, this is not always the case as it can lead to various problems, such as performance degradation and overfitting. In any case, in general, the first layers focus on simpler features, and as the input reaches the deeper layers of the CNN, it begins to recognize and learn more complex features.

As can be deduced from the network's name itself, the convolutional layer is the

core of a CNN and in it most of the computations that characterize this procedure take place. The layer is made up of a set of learnable filters, which are applied to the input to produce a set of outputs. They are a feature detector and have a similar structure to the input, although the size may vary and are generally smaller than the input. For each convolution step, the output is given by the sum of the kernel-weighted input values. In the end, all these multiplications are saved in an output component called a feature map. The number of feature maps is determined by the number of filters in the layer. Furthermore, it should be noted that the kernel values do not change during one epoch, but only before moving to the next. In fact, after this procedure, backward propagation and gradient descent phases occur and they change the kernel values based on an evaluation of the training trend. The mathematical Formula 5.1 of this discrete convolution is as follows:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] \tag{5.1}$$

where $f[n]$ is the input data e $g[n]$ is the kernel. In any case, for training to begin, the convolutional layer must necessarily be initialized and the following three hyperparameters must be defined:

- **The number of filters,** that affects the depth of the output.
- **The stride,** which is the distance traveled by the kernel on the input matrix each time it goes from one product to the next. It is usually 1 by default.
- **Padding,** which is a technique involving all values outside the input. It adds a constant value, usually zero. Padding can be used to make sure that the input data have the same size as the weights and biases of the network. This is important because mismatched sizes can cause errors in the network. It can also be used to add layers to the network or to increase the size of the training data, helping the network learn more effectively.

Furthermore, each convolution usually is provided with an activation function. It is a

mathematical function that is used to determine the output of a neural network. The function is used to map the input of the network to a desired output and it is also called a "transfer function". Activation functions are important because they allow the network to model non-linear relationships. Without them, the network would only be able to model linear relationships, and this would be a major limitation because many real-world relationships are non-linear. In addition, an activation function can even disable a node, that is, set the output to zero so that it has no effect on future nodes. This occurs when a node is not deemed useful for network learning purposes. There are also several types of activation functions, divided into linear and nonlinear. The most common are Sigmoid, Tanh, ReLU and LeakyReLU, where the first two are nonlinear and the last two linear. A graphical representation of how each of them relates the input it receives (expressed along the x-axis) into the output (expressed along the y-axis) can be seen in the following Table 5.1:

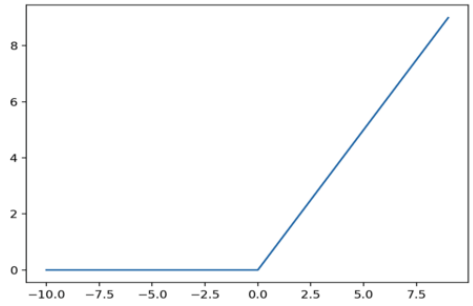
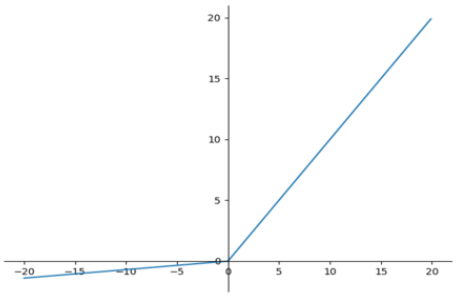
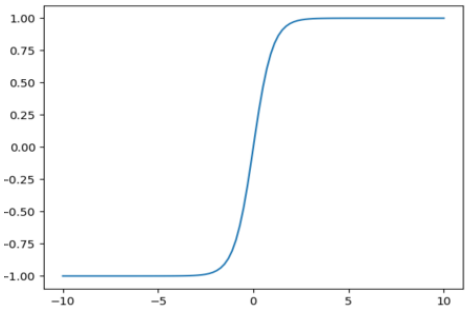
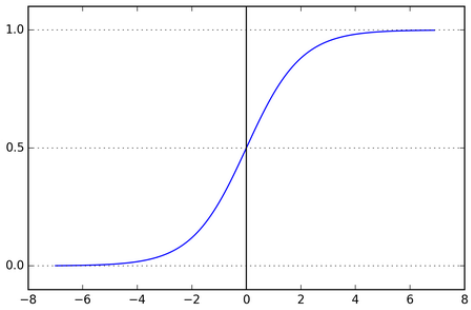
ReLU	**LeakyReLU**
	
Tanh	**Sigmoid**
	

Table 5.1. Table showing how each activation function relates input to output

The ReLU (Rectified Linear Unit) activation function was first mentioned in [41]

and is used in this work. It is essentially a thresholding function that returns 0 if the input is less than or equal to 0, and the input itself for all other inputs. It is Although common and very efficient, it is not without problems: for example, it is not zero differentiable or if the output is zero it does not allow training of a node. The latter fact can be restrictive since the node could become useful in later epochs if only trained. To avoid the fact that nodes are not trained when the ReLU returns zero, several alternatives such as the LeakyReLU have been developed. For example, the LeakyReLU does not return zero when the input is less than zero, but multiplies it by a very small value. In this way, the node can continue to be trained and not affect the network performance when it is not contributing any benefit. However, in general it is not possible to know a priori which activation function is best for the case. Only empirical experimentation can tell that.

Finally, convolutional layers are usually followed by pooling layers, which downsample feature maps to reduce computational load and the number of parameters. This aspect is also called dimensionality reduction. It works similarly to the convolutional layer kernel, but in this case it has no trainable weights. In fact, it applies an aggregation function to the values under consideration, thus defining the output array. There are two main types of function:

- **Maximum pooling:** It selects the maximum value among all those selected by the filter.
- **Average pooling:** It averages among all the values selected by the filter.

While losing a lot of information, pooling has a number of advantages such as reducing complexity, improving efficiency and limiting overfitting.

5.2 Parallel CNNs

Despite the countless advantages that CNNs have over more traditional methods, they present some limitations. For example, as the size of the network increases, several issues can arise such as, performance degradation, overfitting or training issues.

In addition, it becomes increasingly complicated to find optimal hyperparameters for the network, since the correlation that each parameter has with the obtained results becomes more intricate and complex to infer. In order to mitigate these issues, several solutions have been proposed, including CNN parallels. They are a type of network composed of multiple parallel and independent sub-nets. These sub-nets can be connected to each other in various ways depending on their use. They also have several advantages: first, they can speed up training since each parallel branch can run on different GPUs, but more importantly, deeper and more complex networks can be structured. In addition, each sub-network is customizable with different hyperparameters so that each one focuses on different features of the data passed as input, and then joins all the results at the end of the parallel structure. They are also often very useful when there is a multitude of heterogeneous data to analyze. As in the case of the work [42], for example, mobile devices gather a heterogeneous variety of information to process that are not easily usable. In order to mitigate this and other problems with similar characteristics, several studies have been carried out [42–44] to show how the parallel structuring of neural networks increases the computational efficiency and performance obtained. This whole series of work, then, is based on the realization of redundant architectures that, however, have differences in each redundancy. Thus, the idea is to parallelize the convolutional layers of a CNN by slightly varying the settings between them, so that each branch learns different features of the input. This latter technique has proven to be of fundamental importance in studying the noise that afflicts the Raman Spectrum in all its aspects. Regarding how it was implemented in this work, see Subsection 5.4.

5.3 Residual Neural Networks

Another aspect of great utility for efficient operation of the proposed model is the use of a residual architecture to define the structure of the model. As mentioned earlier, several variants of CNNs have been proposed over time in order to improve

the performance of neural networks in performing their tasks. A further step forward was proposed by the Microsoft Research division through the paper "Deep Residual Learning for Image Recognition" [39]. The idea behind the paper was that as a network grew in depth, it became increasingly difficult to train it. Therefore, a method was presented that allows it to reach greater depths without degrading the performance. The method is based on the fact that instead of structuring and optimizing different layers to best fit the overlying mapping, these layers are explicitly allowed to fit a residual mapping, and finally the original mapping is recast before providing the output. In this way, the task of the network is less complex, resulting in a lighter and more optimal model that more efficiently avoids overfitting. In order to accomplish this, a new neural network layer concept was introduced: the residual block.

5.3.1 Residual Block

A residual block [45] is a block composed of a series of weighted layers that have the ReLU activation function. Each weighted layer is generally composed of a convolutional layer and a batch normalization layer. The latter is used to avoid vanishing gradients, although it slows down training and is not always necessary, so it can be turned off. Input data are passed both as input to the residual block and summed at the end of it. The following Figure 5.1 represents this concept:

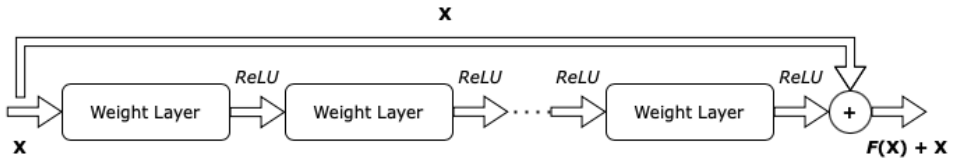


Figure 5.1. Representation of a residual block

In the above equation X represents the identity mapping, while F(X) represents the residual function. The reason this approach is proposed is because it is assumed that it is much easier to optimize a residual mapping than to optimize the original non-referenced mapping, even though both have a similar number of parameters,

i.e., weights. Moreover, if the identity mapping is optimal, it is easier to push the residual to zero than to fit an identity mapping to a stack of nonlinear levels.
Finally, the step by which input is passed both to the weighted layers and toward the output is called the skipped connections technique. Skip connections were introduced in literature even before residual networks. The study of this technique dates back to the 1990s through several papers that sought to validate its usefulness and the criteria for applying it efficiently [46, 47]. The reason for their development and use concerns the degradation of performance with increasing network depth. During the development of the ResNet [39], Batch Normalization layers were added to prevent the gradient from exploding or disappearing, but this addition did not lead to complete alleviation of the problem. In fact, some experiments have shown that the error remains high as depth increases. This phenomenon is often due to random initialization of weights that have an average close to zero, and this would not allow the deeper layers to learn. In order to avoid this, the concept of skip connections was developed and to pass data to the deeper layers and allow such training. The idea is that some branches can skip one or more layers, only to be rejoined later, either after being processed in different ways or simply without undergoing any changes [48]. but in this case the input skips the entire network and is reconnected at the output. In the case of ResNet, this re-connection is done through a sum, but there are other ways to implement it. For example, in the case of DenseNets [49], it concatenates the output feature maps of the layer with the next layer rather than with a summation as in ResNet. In this way, features learned from the previous levels can also be used in the deeper layers to solve a different problem from the previous one. In addition, it is possible to parameterize and make these connections conditional on the input through gating functions [50]. In this way, such connections are either working or not depending on the data configuration received. Obviously, this alternative is not mandatory. In fact, in the current thesis and in the project was presented in the paper [39], skip connections are parameter-free and not dependent on the input data, so they always remain functional.

5.4 Implementation

The residual neural network studied in this thesis, called "SRSdenoiser" for simplicity, is initially composed of a set of parallel branches whose number is decided by a hyper-parameter. The kernel size of the layers in each branch is computed using numpy's linspace function, which returns a set of equidistant values within a given interval. The interval chosen in the current case is between 5 and 88. Furthermore, each parallel branch can have a number of convolutional layers chosen through a hyper-parameter. In the current case, that hyper-parameter is set to 6. Each convolutional layer can be alternated by Batch Normalization layer, but in the actual case this setting has been turned off. Another feature of the network is the desire to keep the number of network parameters fixed as the size of the kernels changes, so the number of channels is computed based on the number of parameters and the size of the kernels. At the end of each parallel branch, a concatenation of the result between the previous branch and the next branch takes place. The last concatenation is then passed to a one-dimensional convolutional layer so as to return to a single channel dimension via a trained weighted average of the remaining channels. In addition, however, as the succession of concatenations increases the number of channels, a series of final convolutions can optionally be added that gradually halve the number of channels, as these convolutions are trained like the rest of the network. Each convolution inserted between the concatenation of parallel branches and the last hand-dimensional convolution halves the number of channels. This tends to improve the efficiency of the network. Finally, a subtraction is performed between the last one-dimensional convolutional layer and the input that, via a skip connection, has passed unchanged from the beginning to this point. It is also worth reporting that all involved convolutions have the ReLU activation function.

In order to better understand the structure of the model, it is possible to see in Figure 5.2 the graph of the network, where for simplicity only three parallel branches, two convolutional layers per parallel branch, and two convolutions between the last

concatenation of parallel branches and the last one-dimensional convolutional layer were included.

Meanwhile, as for the characteristics of network training, it is divided into two phases. Both are characterized by the same optimizer, i.e., Adam, but they differ on several fronts. In the first phase, the learning rate is set to 0.018 and the batch_size is set to 24. Both values are higher than in the second one. In addition, the loss used is a classical MSE (Mean Square Error), the Formula 4.4 that has been previously defined. In the second stage, the starting learning_rate is 0.005 and the batch_size is set to 16. As for the loss, it is a custom_MSE, obtained from the union of the MSE loss and the gradient loss. It is computed by the following formula:

$$LOSS = (1 - Grad\_weight) * MSE + Grad\_weight * Weight * GradLoss \quad (5.2)$$

$$GradLoss = \frac{|(D_{pred} - D_{true})|}{|(D_{true} + 1)|} \quad (5.3)$$

where D_{pred} is the derivative of the predictions, D_{true} is the derivative of corrected samples to compare with predictions, $Grad\_weight$ proportions the influence of MSE and the influence of $loss\_grad$ to 1, while $Weight$ is used to delete or to reduce the overall increase in loss when switching from the classical MSE to this option combined with loss_grad. It is not always possible to completely eliminate the increase in validation loss that occurs when changing the loss used, however, a decrease in the gap is still expected to also improve training performance when the change between the first and second phases occurs.

Finally, two components have been added to improve the training performance of the network. The first is the "Reduce Learning Rate On Plateau" function. It reduces the learning rate when a metric ceases to improve. In the current case it is based on validation loss. During the first phase the patience, i.e., the number of epochs without improvement in val_loss that is expected before reducing the learning rate, is set to 3, while in the second phase it is set to 4. In both cases, the learning rate is multiplied by a factor of 0.9. Instead, the second component is the "Early Stopping"

function. To prevent overfitting from occurring on the dataset, early stopping was included, which ceases each phase of training when a metric stops improving. It is based on validation loss too. During the first phase, patience, i.e., the number of epochs without improvement in val_loss that is waited before stopping the training phase, is set to 7, while in the second phase it is set to 9.

Finally, several experimental variants of this implementation were proposed in Chapter 6 regarding experiments and results.

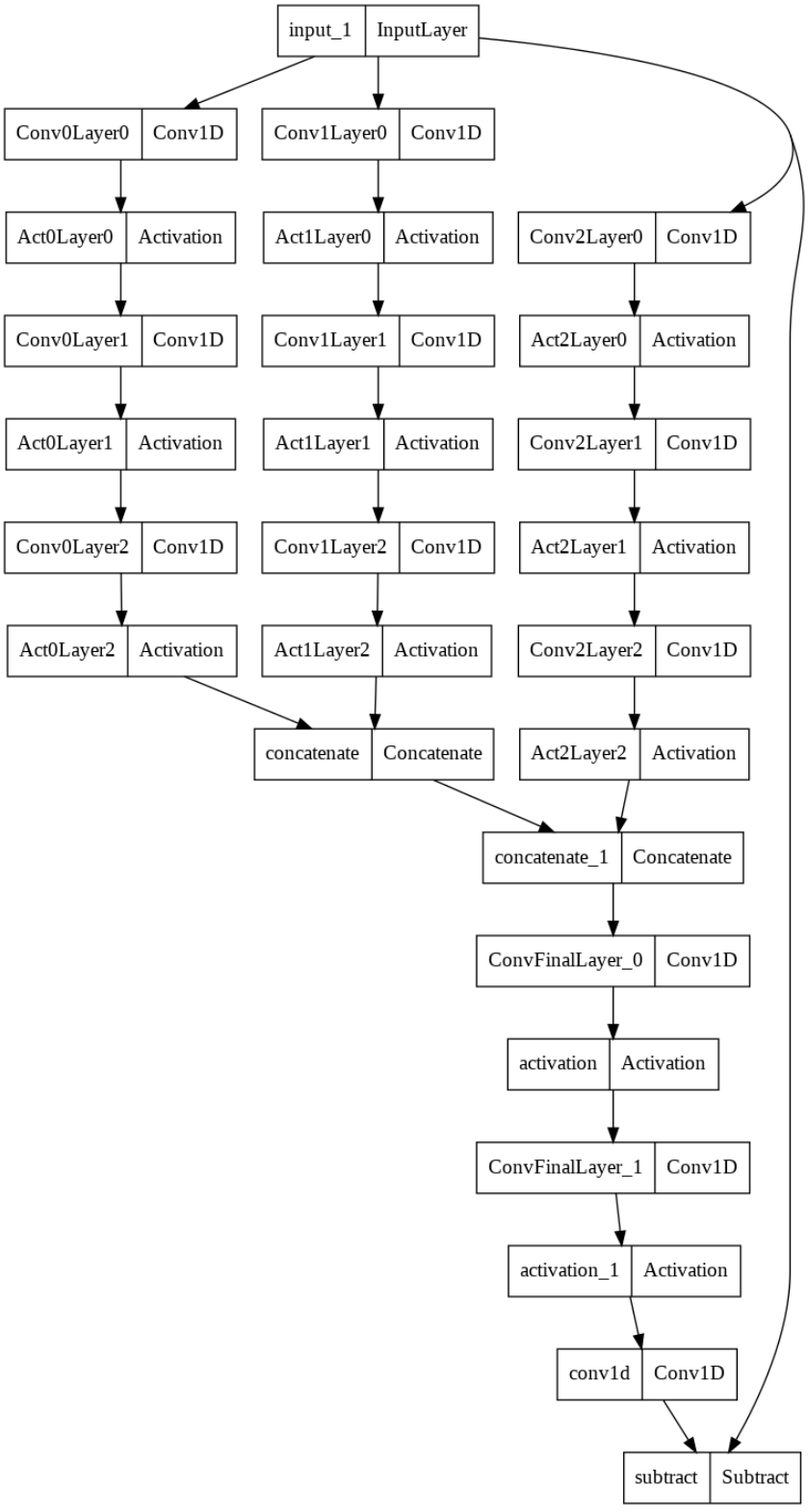


Figure 5.2. Structure of the model

Chapter 6

Experiments And Results

Considering the SRSdenoiser architecture discussed earlier, the aim is to understand if and what network characteristics may affect its performance. Furthermore, it is of interest to understand its generalizability with respect to datasets with varying noise on different scales, whose details were discussed in Chapter 4. Therefore, all experiments performed were done in order the optimized parameters of the neural network. The parametric complexity of the adopted algorithm required a wide range of tests and experiments to gradually reach the optimum both in terms of performance timing and quality of the results obtained.

Most of the more complex tests were carried out through the use of servers provided by INFN (National Institute of Nuclear Physics). Through them it was possible to delegate the computationally more demanding work to GPUs. In particular, the "NVIDIA Tesla V100 SXM2" GPU was used [51]. Built with a 12-nm manufacturing process, it features 32 GB of HBM2 memory coupled to the Tesla V100 SXM2 through a 4096-bit memory interface. Of particular note among its many features is the inclusion of 640 tensor cores that help improve the speed of machine learning applications. Additionally, when the computational load was less, the Colaboratory tool, or "Colab" for short, from Google Research was also used [52]. It is an online tool that allows one to write and execute python code. By making GPUs available, it can also be used to run machine learning algorithms. The GPUs made available are

either the NVIDIA K80 or the NVIDIA T4. The user cannot select which one to use and it is assigned based on availability. They have 12GB and 16GB of RAM memory, respectively. The problem with this tool is that its use is limited in performance and time, so it has been used only for tests that are computationally light and do not require excessive execution time.

This chapter focuses on the experiments, highlighting the different approaches taken, the reasons why they are adopted, their settings and the results obtained, finally comparing advantages and disadvantages. In order to understand whether the results achieved by the neural network are actual improvements of the state of the art, the following experiments will be compared with several algorithms that have already proven good efficiency in denoising the Raman spectrum. They do not make use of neural networks and their formulations have already been addressed in Section 4.2.2.

6.1 Losses Analysis

An initial test on the network focused on identifying the most appropriate losses to be used in the two training phases. In general, regression losses are used in the first phase. They succeed in correcting both the noise along the signal and most of the baseline distortion. On the other hand, the losses used in the second stage are intended both to refine the baseline correction, but more importantly to make the point-by-point fluctuations smoother. In fact, it is important to point out that noise cleaning operations are not factorizable in the case of complex line shapes, i.e., it is not possible to remove all of one type of noise first, and then remove all of the second type later on. Instead, the most effective strategy is to act on both types of noise first with a reconstruction term and then reinforce the smoothing action with a second phase. Specifically, the following losses have been tested in the first phase:

- **MSE** (Mean Squared Error): it is widely used as a loss in regressions because it is a strong measure of how close a predicted value is to the actual value. In fact, it computes the difference between predicted and actual values. It

is easy to calculate and is also particularly robust to outliers as it does not significantly change the final result based on their presence. Its Equation 4.4 has already been discussed previously.

