**COMPLUTENSE UNIVERSITY OF MADRID**

**FACULTY OF PHYSICAL SCIENCES**

DEPARTMENT OF COMPUTER ARCHITECTURE AND AUTOMATION


# BACHELOR'S THESIS

Thesis code: DACYA07

Detección de defectos con Deep Learning en imágenes industriales tomadas con rayos X

Defects detection in X-ray industrial images using Deep Learning

Supervisors: María José Gómez Silva and Jesús Chacón Sombría

**Pablo Suárez Reyero**

Degree in Electronic Communications Engineering

Academic year 2022-2023

JULY Call

*"If you have knowledge, let others light their candles in it.".*

---

*Margaret Fuller*

## Acknowledgments and dedications

As I mentioned in the previous page, if you have knowledge, you have to let others light their candles on it, in my case, thanks to the knowledge of others and the knowledge I have acquired in this career, I have been able to develop this work. I hope that in spite of the errors that this thesis may have, it will serve for others to light their candles on it.

This work is dedicated to my mother, a great woman and a great father, fighter and brave. Thank you for teaching me the importance of love and truth. To my grandmother, my second mother, always on the bench waiting to go out to the field when something bad happens. To my siblings, Sara and Mario, thank you for your experiences and your generosity, although I don't usually tell you, I love you [1]. To my friends and to all those who make me better and who keep a piece of me in them. Thanks to María and Jesús, for having guided me and for those sudden meetings. Thanks to God, for giving me strength and for making me believe in myself. Thanks to those teachers, fair and honest, who made their classes spaces of free thought.

Thank you from the bottom of my heart.

[1] just a little bit

## Resumen

Desde su origen, los rayos X han contribuido de manera notable al desarrollo y progreso de la ciencia. Su papel ha sido fundamental en el descubrimiento de nuevos fenómenos físicos o de nuevas propiedades en materiales de diversa índole. Pero es su uso constante e imprescindible en el campo de la medicina el que ha otorgado el reconocimiento y la fama que actualmente tienen. [Pel12].

Gracias a su potencial, su uso se ha ido extendiendo paulatinamente a diferentes ámbitos, siendo el ámbito de la seguridad y el de las cadenas de producción uno de los principales beneficiarios de este fenómeno ionizante. En el campo de la seguridad su uso es más que evidente, por ejemplo a la hora de abordar un avión o un tren, donde el objetivo de su implementación es prevenir la introducción de armas de cualquier tipo. Dentro del mundo industrial su uso es muy común, pues la integridad de piezas industriales como pueden ser las ruedas de un automóvil o su chasis depende de los rayos X y garantizar la integridad de cada una de estas piezas es crucial, pues de ello depende la vida de millones de personas.

Como los defectos de fabricación en aleaciones suelen ser pequeños, la tarea de detección de los mismos es ardua y lenta si es llevada a cabo por personas que discriminan los defectos rudimentariamente. Esto requiere una gran cantidad de tiempo que se traduce en pérdidas para la empresa. De aquí nace la motivación de este TFG, conseguir detectar defectos en radiografías más rápidamente y de manera automatizada con ayuda de la inteligencia artificial.

Para lograr una correcta y eficaz detección, se ha recurrido al uso de técnicas de *"deep learning"*, i.e., técnicas de aprendizaje profundo. En esto consiste el trabajo de este TFG, en el desarrollo de una red neuronal, capaz de detectar defectos en piezas industriales. Para lograr esto, se ha seleccionado un conjunto de imágenes de radiografías, con las que trabajar, i.e., imágenes con las que se va a entrenar la red y otro conjunto con el que se someterá a prueba la misma. A lo largo de esta memoria, se verá como el entreno de la red, requiere un procesado previo de la información, así como la aplicación de múltiples transformaciones, esto será acometido por una arquitectura de red que recibe el nombre de *Faster R-CNN*.

Es importante destacar la diferencia entre la detección de objetos y la detección de defectos, siendo esta última más complicada, debido a su tamaño y proporción respecto a la imagen. En esta memoria se estudia en detalle las dificultades que ello acarrea a la vez que se intenta dar solución a esta cuestión.

## Abstract

Since their origin, X-rays have contributed significantly to the development and progress of science. Their role has been fundamental in the discovery of new physical phenomena or new properties in materials of various kinds. But it is their constant and essential use in the field of medicine that has given them the recognition and fame they currently enjoy.[Pel12].

Thanks to its potential, its use has gradually spread to different fields, with the field of safety and production lines being one of the main beneficiaries of this ionizing phenomenon. In the field of security its use is more than evident, for example when boarding an airplane or a train, where the objective of its implementation is to prevent the introduction of weapons of any kind. Within the industrial world its use is very common, since the integrity of industrial parts such as the wheels of an automobile or its chassis depends on X-rays and guaranteeing the integrity of each of these parts is crucial, since the lives of millions of people depend on it.

As manufacturing defects in alloys are usually small, the task of detecting them is arduous and time-consuming if carried out by people who discriminate defects rudimentarily. This requires a great deal of time that translates into losses for the company. This is the motivation of this project, to detect defects in radiographs faster and in an automated way with the help of artificial intelligence.

In order to achieve a correct and efficient detection, we have resorted to the use of deep learning techniques. The work of this thesis consists of the development of a neural network capable of detecting defects in industrial parts. To achieve this, a set of X-ray images has been selected to work with, i.e., images with which the network will be trained and another set with which it will be tested. Throughout this memory, it will be seen how the training of the network requires a previous processing of the information, as well as the application of multiple transformations, this will be undertaken by a network architecture that receives the name of *Faster R-CNN*.

It is important to highlight the difference between object detection and defect detection, the latter being more complicated, due to its size and proportion with respect to the image. In this project we study in detail the difficulties that this entails and at the same time we try to find a solution to this issue.

# Contents

# 1 Introduction

This section introduces the motivation of the project, the objectives and the structure of the document. At the end of the section a Gantt chart has been attached to illustrate chronologically the development of the project.

## 1.1 Motivation

The task of detecting faults in x-ray images of industrial parts is slow and laborious, even more so when dealing with small defects, which is precisely what concerns this project. Currently, the authorship of this task has not changed, and it is still a worker who analyzes image by image, which involves a great deal of time and qualified personnel. Moreover, since it is a person who searches for these defects, he/she has his/her own criteria, apart from the one acquired in his/her training.

Over the last few years, it has been demonstrated how *deep learning* and more specifically, neural networks, have a great computational potential that can be extrapolated to any field that requires the technology for its development. It is important to bear in mind that, although they are useful tools, they do not reach a threshold of functionality until they are properly trained. Despite its negative points, the potential of *deep learning* is considerable, and for this reason, it has been chosen to undertake the detection task that occupies this project.

The defects to be detected are very small, hence the difficulty of this task. Therefore, in order to overcome these difficulties, the neural network must be very accurate in order to minimize the localization error. For this, the choice of a correct architecture and the components that form it (types of networks, layers, algorithms, etc.) is fundamental, so that the network does not lose sight of these defects and can easily detect them, since in some cases the difference between "*there is defect*" and "*there is no defect*" is a couple of pixels.

It is important to note the lack of detection models that pursue the goal of this work, as opposed to the wide range of detection models that focus on object search and classification. Therefore, the motivation of this project is to understand the performance of current detection techniques and their correct implementation, in order to adapt them to the mission of this work and to be able to delegate this task to an artificial intelligence.

## 1.2 Goals

Considering that the purpose of this project is the identification of defects in x-ray images, and taking into account the usefulness of neural networks for the achievement of the same, the objectives of this project are the following:

- To understand the theoretical basis of a neural network.
- Correctly preprocess the images to be analyzed.
- Choose the analysis techniques best suited to the problem.
- Assess the functionality of the network and its architecture.

## 1.3 Relationship to bachelor subjects

The knowledge acquired throughout my bachelor's degree has facilitated the development of this work and the understanding of concepts, theorems and techniques implemented in this project.

The subjects that directly or indirectly contributed to the development of this thesis are listed below.

- **Signal Processing**. The knowledge of this subject with respect to the rest of the courses mentioned below is the most similar to the objectives and concepts put into practice in the project. In this course, filters and signal processing techniques such as convolutions have been studied in depth, both for $\mathbb{R}^n$ (vectors [2]) and for $\mathbb{R}^{n \times m}$, i.e., images.

- **Algebra and Systems Optimization**. Much of the content discussed in these two subjects forms the theoretical basis of neural networks. The use of vectors and matrices is the order of the day, e.g., the linear combination of the $n$ input values of a network and their associated weights. Analyzed images [3] are interpreted as tensors and the neural network itself forms a tensor. In addition, one of the main objectives when dealing with neural networks is the optimization of the cost functions. [4], aiming to minimize them, this is what the Systems Optimization course is all about: understanding, mastering and applying different optimization techniques.

- **CS** and **Operating and real-time systems**. These two subjects have played a key role in the development and understanding of the implemented code. The fundamentals of programming have been studied, with special emphasis on the use of memory and process management. Although we have not studied object-oriented programming languages, e.g., Python, we have programmed in C and C++, low-level languages, where the programmer has a greater responsibility.

## 1.4 Document structure

This document is organized as follows:

- **State of the art**. Different current detection methods are studied, as well as the tools that allow it.

- **Methodology**. The techniques implemented are explained, as well as the architecture that has been chosen.

- **Detector modeling**. As its name indicates, the process followed for the network to make a proposal of regions and subsequently detect defects is explained.

- **Training**. In this section we go into detail about the parameters that influence the training process and the metrics that allow to characterize it.

- **Results and corresponding analysis**. This section explains the tests performed, states the evaluation criteria and presents the results, which are subsequently analyzed.

- **Conclusions and future work**. Once the results have been analyzed, the effectiveness of the techniques implemented is studied and an analysis is made of possible improvements.

[2] Vectors are one-dimensional tensors of values.

[3] Images are matrices, which are interpreted as two-dimensional tensors, in the case of black and white. If they are RGB images, then they are described as three-dimensional tensors.

[4] The weights represent the intensity of interaction between each neuron. For more information on this topic, please see the appendix 9.

## 1.5 Gantt Chart

Attached below is the Gantt chart reflecting the time taken for each task in the development of this thesis.

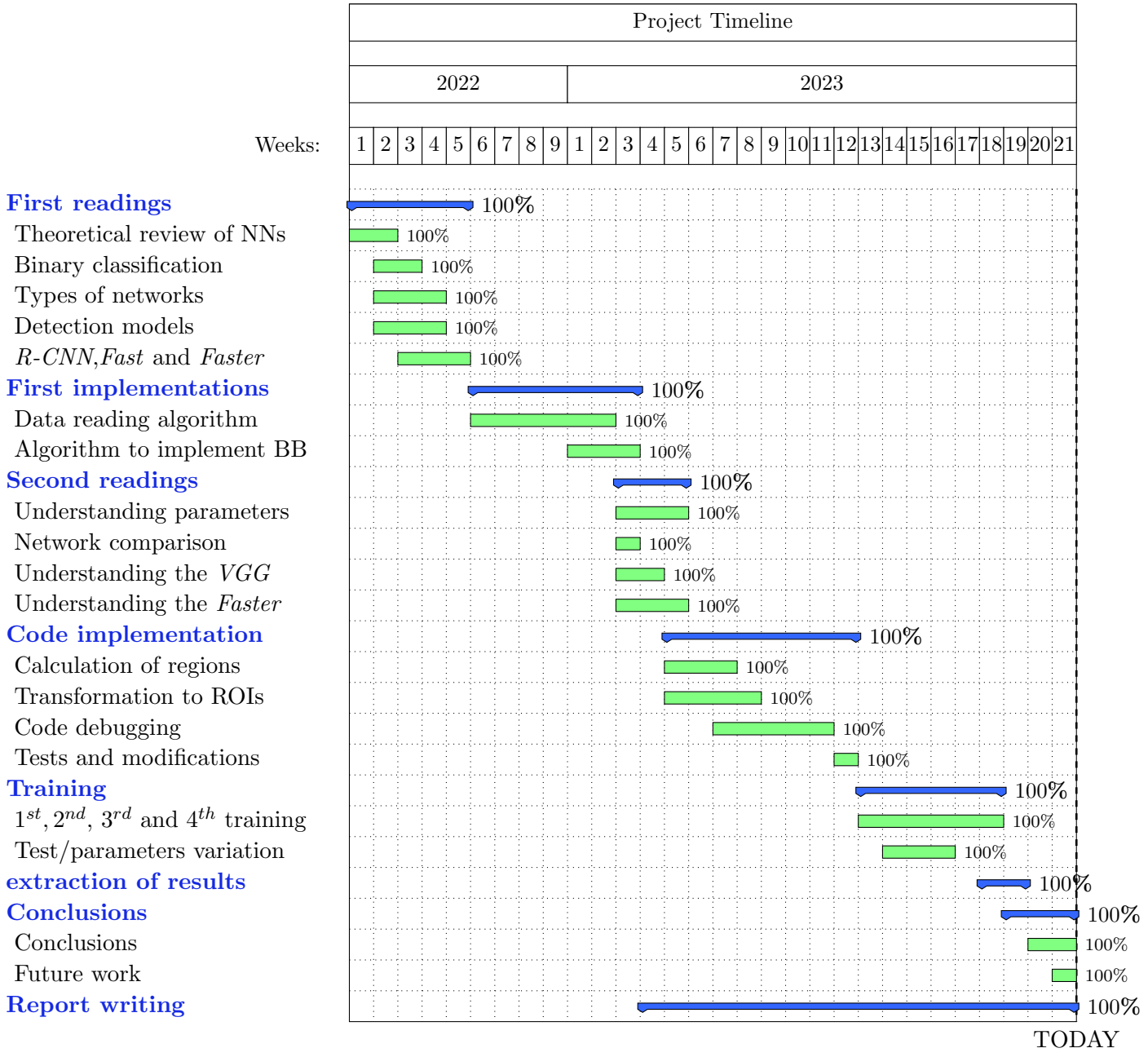



Figure 1: Gantt Chart

# 2 *State of art*

A correct defect detection depends mainly on the network type, the implemented architecture, and the algorithms that are in charge of proposing regions in order to hypothesize about defect localization in images.

In this section, we study the options related to the network type and architecture that could be implemented, taking into account the objectives of this dissertation. Additionally, in the section 9, corresponding to the appendix, the theory that has allowed a better understanding of all the concepts explained in this section can be found. However, it is not mandatory to read what is explained in the annexes in order to understand this project, but it is recommended to consult them to clarify any doubts that may arise throughout this document.

## 2.1 Type of neural network

Inside the *deep learning* world there are different variants of neural networks, each serving a specific set of applications. The four most widespread types of neural networks and their characteristics are listed below.

- **Feed Forward Neural Network.** It is one of the most basic networks, where information passes through several input nodes in one direction until it reaches the output node. The network may or may not include layers of hidden nodes. In addition, this type of network can be implemented for pattern recognition and computer vision, but are not suitable for deep learning. KnowledgeHut et al. [Kno]

- **Radial basis function network.** This is a special kind of neural network consisting of only three layers, input layer, hidden layer and output layer. As is evident from its name, it uses radial basis functions (RBF) such as Gaussian, multi-quadratic, etc., as the activation function for the hidden layers. This type of network is often used for function approximation and time series prediction. ProjectPro et al. [Pro]. Moreover, according to Bonnano et al. [BCN+13] they may require more neurons than **Feed Forward Neural Networks**, but can often be designed in a fraction of the time it takes to train the previous type of network.

- **Recurrent neural networks.** These networks are constructed to understand temporal or sequential data. RNNs improve their predictions by using additional data points in a sequence. To modify the output they take input data and reuse activations from earlier or later nodes in the sequence. It is relevant to mention that this type of networks suffer from the gradient fading problem. KnowledgeHut et al. [Kno]

- **Convolutional neural networks.** According to Zewen Li et al. [LYPL20] is one of the most most significant networks in the field of deep learning. The same author describes convolutional neural networks as networks capable of extracting features from data with convolutional structures, capable of extracting features from data with convolutional structures. Moreover, these types of networks do not extract features manually.

  Shengli Jiang et al. [JQPZ22] recall the initial purpose of this type of network, which was and still is computer vision, and then highlight how flexible this type of network is for representing data, and how this quality has allowed its implementation in many other fields apart from computer vision, such as in the chemical industry or in the field of biology.

  This type of network is the most suitable for object classification and detection, because

it is capable of processing multi-dimensional matrices, e.g., tensors. In the case of color images, their mathematical representation will be characterized by three matrices stacked one after the other. Each of these matrices represents a channel that is mathematically described as a two-dimensional tensor, and by stacking three matrices, one has a 3-dimensional tensor. In the case of having a black and white image you would have a single matrix that would have the dimensions of the original image, see figure 2

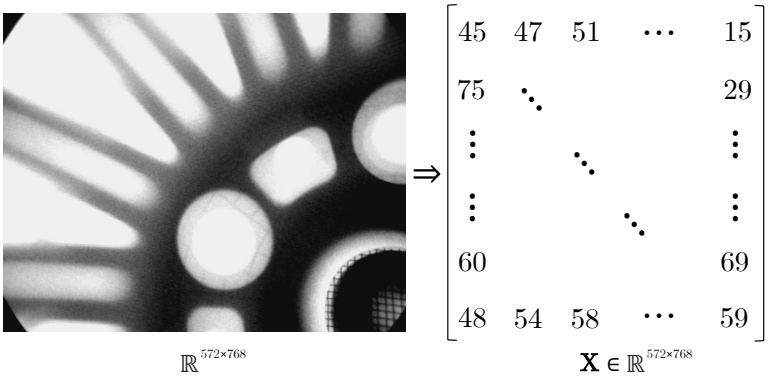



Figure 2: Black and white image of a casting with its pseudo matrix representation.

The network receives an image at the input (as shown in figure 2) and its characteristics are extracted by convolution. From subjects such as Linear Systems or Signal Processing, it is known that convolution can be defined as follows:

$$p(x) = \mathbf{w} * \mathbf{X} = \sum_{a=-s}^{s} \mathbf{w}(a)\mathbf{X}(x-a) \tag{1}$$

Where $\mathbf{X} \in \mathbb{R}^{n\times m}$ represents the input image, $\mathbf{w} \in \mathbb{R}^{i}$ represents the weights and $s$ is determined by the filter's size which represents the *stride* [5].

Given that the aim of this project is to detect defects, by means of the use of the *deep learning*, once the previous options have been reviewed, the choice is not difficult. Therefore, the implementation that has been carried out focuses on the use of convolutional networks.

