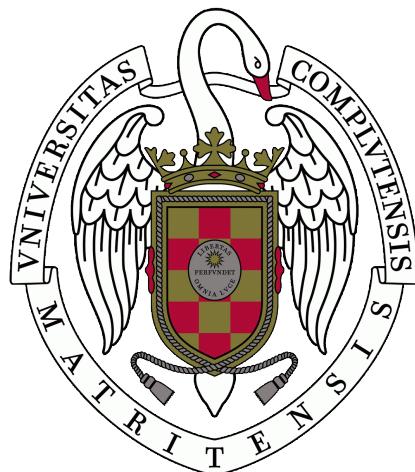


**UNIVERSIDAD COMPLUTENSE DE MADRID**  
**FACULTAD DE CIENCIAS FÍSICAS**

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA



**TRABAJO DE FIN DE GRADO**

Código de TFG: DACYA07

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

Defects detection in X-ray industrial images using Deep Learning

Supervisor/es: María José Gómez Silva y Jesús Chacón Sombría

**Pablo Suárez Reyero**

Grado en Ingeniería Electrónica de Comunicaciones

Curso académico 2022-2023

Convocatoria JULIO

*“Si tienes conocimiento, deja que otros enciendan sus velas en él”.*

---

*Margaret Fuller*

## Agradecimientos y dedicatorias

Como bien he citado en la anterior página, si se tiene conocimiento, hay que dejar que el resto enciendan sus velas en él, en mi caso, gracias al conocimiento de otros y al que he adquirido en esta carrera, he podido desarrollar este trabajo. Espero que a pesar de los errores que pueda tener esta tesis, sirva para que otros enciendan sus velas en ella.

Este trabajo va dedicado a mi madre, una gran mujer y un gran padre, luchadora y valiente. Gracias por enseñarme la importancia del amor y la verdad. A mi abuela, mi segunda madre, siempre en el banquillo esperando a salir al campo cuando algo malo pasa. A mis hermanos, Sara y Mario, gracias por vuestras experiencias y vuestra generosidad, aunque no os lo suela decir, os quiero <sup>1</sup>. A mis amigos y a todos aquellos que me mejoran y que guardan en ellos un trozo de mí. Gracias a María y a Jesús, por haberme guiado y por aquellas tutorías repentinamente. Gracias a Dios, por darme fuerzas y por hacerme creer en mi mismo. Gracias a aquellos profesores, justos y honestos que hicieron de sus clases espacios de pensamiento libre.

Gracias de corazón.

---

<sup>1</sup>solo un poco

## 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 entrenamiento 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.

# Índice

<b>1. Introducción</b>	<b>7</b>
1.1. Motivación . . . . .	7
1.2. Objetivos . . . . .	7
1.3. Relación con la carrera . . . . .	8
1.4. Estructura de la memoria . . . . .	8
1.5. Diagrama de Gant . . . . .	9
<b>2. State of art</b>	<b>10</b>
2.1. Tipo de red neuronal . . . . .	10
2.2. Conceptos y elementos clave . . . . .	11
2.3. Modelos de detección . . . . .	13
2.4. Modelos de redes neuronales . . . . .	14
<b>3. Metodología</b>	<b>20</b>
3.1. Faster R-CNN . . . . .	20
3.1.1. <i>Region Proposal Network</i> . . . . .	21
3.1.2. <i>Anchors</i> . . . . .	22
3.1.3. Dinámica general dentro de la RPN . . . . .	22
3.1.4. <i>Region of Interest</i> . . . . .	24
3.2. Base de datos . . . . .	25
3.2.1. Directorio <b>GDXray</b> . . . . .	25
<b>4. Modelado del detector</b>	<b>27</b>
4.1. Recogida de datos . . . . .	27
4.2. Procesado previo . . . . .	29
4.3. Cálculo de regiones . . . . .	29
4.4. De propuestas de regiones a ROIs . . . . .	32
4.5. Capa final, la <i>ROI Pooling</i> . . . . .	34
<b>5. Entrenamiento</b>	<b>36</b>
5.1. Parámetros . . . . .	36
5.2. Métricas . . . . .	37
<b>6. Resultados y análisis correspondientes</b>	<b>40</b>
6.1. Resultados de los entrenamientos . . . . .	40
6.2. Propuesta de regiones . . . . .	43
<b>7. Conclusiones y trabajo a futuro</b>	<b>46</b>
7.1. Conclusiones . . . . .	46
7.2. Trabajo a futuro . . . . .	47
<b>8. Códigos</b>	<b>48</b>
<b>9. Bibliografía</b>	<b>49</b>
<b>Anexos</b>	<b>53</b>

<b>A. Fundamento teórico</b>	<b>53</b>
A.1. Introducción a las redes neuronales . . . . .	53
A.2. Método de clasificación . . . . .	55
A.3. Función de pérdidas y función de costes . . . . .	57
A.4. Funciones de activación . . . . .	58

# 1. Introducción

En esta sección se introduce la motivación del proyecto, los objetivos y la estructura del documento. Al final de la sección se ha adjuntado un diagrama de Gantt que ilustra cronológicamente el desarrollo del proyecto.

## 1.1. Motivación

La tarea de detección de faltas en radiografías de piezas industriales es lenta y laboriosa, más aún cuando se trata de defectos pequeños, que es precisamente lo que atañe a este TFG. Actualmente, la autoría de esta labor no ha cambiado, y sigue siendo un trabajador el que analiza imagen por imagen, lo que conlleva una gran cantidad de tiempo y de personal cualificado. Además, al ser una persona la que busca estos defectos, esta tiene un criterio propio, a parte del adquirido en su formación.

A lo largo de estos últimos años se ha demostrado como el *deep learning* y más concretamente, las redes neuronales, poseen un gran potencial de cómputo extrapolable a cualquier ámbito que requiera de la tecnología para su desarrollo. Es importante tener en cuenta, que aunque resultan ser herramientas útiles, no alcanzan un umbral de funcionalidad hasta que no son entrenadas correctamente. A pesar de sus puntos negativos, el potencial del *deep learning* es considerable, y por esta razón, se le ha elegido para acometer la tarea de detección que ocupa a este proyecto.

Los defectos que se quieren detectar son muy pequeños, de ahí la dificultad de esta tarea. Por ello, con el fin de superar esas dificultades, la red neuronal ha de ser muy precisa para poder minimizar el error de localización. Para ello, la elección de una correcta arquitectura y los componentes que la forman (tipos de redes, capas, algoritmos, etc.) es fundamental, a fin de que la red no pierda de vista esos defectos y pueda detectarlos fácilmente, ya que en algunos casos la diferencia entre “*hay defecto*” y “*no hay defecto*” es de un par de píxeles.

Es importante destacar la carencia de modelos de detección que persigan el objetivo de este trabajo, frente al amplio abanico de modelos de detección que se centran en la búsqueda y clasificación de objetos. Por ello, la motivación de este proyecto es comprender el funcionamiento de las técnicas de detección actuales y su correcta implementación, para poder adaptarlas a la misión de este trabajo y poder delegar esta labor en una inteligencia artificial.

## 1.2. Objetivos

Considerando que el propósito de este proyecto es la identificación de defectos en radiografías, y teniendo en cuenta la utilidad de las redes neuronales para la consecución del mismo, los objetivos de este proyecto son los siguientes:

- ❖ Comprender las bases teóricas de una red neuronal.
- ❖ Preprocesar correctamente las imágenes que se van a analizar.
- ❖ Elegir las técnicas de análisis que mejor se adaptan al problema.
- ❖ Valorar la funcionalidad de la red y su arquitectura.

### 1.3. Relación con la carrera

Los conocimientos adquiridos a lo largo de la carrera han facilitado el desarrollo de este trabajo y la comprensión de conceptos, teoremas y técnicas implementados en este proyecto. A continuación se enumeran las asignaturas que directa o indirectamente han contribuido al desarrollo de esta tesis.

- **Procesamiento de Señales.** Los conocimientos de esta asignatura respecto al resto de cursos mencionados abajo, son los que más similitud guardan con los objetivos y conceptos puestos en práctica en el proyecto. En este curso, se han estudiado a fondo los filtros y técnicas de procesado de señales como las convoluciones, tanto para  $\mathbb{R}^n$  (vectores<sup>2</sup> de valores) como para  $\mathbb{R}^{n \times m}$ , i.e., imágenes.
- **Álgebra y Optimización de Sistemas.** Gran parte de los contenidos discutidos en estas dos asignaturas conforman la base teórica de las redes neuronales. El uso de vectores y matrices está a la orden del día, e.g., la combinación lineal de los  $n$  valores de entrada de una red y sus pesos asociados. Las imágenes<sup>3</sup> analizadas son interpretadas como tensores y la propia red neuronal conforma un tensor. Además, uno de los principales objetivos cuando se trata con redes neuronales, es la optimización de las funciones de coste<sup>4</sup>, con el fin de minimizarlas, de esto mismo trata el curso de Optimización de Sistemas, de comprender, dominar y aplicar diferentes técnicas de optimización.
- **Informática y Sistemas Operativos.** Estas dos asignaturas han jugado un papel clave en el desarrollo y comprensión del código implementado. En ellas se han estudiado los fundamentos de la programación, haciendo especial énfasis en el uso de la memoria y la gestión de procesos. Aunque no se han estudiado lenguajes de programación orientados a objetos, e.g., Python, se ha programado en C y C++, lenguajes de bajo nivel, dónde el programador tiene una mayor responsabilidad.