- **MAE** (Mean Absolute Error): It is widely used because it is similar to the previous loss, but is more robust to outliers than MSE. It does not square the differences between predicted and actual values like the MSE loss, but considers the absolute difference between predicted and actual values. This makes this metric less influenced by outliers, as the final result is not significantly affected by them. Therefore it is very suitable for the case at hand. Its formulation is the following:

$$MAE(x, y) = \frac{1}{N} \sum_{i=1}^{N} |x_i - y_i| \tag{6.1}$$

- **RSS** (Residual Sum Of Squares): It measures how much the predicted values deviate from the actual values with a formulation very similar to the MSE loss. It is robust to outliers too because it penalizes them greatly. This means that it is more likely to produce a model that fits the data better. Its formulation is the following:

$$RSS(x, y) = \frac{\sum_{i=1}^{N} (x_i - y_i)^2}{2} \tag{6.2}$$

  Division by 2 is useful during the computation of gradient descent because it is simplified with the 2 that is obtained from the derivative of the square.

As for the second phase, both previous losses and new ones are used. The use of the previous ones is justified by the fact that the hyperparameters of the network, such as learning rate, batch size and others, are different in the second phase and thus could lead to different refinement than in the first phase. In addition, other losses that foreground the analysis of microscopic variations in the signal are also considered. They are:

- **Custom_MSE**: Instead of using the previous MSE, it seemed more meaningful to create a customized version that simultaneously weights both the classical

MSE and the gradient loss. In fact, the latter can provide a general evaluation of the deviation between the predicted signal and the corrected one through the first derivative of both. Such information can be useful in refining the general correctness of denoising. Specifically, the gradient helps smooth point-by-point fluctuations, while the reconstruction term acts on both types of noise. Its formulation is as follows:

$$\begin{aligned} Custom\_MSE(x,y) = (1 - Grad\_Weight) * MSELoss \\ + Grad\_Weight * Weight * GradLoss \end{aligned} \tag{6.3}$$

Where GradLoss was previously defined by the Equation 5.3, Grad_weight balances the influence of MSE and the influence of gradient loss GradLoss to 1, while Weight serves to reduce the overall increase in loss when switching from classical MSE to this option combined with GradLoss.

- **SNR** (Signal to Noise Ratio): it is used to measure the strength of a signal relative to the background noise in it. Its Equation 4.15 has already been discussed previously.

- **PSNR** (Peak Signal to Noise Ratio): this metric is very similar to the previous one, but the main difference between the two is that SNR measures signal strength with respect to the noise, while PSNR measures signal accuracy versus the maximum value of the corrected signal. Its Equation 4.16 has already been discussed previously. In addition, it is necessary to mark the fact that PSNR can take both positive and negative values, and this is deleterious to the computation of the loss, which is positive and set to minimize itself. In order to avoid a negative loss that tends to minimize, the absolute value of PSNR is computed.

Finally, the other hyperparameters of the network have been fixed to the best values obtained for similar datasets in a previous work, in order to perform a fair comparison between the various losses. They are shown in Table 6.1.

Hyperparameter	Value
Maximum Kernel Size	88
Number of Parallel Branch	21
Number of Convolutions per Parallel Branch	3
Number of Convolutions after Branch Concatenation	4
Total Number of Parameters in the Parallel Branches	10000
Batch Size - Phase 1	32
Learning Rate - Phase 1	0.0012
Maximum Number of Epochs - Phase 1	80
Batch Size - Phase 2	24
Learning Rate - Phase 2	0.0006
Maximum Number of Epochs - Phase 2	80
Weight for Custom_MSE - Phase 2	0.05
Grad_Weight for Custom_MSE - Phase 2	0.75

Table 6.1. Table of fixed hyperparameters

The results of these trainings present a large difference between some of the losses used. As visible in Figure 6.1, the use of SNR or RSS as a loss in the second phase of training show poor results compared to all other combinations. In contrast, the use of PSNR or the MAE allowed us to obtain better results, but not as good as those obtained in the case of custom MSE loss. To support these observations, a comparison of the values of all validation losses is shown in the Table 6.2. It is ordered from the largest value to the smallest one obtained in the validation loss. Therefore, the best loss in the second phase is the Custom_MSE. However, regarding the first phase, it is difficult to determine which loss is better, since the results obtained are all similar. This is probably due to the fact that the second loss redirects and corrects the training in the right direction, regardless of the results achieved with the loss previously used.

In order to determine which is the best combination, it is necessary to observe the results obtained in the first part of training, thus before the introduction of the second phase. It may be useful to conduct both qualitative and quantitative analysis. Looking at the qualitative Table 6.3, it can be observed that the behaviors of the various neural networks on some samples are all almost identical. In this table it is possible to see two samples from the Mixed Noise dataset that have been cleaned

of noise. Looking at a macroscopic comparison that also includes the noisy signal, the cleaned up signal seems to overlap the actual signal almost perfectly. On the other hand, analyzing in detail only the predicted and current signals, they are not exactly the same, but all cases show very similar denoising with slight differences. For example, when using the Custom_MSE loss in the second phase, the first part of the signal in the second sample has less distortion, but at the same time the third peak is detected with less precision than in the other cases. Furthermore, given the fact that the network is not yet optimized, it is not possible to use these characteristics as a discriminant to choose the best loss to use in the second phase. This implies that a more technical analysis of the results obtained is required, and it is reported in Table 6.4. This table shows both precision results regarding peak detection using a color histogram and Whisker plots showing the values obtained with the evaluation metrics. Of the last mentioned, no single value is reported, but a graph showing the average value obtained among all samples, which is expressed in the Whisker plots, where the mean value of each metric for a given algorithm is indicated by a yellow vertial line, and the corresponding quartiles and standard deviations by the extension of the box and by the error bars, respectively. Looking at the figure that simply measures peak detection accuracy, it would seem that the model that uses RSS as the loss of the first stage is the best. However, looking at all the metrics together, performances of the different losses are similar. In fact, according to the SSIM metric, which is a measure of the similarity of the two signals, the best model is the one with the MSE. In contrast, the NMAE metric, which measures the normalized mean absolute error, indicates that the model making use of the loss RSS is slightly better, but looking at the mean value and range, the RSS loss and the MSE loss have an almost identical result. Similarly, the same tests have been conducted on all types of datasets in order to understand whether losses behaved differently according to the type of noise features. In each of them, the metrics alternate in determining which one between the RSS loss and the MSE loss is the best, while the MAE is always worse, albeit by a small amount. This outcome

of extreme closeness of the results obtained is not so surprising. The three losses used in the first stage have similar characteristics. In fact, the MSE loss and the RSS loss both measure the mean square difference between the predicted output and the true output, and their distinction lies in the fact that the MSE computes the mean of squared errors, while the RSS is based on the sum of squared errors. The difference between the MSE and the MAE, however, is that the MSE measures the absolute difference and not the squared difference. This means that the MSE will be more sensitive to large differences than the MAE, since the squared difference of the largest errors will be greater than the absolute difference. Therefore, in order to understand which loss is better, it is necessary to analyze the validation loss trend during these trainings. This analysis is shown in Table 6.5. It includes both the first and second phases divided by a vertical dotted line, and this is of utmost importance because the study of the change between the two phases is what will allow the final conclusions to be made. What can be learned from that table is that when the model uses RSS loss during the first phase, at the transition between the first and second loss it goes from a value of the validation loss in the magnitude order of 10^{-1} to a value of order 10^{-5}. Similarly happens when the loss is the MAE. In fact, it goes from a value of the validation loss in the magnitude order of 10^{-2} to a value of order 10^{-5}. Only the MSE loss maintains the same order of magnitude when the transition to the next Custom_MSE loss occurs. However, as mentioned in the previous chapters, during the transition between the first and second phases, an attempt is made to have a change in loss that implies an improvement, thus a decrease of the loss, or at most a worsening that is as small as possible. An order of magnitude variation as large as that of MAE and RSS losses may be excessive, while the one of MSE loss is an improvement proportioned to the trend in progress previously. Since even the previous metrics did not determine which of the MSE and RSS losses is the best, it seemed more consistent to use the loss MSE in the first phase to have a smoother transition to the second phase.

Finally, looking at the metrics reported in Table 6.4, one last observation can be

denoted: some classical algorithms seem to perform better than neural networks, both in terms of precision and metrics. Obviously, this does not come unexpected since the hyperparameters have still to be optimized as detailed in subsequent subsections.

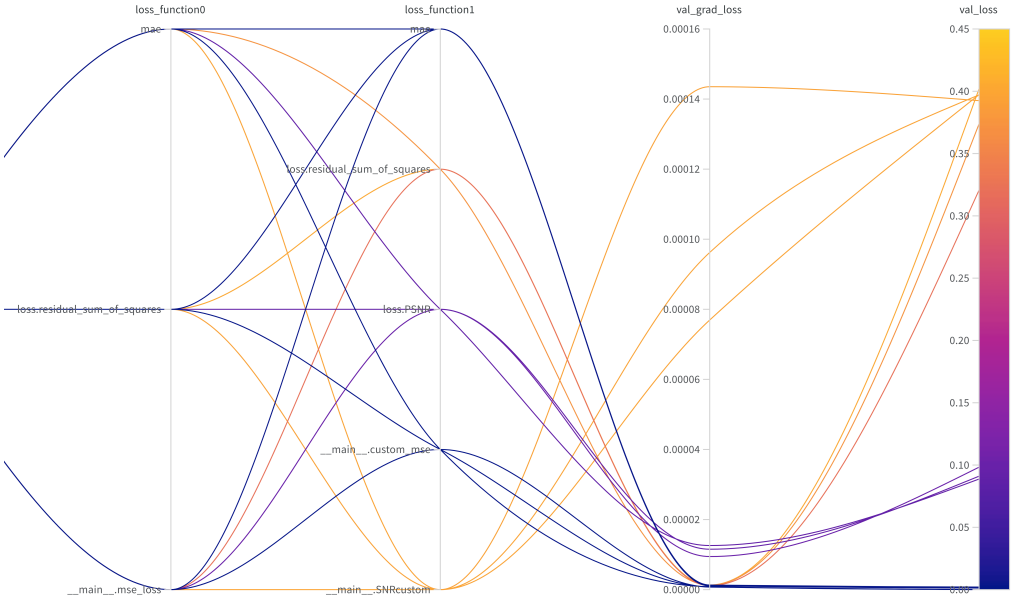


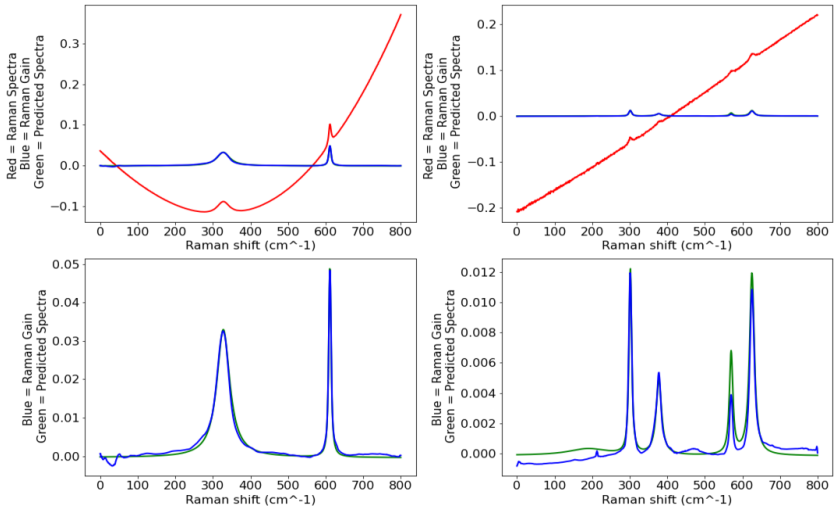
Figure 6.1. Graphical comparison of the Results Obtained from the Losses

Loss Phase 1	Loss Phase 2	Validation Loss
RSS	RSS	$4.020 \cdot 10^{-1}$
MSE	SNR	$3.973 \cdot 10^{-1}$
MAE	SNR	$3.971 \cdot 10^{-1}$
RSS	SNR	$3.925 \cdot 10^{-1}$
MAE	RSS	$3.738 \cdot 10^{-1}$
MSE	RSS	$3.207 \cdot 10^{-1}$
MSE	PSNR	$9.831 \cdot 10^{-2}$
RSS	PSNR	$9.107 \cdot 10^{-2}$
MAE	PSNR	$8.877 \cdot 10^{-2}$
MSE	MAE	$2.064 \cdot 10^{-3}$
RSS	MAE	$1.662 \cdot 10^{-3}$
MAE	MAE	$1.287 \cdot 10^{-3}$
MSE	Custom_MSE	$4.103 \cdot 10^{-6}$
RSS	Custom_MSE	$3.601 \cdot 10^{-6}$
MAE	Custom_MSE	$3.390 \cdot 10^{-6}$

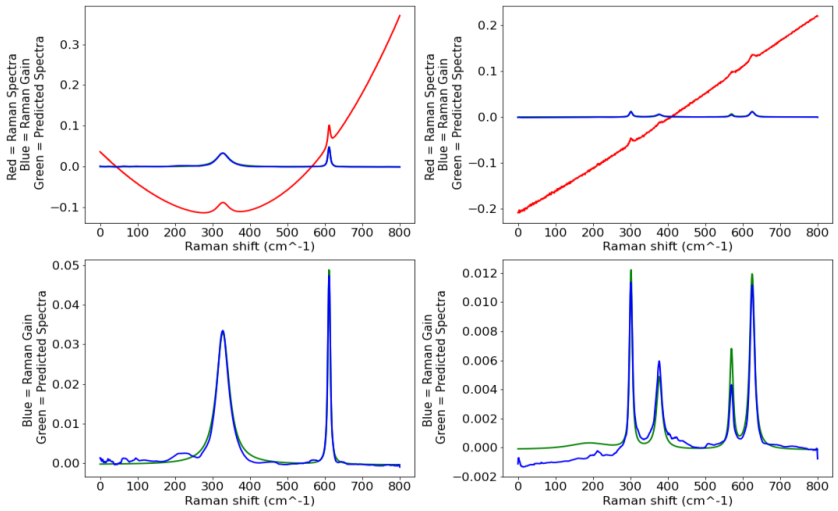
Table 6.2. Table Comparing of Validation Losses Values

Results Obtained with the Mixed Noise Dataset

First Loss MSE and Second Loss Custom_MSE



First Loss RSS and Second Loss Custom_MSE



First Loss MAE and Second Loss Custom_MSE

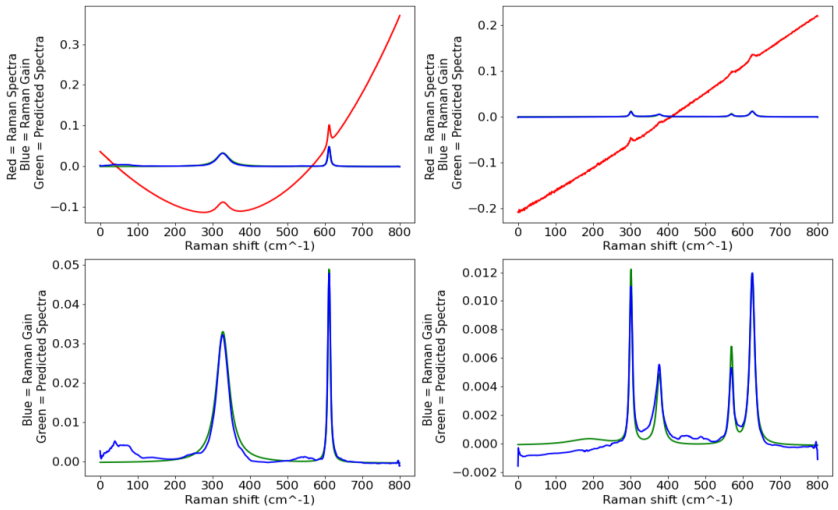


Table 6.3. Table of the graphical analysis of the results obtained in the first phase

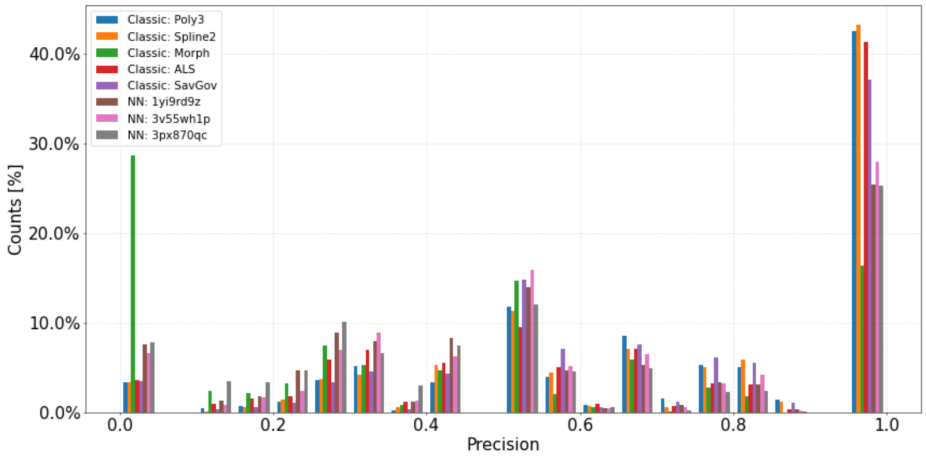
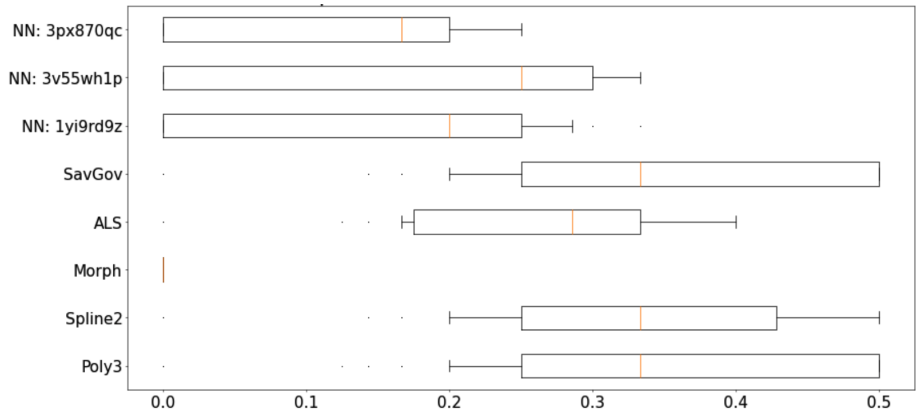
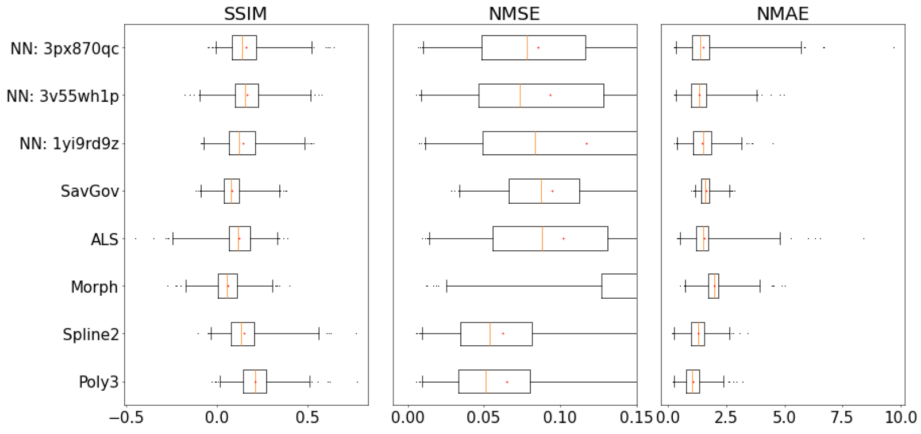
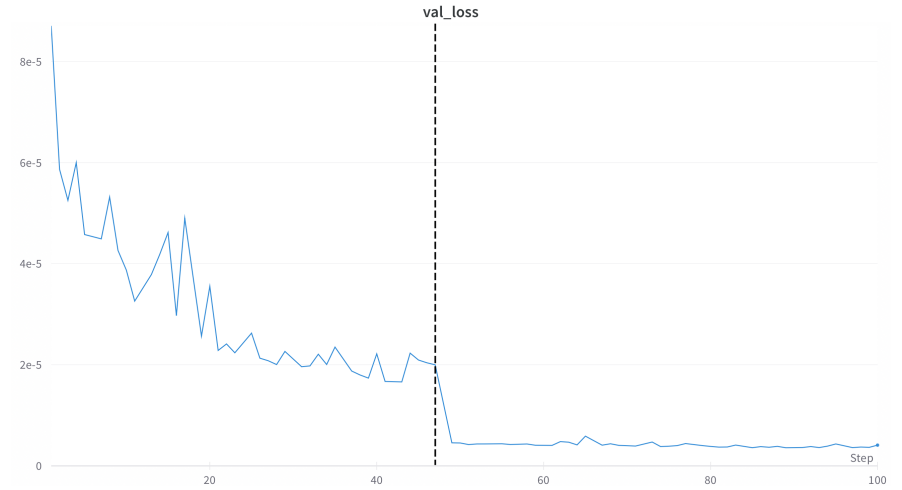
MSE Loss = NN:1yi9rd9z RSS Loss = NN:3v55wh1p MAE Loss = NN:3px870qc
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