In order to detect the defects in the images, the next objective of this work is to implement a network architecture based on convolutional neural networks. In section 2.4, the existing architectures in terms of convolutional networks are studied and the functionalities offered by some of the most relevant ones in the current scenario are analyzed. For this purpose, the fundamental concepts that allow their operation must be previously explained.

## 2.2 Concepts and key elements

Before investigating about the functioning of different network models and explaining their dynamics, it is pertinent to explain the concepts that characterize any architecture and that make

[5]This concept is explained in (2.2)

possible the extraction of its most relevant aspects, in order to subsequently create its feature map [6].

One of this concepts is the *stride*. The *stride* represents the number of pixel offsets over the input matrix. When the *stride* is 1 then the convolution filters move 1 pixel at a time. When the *stride* is 2 then the filters move 2 pixels at a time and so on. In figure 3 this concept is intuitively represented on a matrix of values, which can correspond to a black and white image.

Now suppose that this image is of the form $\mathbf{X} \in \mathbb{R}^{w \times w}$ and that a filter defined as : $\mathbf{K} \in \mathbb{R}^{n \times n}$ is applied on it. When computing the convolution between them, the resulting image will have the following size : $(w - n + 1) \times (w - n + 1)$.

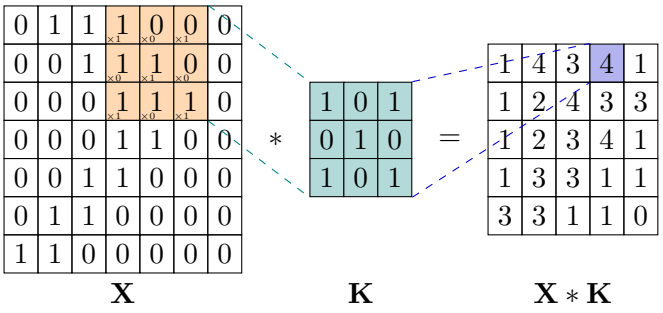



Figure 3: Representation of a convolution. StackExchange et al. [Staa]

It is important to mention how convolving two elements introduces two problems:

- When convoluted, the size of the original image is reduced, take for instance; $w = 6$ and $n = 3$. After applying the convolution, it goes from $\mathbf{X} \in \mathbb{R}^{6 \times 6}$ to $\mathbf{X} \in \mathbb{R}^{4 \times 4}$. As will be seen in the section on network architecture, in the image classification task there are multiple layers of convolution, so that by applying this operation several times, the original image will have been considerably reduced, and this is something we want to avoid.

- The second problem occurs when the kernel[7] is moved over the original images, it passes through the center many times, while it passes through the edges much less frequently. As it passes through the center so many times, it overlaps, the opposite of what happens at the edges, and as a consequence, the information it contains is almost not reflected at the output.

In order to solve these two problems, the concept of *padding* is introduced. This method consists of surrounding the original image with zeros, so that when convolution is applied, the final size of the image is equal to the original. When applying a *padding* $\mathbf{p}$[8] to the image, and by convolving it with a filter of size $n \times n$ and with a *stride* $\mathbf{s}$[9], el tamaño final de la imagen será: $(\frac{w-n+\mathbf{p}+\mathbf{s}}{\mathbf{s}}) \times (\frac{w-n+\mathbf{p}+\mathbf{s}}{\mathbf{s}})$

[6]The feature map of an image is the output of a convolutional layer that represents specific features of the input image.[bae]

[7]A *kernel* in the field of neural networks, represents the filter that is applied to the images, with the objective of extracting certain important characteristics. BootCampAI et al. [Boo]. Not to be confused with the *kernel* of an operating system.

[8]Apply a padding $\mathbf{p}$ to any given image, means to add $\mathbf{p}$ rows full of zeros and $\mathbf{p}$ columns full of zeros.

[9]A *stride* of equal magnitude is assumed when moving the filter from right to left as when moving it from top to bottom.

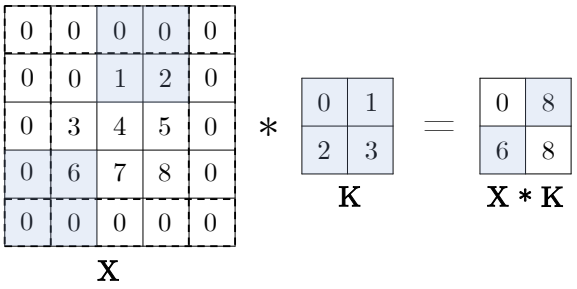



Figure 4: An image convolution with*padding* $\mathbf{p} = 1$. Figure inspired by D2l et al. [D2l]

By applying this technique, the image would not lose its original size, which is precisely what we are looking for.

It is important to notice how in section 3.1.3 we return to the *stride* but although the two terms share a name, they do not represent the same thing. In the context of this section, the term *stride* refers to the size of the filter, while in the 3.1.3 section, it represents the reduction factor of the original image, which is directly linked to the depth of the network, and thus to the number of filters applied to the image in the *backbone*.[10].

## 2.3 Detection models

This section briefly introduces the most relevant detection techniques within the field of computer vision, which are fundamental for the understanding of this thesis.

It is worth mentioning that the implemented network structure (i.e., the *Faster R-CNN*) is the result of an evolutionary process that begins with the **R-CNN** network. Subsequently, in section 3.1, the **Faster R-CNN** architecture is explained in depth.

Since its conception as a scientific discipline, computer vision has always been at the forefront of technology, because as computational capacity advanced, so did the potential of this branch of knowledge. Although there have been many advances in this field after its consolidation in the scientific community, one of the most important milestones was brought by the family of machine learning models: **R-CNN**, whose acronym stands for Region-based Convolutional Neural Networks.

This conglomerate of models that make up the **R-CNN** are focused on computer vision, and have object detection as their main task. The initial purpose of their use lies in the production of a set of *bounding boxes* that frame the objects in an image, but since then, the way of obtaining these frames has changed drastically, allowing this family of algorithms to be extended to other areas of computer vision.

This little revolution began with the launch of the **R-CNN** network in 2013. Given an input image, this architecture **R-CNN** starts by applying a so called "selective search " mechanism[11] to extract regions of interest (*ROIs*), where each **ROI** is represented with a rectangle that can outline the limit of an object in the image. This model applies 2000 times the neural network, thus, it

[10]This term refers to the heart of the network architecture.
[11]In section 3 this algorithm is briefly explained.

could be more than 2000 *ROIs*. Wikipedia et al, [Wik22]

Subsequently, in April 2015, its successor was presented, which would bring the greatest improvement to date in this family of models, the **Fast R-CNN**. This architecture undertakes a considerable improvement computationally speaking, because unlike the first version, the **Fast R-CNN** implements the neural network on the image only once. In addition, this network introduces a set of *pooling*[12] layers at the tail of the architecture. As in the original R-CNN, the **Fast R-CNN** uses selective search to generate its proposed regions. Wikipedia et al, [Wik22]

Two months later, in June 2015, the prominence of the **Fast R-CNN** would be slightly eclipsed by a new member of the family, the **Faster R-CNN**. Although there is constant talk of extracting the maximum potential from artificial intelligence, this new model fulfills this premise, because instead of implementing the selective search of the two previous versions, in this architecture the search for regions of interest is undertaken by including the *ROI pooling* layer in the neural network itself. Wikipedia et al, [Wik22]

In order to grant a higher search accuracy to the network, in 2017 the **Mask R-CNN** is presented, this model focuses on increasing the search resolution by replacing the *ROI pooling* layer, by the *ROIAlign* which is able to represent fractions of a pixel. This model is attractive for implementation in this final degree work, because as explained in section 2.4, the difficulty of this project lies in the size of the defects to be detected, and this network could overcome these resolution constraints. But this model, compared to the previous one, introduces an additional level of complexity that has not been considered appropriate to address in this thesis.

And lastly in June 2018, the **Mesh R-CNN** was developed, a new architecture able to generate 3D meshes, from 2D images. Wikipedia et al, [Wik22]

## 2.4 Neural network models

Zewen Li et al. [LYPL20] explain CNNs models that have been launched throughout history, in figure 5 the day of birth of the most relevant models can be observed.

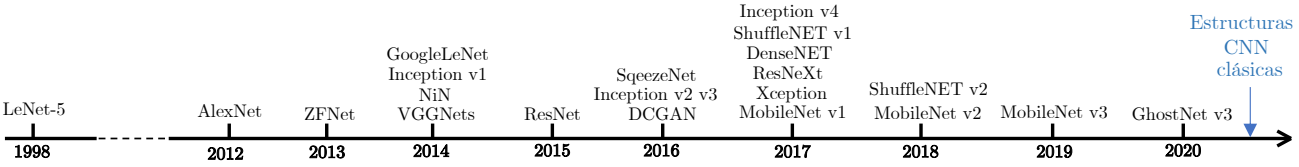



Figure 5: Chronology of the most relevant models. Figure created from an illustration by Zewen Li et al. [LYPL20]

It should be noted that the types of convolutional networks shown in the figure above are (5) have as their main objective, the classification task, and although this work does not focus on defect classification, but on defect detection, this type of networks are usually included as a *backbone* [13] within a more complex network architecture, as the one explained in section 3.1.

[12]In this context, the *pooling* layers slice each ROI from the output tensor of the network and reshape them to finally classify the region of interest.

[13]The term *backbone*, refers to the heart, or core, of a more complex network structure. In this work, its task is to create the feature map from the image it receives as input.

The most important aspects of some of the most relevant architectures shown in the figure are highlighted below. 5.

- **LeNet-5.** Zewen Li et al. [LYPL20] describes **LeNet-5** as an efficient convolutional neural network, trained with the back-propagation algorithm to recognise hand writing. This network has seven training layers, which contain two convolutional layers, two *pooling*[14] layers and three fully-connected layers.

- **AlexNet.** This network was designed by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. In their paper, Alex et al. [KSH12] the architecture of the network is described. This network contains eight weighted layers; the first five are convolutional and the remaining three are fully connected. The second convolutional layer, as well as the fourth and fifth are connected only to the previous layer's *kernels* maps residing on the same GPU. (see figure 6). The third convolutional layer *Kernels* are connected to all the second layer feature maps. Fully-connected neurons are connected to all previous-layer neurons. According to Alex et al. [KSH12], this network model incorporates the ideas of **LeNet** and applies the basic principles of CNNs to a deeper and wider network.

  In the same article, the good results obtained after training the network are highlighted, but Alex et al. [KSH12] also emphasizes the importance of the convolutional layers, because if any of the intermediate layers are removed, the network performance decreases, resulting in a 2% increase in losses. Nevertheless, this model is a good option. Furthermore, this architecture introduced new features, like the **ReLU**[15] function, aiming to mitigate the vanishing-gradient problem in deep neural networks. The use of the *dropout* technique is also highlighted, this technique randomly ignores some neurons during the training process, to avoid overfitting[16]

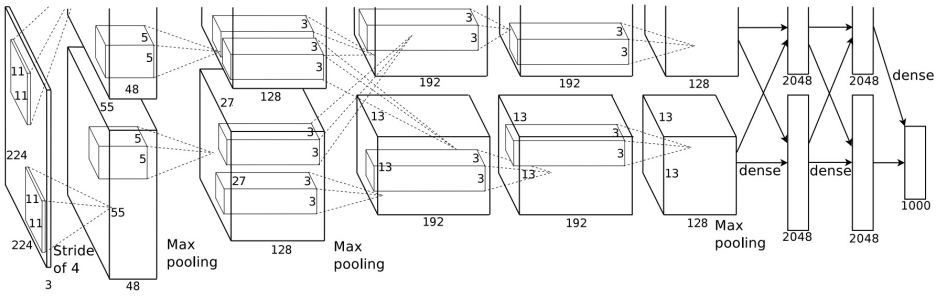



Figure 6: **AlexNet** architecture, Alex et al. [KSH12]

- **VGGNets.** The acronym **VGG** standw for, *Visual Geometry Group*, since it was this group, belonging to Oxford University, who developed this network, and who won a prize for it. Simonyan et al. [SZ15] describes this model as a series of convolutional neural network algorithms, including; VGG-11, VGG-11-LRN, VGG-13, VGG-16 and VGG-19.

[14] Pooling layers, are added after a convolutional stage. This layers reduce the dimensions of feature maps. Thus, the amount of learning parameters is reduced and consequently, the compute operations by the network. Kaggle et al. [Kag]. The purpose and operation of this layer is discussed in more detail below.

[15] Although it is not necessary to understand this activation function to understand this thesis, you can visit the appendix A.4, to learn more about this type of functions.

[16] *Overfitting* occurs when the model cannot generalize, extremely fitting to the training dataset. IBM et al. [IBM].

All of these variants are constituted by blocks, and they are all equipped with two *hidden layers*[17] fully-connected and a fully-connected output layer. In all of the model variants, the image goes through a set of convolutional layers, where Simonyan et al. [SZ15] implement filters of size $3 \times 3$[18].

To preserve the resolution at the output of the convolution, the *stride* of the convolution is set to 1 pixel. Finally, a spatial pooling is implemented, which is a way to compute the image representation from its encoded local features. In addition, this technique divides the image into different subregions and calculates the feature vector of each subregion. The final image representation is a concatenation of all the feature vectors of the subregions. Xinggang Wang et al. [Wan]. This technique is carried out with the implementation of five layers of type *max-pooling*. The *max-pooling* is performed with a window of $2 \times 2$ pixels, with an interval of 2.

For the explanation of this type of **VGG** network, the focus is on the **VGG-16**, however, the rest of the network types within the **VGG** family are constructed in a similar way. See graphically this architecture in figure (7):

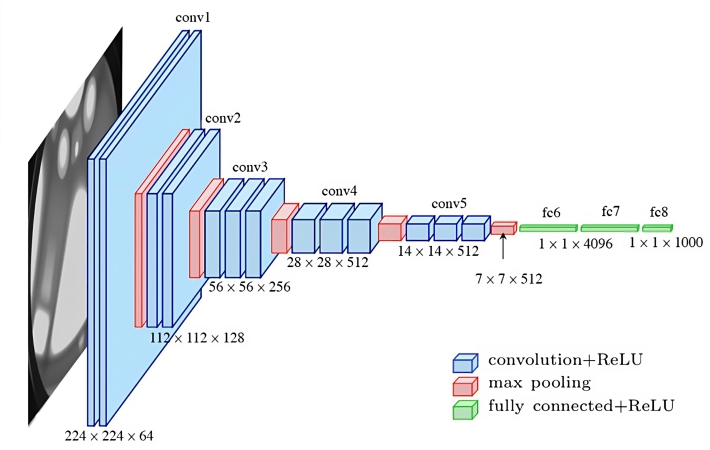



Figure 7: **VGG-16** architecture. Simonyan et al. [SZ15]

VGG-16 consists of 13 convolutional layers, 5 *max pooling layers* and 3 fully connected layers. Therefore, there are 16 layers with adjustable parameters, since there are 13 convolutional layers and 3 fully connected layers, hence the name VGG-16. In the first block there are 64 filters, and this number is doubled in the subsequent blocks to 512 filters. This model is completed with two fully connected hidden layers and an output layer, it should be noted that the two fully connected layers have the same number of neurons and each has 4096 neurons. Originally, the output layer has 1000 neurons corresponding to the number of categories in the dataset Imagenet [19].

The results obtained with this network are good, as VGG-16 was one of the best performing architectures in the 2014 competition, as its authors obtained a classification error of 7.32% (behind only GoogLeNet, with a classification error of 6.66%). It was also the winner in the

[17] In order to delve deeper about this property inside some architectures, one can take a look at the following appendix: A.1.

[18] Este es el tamaño más pequeño para captar la noción de izquierda/derecha, arriba/abajo.

[19] This is the database they used in the competition they won

localization area, with a 25.32% localization error.

Despite its advantages, this network has some drawbacks, such as the slowness in training, and the huge amount of parameters it needs, which can lead to problems with the gradient. Nevertheless, this architecture is still a very good option.

- **ResNet.** This names comes from *Residual Networks*, the **18** and **34** layers variants have been taken into account to analyze the network architecture. In [HZRS15] one can observe how the convolutional layers have mostly $3 \times 3$ filters and follow two simple design rules: (i) for the same output map size, the layers have the same number of filters; and (ii) if the feature map size is halved, the number of filters is doubled to preserve the time complexity per layer. Then, the sampling rate is reduced by convolutional layers having a step of 2. The network ends up with a *global average pooling layer* layer and a fully connected layer of 1000 neurons with *softmax*[20]. Finally, the total amount of weighted layers is 34.

  Furthermore, comparing this architecture with the previous one, Kaiming He et al. [HZRS15] highlights the difference in the number of filters and the lower complexity that exists in this architecture. The 34-layer base model (ResNet) has 3.6 billion FLOPs (multiplications), compared to 19.6 billion FLOPs in VGG networks, this translates to 18% less. The figure 8 illustrates the composition of ResNet, where it can also be seen how its architecture is much deeper than the architecture of the VGG-19.

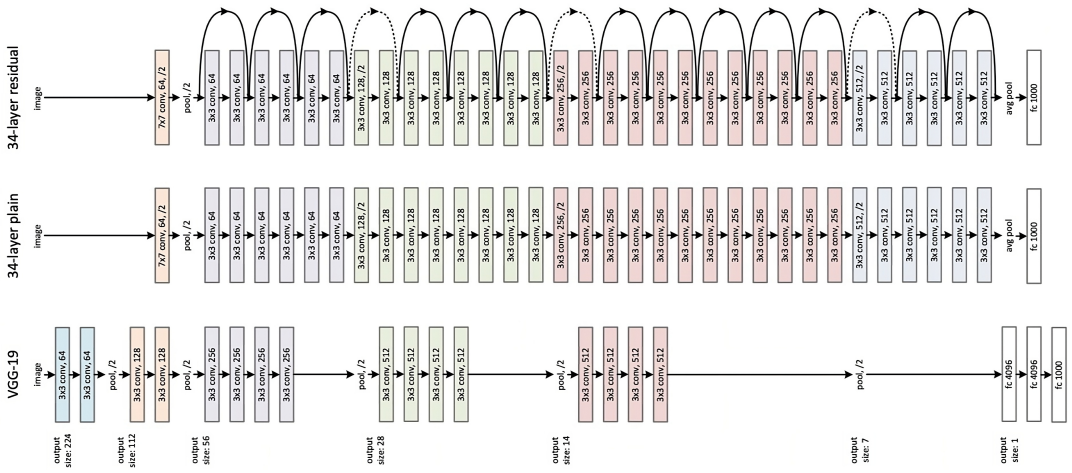



Figure 8: A comparison of **ResNet** architecture, and the VGG-19. Kaiming He et al. [HZRS15]

After this analysis of possible architectures, it is concluded that in order to correctly train a network that detects defects in images, one of the last two architectures mentioned above should be used, i.e., **VGG-16** or **ResNet**.