### 1.4. Estructura de la memoria

Este documento está organizado de la siguiente manera:

- **Estado del arte.** Se estudian distintos métodos de detección actuales, así como las herramientas que lo permiten.
- **Metodología.** Se explican las técnicas implementadas, así como la arquitectura que se ha escogido.
- **Modelado del detector.** Como su nombre indica, se explica el proceso que se ha seguido para que la red haga una propuesta de regiones y posteriormente consiga detectar defectos.
- **Entrenamiento.** En esta sección se entra en detalle acerca de los parámetros que influyen en el proceso de entrenamiento y las métricas que permiten caracterizarlo.
- **Resultados y análisis correspondientes.** En esta sección se explican las pruebas realizadas, se enuncia el criterio de evaluación y se exponen los resultados que posteriormente son analizados.
- **Conclusiones y trabajo a futuro.** Una vez analizados los resultados, se estudia la eficacia de las técnicas implementadas y se hace un análisis sobre posibles mejoras.

---

<sup>2</sup>Los vectores son tensores de una dimensión.

<sup>3</sup>Las imágenes son matrices, que se interpretan como tensores bidimensionales, en el caso de estar en blanco y negro. Si son imágenes RGB, entonces son descritas como tensores tridimensionales.

<sup>4</sup>Los pesos representan la intensidad de interacción entre cada neurona. Para profundizar acerca del tema, se puede consultar el anexo 9.

## 1.5. Diagrama de Gantt

A continuación, se adjunta el diagrama de Gantt que refleja el tiempo que ha tomado cada tarea en el desarrollo de esta tesis.

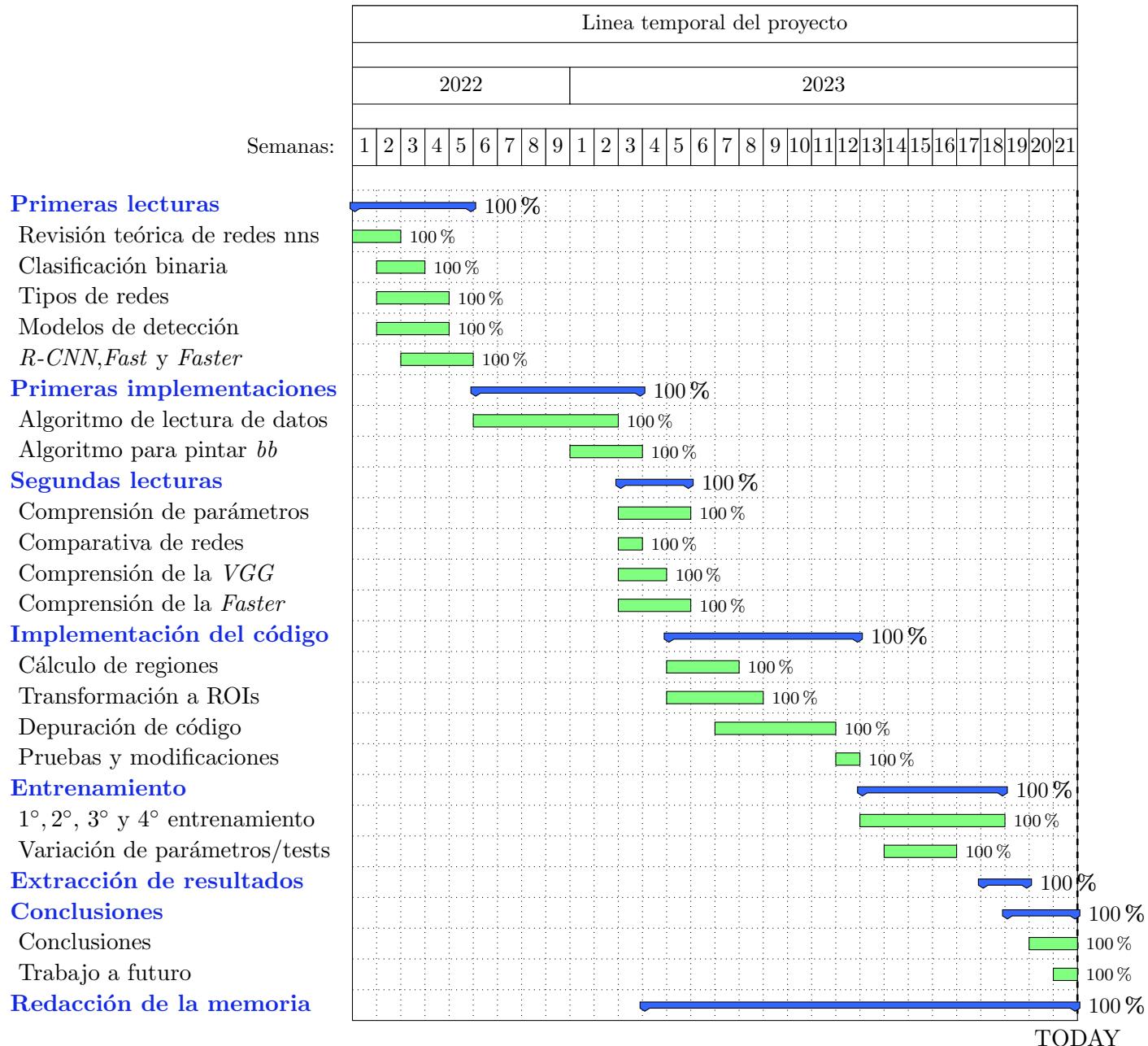


Figura 1: Diagrama de Gantt

## 2. State of art

Una correcta detección de defectos depende principalmente del **tipo** de red, de la **arquitectura** implementada, y de los algoritmos que se encargan de proponer regiones, con la finalidad de hipotetizar acerca de la localización de defectos en imágenes.

En esta sección, se estudian las opciones relativas al **tipo** de red y **arquitectura** que podrían implementarse, teniendo en cuenta los objetivos de este TFG. Adicionalmente, en la sección 9, correspondiente a los anexos, puede encontrarse la teoría que ha permitido comprender mejor todos los conceptos que se explican en esta sección. No obstante, no es obligatorio leer lo que se explica en los anexos, para comprender este proyecto, pero si se recomienda consultarlos para aclarar las dudas que puedan surgir a lo largo de este documento.

### 2.1. Tipo de red neuronal

Dentro del mundo del *deep learning* existen diferentes variantes de redes neuronales, cada una de ellas sirve para un conjunto de aplicaciones específicas. A continuación se listan los cuatro tipos de redes neuronales más extendidos y sus características.

- **Feed Forward Neural Network.** Es una de las redes más básicas, donde la información pasa por varios nodos de entrada en una dirección hasta llegar al nodo de salida. La red puede o no incluir capas de nodos ocultos. Además, este tipo de red puede implementarse para reconocer patrones y para *computer vision*, pero no son adecuadas para el *deep learning*. KnowledgeHut et al. [Kno]
- **Red neuronal funcional de base radial.** Esta es una clase especial de red neuronal que consta de sólo tres capas, capa de entrada, capa oculta y capa de salida. Como se deduce de su nombre, utiliza funciones de base radial (RBF) como la gaussiana, la multicuadrática, etc., como función de activación para las capas ocultas. Este tipo de redes suele utilizarse para la aproximación de funciones y para la predicción de series temporales. ProjectPro et al. [Pro]. Además, según Bonnano et al. [BCN<sup>+</sup>13] pueden requerir más neuronas que las **Feed Forward Neural Networks**, pero a menudo pueden diseñarse en una fracción del tiempo que lleva entrenar el tipo de red anterior.
- **Redes neuronales recurrentes.** Estas redes se construyen para comprender datos temporales o secuenciales. Las RNN mejoran sus predicciones utilizando puntos de datos adicionales en una secuencia. Para modificar la salida toman datos de entrada y reutilizan las activaciones de nodos anteriores o posteriores de la secuencia. Es relevante mencionar que este tipo de redes, sufren del problema del desvanecimiento del gradiente. KnowledgeHut et al. [Kno]
- **Redes neuronales convolucionales.** Según Zewen Li et al. [LYPL20] es una de las redes más significativas en el campo del aprendizaje profundo. Este mismo autor, describe a las redes neuronales convolucionales como redes capaces de extraer características de los datos con estructuras de convolución. Además, este tipo de redes no extraen características manualmente.

Shengli Jiang et al. [JQPZ22] recuerdan el propósito inicial de este tipo de red, que era y sigue siendo, la visión por ordenador, para posteriormente destacar la forma tan flexible que este tipo de red tiene para representar datos, y como esta cualidad ha permitido su implementación en muchos otros campos a parte de la visión por ordenador, como en la industria química o en el campo de la biología.