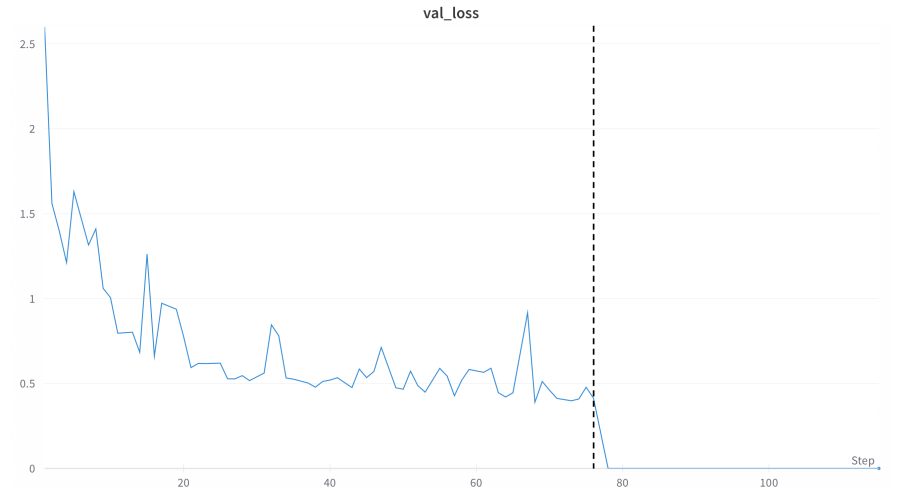
Table 6.4. Metric Analysis of the Results Obtained in the First Phase

The second phase loss is always Custom_MSE

First loss: MSE



First loss: RSS



First loss: MAE

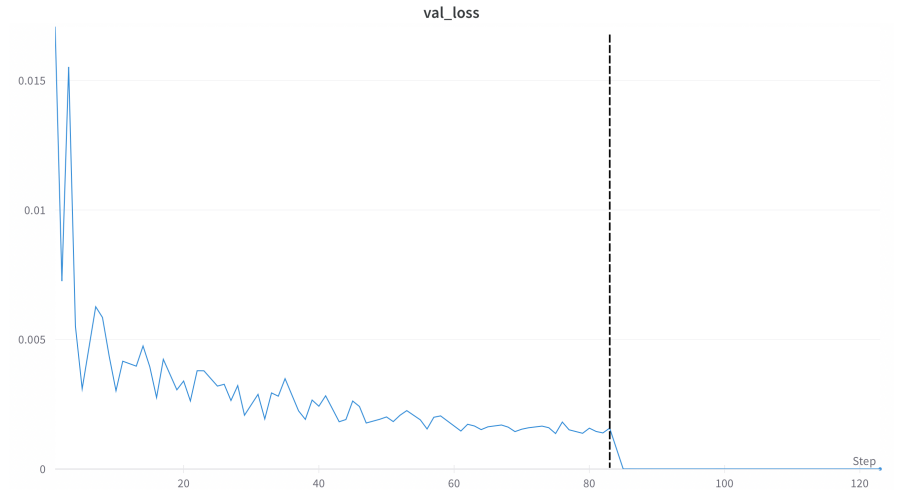


Table 6.5. Comparison of the validation loss trend for different training loss during the first phase of training

6.2 Model Optimization

A fundamental part of the work involved in creating a neural network is its optimization. Although neural networks are able to learn what they have to do more autonomously, optimal initialization results in achieving better performances. Moreover, in the current case, given the presence of several datasets with different types of noise, it is even more relevant to understand whether the datasets can be trained by a network that has the same hyperparameters or whether it is necessary to create different models. In order to better optimize the network, the work is divided into steps, where some hyperparameters are fixed while others have been left variant. Thus, previously fixed values are allowed to vary while previously variable parameters are fixed with the found optimal values. Some variations are discrete and listed, while others are randomly chosen within the reported range. A quite relevant factor is the maximum number of epochs of the first and second phases. In fact, during these tests, trainings did not proceed for a large number of epochs. This is due to the fact that each complete training can take from 15 to 120 minutes depending on some hyperparameters that define the size of the model, such as the number of parameters, the number of parallel branches and so on. A very interesting feature of the training is that in both the first and second stages, the number of initial epochs that produce a large change in validation loss is small and usually less than 10. In fact, subsequent iterations only serve to refine the result. Therefore, in order to speed up the decision-making process regarding the best hyperparameters, it has been decided to reduce the maximum number of epochs, without affecting the accuracy of the analysis. Generally, 30 to 50 training runs were performed for each dataset in order to determine which is the best combination of hyperparameters from each. Obviously, this procedure has been adopted only when the number of possible values of a given tested hyperparameter is large. On the other hand, when there are few tests to be performed, training has been completed with the maximum number of epochs required.

Therefore, the optimizations have been divided into the following steps: the first step focuses on the study of the ideal network size and is covered in Subsection 6.2.1; the second step focuses on the study of hyperparameters related to network learning and is covered in Subsection 6.2.2; in Step 3, how to smooth the step between the change of losses is studied and is covered in Subsection 6.2.3; in Step 4, data pre-processing is studied, and the results are reported in the Subsection 6.2.4; finally in the last step, covered in Subsection 6.2.5, studies on some hyperparameters related to network size are investigated in depth.

6.2.1 Step 1: Network Structure

The most important hyperparameters related to the network structure are the maximum kernel size, the number of parallel branches and the total number of hyperparameters. However, not all the hyperparameters defining the characteristics of the network structure have been addressed in this stage. In fact, the number of parallel branches and the number of convolutions at the end of parallel branches have been analyzed in the fifth and final stage. This is due to the fact that these hyperparameters seemed to have a greater influence on the performance of the network, so their optimization will be discussed separately in Subsection 6.2.5. Therefore, the hyperparameters are fixed for the tests presented in this subsection are as follows:

Fixed Hyperparameter	**Value**
Number of Parallel Branch	21
Number of Convolutions after Branch Concatenation	0
Weight for Custom_MSE - Phase 2	0.05
Grad_Weight for Custom_MSE - Phase 2	0.75
Maximum Number of Epochs - Phase 1	10
Maximum Number of Epochs - Phase 2	10
Batch Size - Phase 1	32
Batch Size - Phase 2	32
Learning Rate - Phase 1	0.0012
Learning Rate - Phase 2	0.0006

Table 6.6. Table of Fixed Hyperparameters

Instead, for what concerns the hyperparameters that vary in this subsection, they are defined as follows:

Variable Hyperparameter	Range of Values
Maximum Kernel Size	76, 88, 100, 122
Number of Convolutions per Parallel Branch	3, 6, 8, 10
Total Number of Parameters in the Parallel Branches	From 8000 to 13500

Table 6.7. Table of Tested Values For Hyperparameters

Figure 6.2, Figure 6.3 and Figure 6.4 show the results of the trainings done with the different combinations of hyperparameters on the Mixed Noise, Low Noise and High Noise datasets respectively. In the case of the Mixed Noise dataset, shown in Figure 6.2, the various obtained results are fairly scattered within the range between the maximum and minimum values of the validation losses obtained. The difference between the latter two values is only about $3 * 10^{-5}$, so it means that all of them obtain almost the same result at the validation loss level. Instead, regarding the validation loss of the gradient, they are mainly concentrated around two values, i.e., a good part of the results are between $8 * 10^{-5}$ and $9 * 10^{-5}$, while the best ones deviate and reach values around $2 * 10^{-5}$. This means that in general almost all combinations are equivalent, although some have slightly better characteristics. The combination of the hyperparameters that led to the two best results and are shown in Table 6.8. Regarding the results reported in the case of the Low Noise dataset, the difference between the maximum value the minimum value obtained in the validation gradient loss is only $1 * 10^{-6}$ and in the validation loss about $1.5 * 10^{-5}$. This means that the results are all more or less similar. Instead, for what concerns the results obtained in the High Noise dataset, the values themselves are the lowest and the difference between the maximum value and the minimum value obtained in the validation loss is just $4.5 * 10^{-6}$. Even the difference between the maximum value and the minimum value of the validation gradient loss is very low and it is only $2 * 10^{-5}$. Therefore, this means that, even though the dataset is the one with the highest amount of noise, the network managed to clean it up really well.

In any case, in order to determine which is the best combination of hyperparameters, it is necessary to carry out some analysis of comparison and averaging among all the hyperparameters that obtained the best result. This procedure is necessary because there are no single common value for all cases, even though some of them are repeated in multiple tests. Therefore, a first value that can be analyzed concerns the convolution number within each parallel branch. In most cases it corresponds to 6. Since it is even present in the best tests obtained for the High Noise dataset, which has the largest amount of noise to be eliminated, it is the best choice for such a hyperparameter. Instead, with regard to the sum of the parameter number of convolutional layers in each branch, it presents different values from test to test. The case of the High Noise dataset is the only one that presents 10000 as the optimal value in both tests. In the other datasets this value takes either a value close to it or much smaller or much larger. In the end, therefore, the most logical choice is to be 10000. This is due to the fact that it is the optimal value of the dataset with the large amount of noise, i.e. the High Noise dataset, and in addition in the other two datasets the number of optimal parameters in either one test of each is very close to it. Therefore, this is the best compromise of all cases. Finally, regarding the maximum kernel size, the situation seems to be easier to analyze. Indeed, no matter whether one test for each dataset reports a value smaller or larger than 100, because there is at least one test in each dataset that presents 100 as the optimal value for this hyperparameter. Therefore, this makes the choice of such a value obvious. Therefore, the best results are:

- **Nparam**, which is the maximum number of parameters in the parallel branches, **equal to 10000**;
- **max_ker**, that is, the maximum number of kernels, **equal to 100**;
- **Layers**, i.e., the number convolutions in the parallel branches, **equal to 6**.

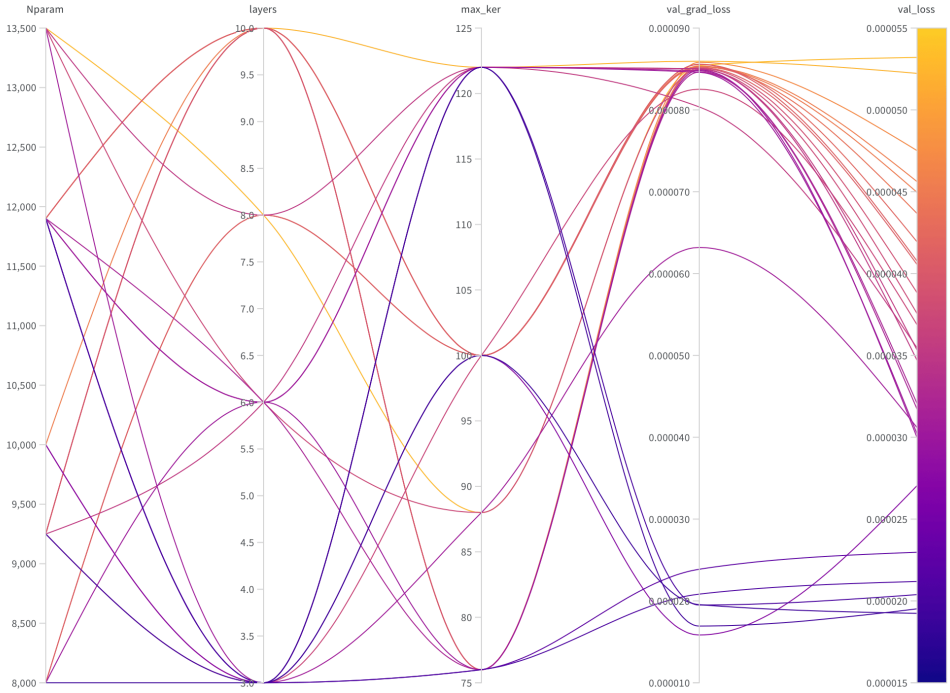


Figure 6.2. First Optimization on the Mixed Noise Dataset

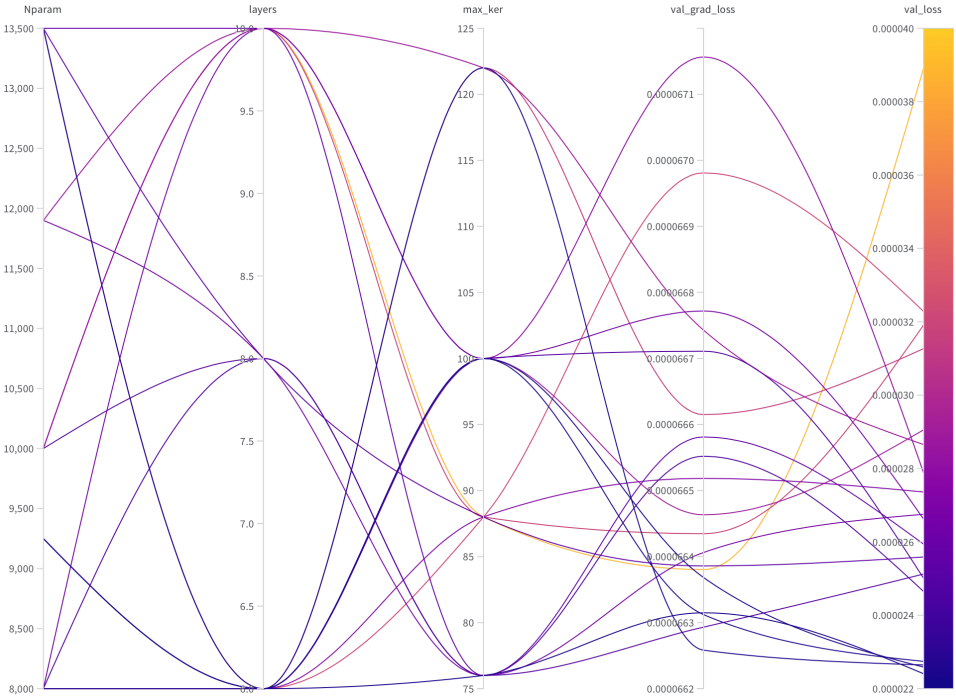


Figure 6.3. First Optimization on the Low Noise Dataset

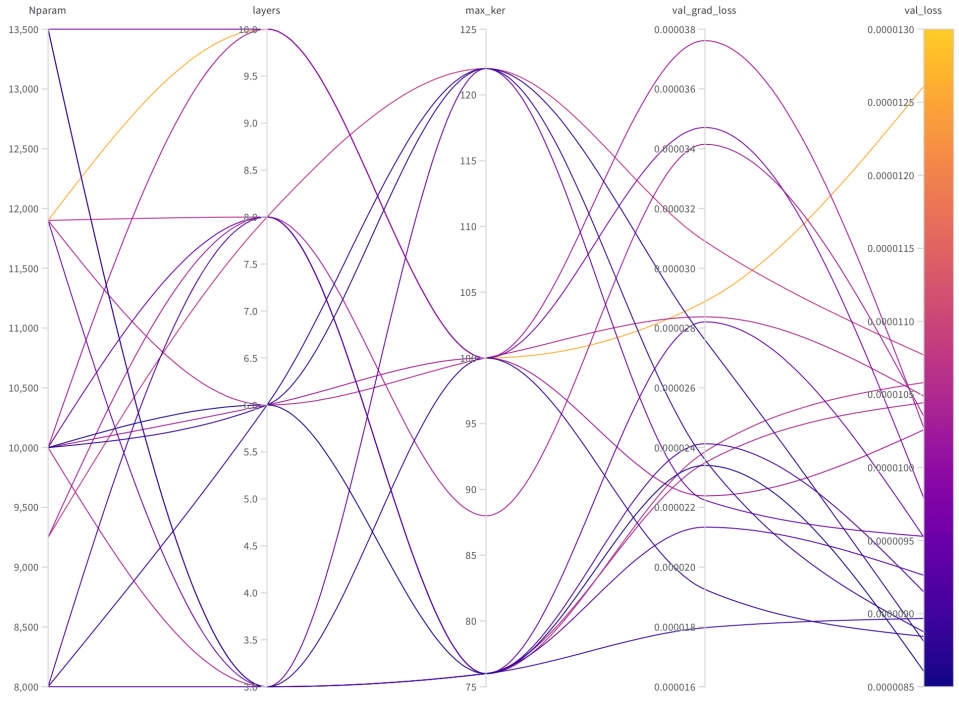


Figure 6.4. First Optimization on the High Noise Dataset

Test N.1:	**Mixed Noise**		**Test N.2:**
Nparam	8000	Nparam	9250
layers	3	layers	3
max_ker	100	max_ker	122
val_grad_loss	$1,955 \cdot 10^{-5}$	val_grad_loss	$1,693 \cdot 10^{-5}$
val_loss	$1,924 \cdot 10^{-5}$	val_loss	$1,952 \cdot 10^{-5}$
Test N.1:	**Low Noise**		**Test N.2:**
Nparam	13500	Nparam	9250
layers	6	layers	6
max_ker	76	max_ker	100
val_grad_loss	$6,631 \cdot 10^{-5}$	val_grad_loss	$6,637 \cdot 10^{-5}$
val_loss	$2,241 \cdot 10^{-5}$	val_loss	$2,259 \cdot 10^{-5}$
Test N.1:	**High Noise**		**Test N.2:**
Nparam	10000	Nparam	10000
layers	6	layers	6
max_ker	100	max_ker	122
val_grad_loss	$2,240 \cdot 10^{-5}$	val_grad_loss	$2,760 \cdot 10^{-5}$
val_loss	$8,605 \cdot 10^{-6}$	val_loss	$8,809 \cdot 10^{-6}$

Table 6.8. Table of the Best Results

6.2.2 Step 2: Network Learning

Once the general architecture of model has been optimized, it is possible to build on them to carry out the subsequent tests. Specifically, in this subsection, features that influence the learning the model will be analyzed, such as the learning rate and the amount of samples handled in each epoch step, handled by the batch size. The variation of these two hyperparameters is defined as follows:

Variable Hyperparameter	Range of Values
Batch Size - Phase 1	16, 24, 32, 64
Batch Size - Phase 2	8, 16, 24, 32
Learning Rate - Phase 1	From 0.0008 to 0.0018
Learning Rate - Phase 2	From 0.0001 to 0.0010

Table 6.9. Table of Tested Values For Hyperparameters

Instead, for what concerns the hyperparameters that don't vary in this subsection, they are defined as follows:

Fixed Hyperparameter	Value
Maximum Kernel Size	100
Number of Convolutions per Parallel Branch	6
Total Number of Parameters in the Parallel Branches	10000
Number of Parallel Branch	21
Number of Convolutions after Branch Concatenation	0
Weight for Custom_MSE - Phase 2	0.05
Grad_Weight for Custom_MSE - Phase 2	0.75
Maximum Number of Epochs - Phase 1	10
Maximum Number of Epochs - Phase 2	10

Table 6.10. Table of Fixed Hyperparameters

Figure 6.5, Figure 6.6 and Figure 6.7 show the results of the trainings done with the different combinations of hyperparameters on the Mixed Noise, Low Noise and High Noise datasets respectively. In the case of the Mixed Noise dataset treated in Figure 6.5, the difference between the minimum and maximum validation loss is only $2 * 10^{-5}$, while the difference between the maximum and minimum validation gradient loss is only $1 * 10^{-7}$. This indicates that the results obtained are similar with

almost all combinations. The Low Noise dataset is shown in Figure 6.6. Both the variation of validation loss and validation gradient loss are equal. The two datasets therefore vary similarly, emphasizing that there is not much difference in training a model that has to perform denoising on signals with few noise along the whole signal and with slight difference in amplitude between baseline and peak peaking. In the end the best performing model is obtained with the High Noise dataset and it is treated in Figure 6.7. In this case, the validation loss varies by $4 * 10^{-6}$. In order to determine which are the best combination of hyperparameters, the two best cases of each dataset are given in Table 6.11. Tests done on the same dataset report very similar metrics to each other. In general, either a value of 32 or smaller is fine as the batch size of the first stage. However, since the best results have been obtained with 32, it has been decided to take that value as optimal. Instead, with regard to the batch size of the second phase, there is a great predominance of value 32, so again it has been selected as the best. On the other hand, with regard to the learning rate, the situation is slightly more complex. This value is actually only indicative because the network and also has the additional function that reduces learning rate when a plateau occurs. This means that the important thing is to choose a value that is neither too high, causing the correction to occur too soon, nor too low, so as not to slow down the achievement of the optimum. A good approach therefore, is to select an average value between the two. In the case of the learning rate of phase 1, it ranges between a minimum of $1.1 * 10^{-3}$ and a maximum of $1.5 * 10^{-3}$, thus its average is equals to $1.4 * 10^{-3}$. Carrying out the same reasoning for the second stage, the learning rate of the latter is $3.5 * 10^{-4}$. Therefore, the best results are:

- **Phase 1 learning rate,** that is, the rate that controls the amount of adjustment of a neural network's weights in response to the estimated error at each training cycle, **equal to** $1.4 * 10^{-3}$;
- **Phase 2 learning rate equal to** $3.5 * 10^{-4}$;
- **Phase 1 batch size,** that is, the number of samples used in one iteration of

training the neural network, **equal to 32**;

- **Phase 2 batch size equal to 32**.