These two architectures are able to generate good results, and in spite of their differences, they become similar in many aspects, but an important aspect is that, ResNet solves one of the problems that arise in VGG, the gradient problem, although for the development of this thesis, it has not been necessary to understand this concept, in the annex A.4 it is deepened more about this problem. In the Resnet this problem is addressed in the first layer, where the number of rows and columns are reduced by a factor of 2, furthermore in the following *max pooling* operation another

[20]This layer is explained in the section 3.1.4.

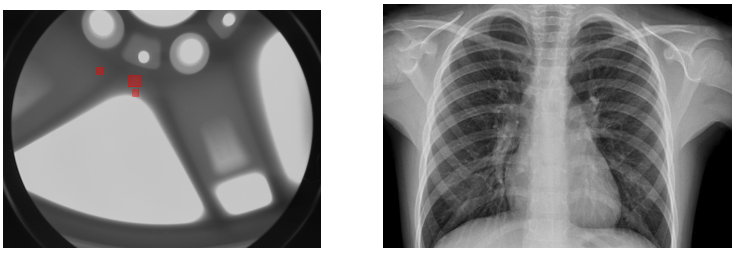
reduction with a factor of 2 is applied.

In order to correctly choose the type of architecture, results have been searched in different detection domains. Victor Ikechukwu et al. [VMDS21] obtains the results seen in Table 1, after implementing pre-trained models of a VGG-19 and a ResNet-50 in order to discriminate radiographs with pneumonia from "healthy" radiographs.

| Métricas | **VGG-19** | **ResNet-50** |
|---|---|---|
| Exactitud (%) | 97.3 | 96.2 |
| Especificidad (%) | 97.2 | 96.4 |
| Precisión (%) | 96.7 | 95.3 |

Table 1: Results

In table 1, it can be seen how the VGG-19 is more precise and accurate than the ResNet, although the difference is $\sim 1\%$. It is important to highlight, how the study performed in [VMDS21] is relevant to this topic, because the images implemented are in grayscale, as well as those used in this project, likewise, Victor Ikechukwu et al. [VMDS21], seeks to detect whether or not there is pneumonia, so there is only one class, as in the images used to train the network in this project. One unfavorable point, is that unlike the defects sought to detect in this work, the proportion of the defect in [VMDS21] is usually larger compared to the defects in this project, see the following comparison.



(a) X-ray of a wheel with 3 defects.

(b) X-ray of a lung with pneumonia

Figure 9: Comparison of proportions (defect/image)

It is important to mention, that the decision made between the two architecture options, **VGG** or **ResNet**, is based on the *stride* and the depth of the architecture, here is why.

Nowadays, one can find many pre-trained models on the Internet, or simply architectures, able to differentiate between dogs and cats, detect cars in images or know if a person is happy or sad. All these examples, usually keep an important similarity, the size of the feature, dogs and cats are usually easily recognizable, since they are considerably large within the image, which means that $\frac{\texttt{size(dog || cat)}}{\texttt{size(image)}} \in (0.4 - 1)$, the same applies to the rest of the examples (most of the time). There are other features that facilitate the task of classification, such as color, since `RGB` images are usually used, which allows to close the search area more quickly, since e.g., there are only cats and dogs of certain colors, the same happens with other objects or beings, such as plants. Another key feature is the silhouettes, as dogs, cats, cars and people have very characteristic shapes, which are easy to recognize.

Most of the models that one can find on the internet and in scientific papers perform well, when the sample set meets several of the above mentioned characteristics. But unfortunately, the images handled in this project, do not meet any of them, since;

- The defect/image ratio is small, i.e., $\frac{\texttt{size(defect)}}{\texttt{size(image)}} \in (0.01 - 0.15)$[21].
- The images are in black and white, so the defect is only noticeable due to a higher contrast or a different shade of gray or black than the rest of the image.
- The defects are usually irregular, although in many of the images the defects are round, in the rest of the images, they are amorphous, so they do not share silhouette.

These features can spoil the training of the network, so the choice of a good architecture is important. Knowing that the defects are very small, one cannot choose a very deep network, this is the same for the other two features above, since the deeper the network, the greater the *stride* is, and as this parameter increases, the feature map decreases, and if it decreases, the defects, small as they are, disappear completely.

For this set of reasons and despite the fact that in [HZRS15] very good results are obtained with the ResNet compared to the VGG, a **VGG-16** architecture has been chosen, as it is not as deep a network as the ResNet, but implements multiple improvements over its predecessors, such as faster training and higher accuracy.

The heart of the network architecture has already been chosen, but more functionality is still needed. The next section introduces the overall architecture of the work, within which will be the **VGG-16**.

[21] These figures are approximate, and have been calculated by dividing the average size of the rectangle covering a defect by the total size of the image.

# 3 Methodology

After having studied the options in terms of types of networks and architectures that can be formed, and having chosen the **VGG-16**, this section analyzes in detail the characteristics of the final architecture of the project.

In the field of computer vision, very good results in object detection have been achieved thanks to convolutional networks and region proposal algorithms. These algorithms, which are usually implemented as a network, have as their main objective to conjecture about possible regions. So the idea is to make use of this technique or similar, to refine the shot when deciding where there may or may not be a defect.

Continuing with the idea of the technique mentioned before, in [USGS13] the *Selective Search* is implemented, this "selective search" has the purpose of hypothesizing about the positioning of objects within an image, this technique combines the potential of two types of search, that of the exhaustive search and that of the segmentation search. This same article demonstrates the potential of this type of structures and the good results achieved. Like this article, there are many others, which justify the use of this type of implementations, in fact, recent advances in object detection have been possible thanks to techniques such as that of J.R.R. Uijlings et al. [USGS13]. The problem with this technique is that it applies 2,000 times the CNN to the proposed areas, and this introduces some computational wear. In the next paragraph we introduce the predecessor of this technique, which only passes the original image once, to a pre-trained model of convolutional networks.

The **RPN** (*Region Proposal Network*) is the predecessor of the above technique. This network is commonly used in more modern object detection networks that rely on region proposal algorithms to hypothesize about the location of objects. Shaoqing Ren et al. [RHGS15].

It is important to mention that these networks turn out to be a bottleneck, however, Shaoqing Ren et al. [RHGS15] have developed an alternative that solves the problem, which is explained in the following section.

## 3.1 Faster R-CNN

Due to its potential and the good results achieved by Shaoqing Ren et al. [RHGS15], their object detection system has been implemented in this thesis. In Figure 10 can be seen the general architecture of the *Faster R-CNN*.

As explained in [RHGS15], this network consists of two main modules. The first module is a fully convolutional network that proposes regions and the second module is the detector *Fast R-CNN*, in charge of using the proposed regions, therefore, as it is said in [RHGS15], the module `RPN` module tells the *Fast R-CNN* module where to search. In the following sections, the functions implemented by this architecture are studied in detail.

### 3.1.1 *Region Proposal Network*

Before explaining this network, the following two questions must be answered:

- **What does this network receive at the input?** This lattice is fitted with an image $\mathbf{A}$ of any size, e.g., $\mathbf{A} \in \mathbb{R}^{n \times m}$ in the case of being monochromatic, or $\mathbf{A} \in \mathbb{R}^{n \times m \times c}$ if it has more than one color. [22]
- **What's at the exit?** At its output, a set of proposals (rectangular "boxes") is issued, each with a score, calculated by the function `non_maximum_suppression`.

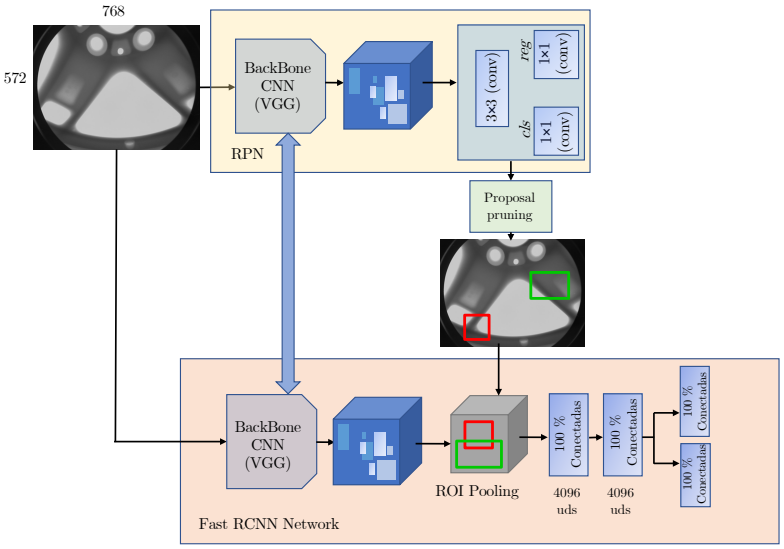



Figure 10: The combined architecture of the `RPN` and the `Fast R-CNN` gives rise to the **Faster R-CNN**. Figure created from an illustration in [Ana].

The **RPN** takes as input an image of any size, and outputs a set of objects in the form of rectangular "boxes" that have been proposed by it, each of these boxes has an associated probabilistic rating that indicates the certainty that this "box" covers a defect in the image. This dynamic is modeled with a convolutional network, which is described in this section.

In order to generate region proposals, a network is slid along the outgoing feature map of the last convolutional layer. This small network takes as input a $n \times n$ ($n = 3$) spatial window of the convolutional feature map, each window that is slid, is "mapped" to a lower dimensional feature, in the case of this project; 512-d[23]. Subsequently, this feature is introduced in two fully connected layers, the box-regresssion (*reg*) and the box-classification layer (*cls*). In summary, this architecture consists of a **VGG-16**, and $3 \times 3$ convolutional layers followed by the two layers mentioned above (*reg* and *cls*) that apply a convolution of $(1 \times 1)$. It should be noted that these last two layers apply a *padding* of 1.

[22] As mentioned at the beginning of this report, in the cases of having more than one color $\mathbf{A}$ represents a three-dimensional tensor.

[23] There will be 512 filters.

### 3.1.2 *Anchors*

The term of *anchor* is defined in [Mat], as *" a set of predefined bounding boxes with a given height and width. These boxes are defined to capture the scale and aspect ratio of the specific object classes to be detected. During detection, the predefined anchors are tiled across the image. Then, the network predicts the probability and other attributes, such as background, intersection over union (IoU) and offsets for each anchor in the mosaic."*

As explained in [RHGS15], the anchors are related to sliding windows, since at each sliding window position, multiple region proposals are simultaneously predicted, the maximum number of proposals for each location is denoted by the letter $k$. The *reg* layer has $k$ outputs which encode the coordinates of $k$ cells, and the layer *cls* outputs $2k$ scores that estimate the probability that there is or is not a defect for each proposal. The $k$ proposals are parameterized relative to $k$ reference boxes, which are named anchors. The anchors are centered in the sliding window and are associated with a scale and an aspect ratio. By default, 3 scales and 3 aspect ratios are used in this project, resulting in $k = 9$ anchors at each slider position.

The anchor concept can be better understood in figures 12 and 13.

Having explained the concept of proposed regions and the idea behind the *anchors*, the next section explains the dynamics of the **RPN**.

### 3.1.3 General dynamics within the RPN

Upon entering the network, the image of $572 \times 768$ pixels, enters the *backbone*, which as mentioned before, is the **VGG-16**. This incoming image, is rescaled to a size of $300 \times 400$ pixels. At the output of the **VGG-16**, there is the feature map of the image. The size of this feature map is determined by the value of the *stride*, and since this is the **VGG-16**, the *stride* will be **16**, therefore, if the rescaled image is $300 \times 400$ pixels, the feature map will have a resolution of $\lfloor \frac{300}{16} \rfloor \times \lfloor \frac{400}{16} \rfloor \equiv 18 \times 25$ pixels. See image 11.

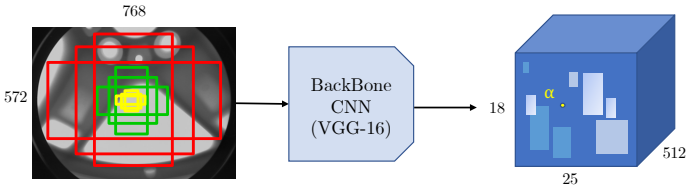



Figure 11: *Anchors* and output of the **VGG-16**. Figure created from an illustration in [Ana].

For each point in the output feature map, the network has to learn whether there is a defect present in the input image at its corresponding location, and estimate its size. To do this, the **VGG-16** places a series of *anchors* in the input image for each point of the output feature map. The *anchors* indicate possible objects of different sizes and aspect ratios [24] at that location (see figure 12).

[24]The *aspect ratio* represents the proportion between the height and width of the image.

Since for each sliding position there are $k = 9$ *anchors*, the total number of *anchors* will be given by the following expression: $18 \times 25 \times 9$, giving as a result, 4050 possible *anchors* (see figure 13). The three scales of *anchors* and aspect ratios, which have been contemplated, are as follows:

$$\texttt{anchor\_box\_scales} \in [[8,16,32],\ [16,32,64],\ [32,64,128]]$$
$$\texttt{anchor\_box\_ratios} = [[1,1],[\tfrac{1}{\sqrt{2}},\tfrac{2}{\sqrt{2}}],[\tfrac{2}{\sqrt{2}},\tfrac{1}{\sqrt{2}}]]$$

Now, as the network moves through each pixel of the output feature map, it has to check whether the corresponding k *anchors* spanning the input image actually contain defects, and thus refine the coordinates of these *anchors* to obtain *bounding boxes* as "proposed objects" or regions of interest. Shaoqing Ren et al. [RHGS15]

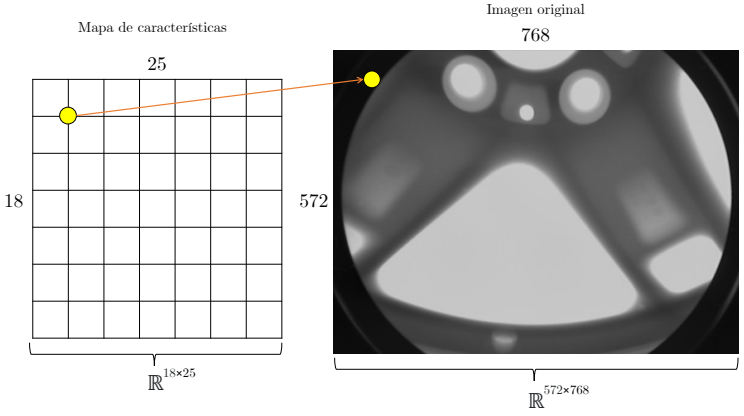



Figure 12: Projection of a feature map *anchor* onto the original image.

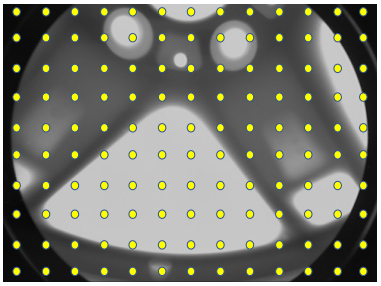


Figure 13: Original image with its corresponding*anchors*.

The initial state of each *anchor* is "negative", and in the code, for a *anchor* to become "positive", it has to fulfill that IoU $> 0.7$. In the case that $0.3 <$ IoU $< 0,7$, the *anchor* is not taken into account. The acronym IoU refers to the term, *Intersection Over Union*, and is calculated by dividing the intersection between *anchor* and *bounding box* of the *groundtruth* by the union of both. Hasty et al. [Has].

Assuming an image of the form $\mathbf{X} \in \mathbb{R}^{n \times m}$ and si $\exists \mathbf{A}, \mathbf{B} \mid \mathbf{A} \subseteq \mathbf{X}$ and $\mathbf{B} \subseteq \mathbf{X}$ then the IOU is defined by the Jaccard index.

$$\text{IoU} = \text{J}(\mathbf{A},\mathbf{B}) = \frac{|\mathbf{A} \cap \mathbf{B}|}{|\mathbf{A} \cup \mathbf{B}|} \tag{2}$$

In the case of this thesis, **A** refers to a *anchor* $k$, and **B** represents the corresponding *bounding box*. This operation allows us to estimate how much the *anchors* match the *bounding boxes* of the *groundtruth*. In addition, it should be noted that the **RPN** has many more "negative" regions than "positive" regions, so some of the negative regions are disabled and both the maximum number of positive regions and the maximum number of negative regions are limited to 256. Logically, the algorithm rewards positive *anchors*, as this condition proves a higher similarity between the *anchor* (prediction) and the *bounding box* (actual data). In the code, `y_is_box_valid` represents whether the *anchor* has a defect, and `y_rpn_overlap` represents whether the *anchor* overlaps with the *bounding box* of the *groundtruth*. The classification according to the IoU result, is as follows:

| Types | `y_is_box_valid` | `y_rpn_overlap` |
|---|---|---|
| "positives" | 1 | 1 |
| "neutrals" | 0 | 0 |
| "negatives" | 1 | 0 |

Table 2: Classification of *anchors*according to their IoU with the *bounding boxes*.

Returning to the architecture of the **RPN**, it is important to mention how at the output of the 18 units of the classification branch (*cls*), the size is $\mathbb{R}^{H \times W \times 18}$ [25]. Subsequently, this information to the output, is used to assign probabilities depending on whether or not each point in the feature map of the **VGG-16** (of size: $\mathbb{R}^{H \times W}$) contains an object within the 9 anchors of that point.

However, the 36 units of the regression branch (*reg*) give an output of size $\mathbb{R}^{H \times W \times 36}$. This output is used to obtain the 4 regression coefficients for each of the 9 *anchors* for each point on the **VGG-16** feature map (of size: $\mathbb{R}^{H \times W}$). These regression coefficients are used to polish the coordinates corresponding to the *anchors* with defects.

### 3.1.4 *Region of Interest*

Once the regions have been proposed and their corresponding *bounding boxes* have been created, the next task consists of grouping the features of the map generated after the VGG-16 from these *bounding boxes*. This task is carried out by the *ROI Pooling* layer (see figure 10), and it follows the following dynamics:

- This layer takes the region corresponding to a proposed feature map.
- Then, divide that region into a fixed number of sub-windows.
- Finally, it applies a *max pooling operation*, which consists of a pooling operation, which calculates the maximum value in each subwindow, in order to obtain an output of fixed size.