Este tipo de red es la más adecuada para la clasificación y detección de objetos, porque es capaz de procesar matrices de varias dimensiones, e.g., tensores. En el caso de las imágenes a color, su representación matemática vendrá caracterizada por tres matrices apiladas una detrás de otra. Cada una de esas matrices representa un canal que matemáticamente es descrito como un tensor de dos dimensiones, y al apilar tres matrices, se tiene un tensor de 3 dimensiones. En el caso de tener una imagen en blanco y negro se tendría una sola matriz que tendría las dimensiones de la imagen original, véase la figura 2

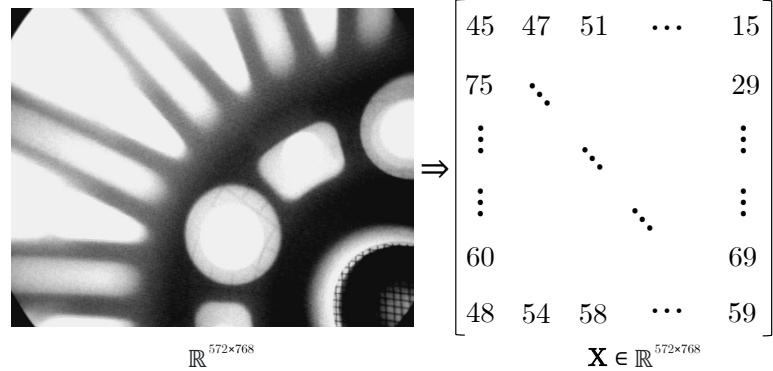


Figura 2: Imagen en blanco negro de una pieza de fundición con su pseudo representación matricial.

La red recibe una imagen a la entrada (como la que se ve en la figura 2) y se extraen sus características mediante una convolución. De asignaturas como Sistemas lineales o Procesamiento de señales, se sabe que la convolución puede definirse de la siguiente manera:

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

Dónde  $\mathbf{X} \in \mathbb{R}^{n \times m}$  representa la imagen de entrada,  $\mathbf{w} \in \mathbb{R}^i$  representa los pesos y  $s$  viene determinada por el tamaño del filtro y representa el *stride*<sup>5</sup>.

Como lo que se persigue en este trabajo, es la detección de defectos mediante el uso del *deep learning*, habiendo revisado las opciones de arriba, la elección no es difícil. Por ello, la implementación que se ha llevado a cabo, está centrada en el uso de redes convolucionales.

Con la finalidad de detectar los defectos en las imágenes, el siguiente objetivo de este trabajo consiste en implementar una arquitectura de red basada en redes neuronales convolucionales. En la sección 2.4, se estudian las arquitecturas existentes en cuanto a redes convolucionales y se analizan las funcionalidades que ofrecen algunas de las más relevantes en el panorama actual. Para ello, se han de explicar previamente, los conceptos fundamentales que permiten su funcionamiento.

## 2.2. Conceptos y elementos clave

Antes de investigar acerca del funcionamiento de distintos modelos de red y explicar su dinámica, es pertinente explicar los conceptos que caracterizan cualquier arquitectura y que hacen posible

---

<sup>5</sup>Este concepto es explicado en la siguiente sección (2.2)

la extracción de sus aspectos más relevantes, para posteriormente crear su mapa de características<sup>6</sup>.

Uno de estos conceptos, es el *stride*. El *stride* representa el número de desplazamientos de píxeles sobre la matriz de entrada. Cuando el *stride* es 1 entonces los filtros de convolución se mueven a 1 píxel a la vez. Cuando el *stride* es 2 entonces los filtros se mueven 2 píxeles a la vez y así sucesivamente. En la figura 3 se representa de manera intuitiva este concepto sobre una matriz de valores, que puede corresponderse con una imagen en blanco y negro.

Ahora, supóngase que dicha imagen es de la forma  $\mathbf{X} \in \mathbb{R}^{w \times w}$  y que a esta se le aplica un filtro definido como:  $\mathbf{K} \in \mathbb{R}^{n \times n}$ . Al realizar la convolución entre estos dos, la imagen resultante tendrá el siguiente tamaño:  $(w - n + 1) \times (w - n + 1)$ .

$$\begin{array}{|c|c|c|c|c|c|c|} \hline
 0 & 1 & 1 & \underset{\times 1}{\textcolor{red}{1}} & 0 & 0 & 0 \\ \hline
 0 & 0 & 1 & \underset{\times 0}{1} & \underset{\times 1}{1} & 0 & 0 \\ \hline
 0 & 0 & 0 & \underset{\times 1}{1} & \underset{\times 0}{1} & \underset{\times 1}{1} & 0 \\ \hline
 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline
 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline
 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline
 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline
 \end{array}
 \quad *
 \quad
 \begin{array}{|c|c|c|} \hline
 1 & 0 & 1 \\ \hline
 0 & 1 & 0 \\ \hline
 1 & 0 & 1 \\ \hline
 \end{array}
 \quad =
 \quad
 \begin{array}{|c|c|c|c|c|} \hline
 \textcolor{blue}{1} & 4 & 3 & 4 & 1 \\ \hline
 1 & 2 & \textcolor{blue}{4} & 3 & 3 \\ \hline
 1 & 2 & 3 & 4 & 1 \\ \hline
 1 & 3 & 3 & 1 & 1 \\ \hline
 3 & 3 & 1 & 1 & 0 \\ \hline
 \end{array}$$

Figura 3: Representación de una convolución. StackExchange et al. [Staa]

Es importante mencionar como la convolución introduce dos problemas:

- Al convolucionar, el tamaño de la imagen original se ve reducido, tómese como ejemplo,  $w = 6$  y  $n = 3$ . Realizada la convolución, se pasa de  $\mathbf{X} \in \mathbb{R}^{6 \times 6}$  a  $\mathbf{X} \in \mathbb{R}^{4 \times 4}$ . Como ya se verá en el apartado de **arquitectura** de redes, en la tarea de clasificación de imágenes hay múltiples capas de convolución por lo que al aplicar esta operación varias veces, la imagen original se habrá reducido considerablemente, y esto es algo que se quiere evitar.
  - El segundo problema ocurre cuando el kernel<sup>7</sup> se desplaza sobre las imágenes originales, ya que este pasa por el centro muchas veces, mientras que por los bordes pasa con mucha menos frecuencia. Al pasar tantas veces por el centro, se va solapando, lo contrario a lo que pasa en los bordes, y como consecuencia, la información que contienen no se ve casi reflejada a la salida.

Con el fin de resolver estos dos problemas, se introduce el concepto de *padding*. Este método consiste en rodear la imagen original de ceros, para que a la hora de aplicar la convolución, el tamaño final de la imagen sea igual al original. Al aplicarle un *padding*  $\mathbf{p}^8$  a la imagen, y al convolucionarla con un filtro de tamaño  $n \times n$  y con un *stride*  $\mathbf{s}^9$ , el tamaño final de la imagen será:  $(\frac{w-n+p+s}{s}) \times (\frac{h-n+p+s}{s})$

<sup>6</sup>El mapa de características de una imagen, es la salida de una capa convolucional que representa características específicas de la imagen de entrada. [bae]

<sup>7</sup>Un *kernel* en el ámbito de las redes neuronales, representa el filtro que se aplica a las imágenes, con el objetivo de extraer ciertas características importantes. BootCampAI et al. [Boo]. No confundir con el *kernel* de un sistema operativo.

<sup>8</sup> Aplicar un padding **p** a una imagen cualquiera, significa añadir **p** filas de ceros y **p** columnas de ceros.

<sup>9</sup>Se supone un **stride** de igual magnitud, al mover el filtro de derecha a izquierda que al moverlo de arriba abajo.

$$\begin{array}{|c|c|c|c|c|} \hline
 0 & 0 & 0 & 0 & 0 \\ \hline
 0 & 0 & 1 & 2 & 0 \\ \hline
 0 & 3 & 4 & 5 & 0 \\ \hline
 0 & 6 & 7 & 8 & 0 \\ \hline
 0 & 0 & 0 & 0 & 0 \\ \hline
 \end{array} \quad * \quad
 \begin{array}{|c|c|} \hline
 0 & 1 \\ \hline
 2 & 3 \\ \hline
 \end{array} \quad = \quad
 \begin{array}{|c|c|} \hline
 0 & 8 \\ \hline
 6 & 8 \\ \hline
 \end{array}$$

**X**

**K**

**X \* K**

Figura 4: Convolución de una imagen con  $padding \mathbf{p} = 1$ . Figura inspirada en D2l et al. [D2l]

Aplicando esta técnica, la imagen no perdería su tamaño original, que es precisamente lo que se busca.

Es importante notar, como en la sección 3.1.3 se vuelve a hablar del *stride* pero aunque los dos términos compartan nombre, estos no representan lo mismo. En el contexto de esta sección, el *stride* hace referencia al tamaño del filtro, mientras que en la sección 3.1.3, representa el factor de reducción de la imagen original, que está directamente ligado a la profundidad de la red, y por ende al número de filtros que se le aplican a la imagen en el *backbone*<sup>10</sup>.

### 2.3. Modelos de detección

En este apartado se introducen brevemente las técnicas de detección más relevantes dentro del campo de *computer vision*, y que son fundamentales para la comprensión de este trabajo de fin de grado.

Cabe mencionar, que la estructura de red implementada (i.e., la *Faster R-CNN*) es fruto de un proceso evolutivo que comienza con la red **R-CNN**. Posteriormente, en la sección 3.1 se explica en profundidad y con todo detalle, la **Faster R-CNN**.

El campo del *computer vision*, en castellano, visión por ordenador, desde su concepción como disciplina científica, siempre ha estado a la vanguardia de la técnica, pues según avanzaba la capacidad de cómputo, el potencial de esta rama del conocimiento lo hacía con el. Aunque ha habido muchos avances en este campo tras su consolidación en la comunidad científica, uno de los hitos más importantes, lo trajo la familia de modelos de *machine learning*: **R-CNN** cuyas siglas significan, Redes neuronales convolucionales basadas en regiones.