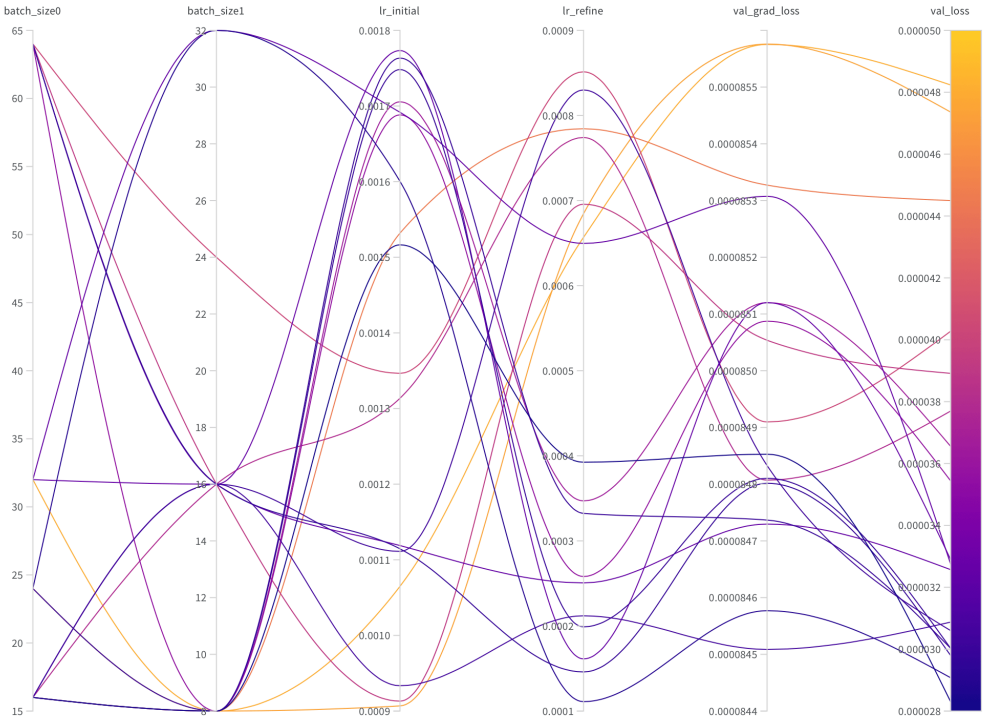


Figure 6.5. Second Optimization on the Mixed Noise Dataset

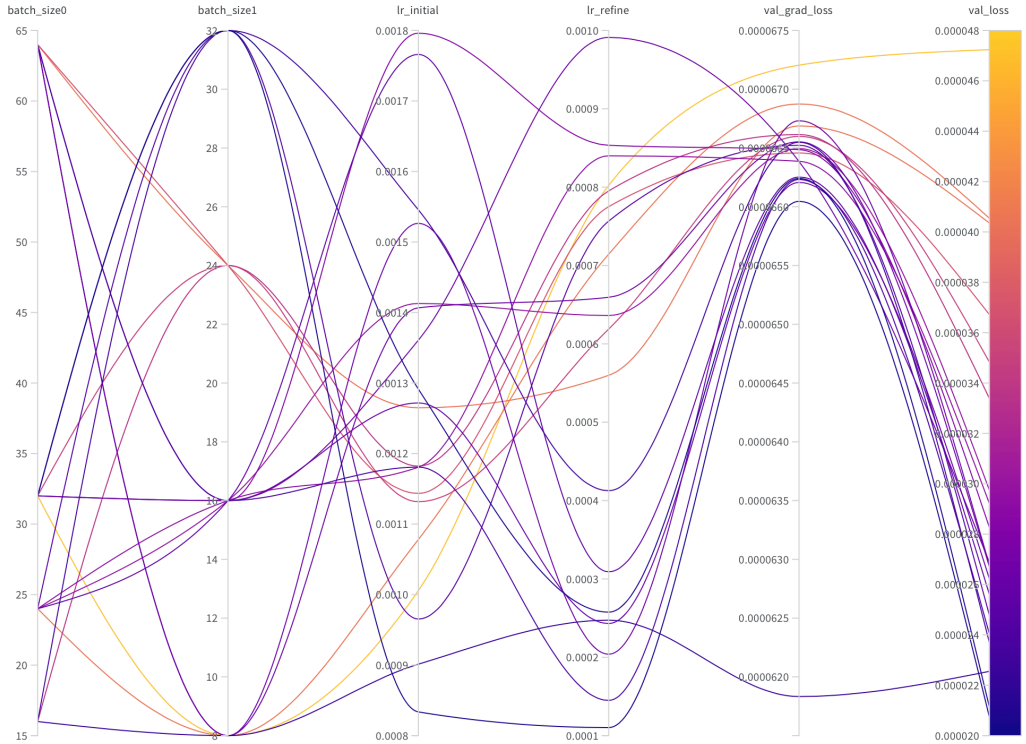


Figure 6.6. Second Optimization on the Low Noise Dataset

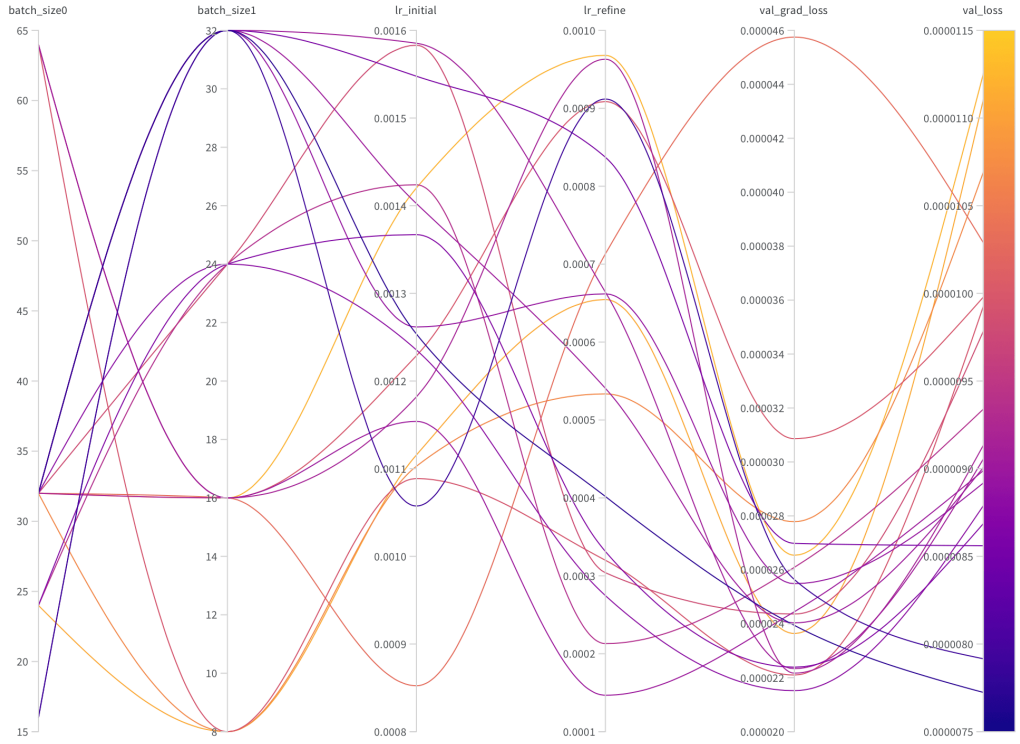


Figure 6.7. Second Optimization on the High Noise Dataset

Test N.1:	**Mixed Noise**	**Test N.2:**
Batch Size - 1	24	Batch Size - 1	16
Batch Size - 2	32	Batch Size - 2	8
L. Rate - 1	$1,6 * 10^{-3}$	L. Rate - 1	$1,5 * 10^{-3}$
L. Rate - 2	$1,1 * 10^{-4}$	L. Rate - 2	$3,9 * 10^{-4}$
val_grad_loss	$8,486 * 10^{-5}$	val_grad_loss	$8,485 * 10^{-5}$
val_loss	$2,908 * 10^{-5}$	val_loss	$2,830 * 10^{-5}$
Test N.1:	**High Noise**	**Test N.2:**
Test N.1:		**Test N.2:**	
Batch Size - 1	32	Batch Size - 1	16
Batch Size - 2	32	Batch Size - 2	32
L. Rate - 1	$1,2 * 10^{-3}$	L. Rate - 1	$1,1 * 10^{-3}$
L. Rate - 2	$4,0 * 10^{-4}$	L. Rate - 2	$1,1 * 10^{-4}$
val_grad_loss	$2,394 * 10^{-5}$	val_grad_loss	$2,564 * 10^{-5}$
val_loss	$7,724 * 10^{-6}$	val_loss	$7,916 * 10^{-6}$
Test N.1:	**Low Noise**	**Test N.2:**
Batch Size - 1	32	Batch Size - 1	16
Batch Size - 2	32	Batch Size - 2	32
L. Rate - 1	$1,2 * 10^{-3}$	L. Rate - 1	$1,5 * 10^{-3}$
L. Rate - 2	$2,6 * 10^{-4}$	L. Rate - 2	$4,1 * 10^{-4}$
val_grad_loss	$6,624 * 10^{-5}$	val_grad_loss	$6,655 * 10^{-5}$
val_loss	$2,113 * 10^{-5}$	val_loss	$2,369 * 10^{-5}$

Table 6.11. Table of the Best Results

6.2.3 Step 3: Improving The Performance When Changing Settings During Training

A fairly critical point of training is the change between the two phases. It can also result in slight performance degradations, visible through the various metrics that evaluate the validation set. The main changes that occur during the change of the two phases are learning rate, batch size and losses. However, the thing that has the greatest impact on the decrease in performance is the new loss. In fact, it is no longer computed as a simple MSE loss (Formula 4.4), but now it is also computed considering the influence of gradient loss, as expressed by Formula 5.2. The addition of this feature may result in a worsening of performance. In order to create training that is as smooth as possible toward a decrease in validation loss, two parameters can be set:

- **Grad_Weight**, which handles how much the loss depends on the MSE and how much on the gradient;
- **Weight**, which decreases the influence of the gradient in order not to increase the value of the loss excessively.

To find their best values, it has been decided to vary only those two parameters, fixing the others at the optimum identified so far. The variations are as follows:

Variable Hyperparameter	Range of Values
Weight for Custom_MSE - Phase 2	From 0.0005 to 0.01
Grad_Weight for Custom_MSE - Phase 2	From 0.4 to 0.8

Table 6.12. Table of Tested Values For Hyperparameters

Instead, as for the hyperparameters that do not vary during the tests conducted in this subsection, they are defined in Table 6.13.

The two best settings for each of the three datasets are shown in Table 6.15. From it it can be seen that the optimal values are more or less the same for each dataset. The value of Grad_Weight ranges between about 0.45 and 0.55, while the value of

Fixed Hyperparameter	Value
Maximum Kernel Size	100
Number of Convolutions per Parallel Branch	6
Total Number of Parameters in the Parallel Branches	10000
Number of Parallel Branch	21
Number of Convolutions after Branch Concatenation	0
Maximum Number of Epochs - Phase 1	10
Maximum Number of Epochs - Phase 2	10
Batch Size - Phase 1	32
Batch Size - Phase 2	32
Learning Rate - Phase 1	0.00140
Learning Rate - Phase 2	0.00035

Table 6.13. Table of Fixed Hyperparameters

Weight ranges between 0.0015 and 0.0052. Generally smaller values of Grad_Weight require higher values of Weight and vice versa. In addition, the validation loss trends for each of these cases are shown in Table 6.14. Through them, it is possible to understand whether a deterioration in performance occurs as the phases change. The change of loss always occurs at epoch number 10. The training of the first phase is not mature enough to force the transition to the second phase via Early Stopping, so it is only needed to study at how the validation loss varies from epoch 10 to the next one. In each of the cases there is almost always an improvement, so the validation loss tends to decrease. Only a few cases are subject to a slight worsening, but this is corrected in later epochs. Of course, it is impossible to always prevent a worsening, since it always depends on random initializations. The important thing is that in most cases there is an improvement, while if there should be a worsening, it should be as mild as possible and corrected a few epochs later. Finally, looking also at the other optimal values of each given dataset, it is possible to notice that they fluctuate between the best values reported. Therefore, median values between them are chosen as the optimal ones. Their values are: **Grad_Weight** equal to **0.50** and **Weight** equal to **0.0035**.

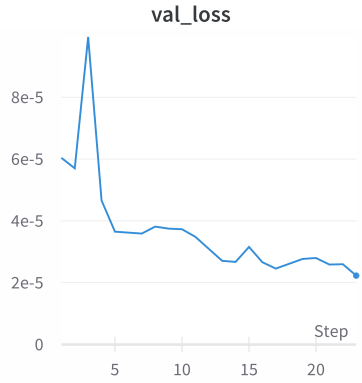
	Test N.1:	Test N.2:
MIXED NOISE		
LOW NOISE		
HIGH NOISE		

Table 6.14. Trend of Validation Losses During the Epoch Steps

Test N.1:	**Mixed Noise**		**Test N.2:**
Grad_Weight	0.44	Grad_Weight	0.56
Weight	0.0018	Weight	0.0052
val_grad_loss	$8,463 * 10^{-5}$	val_grad_loss	$8,476 * 10^{-5}$
val_loss	$2,783 * 10^{-5}$	val_loss	$2,766 * 10^{-5}$
Test N.1:	**Low Noise**		**Test N.2:**
Grad_Weight	0.46	Grad_Weight	0.55
Weight	0.0025	Weight	0.0019
val_grad_loss	$6,645 * 10^{-5}$	val_grad_loss	$6,632 * 10^{-5}$
val_loss	$2,229 * 10^{-5}$	val_loss	$2,300 * 10^{-5}$
Test N.1:	**High Noise**		**Test N.2:**
Grad_Weight	0.53	Grad_Weight	0.41
Weight	0.0015	Weight	0.0066
val_grad_loss	$2,456 * 10^{-5}$	val_grad_loss	$2,176 * 10^{-5}$
val_loss	$7,234 * 10^{-6}$	val_loss	$7,471 * 10^{-6}$

Table 6.15. Table of the Best Results

6.2.4 Step 4: Studying the pre-processing

Two types of pre-processing of the dataset are evaluated. The first pre-processing corresponds to that mentioned in Section 4.1 and it will be called "Type1" for simplicity. Briefly summarized, it shifts each sample of the sample mean, and then normalizes the training set by the standard deviation always computed on the set itself. Then, the same normalization is done for the test set, but the scaling is performed with its own mean and standard deviation. Finally the whole set is normalized to the absolute value of the maximum of the training set. A graphic example of how the samples change can be seen in comparison Image 6.17 where a sample from the High Noise dataset has been pre-processed. Instead, the other normalization tested will be called "Type2" for simplicity, and it calculates the maximum and minimum of the entire training set, in order to then normalize each sample to that value. In this way, all samples have the same maximum and minimum. Then, even in this case the same pre-processing is done for the test set, but scaling is performed with its own maximum and minimum. Finally the whole set is normalized to the absolute value of the maximum of the training set. A graphic example of this other type of pre-processing can be seen in comparison Image 6.18 where the same sample of the High Noise dataset from before has been pre-processed.
Several tables are then reported, each for each dataset: Table 6.19 shows the results of the Mixed Noise dataset where there is few noise along the whole signal and peaks are not excessively soaring compared with the baseline; Table 6.20 shows the results of the Low Noise dataset where there is few noise along the whole signal and peaks are fairly soaring compared with the baseline; and finally Table 6.21 shows the results of the High Noise dataset where there is a larger noise along the whole signal and peaks are excessively soaring compared with the baseline. In addition, in each of these tables and in all future ones, the NMSE metric is analyzed close to zero, even if this involves a truncation of some results achieved by other worse-performing algorithms. This choice was made to better analyze the minimal differences between the best performances obtained, while it is not relevant to know precisely the values

obtained by algorithms that do not perform this task as well.

Moreover, three neural networks are compared in this step: one uses the settings present at the beginning with the normalization "Type1", while the second and third use the optimal network to compare the results obtained with both the first and second preprocessing exposed. Instead, as for all other hyperparameters of the network, they are set to the previously found optimal value. In particular, they are:

Fixed Hyperparameter	**Value**
Number of Parallel Branch	21
Number of Convolutions after Branch Concatenation	0
Maximum Kernel Size	100
Number of Convolutions per Parallel Branch	6
Total Number of Parameters in the Parallel Branches	10000
Weight for Custom_MSE - Phase 2	0.0035
Grad_Weight for Custom_MSE - Phase 2	0.50
Maximum Number of Epochs - Phase 1	80
Maximum Number of Epochs - Phase 2	80
Batch Size - Phase 1	32
Batch Size - Phase 2	32
Learning Rate - Phase 1	0.00140
Learning Rate - Phase 2	0.00035

Table 6.16. Table of Fixed Hyperparameters

First and foremost what is evident from each dataset is that there are vast improvements over the initial settings. However, as for a comparison between the two pre-processings exposed with the network at its current optimum, "Type1" preprocessing always gives better results. The precision is higher, the 0.25% precision of the worst samples is the best, and the metrics that calculate the difference from the clean signal are the lowest. Moreover, as an additional comment, it is even possible to point out that there are cases where the "Type2" pre-processing is even worse than the "Type1" used with the initially non-optimal network. This is the case illustrated in Table 6.19 and Table 6.20, where the results of the Mixed Noise and Low Noise datasets are shown respectively. In any case, each dataset indicates how much the choice made initially was the most suitable and thus to use normalization based on the mean and variance of the entire dataset.

