Universidad Nacional de Educación a Distancia

Escuela Técnica Superior de Ingeniería Informática

Proyecto de fin de Grado en Ingeniería Informática

Towards spectral super-resolution in solar absorption lines from pseudomonochromatic images

Fernando H. Nasser-Eddine López

Dirigido por: Pablo Rodríguez Gil

Codirigido por: Luis Manuel Sarro Baro

Curso 2023/2024, convocatoria junio

Towards spectral super-resolution in solar absorption lines from pseudomonochromatic images

Proyecto de fin de Grado en Ingeniería Informática

de modalidad externa

Realizado por: Fernando H. Nasser-Eddine López

Dirigido por: Pablo Rodríguez Gil

Codirigido por: Luis Manuel Sarro Baro

Fecha de lectura y defensa: 26 junio de 2024

Agradecimientos

A Andrés Asensio, que fue el verdadero artista detrás de la propuesta.
A Luis Sarro, por su apoyo y comprensión, en lo académco y en lo personal.
A mi amigo Pablo Rodríguez por su excepcional mentoría y su infinita paciencia.
A Laura y Jimena por iluminar cada minuto de mi vida.

Resumen

En este proyecto de fin de grado (PFG), exploramos la viabilidad de aplicar técnicas modernas de aprendizaje profundo para abordar el desafío de la superresolución espectral. Nos centramos en un conjunto de 21 imágenes de alta resolución que representan diferentes longitudes de onda en la línea de absorción de hierro neutro(Fe I λ6302) en 6302 Å de la fotosfera del Sol.

Partiendo del estado del arte en técnicas de regresión de imágenes, tales como *SINusoidal REpresentation Networks* (SIREN) y *Fourier Features*, nuestro primer objetivo fue seleccionar la arquitectura que ofreciera la mejor aproximación de nuestras imágenes solares utilizando la métrica de la relación señal-ruido (PSNR). Tras lograr resultados comparables con estos estudios, dimos un paso adicional y extendimos la funcionalidad de la arquitectura al espacio completo de nuestro conjunto de datos incluyendo la longitud de onda para cada píxel de nuestras imágenes, es decir, *I(x, y, λ)*. Esto nos permitió generar imágenes en longitudes de onda intermedias a las 21 originales usando un *neural field*.

En este PFG no solo confirmamos la eficacia de las técnicas de aprendizaje profundo en la aproximación de imágenes pseudomonocromáticas de alta calidad, sino que también extendemos estos métodos al dominio de la superresolución espectral. Al hacerlo, contribuimos a futuras investigaciones que podrían mejorar la forma en que capturamos y procesamos datos espectrales para su uso en multitud de campos desde la astrofísica a la biomedicina.

Palabras clave

Física solar, procesamiento de imágenes, superresolución espectral, inteligencia artificial, visión artificial, aprendizaje profundo, redes neuronales, campos neurales, reconstrucción de imágenes, regresión de imágenes, redes implícitas sinusoidales (SIREN), características de Fourier.

Abstract

In this Bachelor's Degree Final Project (PFG), we explored the feasibility of applying modern deep learning techniques to address the challenge of spectral super-resolution. We focused on a set of 21 high-resolution images representing different wavelengths in the absorption line of neutral iron (Fe I λ6302) at 6302 Å of the Sun's photosphere.

Starting from the state-of-the-art in image regression techniques, such as *SINusoidal REpresentation Networks* (SIREN) and *Fourier Features*, our first objective was to select the architecture that offered the best approximation of our solar images using the signal-to-noise ratio (PSNR) metric. After achieving comparable results with these studies, we took an additional step and extended the functionality of the architecture to the complete space of our dataset including the wavelength for each pixel of our images, i.e., *I(x, y, λ)*. This allowed us to generate images at intermediate wavelengths to the original 21 using a *neural field*.

In this PFG, we not only confirmed the efficacy of deep learning techniques in the approximation of high-quality pseudomonochromatic images but also extended these methods to the domain of spectral super-resolution. By doing so, we contribute to future research that could improve the way we capture and process spectral data for use in a multitude of fields from astrophysics to biomedicine.

Keywords

Solar physics, image processing, spectral super-resolution, artificial intelligence, computer vision, deep learning, neural networks, neural fields, image reconstruction, image regression, sinusoidal representation networks (SIREN), Fourier features.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The absorption lines we observe in the Sun are powerful diagnostic tools for understanding the stratification of the medium where they originate. These lines, also known as the Fraunhofer spectrum, are a direct result of the interaction between the light emitted by the Sun and the different elements present in its atmosphere. When light passes through these layers of hot gas, certain specific wavelengths are absorbed by the electrons of its atoms, preventing them from reaching us. Each chemical element possesses a unique set of absorption lines at specific wavelengths that correspond to the particular energy levels of its electrons. The detection and identification of these lines (Fig. 1.1) thus allow us to recognize the elements present and understand the physical conditions and processes taking place in the Sun.

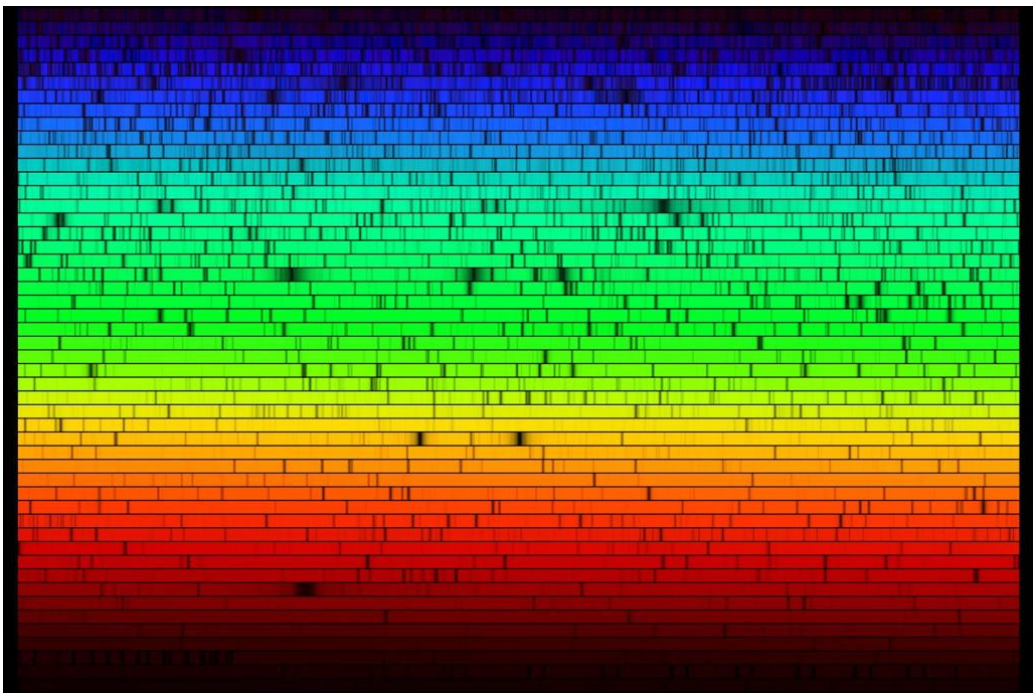


Figure 1.1: Spectral absorption lines in the solar spectrum.

The lines are darker than their surroundings because the temperature at the level of the atmosphere where they form is lower than the temperature of 5800 K at the base of the photosphere, where most of the continuous emission originates. In fact, the formation of an absorption line can also be understood in terms of the decrease in temperature of the local source function (amount of light emitted from a specific point and in a particular direction) towards the center of the line, as shown in Fig. 1.2. As we move towards the surface, the line becomes increasingly stronger, that is, deeper. Its evolution from the background to the surface would be similar to that represented in Fig. 1.3.

The opacity of this medium, κ_ν, can depend on the wavelength, and accounts for its capacity to *stop* the incident photons that come from deeper areas of the Sun. For the case of a uniform

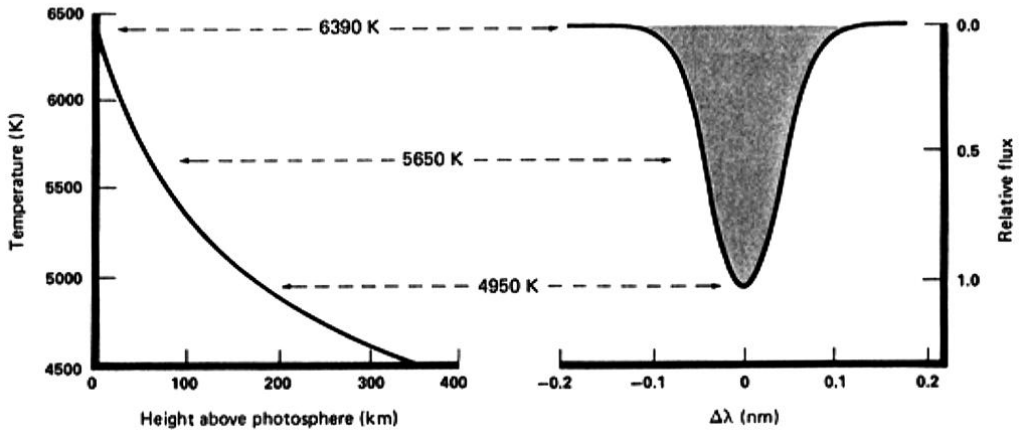


Figure 1.2: Temperature variation in the photosphere and formation of an absorption line.

absorption of the incident intensity, I_0, over a determined distance within the medium:

$$I = I_0 \mathrm{e}^{-\tau_\nu} \ , \tag{1.1}$$

with τ_ν being the optical depth, which depends on κ_ν. The smaller τ_ν is, the deeper we can see through the medium.

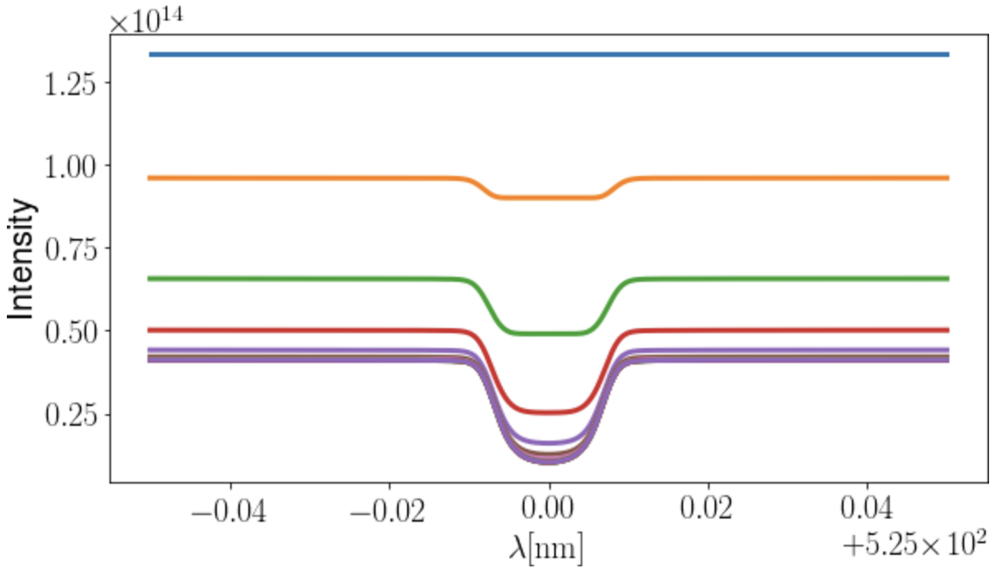


Figure 1.3: From top to bottom, variation of the absorption amount as we ascend in the solar photosphere.

Under the solar photosphere, the gas is dense enough and the interactions among photons, electrons, and ions common enough that radiation cannot directly escape into space. In the photosphere, however, the likelihood of a photon escaping without further interaction with matter depends on its energy. If that energy corresponds to some electronic transition of one of the atoms or ions present in the gas, then the photon may be absorbed before it can travel farther: the more elements of the right type for absorption there are, the lower the escape probability. Conversely, if the photon's energy does not match any of those transitions, then the photon cannot interact further with the gas and can escape from the Sun. Therefore, as illustrated in Fig. 1.4, when we look at the Sun, we are actually looking inward into its

photosphere, to a depth that depends on the wavelength of light considered. Photons with wavelengths far removed from any characteristic absorption tend to come from the depths of the photosphere, while those that form the centers of the absorption lines—with a higher likelihood of interacting with the matter as they travel through the solar gas—escape primarily from the outer and cooler levels.

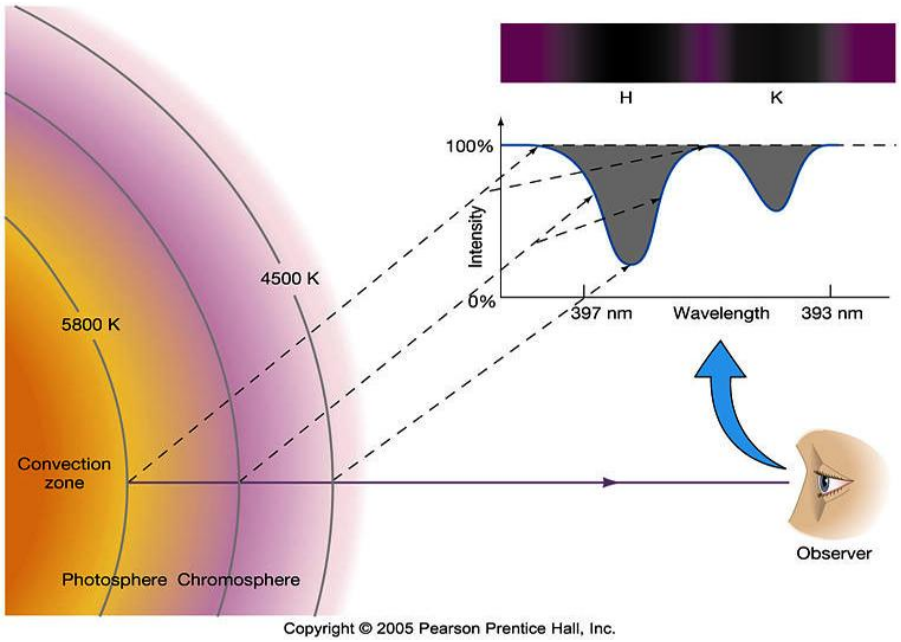


Figure 1.4: Scheme of the formation of the H and K absorption lines of Ca II. Photons with energies close to an electronic transition have a higher probability of being absorbed before escaping, so those that reach Earth come from higher and cooler levels of the solar photosphere.

1.1 Three-dimensional spectroscopy of the Sun

Let's now consider an (artificial) image of a certain region of the Sun (Fig. 1.5), and suppose that for each pixel in the image, we also have its spectrum, that is, we have a 3D dataset, $I(x, y, \lambda)$, with λ being the wavelength. The question is: what does the difference between the spectrum of a brighter pixel (hotter area) and that of a darker one (cooler area) in the images tell us? The different levels of intensity at different wavelengths speak to us of the stratification of temperature in the solar photosphere. Additionally, the granules we see in Fig. 1.5 are hotter the deeper they are, and cooler towards the surface. Therefore, using inversion methods, we could retrieve the variation of different physical parameters according to depth, such as chemical composition, temperature, pressure, or density.

1.2 Motivation and objectives

Precise imaging at different wavelengths is a fundamental necessity in multiple fields of science, from astrophysics to biomedicine. However, traditional techniques generally provide us with spectral data at discrete wavelengths. With this Bachelor's Degree Final Project (from this

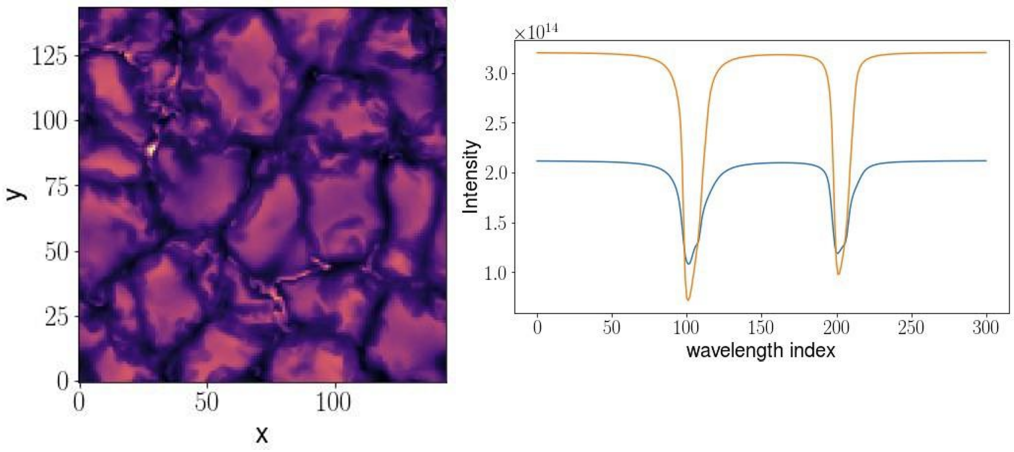


Figure 1.5: Synthetic image of a region of the Sun showing solar granulation (left). For each of its pixels, a spectrum is also available (right). As an example, the spectrum, containing two absorption lines, of a bright pixel (yellow) and a dark one (blue) is shown.

point forward, PFG), we aim to bridge the gaps between these discrete wavelengths to obtain more complete and detailed information. In particular, in this project, we will focus on obtaining a spectral interpolation between images captured at different wavelengths of a specific area of the Sun's surface. These images have been obtained with the help of a Fabry-Perot interferometer (Fig.1.6).

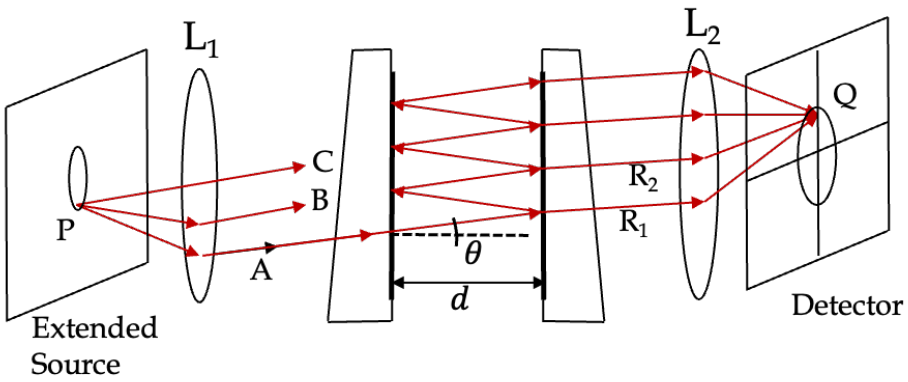


Figure 1.6: Setup of a Fabry-Perot interferometer. Rays coming from a point on the source P emerge parallel after passing through the converging lens L_1. These rays A, B, C enter with the same angle of inclination θ into the space between the two partially reflective mirrors separated by a distance d. Upon exit, the rays $(R_1, R_2, \dots)$ are directed to the focal plane of the lens L_2 and converge at the same point Q on the screen or detector.

A Fabry-Perot interferometer (etalon) [1] is an optical device used in solar spectrography to analyze the light emitted by the Sun and obtain detailed information about its composition, temperature, density, rotation speed, and other physical magnitudes from spectra. It consists of two parallel mirrors, forming a cavity where the light is repeatedly reflected, creating multiple beams that interfere with each other. This interference produces a pattern that can be analyzed to determine its wavelength.

A key feature of etalons is their high spectral resolution, meaning they can distinguish between very closely spaced wavelengths of light. Solar telescope filter spectrographs, such as the TESOS *Triple Etalon Solar Spectrograph* [2, 3] at the Teide observatory, use them as narrowband filters

to provide high-resolution pseudomonochromatic images obtained by taking measurements at specific, predefined points along a spectral line.

Depending on the configuration in which they are used, they can produce deviations in the wavelength of the filter at each pixel. Generally, calibrations are conducted to adjust for these effects, but an interesting possibility to explore is to use these shifts to artificially produce images with higher spectral resolution, as if many more narrowband filters were used along the same absorption line. To do this, it is first necessary to explore the possibility of using machine learning to approximate the observed images. We will do this using neural fields, also known as implicit neural representations [4].

A *neural field* applied to a pixelated image consists of a neural network that takes as inputs the coordinates (x, y) of the pixel and outputs its RGB value (or any other encoding). When a trained *neural field* is asked to provide the output for all pixels, the image is reconstructed. A *neural field* is, therefore, a continuous and differentiable function that approximates a function defined in a space, in our case $(I(x, y))$.

In recent years, modern techniques such as SIREN [5] (*SINusoidal REpresentation Networks*) and Fourier features [6] have emerged, demonstrating very good results in the regression of two-dimensional images from their spatial coordinates. The publications in which they are introduced describe superior performance for this specific task compared to other techniques, particularly in terms of the signal-to-noise ratio (PSNR) metric used to determine the quality of image reproduction [7]. We will start from these experiments that have already proven their effectiveness and reproduce them with our data to subsequently attempt to tackle the problem of spectral interpolation.