Este conglomerado de modelos que conforman la **R-CNN** se centran en la visión por ordenador, y tienen como tarea principal la detección de objetos. El propósito inicial de su uso reside en la producción de un conjunto de *bounding boxes* que recuadran los objetos de una imagen, pero desde entonces, la manera de obtener estos recuadros ha cambiado drásticamente, permitiendo que esta familia de algoritmos se extienda a otras áreas de la visión por ordenador.

Esta pequeña revolución comenzó con el “lanzamiento” de la **R-CNN** en 2013. Dada una imagen de entrada, la **R-CNN** comienza aplicando un mecanismo llamado Búsqueda Selectiva<sup>11</sup> para

---

<sup>10</sup>Este término hace referencia al corazón de la arquitectura de red.

<sup>11</sup>En la sección 3 se explica brevemente este algoritmo.

extraer regiones de interés (*ROI*<sup>12</sup>), donde cada **ROI** es un rectángulo que puede representar el límite de un objeto en la imagen. Este modelo aplica 2000 veces la red neuronal, por lo que podía haber hasta 2000 *ROIs*. Wikipedia et al, [Wik22]

Posteriormente, en Abril del 2015, se presenta su sucesora, la que traería la mayor mejora hasta la fecha en esta familia de modelos, la **Fast R-CNN**. Esta arquitectura acomete una mejora considerable computacionalmente hablando, pues a diferencia de la primera versión, la *Fast R-CNN* implementa la red neuronal sobre la imagen una sola vez. Además, esta red introduce un conjunto de capas *pooling*<sup>13</sup> al final de la arquitectura. Al igual que en la **R-CNN** original, la *Fast R-CNN* utiliza la búsqueda selectiva para generar sus propuestas de regiones. Wikipedia et al, [Wik22]

Dos meses después, en junio del 2015, el protagonismo de la **Fast R-CNN** se vería ligeramente eclipsado por un nuevo miembro de la familia, la **Faster R-CNN**. Si bien se habla constantemente de extraer el máximo potencial a la inteligencia artificial, este nuevo modelo cumple con esta premisa, pues en vez de implementar la búsqueda selectiva propia de las dos versiones anteriores, en esta arquitectura la búsqueda de regiones de interés es acometida al incluir la capa *ROI pooling* en la propia red neuronal. Wikipedia et al, [Wik22]

Con el fin de otorgar una mayor precisión de búsqueda a la red, en 2017 se presenta la **Mask R-CNN**, este modelo se centra en aumentar la resolución de búsqueda sustituyendo la capa *ROI pooling*, por la *ROIAlign* que es capaz de representar fracciones de un píxel. Este modelo resulta atractivo para su implementación en este trabajo de fin de grado, pues como se explica en la sección 2.4, la dificultad de este proyecto reside en el tamaño de los defectos que se quieren detectar, y esta red podría superar estas restricciones de resolución. Pero este modelo respecto al anterior introduce un nivel más de complejidad que no se ha creído oportuno abordar en esta tesis.

Y por último, en junio de 2019, se desarrolló la **Mesh R-CNN**, que traería la capacidad de generar mallas de 3 dimensiones, a partir de imágenes de dos dimensiones. Wikipedia et al, [Wik22]

## 2.4. Modelos de redes neuronales

Zewen Li et al. [LYPL20] explica los modelos de CNNs que han ido surgiendo a lo largo del tiempo, en la figura 5 puede verse el año de nacimiento de los modelos de redes convolucionales más relevantes.

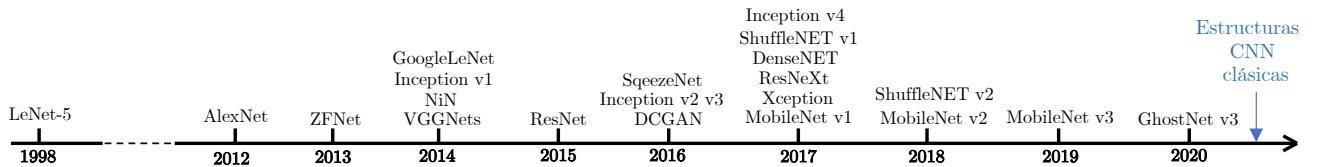


Figura 5: Cronología de los modelos más relevantes. Figura creada a partir de una ilustración de Zewen Li et al. [LYPL20]

<sup>12</sup>Como su traducción en castellano indica, el término *ROI* hace referencia a una región que resulta interesante para la red.

<sup>13</sup>En este contexto, las capas *pooling*, trocean cada ROI del tensor de salida de la red y les da nueva forma para finalmente clasificar dicha región de interés.

Cabe destacar, que los tipos de redes convolucionales que se muestran en la figura de arriba (5) tienen como objetivo principal, la tarea de clasificación, y aunque este trabajo no se centra en la clasificación de defectos, sino en su detección, este tipo de redes suelen incluirse como *backbone*<sup>14</sup> dentro de una arquitectura de red más compleja, como la que se explica en la sección 3.1. A continuación, se destacan los aspectos más importantes de algunas de las arquitecturas más relevantes que se muestran en la figura 5.

- **LeNet-5.** Zewen Li et al. [LYPL20] describe a la red **LeNet-5** como una red neuronal convolucional eficiente, que es entrenada con el algoritmo de retropropagación para el reconocimiento de escritura. Esta red, tiene siete capas de entrenamiento, que contienen dos capas convolucionales, dos capas *pooling*<sup>15</sup> y tres capas totalmente conectadas.
- **AlexNet.** Esta red fue diseñada por Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton. En su artículo, Alex et al. [KSH12] describe la arquitectura de la red. Esta red contiene ocho capas con pesos; las cinco primeras son convolucionales y las tres restantes están totalmente conectadas. La segunda capa convolucional, así como la cuarta y la quinta se conectan únicamente a los mapas de *kernels* de la capa anterior que residen en la misma GPU (véase la figura 6). Los *kernels* de la tercera capa convolucional están conectados a todos los mapas de *kernels* de la segunda capa. Las neuronas de las capas totalmente conectadas están conectadas a todas las neuronas de la capa anterior. Según Alex et al. [KSH12], este modelo de red lleva adelante las ideas de **LeNet** y aplica los principios básicos de las CNNs a una red profunda y amplia.

En el mismo artículo, se destacan los buenos resultados obtenidos tras el entrenamiento de la red, pero Alex et al. [KSH12] también enfatiza la importancia de las capas convolucionales, pues si se elimina cualquiera de las capas intermedias, el rendimiento de la red disminuye, resultando en un aumento de pérdidas del 2 %. No obstante, este modelo es una buena opción. Además, esta arquitectura introdujo novedades, como el uso de la función **ReLU**<sup>16</sup> con el fin de mitigar el problema del desvanecimiento del gradiente en redes profundas. También destaca el uso del *dropout* que ignora aleatoriamente algunas neuronas durante el entrenamiento para evitar el sobre-ajuste<sup>17</sup>

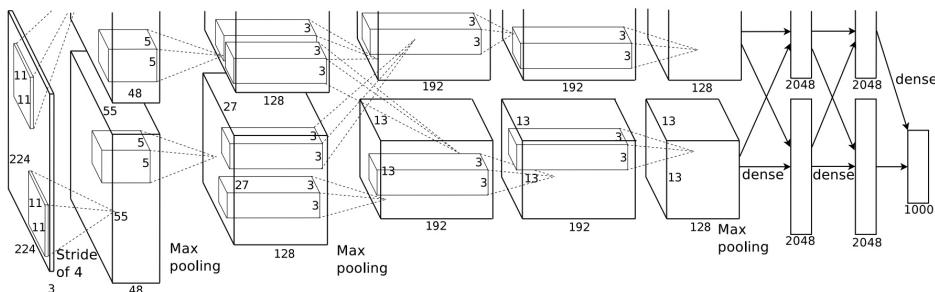


Figura 6: Arquitectura de la red **AlexNet**, Alex et al. [KSH12]

<sup>14</sup>El término *backbone*, hace referencia al corazón, o núcleo de una estructura de red más compleja. En este trabajo, su cometido es el de crear el mapa de características a partir de la imagen que recibe como entrada.

<sup>15</sup>Las capas *pooling*, son capas que se añaden después de la etapa convolucional. Estas capas reducen las dimensiones de los mapas de características. De este modo, se reduce el número de parámetros que hay que aprender y la cantidad de cálculos realizados en la red. Kaggle et al. [Kag]. Más adelante se entra en detalle acerca del propósito y funcionamiento de esta capa.

<sup>16</sup>Aunque para comprender esta tesis no es necesario entender esta función de activación, se puede visitar el anexo A.4, para profundizar más acerca de este tipo de funciones.

<sup>17</sup>El *overfitting*, en castellano, sobre-ajuste, ocurre cuando el modelo no puede generalizar y, en cambio, se ajusta demasiado al conjunto de datos de entrenamiento. IBM et al. [IBM].

- **VGGNets.** Las siglas **VGG** son el acrónimo de, *Visual Geometry Group*, pues fue este grupo perteneciente a la Universidad de Oxford, quien desarrolló esta red, y quien ganó un premio por ella. El autor de esta red, Simonyan et al. [SZ15] describe este modelo como una serie de algoritmos de redes neuronales convolucionales, que incluye, la VGG-11, la VGG-11-LRN, la VGG-13, la VGG-16 y la VGG-19.