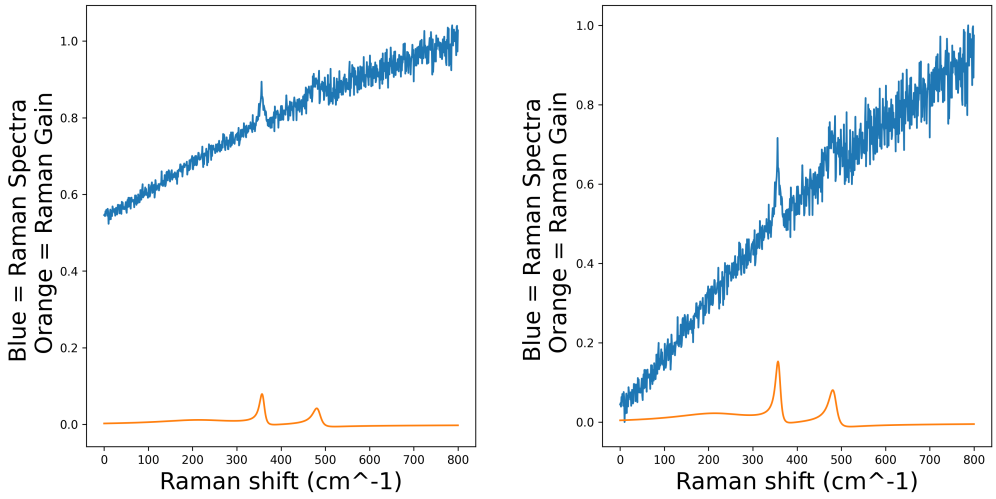


Table 6.17. Before-and-after comparison of type 1 pre-processing

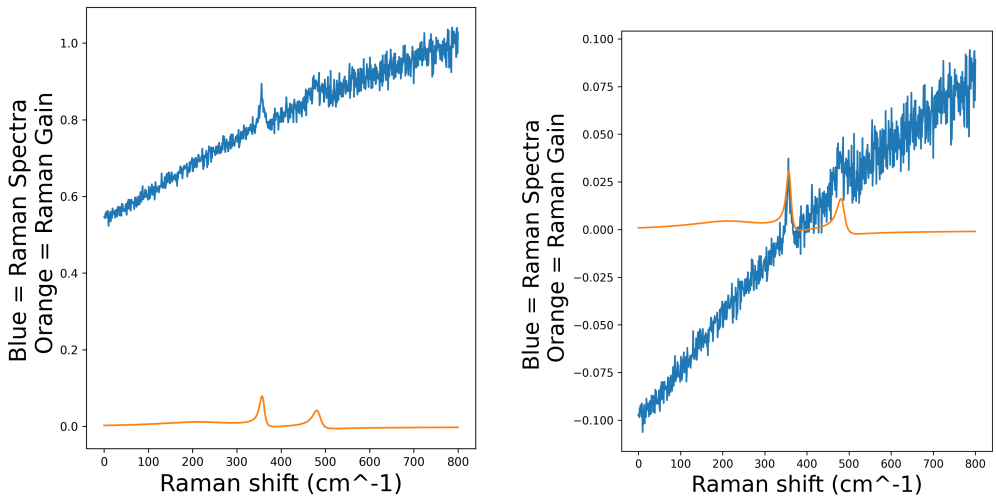
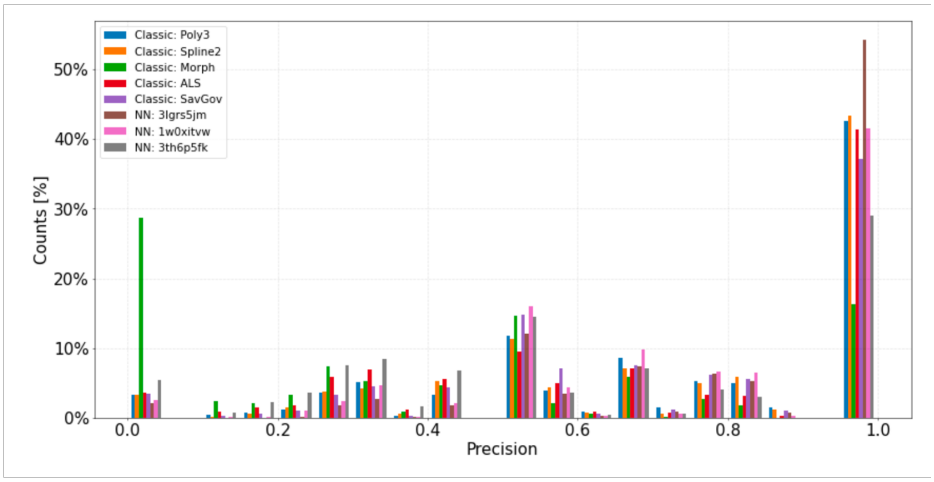


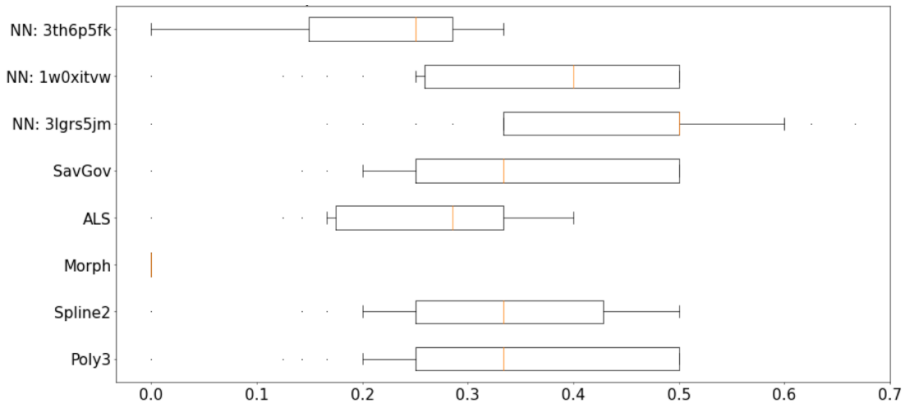
Table 6.18. Before-and-after comparison of type 2 pre-processing

First Attempt with "Type1" Pre-Processing = NN:3th6p5fk
Optimized Network with "Type1" Pre-Processing = NN:3lgrs5jm
Optimized Network with "Type2" Pre-Processing = NN:1w0xitvw

Precision of Detected Peaks



Precision of the Worst 0.25% of the Test Set



Results Obtained from the Other Metrics

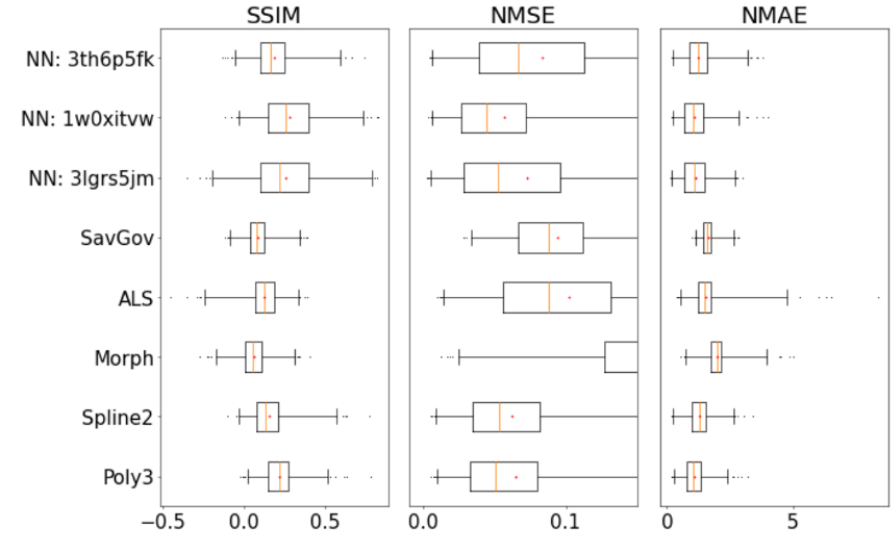
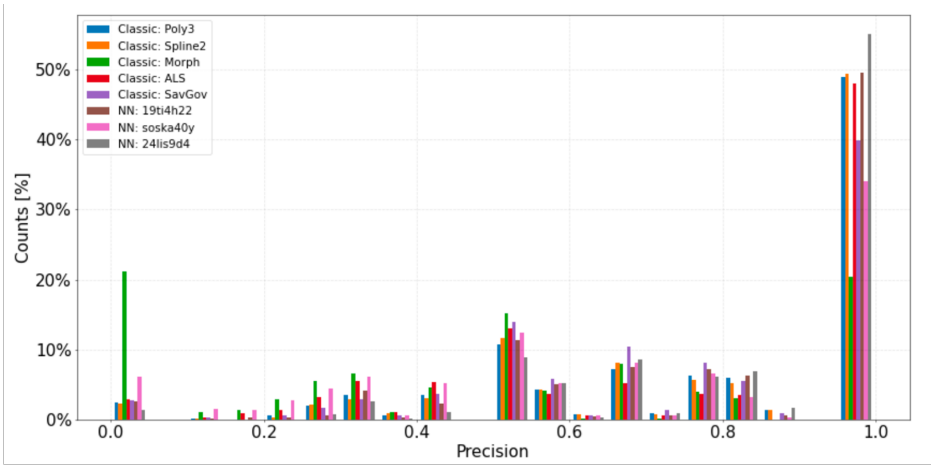


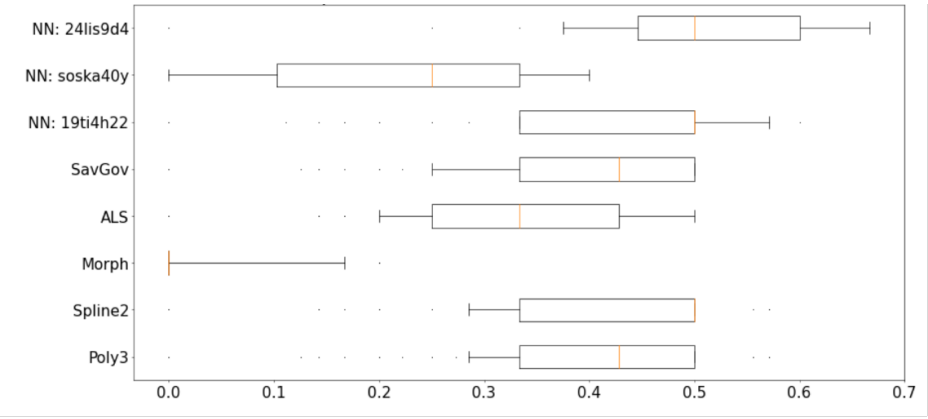
Table 6.19. Metric analysis of the results obtained in the Mixed Noise dataset

First Attempt with "Type1" Pre-Processing = NN:19ti4h22
Optimized Network with "Type1" Pre-Processing = NN:24lis9d4
Optimized Network with "Type2" Pre-Processing = NN:soska40y

Precision of Detected Peaks



Precision of the Worst 0.25% of the Test Set



Results Obtained from the Other Metrics

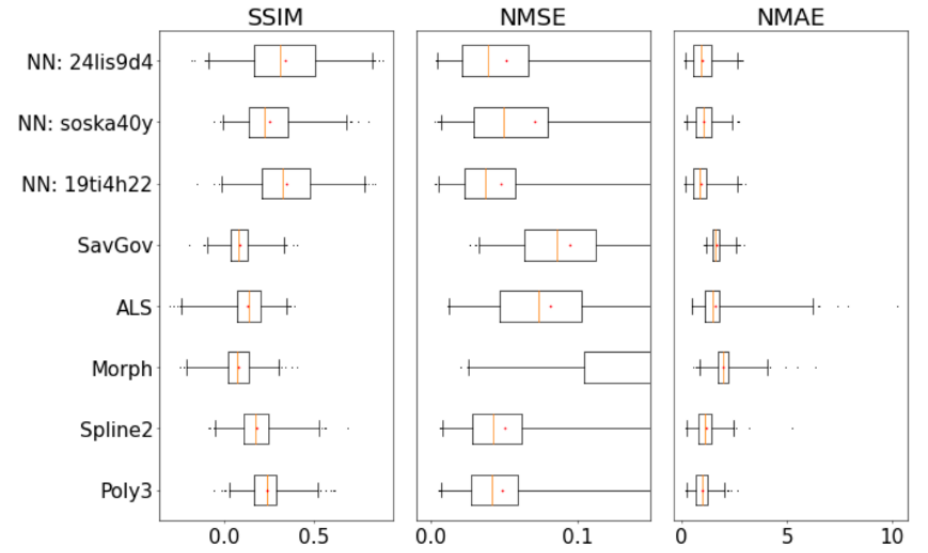
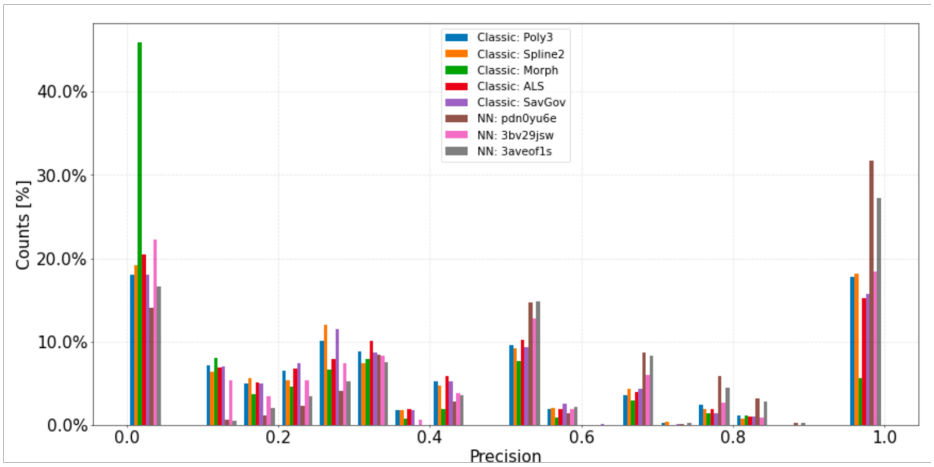


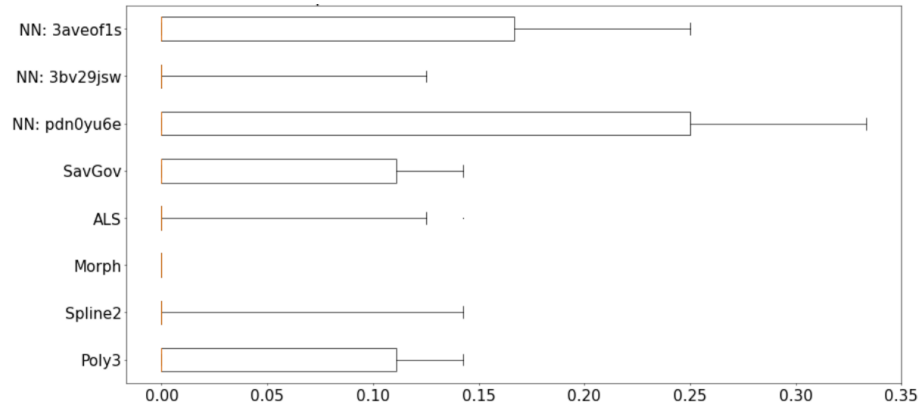
Table 6.20. Metric analysis of the results obtained in the Low Noise dataset

First Attempt with "Type1" Pre-Processing = NN:3bv29jsw
Optimized Network with "Type1" Pre-Processing = NN:pdn0yu6e
Optimized Network with "Type2" Pre-Processing = NN:3aveof1s

Precision of Detected Peaks



Precision of the Worst 0.25% of the Test Set



Results Obtained from the Other Metrics

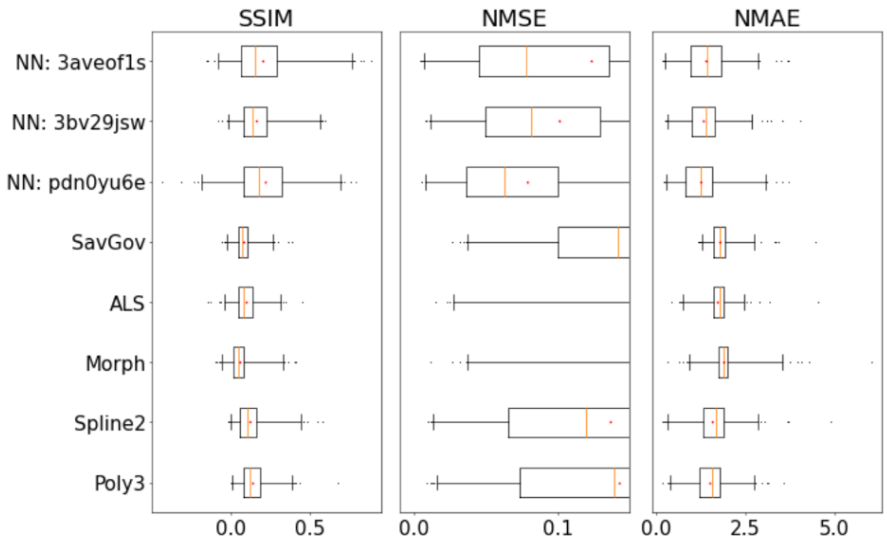


Table 6.21. Metric analysis of the results obtained in the High Noise dataset

6.2.5 Step 5: Insight into Fundamental Hyperparameters

From the many tests conducted, it has been found that two hyperparameters seem to greatly influence performance. They are the number of parallel branches and the convolution number at the end of parallel branches. In order to determine more precisely what their optimal value is, it has been decided to wait for the optimization of all others by parameters and then study their performance. All tests involving these hyperparameters are carried out by fixing the others as follows:

Fixed Hyperparameter	**Value**
Maximum Kernel Size	100
Number of Convolutions per Parallel Branch	6
Total Number of Parameters in the Parallel Branches	10000
Weight for Custom_MSE - Phase 2	0.0035
Grad_Weight for Custom_MSE - Phase 2	0.50
Maximum Number of Epochs - Phase 1	80
Maximum Number of Epochs - Phase 2	80
Batch Size - Phase 1	32
Batch Size - Phase 2	32
Learning Rate - Phase 1	0.00140
Learning Rate - Phase 2	0.00035

Table 6.22. Table of Fixed Hyperparameters

The tests to vary the two hyperparameters discussed in this subsection are performed separately, and the combinations tried for each are shown together in the table below:

Variable Hyperparameter	**Range of Values**
Number of Parallel Branch	18, 21, 40, 60
Number of Convolutions after Branch Concatenation	0, 4, 7

Table 6.23. Table of Tested Values For Hyperparameters

Regarding the final number of convolutions, this parameter is of crucial importance because it determines how much the network should process the number of channels that are obtained from the concatenation of the parallel branches before passing them to a one-dimensional convolution. If it is set to zero, it means that the network

averages all the values at the same positions of the parallel channels, whereas if there are convolutions that layer to layer halve in number of channels, then the value to be put into the output is intelligently learned during training based on the input values. From what can be seen in the results shown in Table 6.24, excessively increasing the number of convolutions greatly decreases performance. Not only it decrease the accuracy of the number of correctly predicted spikes, but it also worsens the values of NMSE, NMAE and SSIM. These last three metrics report the difference between the predicted and correct signal. The increase in the first two and the decrease in the last one indicate a poor prediction with respect to the corrected signal. In contrast, as for the case where the convolution number is zero or four, they report similar results. In any case, however, the case study with 0 final convolutions after parallel branches reports slightly better results in both precisions and assessment metrics of similarity and error. Moreover, the same result is also encountered in the other two datasets. Therefore, without reporting further comparison metrics that would only be redundant, it can be established that the best number of convolutions after the concatenation of the parallel branches and before the convolution that returns the data to a single channel is 0.

Instead, regarding the number of parallel branches, the case is more complicated. In fact, each dataset reports slightly different behaviors. The results reported with the High Noise dataset are expressed in Table 6.25. In it it can be noted that all three options for the number of parallel branches report almost identical accuracy in peak detection. Moreover, it is even higher than any other more conventional method that does not make use of neural networks. Furthermore, the same result can also be found in the accuracy among the worst samples and among the similarity and error evaluation metrics. Then comparing only the cases with neural networks, the one that reports slightly better results is with 21 parallel branches. In contrast, with regard to the results obtained with the Low Noise dataset and shown in Table 6.26, there are many considerations to be made. Regarding the precisions of the peaks correctly predicted by the neural networks, they are all very close to each other,

although the case study with 21 parallel branches is slightly better to the others. In any case, however, all the networks are slightly worse than other algorithms that do not make use of artificial intelligences. However, this aspect is not of great importance since the task of these algorithms should not only be to detect peaks, but also to perform denoising throughout the signal. In fact, according to the other error and similarity evaluation metrics, neural networks are the algorithms with the least difference and most similarity between the predicted signal and the correct signal. The other algorithms that do not make use of neural networks, on the other hand, present worse evaluations. Moreover, again considering the latter metrics, it can be seen that the case study with 60 parallel branches is the one with slightly better metrics than the other two neural networks. Considering the fact that the accuracy of the detected spikes is roughly similar among all the case histories and is a less relevant metric than the latter, it can be established that in the case of the lo noise dataset the most suitable number of parallel branches is 60. Finally, regarding the results obtained with the Mixed Noise dataset and shown in Table 6.27, the considerations are similar to those made for the Low Noise dataset. In fact, the accuracy of the detected peaks is slightly higher for the classical algorithms than with the neural networks, and the neural networks obtain very similar results to each other. Moreover, looking solely at this evaluation metric, the best result is again the case with 21 parallel branches. In any case, however, looking at the other metrics as well, the best results seem to be obtained by the neural networks. They report lower error greater similarity between the predicted signal and the correct signal. Moreover, again looking at these metrics, the case that reports better results is the case with 40 parallel branches. Considering that the precision of the predicted peaks is very similar among all neural networks, it is possible to establish that in this case the optimal number of branches is 40. In general, it is possible to assume that each dataset needs a different number of parallel branches than the others based on the amount of noise and the types of noise that are contained in it.