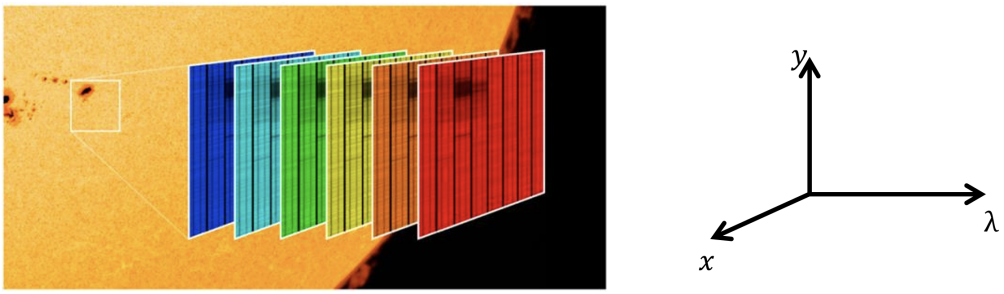


Figure 1.7: Illustrative example of a cube of six images of a region of the solar photosphere at six different wavelengths.

For this PFG, we have a cube of real spectral data $I(x, y, \lambda)$ from the photospheric absorption line of neutral (i.e., not ionized) iron centered at 6302 Å (Fe I λ6302). The data come from the *Swedish Solar Telescope*, in La Palma and were collected using the CRISP (*CRisp Imaging SpectroPolarimeter*) instrument [8]. Specifically, there are 21 pseudomonochromatic images of 966×964 pixels each of a large area of the Sun's surface. The instrument's resolution is $0.057 arcsec/pixel$. An *arcsec* corresponds, on the surface of the Sun, to about $725 km$. Thus, each image is approximately $55.1 \times 54.9 arcsec$, which are about $39.9 \times 39.8 Mm$. These images *scan* the Fe I λ6302 absorption line in 21 narrow wavelength intervals. Broadly speaking, we are facing a scheme similar to the one illustrated by Fig. 1.7, which presents six images from blue to red, but in our case focused on a single absorption line from the Sun's photosphere. Fig. 1.8 shows a schematic of the scanning of the neutral iron absorption line at 6302 Å, and

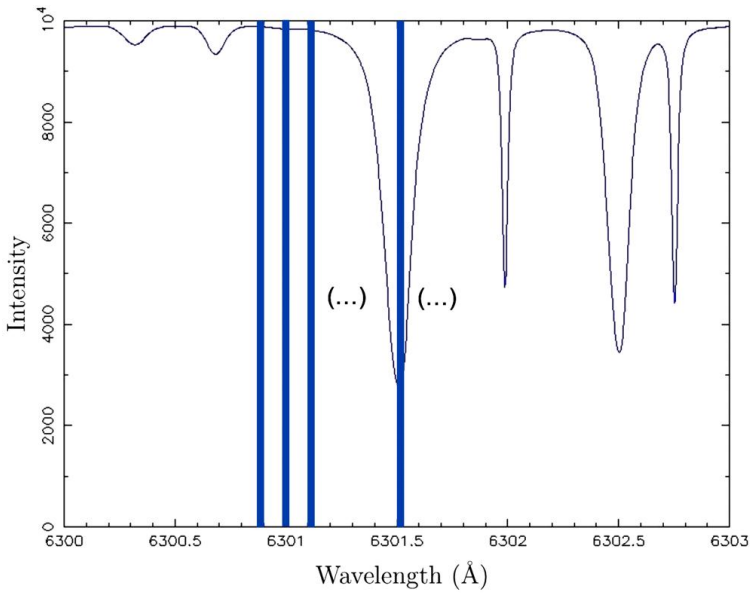


Figure 1.8: Each vertical line indicates one of the 21 pseudomonochromatic spectral bands in which the Fe I absorption line $\lambda 6302$ is scanned.

Fig. 1.9 three of the 21 images we will work with.

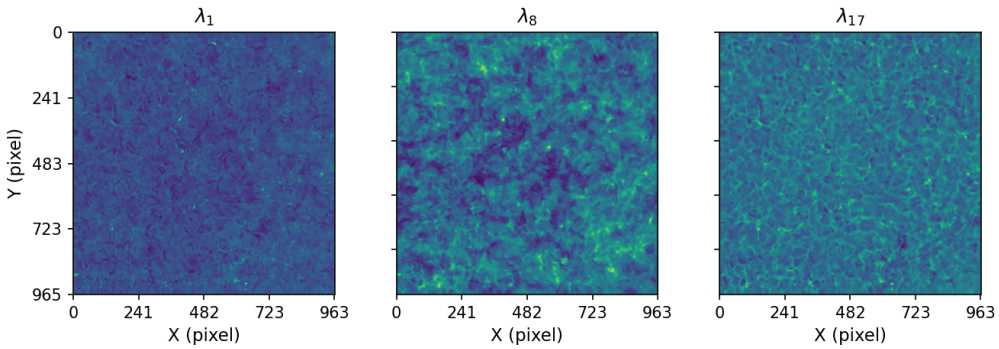


Figure 1.9: Three of the 21 images from the scan of the Fe I $\lambda 6302$ absorption line in the same region of the Sun. Note the different intensities in various parts of the line.

The first attempt will be to verify the architecture that best approximates the pseudomonochromatic images. If this is achieved, we will take the step to extend this approximation to the complete space including the wavelength, that is, $I(x, y, \lambda)$. This will allow us to verify that it is possible to compact the cube of 21 images into the weights of a neural network. The training is relatively simple: it only requires *fully connected* networks and *stochastic gradient* optimizers, widely known in the field of *deep learning*. Such an experiment has limited scope, except for the fact that we can interpolate between images with total freedom. The final goal is to verify if we can use this approach to achieve super-resolution using a *neural field*.

Chapter 2

Neural fields

2.1 Fields

The concept of the field in physics was primarily developed by Michael Faraday in the 19th century during his experimental works on electromagnetism. Faraday introduced the idea of lines of force to represent the electric and magnetic fields [9]. This visual representation provided an intuitive way to understand how forces could be transmitted through space, which contradicted the dominant theories of the time that held that forces could only be transmitted by direct contact.

Faraday's vision of fields was formalized by James Clerk Maxwell in his famous equations, presented in 1864 [10], which describe how electric and magnetic fields operate and on which classical electromagnetic theory is based.

The concept of field in physics has extended beyond electromagnetism to encompass a wide variety of physical phenomena. It now includes gravitational fields (described by Einstein's theory of general relativity [11]) and quantum fields that underlie particle physics. Within quantum field theory, propelled by Dirac [12], strong and weak interactions, which are fundamental in particle physics, are described and represented by their respective quantum fields. In all these contexts, a field is generally understood as a physical entity that has a value at every point in space and time.

The concept of a field is addressed in countless physics books, but as a conceptual synthesis, it can be described as a function that assigns to each point in space (and possibly time) a value of some physical magnitude. That is, it describes the way in which this magnitude changes or is distributed in space and time. The physical magnitude can be any measurable property, such as temperature, velocity, density, pressure, electrical charge, and even the intensity of a pixel. Depending on the physical quantity they represent, fields can be scalar, vectorial, or tensorial.

Therefore, fields are a fundamental tool for describing how physical magnitudes vary in space and time and how they affect the particles and objects in their surroundings.

2.2 Neural networks

Neural networks constitute a fundamental pillar in the rise of the field of artificial intelligence and machine learning. Inspired by the functioning of the human brain, neural networks are computing paradigms designed to mimic the way humans learn and process information. Since their inception in the 1940s, these networks have evolved considerably, outperforming conventional algorithms in a series of complex and varied tasks, ranging from natural language

processing [13] to image recognition [14] and text generation [15].

A neural network is composed of a series of computing elements, known as neurons. These are organized into different layers and are based on the mathematical model of the neuron proposed by Warren McCulloch and Walter Pitts in 1943 [16].

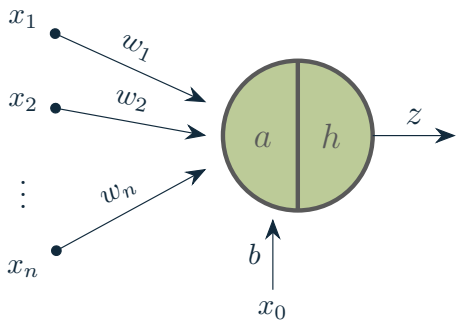


Figure 2.1: McCulloch-Pitts neural model including the set of input data $x_1, x_2, ..., x_n$ and their associated weights $w_1, w_2, ..., w_n$. The letter a corresponds to the weighted sum of the inputs plus the bias b. The letter h represents the transfer or activation function applied to the weighted sum to obtain the output z.

Each neuron receives a set of inputs $x_1, x_2, ..., x_N$, performs calculations, and passes the output to the neurons of the next layer. The inputs to a neuron can be the raw data of a problem or the outputs from neurons in the previous layers. The output z is calculated by applying an activation function h to the weighted sum of its inputs. The weights $w_1, w_2, ..., w_N$ that are applied to the inputs determine the strength of the connection and are the parameters that the network learns during training:

$$z = h(a) = h\left(\sum_{i=1}^{N} x_i w_i\right) \ . \tag{2.1}$$

Although it was not originally included, most modern models of neural networks add a bias term to each neuron. The bias, represented as b in Fig. 2.1, is a parameter that is added after the weighted sum of the inputs and provides an additional way to adjust the neuron's output, which can improve the model's ability to represent data and perform more accurate classification or prediction tasks:

$$z = h(a) = h\left(\sum_{i=1}^{N} x_i w_i + x_0 w_0\right) \ . \tag{2.2}$$

If we consider the input $x_0 = 1$ and the bias $b = w_0$ in Fig. 2.1, we can write equation (2.2) in a more compact form:

$$z = h(a) = h\left(\sum_{i=1}^{N} x_i w_i + b\right) = h\left(\sum_{i=0}^{N} x_i w_i\right) \ . \tag{2.3}$$

Finally, the activation function is a mathematical function that transforms the weighted sum of the inputs of a neuron. It introduces nonlinearity into the model, allowing the neural network

to learn and represent more complex relationships between the inputs and outputs. Currently, the most common activation functions are the following:

- **Step function**: it is the function used in the basic McCulloch-Pitts model. It produces a binary output (such as 0 or 1) based on a certain threshold (Fig. 2.2a):

$$h(a) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{if } a \geq 0 \end{cases} \tag{2.4}$$

- **Sigmoid function**: produces an output between 0 and 1. It has an S shape and its output can be interpreted as a probability (Fig. 2.2b):

$$h(a) = \frac{1}{1 + e^{-a}} \tag{2.5}$$

- **Hyperbolic tangent function (tanh)**: similar to the sigmoid function, but it produces an output between -1 and 1. This allows the output data to be centered around 0, which can improve the representation capacity and facilitates data normalization (Fig. 2.2c):

$$h(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{2.6}$$

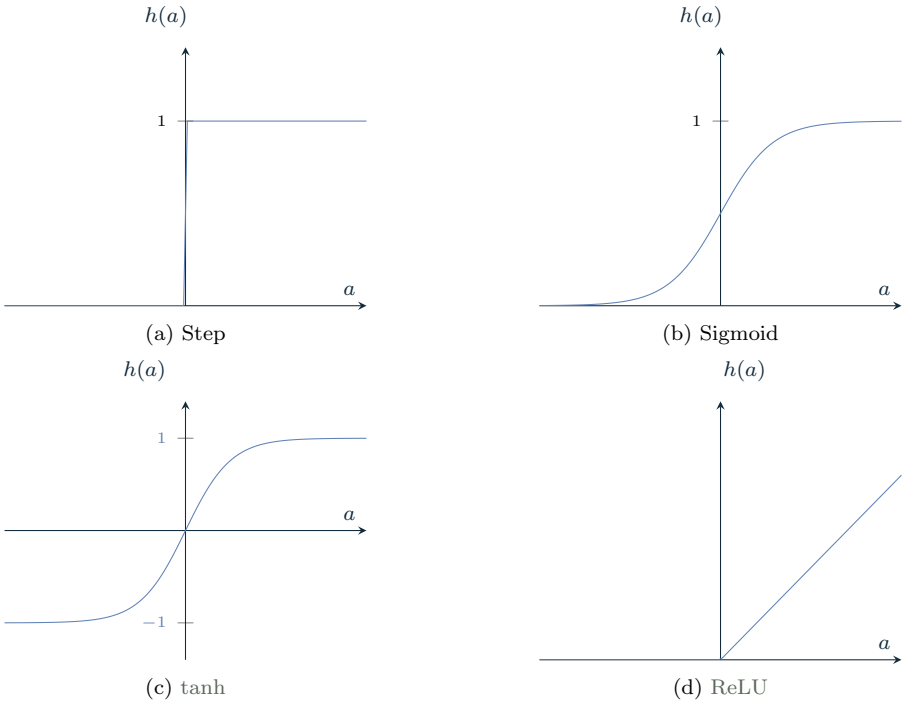
- **ReLU function (Rectified Linear Unit)**: it is the most used activation function in convolutional neural networks and backpropagation networks (Fig. 2.2d):

$$h(a) = \begin{cases} 0 & \text{si } a < 0 \\ a & \text{si } a \geq 0 \end{cases} \tag{2.7}$$

In its simplest form, a neural network consists of three layers: an input layer, a hidden layer, and an output layer (Fig 2.3). The **input layer**, which does not perform any processing by itself, transfers the information to the next layer, known as the **hidden layer**. The number of nodes (neurons) in the **input layer** generally corresponds to the number of features or attributes of the input data. The adequacy of these data for the network is one of the most critical aspects of deep learning, given that the quality and presentation of the input data can significantly influence the model's performance.

The **hidden layer** is the computational core of the neural network, where most of the processing is carried out. Composed of a defined number of neurons, the **hidden layer** processes the data received from the **input layer**. Through the calculations performed in these layers, the network manages to learn from the input data and improves the accuracy of its predictions. More complex and deeper networks may have multiple **hidden layers**.

Finally, the **output layer** is responsible for providing the final result generated by the neural network. Depending on the type of problem the network is designed to solve, this result can be a classification (for classification problems) or a specific value (for regression problems). The number of nodes (neurons) in the **output layer** usually corresponds to the number of classes or possible outcomes the network can produce.



(a) Step (b) Sigmoid

(c) tanh (d) ReLU

Figure 2.2: Most common activation functions h.

2.2.1 Training a neural network

The training of a neural network is an iterative process that seeks to optimize its weights and biases so that it can make accurate predictions. This process is carried out using a set of labeled training data, which provide examples of the inputs and the expected outputs. Below, the processes of obtaining results (forward pass) and adjusting the weights and biases (backpropagation) that take place in each iteration are described.

Forward pass

In each iteration of training, the neural network performs a series of calculations in a process known as *forward pass* or forward step. During this step, the inputs are multiplied by the weights and the biases are added at each neuron of the hidden and output layers. These weighted sums are then passed through the corresponding activation functions. Thus, the output expressions of the neurons of the hidden layer of the simple neural network illustrated in Fig. 2.3 in a forward step would be:

$$\begin{aligned} z_1 &= h(a_1) = h\left(\sum_{i=1}^{n} x_i w_{1,i} + b_1\right) \\ z_2 &= h(a_2) = h\left(\sum_{i=1}^{n} x_i w_{2,i} + b_2\right) \\ &\vdots \\ z_m &= h(a_m) = h\left(\sum_{i=1}^{n} x_i w_{m,i} + b_m\right) \ , \end{aligned} \tag{2.8}$$

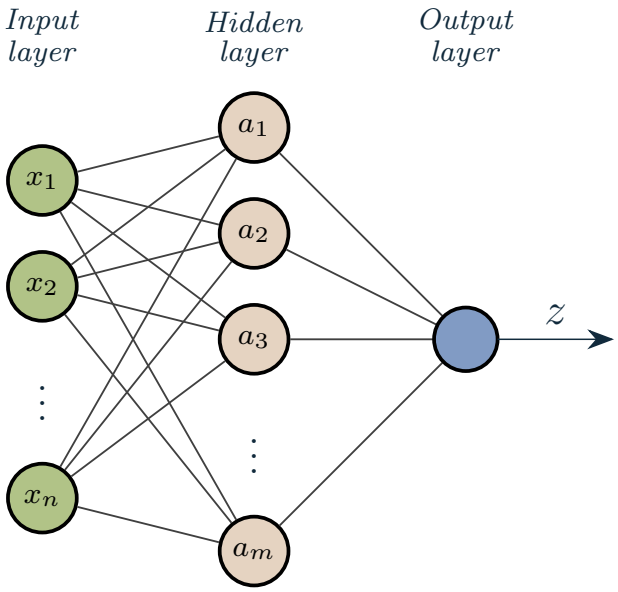


Figure 2.3: A simple configuration of a neural network that includes an input layer, a hidden layer, and an output layer.

which in matrix form is:

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = h\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) , \tag{2.9}$$

and which can be written as:

$$\mathbf{z} = h(\mathbf{W}\mathbf{x} + \mathbf{b}) , \tag{2.10}$$

This process propagates from the input layer to the output layer, producing an output that is the neural network's estimate based on the input data. Such output can be expressed mathematically as:

$$z_s = h_s\left(\sum_{i=1}^{m} z_i w_{s,i} + b_s \right) , \tag{2.11}$$

Which is:

$$z_s = h_s\left(\begin{bmatrix} w_{s,1} \\ w_{s,2} \\ \vdots \\ w_{s,m} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} + b_s \right) = h_s(\mathbf{w}_s\mathbf{z} + b_s) . \tag{2.12}$$

In the context of the regression problem, where the goal is to predict the pixel intensity from its spatial coordinates (x, y), the difference between the intensity value predicted by the neural network and the actual value is known as the error for that specific sample. To evaluate the effectiveness of the model across the entire training set, a loss function is used. In this case, one of the most common is the Mean Squared Error (MSE). This metric represents a penalty for the accumulated error across all samples in the training set, and it is sought to be minimized during the training process:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (z_{\text{true},i} - z_{s,i})^2 \ . \tag{2.13}$$

In this equation:

N represents the total number of samples,

$z_{\text{true},i}$ is the true label value corresponding to the i-th data entry,

$z_{s,i}$ is the prediction made by the neural network for that entry.

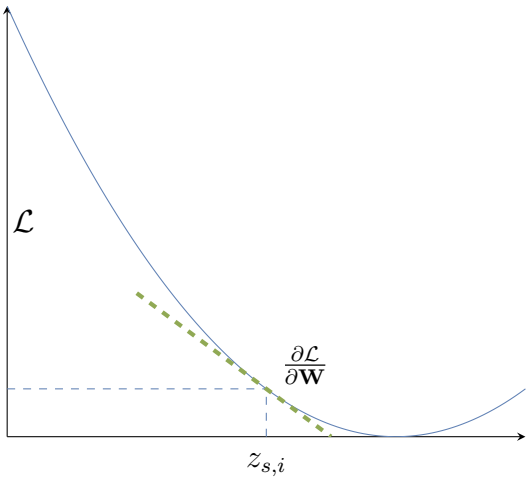


Figure 2.4: Evolution of the Loss Function $\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (z_{\text{true},i} - z_{s,i})^2$.

When $z_{s,i}$ is equal to the true value $z_{\text{true},i}$, the loss is 0, indicating that the prediction is correct. Conversely, as the prediction deviates from the true value, the loss increases. The slope of the loss function at the point $z_{s,i}$ is used in the gradient descent algorithm to update the model parameters (Fig. 2.4).

Due to the square in its formula, MSE more heavily penalizes predictions that deviate from the true value. However, it can also be sensitive to input values that significantly differ from the general pattern of the data, known as *outliers*. Therefore, the Mean Absolute Error (MAE) is sometimes used, which may be more appropriate if the dataset is expected to contain such values:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} |z_{\text{true},i} - z_{s,i}| \ . \tag{2.14}$$

Although MSE is also used in classification problems, the most commonly used loss function is cross-entropy (*Cross-Entropy Loss*). This metric evaluates the performance of a classification model whose outputs are probabilities between 0 and 1. For binary classification, *Binary Cross-Entropy* is used, and for multiclass classification, *Softmax Cross-Entropy* is employed.

Backpropagation

The discrepancy obtained after applying the loss function is utilized to adjust the weights and biases of the network towards a direction that minimizes error. This adjustment is executed via an algorithm known as *backpropagation*, which leverages the chain rule of differential calculus to propagate the error from the output of the network back to its preceding layers, thus ensuring that each node contributes proportionately to the minimization of the overall error.

In general terms, the objective is to update the parameters w and b to minimize the loss function ($\mathcal{L}$). Hence, the optimization process should enable reaching the minimum of $\mathcal{L}$ by moving in the gradient's direction. If we simplify the problem's dimensionality, the gradient indicates the direction of the steepest ascent. Consequently, we will update our parameters following the negative of the gradient (loss function minimization).

Thus, to update our parameters w and b, we will follow the subsequent equations:

$$w_{\text{updated}} = w - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \tag{2.15}$$

$$b_{\text{updated}} = b - \alpha \frac{\partial \mathcal{L}}{\partial b} \ , \tag{2.16}$$

where α is the step size (or *learning rate*). The value of this parameter is critical, as choosing it too small can result in a prolonged journey to reach the minimum, whereas selecting it too large may cause jumps that lead us to the opposite slope.

The partial derivatives $\partial\mathcal{L}/\partial\mathbf{W}$ and $\partial\mathcal{L}/\partial b$ represent how a small change in the weights or biases can affect the loss function, and are obtained by applying the chain rule during the *backpropagation* algorithm. The appropriate selection of the α value requires some tuning and experimentation, and there are various strategies and advanced optimization algorithms that adjust this value adaptively. During the backpropagation process, the gradient of the error with respect to each weight and each bias in the network is calculated. This gradient represents the direction and magnitude of adjustment for the weights and biases to minimize the error. Subsequently, an adjustment to each weight and bias is made, proportional to the opposite of its gradient, in a process known as gradient descent. This optimization algorithm is a procedure aimed at iteratively minimizing the loss function until the network converges to a solution. Although this solution is not always the global optimum, it is usually good enough for the intended purpose.