Todas las configuraciones de este tipo de red, están constituidas por bloques, además todas cuentan con dos *hidden layers*<sup>18</sup> totalmente conectadas y una capa a la salida completamente conectada. En todas las variantes de este modelo, la imagen pasa por una torre de capas convolucionales, en las que Simonyan et al. [SZ15] implementa filtros de tamaño  $3 \times 3$ <sup>19</sup>.

Para conservar la resolución a la salida de la convolución, el *stride* de la convolución se fija en 1 píxel. Por último, se implementa una agrupación espacial (en inglés: *spatial pooling*), que es una forma de calcular la representación de la imagen a partir de sus características locales codificadas. Además, esta técnica divide la imagen en diferentes subregiones y calcula el vector de características de cada subregión. La representación final de la imagen es una concatenación de todos los vectores de características de las subregiones. Xinggang Wang et al. [Wan]. Esta técnica se lleva a cabo con la implementación de cinco capas de tipo *max-pooling*. El *max-pooling* se realiza con una ventana de  $2 \times 2$  píxeles, con un intervalo de 2.

Para la explicación de este tipo de red **VGG**, se pone el foco sobre la **VGG-16**, no obstante, el resto de tipos de red dentro de la familia **VGG** están construidos de manera similar. Véase gráficamente esta arquitectura en la siguiente figura (7):

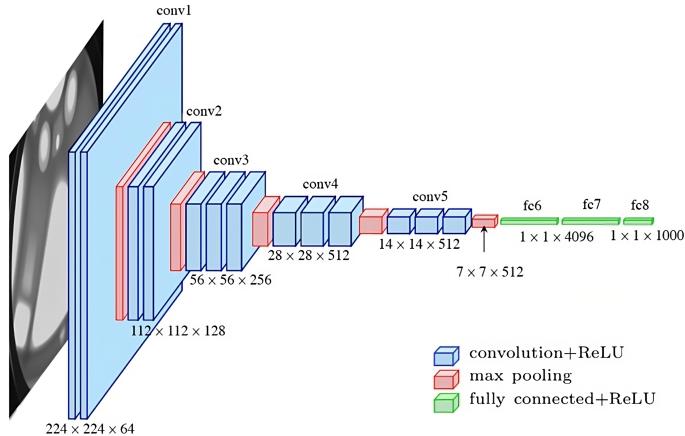


Figura 7: Arquitectura de la red **VGG-16**. Simonyan et al. [SZ15]

La VGG-16 se compone de 13 capas convolucionales, 5 *max pooling layers* y 3 capas totalmente conectadas. Por lo tanto, hay 16 capas con parámetros ajustables, ya que son 13 capas convolucionales y 3 capas totalmente conectadas, de ahí el nombre VGG-16. En el primer bloque hay 64 filtros, y este número se ve duplicado en los bloques posteriores hasta llegar a 512 filtros. Este modelo se completa con dos capas ocultas totalmente conectadas y una capa de salida, cabe destacar que las dos capas completamente conectadas tienen el mismo número de neuronas y cada una cuenta con 4096 neuronas. Originalmente, la capa de salida tiene

<sup>18</sup>Para profundizar más acerca de esta propiedad dentro de las redes neuronales, se puede visitar el anexo A.1.

<sup>19</sup>Este es el tamaño más pequeño para captar la noción de izquierda/derecha, arriba/abajo.

1000 neuronas correspondientes al número de categorías del conjunto de datos Imagenet<sup>20</sup>.

Los resultados obtenidos con esta red son buenos, ya que VGG-16 fue una de las arquitecturas con mejor rendimiento en la competición del 2014, pues sus autores obtuvieron un error de clasificación del 7,32 % (solo por detrás de GoogLeNet, con un error de clasificación del 6,66 %). También fue la ganadora en el ámbito de localización, con un 25,32 % de error de localización.

A pesar de sus ventajas, esta red tiene algunos inconvenientes, como son la lentitud a la hora de entrenar, y la cantidad ingente de parámetros que necesita, que puede concluir en problemas con el gradiente. No obstante, esta arquitectura sigue siendo una muy buena opción.

- **ResNet.** Este nombre viene de *Residual Networks*, para el análisis de esta arquitectura, se tienen en cuenta dos variantes, la de **18** capas y la de **34**. En [HZRS15] se observa como las capas convolucionales tienen en su mayoría filtros de  $3 \times 3$  y siguen dos sencillas reglas de diseño: (i) para un mismo tamaño del mapa de salida, las capas tienen el mismo número de filtros; y (ii) si el tamaño del mapa de características se reduce a la mitad, el número de filtros se duplica para preservar la complejidad temporal por capa. Luego, se reduce la tasa de muestreo mediante capas convolucionales que tienen un paso de 2. La red termina con una capa *global average pooling layer* y una capa totalmente conectada de 1000 neuronas con *softmax*<sup>21</sup>. Finalmente, el número total de capas ponderadas (*weighted layers*) es de 34.

Además, comparando esta arquitectura con la anterior, Kaiming He et al. [HZRS15] destaca la diferencia en el número de filtros y la menor complejidad que existe en esta arquitectura. El modelo base de 34 capas (ResNet) tiene 3.600 millones de FLOPs (multiplicaciones), frente a los 19.600 millones de FLOPs en redes VGG, esto se traduce en un 18% menos. La figura 8 ilustra la composición de ResNet, donde además puede verse como la arquitectura de la misma es mucho más profunda<sup>22</sup> que la arquitectura de la VGG-19.

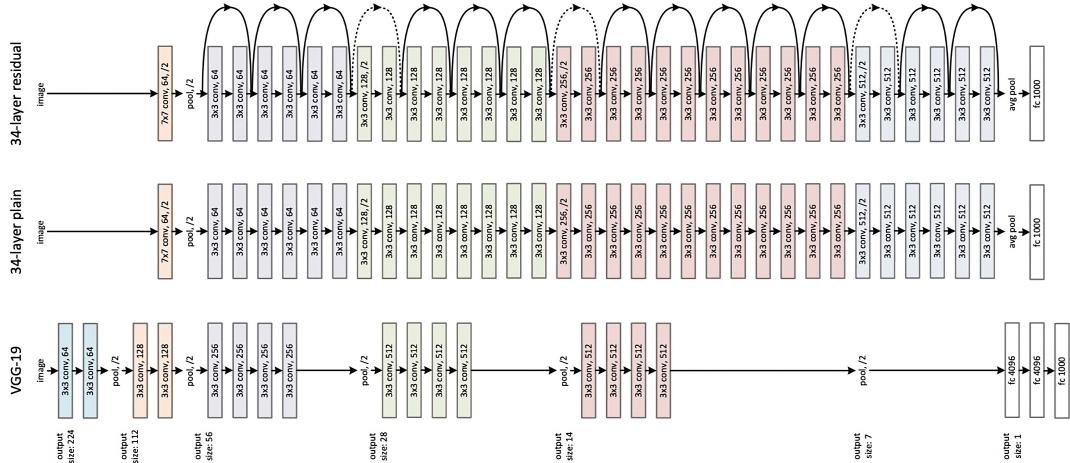


Figura 8: Arquitectura de la red **ResNet**, comparada con una VGG-19. Kaiming He et al. [HZRS15]

Tras este análisis de posibles arquitecturas, se llega a la conclusión de que para poder entrenar

<sup>20</sup>Esta es la base de datos que usaron en la competición que ganaron.

<sup>21</sup>Esta capa está explicada en la sección 3.1.4.

<sup>22</sup>El término profundidad en este caso, hace referencia al número de capas de cada tipo.

correctamente una red que detecte defectos en imágenes, se debería de recurrir a una de las dos últimas arquitecturas que se han mencionado arriba, i.e., **VGG-16** o **ResNet**.

Estas dos arquitecturas son capaces de generar buenos resultados, y a pesar de sus diferencias, llegan a ser semejantes en muchos aspectos, pero un aspecto importante es que, ResNet resuelve uno de los problemas que surgen en VGG, el problema del gradiente<sup>23</sup>. En la Resnet este problema es abordado en la primera capa, donde el número de filas y columnas se reducen en un factor de 2, además en la siguiente operación de *max pooling* se aplica otra reducción con un factor de 2.

Con el fin de elegir correctamente el tipo de arquitectura, se han buscado resultados en distintos ámbitos de detección. Victor Ikechukwu et al. [VMDS21] obtiene los resultados que se ven en el cuadro 1, tras implementar modelos pre-entrenados de una VGG-19 y de una ResNet-50 con el fin de discriminar radiografías con neumonía de radiografías “sanas”.

Métricas	<b>VGG-19</b>	<b>ResNet-50</b>
Exactitud (%)	97.3	96.2
Especificidad (%)	97.2	96.4
Precisión (%)	96.7	95.3

Cuadro 1: Caption

En el cuadro 1 puede verse como la VGG-19 es más precisa y exacta que la ResNet, aunque la diferencia es de  $\sim 1\%$ . Es importante destacar, como el estudio realizado en [VMDS21] resulta relevante para este tema, porque las imágenes implementadas están en escala de grises, al igual que las que se utilizan en este proyecto, así mismo, Victor Ikechukwu et al. [VMDS21], busca detectar si hay o no neumonía, por lo que solo hay una clase, al igual que en las imágenes que se utilizan para entrenar la red en este proyecto. Un punto desfavorable, es que a diferencia de los defectos que se buscan detectar en este trabajo, la proporción del defecto en [VMDS21] suele ser más grande en comparación a los defectos de este proyecto, véase la siguiente comparativa.