Basically, the layer splits the selected proposal window features coming from the **RPN** into subwindows of size $h/H \times w/W$ and performs a *pooling* operation on each of these subwindows.

[25] The letters $H$ and $W$ represent the height and width of the output respectively

This results in output features of fixed size ($H \times W$) regardless of the input size.

The parameters $H$ and $W$ are chosen so that the output is compatible with the first fully connected layer of the network. The chosen values of $H$ and $W$ in the *Fast R-CNN* is 7, since in [RHGS15] good results are obtained for this value. Like normal *pooling*, the *pooling ROI* is performed on each channel individually.

The output features of the *ROI Pooling* layer ($N \times 7 \times 7 \times 512$, where $N$ is the number of proposals) are fed into the *fully connected layers* and the *softmax* and *BB-regression* branches. The *softmax* classification branch implements the *softmax* function which is a normalized exponential function. This function takes as input a vector $\mathbf{z}$ of K real numbers, and normalizes it to a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. Wikipedia et al. [Wik23].

The standard *softmax* function $\sigma : \mathbb{R}^K \to (0,1)^K$ is defined $\forall K \geq 1$ as follows:

$$\sigma(\mathbf{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \forall i \in (1, ..., K) \text{ and } \mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K \tag{3}$$

In short, this function applies the exponential to each element $z$ of the vector $\mathbf{z}$ and normalizes it by the sum of all exponentials, producing probability values for each ROI belonging to K categories and to a general background category [26].

Subsequently, the regression branch (*BB-regression*), as the name suggests, applies regression, to make the *bounding boxes* of this region proposal algorithm more accurate.

## 3.2 Database

Once the project architecture has been explained, this section shows the images that have been used to train this network. As the aim is to correctly detect defects in radiographs of industrial parts, the images to be used correspond to images taken with X-rays of different types of castings.

These images are organized in $\mathbb{GDX}$ray, a public database containing everything from images of weldment parts to images of luggage taken at airports. All of these images and many more can be found at GDXRAY+.

### 3.2.1 $\mathbb{GDX}$ray repository

Mery et al. [MRZ+15] have collected all the images that can be found in $\mathbb{GDX}$ray, in order to test computer vision algorithms and to make available for the first time a public database of X-ray images. This database contains a total of 19407 images, but for this thesis only the images within the *Castings* directory have been used.[27]. The images contained in this directory show weldments that are mostly, but not all, defective.

In each of the Series, inside the directory *Castings*, there are the images corresponding to a type of industrial part, with a file *groundtruth*. This file contains the titles of the images inside the "Series" folder with the coordinates of the *bounding boxes* in charge of boxing the defects in the image (if any). If an image has no defects, then it will not be listed in the file. The format of the *groundtruth* for the `C0001` series can be seen in the box 4.

[26] The general background category in this context, makes reference to a space that does not contain any defect.
[27] Table 3 illustrates the structure of the repository.

| Base de datos | | Grupo | | Serie | | Imágenes de rayos-X |
|---|---|---|---|---|---|---|
| GDXray | → | *Castings* | → | C0001 | → | C0001_0001.png ... C0001_00072.png |
| | | | | ⋮ | | ⋮ |
| | | | | C0067 | | C0067_0001.png ... C0067_00083.png |

Table 3: Respository organization.

| ID imagen | x1 | y1 | x2 | y2 |
|---|---|---|---|---|
| 1.0000000e+00 | 1.7100000e+02 | 1.9800000e+02 | 4.7100000e+02 | 4.9300000e+02 |
| 2.0000000e+00 | 2.2600000e+02 | 2.5200000e+02 | 4.8700000e+02 | 5.1200000e+02 |
| 3.0000000e+00 | 2.7100000e+02 | 2.9700000e+02 | 4.9400000e+02 | 5.1900000e+02 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 7.2000000e+01 | 4.7100000e+02 | 4.8800000e+02 | 2.7500000e+02 | 2.9600000e+02 |

Table 4: *Groundtruth's* format of series C0001.

Some of the images that can be found in the directory are *Castings* son:

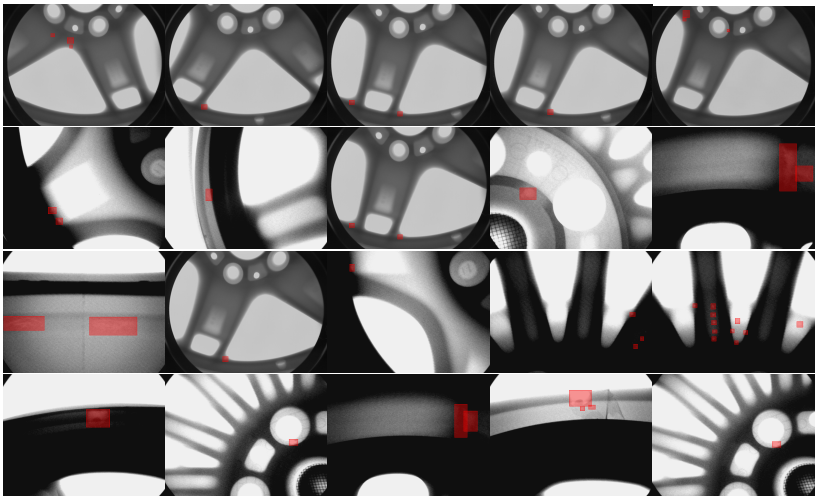


Figure 14: Some images of the *Castings* directory with its corresponding *bounding boxes.*

Looking at the images in figure 14, one can see how all images share the same dimensions ($\mathbb{R}^{572\times768}$), although not all images within the *Castings* directory have the same size. As mentioned in the 3.1.1 section, the network accepts any image dimension, since within the code all incoming images are rescaled to the same size.

# 4 Detector modeling

Recapitulating all the theory and technique that has been seen so far, the following can be inferred:

- Detecting objects is not the same as detecting small defects. The reason for this lies in the ratio of object or defect to background, and the choice of network architecture is based on this premise.
- A good training to undertake this task requires an adequate architecture, and not a very deep one, since a greater depth implies a greater number of parameters and descriptive capacity, and this increases the tendency to over-fitting. Therefore, by not needing so much descriptive capacity, the depth is reduced to avoid over-fitting.
- Detecting defects requires a main backbone, but also requires region proposal algorithms.
- The relationship between the input image and the feature map encoded by the backbone is important, as the detection mechanism depends entirely on this correspondence.

This section is not intended to explain the implemented code step by step, but rather to briefly explain the path of the image from the time it enters the network until it exits the region proposal algorithm, thus providing a better understanding of the structure of the code.

To implement the architecture studied in the section 3, we started from a code, which allowed us to better understand all the layers through which the image passes, as well as the training process, in addition to helping to strengthen the concepts related to the *stride*, the *padding* and the *anchors*. This code can be found in the following GitHub repository: `Faster R-CNN for Open Images Dataset by Keras`.

It is important to note that this code has been developed to train a neural network to detect people, cars, and cell phones. Although the author later creates more classes, such as "scissors", "Apple pencil", "sleeping monkey" and "cocoa". The author of this modified code (RockyXu66) was based on code from GitHub user Yhenon (see this link: `Yhenon's repository`).

The results obtained are good, with final probabilities between 75% and 99%, but as mentioned before, this type of architectures work well recognizing large objects. Throughout this section, it will be seen how problems arise when training the network.

## 4.1 Collecting data

In order for the network to pick up images, a data structure containing the directory where the image is located, and information related to it, such as its width (w), its height (h) and the coordinates of the *bounding boxes* that frame the defects that may be in it, must be passed to the network. To accomplish this task, a search algorithm has been implemented, which goes directory by directory memorizing all the images contained in it as well as its *groundtruth*. Although it may seem an easy task, it is not, because even if you know how to code in `Python`, you may not know many of the functions that help to implement an iterative search. Added to this is the difficulty that in the *groundtruth* does not appear the name of the images, but the number, so it has been necessary to extract from the titles of the images, the associated numbers, which are the same that appear in the *groundtruth*.

The algorithm I have implemented can be better understood with the following pseudocode;

**Algoritmo 1** Data collection

```
procedure ITERATIVE SEARCH
    dir = /User/Documents/GDXray/Castings
    for folders ∈ {dir} do
        if folders start with 'C' then
            image_folders.append(folders)
        end if
    end for
    for iter ∈ {image_folders} do
        new_dir = dir +' /' + image_folders.append(iter)
        Change to new_dir
        for imgs ∈ {new_dir} do
            if imgs end with '.png' then
                images.append(imgs)
            end if
        end for
    end for
    titulo,ID,x1,y1,x2,y2 = read_groundtruth(∀images)
    image_dict = {'/Users/.../C000X' ,{'w': .. ,'h': .. ,'boxes':[{'class': .. 'x1': .. ,'y1': ..
,'x2': .. ,'y2': ..}]}
    Return w and b.
end procedure
```

The next part consisted in drawing the *bounding boxes* in the original images. The code corresponding to this task can be seen in the project repository, whose link is in the section 8.

To draw the *bounding boxes*, a code has been implemented that goes through all the directories and their corresponding *groundtruths*, to read the coordinates of each defect corresponding to each of the images. Subsequently, all these coordinates are stored in a list that is then transformed into a tensor to be able to pass to the function in charge of painting the *bounding boxes* those coordinates, and the image where it has to paint them. With this implementation, it has been possible to represent the images seen previously in the figure 14.

Once the *bounding boxes* were represented in the original image, it is time to create the structure that houses the data corresponding to each of the images to be introduced into the network. To accomplish this task, a list of dictionaries has been created, where the name of the image, its height, width, and coordinates of the *bounding boxes* corresponding to the defects of the image, as well as the class of the defect, which is the same for all cases, are collected.

It should be noted that the dictionary that holds all this information for each of the images in the *Castings* directory is very large, because in total there are 2367 defects among all the images. This supposes a disadvantage, because when containing so many images, it is impossible to load all this information at once, because the computer does not have enough memory, the solution to this inconvenience, is given by the data generators.

The data generators allow iterating over a large sample set, in this case, it is iterated over the dictionary containing the path of each of the images and their corresponding information. As a usual function can return parameters after its execution, when one implements a data generator,

this one returns the values inside any **function** with a `yield`(x), after calling that **function**, the next line makes use of the `next`(x) to continue iterating, see the example below:

```
train_data_gen = get_anchor_gt(all_img_data, C, get_img_output_length, mode='train')
X, Y, image_data, debug_img, debug_num_pos = next(train_data_gen)
```

## 4.2 Preprocessing

Once the network is able to take the images and their corresponding information from the dictionary, it proceeds with the processing of the information.

The images that appear in the dictionary are ordered as they appear in the *Castings* directory, therefore, to avoid that the training of the network can be affected by the order of the images, a random seed is introduced so that the network picks up the images out of order.

When the network picks up an image, different transformations are applied to this image, such as rotations, but it can also be flipped up or down. The purpose of these transformations is to increase the sample set, because although the *Castings* directory contains 2727 images, these might not be enough to train the network, therefore, the modifications mentioned before are applied to the images, in order to "enrich" the sample set. As a consequence of these transformations, an increase in performance is observed in some models in [Tea].

However, when modifying an image, the coordinates of the *bounding boxes* of the *groundtruth* no longer coincide with the defects in the modified image, so in this section the new coordinates are also calculated and then dumped into the dictionary.

Once the image has been rotated or flipped, it is re-scaled to a size of $\mathbb{R}^{300\times400}$ in order to make model training faster and save time.

## 4.3 Region calculation

At this point, the network has already applied the necessary transformations and rescaled the image to adapt it to the training. From this new data, the image regions are calculated.

As already mentioned in the section 3.1.3, there will be $450 \times 9 = 4050$ *anchors* if the geometry of the feature map is $\mathbb{R}^{18\times25}$ and if each *anchor* has $3 \times 3 = 9$ corresponding *boxes* in the original image (see figure 11).

Initially, all the *anchors* are categorized as "negative" but depending on the result of the **IoU** between the *anchors* and the *bounding boxes* they may become "positive". This information is collected in the values that the function in charge of proposing regions returns, i.e., `calc.rpn`. (see table 2)

The values returned by the network are as follows:

- `y_rpn_cls = list(num_bboxes, y_is_box_valid + y_rpn_overlap)`
- `y_rpn_regr = 0 o 1 (0: the box is not valid and 1: the box is valid)`
- `y_rpn_overlap = 0 o 1 (0: the box has no defect and 1: the box has a defect)`

- `y_rpn_regr = list(num_bboxes, 4*y_rpn_overlap + y_rpn_regr)`
- `y_rpn_regr = x1,y1,x2,y2 (coordinates of the bounding boxes)`

The dimensions corresponding to the data structures shown in the list above are listed below:

- `y_rpn_cls` is a tensor of the form `y_rpn_cls` $\in \mathbb{R}^{1\times 18\times 25\times 18}$.[28] In orange the dimension of the map of features is highlighted, and in cyan the $9 \times 2$ gets represented, since each point in the map of features has 9 *anchors*, and each *anchor* has 2 values, one for `y_is_box_valid` and another one for `y_rpn_overlap` respectively.
- `y_rpn_regr` is also a tensor, but with the following dimensions `y_rpn_regr` $\in \mathbb{R}^{1\times 18\times 25\times 72}$. In this case, in cyan the product between all 9 *anchors* and the 4 values for `tx`, `ty`, `tw` and `th` that each *anchor* has, is represented, likewise, this has to be multiplied by the two possible values that can have `tx`, `ty`, `tw` and `th`, i.e., `y_is_box_valid` and `y_rpn_overlap`. This product yields 72 possible values ($9 \times 4 \times 2$).

Now, to check the final *anchor* that the network has taken, we represent in the figures 15 and 16, a set of figures where in all of them the same image is represented, but for different scales of *anchors* (see the second paragraph of the section 3.1.3).

As can be seen in figures 15 and 16 as the scale size of the *anchors* is increased, the space it covers is larger and this is not good, since it decreases the result of the **IoU**, since the defect is very small.

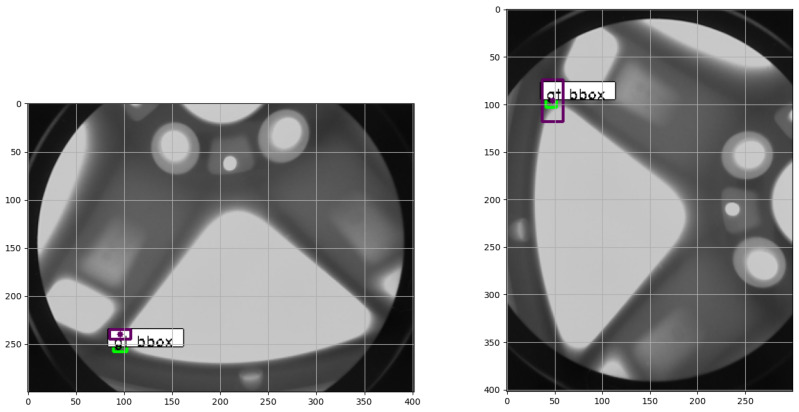
Moreover, if the size of the *anchor* with respect to the size of the *bounding box* is too small, then the result of the **IoU** will be low, and this is also not good for training, because even though the *anchor* covers a defect, it might be discarded due to its low score after performing the intersection over the joint. Some **IoU** measurements for the 15 and 16 images, can be seen in Table 5.

As can be seen in Table 5, as the scales increase, the maximum value of the intersection over the joint is reduced, but it is also worth mentioning that for larger scales, there are fewer **IoU** results that are 0.0, while for smaller scales, the number of **IoU** results that turn out to be zero are considerably more than those that are non-zero. This is logical, since for smaller scales of the *anchors*, the **IoU** will be larger than for larger scales, because most of the *anchor* would overlap with much of the *bounding box*.

| **Escalas** | **IoU**$\|_{min}$ | **IoU**$\|_{max}$ |
|---|---|---|
| $\{8, 16, 32\}$ | 0.0 | 0.1593 |
| $\{16, 32, 64\}$ | 0.0 | 0.1592 |
| $\{16, 32, 64\}$ | 0.0 | 0.1333 |
| $\{64, 128, 256\}$ | 0.0 | 0.03981 |

Table 5: Classification of *anchors* according to their IoU with the *bounding boxes*.

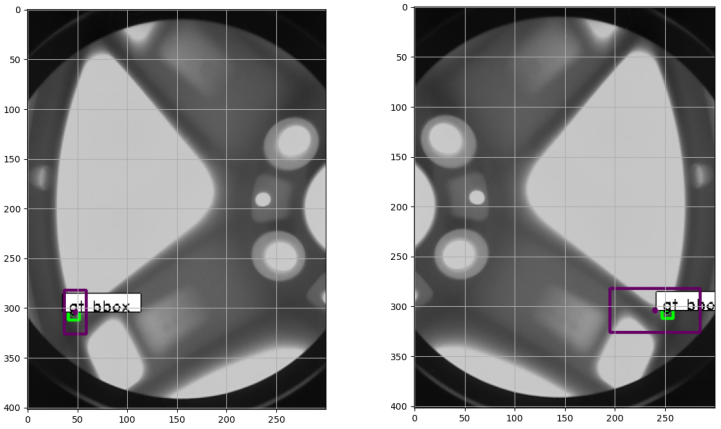
[28] The first dimension of the tensor `y_rpn_cls` $\in \mathbb{R}^{1\times 18\times 25\times 18}$ makes reference to the *bacth*. The *batch* is the number of images that are passed to the algorithm in each iteration of the training, as in this case it is 1, this means that in each iteration only one image is passed to the network.




(a) Resulting *anchor* of scale $\{8, 16, 32\}$

(b) Resulting *anchor* of scale $\{16, 32, 64\}$

Figure 15: Results obtained after the proposal of regions, the boxes in green represent the *bounding boxes* of the *groundtruth* and in purple the *anchors*.




(a) Resulting *anchor* of scale $\{32, 64, 128\}$

(b) Resulting *anchor* of scale $\{64, 128, 256\}$

Figure 16: Results obtained after the proposal of regions, the boxes in green represent the *bounding boxes* of the *groundtruth* and in purple the *anchors*.

It is important to note another fundamental aspect, which is the position of the center of the *anchor* with respect to the *bounding box* in the image. While the premise in the paragraph before the figure 15 is correct, it cannot be taken for granted that the center of the *anchor* calculated in the proposed regions will always fall in the center of the corresponding *bounding box*. Now it is necessary to understand why the centers of the *anchors* and the *bounding boxes* do not always coincide, for this, if one reviews the section 3.1.3 and observes again the images 12 and 13, one can notice how the position of the *anchors* is directly linked with the resolution of the image, therefore, the following mathematical probability relationship can be inferred.