If we consider the operations that take place in a neuron and represent them separately to understand how the flow of data occurs, we see that for an input $\mathbf{x}$ and a weight $\mathbf{w}$, an intermediate function is generated which can be expressed as $\mathbf{t} = \mathbf{w} \cdot \mathbf{x}$. Subsequently, after adding the bias b, and applying the activation function h, we obtain the neuron's output function $z = h(\mathbf{t} + b)$.

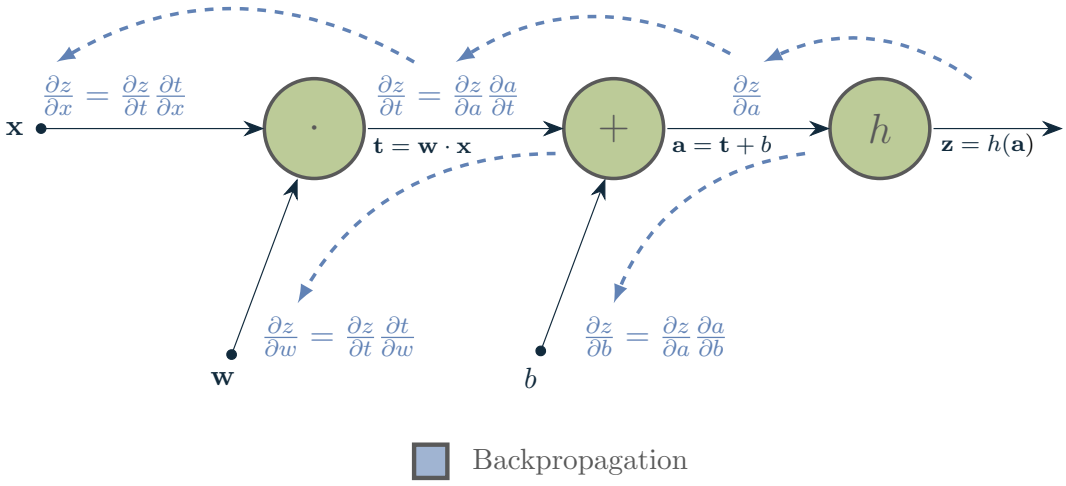


Figure 2.5: Representation of how data flow and backpropagation occur in a neuron based on the chain rule of differential calculus.

At the input of each operation, we have the local gradients:

$$\frac{\partial t}{\partial x}, \frac{\partial t}{\partial w}, \frac{\partial a}{\partial t}, \frac{\partial a}{\partial b} \text{ y } \frac{\partial z}{\partial a} .$$

By applying the chain rule, we obtain the gradients of the output function with respect to each parameter. In Fig. 2.5, it is considered that the input variable $\mathbf{x}$ comes from a previous neuron and is not the value of the input data to the network (in which case backpropagation would not make sense). The values of the obtained gradients will allow us to update our parameters using Eqs. (2.15) and (2.16).

This cyclical process of error-based adjustment is known as supervised learning. At each stage, the network receives supervision through the correct output labels that are supplied to it and uses this supervision to learn and optimize itself.

With improvements in computing hardware and the growth of datasets, neural networks have become deeper and more complex, leading to the development of deep learning. Deep neural networks can have hundreds or even thousands of layers and are capable of learning very abstract and complex representations of their input data. These advances have enabled neural networks to solve tasks previously considered beyond their reach, such as natural language processing, image recognition, or the detection of diseases or identification of anomalous patterns in magnetic resonance images or computed tomographies.

2.3 Neural fields

The universal approximation theorem, first proven by George Cybenko in 1989, states that a neural network with a single hidden layer can approximate any continuous function on closed and bounded subsets of $\mathbb{R}^n$, given certain conditions on the neuron's activation function [17]. This theorem was later extended by Kurt Hornik in 1991, who demonstrated that the result is valid for any non-constant, bounded, and monotonically increasing activation function [18].

With this property of neural networks, the concept of a neural field has recently been introduced in the field of deep learning and computer vision research.

> A neural field is one that is wholly or partially parameterized by a neural network [4].

Neural fields represent a powerful tool for data processing and analysis, especially in regression problems. Moreover, they have proven to be exceptional in modeling image properties based on their spatial coordinates. This means they are capable of identifying and extracting relevant information from an image based on the spatial distribution of its pixels. This innovative feature opens up a completely new range of applications, significantly expanding the boundaries of what is possible in the field of image processing and analysis, such as object detection and recognition in images [19], semantic image segmentation [20], or resolution enhancement [21].

In the following chapter, we evaluate the performance of these neural fields in a particular scenario: the reconstruction of two-dimensional images.

Chapter 3

Two-dimensional image regression

In this chapter, we will explore two modern deep learning techniques for the reconstruction of two-dimensional images: Sinusoidal Representation Networks (SIREN) [5] and Fourier features [6]. Both have proven to be very effective in various applications, including image generation [5, 6] and the inference of three-dimensional scenes from two-dimensional images [5, 6].

In the context of our project, we will apply these techniques to several segments of the 21 pseudomonochromatic images of the solar surface described in Chapter 1. Our goal is not limited to reproducing the demonstrated efficacy of these deep learning techniques in image reconstruction. We will go further and demonstrate that, by making the necessary adjustments to adapt them to our dataset, we can achieve equally satisfactory results and advance on our path towards spectral super-resolution.

3.1 Preliminary concepts

Before delving into these techniques, it is essential to introduce some aspects that are fundamental in the field of deep learning and play a key role. These aspects include parameter initialization, hyperparameter tuning, optimization algorithms, and data preprocessing. Each of these elements contributes to enhancing the performance and effectiveness of our models. Below, we provide an intuitive overview of these concepts to lay the necessary groundwork for the discussion on SIREN and Fourier features.

3.1.1 Parameter initialization

The weights and biases, which are adjusted to minimize the loss function (Eq. 2.13), constitute the parameters of the network, and their initialization is a critical step in the training of neural networks. The parameters are initially set with random values or determined by some predefined rule. The choice of these initial values can have a significant impact on the training process and the effectiveness of the final model [22]. Inadequate initialization can lead to gradient vanishing, hindering learning during the training process.

Gradient vanishing is one of the most common issues during the training process of deep neural networks and occurs as a result of the reduction in the absolute value of the loss function's gradient as the error is propagated back from the network's latter layers. During backpropagation, gradients are calculated by multiplying the partial derivatives of the error with respect to each weight in the network (Fig. 2.5). If we consider a neural network with sigmoidal or hyperbolic activation functions, the derivatives of these functions will be in the range (0, 0.25) and [0, 1], respectively. Therefore, during backpropagation, increasingly smaller values are multiplied together, and the gradient can decrease exponentially [23] as we move back through

the layers. When the gradient approaches zero, the update of the parameters in the earlier layers becomes increasingly insignificant, leading to slow or almost no learning. As a result, the training process becomes inefficient, and the prediction performance significantly worsens.

Next, we provide a basic description of a series of parameter initialization techniques with the goal of offering an overview of their differences and most common uses in the field of deep learning:

- **Xavier-Glorot initialization**: This initialization technique, proposed in 2010, is applied before the start of training and aims to ensure that, on average, the values that pass through the neurons during propagation (activations) and during backpropagation (gradients) maintain a constant variance across the different layers of the network. By doing so, this technique helps to mitigate the problem of gradient vanishing. To achieve this goal, the weights W_{ij} in the weight matrix $\boldsymbol{W}$ are initialized following one of two distributions: a normal distribution centered at zero with variance $\frac{1}{n}$:

$$W_{ij} \sim \mathcal{N}\left(0, \frac{1}{n}\right), \tag{3.1}$$

  or a uniform distribution in the range $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$:

$$W_{ij} \sim \mathcal{U}\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right], \tag{3.2}$$

  where n is the number of inputs to the neuron.

- **He initialization**: developed in 2015, this technique was designed to be more effective in combination with asymmetric activation functions such as ReLU (Fig. 2.2d). Although its primary goal is to optimize the variance of activations, a beneficial side effect is that it helps reduce the problem of gradient vanishing, especially in the initial layers of deep neural networks. This is achieved by setting the variance of the weights at $\frac{2}{n}$, where n is the number of inputs to the neuron. The weights can be initialized using a normal distribution:

$$W_{ij} \sim \mathcal{N}\left(0, \frac{2}{n}\right), \tag{3.3}$$

  or a uniform distribution:

$$W_{ij} \sim \mathcal{U}\left[-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}\right]. \tag{3.4}$$

- **Normal initialization**: this strategy assigns to the weights random values from a standard normal distribution, i.e.,

$$W_{ij} \sim \mathcal{N}(0, 1)\ . \tag{3.5}$$

  Although simple, this method of initialization does not take into account the number of inputs or outputs of the neuron. This can lead to problems of gradient vanishing, especially in deep networks. In particular, the lack of variance adjustment according to the number of inputs or outputs of the neuron can result in gradients that reduce too quickly during backpropagation, thus preventing effective learning of the network.

- **Uniform initialization**: this strategy assigns random values to the weights from a uniform distribution within a specific range, commonly [-1, 1]:

$$W_{ij} \sim \mathcal{U}(-1, 1) \ , \tag{3.6}$$

or [0, 1]:

$$W_{ij} \sim \mathcal{U}(0, 1). \tag{3.7}$$

Like normal initialization, this technique can contribute to gradient vanishing, especially in deep networks, because it does not adjust the variance of the weights based on the number of inputs or outputs of the neuron.

3.1.2 Hyperparameter tuning

Hyperparameters are variables that define the structure of the machine learning model and influence its process. They are different from the model parameters, which are learned during training from the input data. Hyperparameters, on the other hand, are set before training and generally do not change during it. The appropriate selection of hyperparameters can be crucial for model performance, and their tuning is one of the challenges in machine learning. Some of the most common hyperparameters in neural networks are described below:

- **Learning rate (α)**: this is perhaps the most important hyperparameter. It is a scaling factor that determines how much the weights are adjusted in each update during training (Eq. 2.15 and 2.16). A too high learning rate can make learning unstable and cause the model to jump over the minimum of the loss function, while a too low learning rate can make learning too slow or cause the model to get stuck in local minima.
- **Number of hidden layers and number of neurons per layer**: these hyperparameters determine the architecture of the neural network. In general, networks with more layers and/or more neurons can model more complex functions but are also more prone to overfitting and require more data to be trained effectively.
- **Activation function**: as seen in section 2.2, the activation function determines the output of a neuron based on its input. Some common activation functions include the sigmoid function (Fig. 2.2b), the hyperbolic tangent function (Fig. 2.2c), or the rectified linear activation function (ReLU) (Fig. 2.2d). The choice of activation function can have a significant impact on the network's performance.
- **Epoch**: defined as one complete pass of all the training data through the neural network during the learning process. Thus, an epoch is completed when every training example has been presented to the neural network once. The number of epochs determines how many times the entire training dataset will be passed through. The optimal number of epochs generally is obtained through experimentation and can vary depending on the specific problem and dataset.
- **Batch size or Mini-Batch**: the batch size refers to the number of training examples used in one iteration of the optimization algorithm. A batch is simply a subset of the training dataset. A smaller batch size can lead to faster convergence, but it can also make the learning more unstable. A larger batch size can provide more accurate gradient estimates, but it can also make learning slower. A training epoch is completed when all batches have been used once to update the weights.

3.1.3 Optimization algorithms

Optimization algorithms are iterative procedures that seek to minimize (or maximize) an objective function. As we saw in Chapter 2, in the context of deep learning, the objective function is typically a loss function (Equation 2.13) that quantifies the discrepancy between the model's predictions and the actual training data. The goal of optimization is to find the parameters that minimize this function:

- **Gradient Descent (GD)**: it is the simplest optimization algorithm (2.2.1) and is widely used in machine learning and deep learning. In each iteration, GD updates the model parameters in the opposite direction of the loss function's gradient with respect to the current parameters following Eqs. 2.15 and 2.16.
- **Stochastic Gradient Descent (SGD)**: it is a variant of GD that estimates the gradient of the loss function using a random subset of the training data, known as a mini-batch, in each iteration. This makes SGD computationally more efficient than GD, especially for large datasets.
- **Adaptive optimization methods**: these algorithms, such as *Adagrad* [24], *RMSprop* [25], *Adam* [26], among others, modify the learning rate during training. They typically decrease it as training progresses and even scale it based on other parameters and hyperparameters of the network, which can help improve convergence and training stability.

The choice of optimization algorithm can have a significant impact on the performance of deep learning models and is an important part of the tuning process. However, there is no single answer to the question of which optimization algorithm is the best, as the optimal choice may depend on the specific problem, model, and data [27, 28].

3.1.4 Preparation of input data

Datasets play a fundamental role in the field of machine learning and artificial intelligence in both the development and evaluation phases of algorithms and models [29]. The construction, handling, and analysis of these datasets are among the most relevant parts of any data science project, as their quality and representativeness can directly affect the effectiveness and accuracy of the developed models.

Data preprocessing, which includes tasks such as data cleaning, the removal of irrelevant or erroneous data, and the transformation of data into a suitable form for analysis, is essential to ensure their quality [30]. Data may contain noise, errors, inconsistencies, or duplicates that can negatively affect a model's ability to learn. Preprocessing helps to address these issues and ensures that the data are consistent and well-structured, which can facilitate model learning and improve its performance.

Furthermore, it is essential to split the dataset into training, validation, and test sets [31, 32]. The training set is used to train the model and adjust its parameters. The validation set is used for fine-tuning the model and making adjustments, such as hyperparameter selection or detecting overfitting [32, 33]. Finally, the test set is used to evaluate the model's performance on previously unseen data and provide an estimate of its generalization ability.

The proportion in which these datasets are divided can vary depending on their size and characteristics, although a commonly used division is 70% for training, 15% for validation, and 15% for testing [34]. The goal of this division is to provide a balance between maximizing the amount of data available for learning and ensuring that there are enough data to validate and evaluate the model effectively.

3.2 The reconstruction of two-dimensional images

Each of the aspects discussed in the previous section plays a fundamental role in the training of neural networks in general and in their application in image reconstruction in particular. Next, we will explore the basis of the SIREN and Fourier features techniques, focusing specifically on the problem of two-dimensional image regression and the results obtained when applying them to our dataset.

3.2.1 Sinusoidal Representation Networks (SIREN)

Sinusoidal Representation Networks, better known as SIREN [5], are a class of artificial neural networks that use a sinusoidal function as the activation function. This choice of activation function is unique and flexible as it allows the network to model a wider range of functions.

Activation function

Mathematically, the activation of a neuron in a SIREN is described as:

$$h_i = \sin(\boldsymbol{\omega}_i \cdot \boldsymbol{x} + b_i) \ , \tag{3.8}$$

where:

h_i is the activation of neuron i,

$\boldsymbol{\omega}_i$ is the weight vector associated with neuron i,

$\boldsymbol{x}$ is the input vector to the neuron, and

b_i is the bias (scalar) of the neuron.

Conceptually, the basis of SIREN lies in Fourier analysis [35]. This analysis is based on the idea that any arbitrarily complex function, such as audio signals or images, can be expressed as a combination of multiple sinusoidal functions. Each neuron in the network acts as a sinusoidal wave, oscillating at a particular frequency and with a specific phase. By combining these oscillations across multiple layers of the network, SIREN can model highly complex functions with great precision [5].

On the other hand, the choice of a sinusoidal function as the activation function has several additional advantages: first, it is differentiable, which greatly benefits the learning of the network. Being differentiable, it allows for more accurate and smooth calculations of variations in the network's weights during the training process, thus facilitating convergence towards an optimal solution. Secondly, a sinusoidal function presents another convenient mathematical property:

the fact that its derivative is also a sinusoidal function implies that calculating derivatives (an essential operation during training) is computationally efficient. This efficiency contributes to streamlining the process of adjusting weights and improves training speed [5].

Initialization of network parameters

For parameter initialization in SIREN, its authors propose a special initialization for weights and biases [5]: for biases, from a uniform distribution between $[-\pi,\ \pi]$, and for weights, using the following formula:

$$W_{ij} \sim \mathcal{U}\left(-\sqrt{\frac{6}{n}}, \sqrt{\frac{6}{n}}\right), \tag{3.9}$$

where n is the number of inputs to the neuron. This ensures that the activations of the neurons are uniformly distributed in the interval $[-1, 1]$ for an input that follows a distribution $\mathcal{N}(0, 1)$.

Applications

SIREN excels in the representation and reconstruction of two-dimensional images and three-dimensional scenes. SIREN can capture the detailed structure of two-dimensional images with high fidelity, handling fine textures and complicated patterns with great precision. In three-dimensional scenes, SIREN is capable of recreating complex and heterogeneous geometries. It can also handle incident light fields, facilitating the generation of detailed three-dimensional scenes [5].

Beyond images and scenes, SIREN can also be applied to the modeling of vector fields, an important component in areas such as fluid physics or electromagnetism. In this context, SIREN can be used to represent the complexity of these fields with a high degree of accuracy [5].

3.2.2 Fourier features

Fourier features [6], like SIREN, have their roots in Fourier analysis [35]. Both are based on the same fundamental principle of decomposing functions into sinusoidal waves. However, while SIREN integrates sinusoidal periodicity into the layers of a neural network, Fourier features transform the inputs to the network into a set of frequency-based features.

Input mapping

To perform this transformation, the Gaussian Fourier features mapping is used [6]:

$$\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{B}\mathbf{v}), \sin(2\pi\mathbf{B}\mathbf{v})]^{\mathrm{T}}\ , \tag{3.10}$$

where each entry in the mapping matrix $\mathbf{B} \in \mathbb{R}^{m\times d}$ is sampled from a Gaussian distribution $N(0, \sigma)$ and

$\mathbf{v}$ is the vector of normalized spatial coordinates $\mathbf{v} \in \mathbb{R}^{d\times 1}$, in this case, the pixel positions in the image (ranging from 0 to 1);

$\mathbf{B}$ is the matrix of Fourier bases $\mathbf{B} \in \mathbb{R}^{m \times d}$. It's a projection matrix that maps the spatial coordinates to a Fourier features space, in our case d=2, and m is a hyperparameter that depends on the resolution of the images and the complexity of the model;

$\gamma(\mathbf{v})$ is the Fourier features vector, obtained by applying the cosine and sine functions to the spatial coordinates projected to the dimension $k = 2m$. These features will be used as input to the neural network.

We consider the problem of 2D image regression, where we are interested in predicting pixel values for an output image based on the pixel coordinates of an input image. When we map this input into the network using the matrix $\mathbf{B}$, each coordinate is decomposed into a sum of sinusoidal waves of different frequencies. These new sinusoidal features are used as input to a neural network. The network then generates predictions that can be interpreted as the intensity values of the pixels. In this way, the initial sinusoidal mapping allows the network to capture complex patterns in the coordinates of the input image, resulting in a more accurate regression of the output image. Lastly, it is important to clarify that these sinusoidal features are calculated directly from the input coordinates and are not part of the network's learning process.

A notable aspect of Fourier features is their capacity to enhance the performance of neural networks without altering their overall architecture. This process involves preprocessing the inputs to increase their dimensionality, but it does not require any modifications to their hidden or output layers. This means that Fourier features can be easily integrated into any existing network architecture, making them a powerful and flexible tool that enables neural networks to handle highly complex tasks more effectively.

Applications

Fourier features have proven useful in various tasks such as image reconstruction, representation and rendering of three-dimensional scenes, learning of high-frequency textures, and illumination prediction [6]:

- **Image reconstruction**: enables neural networks to learn and recreate images with fine details and complex periodic patterns. A network equipped with Fourier features can learn the detailed texture of an object or the periodic repetition of patterns in an image, significantly improving the quality of the recreated image [6].
- **Representation and rendering of three-dimensional scenes**: using a function regression approach, the authors demonstrate that their method can be used to learn to render three-dimensional scenes from low-dimensional inputs (two-dimensional or three-dimensional coordinates) [6].
- **Learning of high-frequency textures**: allows neural networks to learn to reproduce high-frequency textures, which can be useful in a variety of image processing and graphics tasks [6].
- **Illumination prediction**: using a dataset of real-world lighting, networks of this type can predict the distribution of light in a scene from a low-dimensional input (the direction of light) [6].

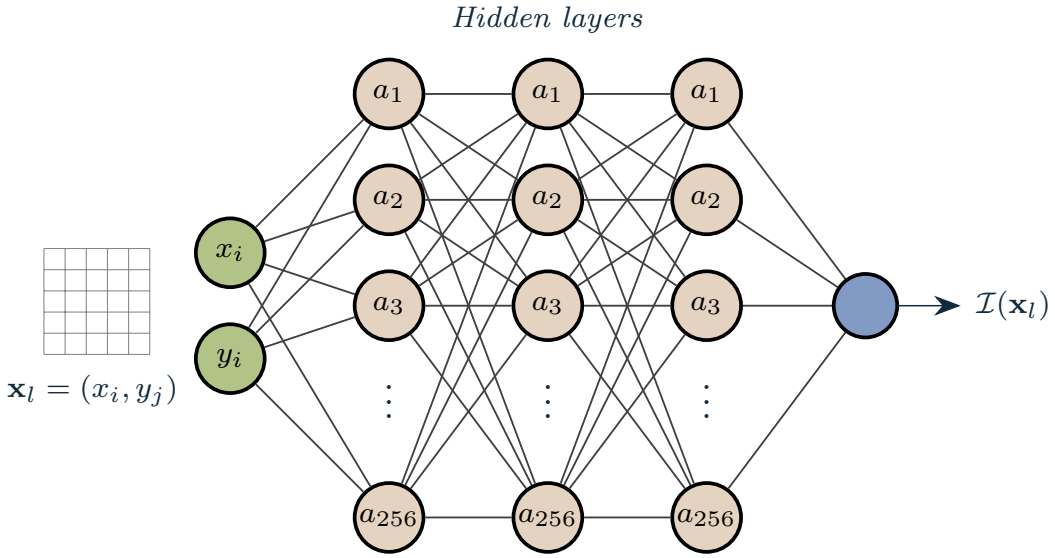


Figure 3.1: Neural network configuration for the experiment with SIREN. The input consists of the spatial coordinates of the image normalized to the interval $[0, 1]$, includes 3 hidden layers of 256 neurons each, and the output layer $\mathcal{I}(\mathbf{x}_l)$, which represents the pixel intensity for the input coordinates.