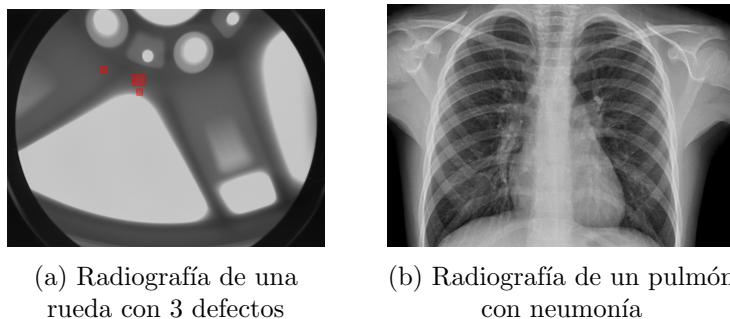


Figura 9: Comparativa de proporciones (defecto/imagen)

Es importante mencionar, que la decisión que se ha tomado entre las dos opciones de arquitectura, **VGG** o **ResNet**, está basada en el *stride* y en la profundidad de la arquitectura, a continuación se explica porque.

---

<sup>23</sup>Aunque para el desarrollo de esta tesis, no ha sido necesario comprender este concepto, en el anexo A.4 se profundiza más acerca de este problema.

En la actualidad, uno puede encontrar muchos modelos pre-enetrenados en internet, o simplemente arquitecturas, capaces de diferenciar entre perros y gatos, detectar coches en imágenes o saber si una persona está feliz o triste. Todos estos ejemplos, suelen guardar una similitud importante, el tamaño de la característica, los perros y los gatos suelen ser fácilmente reconocibles, puesto que son considerablemente grandes dentro de la imagen, lo que significa que  $\frac{\text{size}(\text{perro} \sqcup \text{gato})}{\text{size}(\text{imagen})} \in (0,4 - 1)$ , esto mismo aplica al resto de ejemplos (la mayoría de las veces). Hay otras características que facilitan la tarea de clasificación, como puede ser el color, ya que se suelen utilizar imágenes RGB, lo que permite cercar el área de búsqueda más rápidamente, ya que e.g., solo hay gatos y perros de determinados colores, lo mismo pasa con otros objetos o seres, como las plantas. Otra característica fundamental son las siluetas, ya que los perros, los gatos, los coches y las personas, tienen formas muy características, que resultan fáciles de reconocer.

La mayoría de los modelos que uno puede encontrar en internet y en artículos científicos se desenvuelven bien, cuando el conjunto muestral cumple con varias de las características mencionadas arriba. Pero desafortunadamente, las imágenes que se manejan en este proyecto, no cumplen con ninguna de ellas, ya que;

- La proporción defecto/imagen es pequeña, i.e.,  $\frac{\text{size}(\text{defecto})}{\text{size}(\text{imagen})} \in (0,01 - 0,15)^{24}$ .
- Las imágenes están en blanco y negro, por lo que el defecto solo es apreciable gracias a un mayor contraste o un tono de gris o negro diferente al resto de la imagen.
- Los defectos suelen ser irregulares<sup>25</sup>, por lo que no comparten silueta.

Estas características pueden torpedear el entrenamiento de la red, por lo que la elección de una buena arquitectura es importante. Sabiendo que los defectos son muy pequeños, uno no puede escoger una red muy profunda, esto mismo ocurre para las otras dos características de arriba, ya que a mayor profundidad de red, el *stride* es mayor, y al aumentar este parámetro, el mapa de características disminuye, y si este disminuye, los defectos de por si pequeños, desaparecen por completo.

Por este conjunto de razones y a pesar de que en [HZRS15] se obtengan muy buenos resultados con la ResNet en comparación a la VGG, se ha optado por una arquitectura **VGG-16**, pues no es una red tan profunda como la ResNet, pero implementa múltiples mejoras respecto a sus predecesores, como un entrenamiento más rápido y una mayor exactitud.

Ya se ha escogido el corazón de la arquitectura de red, pero aún se necesitan más funcionalidades. En la siguiente sección se introduce la arquitectura general del trabajo, dentro de la cual estará la **VGG-16**.

---

<sup>24</sup>Estas cifras son aproximadas, y se han calculado dividiendo el tamaño medio del rectángulo que cubre un defecto, entre el tamaño total de la imagen.

<sup>25</sup>Aunque en muchas de las imágenes los defectos son redondos, en el resto de imágenes, estos son amorfos,

### 3. Metodología

Después de haber estudiado las opciones en cuanto a tipos de redes y arquitecturas que pueden formar, y habiendo escogido la **VGG-16**, en esta sección se analiza en detalle las características de la arquitectura final del proyecto.

En el campo de *computer vision*, se han conseguido muy buenos resultados en cuanto a la detección de objetos, gracias a las redes convolucionales y a los algoritmos de propuestas de regiones. Estos algoritmos, que normalmente son implementados a modo de red, tienen como objetivo principal, conjutar sobre posibles regiones. Por lo que la idea, es hacer uso de esta técnica o similares, para afinar el tiro a la hora de decidir donde puede o no haber un defecto.

Continuando con la idea de la técnica mencionada antes, en [USGS13] se implementa la *Selective Search*, esta “búsqueda selectiva” tiene como fin, hipotetizar acerca del posicionamiento de objetos dentro de una imagen, esta técnica combina el potencial de dos tipos de búsqueda, el de la búsqueda exhaustiva y el de la búsqueda por segmentación. En este mismo artículo se demuestra el potencial de este tipo de estructuras y los buenos resultados alcanzados. Como este artículo, hay muchos otros, que justifican el uso de este tipo de implementaciones, de hecho, avances recientes en la detección de objetos han sido posibles gracias a técnicas como la de J.R.R. Uijlings et al. [USGS13]. El problema con esta técnica, es que aplica 2.000 veces la CNN a las áreas propuestas, y esto introduce cierto desgaste computacional. En el siguiente párrafo se introduce el predecesor de esta técnica, el cual solo le pasa la imagen original una sola vez, a un modelo pre-entrenado de redes convolucionales.

La **RPN** (*Region Proposal Network*) es la predecesora de la técnica anterior. Esta red es comúnmente usada en redes de detección de objetos más modernas que dependen de algoritmos de propuesta de regiones para hipotetizar acerca de la ubicación de objetos. Shaoqing Ren et al. [RHGS15].

Es importante mencionar, que estas redes resultan ser un cuello de botella, no obstante, Shaoqing Ren et al. [RHGS15] han desarrollado una alternativa que soluciona el problema, y que se explica en la siguiente sección.

#### 3.1. Faster R-CNN

Debido a su potencial y los buenos resultados alcanzados por Shaoqing Ren et al. [RHGS15], en esta tesis se ha implementado su sistema de detección de objetos. En la figura 10 puede verse la arquitectura general de la *Faster R-CNN*.

Tal y como se explica en [RHGS15], esta red esta formada por dos módulos principales. El primer módulo es una red totalmente convolucional que propone regiones y el segundo módulo es el detector *Fast R-CNN*, encargado de utilizar las regiones propuestas, por lo tanto, tal y como se dice en [RHGS15], el módulo RPN indica al módulo *Fast R-CNN* dónde ha de buscar. En los siguientes apartados se estudia en detalle las funciones que implementa esta arquitectura.

### 3.1.1. Region Proposal Network

Antes de explicar esta red, se han de responder a las siguientes dos preguntas:

- **¿Qué recibe esta red a la entrada?** A esta red le entra una imagen  $\mathbf{A}$  de cualquier tamaño, e.g.,  $\mathbf{A} \in \mathbb{R}^{n \times m}$  en el caso de ser monocromática, o  $\mathbf{A} \in \mathbb{R}^{n \times m \times c}$  si tiene más de un color<sup>26</sup>
- **¿Qué hay a su salida?** A su salida, se expide un conjunto de propuestas (“cajas” rectangulares), cada una de ellas con una puntuación, calculada por la función `non_maximum_suppression`.

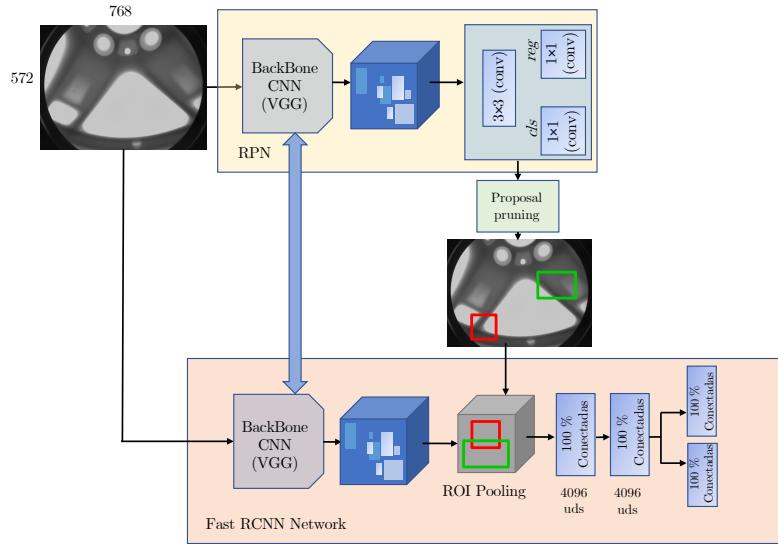


Figura 10: El conjunto de la arquitectura de la **RPN** y la **Fast R-CNN** dan lugar a la **Faster R-CNN**. Figura creada a partir de una ilustración en [Ana]

La **RPN** toma como entrada una imagen de cualquier tamaño, y a la salida expide un conjunto de objetos en forma de “cajas” rectangulares que han sido propuestas por ella, cada una de esas cajas tiene asociada una calificación probabilística que indica la certeza de que esa “caja” cubra un defecto en la imagen. Esta dinámica se modela con una red convolucional, que se describe en esta sección.