If $\mathbf{X}$ represents any image.

$$\exists \mathbf{X} \in \mathbb{R}^{n \times m}, \exists \mathbf{X} \in \mathbb{R}^{\nu \times \mu} \mid n > \nu, m > \mu \tag{4}$$

And the centrioid [29] of any *bounding box* is $\bar{\beta}$ and if the associated centroid of any *anchor* is $\bar{\alpha}$, then:

$$Prob(\bar{\alpha} = \bar{\beta})|_{\forall \mathbf{X} \in \mathbb{R}^{n \times m}} > Prob(\bar{\alpha} = \bar{\beta})|_{\forall \mathbf{X} \in \mathbb{R}^{\nu \times \mu}} \tag{5}$$

With relation 4.3 we want to illustrate more intuitively the concept that for an image denoted by $\mathbf{X}$ of higher resolution (i.e., $n \times m$), the probability that $\bar{\alpha}$ and $\bar{\beta}$ are equal is greater, than for an image $\mathbf{X}$ of lower resolution (i.e., $\nu \times \mu$) simply because the amount of *anchors* in an image of $n \times m$ is greater, and therefore, the probability that the two *anchors* coincide is greater.

It should also be noted that the defect may fall directly below the position of an *anchor*, in which case the **IoU** will be greater than if the defect did not coincide with the *anchor*. In figure 15 for a scale of *anchors* of $\{8, 16, 32\}$, it can be seen how the generated *anchor*, does not completely match with the *groundtruth bounding box*.

### 4.4 From region proposals to ROIs

Once the **Fast R-CNN** architecture (see figure 10) has proposed the corresponding regions, and after generating the set of boxes that highlight these regions, the coordinates, which are in a `rpn` format, have to be transformed in coordinates that fit the modified image of size $300 \times 400$.

Although conceptually this layer has already been explained in the 3.1.4 section, the focus will now be on operations and the steps followed at this stage will be explained at a high level.

The function `rpn_to_roi` is in charge of transforming what the network has generated into regions of interest that can be represented as a *bounding box* in the image of $300 \times 400$. This function receives the following parameters as arguments:

- `rpn_layer` denotes the prediction made by the network model, of the regions calculated by the function Òcalccalc explained in the previous section. (4.3). The dimensions of this argument are the same as `y_rpn_cls`, which has already been discussed in the previous section (4.3). Therefore, the predictor model does not and should not alter the dimensions of the returned variable from `calc_rpn`.
- `regr_layer` denotes the output after applying a regression of the boxes highlighting the regions. As with `rpn_layer`, the dimensions of this argument are not altered by the predictor model, and these can be seen in section 4.3.
- `max_boxes` specifies the maximum number of *bounding boxes* to be drawn. However, this data will be used in the *bounding boxes* screening step at the end of this function.
- `overlap_thresh` determines the minimum overlap of a *bounding box* for it to be considered and subsequently represented. In this case, as the defects are very small, this limit will be small., $\approx (0.2 - 0.4)$.

[29] The centroid of an object is the geometric center of the subspace formed by the object, e.g., the centroid of a rectangle of dimension $b \times h$ will have its centroid at $\frac{b}{2}$ or $\frac{h}{2}$ depending on whether it is calculated with respect to the x-axis or y-axis, although in both cases if a straight line is drawn, it always passes through the center, so the reference is not a problem.

In order to transform these coordinates into representable elements of the image, one has to iterate through all the sizes of *anchors* (i.e., 9) and then, for each *anchor*, one has to consider each *aspect ratio*[30]. The purpose of this transformation is important, since it is intended to obtain a list of *bounding boxes*, where each *bounding box* is a vector of 4 coordinates. However, we do not have a list, but 2 matrices of several dimensions, so that the position of each element, its coordinate in each dimension determines the *anchor* to which it refers through the information of its scale, *aspect ratio* and the position of its center.

Each *anchor* is rescaled minding its size, the corresponding *aspect ratios* and the stride with value 16[31]. Subsequently, `regr_layer` dimensions, goes from $\mathbb{R}^{1\times18\times25\times18} \rightarrow \mathbb{R}^{4\times18\times25}$, staying only with the feature map and maintaining the 4 values of each *anchor* (i.e., `tx`, `ty`, `tw` and `th`). Once this is done, a mesh of the size of the feature map is created, and from it the position and size of each of the *anchors* (i.e. `x`, `y`, `w` and `h`). Finally, to refine the values of position and size, a regression is applied.

In addition, the code prevents any *anchor* from exceeding the height and width. Next, we proceed with the translation of coordinates, from `(x,y,w,h)` to `(x1,y1,x2,y2)`[32]. Finally, and following the dynamics at the beginning of this paragraph, the drawing of *bounding boxes* outside the feature map is avoided.

Once we have the *bounding boxes* in a list, we apply the function `non_max_suppression_fast` which is in charge of sifting that list of *bounding boxes* in order to keep those boxes that meet the parameters of `max_boxes` and `overlap_thresh`, this sifting is implemented in the following way;

1. Ordering the list of probabilities.[33]
2. Search for the largest probability in the list (the last one) and save it in the selection list.
3. Calculation of the **IoU** with the last box and the rest of the boxes in the list. If the **IoU** is greater than the overlapping threshold (i.e., `overlap_thresh`), then this box is removed from the list.
4. Repeat steps 2 and 3, so that there are no items in the list of probabilities.

At the output of this function, the final list of *bounding boxes* is obtained, with the list of probabilities associated to each of the *bounding boxes*. From this data, one can now represent the transformed image, with the *bounding boxes* on top, this is what is done in section 6.

### 4.5 Last layer: *ROI-Pooling*

The calculation of regions and their translation into representable data is done, now it remains to prepare the network for training. But before, it remains to describe the last layer of the model, the **ROIPooling layer** that has already been introduced in the section 2.3, but that is explained in this section.

As written by Jifeng Dai et al. [DQX+17] the **ROIPooling layer** is a layer that operates in a two-dimensional space, and is used in most object detection-based region proposal methods.

[30] Seen *aspect ratios* in section 3.1.3 (i.e., `anchor_box_ratios` = $[[1,1],[\frac{1}{\sqrt{2}},\frac{2}{\sqrt{2}}],[\frac{2}{\sqrt{2}},\frac{1}{\sqrt{2}}]]$).

[31] We recall that in this project, the *stride* is 16. However, the fact that the name **VGG-16** has the number 16 is a coincidence, since the 16 in the name of the *backbone* determines the number of layers.

[32] The coordinates are transformed to the same coordinate format that the *groundtruth* follows.

[33] Remember that each *bounding box* has a probability associated with it.

This layer performs the work of converting a rectangular input region of arbitrary size into features of fixed size. In other words, this is the function that processes the *rois* to obtain an output of specific size by means of a *max pooling* layer. Each input *roi* is divided into several subcells, to which a *max pooling* layer is applied, where the number of subcells must be of the dimension of the output shape. Yinghan Xu et al. [Xu]

This process can be better understood with the following figure.

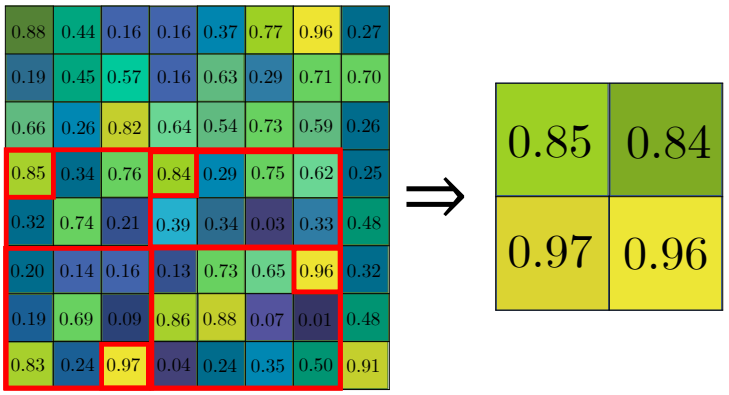



Figure 17: Example of the action carried out by the **ROIConv** layer. Figure created from an illustration in [Xu].

After this layer, there is a classification stage, which constitutes the end of the *Faster R-CNN* architecture, and is responsible for predicting the class name for each input *anchor* and the regression of its corresponding box.

As already mentioned in the section 2.4, in this project there are no defect classes, but there are two kinds of detections which are defect and background. This function predicts the name of a class[34] and applies another regression that enhances the box coordinates.

That said, the arguments received by this function are as follows:

- `base_layers` this parameter denotes the neural network model, i.e., the architectural *backbone*, which, as is known from the 2.4, it is about the **VGG-16**.
- `input_rois` the dimensions of this argument are, `(1,num_rois,4)` and contains the computed *rois* by the previous function.
- `num_rois` represents the maximum number of *rois* that the network is able to process in each iteration. In this project, this parameter is set to 4, in order not to drastically increase the computational load when training the network and to avoid slowing down the process even more.

At its output, the network will issue a list with two variables:

[34]This refers to defect or background.

- `out_class`. This parameter denotes the output of the classification layer.
- `out_regr`. This other parameter denotes the output of the regression layer.

As seen in section 3.1.4, the classification was carried out by the branch implementing the *softmax* function, while the other branch proceeded with the regression calculation of the final boxes.

The network has already done its job, the next step is to train it to detect those small defects in images, such as those seen in figure 14, although this is already discussed in section 5.

# 5 Training

The objective of this section is not to show the results of the different training sessions, since this is already done in the section on 6, but rather to explain the parameters involved in this process and then to explain the metrics that make it possible to represent the results graphically and analytically.

But before describing any parameter, it is necessary to be clear about the idea behind the training of a neural network. As written in [Xer], the training of a neural network consists of adjusting each of the weights[35] of the inputs of all the neurons that are part of the neural network, so that the responses of the output layer match as closely as possible the data already known[36].

Now, in order to understand what happens in training, one must know the parameters that make it possible, as well as the variables that influence the outcome of the training.

## 5.1 Parameters

As has been observed throughout this report, delegating the detection task of this thesis to an artificial intelligence requires the understanding of many concepts and tools. The training stage of a neural network is no exception, since it involves a multitude of metrics and concepts, however, in this section we intend to mention and explain exclusively those concepts and parameters that influence the training process to a greater extent.

The most relevant parameters that influence the training process are the following:

- The **depth**. Logically, the training time of a deep network will be longer than that of a network with few convolutional layers.
- The ***batch***, is a hyperparameter that defines the number of samples to work with before updating the internal parameters of the model. In other words, and as already mentioned in one of the annotations of the section 4.3, the *batch* in practical terms, is the number of images that are passed to the algorithm in each iteration of the training, as in this case is 1, it means that in each iteration only one image is passed to the network.

  It is important to mention that if one wants to reduce the training time, it is reasonable to pass to the network one image after another and not several at the same time.
- The number of ***rois***, is the number of regions of interest that the network will take into account simultaneously, as already mentioned, this parameter has been set to 4. A larger number of *rois* implies a longer training.
- The **size** of the sample set (i.e., the images used for training the network). As is logical, if the sample set is large ($\sim$ 10000 images) then the training will take quite some time, and if the number of *rois* and the *batch* is also increased, then an uninterrupted training could last almost a week.
- The ***epoch***, according to Jason Brownlee in [Bro], is a hyperparameter that defines the number of times the model to be trained will work through the entire training data set.

[35] Information about the definition of the weights and other parameters of neural networks can be found in the appendix A.1.

[36] In the case of this project, the data known are the coordinates of the defects contained in the *groundtruth* for each of the training images.

A *epoch* means that each sample in the training data set has had the opportunity to update the internal parameters of the model. In addition, this parameter is composed of one or more *batches*.

- The **resolution** of the incoming images. In the case that these were of high resolution, their processing along the convolutional layers would slow down the training stage considerably, since, a higher resolution implies more pixels, and more pixels imply more *anchors* which translate into more regions of interest to calculate, although this could improve the training at the level of results, since a higher resolution image would help to preserve to a greater extent the already small defect.

- The **maximum** number of ***boxes***. As explained in section 4.4 this parameter determines the maximum number of final "boxes" that will be represented, therefore if this parameter is increased, more boxes will be generated, which will take more time, since their coordinates and the associated probability have to be calculated.

- The ***overlap threshold*** establishes the overlap threshold, which the "boxes" must meet in order to be taken into consideration.

### 5.2 Metrics

Knowing the parameters that characterize and influence the training process, we then study the metrics that allow us to judge whether the network has been trained correctly. Although it is not the only metric, the loss function of the model is of considerable importance, since it determines whether the network has been reasonably trained. However, it is not possible to conclude from it whether the training result is correct, so this role is shared with the parameter **mAP**, which is explained later.

In order to train the RPN model, each *anchor* is assigned a binary class label, which determines whether it is a defect or not. Now, a *anchor* can be assigned a positive label if either of the two conditions are met:

- The *anchor*/*anchors* with the greatest **IoU** overlaps with a *bounding box* of the *groundtruth*.

- *Anchors* with an **IoU** over 0.7 overlaps with a *bounding box* of the *groundtruth*.

As mentioned in [RHGS15], a single *bounding box* can assign positive labels to many *anchors*. Usually, the second condition is sufficient to determine the positive samples, although the first condition is still contemplated, just in case the second condition does not find any positive samples. A non-positive *anchor* is assigned a negative label if its **IoU** is less than 0.3 for all the *bounding boxes* of the *groundtruth*. However, if there are *anchors* that are neither positive nor negative, they do not contribute to the training of the network and are discarded.

That said, the objective function that is minimized in the **Fast R-CNN** of this project for an image, is defined in [RHGS15] como sigue:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_{i \in \mathcal{B}} L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{cls}} \sum_{i \in \mathcal{B}} p_i^* L_{reg}(t_i, t_i^*) \tag{6}$$

Where $i$ denotes an *anchor* inside a *batch* $\mathcal{B}$ (i.e., an *anchor* within an image) and $p_i$ is the probability of an *anchor* being an object (i.e., `y_is_box_valid`). Then, $p_i^* = 1$ if the *anchor* turns to

be positive and 0 if the *anchor* is negative.

The parameter $t_i$ denotes a vector that contains all 4 coordinates of the network-generated *bounding box*, while $t_i^*$ contains the same type of coordinates, but for the *bounding box* of the *groundtruth*, associated to a positive *anchor*.

As known from the 3.1.4 and 4.5 sections the **Faster R-CNN** returns a list containing the output of the classification layer and the output of the regression layer, i.e., **out_class** and **out_regr** respectively. Hence the parameters $L_{cls}$ and $L_{reg}$.

- The $L_{cls}$ parameter denotes the losses on a logarithmic scale resulting from the comparison "is there a defect or not".
- While $L_{reg}$ represents the losses associated with the regression.

Finally, the term $p_i^* L_{reg}$ activates regression losses only when $p_i^* = 1$ (i.e., just for positive *anchors*) because $p_i^* \in \{0, 1\}$. Both *cls* and *reg* terms, mentioned in section 3.1.1 are represented by $p_i$ and $t_i$ respectively. These two parameters, corresponding to the output of the regression and classification layers, are normalized by the terms $N_{cls} = 2$[37] and $N_{reg} = 4096$[38] respectively, then these are weighted by the parameter $\lambda = 10$.

In this project, two loss functions have been applied, one to the RPN model and the other to the classifier model. But in addition, as the region proposal layer issued two values, the one associated with the classification of the detection[39], and the one corresponding to the regression of the coordinates of the *bounding boxes*. For this reason, there are two types of loss functions, the one corresponding to the output of the classification stage and the one associated with the regression stage. This makes a total of four loss functions which are briefly explained below:

- `loss_rpn_cls`. This function corresponds to the losses of the classification result of the RPN model.
- `loss_rpn_regr`. Similar to the previous one, this also corresponds to an output of the RPN, but this time, it is associated with the regression of the coordinates of the *bounding boxes*.
- `loss_class_cls`. This function and the following one correspond to the output of the classifier model, this one in particular refers to the losses associated with the output of the classifier stage.
- `loss_class_regr`. Unlike the previous one, this one corresponds to the output of the regression stage of the *bounding boxes*.
- `curr_loss`. This last loss function sums the values of the four previous functions, and normally has a decreasing trend.

In addition to the loss function[40], explained before, there are other metrics implemented in this project, which also allow to analyze the result of the training. These are:

[37] This parameter indicates the number of classes, in the case of this project $N_{cls} = 2$, since we have, defect and background.

[38] It is recalled that 4096 was the number of *anchors* in the image.

[39] Whether it is a defect or not.

[40] In the appendix A.3, we go into more detail about why we seek to minimize this function and describe closely related concepts and tools.

- `class_acc`. Determines between 0-1 the accuracy of whether or not a box contains a defect.
- El `mAP`. This parameter is described by Jonathan Hui in [Huia], as a very popular and commonly used metric to measure the accuracy of object detectors such as the **Faster R-CNN**. This parameter represents the average AP (*Average Precision*). In the context of this thesis, precision measures the accuracy of the predictions, i.e., the percentage of correct predictions. Precision can be calculated with the following formula:

$$Precision = \frac{TP}{TP + FP} \quad \text{where} \begin{cases} (TP) & : True\ Positive \\ (FP) & : False\ Positive \end{cases} \tag{7}$$

Next, the term *recall* is introduced, which measures how well all positives are found. This parameter can be calculated as follows:

$$Recall = \frac{TP}{TP + FN} \quad \text{where} \begin{cases} (TP) & : True\ Positive \\ (FN) & : False\ Negative \end{cases} \tag{8}$$

It is important to mention the close relationship between the **mAP** and the **IoU**, because as explained in the 3.1.3 section, for each *bounding box*, the overlap between the *bounding box* predicted by the network, and that of the *groundtruth*, is measured by the **IoU**. Therefore, for the defect detection task of this project, the accuracy is calculated using the value of the **IoU** given an overlap threshold. Shivy Yohanandan et al. [Yoh].

The above explanation can be better understood with the following example extracted from [Yoh]. If we take as an example an overlap threshold (i.e., threshold of the **IoU**) of 0.5 and an **IoU** of 0.7 for a specific detection, then the prediction is classified as *True Positive*(TP). Now, if the **IoU** is 0.3, then the prediction is classified as *False Positive*(FP)

By setting an overlap threshold, only those detections that pass this threshold remain. If it is decided to increase this threshold, the false positives (FP) decrease, thus improving the accuracy, but worsening the *recall*, as detections may be lost. In the case of decreasing the threshold, false positives increase and consequently, accuracy decreases.