3.3 Experiments with solar images

For our first objective, we will focus on replicating the experiments presented in the papers on SIREN [5] and Fourier features [6] regarding two-dimensional image regression using our own data. As outlined in Chapter 1, we need to adjust a fully connected *neural field* type neural network to recreate a portion of one of the 21 images in our dataset, and manage to map the input coordinates to the intensity values of each pixel. In our case, we must find the function:

$$\Phi : \mathbb{R}^2 \mapsto \mathbb{R}, \mathbf{x} \to \Phi(\mathbf{x}) \ , \tag{3.11}$$

that parametrizes a discrete image, $\mathcal{I}$, in a continuous manner. The image defines a dataset $\mathcal{D} = \{(\mathbf{x}_l, \mathcal{I}(\mathbf{x}_l))\}_m$ of pixel coordinates $\mathbf{x}_l = (x_i, y_j)$ associated with their intensities $\mathcal{I}(\mathbf{x}_l)$. With this transformation, all values of x will be mapped to the interval [0,1].

3.3.1 Two-dimensional image regression with SIREN

The architecture we use in this experiment is a fully connected multilayer perceptron (MLP) neural network [4, 36], consisting of an input layer, which receives the spatial coordinates, three hidden layers of 256 neurons each, and an output layer with a single neuron (Fig. 3.1), where the comparison between the pixel intensity value obtained by the network for the input coordinates and the corresponding label with the actual pixel intensity for those coordinates will be made. We scale both the input to the model (image pixel coordinates) and the data labels (each pixel's intensities) to values uniformly distributed in the interval $[0, 1]$. This technique is widely used in machine learning to improve the convergence and efficiency of algorithms. When different variables have varied ranges, algorithms, especially those based on gradients, can take longer to converge or can be misleadingly guided towards suboptimal local minima. Scaling the variables to the same range mitigates these problems and enhances the effectiveness of training. This optimizes the network's learning, facilitates an appropriate representation, and ensures an

Parameter	Value
Coordinate normalization	$[0, 1]$
Pixel normalization	$[0, 1]$
Image crop size	512 x 512 pixels
Data split	50% training, 100% test
Weight initialization (first layer)	$\mathcal{U}[-\frac{1}{n}, \frac{1}{n}]$
Weight initialization (hidden and output layers)	$\mathcal{U}[-\sqrt{\frac{6}{n}}/\omega_0, \sqrt{\frac{6}{n}}/\omega_0]$
Bias initialization	0
Frequency parameter (ω_0)	10, 30, 50, 100
Frequency parameter (ω)	10, 30, 50, 100
Hidden layers	3
Neurons in hidden layers	256
Optimizer	*Adam*
Learning rate (α)	10^{-4}
Number of epochs	2000
Metrics	MSE, PSNR

Table 3.1: Summary of parameters used in the two-dimensional image regression experiment using SIREN.

efficient gradient propagation across its layers.

As we mentioned in Chapter 1, we will use a fragment of the original image for reconstruction. This allows for more efficient processing [14] and is particularly useful when using hardware with memory limitations, such as GPUs [37, 38]. The size we have chosen for the image fragment is 512×512 pixels. For the training dataset, we have taken a subsample of the image composed of every odd-subindex pixel from both dimensions (height and width, with indexing from 0). We will evaluate the model during training using the entirety of the image's pixels as the test set.

Following the methodology of the original publication, the network parameters were adjusted as follows: to initialize the weights of the first layer, we used a uniform distribution in the range $[-\frac{1}{n}, \frac{1}{n}]$, where n is the number of inputs to the layer. For the hidden and output layers, a uniform distribution in the range $[-\sqrt{\frac{6}{n}}/\omega_0, \sqrt{\frac{6}{n}}/\omega_0]$. The biases of all layers were initialized with the value $b_i = 0$. Additionally, with the goal of finding the appropriate sinusoidal activation function frequency parameter ω_0 for our dataset, we trained the network with different values (10, 30, 50, 100). This parameter remains constant during training and is crucial for the model's convergence.

The network training was carried out over 2000 cycles (*epochs*) using the *Adam* optimization algorithm [26] with a learning rate (Eq. 2.15 and 2.16) of 10^{-4}. The metric we have used to evaluate the network's performance is the signal-to-noise ratio (PSNR), which is calculated as:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}_I^2}{\text{MSE}} \right) , \tag{3.12}$$

where MAX_I is the maximum possible pixel value in the image (1 in our experiment) and

MSE is the mean squared error. This metric is particularly useful for comparing the fidelity of reconstructed images. Using a logarithmic scale, it emphasizes errors that are perceptually more significant. Additionally, by accounting for MAX_I, PSNR provides context for judging whether a given MSE is high or low relative to the image's range.

Table 3.1 summarizes the data normalization values, its size and distribution, weight and bias initialization, and the activation function frequency of the model, as well as the parameters and metrics used during the training phase.

PSNR	$\omega_0 = 10$	$\omega_0 = 30$	$\omega_0 = 50$	$\omega_0 = 100$
Training	29.51	37.41	39.66	49.13
Test	29.49	37.34	**38.50**	38.39

Table 3.2: PSNR values for training and test data as a function of different ω_0 values in the two-dimensional image regression experiment with SIREN.

The results in Table 3.2 show how the SIREN model behaves in the task of two-dimensional image regression. As the value of ω_0 increases, the PSNR for the training data also tends to increase. However, for the test data, the PSNR peaks around $\omega_0 = 50$ and then slightly decreases for $\omega_0 = 100$. This suggests that while the model can accurately capture the pixel intensities in the training set for higher values of ω_0, it may not generalize as well on unseen data, possibly due to overfitting.

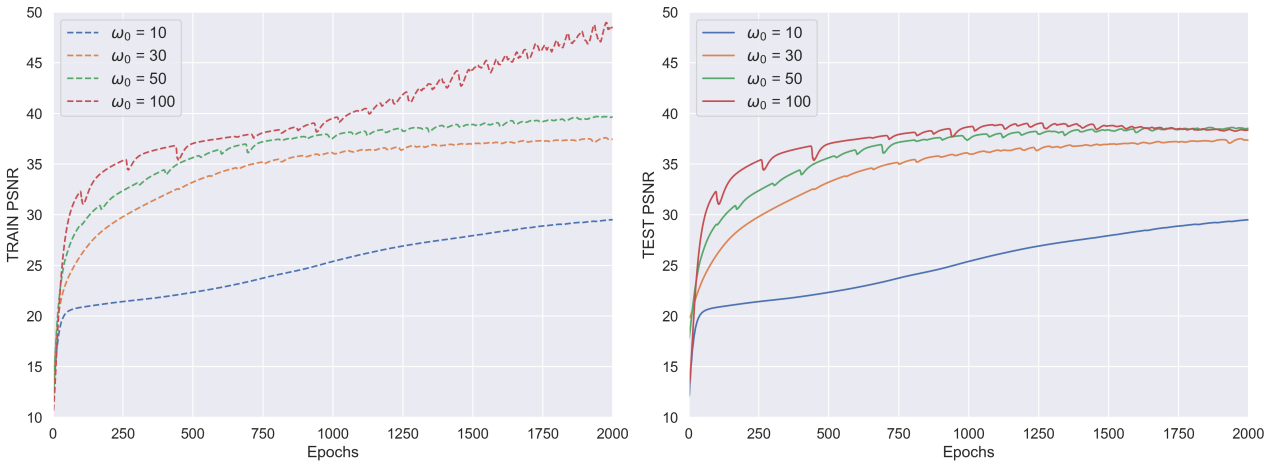


Figure 3.2: Evolution of the PSNR for training (*train*) and test (*test*) data with different ω values over 2000 epochs (*epochs*).

In Fig. 3.2, the evolution of PSNR over the epochs for different ω values is clearly visible. In the training graph (left), it's observed that the PSNR for all ω values tends to increase with the number of epochs, corroborating what was mentioned earlier about how the model better fits the training data with higher ω values. Particularly, the value of $\omega = 100$ shows continuous growth, reaching the highest PSNR values. On the other hand, in the test graph (right), although all ω values show an initial increase in PSNR, there are significant variations in their behavior as the epochs progress. $\omega = 50$ appears to have the best performance, showing sustained growth and reaching a peak before stabilizing, whereas $\omega = 100$, after a rapid initial growth, performs lower compared to the other values as the number of epochs advances. On

closer inspection of the test graph, it seems that overfitting for $\omega = 100$ starts around after 1000 epochs, as the PSNR begins to stabilize and then slightly decreases. This supports the previous observation about potential overfitting with a very high ω.

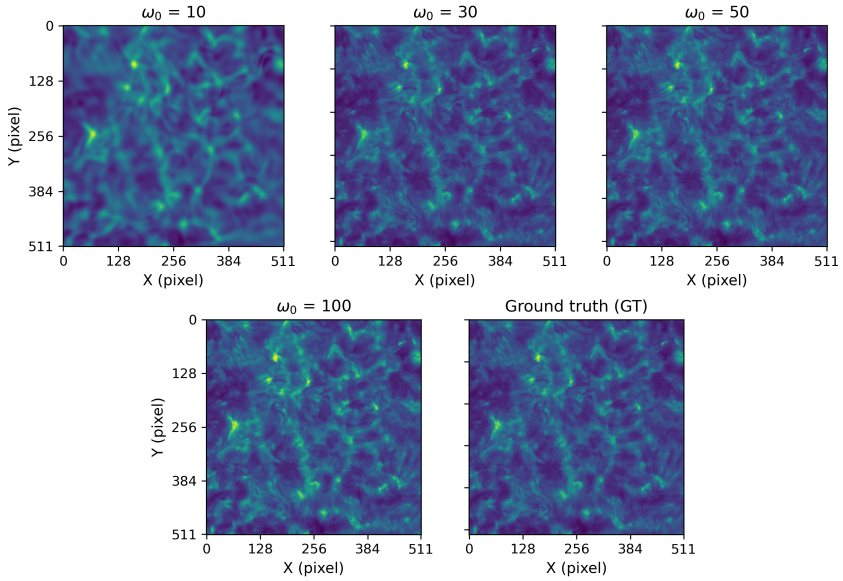


Figure 3.3: Reconstruction of a 512×512 pixel portion of an image from the dataset using the SIREN architecture. The images represent reconstructions with the test dataset for different ω_0 values and the original image (Ground Truth, GT).

Generally, an image reconstruction is considered high quality when the PSNR value is around 30 or higher [7]. High values, both in training and test data, indicate that the images reconstructed by the network are close to the original images. The quality of the visual reconstructions shown in Fig. 3.3 is consistent with these high PSNR values. The reconstructed images that exceed this PSNR threshold usually have a visual fidelity that closely resembles the original images, confirming the network's ability to produce high-quality reconstructions.

3.3.2 Two-dimensional image regression with Fourier features

For the regression experiment with Fourier features, we also used a fully connected multilayer perceptron (MLP) neural network [4, 36]. This network consists of a layer that maps the input coordinates to a first layer of $k = 256$ neurons, followed by three more hidden layers and an output layer. The three hidden layers have 256 neurons each that use a ReLU activation function (Fig. 2.2d). The output layer, responsible for generating the pixel intensities of the image, consists of a single neuron and uses a sigmoid activation function (Fig. 2.2b) to ensure that the generated values are in the range of 0 to 1.

We prepared the input data following the same procedure as the original article [6]: firstly, we fragmented the images to a size of 512×512 pixels and normalized their intensity values between 0 and 1. Next, we created the unit grid, mapping all coordinates to values within the interval [0, 1] [23]. This strategy is particularly effective with the ReLU activation function (Fig. 2.2d), as it ensures that all function outputs are non-negative, thus avoiding truncation to zero and allowing neurons to produce non-null responses [39]. This, in turn, can improve the network's ability to learn and capture significant data features [40]. Moreover, keeping all inputs on a similar scale is important, as it can help avoid stability issues during training, leading to better

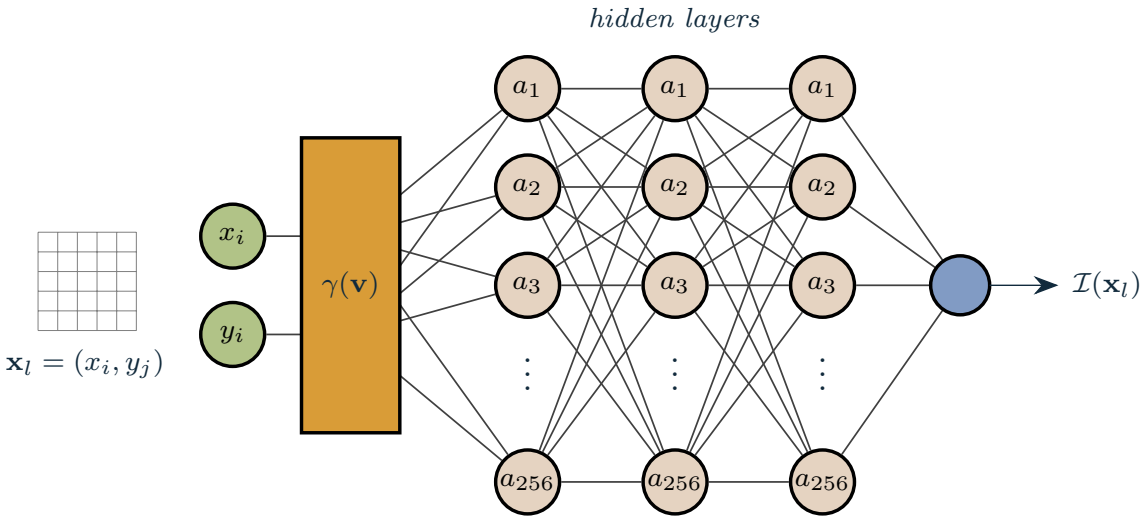


Figure 3.4: Neural network configuration for the Fourier features experiment. The input consists of the image spatial coordinates $\mathbf{x}_l = (x_i, y_j)$ normalized to the unit grid in the interval [0,1], which are subsequently mapped to obtain the Fourier feature vector (Eq. 3.10) with k neurons (in this case 256). The network has 3 hidden layers with 256 neurons each. Finally, the output layer represents the learned pixel intensity.

model performance and accuracy.

Before feeding the network, the image coordinates undergo a transformation using the Fourier features mapping [6]. To appreciate the impact of applying Fourier features on the network's performance, we compare the following mappings of the feature vector $\gamma(\mathbf{v})$:

- No mapping: $\gamma(\mathbf{v}) = \mathbf{v}$.
- Basic mapping: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{v}), \sin(2\pi\mathbf{v})]^{\mathrm{T}}$.
- Gaussian Fourier features mapping: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{B}\mathbf{v}), \sin(2\pi\mathbf{B}\mathbf{v})]^{\mathrm{T}}$, where each entry in $\mathbf{B} \in \mathbb{R}^{m\times d}$ is selected from $N(0, \sigma)$ and $k = 2m$ is the number of Fourier features (in line with the reference experiment [6], we will use $k = 256$). As mentioned in section 3.2.2, the mapping matrix $\mathbf{B}$ is initialized once at the beginning of the experiment and the features calculated from it by applying to the image coordinates are what feed the network and are not updated during the training process.

To compare both models, we trained the network for 2000 epochs using the *Adam* optimization algorithm [26] and a learning rate (Eq. 2.15 and 2.16) of 10^{-4}. Just as with SIREN, the metrics we used to evaluate the network's performance are MSE and PSNR. Table 3.3 summarizes the values for the experiment.

The results presented in Table 3.4 reveal an interesting pattern regarding the value of σ. When a value of $\sigma = 1$ is used, a quite acceptable PSNR is achieved with both the training and test sets, indicating that the model can learn and generalize reasonably well. By increasing σ to 10, an improvement in the PSNR is achieved in both sets, with values of 40.76 in the training set and 39.02 in the test set. This demonstrates that using a larger σ value allows the model to learn more complex and detailed patterns in the image, significantly improving the quality of the reconstruction.

Parameter	Value
Coordinate normalization	[0, 1]
Pixel normalization	[0, 1]
Image crop size	512 x 512 pixels
Coordinate transformation	Fourier feature mapping
Fourier features mappings $\gamma(\mathbf{v})$	No mapping, basic mapping, Gaussian Fourier
Number of Fourier features (k)	256
$\mathbf{B}$	$\in \mathbb{R}^{128\times 2}$
Standard deviation in Gaussian Fourier	$\sigma = 1$, $\sigma = 10$ (optimal), $\sigma = 100$ (overfitting)
Hidden layers	3
Neurons in the first layer	k
Neurons in hidden layers	256
Optimizer	*Adam*
Learning rate (α)	10^{-4}
Number of epochs	2000
Metrics	MSE, PSNR

Table 3.3: Summary of the parameters used in the two-dimensional image regression experiment using Fourier features.

			Gaussian Mapping		
PSNR	No Mapping	Basic Mapping	$\sigma = 1$	$\sigma = 10$	$\sigma = 100$
Train	21.55	26.05	29.92	39.63	36.03
Test	21.55	26.03	30.09	**38.22**	17.85

Table 3.4: Signal-to-Noise Ratio (PSNR) results in the two-dimensional image regression task for training and test datasets, with the different mappings of Fourier features.

However, by increasing the standard deviation to $\sigma = 100$, the model appears to suffer from overfitting. Although the model's performance regarding the training data continues to improve (PSNR = 36.03), the PSNR for the test data decreases to 17.85. This indicates that, despite the model's ability to learn the training data with an extreme level of detail, its ability to generalize to new data is seriously hindered. In contrast, without mapping or with basic mapping, the model is unable to adequately capture the complexity of the image, resulting in inferior reconstruction quality and decreased PSNR (Fig. 3.5).

In general, these results can be visually contrasted with the quality of the reconstructions for each of the mappings shown in Fig. 3.6, where the importance of carefully selecting and adjusting the parameters of Fourier features when applied to image regression tasks becomes evident.

3.4 Discussion of the results

In this chapter, we have demonstrated that networks based on implicit neural fields that use sinusoidal activation functions (SIREN) [5] or Fourier features [6] offer real and very effective

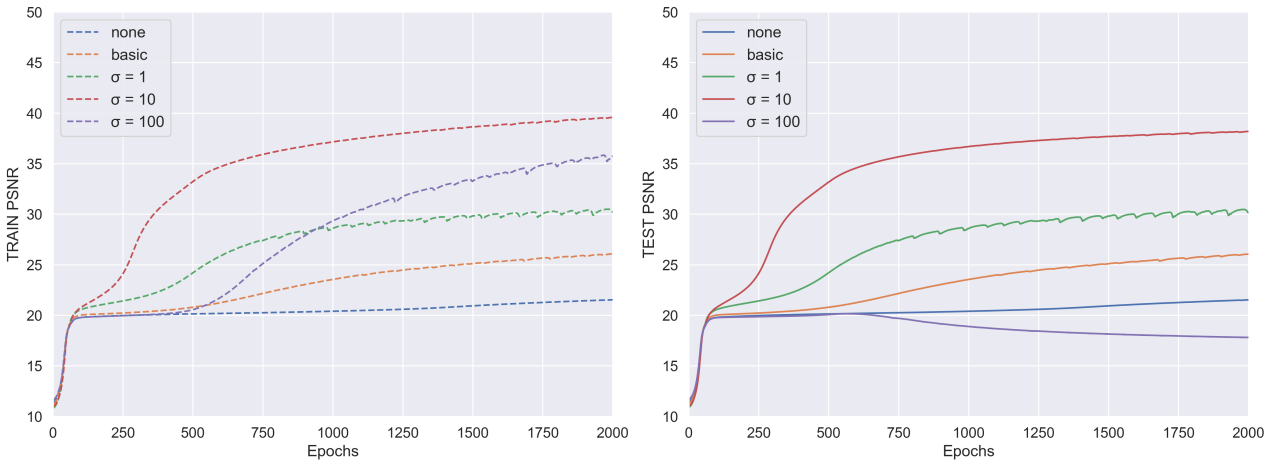


Figure 3.5: Evolution of the Signal-to-Noise Ratio (PSNR) throughout the 2000 epochs of network training for the training (*train*) and test (*test*) sets. Each of the lines corresponds to the mapping used in training: No Mapping (*none*): $\gamma(\mathbf{v}) = \mathbf{v}$, Basic Mapping (*basic*): $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{v}), \sin(2\pi\mathbf{v})]^{\mathrm{T}}$, and Gaussian Fourier features mapping: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{B}\mathbf{v}), \sin(2\pi\mathbf{B}\mathbf{v})]^{\mathrm{T}}$, with $\sigma = 1$, $\sigma = 10$ and $\sigma = 100$.

approaches for the task of image regression. In particular, these methods have proven their ability to learn and accurately reproduce complex and high-frequency patterns, as evidenced by the high PSNR values obtained in the experiments conducted in section 3.3.