Para poder generar propuestas de regiones, se desliza una red a lo largo del mapa de características saliente de la última capa convolucional. Esta pequeña red toma como entrada una ventana espacial de  $n \times n$  ( $n = 3$ ) del mapa de características convolucionales, cada ventana que se desliza, es “mapeada” a una característica de menor dimensión, en el caso de este proyecto; 512-d<sup>27</sup>. Posteriormente, esta característica se introduce en dos capas completamente conectadas, “*fully connected layers*”, la *box-regression layer* (*reg*) y la *box-classification layer* (*cls*). En resumen, esta arquitectura está formada por una **VGG-16**, y por  $3 \times 3$  capas convolucionales seguidas de las dos capas mencionadas antes (*reg* y *cls*) que aplican una convolución de  $(1 \times 1)$ . Cabe destacar que estas dos últimas capas aplican un *padding* de 1.

<sup>26</sup>Como ya se mencionó al inicio de esta memoria, en los casos de tener más de un color  $\mathbf{A}$  representa un tensor de tres dimensiones.

<sup>27</sup>Esto quiere decir que habrá 512 filtros.

### 3.1.2. Anchors

El término *anchor* (en castellano: ancla) se define en [Mat], como “*un conjunto de cuadros delimitadores predefinidos con una determinada altura y anchura. Estos recuadros se definen para capturar la escala y la relación de aspecto de las clases de objetos específicos que se desea detectar. Durante la detección, los anchors predefinidos se colocan en mosaico a lo largo de la imagen. Luego, la red predice la probabilidad y otros atributos, como el fondo, la intersección sobre la unión (IoU) y los desplazamientos para cada anchor en el mosaico.*”

Tal y como se explica en [RHGS15], los *anchors* están relacionados con las ventanas deslizantes, pues en cada posición de la ventana móvil, se predicen simultáneamente múltiples propuestas de regiones, el número máximo de propuestas para cada lugar es denotado con la letra  $k$ . La capa *reg* tiene  $k$  salidas que codifican las coordenadas de  $k$  casillas, y la capa *cls* emite  $2k$  puntuaciones que estiman la probabilidad de que haya o no un defecto para cada propuesta. Las  $k$  propuestas se parametrizan en relación con  $k$  cajas de referencia, que reciben el nombre de “*anchors*”. Los *anchors* están centrados en la ventana deslizante y se asocian a una escala y una relación de aspecto. Por defecto, en este proyecto se usan 3 escalas y 3 relaciones de aspecto, lo que resulta en  $k = 9$  *anchors* en cada posición deslizante.

El concepto de *anchors* puede entenderse mejor con las figuras 12 y 13.

Habiendo explicado el concepto de propuesta de regiones y la idea detrás de los *anchors*, en la siguiente sección se explica la dinámica de la **RPN**.

### 3.1.3. Dinámica general dentro de la RPN

Al entrar a la red, la imagen de  $572 \times 768$  píxeles, entra en el *backbone*, que como se ha dicho antes, es la **VGG-16**. Esta imagen entrante, es re-escalada a un tamaño de  $300 \times 400$  píxeles. A la salida de la **VGG-16**, se haya el mapa de características de la imagen. El tamaño de este mapa de características viene determinado por el valor del *stride*, y al tratarse de la **VGG-16**, el *stride* será **16**, por lo tanto, si la imagen re-escalada es de  $300 \times 400$  píxeles, el mapa de características tendrá una resolución de  $\lfloor \frac{300}{16} \rfloor \times \lfloor \frac{400}{16} \rfloor \equiv 18 \times 25$  píxeles. Véase la imagen 11.

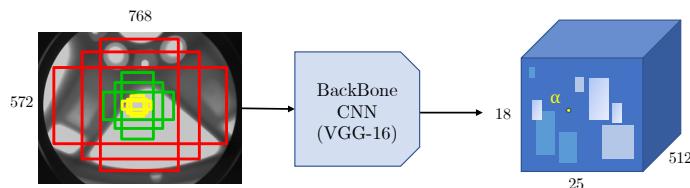


Figura 11: *Anchors* y salida de la **VGG-16**. Figura creada a partir de una ilustración en [Ana]

Para cada punto del mapa de características de salida, la red tiene que aprender si hay un defecto presente en la imagen de entrada en su ubicación correspondiente, y estimar su tamaño. Para ello, la **VGG-16** coloca una serie de *anchors* en la imagen de entrada para cada punto del mapa de características de salida. Los *anchors* indican posibles objetos de distintos tamaños y relaciones de aspecto<sup>28</sup> en esa ubicación (ver la figura 12).

<sup>28</sup>El término relación de aspecto, del inglés, *aspect ratio* queda definido como la proporción que hay entre la altura y la anchura de una imagen.

Como para cada posición deslizante hay  $k = 9$  *anchors*, el número total de *anchors* vendrá dado por la siguiente expresión:  $18 \times 25 \times 9$ , dando como resultado, 4050 posibles *anchors* (ver la figura 13). Las tres escalas de *anchors* y la relación de aspectos, que se han contemplado, son las siguientes:

$$\begin{aligned}\text{anchor\_box\_scales} &\in [[8,16,32], [16,32,64], [32,64,128]] \\ \text{anchor\_box\_ratios} &= [[1,1], [\frac{1}{\sqrt{2}}, \frac{2}{\sqrt{2}}], [\frac{2}{\sqrt{2}}, \frac{1}{\sqrt{2}}]]\end{aligned}$$

Ahora bien, a medida que la red se desplaza por cada píxel del mapa de características de salida, tiene que comprobar si los  $k$  *anchors* correspondientes que abarcan la imagen de entrada, realmente contienen defectos, y así perfeccionar las coordenadas de estos *anchors* para obtener *bounding boxes* como “propuestas de objetos” o regiones de interés. Shaoqing Ren et al. [RHGS15]

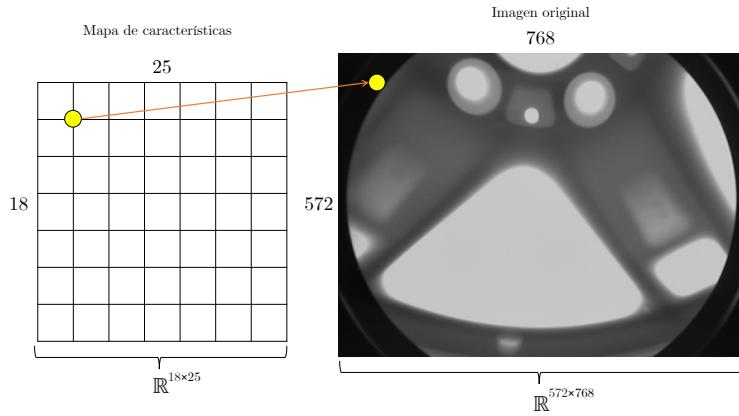


Figura 12: Proyección de un *anchor* del mapa de características en la imagen original.

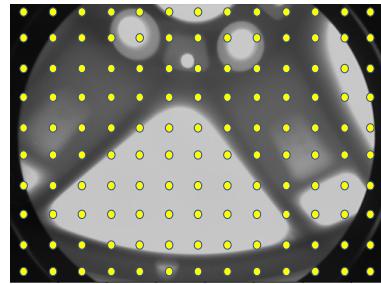


Figura 13: Imagen original con sus *anchors* correspondientes.

El estado inicial de cada *anchor* es “negativo”, y en el código, para que un *anchor* pase a ser “positivo”, ha de cumplir que  $\text{IoU} > 0,7$ . En el caso de que  $0,3 < \text{IoU} < 0,7$ , el *anchor* no se toma en cuenta. Las siglas IoU, hacen referencia al término, *Intersection Over Union*, que en castellano significa, intersección sobre la unión, y se calcula dividiendo la intersección entre *anchor* y *bounding box* del *groundtruth* por la unión de ambas. Hasty et al. [Has].