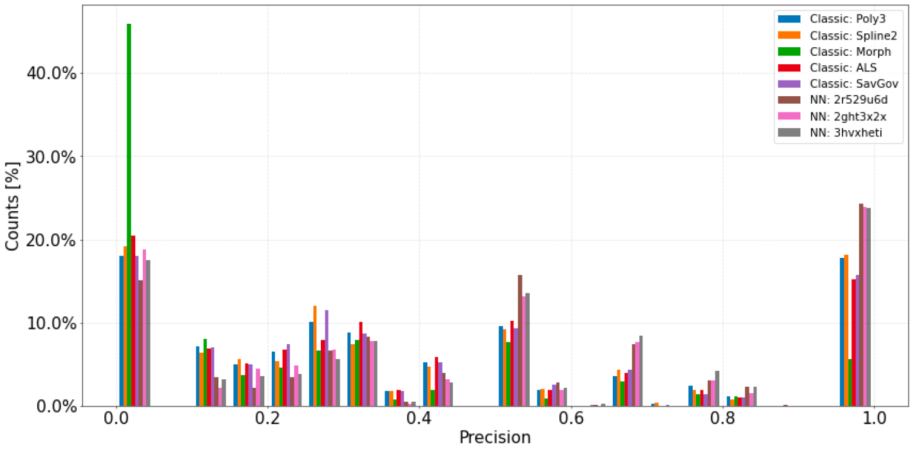
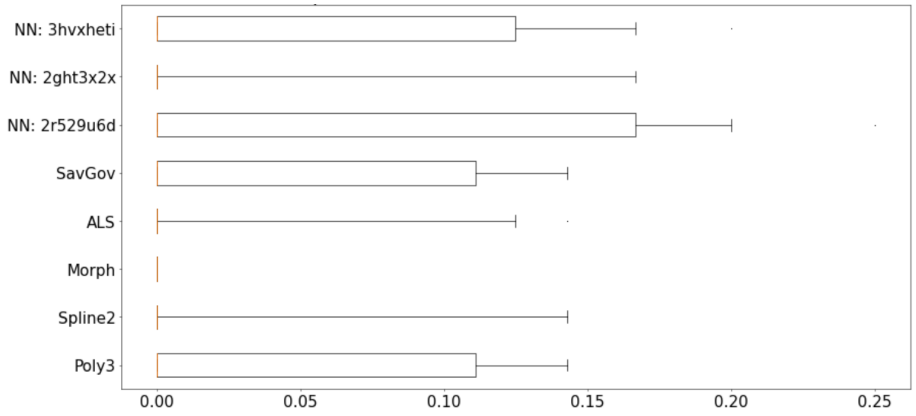
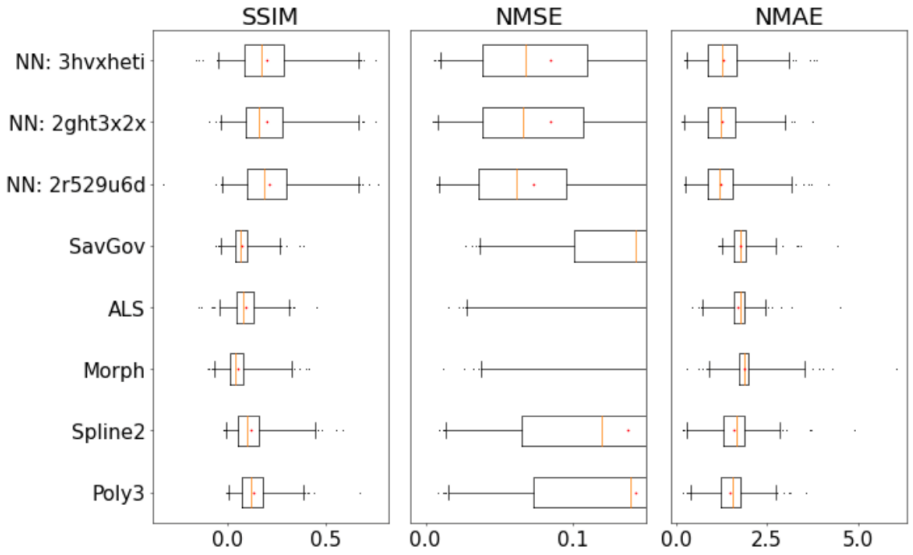
0 Final Convolution After Parallel Branches = NN:2r529u6d 4 Final Convolution After Parallel Branches = NN:2ght3x2x 7 Final Convolution After Parallel Branches = NN:3hvxheti
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


Table 6.24. Metric analysis of the results obtained on the High Noise dataset regarding the number final convolution after parallel branches

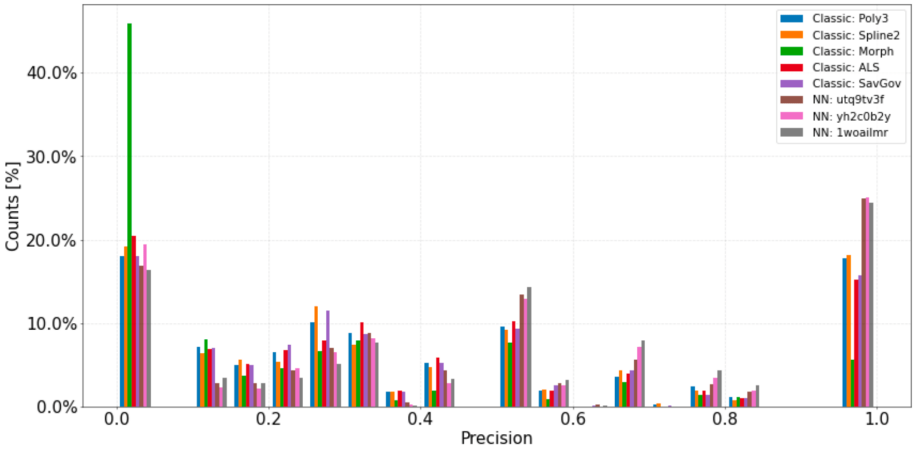
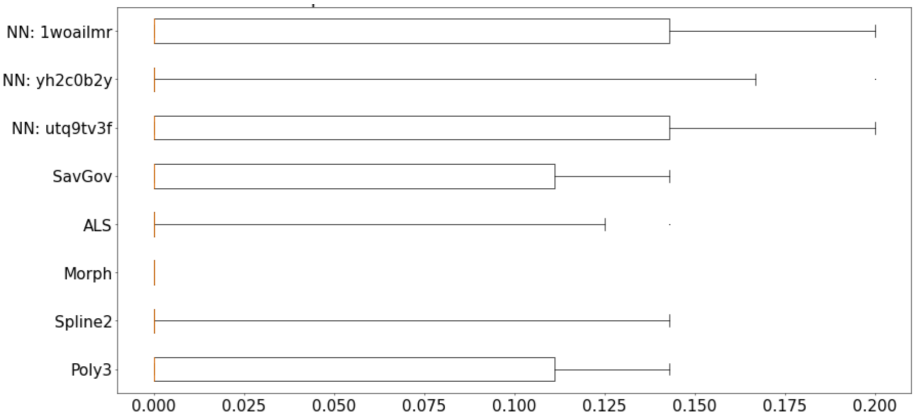
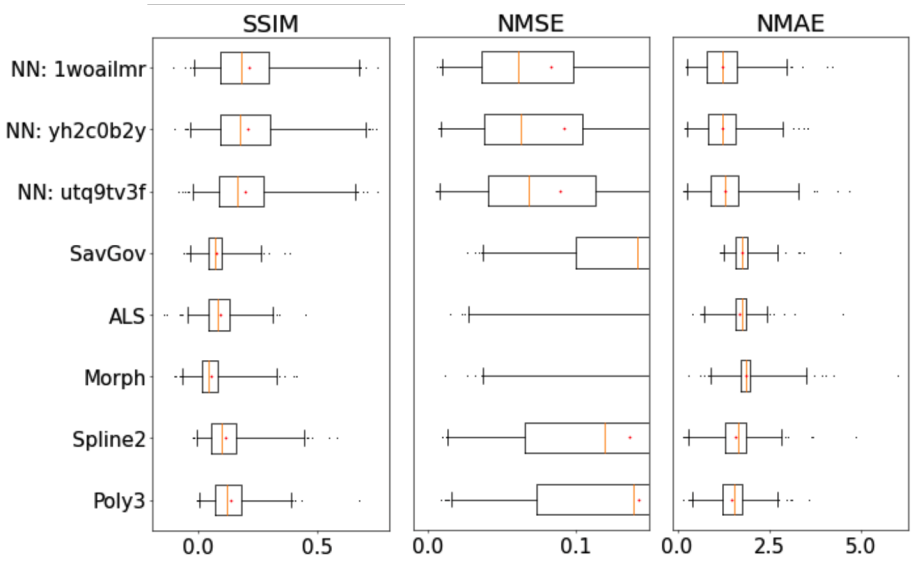
21 Parallel Branches = NN:utq9tv3f 40 Parallel Branches = NN:yh2c0b2y 60 Parallel Branches = NN:1woailmr
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


Table 6.25. Metric analysis of the results obtained on the High Noise dataset regarding the number of parallel branches

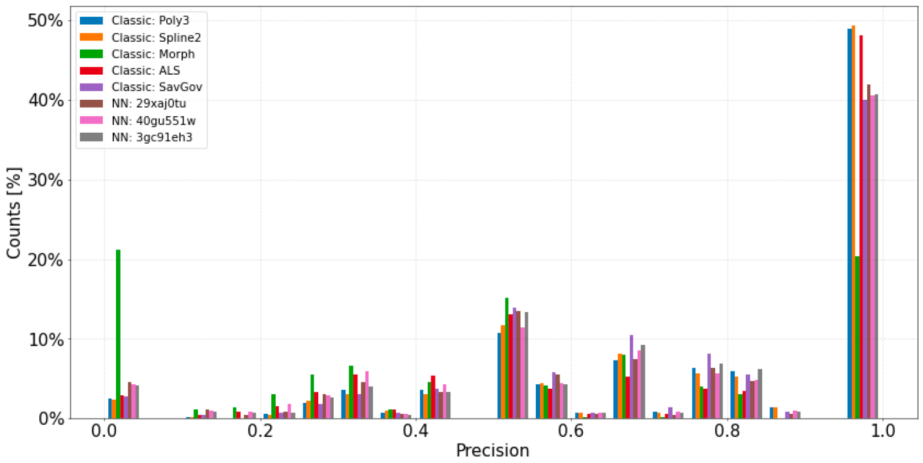
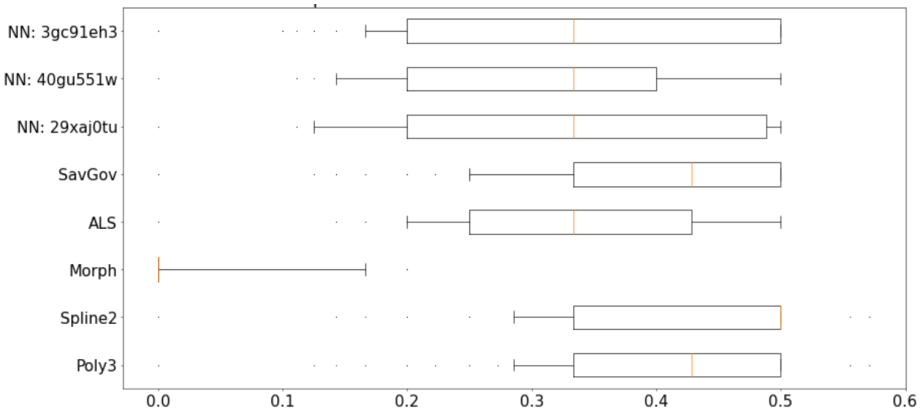
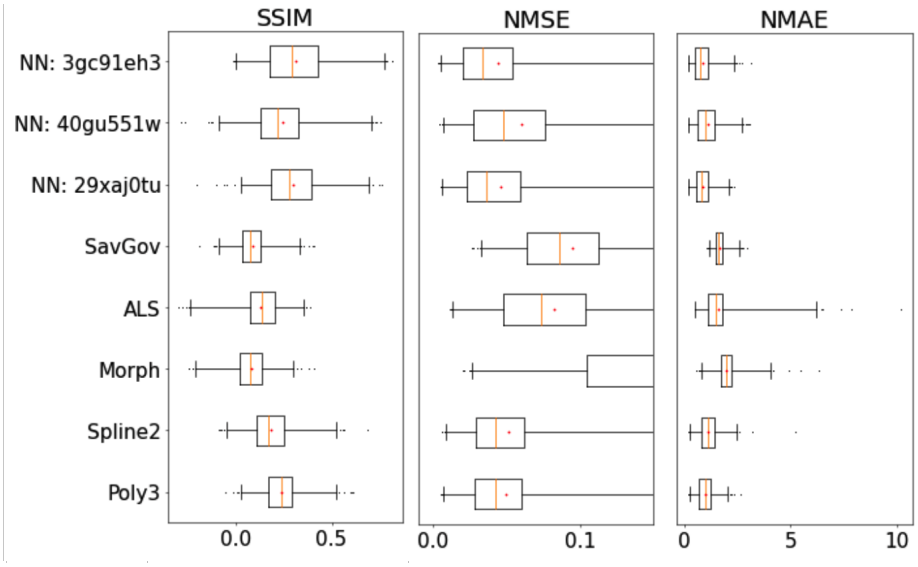
21 Parallel Branches = NN:29xaj0tu 40 Parallel Branches = NN:40gu551w 60 Parallel Branches = NN:3gc91eh3
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


Table 6.26. Metric analysis of the results obtained on the Low Noise dataset regarding the number of parallel branches

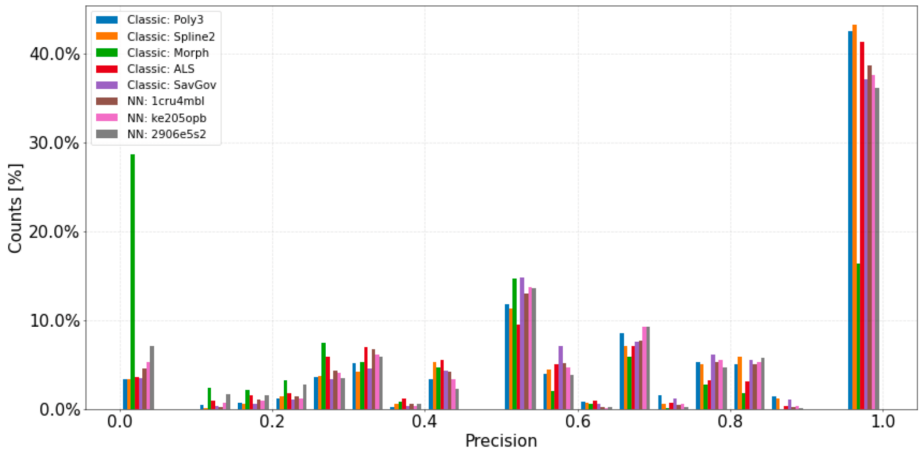
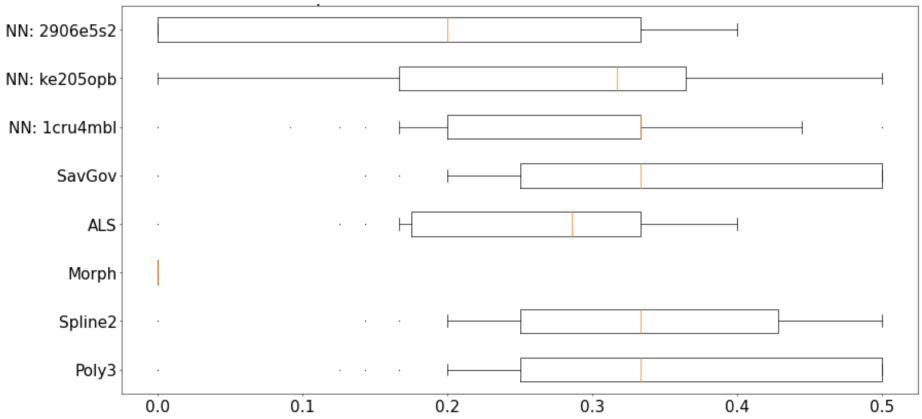
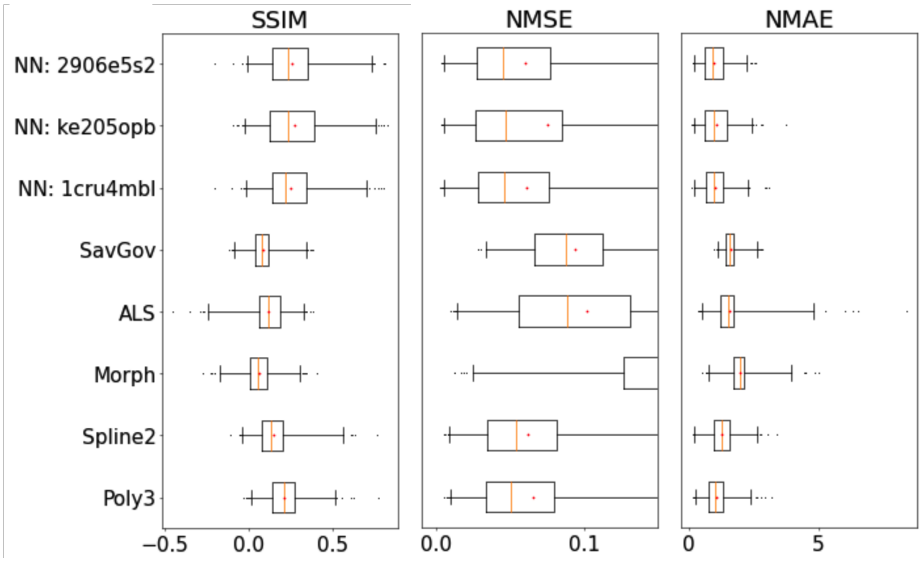
21 Parallel Branches = NN:1cru4mbl 40 Parallel Branches = NN:ke205opb 60 Parallel Branches = NN:2906e5s2
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


Table 6.27. Metric analysis of the results obtained on the Mixed Noise dataset regarding the number of parallel branches

6.3 Customization and Improvement of Losses

The optimized network so far has achieved excellent levels of performance in denoising the Raman spectrum. However, there are still outlier samples that are not cleaned up perfectly. They are signals that either have excessive baseline distortion or an excessive amount of noise along the signal. To improve the performance of the model even more, further work on the losses is necessary. In particular, the loss used in the second stage already makes use of the gradient loss. In addition to it, however, it might be possible to include additional metrics in the loss computation. In this way, metrics that evaluate different aspects of the signal could contribute positively to improving the denoising of the signal during the training itself. Two very important metrics in the evaluation of noise in signal are SNR and SSIM. Compared with NMAE and NMSE, they deviate from the simple calculation of the difference between the predicted and the correct signal, but seek to provide a quantification of the noise and a quantification of the similarity between the two signals, respectively. The use of SNR is discussed in Subsection 6.3.1, while the use fo the SSIM is discussed in Subsection 6.3.2.

6.3.1 Inclusion of SNR in the Second Phase Loss

As mentioned earlier, signal to noise ratio (SNR) is a measure that compares the level of a desired signal to the level of background noise. Basically, it returns an assessment of the noise that is still present in the signal. Its implementation within the loss used in the second phase has been made made in three different ways:

$$LOSS = (1 - SNR\_weight\_2) * MSE + SNR\_weight\_2 * Weight\_2 * SNR \tag{6.4}$$

$$\begin{aligned} LOSS = (1 - Grad\_weight - SNR\_weight\_2) * MSE \\ + Grad\_weight * Weight * loss\_grad \\ + SNR\_weight\_2 * Weight\_2 * SNR \end{aligned} \tag{6.5}$$

$$LOSS = (1 - Grad\_weight - SNR\_weight\_2) * MSE$$
$$+(Grad\_weight * loss\_grad + SNR\_weight\_2 * SNR) * Weight \tag{6.6}$$

where $loss\_grad$ is computed as expressed in Formula 5.3, SNR is calculated as expressed in Formula 4.14, $Grad\_Weight$ and SNR_weight_2 are used to proportion the influence of SNR, $loss\_grad$ and MSE to 1, while $Weight$ and $Weight\_2$ are used to mitigate the overall increase in loss when switching from the classical MSE to this option combined with $loss\_grad$ and SNR. In essence, SNR_weight_2 and $Weight\_2$ have the same function as the homologues used for $loss\_grad$. The substantial difference between Formula 6.5 and Formula 6.6 is that in the former there is the presence of a fourth hyperparameter that attempts to dampen the positive change in validation loss due to SNR separately from $loss\_grad$, whereas in the second case there is only one hyperparameter that attempts to do simultaneously. Unfortunately, however, the loss tends toward gradient explosion. After a small number of epochs, the loss explodes and the training fails. Several solutions have been tried to mitigate this behavior:

- **Addition of an epsilon in SNR computation** to avoid division with zero or to make sure that the SNR value does not become too small;
- **Using Gradient Clipping** in the Adam optimizer. This should prevent excessive rise or fall of gradients. It has been implemented either by clipping on the norm of all values (Clipnorm) or by clipping on all values individually (Clipvalue). In particular, the former changes the derivative of the loss when the L2 vector norm of the gradient vector exceeds a certain threshold, while the second acts directly on the value of the gradient;
- **Changing Optimizer.** It is advisable to use adaptive learning rate method, such as Adam or RMSProp. The former was in use from the beginning, and the latter did not solve the problem. The use of stochastic gradient descent has also been attempted, even if is not an adaptive learning rate method, but it did not solve the problem;

- **Reduce the number of layers** in the neural network;
- **Lowering the learning rate** to reduce the gradient magnitude. Another attempt that can be made related to the learning rate is to gradually reduce it as the training progresses. This also prevents gradients from becoming too large. In any case, this technique has always been used from the beginning;
- **Adding an L1 or L2 regulator** in network convolutions. This attempt seeks to control the size of the network weights and it applies a penalty to the network loss function when there are high weight values;
- **Increasing the batch_size** of the second phase;
- **Change the fixed seed.** On rare occurrence, fixed seeds may be the cause of an unfortunate neural network initialization, and thus cause the gradient to explode;
- **Using non-saturating activation functions.** This means changing the ReLU activation function with similar ones. Therefore, LeakyReLU and SeLU (Scaled Exponential Linear Unit) activations are also tested. The latter also requires a special kernel initialization;
- **Adding batch normalization layers.** Several studies, such as those carried out in paper [53], show how the addition of such layers can solve the explosion gradient problem.