While the original SIREN experiments used a single dataset for training [5], our adaptation of the method showed encouraging results by splitting the data into training and test sets

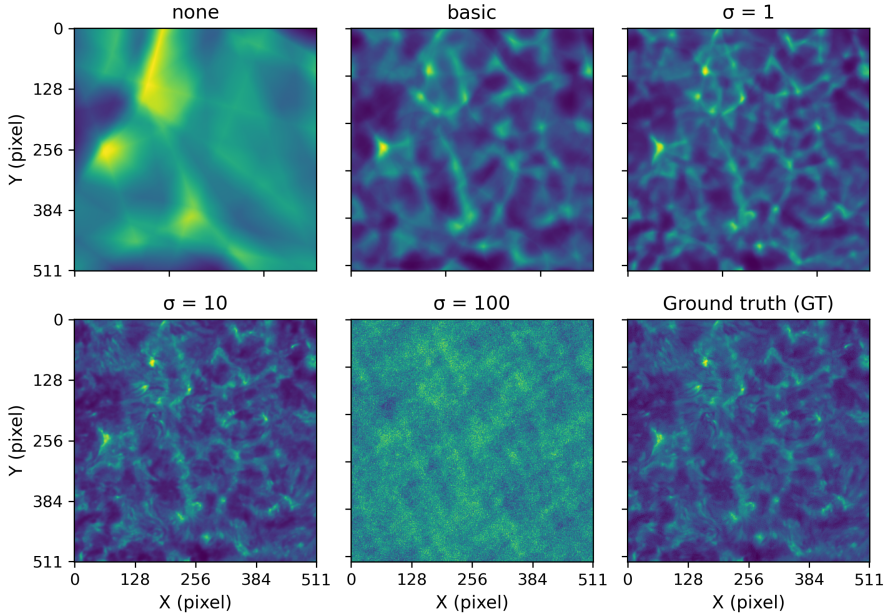


Figure 3.6: Result of the regression of a 512×512 pixel portion of one of the images from our dataset using Fourier features. The reconstructed images are shown using the test data and different mappings: No mapping (*none*): $\gamma(\mathbf{v}) = \mathbf{v}$, basic mapping (*basic*): $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{v}), \sin(2\pi\mathbf{v})]^{\mathrm{T}}$ and Gaussian Fourier features mapping: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{B}\mathbf{v}), \sin(2\pi\mathbf{B}\mathbf{v})]^{\mathrm{T}}$, with $\sigma = 1$, $\sigma = 10$ and $\sigma = 100$. The original image or true data (*Ground truth, GT*) is also shown.

(Fig. 3.2). This highlights the robustness of SIREN, even when attempting to generalize from seen to unseen data, a fundamental criterion for our specific application.

The use of Fourier features also provided promising results (Fig. 3.5). When properly adjusted, they allow the model to effectively and efficiently learn complex patterns and structures from the data. Moreover, the results also revealed the importance of carefully selecting parameters, particularly the value of the standard deviation σ, to avoid overfitting and maximize the model's generalization ability.

The results obtained with both Fourier features and SIREN are promising in terms of the PSNR achieved (Fig. 3.7), suggesting that both may be suitable techniques for our goal of spectral super-resolution in solar absorption lines. While image regression may seem like a concrete and specific task, generalization has been crucial in our scenario. We seek the ability to generate intermediate images in the solar data cube from the learned images. That is, the model must be capable of generating a new image that is not present in the training data. Therefore, a rigorous evaluation of the models' performance on training and test sets is essential.

To conclude, considering the results discussed in this chapter and based on the quality of the images reproduced by the models (Fig. 3.8), both techniques present themselves as a viable and attractive option for advancing towards spectral super-resolution, given their flexibility to adapt to any of the existing architectures and their capability to capture complex patterns in the data.

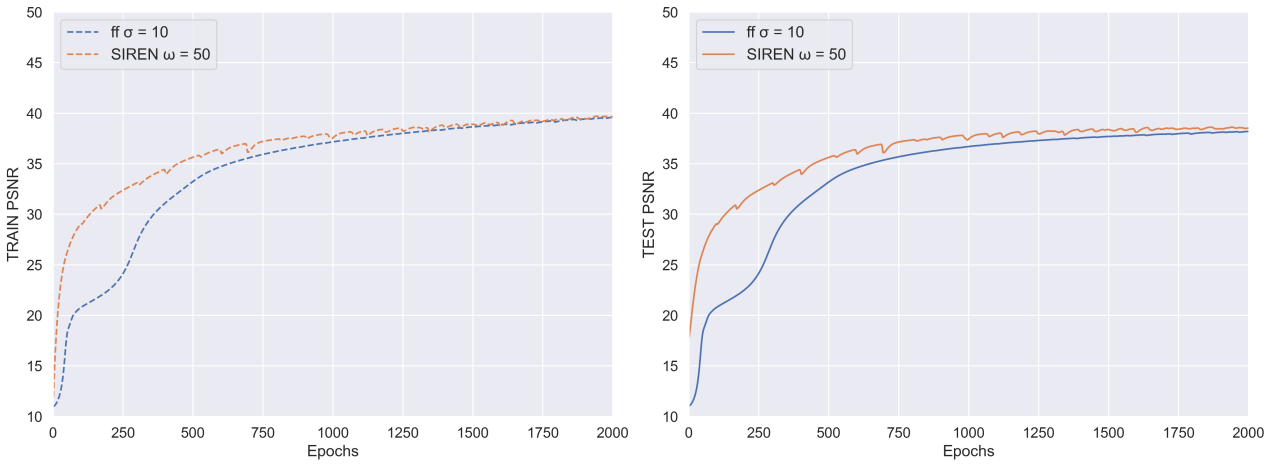


Figure 3.7: Evolution of the Signal-to-Noise Ratio (PSNR) throughout the 2000 epochs of network training for the training (*train*) and test (*test*) sets that achieved the best performance (FF = Fourier features).

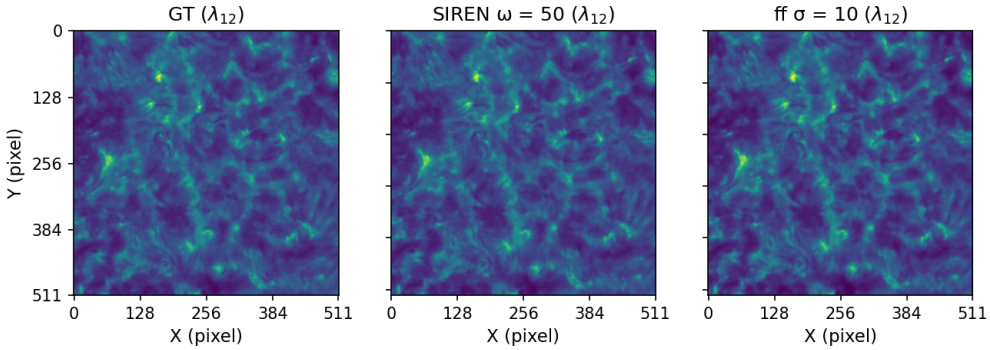


Figure 3.8: Result of the regression of a 512×512 pixel portion of one of the images from our dataset using SIREN and Fourier features (FF). The original image or true data (*Ground truth, GT*) is also shown.

Chapter 4

Towards spectral super-resolution

In the previous chapter, we demonstrated that networks based on implicit neural fields, using sinusoidal activation functions (SIREN) [5] or Fourier features [6], are effective for the task of regressing pseudomonochromatic images of the solar surface. Detailed experiments revealed that these techniques can learn and reproduce high-frequency patterns with a high degree of complexity. Moreover, we argued that both are an effective choice and a solid foundation on which to base our advancement in the challenge of spectral super-resolution in the absorption lines of the Sun's spectrum.

In this chapter, we will introduce the wavelength axis to our series of pseudomonochromatic images, creating a three-dimensional data cube (Fig. 4.2) that allows for continuous spectral interpolation. To do this, we will begin by extending the experiment described in section 3.3.2 with our new dataset, aiming to generate intermediate images in the solar data cube from the learned images. We will provide a detailed view of this process, examining data preparation and selecting the most suitable network architecture for our task.

4.1 The dataset

As we saw in Chapter 1, we have a real spectral data cube $I(x, y, \lambda)$ from the photospheric absorption line of neutral iron centered at 6302 Å (Fe I λ6302). This set consists of 21 pseudomonochromatic images, each with dimensions of 966×964 pixels, corresponding to an extensive area of the solar surface (Fig. 4.1). In other words, these images represent a spectral sampling of the Fe I λ6302 absorption line at 21 specific wavelength segments.

Intuitively, we can visualize this dataset as a three-dimensional cube where the dimension x and dimension y correspond to the pixels in the image, and the dimension λ represents the different wavelengths in the Fe I λ6302 absorption line. The 21 images are equispaced along the λ axis, forming a stack or a cube of images in three dimensions. Fig. 4.2 shows a graphical representation of this cube, where each layer corresponds to an image labeled with a specific λ_t value from 1 to 21. This 3D visualization allows for a clearer understanding of how the absorption line is sampled at different wavelengths, revealing features and patterns that may not be evident when examining the images individually. The choice to label the images from 1 to 21 is made for convenience and helps to conceptualize the spatial and spectral relationship of the images within the cube.
In Fig. 4.3, the variation in the average pixel intensity of each of the 21 images can be observed, which shows the distinctive profile of the absorption line, with a minimum in the average intensity at its center.

This absorption profile not only underscores the three-dimensional nature of the dataset but also provides an essential perspective on how the observed structures in the studied region of

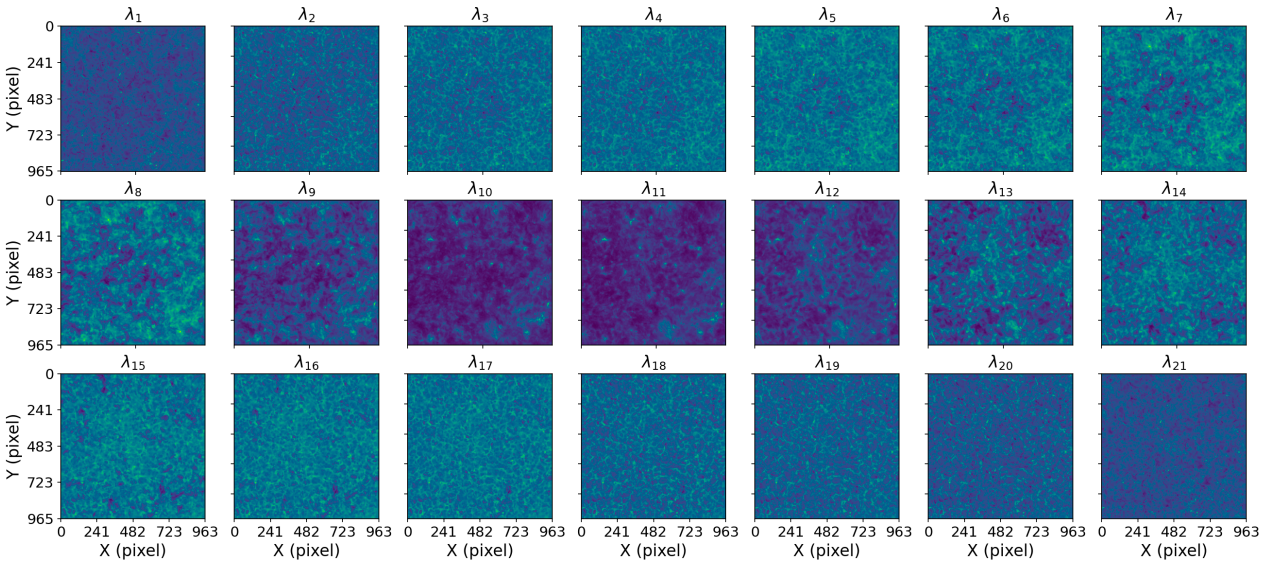


Figure 4.1: 21 images scanning the photospheric absorption line of neutral iron centered at 6302 Å (Fe I λ6302). Each image is labeled as λ_t with t from 1 to 21.

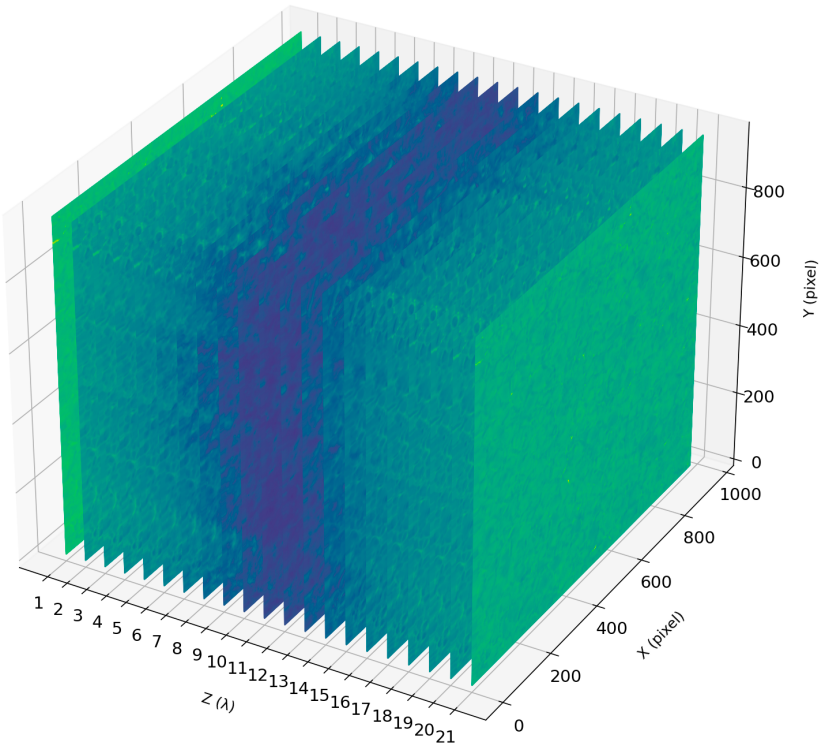


Figure 4.2: Three-dimensional representation of the image cube scanning the photospheric absorption line of neutral iron centered at 6302 Å (Fe I λ6302). The 21 images are equispaced in wavelength and labeled as λ_t with t from 1 to 21.

the Sun change with the wavelength within the Fe I λ6302 absorption line. This information is vital for understanding the physical properties of this region and serves as a valuable tool for the interpretation and analysis of the photospheric absorption line of neutral iron. Recall that in a spectral line, the change in absorption intensity as a function of wavelength is related to

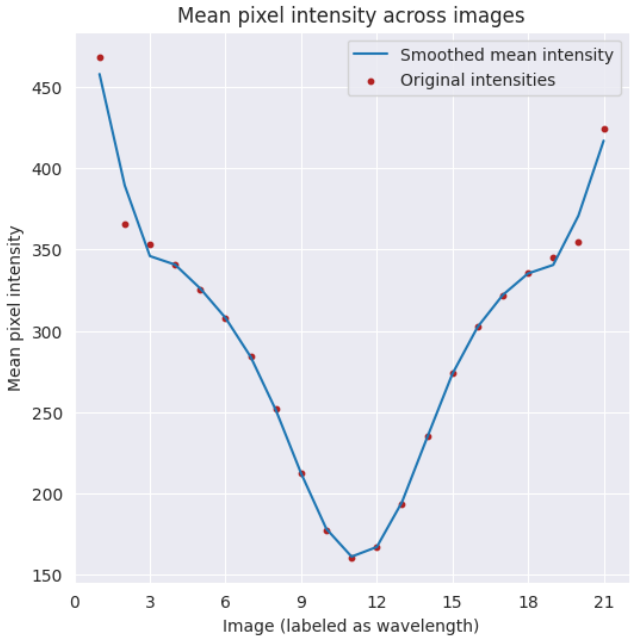


Figure 4.3: Profile of the Fe I λ6302 absorption line, showing the variation of the average pixel intensity across the 21 equispaced images. The original pixel intensities are shown with red dots (*original intensities*) and in blue a smoothed regression of the change in average intensity across the absorption line (*smoothed mean intensity*).

the depth in the solar photosphere (see Chapter 1).

4.2 Reconstruction of the solar image cube

With our new approach, we aim not only to map the spatial coordinates x and y of the image but also to incorporate the wavelength axis λ, allowing us to find a function

$$\Phi : \mathbb{R}^3 \mapsto \mathbb{R}, (\mathbf{x}, \lambda) \rightarrow \Phi(\mathbf{x}, \lambda) \tag{4.1}$$

that parameterizes our data cube continuously. The three-dimensional image defines a dataset $\mathcal{D} = \{(\mathbf{x}_l, \lambda_t, \mathcal{I}(\mathbf{x}_l, \lambda_t))\}_m$ of pixel coordinates $\mathbf{x}_l = (x_i, y_j)$ and wavelengths λ_t associated with their intensities $\mathcal{I}(\mathbf{x}_l, \lambda_t)$. Using the same *neural fields* approach as in the experiment in section 3.3.2, we now need to analyze how this new dimension influences our mapping function.

The way we feed data into the network is essential for its performance. We are now working with a three-dimensional data cube, so each input will consist of the coordinates (x_i, y_j, λ_t). The structure of this data requires prior manipulation to ensure that the network can learn both spatial and spectral patterns within the cube. Firstly, following the procedure presented in section 3.3.2, we crop each of the images to a uniform size. Secondly, we normalize the pixel intensities to the interval $[0, 1]$ relative to the maximum and minimum of each image to ensure that all values are on the same scale. This facilitates convergence during training and improves the stability and performance of the model as discussed in section 3.3.2. Finally, we create a three-dimensional grid that maps all coordinates to values within the interval $[0, 1]$. We do this by generating equispaced points in each dimension (x_i, y_j, λ_t) and combining them into a three-dimensional grid (Fig. 4.4).

4.2.1 Experiments with SIREN

The SIREN architecture we used in our implementation is shown in Fig. 4.4. It consists of an input layer that takes pixel coordinates (x_i, y_j, λ_t) and applies a sinusoidal activation function with a frequency parameter ω_0, 3 hidden layers that also use sinusoidal activation functions, each with its own frequency parameter ω, and an output layer that produces the pixel intensity value for the given coordinates.

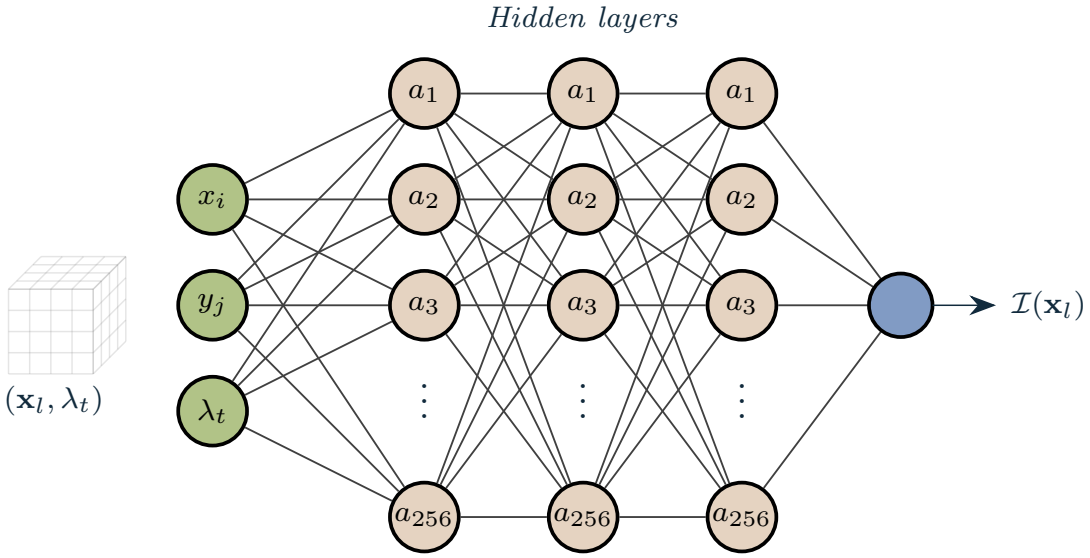


Figure 4.4: Configuration of the neural network for the experiment with SIREN. The input consists of the spatial and wavelength coordinates of each image $(\mathbf{x}_l, \lambda_t)$ normalized to the three-dimensional unit grid, comprises 3 hidden layers of 256 neurons, and the output layer $\mathcal{I}(\mathbf{x}_l)$, which represents the pixel intensity for the input coordinates.

We explored different frequency parameters ω_0 for the input layer and ω for the hidden layers. Specifically, we tested the values with which we obtained the best results in the experiment in section 3.3.1, namely $\omega_0 = 30$ and $\omega_0 = 50$ for the frequency of the input layer and $\omega = 30$ and $\omega = 50$ for the frequency of the hidden layers. The network was trained over 2000 *epochs* using *Adam* as the optimization algorithm [26], with a learning rate set at 10^{-4} (Eq. 2.15 and 2.16). Lastly, we evaluated performance using the signal-to-noise ratio (PSNR) metric.

Table 4.1 shows that the PSNR values obtained with the test data exceed the threshold of 30 for both $\omega_0 = 30$ and $\omega_0 = 50$. These values suggest that the network is capable of approximating the original images with a high degree of accuracy [7], which is crucial for addressing spectral super-resolution.