Once the most influential metrics and parameters have been explained, the following section presents the training results, where the values obtained with the metrics explained in this section are also justified.

# 6 Results and corresponding analysis

After explaining the metrics in the previous section (5), this part of the report presents the results of the four trainings that have been carried out, as well as their corresponding analyses. After exposing these results, the output of `rpn_to_roi` for different images is shown, in order to observe the proposals of regions that the network has carried out, and thus justify the results of the trainings.

## 6.1 Training results

This section presents the results of the four training sessions that can be found in the figures below 18, 19, 20 and 21.

In the first training, the maximum number of *bounding boxes* was set to 50, and the *aspect ratio* of the *anchors* was $(128, 256, 512)$. In the second training, a maximum of 400 *bounding boxes* and an *aspect ratio* for the *anchors* of (16,32,64) were set. Subsequently, for the third and fourth training, the *aspect ratios* were (16,32,64) and (32,64,128) respectively, while the maximum number of *bounding boxes* remained at 400.

As can be seen in the first two trainings, and especially in the first one, the different loss functions have many oscillations and there seems to be no behavior that allows them to converge. This is logical, since in the first training, the weights within the network are being regulated for the first time. Therefore, in the second training, a less oscillatory trend is already observed. (see figure 19), although it is not until the fourth training that the loss functions converge at the end of the process. (see figure 21). The size of the *aspect ratios* also plays a role, since in the first training they are much larger than in the rest, and this may have negatively affected the training of the network, as discussed in the section 4.3.

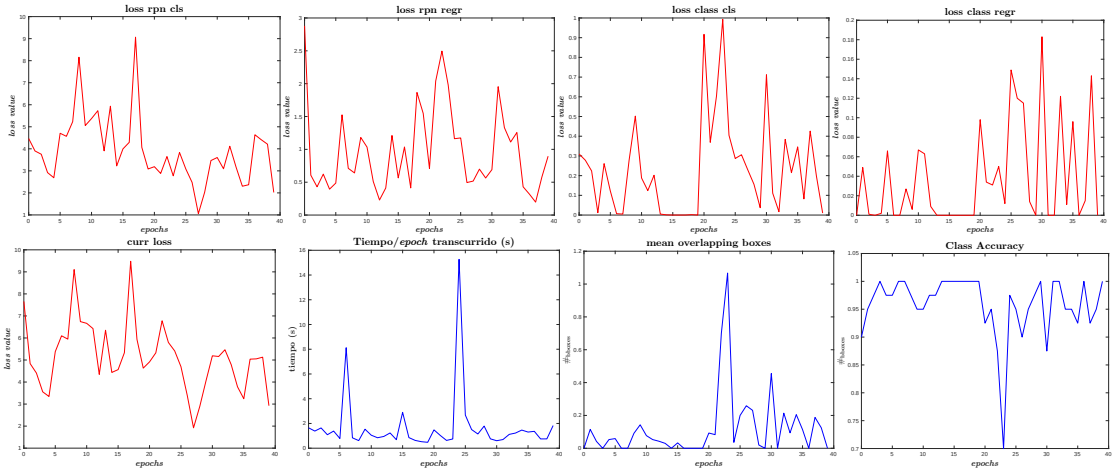



Figure 18: First training results.

In order to differentiate between the types of metrics, the results corresponding to the different loss functions are shown in red, while the rest of the metrics that can be read in the title of each of the graphs are shown in blue. In addition, it is worth noting that the order of the training results shown in the figures 18, 19, 20 and 21, is relevant, since in the first training, the relative values of the loss functions are higher with respect to the third and fourth, which means that subsequent trainings will achieve better results, unless the parameters specified to the network are radically different.

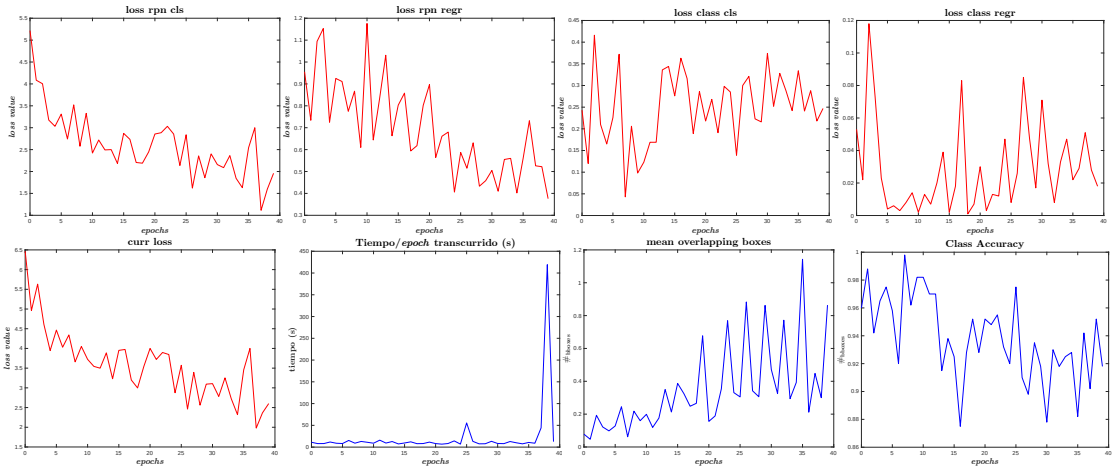



Figure 19: Second training results.

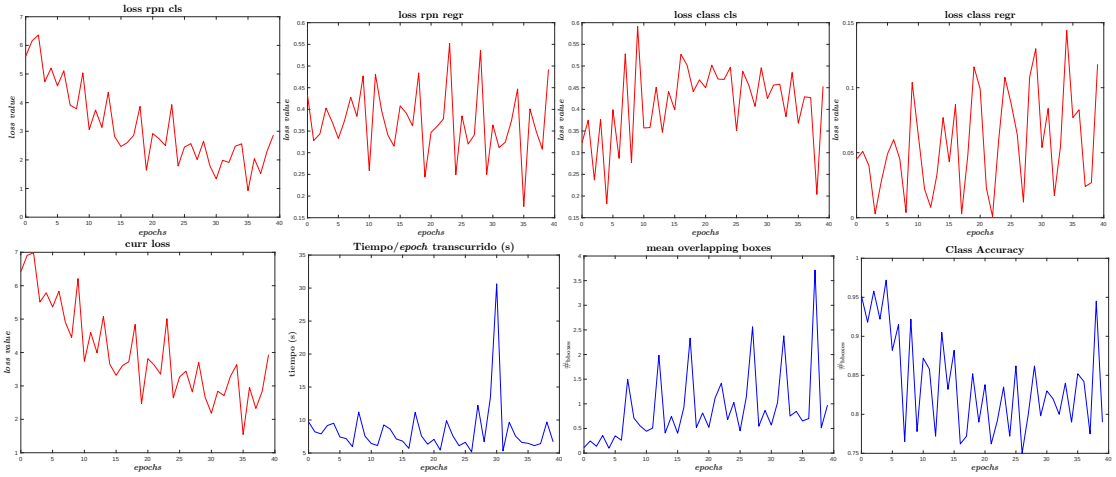



Figure 20: Third training results.

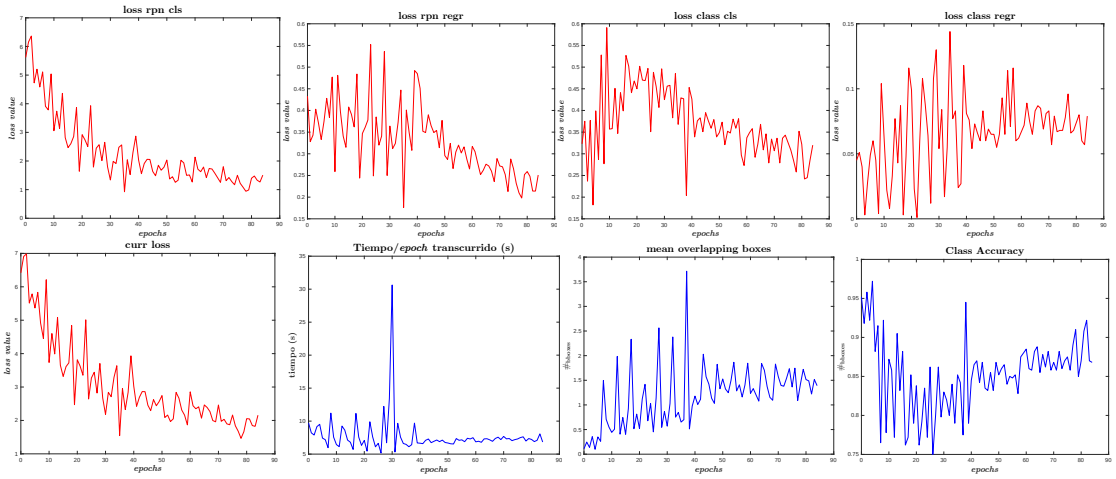



Figure 21: Fourth training results.

In the fourth training (see figure 21) it is observed how the losses associated with the RPN model have a decreasing behavior throughout the whole process, although in the losses associated with

the regression stage of the *bounding boxes*, a slight growth is observed. In addition, for the losses associated with the detection classification, a rapid fall of the slope is observed, which would imply a more rapid learning[41]. Likewise, a converging trend is observed, while the regression stage losses continue to decrease, reaching a lower value. This could mean that the defect accuracy is already high at the initial stage of training, but at the same time, the accuracy of the coordinates of the *bounding boxes* is still low and would need more learning time.

In the two loss functions applied to the classifier model, an increasing trend is observed at the beginning of the training, and in the case of the losses associated with the regression of coordinates of the *bounding boxes*, this increasing trend dominates the whole process.[42].

Below is a table with the average of the results of each of the loss functions for the four training sessions:

| # | mOB | cls_acc | loss_rpn_cls | loss_rpn_rgr | loss_c_cls | loss_c_rgr | curr_loss | mAP |
|---|---|---|---|---|---|---|---|---|
| 1° | 0.1206 | 0.9594 | 3.9131 | 0.9726 | 0.2331 | 0.0371 | 5.1559 | 0 |
| 2° | 0.3508 | 0.9388 | 2.6225 | 0.6937 | 0.2485 | 0.0286 | 3.5933 | 0 |
| 3° | 0.8966 | 0.8409 | 3.1588 | 0.3705 | 0.4135 | 0.0576 | 4.0006 | 0 |
| 4° | 1.1767 | 0.8537 | 2.3079 | 0.3290 | 0.3710 | 0.0660 | 3.0739 | 0 |

Table 6: Average values of the metrics obtained after the four training sessions.

The averaging times per *epoch*[43] were as follows for each of the four training sessions:

| # | times(s) |
|---|---|
| 1° | 1.6836 |
| 2° | 22.2965 |
| 3° | 8.3643 |
| 4° | 7.7076 |

Table 7: Average of times for each epoch of each training session

From these calculated averages, and from the results in general, the following can be inferred:

- The optimization algorithm is correctly implemented, since throughout the training process for the 4 cases, the losses decrease gradually (see figures 18, 19, 20 and 21), this can best be seen in table 6. This means, that the network is following the normal training process, but this does not mean that it is training properly. In fact, it can be seen how the parameter **mAP** remains constant and null for all training (see section 5.2).
- As concluded in the previous point, the parameter **mAP**, indicates that the "average precision" is 0, looking back to the equation 5.2, it can be seen how **mAP** = *recall*, and for the *recall* to be zero, this implied that $TP = 0$. This in practical terms, translates to very few positive regions arriving on the network, or simply, these are very few ($\sim 1\%(zonas_{totales})$)

[41] Further on, the argument is made as to why the learning that has taken place is not correct.
[42] The explanation of this paragraph and the previous one is based on an argument contained in [Xu].
[43] In the four training sessions, the *epoch* is 40 (see explanation in section 5.1).

while almost 99% of the zones are negative. This means that due to the established threshold, most of the detections have been classified as *False Positives*

Considering the final conclusion of the above list, the question that may arise is, how is it possible that the network is learning and that the optimization algorithm works correctly, if the network is missing negative zones. Well, even if the optimization algorithm manages to reduce the loss function, it does so by finding incorrect minima, since the optimization function itself is not correct. This does not mean that the equation 5.2 is invalid, but that the parameters related to the proposal of regions within the equation 5.2 are not entirely true.

If one delves deeper into this problem, one can see how, as the network goes through the training stage, it is given a series of rewards each time it detects a negative zone. These awards translate into an increase of weights[44], and in a non-penalization by the loss function so the network stops learning. Moreover, since all that the network detects are negative areas, since almost everything that reaches the classification stage are negative areas, either because of a low **IoU** (which is what happens in this thesis) or because it does not overlap with any *bounding box* of the *groundtruth*, the weights are increased considerably, and this is not good.

However, even if what was discussed in the previous paragraph is true, what has happened in the four training sessions is that since there is an imbalance of classes, i.e., there are many more proposed regions without defect than with defect, the network collapses as a consequence of not being able to learn due to the lack of losses that guide the training. Therefore, the network learns a little to detect "negative" regions but this learning does not go very far due to the above.

If we look at the results presented in section 6.2, we can see that the problem with these results lies in the proposal of regions, since the network is being rewarded for detecting areas classified as negative, and therefore the loss function is decreasing, but the number of positive regions is zero, since **mAP** remains constant with a value of 0.

## 6.2 Proposal of regions

As the network is not able to learn correctly, the focus has to be put on the proposal of regions, therefore, before the classification layer in the code, the regions proposed by the network have been represented, in order to know if the calculations are coherent.

The images in this section are the result of numerous simulations with different variations of the maximum number of *boxes* and the overlap threshold. See figure below:

The images in figures 22 and 23 correspond to a set of proposed regions made by the network, for different input images. To facilitate the comparison task, the original image with its *bounding boxes* corresponding to the defects indicated in the *groundtruth* has been represented on the left of each pair of images, and the modified image with the regions proposed by the network is represented on the left. Thus, one can observe whether the boxed regions encompass defects or not.

It is worth noting the degrading effect that all the convolution layers have on the resolution of the image, as it passes through the different layers of the architecture. Although it cannot be seen, in several images the defect disappears completely (with respect to the original image), but this is not due to the degradation of the image resolution, but because in the feature map this defect

[44] To learn more about the weights and their influence within a neural network, see the sections A.1 and A.2 in the appendices.

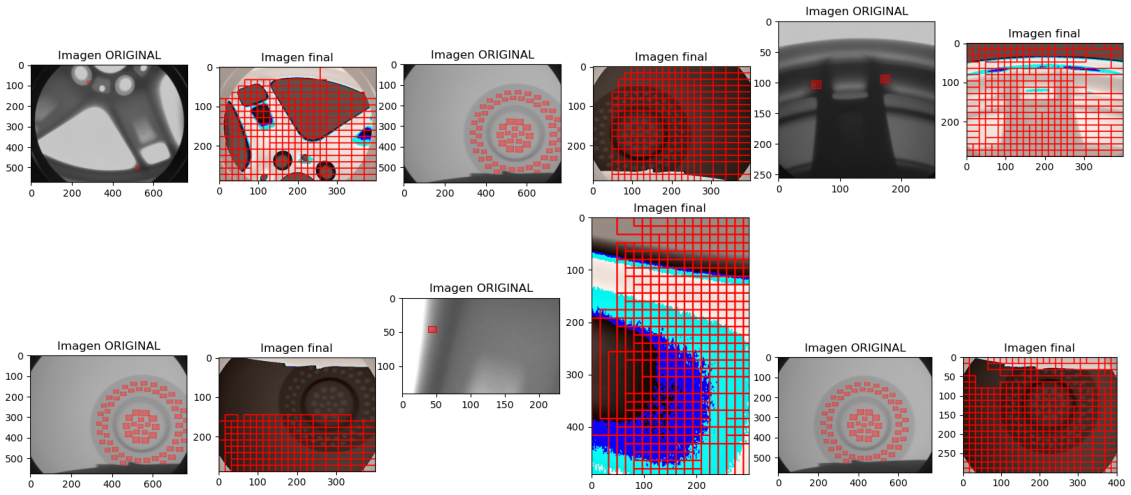



Figure 22: Some proposals of regions for some images

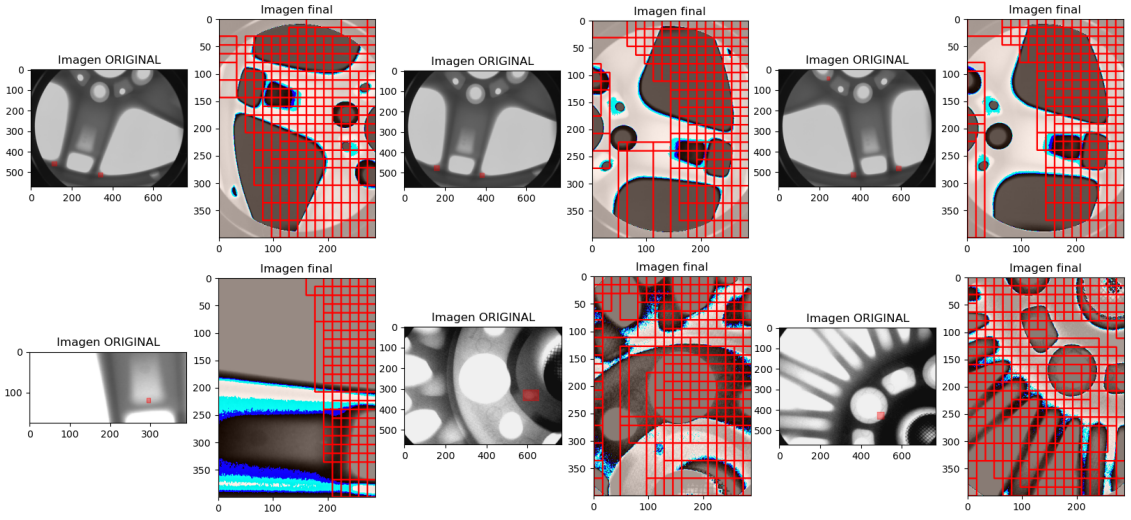



Figure 23: Some proposals of regions for some images

disappears after passing through several layers of the *backbone*, which causes an erroneous proposal of regions.

The results in figure 22 and 23 have been achieved for a threshold of **IoU** of 0.4 and a maximum number of representable *boxes* of 50 for some images and 400 for other images. This big difference between 50 and 400, is due to the desire to observe in which position within the image the boxes are placed, because if only 50 *bounding boxes* are drawn, these will correspond to the *bounding boxes* that have better qualification and therefore, they will be the most accurate *bounding boxes*. In this way, by representing few boxes it is easier to see which are the "best" regions.