None of these solutions solved the problem. At most, solutions such as inserting batch normalization layers or clipping the gradient slowed down the timing of occurrence of this phenomenon. In any case, however, this is more due to the fact that through such solutions, the training itself is slower to learn the features, so they are not really correcting the problem, but simply delaying the reaching of the critical point. Therefore, further studies are needed to include SNR-like terms in training loss.

6.3.2 Inclusion of SSIM in the Second Phase Loss

As carried out in the previous subsection, the same reasoning is also applicable to the case of the SSIM metric. It is used to compute a similarity index between the two signals passed as input. When it is worth 0 it means they are completely different, while when it tends to 1 it means they are very similar. Its implementation is very similar to the implementation made for the SNR attempt. In fact, three variants are tested:

$$\begin{aligned} LOSS = (1 - SSIM\_weight\_2) * MSE \\ +SSIM\_weight\_2 * Weight\_2 * SSIM \end{aligned} \tag{6.7}$$

$$\begin{aligned} LOSS = (1 - Grad\_weight - SSIM\_weight\_2) * MSE \\ +(Grad\_weight * loss\_grad + SSIM\_weight\_2 * SSIM) * Weight \end{aligned} \tag{6.8}$$

$$\begin{aligned} LOSS = (1 - Grad\_weight - SSIM\_weight\_2) * MSE \\ +Grad\_weight * Weight * loss\_grad \\ +Grad\_weight_2 * Weight\_2 * SSIM \end{aligned} \tag{6.9}$$

where $loss\_grad$ is computed as expressed in Formula 5.3, SSIM is calculated as expressed in Formula 4.6, $SSIM\_weight\_2$ is used to proportion the influence of SSIM, $loss\_grad$ and MSE to 1, while $Weight$ is used to mitigate the overall increase in loss when switching from the classical MSE to this option with SSIM. In addition, numerous executions are carried out with all formulas in order to find the best value for each hyperparameter. In any case, however, the performance of the network deteriorated compared with the case without SSIM. Table 6.31 shows a comparison between networks two neural networks that make use of the previously mentioned formulas and a neural network that uses the previous identified optimal network. All of them are trained on the Mixed Noise dataset, which is characterized by a signal with peaks not very soaring from the baseline, not much noise throughout the whole signal, and a frequently very distorted baseline. The used hyperparameters are as follows:

Parameters	Without SSIM	Formula 6.7	Formula 6.8	Formula 6.9
Grad_weight	0.5	0.5	0.68	0.6
SSIM_weight_2	-	0.1	$7.0 * 10^{-4}$	$3.0 * 10^{-3}$
Weight	$3.5 * 10^{-3}$	$3.5 * 10^{-3}$	$9.6 * 10^{-4}$	$5.0 * 10^{-3}$
Weight_2	-	$3.4 * 10^{-3}$	-	$2.6 * 10^{-4}$

Table 6.28. Table of Tested Values For Hyperparameters

In this table some hyperparameters previously established as optimal if they take a certain value are varied. This is due to the fact that now they are related to other hyperparameters, and thus an additional search for the optimum is necessary. Therefore, without again reporting the analyses on the choice of these values, they are directly reported as such. Instead, for what concerns the other fixed hyperparameters, they are:

Fixed Hyperparameter	Value
Number of Convolutions after Branch Concatenation	0
Maximum Kernel Size	100
Number of Convolutions per Parallel Branch	6
Total Number of Parameters in the Parallel Branches	10000
Maximum Number of Epochs - Phase 1, 2 And 3	80
Batch Size - Phase 1, 2 And 3	32
Learning Rate - Phase 1	0.00140
Learning Rate - Phase 2 And 3	0.00035

Table 6.29. Table of Fixed Hyperparameters

Regarding the number of parallel branches, it varies with respect to the dataset. Specifically, it is 21 for the High Noise dataset, 40 for the Mixed Noise dataset, and 60 for the Low Noise dataset.

The best result is obtained when attempting to add SSIM to the second phase loss that already makes use of MSE loss and gradient loss, i.e., Formula 6.9. However, it still does not equal the previous performance. In fact, compared with the case without SSIM, the total accuracy and that on the worst samples decreases with the presence of this metric. Even error metrics report a greater difference between the predicted and corrected signal in cases where there is SSIM. Probably the addition of

this metric introduces too much instability in the validation loss. In fact it computes a point-to-point similarity in the signal, and if there is still too much noise in the signal, the SSIM value becomes very small. This can result in two cases: an almost useless contribution to the loss if the weights by which it is multiplied are too small; an excessive contribution to the loss with respect to the other metrics if the weights are too high, leading at the same time to less dependence of the loss on the important metrics that calculate the difference between the predicted and the correct signal. This can also be validated by the neural network that incorporated only SSIM in the second validation loss. In fact, it achieves much worse performance than all other cases. In addition, the case with Formula 6.7 has not been reported because the performance is much worse than even the algorithms that do not make use of neural networks. Therefore, the tests showed that the SSIM term in the loss of the second training phase does not achieve improvements in network performance.

In any case, however, the presence of such a metric in the loss does not lead to a gradient explosion, so another approach can be tried. While adding SSIM in the second stage may not be the best thing since the network is still learning to denoise the signal more finely and accurately, it could be attempted to include it in a third stage of training. In fact, in the first two one could continue to use the same settings already tested, so with an MSE loss in the first phase and a custom loss that also makes use of the gradient in the second phase, and then perform a third refinement phase where the influence of that metric is also added. Therefore, the model that is passed to the third phase is at maximum efficiency and the residual noise is minimal. On the other hand, as for the loss formulations used in the latter phase, they are the same as those used previously in the second phase, namely Formula 6.7, Formula 6.8 and Formula 6.9. Regarding the fixed hyperparameters and the number of parallel branches, they are the same as those used in the previous attempt. Instead, the values of the various loss weights are different, as they required different trainings for their optimization. They are:

Tests conducted on the High Noise, Low Noise and Mixed Noise datasets report

Parameters	Without SSIM	Formula 6.7	Formula 6.8	Formula 6.9
Grad_weight	0.5	0.5	0.65	0.6
SSIM_weight_2	-	$7.6 * 10^{-3}$	$2.8 * 10^{-2}$	$4.8 * 10^{-2}$
Weight	$3.5 * 10^{-3}$	$3.5 * 10^{-3}$	$4.0 * 10^{-4}$	$5.0 * 10^{-3}$
Weight_2	-	$2.5 * 10^{-4}$	-	$1.3 * 10^{-4}$

Table 6.30. Table of Tested Values For Hyperparameters

different results and are shown in Table 6.32, Table 6.33 and Table 6.34 respectively. In the case of the High Noise dataset (Table 6.32), the best result that has been obtained with the presence of the SSIM metric in the third phase (Formula 6.9). The percentage precision of the number of spikes correctly detected is the highest compared to all other cases. Even the precision of the worst cases is the best, although the average of those cases is equally at zero for all algorithms. This means that only a few peaks are predicted better. Finally, the same result can also be found in metrics that measure the similarity and error between the predicted and corrected signal. Although the differences are slight, they indicate a possible improvement in denoising signals that have a large baseline distortion, high noise throughout the signal, and a high relative difference between the peaks and the baseline. In contrast, regarding the case of the Low Noise dataset (Table 6.33), it shows that the best case is the one that does not implement the third phase with SSIM. Not only the precision of detected peaks is higher, but the other metrics also report slight deteriorations in the performance of the networks that make use of the SSIM metric. This could always be attributable to the fact that this metric takes into account the point-to-point similarity between the predicted signal and the correct signal. In fact, in that case there is not much noise along the whole signal and the first two phases already perform very good denoising of the signal. This metric, however, may be too susceptible to the slight remaining noise, and therefore causes excessive fluctuations in the validation loss that would not allow it to further correct for the remaining noise in the third phase. An analogous thing happens in the Mixed Noise dataset (Table 6.34). Here again the best network is the one that does not make use of the

SSIM metric. In fact, although this has always been the most difficult dataset and in which the performance obtained is more similar to that of algorithms that do not make use of neural networks, this dataset also has few noise along the entire signal and the first two phases already correct the baseline distortions quite well.
From these tests, it is evident that when there is high noise, especially along the entire signal, the work done by the first two phases can be refined by a third phase that makes use of the SSIM metric with formula 1. In contrast, in other cases there are slight improvements. Given the relationship that exists between signal noise and two network components, namely the number of parallel branches and the SSIM metric in the loss of a third training phase, it would be appropriate to conduct further tests in order to understand with certainty how much these three components are related and whether there is an optimal combination that can lead to steady improvement in each noise case and dataset.

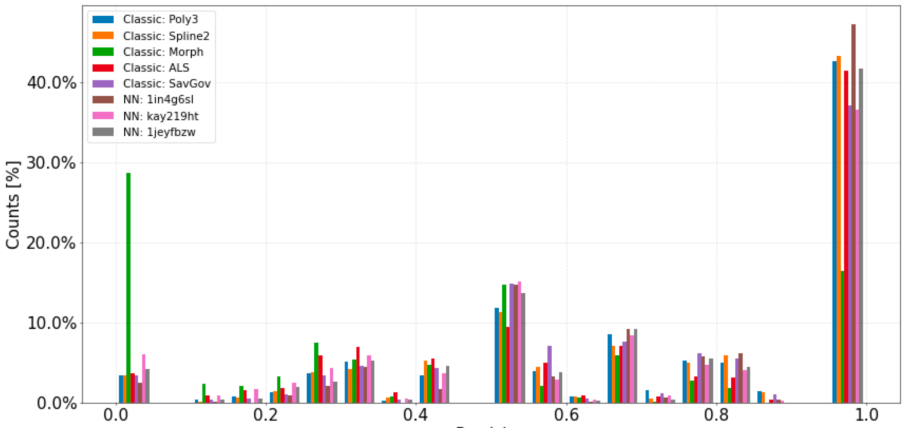
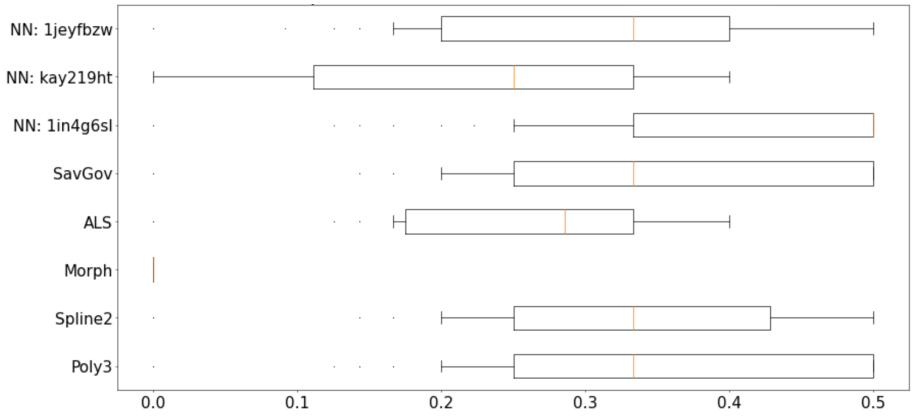
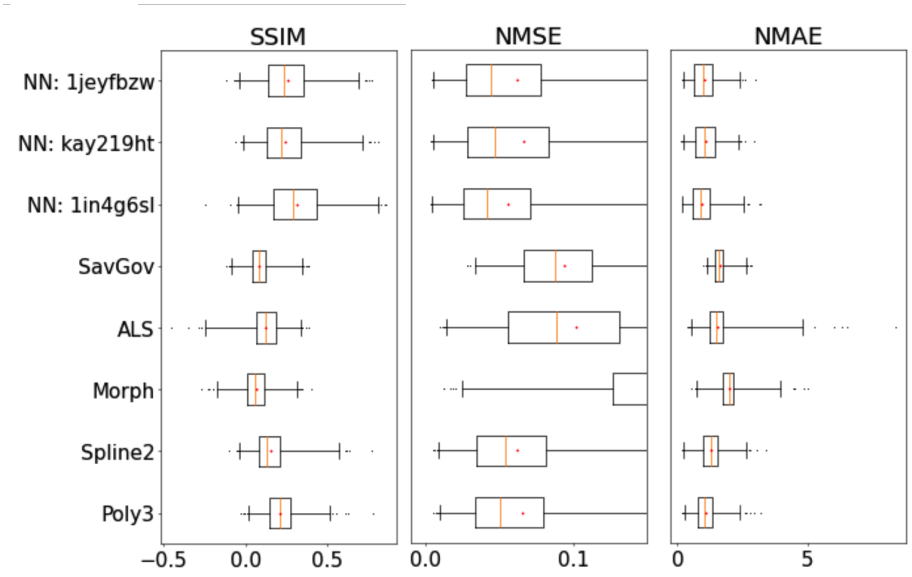
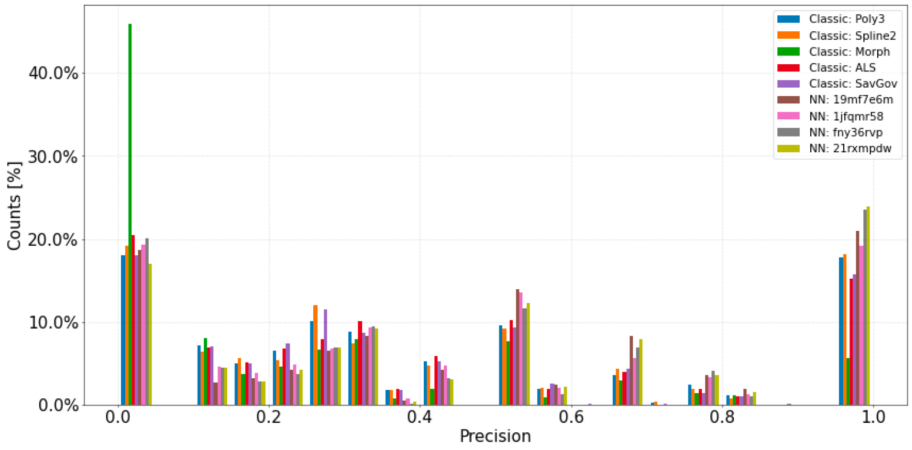
Neural Network Without SSIM = NN:1in4g6sl Neural Network with the SSIM in the loss Formula 6.8 = NN:kay219ht Neural Network with the SSIM in the loss Formula 6.9 = NN:1jeyfbzw
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


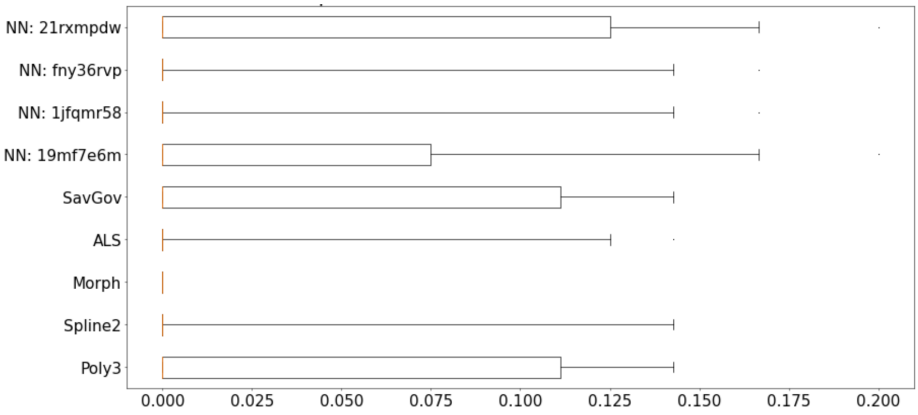
Table 6.31. Comparison metrics applied to neural network with SSIM in the second phase of loss

Neural Network Without Third Phase Loss = NN:19mf7e6m
Neural Network with the SSIM in the loss Formula 6.7 = NN:fny36rvp
Neural Network with the SSIM in the loss Formula 6.8 = NN:1jfqmr58
Neural Network with the SSIM in the loss Formula 6.9 = NN:21rxmpdw

Precision of Detected Peaks



Precision of the Worst 0.25% of the Test Set



Results Obtained from the Other Metrics

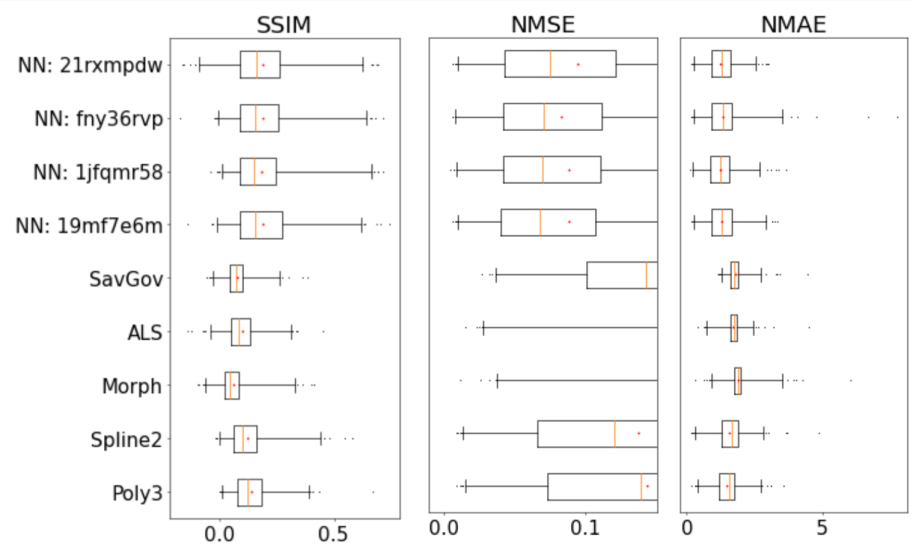


Table 6.32. Comparison metrics applied to neural network with SSIM in the third loss stage and with High Noise dataset

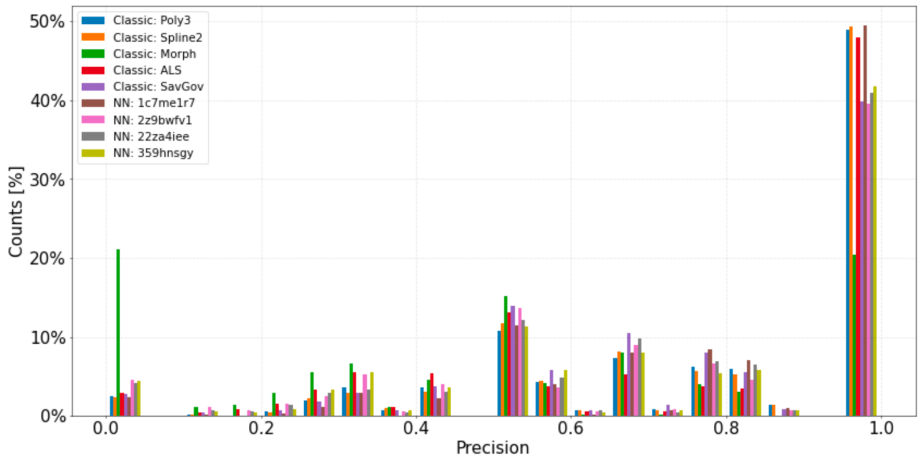
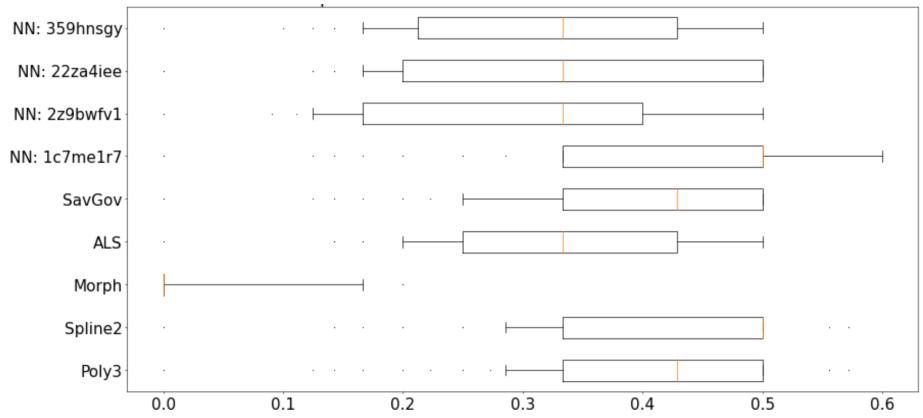
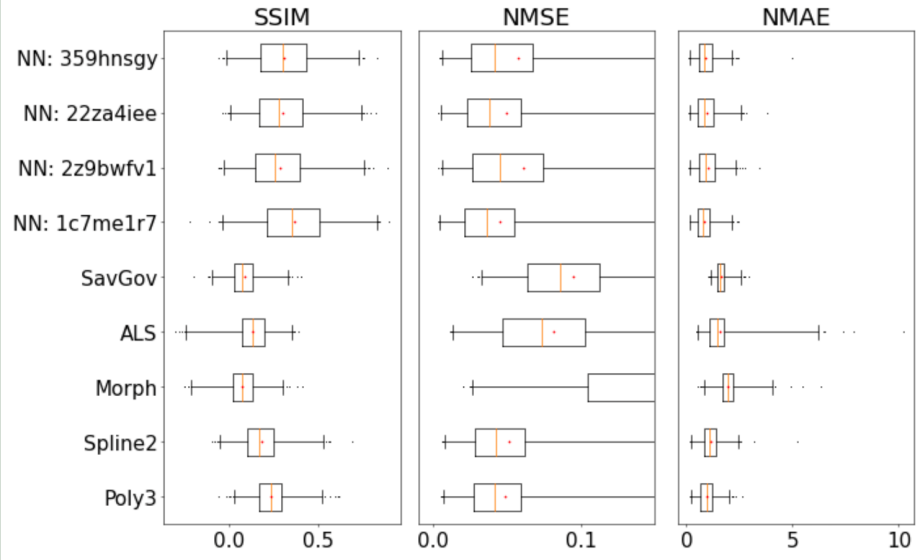
Neural Network Without Third Phase Loss = NN:1c7me1r7 Neural Network with the SSIM in the loss Formula 6.7 = NN:359hnsgy Neural Network with the SSIM in the loss Formula 6.8 = NN:2z9bwfv1 Neural Network with the SSIM in the loss Formula 6.9 = NN:22za4iee
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