PSNR	$\omega_0 = 30$	$\omega_0 = 50$
Training	33.57	34.83
Test	33.72	**35.02**

Table 4.1: PSNR values for training and test data based on different ω_0 values in the regression experiment of the 21 pseudomonochromatic images with SIREN.

Effective image reconstruction is most evident in the visual representations. As shown in

Fig. 4.5, the images generated by the network (top part of each subfigure) exhibit remarkable similarity to the original images (bottom part of each subfigure), highlighting the network's ability to spectrally interpolate within the data cube.

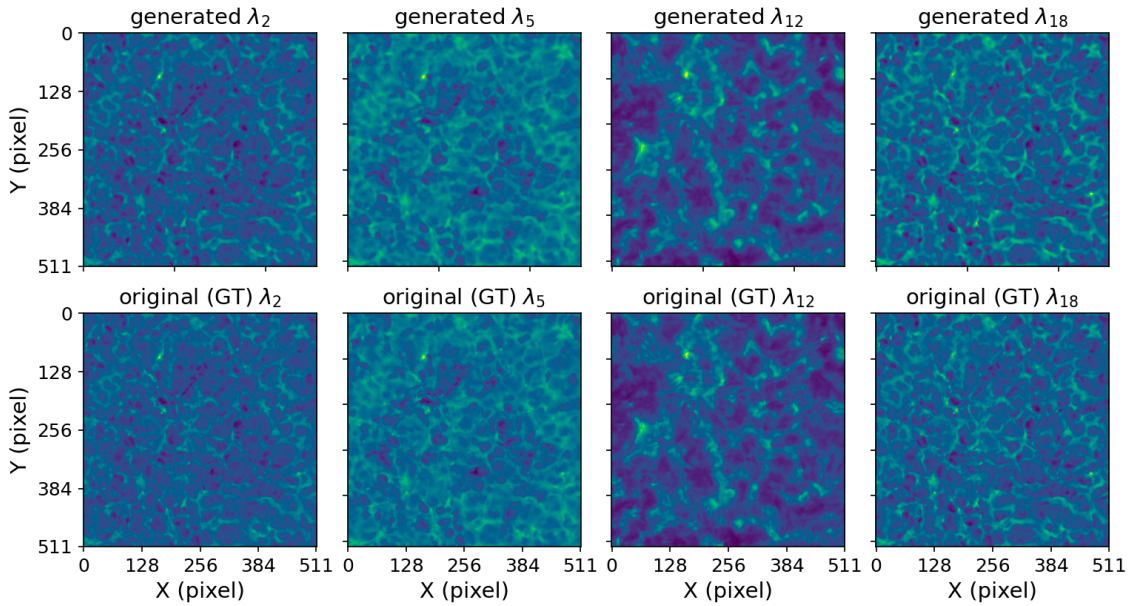


Figure 4.5: Reconstruction of a 512 × 512 pixel portion of four images from the dataset using the SIREN architecture. The upper images represent the reconstructions using a $\omega_0 = 50$ value and the lower ones the originals (Ground truth, GT).

The evolution of the PSNR during the training and testing phases, represented in Fig. 4.6, reflects the stability and consistency in the performance of the model with behavior that favors generalization. This is particularly relevant when we consider continuous spectral interpolation in our 3D data cube, a key challenge in image reconstruction where accuracy is crucial.

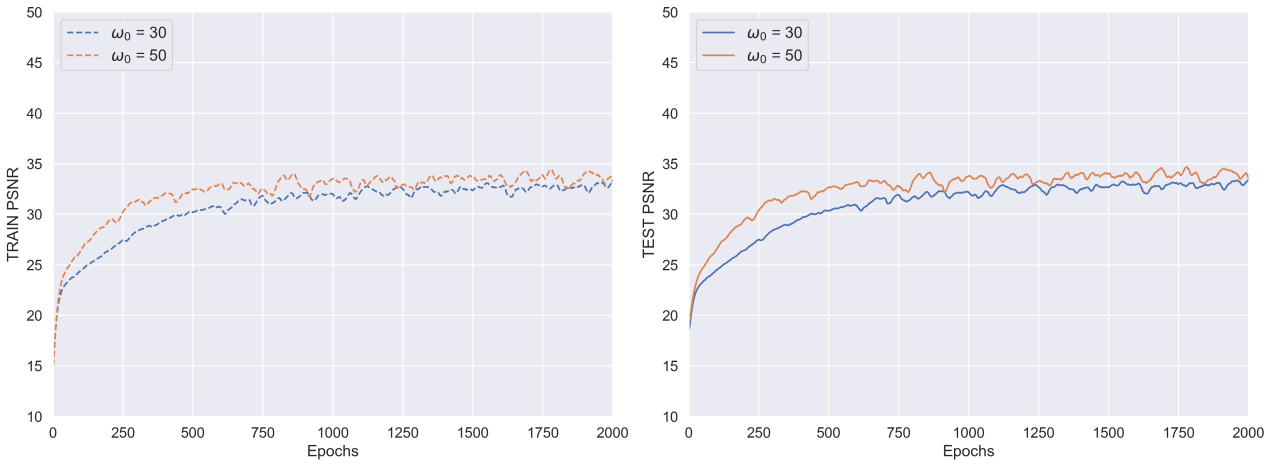


Figure 4.6: Evolution of the PSNR for the training (*train*) and testing (*test*) data with different ω values throughout the 2000 epochs (*epochs*).

4.2.2 Experiments with Fourier features

We will use the same network architecture as in section 3.3.2 to test the performance of the *neural field* with our new dataset (Fig. 4.7):

- **First layer:** the number k of neurons matches twice the number of features used to generate the matrix $\mathbf{B} \in \mathbb{R}^{m \times d}$; $k = 2m$ of Fourier bases. The Fourier features vector $\gamma(\mathbf{v})$ is obtained by mapping the three-dimensional coordinates of the image $(\mathbf{x}_i, \lambda_t)$ following Eq. 3.10.
- **Hidden layers:** the model includes 3 hidden layers, each with 256 neurons. We use the ReLU activation function (Fig. 2.2d) in these layers, which is particularly useful for learning nonlinear relationships in the data.
- **Output layer:** the output layer consists of a single neuron with a sigmoid activation function (Fig. 2.2b) that produces the final output of the model.

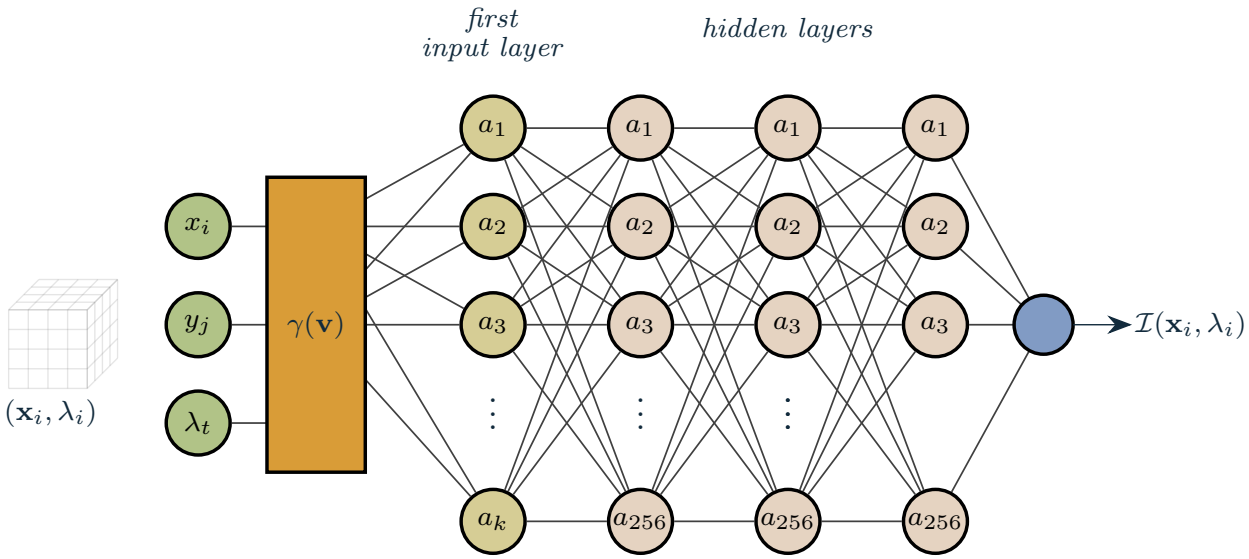


Figure 4.7: Configuration of the neural network. The input consists of the spatial and wavelength coordinates of each image $(\mathbf{x}_i, \lambda_i)$ normalized to the three-dimensional unit grid. These coordinates are then mapped to obtain the Fourier features vector (Eq. 3.10) and the first layer with k neurons. The remaining hidden layers of 256 neurons each are then distributed, and finally, the output layer that contains the pixel intensity value learned by the network.

Search for network hyperparameters

We want to determine if the choice of the number of features k has a direct impact on the model's ability to capture fine details and complex patterns within the images without incurring overfitting, so we will explore three different values of k: 128, 256, and 512. The selection of these values is not arbitrary: the value $k = 256$ has already shown to yield good results (see section 3.3.2), and the additional values $k = 128$ and $k = 512$ will allow us to evaluate how reducing or increasing complexity affects both the quality of the reconstruction and the training time. Another important hyperparameter is the standard deviation of the distribution $N(0, \sigma)$

used to select the entries of the matrix $\mathbf{B}$. As discussed in section 3.4, the proper choice of this parameter prevents overfitting and maximizes the model's generalization ability. For our experiment, we will use the value with which we obtained the best result $\sigma = 10$ (see section 3.3.2), and two additional unexplored values, $\sigma = 0.8$ and $\sigma = 25$. The selection of these values aims to examine how different degrees of transformation in the mapping matrix can influence the model's effectiveness in representing the three-dimensional structure of the data. For each value of k, we explore all values of σ over a training period of 20 epochs. Table 4.2 provides this initial approximation. We use the same metric (PSNR) as in the experiments of chapter 3 to quantify the quality of the reconstruction, where a higher value indicates greater fidelity.

Gaussian Mapping

	$k = 128$	$k = 256$	$k = 512$
σ	PSNR Test	PSNR Test	PSNR Test
0.8	**19.82**	**19.74**	**19.87**
10	19.51	19.30	19.74
25	19.65	19.66	19.67

Table 4.2: Results of the signal-to-noise ratio (PSNR) in tuning the parameters k and σ for the test datasets using the values of $k = 128,\ k = 256,\ k = 512$ and $\sigma = 0.8$, $\sigma = 10$, and $\sigma = 25$ over 20 epochs

We observe that configurations with a $\sigma = 0.8$ achieve, although not significantly, the highest PSNR values across all tested k dimensions. This result may suggest that a more concentrated Gaussian mapping favors the representation of relevant image features without introducing unnecessary noise. On the other hand, the variation among different k values is even less relevant, indicating that increasing dimensionality does not produce significant improvements in PSNR within the range of tested values. This might indicate a point of diminishing returns for higher values of σ, where increasing the complexity of the model does not necessarily translate to better generalization capability. The results highlighted in bold represent the parameter combinations that achieved the best performance, providing a guideline for parameter selection in future trainings. For $\sigma = 0.8$, the test results are better than those experiments with larger σ. This marks a shift from the case of two-dimensional regression in section 3.3.2, where $\sigma = 10$ yielded optimum results. This change in the efficacy of σ could be related to the introduction of the wavelength axis. In a three-dimensional representation, this axis could introduce additional variations and complexities in the data that require a lower standard deviation. Reducing σ to 0.8 could help to better capture these variations, allowing more flexibility in adapting to the three-dimensional structure, where frequencies may be distributed more complexly. A smaller σ might reflect a need to fine-tune the Fourier transformation to capture these additional frequencies. However, we might be able to approach a solution without excluding previous results. In the commonly used Gaussian mapping, an isotropic distribution is employed, where each entry in $\mathbf{B} \in \mathbb{R}^{m \times d; k=2m}$ is sampled from a normal distribution $\mathcal{N}(0, \sigma)$ and σ is chosen uniformly across all dimensions [6]. The functional form of the mapping is given by Eq. 3.10, where isotropic distribution means that the frequency spectrum of the signal is uniform in all directions. In our approach, besides using the value $\sigma = 0.8$, we will employ a vector $\boldsymbol{\sigma} = [\sigma_x, \sigma_y, \sigma_\lambda]$, allowing each dimension to have its own standard deviation. This reflects the unique nature of each dimension in the image cube. Thus, we construct the matrix $\mathbf{B}$ as follows:

$$\mathbf{B} = \begin{bmatrix} \mathcal{N}(0, \sigma_x) & \mathcal{N}(0, \sigma_y) & \mathcal{N}(0, \sigma_\lambda) \end{bmatrix},$$

where each column corresponds to a different dimension, introducing greater flexibility and adaptation to the individual properties of each dimension, enabling the network to learn features that are more representative. As a result of the results from Table 4.2 and the preceding reasoning, we will use two standard deviation values to train the network: the scalar value $\sigma = 0.8$ and the vector $\boldsymbol{\sigma} = [10, 10, 0.8]$, where we maintain the value of 10 used in the two-dimensional regression for the coordinates σ_x, σ_y and introduce the value $\sigma_\lambda = 0.8$. Additionally, for each standard deviation value, we will use the k values of $k = 128$ and $k = 512$.

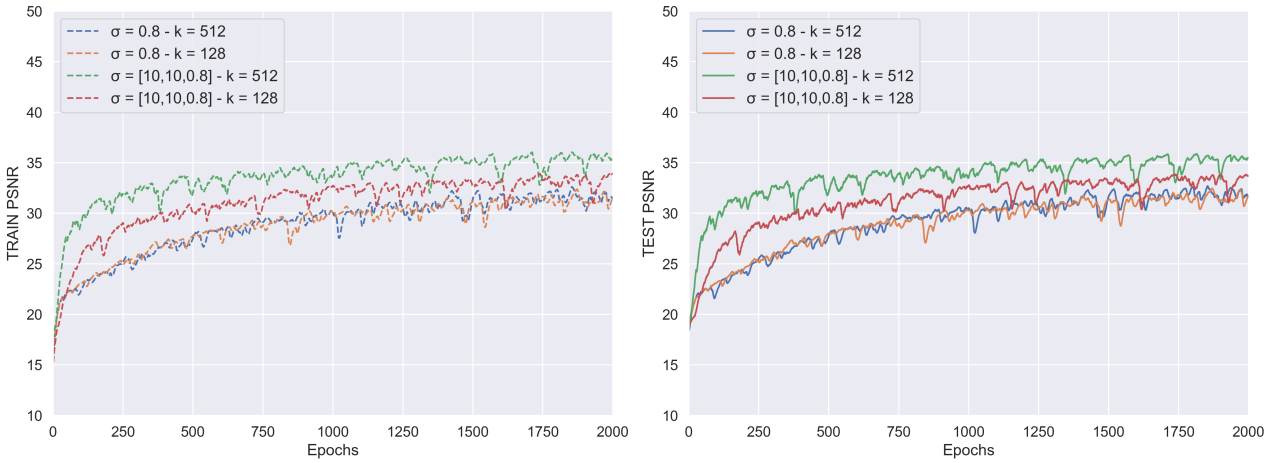


Figure 4.8: Evolution of the PSNR for the training (*train*) and testing (*test*) data with different values of σ and k throughout the 2000 epochs (*epochs*).

As we saw in Chapter 2, the learning rate (Ec. 2.15, 2.16) is one of the most critical hyperparameters in the training of a neural network. It controls the size of the steps the model takes during optimization and, therefore, has a strong influence on the convergence of training. To find its optimal value, we have implemented a selective search within a range of values using a two-stage strategy:

1. **Exponential search (coarse search):** in the first stage, we perform a search on a logarithmic scale, testing exponential learning rate values. This allows for a rapid and broad exploration of the space, which helps identify the most suitable range for a more detailed search. Exponential search is especially useful for understanding the model's sensitivity to different orders of magnitude of the learning rate.

2. **Uniform search within an interval (fine search):** in the second stage, we conduct a more focused search within the interval identified in the previous stage, using a uniform distribution. This search allows for more precise exploration, which helps fine-tune the selection of the optimal learning rate within the previously identified range.

This two-stage strategy combines the efficiency of exponential search with the precision of uniform search, allowing for a well-balanced selection of the learning rate. Table 4.3 summarizes the parameters used in the experiment.

The results of the experiments conducted with Fourier features, illustrated in Table 4.4, reveal the precision of the image set reconstruction. It is observed that the use of a vector for $\boldsymbol{\sigma}$ combining different scales, specifically $[10, 10, 0.8]$, achieves outstanding performance in PSNR

Parameter	**Value**
Coordinate normalization	[0, 1]
Pixel normalization	[0, 1]
Image crop size	512×512 pixels
Coordinate transformation	Fourier features mapping
Fourier features mappings $\gamma(\mathbf{v})$	Gaussian Fourier
Number of Fourier features (k)	128, 512
$\mathbf{B}$	$\in \mathbb{R}^{m\times 3;\ k=2m}$
Standard deviation in Gaussian Fourier	$\sigma = 0.8$, $\boldsymbol{\sigma} = [10, 10, 0.8]$
Neurons in the first layer	k
Number of hidden layers	3
Neurons in hidden layers	256
Optimizer	*Adam*
Learning rate (α)	0.002
Number of epochs	2000
Metric	PSNR

Table 4.3: Summary of the parameters used in the regression experiment of the three-dimensional data cube using a *neural field* with Fourier features.

metrics for both training and test data. The Fourier features model also surpasses the PSNR of 30, considered an indicator of high quality in image reconstruction, indicating the good fit of the model and its ability to capture the complexity of the data.

	Gaussian Mapping			
	$k = 128$		$k = 512$	
$\boldsymbol{\sigma}$	PSNR Train	PSNR Test	PSNR Train	PSNR Test
0.8	32.77	32.95	33.39	33.42
$[10, 10, 0.8]$	34.57	34.52	36.74	**36.38**

Table 4.4: Results of the signal-to-noise ratio (PSNR) in the task of regressing the entirety of the images for the training datasets using, on the one hand, the values of $k = 128,\ k = 512$ and on the other a scalar $\sigma = 0.8$ and a vector $\boldsymbol{\sigma} = [10,\ 10,\ 0.8]$.

Fig. 4.8 shows the evolution of the PSNR throughout the training for various configurations of k and σ. The consistency in improvement and the maintenance of high PSNR values across epochs emphasize the robustness of the model. The configuration with $\sigma = [10, 10, 0.8]$ and $k = 512$ proves to be particularly effective, suggesting that the introduction of variety in the scale of the Gaussian mapping can indeed benefit the generalization of the model, which was timidly intuited from the values in Table 4.2.

Finally, the visual fidelity of the reconstructed images, which can be observed in Fig. 4.9, is objectively high and accurately reflects the original images. This visual correspondence supports the model's generalization capability, which is essential for our purpose. The results obtained are promising and reinforce the methodology used to optimize the model's parameters.

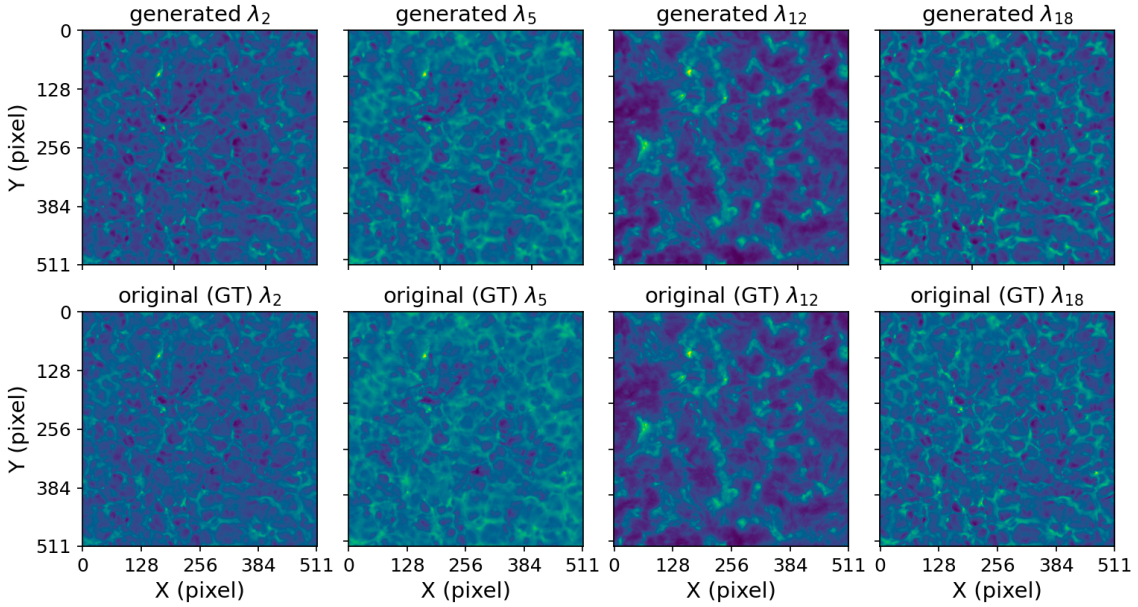


Figure 4.9: Reconstruction of a 512×512 pixel portion of four images from the dataset using the *Fourier features* architecture. The upper images represent the reconstructions using the values of $\boldsymbol{\sigma} = [10,\ 10,\ 0.8]$, $k = 512$ and the lower images are the originals (Ground truth, GT).