In many cases it is observed how the defect is not surrounded, but some surrounding areas, to avoid this we have tried with different overlapping thresholds (usually lower, to filter as little as possible) and a drawing of more boxes, but we have not seen a considerable improvement, this may indicate that the problem comes from before.

It was to be expected that the network would not perfectly propose the regions within the images, since as mentioned in the section 3 and in [RHGS15], the **Faster R-CNN** is an architecture intended for object detection in color images. And as mentioned in the 2.4 section, we do not want to detect objects, but defects. In addition, the proportion of defects does not help either, as they are much smaller compared to those of an ordinary object. To these two restrictions, we must add that the images are in black and white, which prevents the network to learn from some visual feature to better differentiate the defects, since the difference between the defect and the background of the image is the shade of black and white.

In short, the network is trying to propose regions from black and white, low resolution images with very small defects and proportions. This means that when extracting the feature map from the image, the defect is lost in the *backbone* due to the drawbacks exposed in the previous paragraph and the number of convolutional layers of the *backbone*. Although the choice of the *backbone* is not perfect, having avoided a *Resnet* architecture was a good idea, as this has a larger number of layers, which would further reduce the chance of any defect reaching the classification stage.

Although everything points to the fact that the feature map is not including many of the defects in the images, it can be seen as in Figure 22 more specifically in the second pair of images in the top row and in the last pair of images in the bottom row, when the image contains many defects and the background is monotonous, the network usually retains many of those defects, as they are subsequently boxed as proposed regions. Therefore, in the rest of the images it is seen how even if there are no defects in some areas of the images, the network proposes regions due to changes in contrast or color due to the characteristics of each of the castings.

Having said that, it would be wrong and unfair to blame only the *backbone* for discarding the defects and not to include them in the feature map, since there is currently no limitation in terms of modifying the operation of the backbone and that is why improvements could be introduced in the backbone itself.[45].

In the following section (7.2) some techniques and improvements in the implemented architecture are mentioned for future work.

[45]These improvements are introduced in the 7.2 section.

# 7 Conclusions and future work

As seen in the 6 section, the network has not been able to model the classification stage correctly, due to a proposal of regions that is not accurate enough.

This section aims to conclude the reasoning of the results obtained from a constructive perspective, in order to introduce improvements in the detection process to train a robust network in the future.

## 7.1 Conclusiones

Although it has not been possible to train the model of the **Faster R-CNN** for defect detection, after the different analyses and studies that have been carried out in the sections on 6 and 4, it has been possible to understand in detail, the triggers of a bad learning and the characteristics of the architecture that hinder the training process. The analysis has also been performed with the objective of optimizing the performance of the **Faster R-CNN** to adapt its region proposal stage for a defect detection task.

Although the more theoretical knowledge about neural networks is not mandatory to develop this project, it has served to cement many of the explanations and assumptions carried out, as well as to clarify concepts related to the internal dynamics of a neural network and the effect of the most essential parameters within it.

It has been shown that even with a shallow network architecture, it is not possible to detect defects of small proportions within an image. However, it is not only the network that is to blame, because as we will see later in section 7.2, there are complementary improvements that the programmer can implement to mitigate these difficulties. It is important to know that the network learns what it is asked to learn, therefore, the difficulty of a project like this, lies in knowing how to give that information to the network, and how to lead the network to the realization of a task of unbalanced[46] nature without collapsing the training.

As has been corroborated in section 4.3, the parameters that most influence the region proposal are the scales of the *anchors*, the image sizes, and the image resolution, since all these parameters determine the image feature map and its subsequent training. In turn, this affects the training of the region proposal and the classification, and as we have seen in the section 6, a good training of the region proposal starts from a good feature map.

In addition to what was mentioned in the previous paragraph, in order to undertake a good training, the sample set must be large and varied, there must be no determinism in the order of the images that enter the network, the sample set must have important annotations and characteristics that the network can take into account and the size of the *batch* must be considerable[47]. It should also be noted that longer training usually means better results, as long as the proposed regions are correct.

Finally, it has been seen how the implementation of an object detector model performs the basic tasks for detecting small defects, but it fails to give accurate results, because more detection accuracy[48] is needed and techniques to reuse knowledge acquired by the network during training

[46] This problem was explained in the penultimate paragraph of the section 6.1.
[47] By making the *batch* larger, the training time will be longer.
[48] Such as that offered by the Mask R-CNN architecture.

to perform easier tasks, such as *transfer learning*, and then train the network to perform more complex detection tasks.

Ultimately, this project highlights the need for additional techniques when detecting defects with object detection techniques, and highlights the major differences between detecting objects and detecting small defects.

## 7.2 Future work

The analysis carried out has allowed the identification of strategies that could be used to improve the modeling of the defect detector, these strategies are explained below:

- Higher resolution images. To improve the results of any training, higher resolution images can be used, because as explained in the section 4.3, the higher the resolution, the greater the overlap between the *bounding box* of the *groundtruth* and that of the *anchor*.

- *Transfer learning*. This is a technique, which focuses on taking more control in training. This technique involves reusing a pre-trained model on a new problem. It is currently very popular in deep learning because it can train deep neural networks with relatively little data. Niklas Donges [Don].

  This technique would apply to this graduate work as follows. First, the network would be trained to detect much larger defects in black and white images[49], as defects in image 9 seen in section 2.4. In this way, the network would get used to the black and white images and after a long training, it would avoid highlighting other details of the images, e.g., ribs, clavicles, and other bones that may be contained.

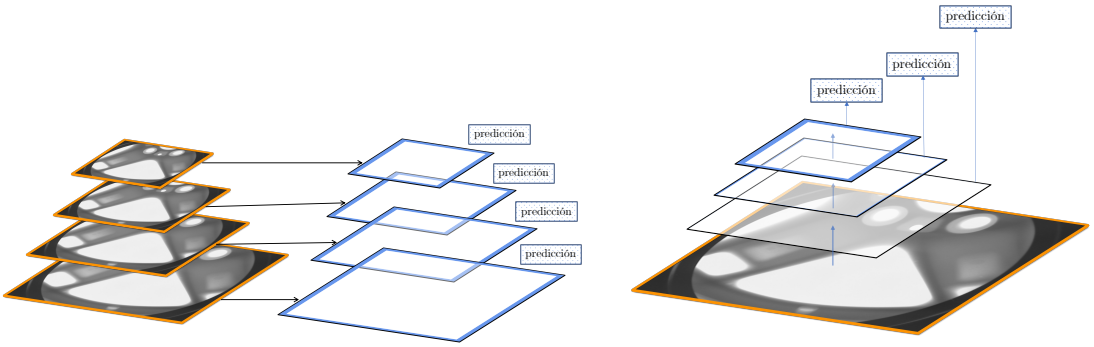
  Thus, the weights of the first layers of the network, would be kept for further training, since the first layers of the network are focused on extracting textures, contours, silhouettes and general variations of the image background. Subsequently, the network would already be trained with the images from the $\mathbb{GDX}$ray database, taking advantage of the network's prior knowledge of the general characteristics of black and white images.

- **Mask R-CNN** implementation. This architecture has already been mentioned in section 2.3, and its main advantage is its higher analysis accuracy, since it is capable of analyzing an image with an accuracy of fractions of pixels, which is of special interest due to the small size of the defects. However, the price to pay is a higher complexity, since although it is very similar to the **Faster R-CNN**, it has one more stage for segmentation, so it was thought appropriate to run first the basic stages prior to segmentation, which are those that have the **Faster R-CNN**. However, once this final degree work was done, this architecture could be approached with more security and therefore this option is considered as the main substitute for the **Faster R-CNN**.

- La **FPN**. As seen in this thesis, detecting small defects in black and white images is a complicated task. To overcome most of the complications involved in this type of detection, one can create a pyramidal structure containing the image to be studied at different scales, from top to bottom. But, this involves a large amount of memory and time, a consequence of processing images at different scales. To reduce the amount of memory, instead of working with multi-scale images, one can work with their associated feature maps instead.

[49] Or even images of other types.

Jonathan Hui, in [Huib], describes this network as a feature extractor designed for the pyramid concept introduced earlier. The goal of this model is to generate multiple layers of feature maps (multiscale feature maps) with better quality information than the normal feature pyramid for object detection (left image in Fig. 24). This model can be better understood with the following figure (24).




(a) Image processing at different scales.

(b) Processing with image feature maps, at different scales.

Figure 24: Dynamics of the **FPN**. Figure created from an illustration in [Huib].

Nevertheless, Jonathan Hui, in [Huib], clarifies that the **FPN** is not an object detector itself, but a feature extractor that works with object detectors, such as the one implemented in this project, the **RPN**. This is an advantage, since the **VGG-16** would extract the feature maps, which would then be fed into the **FPN** to generate a pyramid with them. Finally, these pyramid-like maps are introduced into the **RPN** in order to propose regions from them, following the same dynamics already explained in section 3.1.3.

It should be noted that the *transfer learning* technique can be complemented with the **Mask R-CNN** architecture as well as with the implementation of the **FPN** in the **Faster R-CNN**.

# 8 Codes

In this section you can find the link to the `GitHub` repository, where you can find all the implemented codes, including the one developed in `Matlab`, to represent the training results.

In addition, the same repository includes most of the images produced throughout this dissertation, the Matlab code implemented to represent the results of the four training sessions and the resulting files of the four training sessions in `.csv` format.

The link to the repository is as follows, `https://github.com/jcsombria/tfg-psr`.

# 9 Bibliography

## References


[AN] Coursera Andrew Ng. Deep learning specialization. https://www.coursera.org/specializations/deep-learning[Online; último acceso 18-junio-2023].

[Ana] Shilpa Ananth. Faster r-cnn for object detection. https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46[Online; último acceso 21-junio-2023].

[bae] baeldung. What is the purpose of a feature map in a convolutional neural network. https://www.baeldung.com/cs/cnn-feature-map#:~:text=In%20CNNs%2C%20a%20feature%20map,to%20produce%20multiple%20feature%20maps.[Online; último acceso 22-junio-2023].

[BCN+13] Francesco Bonanno, Giacomo Capizzi, Christian Napoli, Giorgio Graditi, and Giuseppe Marco Tina. A radial basis function neural network based approach for the electrical characteristics estimation of a photovoltaic module. *CoRR*, abs/1308.2375, 2013.

[Boo] BootCampAI. Intro a las redes neuronales convolucionales. https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8[Online; último acceso 18-junio-2023].

[Bro] Jason Brownlee. Difference between a batch and an epoch in a neural network. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/[Online; último acceso 22-junio-2023].

[Cou] Coursera. Binary classification deep leanirg course coursera. https://acrobat.adobe.com/id/urn:aaid:sc:EU:61eb58e2-9982-49a3-8cd1-dbf0e9ac06a1[Online; último acceso 18-junio-2023].

[D2l] D2l. Deep dive into deep learning. https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html[Online; último acceso 20-abril-2023].

[Dee] DeepChecks. Learning rate in machine learning. https://deepchecks.com/glossary/learning-rate-in-machine-learning/#:~:text=The%20learning%20rate%2C%20denoted%20by,network%20concerning%20the%20loss%20gradient%3E. [Online; último acceso 20-abril-2023].

[Don] Niklas Donges. What is transfer learning? exploring the popular deep learning approach. https://builtin.com/data-science/transfer-learning[Online; último acceso 25-junio-2023].

[DQX+17] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR*, abs/1703.06211, 2017.

[DSC21] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. A comprehensive survey and performance analysis of activation functions in deep learning. *CoRR*, abs/2109.14545, 2021.

[Has] Hasty. Intersection over union (iou). https://hasty.ai/docs/mp-wiki/metrics/iou-intersection-overunion#:~:text=To%20define%20the%20term%2C%20in,matches%20the%20ground%20truth%20data. [Online; último acceso 10-junio-2023].

[Huia] Jonathan Hui. map (mean average precision) for object detection. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173[Online; último acceso 22-junio-2023].

[Huib] Jonathan Hui. Understanding feature pyramid networks for object detection (fpn). https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c[Online; último acceso 25-junio-2023].

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[IBM] IBM. What is overfitting? https://www.ibm.com/topics/overfitting[Online; último acceso 2-junio-2023].

[JQPZ22] Shengli Jiang, Shiyi Qin, Joshua L. Pulsipher, and Victor M. Zavala. Convolutional neural networks: Basic concepts and applications in manufacturing, 2022.

[Kag] Kaggle. Pooling layer and it's types explained. https://www.kaggle.com/general/175896[Online; último acceso 29-mayo-2023].

[Kno] KnowledgeHut. 9 types of neural networks: Applications, pros, and cons. https://www.knowledgehut.com/blog/data-science/types-of-neural-networks[Online; último acceso 10-mayo-2023].

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[LYPL20] Zewen Li, Wenjie Yang, Shouheng Peng, and Fan Liu. A survey of convolutional neural networks: Analysis, applications, and prospects. *CoRR*, abs/2004.02806, 2020.

[Mar] Daniel Jurafsky James H. Martin. Chapter 5: Logistic regression. https://web.stanford.edu/~jurafsky/slp3/5.pdf[Online; último acceso 6-mayo-2023].

[Mat] MathWorks. Anchor boxes for object detection. https://es.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html[Online; último acceso 18-junio-2023].

[MRZ+15] Domingo Mery, Vladimir Riffo, Uwe Zscherpel, German Mondragón, Iván Lillo, Irene Zuccar, Hans Lobel, and Miguel Carrasco. Gdxray: The database of x-ray images for nondestructive testing. *Journal of Nondestructive Evaluation*, 34(4):42, 2015.

[Pel12] C. Pellegrini. The history of x-ray free-electron lasers. *The European Physical Journal H*, 37(5):659–708, 2012.

[Pet] StackExchange Peter. How to format table with images?. https://tex.stackexchange.com/questions/431200/how-to-format-table-with-images[Online; último acceso 25-junio-2023].

[Pro] ProjectPro. 5 different types of neural networks. https://www.projectpro.io/article/5-different-types-of-neural-networks/431[Online; último acceso 10-mayo-2023].

[RHGS15] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[Staa] StackExchange. Drawing a convolution with tikz. https://tex.stackexchange.com/questions/437007/drawing-a-convolution-with-tikz[Online; último acceso 16-mayo-2023].

[Stab] StackExchange. Drawing a sigmoid function and its derivative with tikz. https://tex.stackexchange.com/questions/497949/drawing-a-sigmoid-function-and-its-derivative-in-tikz[Online; último acceso 15-mayo-2023].

[Stac] StackExchange. Illustrating gradient descent with tikz. https://tex.stackexchange.com/questions/561921/replicating-a-plot-using-tikz[Online; último acceso 20-abril-2023].

[Stad] StackExchange. Plot gradient descent. https://tex.stackexchange.com/questions/544796/plot-gradient-descent [Online; último acceso 20-abril-2023].

[Stu] David Stutz. Illustrating (convolutional) neural networks in latex with tikz. https://davidstutz.de/illustrating-convolutional-neural-networks-in-latex-with-tikz/[Online; último acceso 2-abril-2023].

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[Tea] Great Learning Team. Understanding data augmentation, what is data augmentation how it works? https://www.mygreatlearning.com/blog/understanding-data-augmentation/#:~:text=Data%20augmentation%20can%20be%20used,imbalance%20problem%20in%20classification%20tasks.[Online; último acceso 15-junio-2023].

[USGS13] Jasper Uijlings, K. Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013.

[VMDS21] A. Victor Ikechukwu, S. Murali, R. Deepu, and R.C. Shivamurthy. Resnet-50 vs vgg-19 vs training from scratch: A comparative analysis of the segmentation and classification of pneumonia from chest x-ray images. *Global Transitions Proceedings*, 2(2):375–381, 2021. International Conference on Computing System and its Applications (ICCSA- 2021).

[Wan] Xinggang Wang. What is spatial pooling in computer vision? https://www.quora.com/What-is-spatial-pooling-in-computer-vision[Online; último acceso 25-mayo-2023].

[Wik22] Wikipedia contributors. Region based convolutional neural networks — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Region_Based_Convolutional_Neural_Networks&oldid=1101681994, 2022. [Online; último acceso 20-junio-2023].

[Wik23] Wikipedia. Softmax function, Apr 2023. https://es.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html[Online; accessed 14-June-2023].

[Xer] Xeridia. Redes neuronales artificiales: Qué son y cómo se entrenan. https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i#:~:text=Entrenar%20una%20red%20neuronal%20consiste,a%20los%20datos%20que%20conocemos.[Online; último acceso 22-junio-2023].

[Xu] Yinghan Xu. Faster r-cnn (object detection). https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-op último acceso 21-junio-2023].

[Yoh] Shivy Yohanandan. map (mean average precision) might confuse you! https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2[Online; último acceso 25-junio-2023].

# Appendices

## A Theoretical basis

Although the objective of this project is not the theoretical study of neural networks, it is considered pertinent to add an appendix such as this one, which briefly explains the most important theoretical concepts that have helped in the development of this thesis. Therefore, this section presents the theoretical foundation of the work as well as some advances in classification tasks.

### A.1 Introduction to neural networks

Before dealing with more advanced topics, it is pertinent to introduce the theory that justifies the operation of the most elementary relative to this thesis, a simple neural network, up to the operation of something as elaborate as the architecture implemented in this work.

A neural network can be described as a computational system capable of performing a task for which it has been trained, from an input data and a set of variables with initial values that change value depending on how well or poorly they perform the task. The first mathematical model of a neural network was introduced by Walter Harry Pitts, a self-taught American mathematician and logician who worked in the field of computational neuroscience. Later, together with Warren Sturgis McCulloch, an American neurologist and cyberneticist, they would carry out different tests that allowed advances in the fields of biophysics and mathematics.