Table 6.33. Comparison metrics applied to neural network with SSIM in the third loss stage and with Low Noise dataset

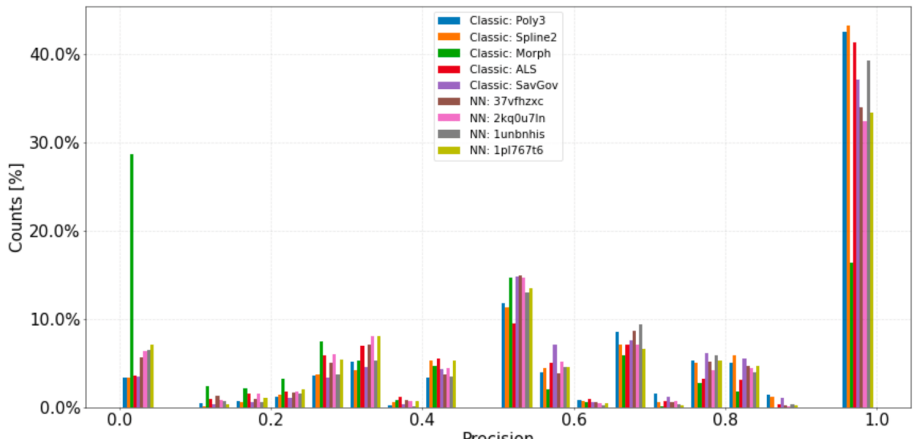
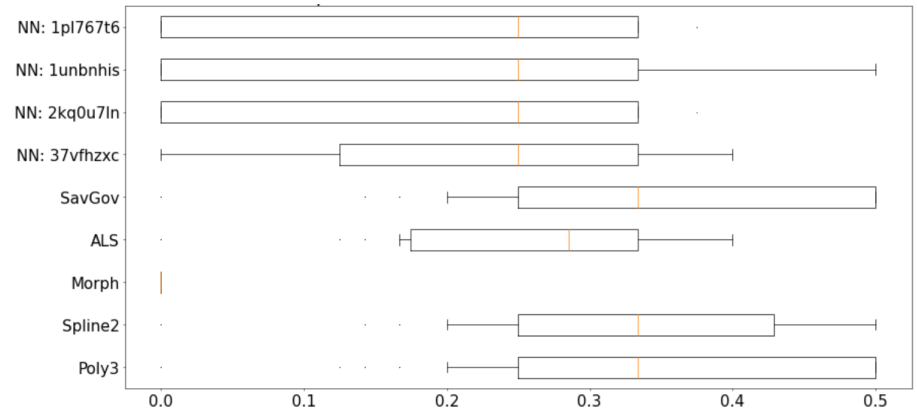
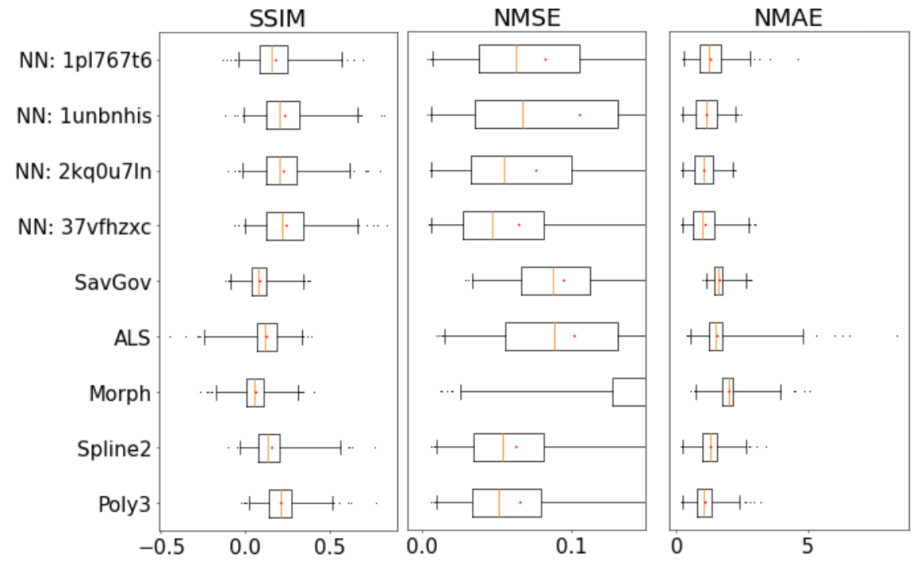
Neural Network Without Third Phase Loss = NN:37vfhzxc Neural Network with the SSIM in the loss Formula 6.7 = NN:1pl767t6 Neural Network with the SSIM in the loss Formula 6.8 = NN:2kq0u7ln Neural Network with the SSIM in the loss Formula 6.9 = NN:1unbnhis
Precision of Detected Peaks

Precision of the Worst 0.25% of the Test Set

Results Obtained from the Other Metrics


Table 6.34. Comparison metrics applied to neural network with SSIM in the third loss stage and with Mixed Noise dataset

Chapter 7

Conclusions

In conclusion, the aim of this work was to show that spectral analysis of stimulated Raman data can benefit from the application of deep learning denoisers, because of the complexity of line shapes and noise in this type of spectroscopy. For this purpose, the SRSDenoiser network was developed and a study was carried out on the generalizability of its training on datasets of different types and the parameters most sensitive to different types and amounts of noise, to finally identify the optimal parameters under various signal-noise conditions and new optimization strategies, involving not only the architecture of the network but also the training methods. The complex intrinsic nature of the noise present in the Raman spectrum required several arrangements during the creation of the neural network-based algorithm, but through in-depth optimization efforts it was possible to achieve excellent results under different noise conditions.

It was shown how a neural network characterized by parallel branches and a residual structure achieved high performance with a structure that was not overly complex and heavy. The parallel nature of the network proved to be of paramount importance in performing excellent signal denoising. In fact, it was the combination of what was learned from each branch that led to the excellent results shown, since the conditions and quantities under which noise occurs required different amounts of parallel branches. In addition, the residual structure also proved to be of great importance.

In fact, regardless of network size or amount of parameters, the trainings never overfitted or showed training issues related to network size. This means that this feature allowed the creation of a more complex model while avoiding the problems just mentioned.

Based on the error metrics and similarity of the signals, the results obtained by the neural network are better in every aspect with respect to the algorithms that do not make use of neural networks. The only metric that sometimes shows slightly better results for the latter type of algorithms is the accuracy of the peaks detected when dealing with the Low Noise and Mixed Noise datasets, that is, when the noise along the signal is small and the relative intensity between the peaks and the baseline is variable. This aspect, however, can be justified by the fact that these algorithms rely primarily on correcting general baseline trends and repositioning peaks, but are not optimized to perform denoising along the entire signal, especially if there is a large amount of noise. In fact, this is the reason why the other signal comparison metrics show a larger difference in the totality of the predicted signal than the corrected signal when it comes to the algorithms that do not make use of neural networks compared to the developed artificial intelligence model. Furthermore, when noise types occur with high intensity as in the case of the High Noise dataset, no other algorithm achieves the same performance as the neural network. Therefore, since it is of crucial importance to perform proper denoising of the entire signal, neural networks remain the best choice to accomplish this task.

Concerning the goal of creating a single model capable of performing Raman spectrum denoising, it was not possible due to the number of parallel branches. In fact, as mentioned earlier, the conditions and quantities under which noise occurs require a different number of parallel branches, which results in a different number and settings of kernels. In fact, as much as the various combinations of parallel branches tried have yielded similar results in each dataset, it is always necessary to adopt the best-case setting, which varies from dataset to dataset. Moreover, even if the same number of parallel branches were obtained for each dataset, the weights might

be different with respect to the dataset. The combination of these two factors, which seem to be most sensitive to the noise under consideration, are the real reason why a unique model cannot be created. To overcome this obstacle, further studies on parallel branches could be carried out. For example, the implementation of continuous kernels could be evaluated. In the current algorithm, the size of the kernel is determined before training begins and is cleared based on the number of parallel branches. This network characteristic is of fundamental importance for learning input features. Recent work has demonstrated how neural networks can improve their performance if the kernel size varies across different layers [54, 55], but exploring all possible combinations is unfeasible in practice. The best possibility would be to learn the correct kernel size during the training itself. In this way, it is possible to create a network with a fixed number of parallel branches and allow the network to autonomously change only the kernel size according to the data set under consideration. Moreover, an additional aspect that could be further evaluated is the inclusion of SSIM metrics in the validation loss of the third training phase. In fact, it improves performance in the case of the High Noise dataset, so this implies that further studies could be carried out in order to understand how to properly include and weight its influence in the validation loss. For example, its inclusion could be re-evaluated already just after applying continuous kernels in case the latter improves performance by themselves. In fact, the addition of this network function alone could help to better exploit the contribution of SSIM metrics in loss validation. Otherwise, since the idea undertaken could lead to performance improvements, another possibility might be to evaluate other metrics similar to SNR and SSIM to be included in the loss.

Once optimized, following the strategies presented in this thesis, such neural network-based models would exhibit high versatility to the types of noise and the amount with which they occur in the signal, thus achieving very high performance. These results would pave the way to the introduction of a new state-of-the-art approach for denoising the Raman spectrum.

Bibliography

[1] Chandrasekhara Venkata Raman. A new radiation. *Indian Journal of physics*, 2:387–398, 1928.

[2] Katharina Eberhardt, Clara Stiebing, Christian Matthäus, Michael Schmitt, and Jürgen Popp. Advantages and limitations of raman spectroscopy for molecular diagnostics: an update. *Expert review of molecular diagnostics*, 15(6):773–787, 2015.

[3] Spettroscopia raman. https://www.mt.com/it/it/home/applications/L1_AutoChem_Applications/Raman-Spectroscopy.html. accessed 29 Oct, 2022.

[4] Guide to raman spectroscopy. https://www.bruker.com/en/products-and-solutions/infrared-and-raman/raman-spectrometers/what-is-raman-spectroscopy.html. accessed 29 Oct, 2022.

[5] Andrew R. Barron Pavan M. V. Raja. Raman and surface-enhanced raman spectroscopy. https://chem.libretexts.org/Bookshelves/Analytical_Chemistry/Physical_Methods_in_Chemistry_and_Nano_Science_(Barron)/04%3A_Chemical_Speciation/4.03%3A_Raman_Spectroscopy. accessed 30 Oct, 2022.

[6] Synthesis of germanium-tin alloys by ion implantation and pulsed laser melting: Towards a group iv direct band gap semiconductor - scientific figure on researchgate. https://www.researchgate.net/figure/Raman-spectrum-o

f-CCl4-4880-nm-excitation-The-full-spectrum-includes-a-Rayleigh_fig21_320426698. accessed 14 Nov, 2022.

[7] infn. Raman spectrum. https://agenda.infn.it/event/14815/contributions/26662/attachments/19038/21539/Tera_Hertz__10_aprile_frascati.pdf. accessed 15 Nov, 2022.

[8] Philipp Kukura, Sangwoon Yoon, and Richard A Mathies. Femtosecond stimulated raman spectroscopy. https://pubs.acs.org/doi/pdf/10.1021/ac0694501, 2006. accessed 17 Nov, 2022.

[9] Giuseppe Fumero, Giovanni Batignani, Konstantin E Dorfman, Shaul Mukamel, and Tullio Scopigno. On the resolution limit of femtosecond stimulated raman spectroscopy: Modelling fifth-order signals with overlapping pulses. *ChemPhysChem*, 16(16):3438–3443, 2015.

[10] Yanru Bai and Quan Liu. Denoising raman spectra by wiener estimation with a numerical calibration dataset. *Biomedical Optics Express*, 11(1):200–214, 2020.

[11] Matt Dyer Ben Byer Max Lambert, Andrew Engroff. Empirical mode decomposition. https://www.clear.rice.edu/elec301/Projects02/empiricalMode/index.html, 2018. accessed 18 Nov, 2022.

[12] Fabiola León-Bejarano, Miguel Ramírez-Elías, Martin O Mendez, Guadalupe Dorantes-Méndez, Ma del Carmen Rodríguez-Aranda, and Alfonso Alba. Denoising of raman spectroscopy for biological samples based on empirical mode decomposition. *International Journal of Modern Physics C*, 28(09):1750116, 2017.

[13] Robertas Damaševičius, Christian Napoli, Tatjana Sidekerskienė, and Marcin Woźniak. Imf mode demixing in emd for jitter analysis. *Journal of Computational Science*, 22:240–252, 2017.

[14] Hans FM Boelens, Paul HC Eilers, and Thomas Hankemeier. Sign constraints improve the detection of differences between complex spectral data sets: Lc- ir as an example. *Analytical chemistry*, 77(24):7998–8007, 2005.

[15] Vitaly I Korepanov. Asymmetric least-squares baseline algorithm with peak screening for automatic processing of the raman spectra. *Journal of Raman Spectroscopy*, 51(10):2061–2065, 2020.

[16] Sergio Oller-Moreno, Antonio Pardo, Juan Manuel Jiménez-Soto, Josep Samitier, and Santiago Marco. Adaptive asymmetric least squares baseline estimation for analytical instruments. In *2014 IEEE 11th International Multi-Conference on Systems, Signals & Devices (SSD14)*, pages 1–5. IEEE, 2014.

[17] Juan José González-Vidal, Rosanna Pérez-Pueyo, and María José Soneira. Automatic morphology-based cubic p-spline fitting methodology for smoothing and baseline-removal of raman spectra. *Journal of Raman Spectroscopy*, 48(6):878–883, 2017.

[18] Liangrui Pan, Pronthep Pipitsunthonsan, Peng Zhang, Chalongrat Daengngam, Apidach Booranawong, and Mitchai Chongcheawchamnan. Noise reduction technique for raman spectrum using deep learning network. In *2020 13th International Symposium on Computational Intelligence and Design (ISCID)*, pages 159–163. IEEE, 2020.

[19] Sinead Barton, Salaheddin Alakkari, Kevin O'Dwyer, Tomas Ward, and Bryan Hennelly. Convolution network with custom loss function for the denoising of low snr raman spectra. *Sensors*, 21(14):4623, 2021.

[20] Pedram Abdolghader, Andrew Ridsdale, Tassos Grammatikopoulos, Gavin Resch, François Légaré, Albert Stolow, Adrian F Pegoraro, and Isaac Tamblyn. Unsupervised hyperspectral stimulated raman microscopy image enhancement: denoising and segmentation via one-shot deep learning. *Optics Express*, 29(21):34205–34219, 2021.

[21] Bryce Manifold, Elena Thomas, Andrew T Francis, Andrew H Hill, and Dan Fu. Denoising of stimulated raman scattering microscopy images via deep learning. *Biomedical optics express*, 10(8):3860–3874, 2019.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[23] Irem Loc, Ibrahim Kecoglu, Mehmet Burcin Unlu, and Ugur Parlatan. Raman signal denoising using fully convolutional encoder decoder network. *bioRxiv*, 2022.

[24] David Breakey and Craig Meskell. Comparison of metrics for the evaluation of similarity in acoustic pressure signals. *Journal of Sound and Vibration*, 332(15):3605–3609, 2013.

[25] scikit learn. Module: Metrics. https://scikit-image.org/docs/stable/api/skimage.metrics.html. accessed 29 Oct, 2022.

[26] Kevin Liao. Alternating least square (als) matrix factorization in collaborative filtering. https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1, November 2018. accessed 25 Nov, 2022.

[27] Jianhua Zhao, Harvey Lui, David I McLean, and Haishan Zeng. Automated autofluorescence background subtraction algorithm for biomedical raman spectroscopy. *Applied spectroscopy*, 61(11):1225–1232, 2007.

[28] Chad A Lieber and Anita Mahadevan-Jansen. Automated method for subtraction of fluorescence from biological raman spectra. *Applied spectroscopy*, 57(11):1363–1367, 2003.

[29] pybaselines documentation. Polynomial baselines. https://pybaselines.readthedocs.io/en/stable/algorithms/polynomial.html. accessed 25 Nov, 2022.

[30] Ronald W Schafer. What is a savitzky-golay filter?[lecture notes]. *IEEE Signal processing magazine*, 28(4):111–117, 2011.

[31] scikit learn. Baseline-determination. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html. accessed 25 Nov, 2022.

[32] Vitaly Korepanov (github nickname "the-different name"). Baseline subtraction algorithms including als-based (als, psalsa and derpsalsa), morphological and wavelet-based. https://github.com/the-different-name/spectral_baseline/blob/master/der_baseline.py. accessed 29 Oct, 2022.

[33] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559, 2003.

[34] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[35] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[40] IBM Cloud Education. Convolutional neural networks. https://www.ibm.com/cloud/learn/convolutional-neural-networks, October 2020. accessed 30 Nov, 2022.

[41] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[42] Jilin Zhang, Junfeng Xiao, Jian Wan, Jianhua Yang, Yongjian Ren, Huayou Si, Li Zhou, and Hangdi Tu. A parallel strategy for convolutional neural network based on heterogeneous cluster for mobile information system. *Mobile Information Systems*, 2017, 2017.

[43] Xiaoya Chen, Baoheng Xu, and Han Lu. Effects of parallel structure and serial structure on convolutional neural networks. In *Journal of Physics: Conference Series*, volume 1792, page 012074. IOP Publishing, 2021.

[44] Al-Mustafa Khafaji. Parallel convolutional neural network architectures for improving misclassifications of perceptually close images. https://www.diva-portal.org/smash/get/diva2:1526942/FULLTEXT01.pdf, 2020. accessed 02 Dec, 2022.

[45] Dive Into Deep Learning. Residual networks (resnet) and resnext. https://d2l.ai/chapter_convolutional-modern/resnet.html. accessed 02 Dec, 2022.

[46] Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[47] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[48] Sivaram T. All you need to know about skip connections. https://www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections/, October 2021. accessed 03 Dec, 2022.

[49] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

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

[51] techpowerup. Nvidia tesla v100 sxm2 32 gb. https://www.techpowerup.com/gpu-specs/tesla-v100-sxm2-32-gb.c3185. accessed 05 Dec, 2022.

[52] Amirhossein Kazemnejad. How to do deep learning research with absolutely no gpus. https://kazemnejad.com/blog/how_to_do_deep_learning_research_with_absolutely_no_gpus_part_2/, August 2019. accessed 05 Dec, 2022.

[53] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[54] David W Romero, Robert-Jan Bruintjes, Jakub M Tomczak, Erik J Bekkers, Mark Hoogendoorn, and Jan C van Gemert. Flexconv: Continuous kernel

convolutions with differentiable kernel sizes. *arXiv preprint arXiv:2110.08059*, 2021.

[55] Silvia L Pintea, Nergis Tömen, Stanley F Goes, Marco Loog, and Jan C van Gemert. Resolution learning in deep convolutional networks using scale-space theory. *IEEE Transactions on Image Processing*, 30:8342–8353, 2021.