4.3 Towards spectral super-resolution

The similarity observed in the results, in terms of PSNR, as shown in Fig. 4.10, suggests a comparable evolution and notable fidelity in the reconstruction of images from the dataset using both techniques. This establishes a solid foundation for moving forward to the next step: evaluating the models' capability for spectral super-resolution.

For this, we perform an interpolation of the three-dimensional coordinates between each pair of consecutive images, which are then processed by the trained model. As a result, we obtain a series of intermediate images; each represents a specific step in the spectral transition between the original images. This technique will be applied to the four models presented in Fig. 4.10 to analyze how each interprets these intermediate images.

Fig. 4.11 shows the evolution of the mean pixel intensity per image, comparing the results generated by the SIREN and Fourier features models with the values of the real images (GT). In the graph, the red dots represent the mean pixel intensity of the real images, normalized to the maximum and minimum values of each image. These points are located at specific intervals, corresponding to each of the 21 wavelengths of the images in the dataset. Between each pair of consecutive images, eight intermediate images have been generated by each model, resulting in a detailed sequence that allows observing the continuous spectral transition. The

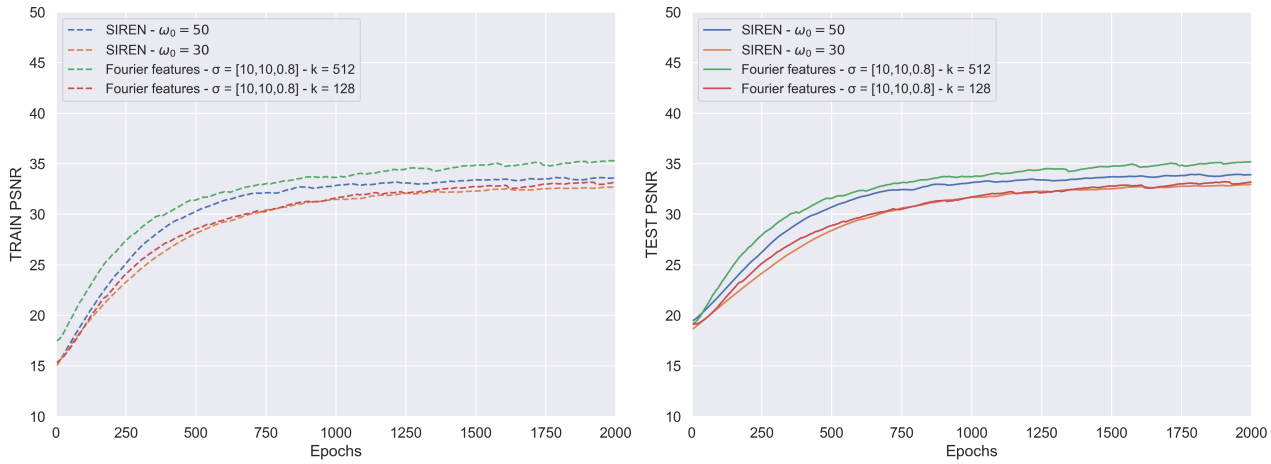


Figure 4.10: Evolution of the PSNR for the training (*train*) and testing (*test*) data of SIREN models with $\omega_0 = 30$ and $\omega_0 = 50$ and *Fourier features* with $\boldsymbol{\sigma} = [10,\ 10,\ 0.8]$ and k values of 512 and 128 throughout the 2000 epochs (*epochs*).

presence of red dots on the graph enables a reading of the accuracy of the models in the task of reconstructing the image intensities.

Figure 4.11: Comparison of the evolution of the mean pixel intensity between images generated by the SIREN and Fourier features models, along with the corresponding points for the mean intensity of each of the real images (GT).

It is observed that the Fourier features model with $k = 512$ and $\sigma = [10, 10, 0.8]$ closely follows the trend set by the real images, indicating a good reconstruction capability without evident signs of overfitting. On the other hand, the SIREN model with $\omega_0 = 50$ shows notable agreement with the points of the original images, though with intermediate fluctuations that could be interpreted as variability in the reconstruction. The SIREN model with $\omega_0 = 30$ and the Fourier features model with $k = 128$ also replicate the general trends of the original images but with less precision in the reconstruction. This behavior could suggest a lower sensitivity to the particulars of each individual image. The fluctuations observed in Fig. 4.11 for the case of SIREN with $\omega_0 = 50$ could be an early indication of an excessive fit to the specifics of the training data, and perhaps evidence some overfitting of the model. Although the comparison suggests that all models manage to follow the trend of the original images, the choice of the optimal model will require a balance between precision and generalization, with a slight hint

that the Fourier features model with $k = 512$ might offer the best of both. Fig. 4.11 reveals uneven behavior in the evolution of the mean pixel intensity of the models in the interval between wavelengths λ_9 and λ_{11}. This interval shows an intersection in the trajectories of the models, suggesting a zone of spectral transition where the models might significantly differ in their interpretation of the features of the generated data. We chose this range to be able to perform a visual analysis of how each model manages the spectral transitions and to validate if the reconstructions maintain coherence with the original images in this region.

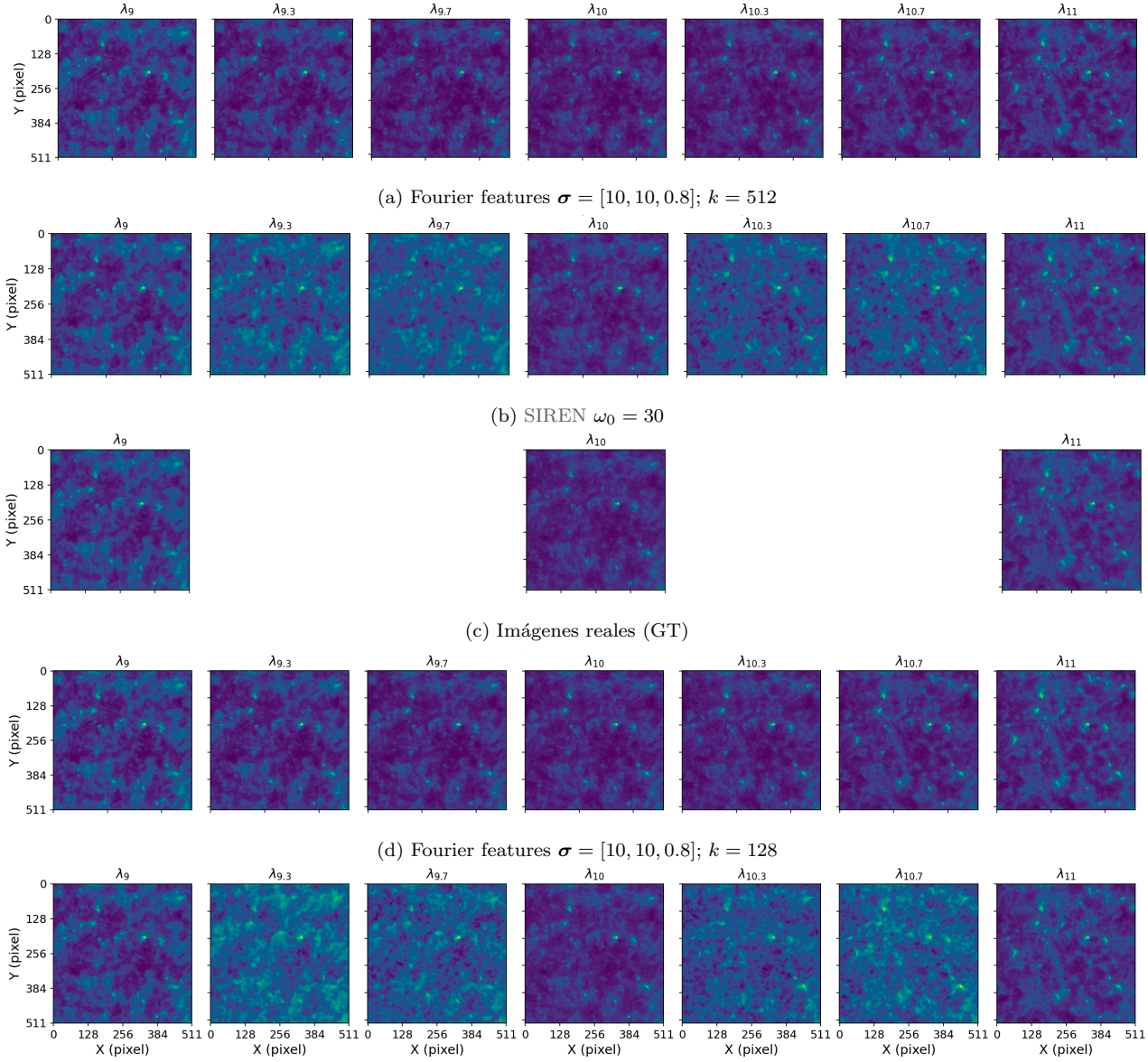
Fig. 4.12 displays a series of generated images that demonstrate the ability of the SIREN and Fourier features models to perform spectral interpolations between wavelengths λ_9 and λ_{11}. This specific selection offers an opportunity to assess the ability of the models to generate complex transitions in the spectral details of the images. The reconstructions obtained through Fourier features, both with $k = 512$ (4.12a) and $k = 128$ (4.12d), exhibit great visual coherence with the real images (GT) (4.12c), indicating precise interpolation and high fidelity in the reconstruction. Although the images corresponding to Fourier features with $k = 128$ (4.12d) show a slight decrease in this coherence, it can be suggested that a lower model complexity may be sufficient for the task of spectral super-resolution in certain wavelength ranges.

On the other hand, the images generated by the SIREN model with $\omega_0 = 30$ (4.12b) and $\omega_0 = 50$ (4.12e) reflect a high fidelity in the reconstruction of the images from the dataset, although the intermediate ones seem not to follow a visual evolution as consistent as those generated by the Fourier features models. This phenomenon could be interpreted as a tendency towards overfitting, whereby the model has excessively adjusted to the peculiarities of the training set, compromising generalization.

The qualitative observations of the generated images agree with the previously analyzed quantitative metrics, underscoring the importance of finding a balance between the model's capability and its ability to generalize, with the goal of achieving spectral super-resolution with a certain degree of precision. The images selected for Fig. 4.12 have helped us determine whether the patterns and essential characteristics of the original images are preserved in the reconstructions and whether they maintain some coherence in the intermediate representations, indicating good generalization and adaptability of the models to new wavelengths within the spectrum of interest. By presenting the reconstructions generated for this specific range, we sought to visually verify whether the models are overfitting the features of the training images or instead, are effectively extrapolating, which is the most decisive criterion for the practical application of spectral super-resolution. The visual consistency in this interval has shown us additional evidence to judge the precision and realism of the reconstructions generated by the SIREN and Fourier features models.

4.4 Conclusions

This PFG has been motivated by the need to obtain precise images at different wavelengths. Capturing this data is fundamental in various scientific fields, from astrophysics to biomedicine, but conventional methodologies limit us to discrete wavelengths, leaving significant gaps in our ability to analyze and comprehend numerous phenomena. This project has attempted to fill these gaps to provide a more complete and detailed image, specifically of the solar photosphere.



(a) Fourier features $\boldsymbol{\sigma} = [10, 10, 0.8]$; $k = 512$

(b) SIREN $\omega_0 = 30$

(c) Imágenes reales (GT)

(d) Fourier features $\boldsymbol{\sigma} = [10, 10, 0.8]$; $k = 128$

(e) SIREN $\omega_0 = 50$

Figure 4.12: Generated images corresponding to the intermediate wavelengths between λ_9 and λ_{11}. The selection of this range is based on the intersection of the intensity curves of the models in Fig. 4.11, providing a test case to examine the spectral interpolation capability and the fidelity of the reconstruction.

In this project, we worked with an existing spectral dataset, specifically the Fe I λ6302 absorption line, obtained with the CRISP instrument at the Swedish Solar Telescope in La Palma. These data consist of a series of 21 pseudomonochromatic images ($\lambda_1 - \lambda_{21}$) of 966×964 pixels (Fig. 4.1). The central focus of the project was to integrate these data into the weights of a neural network. For this purpose, fully connected neural network architectures were employed, and gradient descent algorithms were used in training. The main goal has been to evaluate the possibility of performing interpolations between images and achieving spectral super-resolution using neural fields, known as implicit representations.

The first step in our project involved replicating and evaluating experiments based on SIREN techniques [5] and Fourier features [6], applied to our solar images. The aim is to explore and understand the behavior of these advanced deep learning methods using our specific data. To

this end, we selected one image (λ_{12}) from our set of 21 images and extracted a central region of 512×512 pixels. This decision was made to maintain a balance between computational complexity and the ability to capture the details of the image, following the method used by other authors [6, 5] regarding the sample size.

After selecting this region, we proceeded to apply the SIREN and Fourier features models, aiming to investigate the efficiency of these techniques in the reconstruction and regression of two-dimensional images and validate them on our specific dataset. This approach allowed us to delve into differences in performance, sensitivity to parameters and hyperparameters, and the ability to generalize from the training data. The results obtained in the experiments indicate the viability of these techniques in the context of spectral super-resolution applications.

In the methodology used with SIREN, the hyperparameter ω_0 plays a key role, as it determines the frequency of the sinusoidal activation function used in the network. A higher value of ω_0 implies a greater capacity to capture high-frequency variations in the data, which is essential for accurately reproducing the fine details of the images. We observed that increasing ω_0, the signal-to-noise ratio in the training set experienced a progressive increase, reaching its peak of PSNR = 49.13 for $\omega_0 = 100$. However, in the test set, the most appropriate value was found to be $\omega_0 = 50$, with a PSNR = 38.50 (Table 3.2). This result reflects a balance between the model's ability to learn the details of the training data and its ability to generalize correctly to new data not seen before.

In the realm of Fourier features, the most relevant hyperparameter is σ, which acts as the standard deviation of the Gaussian distribution used to sample the matrix $\mathbf{B}$ in the feature mapping. This parameter is crucial as it regulates the amplitude of variability in the transformation of input coordinates, significantly affecting how the model interprets and processes the data's frequencies. An appropriate value of σ, as verified with $\sigma = 10$, enables the model to effectively capture image details, reflected in a PSNR = 39.02 in the test set. On the other hand, an excessively high σ, like the value $\sigma = 100$, can significantly distort this interpretation, leading to overfitting in the training data and a reduction in the model's ability to generalize to new data, as evidenced by a PSNR = 17.85 in the test set (Table 3.4).

In conclusion, the experiments carried out with SIREN and Fourier features techniques provided results that reinforce their potential for spectral super-resolution applications, particularly visible in the PSNR metrics shown in Fig. 3.7. The ability of both models to generalize from training data allowed us to progress towards our goal of synthesizing intermediate images that are not found in the cube of observed images and to verify the models' versatility in interpreting and complementing the spectrum of available data. Lastly, the quantitative evaluations, reflected in PSNR values, were consistent with the visual quality of the reconstructed images (Fig. 3.8). The generated images demonstrated high fidelity in reproducing the complex details of the solar photosphere.

With the results obtained in the regression of two-dimensional images, we approached the next step of our project: trying to compact the three-dimensional data cube to fill the spectral gaps produced by discrete sampling (Fig. 4.2). One of the challenges was to extend the experiment while keeping the network architecture unchanged, thus preserving the integrity of the implicit representations. Therefore, work in this phase focused on preparing the data for proper feeding into the network and adjusting the hyperparameters. The models had to reconstruct the images accurately and provide smooth transitions between the images generated along the wavelength

axis.

The preprocessing of the images consisted of normalizing the intensities to the interval $[0, 1]$ to adjust the values on the same scale, thus facilitating convergence during training and improving the stability and performance of the model. Additionally, we created a three-dimensional mesh that mapped all coordinates (x_i, y_j, λ_t) to values within the interval $[0, 1]$, allowing the network to learn both spatial and spectral patterns. This data preparation, along with the selection of hyperparameters, constituted our initial effort to achieve the neural networks' accurate reproduction of the set of 21 pseudochromatic images.

The PSNR results obtained with SIREN exceeded the threshold of 30 for both $\omega_0 = 30$ and $\omega_0 = 50$, as can be seen in Table 4.1, where the PSNR reached 35.02 for $\omega_0 = 50$ with the test set, indicating excellent reproduction of the original images. The validity of the SIREN network was also demonstrated in the visual comparisons of the reconstructed images with the originals, as shown in Fig. 4.5. The model's ability to faithfully capture spectral and spatial information is evident in the high visual similarity between the generated and real images. Additionally, Fig. 4.6 shows the evolution of the PSNR throughout the training, demonstrating stability and consistency in model performance, indicating its capacity for generalization, an important aspect for continuous spectral interpolation in the context of superresolution.

In our experiment with Fourier Features, optimizing the hyperparameters was essential to adapt the model to the three-dimensional complexity of the data. We focused on determining how the number of features k and the standard deviation σ affected the model's ability to capture details and complex patterns in the images. We selected values of k (128, 256, and 512) to assess their impact on the quality of the reconstruction and on training time. In addition, we tested various values of σ (0.8, 10, and 25) to examine how the Fourier transformation influenced the representation of the three-dimensional data. A significant finding was that a $\sigma = 0.8$ provided better results than higher values, indicating that a more concentrated Gaussian mapping is more suitable for the new three-dimensional structure. This discovery led us to consider σ not as a single value but as a vector $\boldsymbol{\sigma} = [\sigma_x, \sigma_y, \sigma_\lambda]$, allowing each dimension to have its own standard deviation. This approach reflects the unique nature of each dimension of the image cube and offers greater flexibility to capture the representative features of each axis. To determine the learning rate, we used a strategy that included an exponential search to identify an appropriate range and a uniform search for precise selection within that range. All selected parameters are detailed in Table 4.3. With them, we proceeded to train the network.

The use of a vector for $\boldsymbol{\sigma}$ combining different scales, particularly $\boldsymbol{\sigma} = [10, 10, 0.8]$, and the number of features $k = 512$, achieved the best performance, which was reflected in the PSNR metrics obtained both for training and test data. The PSNR values surpassed the threshold of 30, reaching a PSNR = 36.38, confirming the model's capability to effectively capture the complexity of the data. Additionally, the visual fidelity of the reconstructed images, shown in Fig. 4.9, was objectively high. This visual correspondence not only supports the quality of the reconstruction but also endorses the model's generalization ability.

We finally reached the moment to perform interpolations to evaluate how each model handles the generation of intermediate images along the wavelength axis. After obtaining comparable results in terms of PSNR with both techniques, as illustrated in Fig. 4.10, we set out to examine their capability for spectral superresolution. We implemented a technique of interpolating three-dimensional coordinates between pairs of consecutive images, which were then processed by the

trained models. This generated the intermediate images, representing specific steps in the spectral transition between the original images. This technique was applied to the four models discussed earlier, providing a detailed perspective on how each interprets these intermediate images. Fig. 4.11 compares the evolution of the mean pixel intensity between the models and the real images (GT), and reveals the accuracy of the models in reconstructing the image intensities. We observed that the Fourier features model with $k = 512$ and $\boldsymbol{\sigma} = [10, 10, 0.8]$ closely follows the trend of the real images, showing a high capacity for reconstruction without signs of overfitting. The SIREN model with $\omega_0 = 50$ also shows a good match with the values of the original images, although with some fluctuations that could be interpreted as variability in the reconstruction. The models with less complex parameters, such as SIREN with $\omega_0 = 30$ and Fourier features with $k = 128$, replicate the general trends but with less precision.

The generation of intermediate images in the interval between λ_9 and λ_{11} allowed us to visually assess the models' ability to handle complex spectral transitions. The images reconstructed by Fourier features with $k = 512$ (4.12a) and $k = 128$ (4.12d), as well as by SIREN with $\omega_0 = 30$ (4.12b) and $\omega_0 = 50$ (4.12e), were compared with the real images (4.12c). This comparison revealed that Fourier features with $k = 512$ achieves smoother interpolation and high fidelity in the reconstruction, while the model with $k = 128$ shows a slight decrease in coherence. Meanwhile, the images generated by SIREN reflect high fidelity, although with less visual consistency in spectral transitions than Fourier features, suggesting a possible tendency to overfit in certain cases.

Reflecting on the journey taken, this PFG has given us the opportunity to delve into neural fields. We have experimented with advanced techniques such as SIREN and Fourier features, which enhance the performance of gradient descent algorithms by applying them to a complex task like spectral image interpolation. We have tried to address the challenges this task presents and have been able to assess the models' ability to generate high-fidelity reconstructions and coherent intermediate representations. We have confirmed that careful selection of parameters, such as the standard deviation in the Fourier mapping and the frequency in the SIREN architecture, is essential for capturing the spectral and spatial variability of the data. The adaptability of these parameters is crucial, especially when extending our considerations to the additional dimension of wavelength. The visualizations generated have provided an essential tool for qualitatively assessing the coherence of spectral superresolution. By selecting a critical interval based on the intersection of the mean intensity curves, we have been able to examine the models' ability to handle complex spectral transitions and maintain the coherence of fundamental spectral features.

The completion of this work has enriched our theoretical and practical understanding of neural networks and spectral super-resolution, and its results would likely improve if we delved into some of the issues that have arisen during its development: What is the theoretical relationship, beyond mere intuition, between Fourier features and SIREN techniques in the context of neural fields? What other hyperparameter optimization techniques could we apply besides those used here? How could we evaluate and formalize the preparation and structuring of the data to optimize the performance of the models?