After numerous advances on his part and on the part of other scientific personalities, the concept of the **neuron** as the elementary unit of a network was established and explained. An **artificial neuron** can be understood by the following figure:

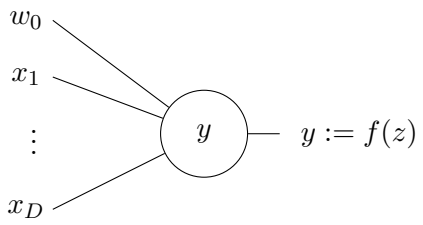



Figure 25: A single processing unit and its components. The activation function is denoted by $f$ and is applied to the input $z$ of the "neuron" to form its output $y = f(z)$. The set of parameters $x_1, \ldots, x_D$ form a vector, i.e., $x = (x_1, \ldots, x_D)$ where $x \in \mathbb{R}^D$. These values represent the input of other units within the network. The parameter $w_0$ is called bias and represents an external input to the unit. All inputs are mapped to the input $z$ using the propagation rule. David et al. [Stu]

Further on, in figure 26 a perceptron is shown. A perceptron is a linear system of the following form: $y_i = \mathbf{w}^\top \mathbf{x} + \mathbf{b}$ where $\mathbf{x}_0 = 0$ to include $w_0$, usually denoted by $\mathbf{b}$. The $\mathbf{w}$ parameter is called **weight**, and represents the intensity of interaction between each neuron.

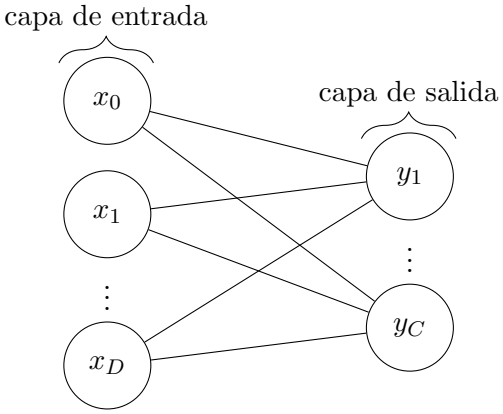



Figure 26: The perceptron consists of $D$ input units and $C$ output units. All units are labeled according to their output: $y_i = f(z_i)$ in the case of output units, and $x_i$ in the case of input units. The input values $x_i$ are propagated to each output unit by the weighted sum propagation rule: $y_i = \mathbf{w}^\top \mathbf{x} + \mathbf{b}$. The additional input value $x_0 := 1$ is used to include the biases as weights. David et al. [Stu]

Important highlights of the figures 25 and 26:

- **Activation function.** It is the function that is applied to the input of each of the units. The purpose of an activation function is to add nonlinearity to the neural network. This nonlinearity is very important, as it ensures that the output is not a "multiple of the input" and therefore that the output does not have a linear relationship to the input. The less linear the activation function is, the more complex the decisions made by the network will be and therefore, the more interesting and suitable results to work with will be obtained.

- **Inputs** ($\mathbf{x}$). Neural networks can serve a wide variety of purposes. Some of the most popular applications of neural networks are computer vision, speech recognition and natural language processing. Therefore, depending on the application, the format of the input data will vary. In this case, since the intention is to detect defects in images, the inputs will represent the image to be analyzed. [50].

- **Propagation rule.**. This term includes two very important terms: *"forward propagation"* and *"backwards propagation"*

  → *"Forward propagation"*. Is the way to move from the input layer (left) to the output layer (right) in the neural network.

  → *"Backward propagation"* or *"back propagation"*. It is the process of shifting from right to left, i.e., from the output layer to the input layer. The best way to correct or adjust the weights ($w$) in order to achieve minimization of the loss function[51] is through backward propagation.

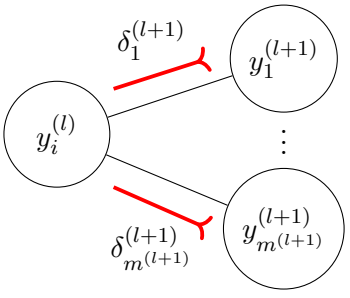



Figure 27: Once the errors $\delta_i^{(L+1)}$ have been evaluated in all output units, they are propagated backwards. David et al. [Stu]

[50] It is important to highlight the pre-processing that each of the images undergoes before being uploaded to the network (see section 3)

[51] This is discussed further below, see equation (9)

Once the essential parameters that determine the functioning of a set of neurons are known, the next step is the study of a complete network, with different *"hidden layers"* and *"hidden units"*. Below is the figure 28 illustrating an arbitrary neural network, where the nomenclature used is as follows;

$$y_{\#}^{(\nu)} \quad \text{donde} \begin{cases} (\nu) & : \text{determines the number of the layer in which the neuron } \# \text{ is located.} \\ \# & : \text{determines the number of the layer } (\nu) \text{ in which the neuron is located .} \end{cases}$$

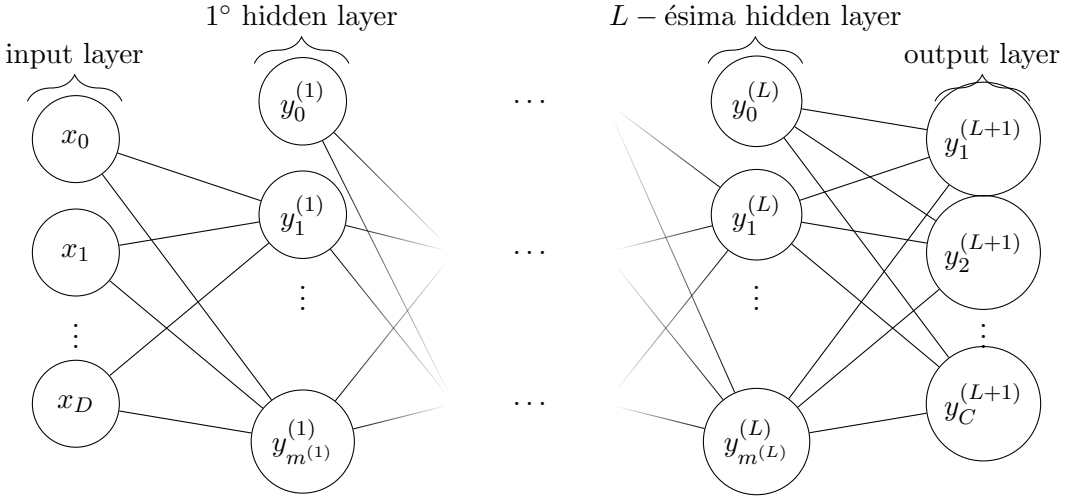



Figure 28: Graph of a $(L+1)$ layered perceptron with $D$ input units and $C$ output units. The $l^{\text{th}}$ hidden layer contains $m^{(l)}$ hidden units. David et al. [Stu]

The input layers do not perform any calculations, they simply take the input values and propagate them to the hidden layers, where the corresponding activation function is applied. Normally, all neurons in a layer apply the same activation function.

Following the hierarchy, we arrive at the *"hidden layers"*. These layers introduce the abstraction necessary to perform and make complex tasks and decisions by performing numerous operations on the input they receive.

Finally, the output layer carries the result of propagating the information from the first hidden layer to the last one, to the output. Normally the *output* of this layer is a probability, e.g., *"The probability that there is a defect in the input image is* $\mathbb{P}$*"*.

## A.2 Classification method

Understand the method of translating the results obtained in each of the neurons to a probabilistic value[52], is somewhat abstract and even complicated. Therefore, this section aims to explain this process concisely.

The quintessential algorithm for relating the characteristics of an image to a particular probabilistic outcome is Logistic Regression. To take another example, Jurafsky et al. [Mar] write the following, *"Logistic regression can be used to classify an observation into one of two classes (as positive sentiment and negative sentiment)"*. This is applicable for the topic of this thesis, therefore, logistic regression can be used to classify an observation into one of two classes such as, there

[52]Refers to: *"What is the probability that there is a defect in the input image?"*

is defect and there is no defect.

This is a discriminative classification algorithm, which means that it is based on differentiating the classes without learning much about them, e.g., knowing how to differentiate between a dog or a cat, because in all the pictures of dogs, they are wearing collars and cats are not. Jurafsky et al. [Mar]. With this in mind, the model will learn to assign a high $\mathbf{w}$ weight to the features of the image that improve its ability to discriminate between possible classes.

Since the target variable is binary, the regression model is binary, and the name it receives is: *"binary logistic regression"*. To pass from features to a binary value, the sigmoid function, which has the following form is used:

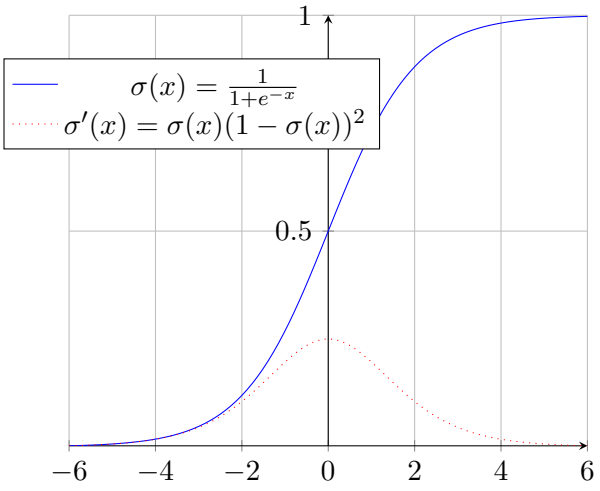



Figure 29: Graphical representation of the sigmoid function. StackExchange et al. [Stab]

With the above clear, it is time to study the dynamics of neural networks. Consider a set of features of the form: $[x_1, x_2, \ldots, x_n]$ where $\mathbf{x} \in \mathbb{R}^n$, and where the estimation carried out by the classifier model is denoted with $\hat{y}$. Each of the input characteristics is associated with a weight of the form $[w_1, w_2, \ldots, w_\#]$ where $\mathbf{w} \in \mathbb{R}^\#$. The output of the classifier will be: $\hat{y} = \mathbb{P}(\mathbf{y} = 1|\mathbf{x})$ if there are defect(s) in the image and $\hat{y} = \mathbb{P}(\mathbf{y} = 1|\mathbf{x})$ if the image contains no defects. In addition, since $\hat{y}$ represents a probability, it can be defined as follows: $\hat{y} \in \{0, 1\}$.

The logistic regression model will learn thanks to the implementation of the sigmoid function and the features $\mathbf{x}$ and the weights $\mathbf{w}$ associated to each of them. The sigmoid function is very important, because if we retake the following expression stated in the previous section: $y_i = \hat{y} = \mathbf{w}^\top \mathbf{x} + b$, one knows that $y_i$ can be negative, or much larger than 1, thus violating the above definition. But if one implements the function $\sigma(z)$ with $z = \mathbf{w} \cdot \mathbf{x} + b$[53] and look at its shape, it can be easily checked $\sigma(z) \approx 1$ if $z \uparrow\uparrow$ because $e^{-z} \to 0$. It also verifies that $\sigma(z) \approx 0$ if $z \downarrow\downarrow$ since $e^{-z}$ ends up being a negative number of large magnitude. So the signmoid function, bounding the final values between 0 and 1.

[53]The product of two element-by-element vectors is represented with the notation "$\cdot$". In addition, the vectors are represented in bold.

## A.3 Loss function and cost function

To train the regression model, it is necessary to define a cost function. For this purpose, a set of $m$ training samples is considered, which is defined as follows: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$. For each of the training samples, we will have; $\hat{y}^{(i)} = \sigma(z^{(i)})$ with $z^{(i)} = w^{\top} x^{(i)} + b$. It is also important to mention the wish that $\hat{y}^{(i)} \approx y^{(i)}$, i.e., that the estimate is similar to the real value. Once the variables have been defined, the loss function is defined as follows:

$$\mathcal{L}(\hat{y}, y) = -y \log \hat{y} - \log(1 - \hat{y}) + y \log(1 - \hat{y}) \tag{9}$$

The loss function is defined with respect to a single sample, i.e., $(x^{(i)}, y^{(i)})$ and lets you know how well you are doing in the training process. In addition, a good training will be determined by 9, which, being a loss function, the aim is to minimize its value.

Once the loss function is defined, the cost function is defined in the following equation:

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) \tag{10}$$

Substituting 9 in 10 it follows that:

$$\mathcal{J}(w, b) = \frac{1}{m} \sum_{i=1}^{m} -y \log \hat{y} - \log(1 - \hat{y}) + y \log(1 - \hat{y}) \tag{11}$$

It should be noted that while the cost function measures model error in an ensemble, the loss function handles a single instance of data.

Now the goal is to minimize $\mathcal{J}(w, b)$, to do so, it is necessary to minimize $w$ and $b$. This is where the *"gradient descent"* algorithm comes into play. This algorithm can be understood intuitively with the following figure, where the algorithm starts at a point and calculates the derivative, until the minimum of the function is found.

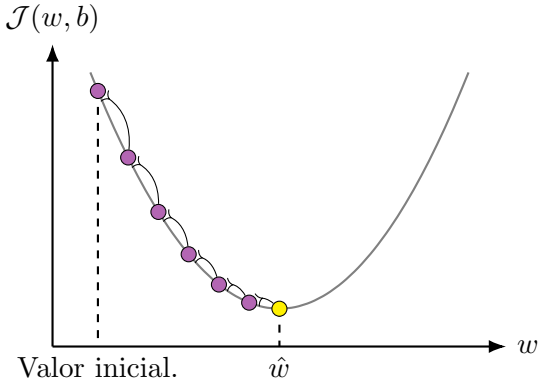



Figure 30: Graphical illustration of the gradient descent algorithm. StackExchange et al. [Stac]

The pseudocode for this algorithm can be found at the top of the next page.
The $\alpha$ parameter determines the index or learning rate. The learning rate is a hyperparameter (like **w** and **b**) used to regulate the rate at which an algorithm updates or learns the values of a parameter estimate. In other words, the learning rate regulates the weights of the neural network in relation to the loss gradient. In addition, it indicates the frequency with which the neural network updates the notions it has learned. DeepChecks et al. [Dee].

**Algoritmo 2** Gradient decrease

**procedure** GRADIENT DESCENT($\{(\mathbf{X}_i, Y_i)\}_{i\in(1,..,m)}, \alpha, \mathbf{w}_0, \mathbf{b}_0$)
  $\mathbf{w} \leftarrow \mathbf{w}_0$.
  $\mathbf{b} \leftarrow \mathbf{b}_0$.
  **for** $i \in \{1, \dots m\}$ **do**
    $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}(\mathbf{w},\mathbf{b})}{\partial \mathbf{w}.}$
    $\mathbf{b} \leftarrow \mathbf{b} - \alpha \frac{\partial \mathcal{J}(\mathbf{w},\mathbf{b})}{\partial \mathbf{b}}$.
  **end for**
  **Return** $\mathbf{w}$ and $\mathbf{b}$.
**end procedure**

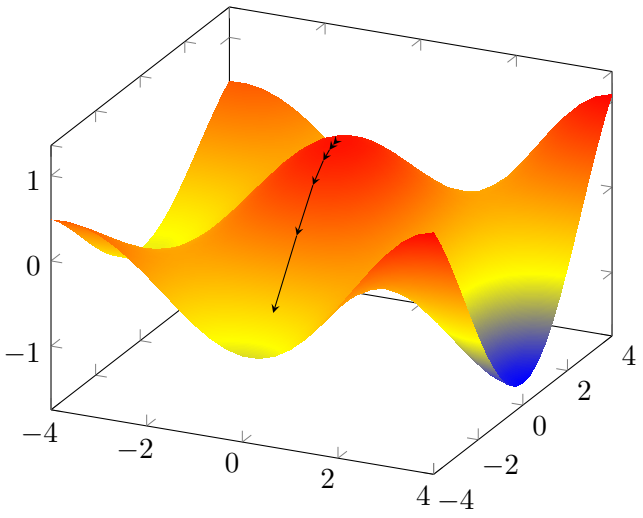



Figure 31: Arbitrary cost function. StackExchange et al. [Stad]

In the figure above, the vertical axis represents the value of the cost function, i.e., $\mathcal{J}(\mathbf{w}, \mathbf{b})$, while the two parallel axes at the bottom represent the hyperparameters $\mathbf{w}$ and $\mathbf{b}$. It should be noted that the figure represents an arbitrary cost function and depending on the problem to be solved, it will take a certain characteristic shape. As mentioned in section 1.3, the design of neural networks involves knowledge of optimization, since one of the ultimate goals of neural computation is that of cost reduction. This, from the mathematical perspective, translates into finding the minimum of a function and this is what we want to represent with the black arrow in the figure 31.

## A.4 Activation functions

The main objective of any neural network is to transform non-linearly separable input data into more linearly separable abstract features using a hierarchy of layers. The most popular nonlinear layers are activation functions. Dubey et al. [DSC21].

The activation functions play a fundamental role in the training of the network, since they condition its learning and consequently, the result of the same. It is important to mention that there is no activation function that is better than the rest, it all depends on the application to be developed, and depending on this, there will be some functions that enhance the training to a greater or lesser extent.

If $\exists x \in \mathbb{R}^{n \times m}$, or if $\exists x \in \mathbb{R}^{n}$, then the following nonlinear activation functions can be defined:

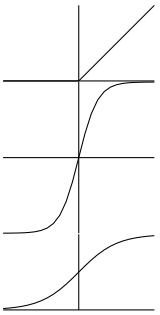
| Function | Expression | Derivative | Figure |
|---|---|---|---|
| ReLU | $f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 1 \geq 0. \end{cases}$ |  |
| tanh | $\sigma(x) = \frac{e^x - e^{-x}}{e^z + e^{-z}}$ | $f'(x) = 1 - f(x)^2$ | |
| Sigmoid | $\sigma(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)(1 - f(x))^2$ | |
| Softmax | $f(x) = \frac{e^x}{\sum_i e^x}$ | $f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$ | |

Table 8: Non-linear activation functions. The code associated with this table can be found in [Pet].

Among all the functions in the table above, the **ReLU** function has demonstrated a greater ability to obtain better results after use, compared to the rest.

The real reason why the **ReLU** is used more is that, as the layers in a CNN increase, training it becomes easier and faster, than if the *tanh* function, or the $\sigma(x)$ function is used. This is because the *tanh* function has the *vanishing gradient* problem, which is that as more layers are stacked, the gradients get smaller and smaller, and since the step in the weight space of the backpropagation algorithm is proportional to the magnitude of the gradient, when the gradients disappear, it means that the neural network can no longer be trained. The impact on training times is direct, translating into an exponential increase as the number of layers increases. For this reason, the activation function **ReLU** has been assigned to each unit of the implemented network.

It should be noted that the algorithms, functions and techniques explained throughout this section apply to any type of network, but given the complexity of the task to be performed in this project, a neural network is not sufficient to detect defects in images. To undertake a task of this scale, a set of networks and layers are required to perform specific tasks related to the processing of the input images, feature extraction and region proposal. The latter is exactly what this project has been about.

Much of the theory presented in this appendix has been extracted from the notes taken in several courses taught on the Coursera platform by Andrew Ng. These notes can be viewed at [Cou] and the link to these courses can be accessed at [AN].