Finally, we hope that the methodologies tested in our experiments are applied to a wide range of datasets, with the aim of assessing the generalization capacity and the effectiveness of the proposed techniques. We trust that this PFG will serve both students and future developments in the field of spectral super-resolution, highlighting not only the considerable potential of

current approaches but also the promise of new discoveries in this fascinating and constantly evolving field.

Bibliography

[1] P. Bootcamp, "Section: Fabry-perot interferometer," 2019, [Online]. [Online]. Available: http://www.physicsbootcamp.org/section-fabry-perot-interferometer.html

[2] A. Tritschler, W. Schmidt, K. Langhans, and T. Kentischer, "High-resolution solar spectroscopy with tesos - upgrade from a double to a triple system," *Solar Physics*, vol. 211, p. 17, 2002.

[3] T. Kentischer, W. Schmidt, M. Sigwarth, and M. von Uexkuell, "Tesos, a double fabry-perot instrument for solar spectroscopy," *Astronomy & Astrophysics*, vol. 340, p. 569, 1998.

[4] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar, "Neural fields in visual computing and beyond," *arXiv preprint arXiv:2111.11426v4*, 2022. [Online]. Available: https://arxiv.org/abs/2111.11426v4

[5] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," in *Proc. NeurIPS*, 2020.

[6] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," in *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2020. [Online]. Available: http://arxiv.org/abs/2006.10739v1

[7] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of psnr in image/video quality assessment," *Electronics letters*, vol. 44, no. 13, pp. 800–801, 2008.

[8] Swedish Solar Telescope Team, "Crisp - crisp imaging spectro-polarimeter," Swedish Solar Telescope, La Palma, 2006. [Online]. Available: https://dubshen.astro.su.se/wiki/index.php/CRISP

[9] M. Faraday, "On the physical character of the lines of magnetic force," *Philosophical Transactions of the Royal Society of London*, vol. 142, pp. 1–20, 1852.

[10] J. C. Maxwell, "A dynamical theory of the electromagnetic field," *Philosophical Transactions of the Royal Society of London*, vol. 155, pp. 459–512, 1865.

[11] A. Einstein, "Die feldgleichungen der gravitation," *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Berlin*, pp. 844–847, 1915.

[12] P. A. M. Dirac, "The quantum theory of the emission and absorption of radiation," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 114, no. 767, pp. 243–265, 1927.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[15] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *Openai Blog*, vol. 1, no. 8, 2019.

[16] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[17] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[18] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[20] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[21] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," in *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, 2016, pp. 295–307.

[22] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *International conference on machine learning*, vol. 28, pp. 1139–1147, 2013.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[24] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[25] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, no. 2, pp. 26–31, 2012.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[28] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[29] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

[30] S. García, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining*. Springer, 2016.

[31] Z. Reitermanova, "Data splitting," *WDS'10 Proceedings of Contributed Papers*, pp. 31–36, 2010.

[32] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[33] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[34] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[35] J. Fourier, *The Analytical Theory of Heat*. Cambridge University Press, 1822.

[36] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1994. [Online]. Available: https://books.google.com/books?id=JZ3vQgAACAAJ

[37] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2019.

[38] NVIDIA, "Nvidia tesla v100 gpu architecture," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-v100/pdf/437451-Tesla-V100-DS-NV-US-WEB.pdf, 2020.

[39] C. Li, H. Zheng, and W. Tompkins, "Data preprocessing for machine learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 3, pp. 609–620, 2015.

[40] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[42] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[43] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[44] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in neural information processing systems*, 2007, pp. 1177–1184.

[45] J. Pons and X. Serra, "End-to-end learning for music audio," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 696–700.

[46] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[47] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, *Digital image processing*. Pearson Prentice Hall, 2009.

[48] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, "Siren: Implications of representational differences between neural and sinusoidal activations," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[49] N. Doshi, "Understanding activation functions in neural networks," *Medium, Towards Data Science*, 2018. [Online]. Available: https://towardsdatascience.com/understanding-activation-functions-in-neural-networks-9491262884e0

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[52] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[53] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[54] "Google colaboratory," Plataforma en línea, Google, 2021, accedido: 2024-04-10. [Online]. Available: https://colab.research.google.com/

[55] M. Riedmiller and H. Braun, "The backpropagation algorithm: A tutorial for practical application," *Neural networks*, vol. 6, no. 4, pp. 593–603, 1993.

List of terms and acronyms

Adagrad Adagrad Optimization Algorithm (Adaptative Gradient). 20

Adam Adam Optimization Algorithm (Adaptive Moment Estimation). 20, 25, 28, 29, 36, 41

Ca II Ionized Calcium. 3, XV

CRISP CRisp Imaging SpectroPolarimeter. 5, 45

Fe I Neutral Iron. 5, 6, 33–35, III, VII, XV–XVII

FF Fourier features. 31, 32, XVI

GD Gradient Descent. 20

H and K absorption lines Absorption lines in the solar spectrum associated with ionized calcium (Ca II). 3, XV

MAE Mean Absolute Error. 12

MSE Mean Squared Error. 12, 13, 25, 26, 28, 29

PFG Bachelor's Degree Final Project (Proyecto Fin de Grado). 4, 5, 44, 48, III, VII

PSNR Peak signal-to-noise ratio. 5, 25–31, 36, 37, 39–43, 46, 47, III, VII, XIII, XVI, XVII

ReLU Rectified Linear Unit. 9, 10

RGB Red, Green, Blue. 5

RMSprop RMSprop Optimization Algorithm (Root Mean Square Propagation). 20

SGD Stochastic Gradient Descent. 20

SIREN SINusoidal REpresentation Networks. 5, 17, 21, 22, 24–33, 36, 37, 42–48, III, V, VII, IX, XIII, XVI, XVII

tanh Hyperbolic tangent function. 9, 10

TESOS Triple Etalon Solar Spectrograph. 4

Appendix A

Analysis of the model cost

Introduction

In this appendix, we analyze the performance of the models implemented in the conducted experiments. The temporal cost, i.e., the computational time required to train and execute these models, and the spatial cost, related to the storage of their parameters and intermediate data structures, are important for assessing their viability and efficiency in practical applications.

Normally, the cost analysis of algorithms is performed using Big O notation (O), which allows for establishing a conceptual relationship between the input size n and complexity, both in terms of operations and memory. However, although Big O notation provides useful insights into the asymptotic behavior of algorithms, practical considerations such as network architecture, activation functions, optimization algorithms, regularization methods, and the use of parallelization on specific hardware also affect the complexity of training. Therefore, we will begin the analysis by briefly describing the infrastructure used. Then we will provide an approximate calculation of the temporal and spatial costs required for the execution of the models. Finally, we will present the empirical data obtained through PyTorch Profiler, offering a detailed and practical view of the real performance of the models in our experiments.

A.1 Infrastructure used

For the development of this PFG, we have used Google Colab [54], a research platform that facilitates the execution of Python code in the browser through a Jupyter notebook environment without requiring any prior setup. This tool operates on Google's cloud infrastructure and provides access to advanced computational resources, such as CPUs and GPUs. Due to its accessibility and user-friendly interface, Google Colab is widely adopted in academic and research environments for machine learning and data science applications.

Google Colab provides a range of hardware resources, including standard CPUs and GPUs, like NVIDIA Tesla T4, L4, V100, and A100. Built on Jupyter Notebooks, it allows users to write, execute, save, and share code easily. Additionally, it integrates with Google Drive and other services, facilitating direct data upload and storage.

Google Colab employs a pricing model based on the use of *compute units* (CUs). Prices vary depending on the amount of resources consumed, such as CPU, GPU, or RAM, and is contingent on the geographical area where the service is accessed. In the Canary Islands, Spain, the cost of 500 CU is 42.25 euros. Table A.1 compares the costs in CU with the corresponding usage hours and their cost in euros as well as the characteristics of each model.

Infrastructure	Cost (CU/hour)	Usage Hours	Price (euro/hour)	TFLOPS Operations/s	System RAM	GPU RAM
CPU	0.07	7,142.86	0.01	-	14 GB	-
	0.13	3,846.15	0.01		55 GB	
T4 GPU	1.76	284.09	0.15	8.1 TFLOPS	14 GB	15 GB
	1.84	271.74	0.16		55 GB	
L4 GPU	4.82	103.73	0.41	7.4 TFLOPS	64 GB	22.5 GB
V100 GPU	4.82	103.73	0.41	14.1 TFLOPS	14 GB	16 GB
	4.91	101.83	0.41		55 GB	
A100 GPU	11.77	42.48	0.99	19.5 TFLOPS	90 GB	40 GB

Table A.1: Comparison of infrastructures available on Google Colab, including operational costs and processing capabilities.

A.2 Temporal cost

The **temporal cost** refers to the time required for an algorithm to complete. It is generally measured in terms of the number of fundamental operations (comparisons, assignments, accesses to data structures, etc.) performed by the algprithm as a function of the input size n.

SIREN

For the SIREN model depicted in Fig. 3.1 and Fig. 4.4, this cost is analyzed in terms of the number of operations required by each layer during forward and backward propagation through the network.

During forward propagation, each dense layer performs matrix operations to transform inputs into outputs. The complexity of a matrix operation is proportional to the number of multiplications and additions performed.

- For the first layer, with an input size of $n = 3$ and $m = 256$ neurons, the total number of operations is $n \times m = 3 \times 256 = 768$.
- For each of the intermediate layers, with $n = m = 256$, the total number of operations is $n \times m = 256 \times 256 = 65,536$.
- For the output layer, which converts an input of $n = 256$ to an output of $m = 1$, the total number of operations is $n \times m = 256 \times 1 = 256$.

By summing up the total number of operations across all layers, we obtain an estimate of the total operations required per input. That is, for a single set of coordinates (x_i, y_i, λ_i), the model performs approximately $768 + 65536 \times 2 + 256 = 132,096$ operations.

During the backward propagation, necessary for the training of the model, gradients with respect to the weights and biases of each layer are calculated. This involves additional operations

that, in general, are k times the operations performed during forward propagation ($k \geq 2$) [55]. Considering both, the total number of operations, in the best-case scenario ($k = 2$), approximates to $132,096 \times 3 = 396,288$ for one training step of a sample.

Fourier features

For the Fourier features model with the sequential architecture shown in Fig. 3.4 and Fig. 4.7, the calculation is performed in the same manner, with the exception that the first layer may be composed of either 128 or 512 neurons depending on the mapping performed. If we consider a first layer with 512 neurons:

- For the first layer, with an input size of $n = 512$ and $m = 256$ neurons, the total number of operations is $n \times m = 512 \times 256 = 131072$.
- For each of the intermediate layers, with $n = m = 256$, the total number of operations is $n \times m = 256 \times 256 = 65536$.
- For the output layer, which converts an input of $n = 256$ to an output of $m = 1$, the total number of operations is $n \times m = 256 \times 1 = 256$.

The total number of matrix operations for forward propagation is approximately $131072 + 65536 \times 2 + 256 = 262400$. Considering that the number of operations for backward propagation is approximately double ($k = 2$), the total temporal cost to train one sample is approximately $262400 \times 3 = 787200$ operations.

A.3 Spatial cost

The **spatial cost** refers to the total amount of memory required by an algorithm during its execution. This includes both the static space (the code and fixed variables) as well as the dynamic space (the memory used by variable data structures).

SIREN

For a dense layer (fully connected) that transforms an input of dimension n to an output of dimension m, the number of parameters (weights and biases) is $n \times m + m$.

In the context of the SIREN model and assuming that each parameter is stored as a `float32` (4 bytes), the total spatial cost is as follows:

- The first dense layer, which accepts the input of the model $n = 3; m = 256$, has a cost of $(n \times m + m = 3 \times 256 + 256) \times 4 = 4096$ bytes.
- Each intermediate layer where $n = m = 256$ has a spatial cost of $(n \times m + m = 256 \times 256 + 256) \times 4 = 263168$ bytes.

- The last dense layer that produces the output of the model $n = 256; m = 1$, has a cost of $(n \times m + m = 256 \times 1 + 1) \times 4 = 1028$ bytes.

The model must store a total of $(3 \times 256 + 256) + 2 \times (256 \times 256 + 256) + (256 \times 1 + 1) = 132865$ parameters and its spatial cost is $132865 \times 4 = 531460$ bytes = 519 kB. If we consider that the spatial cost of the gradients is approximately the same as that of the parameters, then the total cost is 1138 kB $\approx$ 1 MB.

Fourier features

In the model with Fourier features, the spatial cost is equally determined by the total number of parameters, including weights and biases, stored as `float32` (4 bytes per parameter).

- The first dense layer, which accepts the input of the model $n = 512; m = 256$, has a cost of $((n + m) \times m = 512 \times 256 + 512) \times 4 = 525312$ bytes.
- Each intermediate layer where $n = m = 256$ has a spatial cost of $(n \times m + m = 256 \times 256 + 256) \times 4 = 263168$ bytes.
- The last dense layer that produces the output of the model $n = 256; m = 1$, has a cost of $(n \times m + m = 256 \times 1 + 1) \times 4 = 1028$ bytes.

The model must store a total of $(512 \times 256 + 512) + 2 \times (256 \times 256 + 256) + (256 \times 1 + 1) = 263169$ parameters and its spatial cost is $263169 \times 4 = 1052676$ bytes $\approx$ 1028 kB $\approx$ 1 MB. Considering, as before, that the spatial cost of the gradients is approximately the same as that of the parameters, the total cost is 2056 kB $\approx$ 2 MB. These calculations allow us to anticipate the growth of the spatial cost of the model depending on the architecture of its layers.

A.4 Performance obtained with *pytorch profiler*

PyTorch Profiler is a tool integrated into the PyTorch library that allows analyzing and optimizing the performance of neural networks during training. It provides an analysis of the time and resource use at each step of the process, offering information about the computational cost of the network's different operations. It enables the identification of bottlenecks and other issues to optimize the performance of models.

In this section, we present the performance data obtained during the training of experiments 4.2.1 and 4.2.2, using the lowest-range GPU model available, the Tesla T4, and the highest-range one, the A100. The visualization of these data has been done with TensorBoard[1].

[1]The rest of the data obtained with PyTorch Profiler for each experiment and for each GPU model (T4, L4, V100, and A100) are located on the DVD that accompanies this project, along with the instructions for viewing them.

GPU performance and usage time comparison

The parameters measured in this section are the following:

- **GPU utilization**: Indicates the percentage of time the GPU is actively performing calculations.
- **Estimated SM efficiency**: Represents the efficiency of the Streaming Multiprocessors (SM), which are the cores that execute the kernels on the GPU.
- **Estimated achieved occupancy**: Measures the efficiency in the use of hardware resources on the multiprocessors.
- **Average step time**: Average time taken for a complete training step, including the forward and backward pass, and in our case, the evaluation time (test).
- **Kernel (%)**: Percentage of time dedicated to executing kernels on the GPU.
- **Memcpy (%)**: Time spent copying data between the CPU and GPU memory.
- **CPU Exec (%)**: Time spent on calculations performed on the CPU.

Parameter	**Fourier features** $\boldsymbol{\sigma = [10, 10, .8]}$ $\boldsymbol{k = 512}$		**SIREN** $\boldsymbol{\omega = 50}$	
GPU	Tesla T4	NVIDIA A100	Tesla T4	NVIDIA A100
Memory (GB)	14.75	39.56	14.75	39.56
Compute Capability	7.5	8.0	7.5	8.0
Number of Workers	4	4	4	4
GPU Utilization (%)	97.91	92.11	97.67	91.98
Est. SM Efficiency (%)	97.87	91.46	97.87	91.86
Est. Achieved Occupancy (%)	66.24	44.41	69.75	53.25
Average Step Time (μs)	2,359,548	499,543	1,994,978	402,831
Kernel (%)	97.91	92.11	97.67	91.98
Memcpy (%)	0.03	0.01	0.03	0.04
CPU Exec (%)	0.59	4.2	0.51	2.91

Table A.2: Comparison of parameters between Fourier features and SIREN with GPUs T4 and A100

The results from Table A.2 show that the NVIDIA A100 GPU is significantly faster than the Tesla T4, with substantially lower average step times. However, the achieved occupancy on the A100 is lower, which might be improved with adjustments to the kernels and batch size. Lastly, the SM efficiency and kernel usage remain high on both GPUs, indicating effective use of resources. Fig. A.1 shows the view with the general data obtained during the execution of the Fourier features model with a $\sigma = [10, 10, .8]$ and $k = 512$ on the A100 GPU.

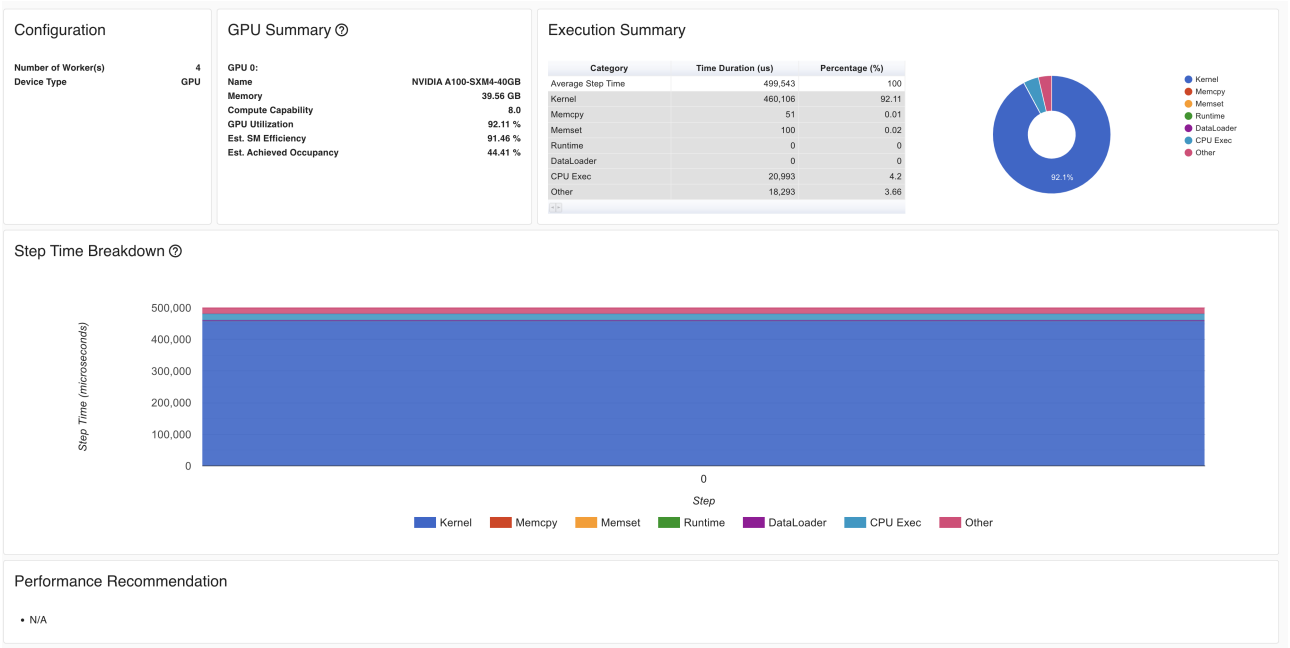


Figure A.1: TensorBoard view of the execution summary for the Fourier features model using the A100 GPU on Google Colab.

Parameter	Fourier features (T4)	Fourier features (A100)	SIREN (T4)	SIREN (A100)
Peak Memory Usage (MB)	5,297.1	5,294.1	6,400.9	18,915.4
Allocated Memory (MB)	Variable during training			
Reserved Memory (MB)	Variable during training			

Table A.3: Comparison of memory usage between Fourier features and SIREN with GPUs T4 and A100

Memory usage comparison

The comparison of memory usage between the SIREN and Fourier features models on T4 and A100 GPUs reveals the following key parameters:

Table A.3 illustrates the highest peak of memory usage during the training of the SIREN model on the A100 GPU, while the Fourier features model demonstrates more moderate usage on both GPUs. This is can be attributed to the considerably larger batch size employed in training the SIREN model. The allocated and reserved memory vary during training owing to PyTorch's

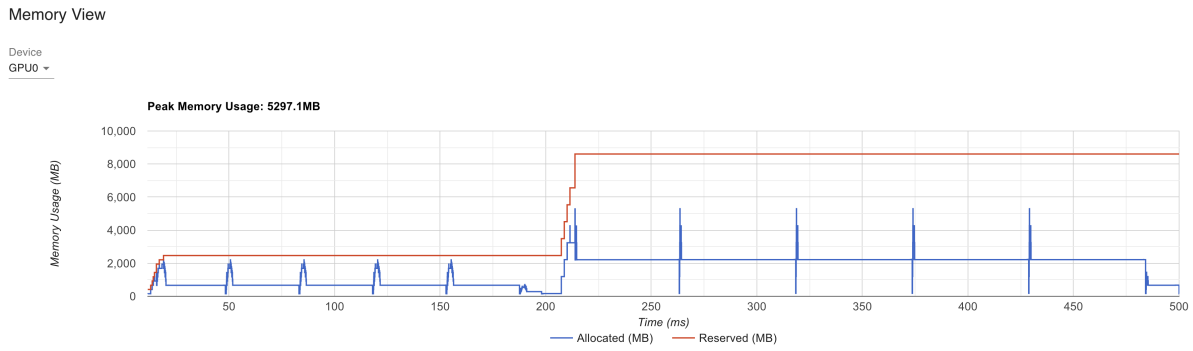


Figure A.2: TensorBoard view of the allocated and reserved memory usage for the Fourier features model using the A100 GPU on Google Colab.

memory management. Fig. A.2 shows this variation in the Fourier features model.