Suponiendo una imagen de la forma  $\mathbf{X} \in \mathbb{R}^{n \times m}$  y si  $\exists \mathbf{A}, \mathbf{B} \mid \mathbf{A} \subseteq \mathbf{X}$  y  $\mathbf{B} \subseteq \mathbf{X}$  entonces la IoU se define mediante el índice de Jaccard

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

En el caso de esta tesis,  $\mathbf{A}$  hace referencia a un *anchor*  $k$ , y  $\mathbf{B}$  representa el *bounding box* correspondiente. Esta operación permite estimar cuánto coinciden los *anchors* con los *bounding boxes* del *groundtruth*. Además, hay que tener en cuenta que la **RPN** tiene muchas más regiones “negativas” que “positivas”, por lo que algunas de las regiones negativas se desactivan y tanto el número máximo de regiones positivas como el de regiones negativas se limitan a 256. Como es lógico, el algoritmo premia a los *anchors* positivos, pues esta condición prueba una mayor semejanza entre el *anchor* (predicción) y el *bounding box* (dato real). En el código, `y_is_box_valid` representa si el *anchor* tiene un defecto, e `y_rpn_overlap` representa si dicho *anchor* se solapa con el *bounding box* del *groundtruth*. La clasificación según el resultado de la IoU, es la siguiente:

Tipos	<code>y_is_box_valid</code>	<code>y_rpn_overlap</code>
“positivos”	1	1
“neutrales”	0	0
“negativos”	1	0

Cuadro 2: Clasificación de *anchors* según su IoU con los *bounding boxes*

Volviendo a la arquitectura de la **RPN**, es importante mencionar como a la salida de las 18 unidades de la rama de clasificación (*cls*), el tamaño es de  $\mathbb{R}^{H \times W \times 18}$ <sup>29</sup>. Posteriormente, esta información a la salida, se utiliza para asignar probabilidades dependiendo de si cada punto del mapa de características de la **VGG-16** (de tamaño:  $\mathbb{R}^{H \times W}$ ) contiene o no un objeto dentro de las 9 anclas de ese punto.

No obstante, las 36 unidades de la rama de regresión (*reg*) dan una salida de tamaño  $\mathbb{R}^{H \times W \times 36}$ . Esta salida se utiliza para obtener los 4 coeficientes de regresión de cada uno de los 9 *anchors* para cada punto del mapa de características de la **VGG-16** (de tamaño:  $\mathbb{R}^{H \times W}$ ). Estos coeficientes de regresión se utilizan para “pulir” las coordenadas correspondientes a los *anchors* con defectos.

### 3.1.4. Region of Interest

Una vez se ha realizado la propuesta de regiones y ya se tienen sus *bounding boxes* correspondientes, la siguiente tarea consiste en agrupar las características del mapa generado tras la **VGG-16** a partir de esos *bounding boxes*. De esta tarea se encarga la capa *ROI Pooling* (ver figura 10), y esta, sigue la siguiente dinámica:

- Esta capa toma la región correspondiente a una propuesta del mapa de características.
- Luego, divide esa región en un número fijo de sub-ventanas.
- Por último, aplica un *max pooling operation*, que consiste en una operación de agrupación, que calcula el valor máximo en cada subventana, con el fin de obtener una salida de tamaño fijo.

---

<sup>29</sup>Las letras  $H$  y  $W$  representan el alto y el ancho de la salida respectivamente.

Básicamente, la capa divide las características de las ventanas de propuesta seleccionadas que proceden de la **RPN** en subventanas de tamaño  $h/H \times w/W$  y realiza una operación de *pooling* en cada una de estas subventanas. Esto da lugar a características de salida de tamaño fijo ( $H \times W$ ) independientemente del tamaño de entrada.

Los parámetros  $H$  y  $W$  se eligen de forma que la salida sea compatible con la primera capa totalmente conectada de la red. Los valores elegidos de  $H$  y  $W$  en la *Fast R-CNN* es 7, pues en [RHGS15] se obtienen buenos resultados para este valor. Al igual que el *pooling* normal, el *pooling ROI* se lleva a cabo en cada canal individualmente.

Las características de salida de la capa *ROI Pooling* ( $N \times 7 \times 7 \times 512$ , donde  $N$  es el número de propuestas) se introducen en las *fully connected layers* y en las ramas *softmax* y *BB-regression*. La rama de clasificación *softmax* implementa la función *softmax* que es una función exponencial normalizada. Esta función toma como entrada un vector  $\mathbf{z}$  de  $K$  números reales, y lo normaliza en una distribución de probabilidad consistente en  $K$  probabilidades proporcionales a las exponenciales de los números de entrada. Wikipedia et al. [Wik23].

La función *softmax* estándar  $\sigma : \mathbb{R}^K \rightarrow (0, 1)^K$  está definida  $\forall K \geq 1$  por la siguiente fórmula

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

En definitiva, esta función aplica la exponencial a cada elemento  $z$  del vector  $\mathbf{z}$  y lo normaliza por la suma de todas las exponenciales, produciendo valores de probabilidad para cada ROI perteneciente a  $K$  categorías y a una categoría de fondo general<sup>30</sup>.

Posteriormente, la rama de regresión (*BB-regression*), como su nombre indica, aplica una regresión, para hacer que los *bounding boxes* de este algoritmo de propuesta de regiones sean más precisos.

### 3.2. Base de datos

Una vez explicada la arquitectura del proyecto, en esta sección se muestran las imágenes que se han utilizado para entrenar esta red. Como se persigue una correcta detección de defectos en radiografías de piezas industriales, las imágenes que se van a utilizar se corresponden con imágenes tomadas con rayos X de distintos tipos de piezas de fundición.

Estas imágenes se organizan en **GDXray**, una base de datos pública que contiene desde imágenes de piezas de soldaduras, hasta imágenes de equipajes tomadas en aeropuertos. Todas estas imágenes y muchas más pueden encontrarse en **GDXRAY+**.

#### 3.2.1. Directorio **GDXray**

Mery et al. [MRZ<sup>+</sup>15] han recabado todas las imágenes que pueden encontrarse en **GDXray**, con el fin de poner a prueba algoritmos de visión por computador y para que por primera vez haya una base de datos pública de imágenes tomadas con rayos X. Esta base de datos contiene un total de 19407 imágenes, pero para este TFG solo se han utilizado las imágenes dentro del directorio *Castings*<sup>31</sup>. Las imágenes contenidas en dicho directorio, muestran piezas de soldadura que en su mayoría tienen defectos, aunque no todas.

---

<sup>30</sup>El término fondo general en este contexto, hace referencia a un espacio que no corresponde con un defecto.

<sup>31</sup>La tabla 3 ilustra la estructura del directorio.

Base de datos	Grupo	Serie	Imágenes de rayos-X
GDXray	→ <i>Castings</i>	→ C0001 → C0001_0001.png ... C0001_00072.png	
		⋮ ⋮	
		C0067	C0067_0001.png ... C0067_00083.png

Cuadro 3: Organización de directorios.

En cada una de las Series, dentro del directorio *Castings*, se encuentran las imágenes correspondientes a un tipo de pieza industrial, con un archivo *groundtruth*. Este archivo contiene los títulos de las imágenes que se encuentran dentro de la carpeta “Series” con las coordenadas de los *bounding boxes* encargados de recuadrar los defectos que hay en la imagen (en el caso de haberlos). En el caso de que una imagen no tuviese defectos, entonces esta no aparecería listada en el archivo. El formato del *groundtruth* para la serie **C0001** puede verse en el cuadro 4.

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

Cuadro 4: Formato del *groundtruth* de la serie **C0001**.

Algunas de las imágenes que pueden encontrarse en el directorio *Castings* son:

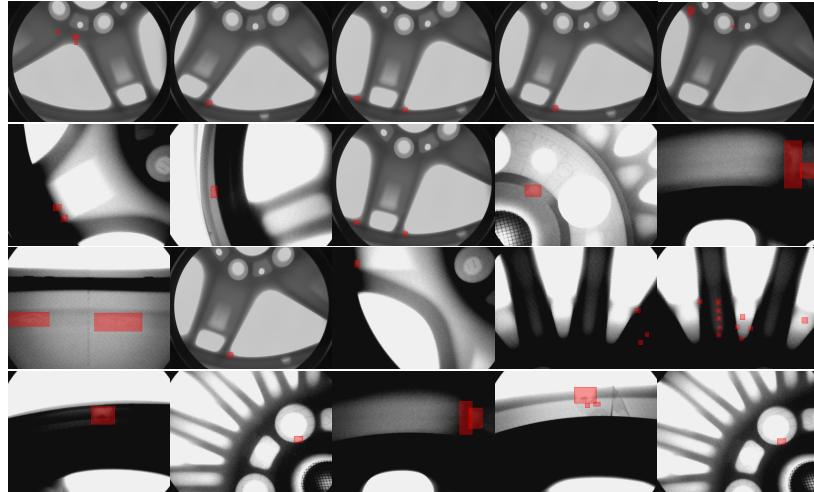


Figura 14: Algunas imágenes del directorio *Castings* con sus *bounding boxes* correspondientes

Observando las imágenes de la figura 14, uno puede observar como todas las imágenes comparten las mismas dimensiones ( $\mathbb{R}^{572 \times 768}$ ), aunque no todas las imágenes dentro del directorio *Castings* tienen el mismo tamaño. Como se mencionó en la sección 3.1.1, la red acepta cualquier dimensión de imagen, ya que dentro del código todas las imágenes entrantes se re-escalán a un mismo tamaño.

## 4. Modelado del detector

Recapitulando toda la teoría y la técnica que se ha visto hasta ahora, se infiere lo siguiente:

- No es lo mismo detectar objetos, que detectar defectos pequeños. El porque de esto reside en la proporción de objeto o de defecto respecto al fondo, y la elección de la arquitectura de red parte de esta premisa.
- Un buen entrenamiento para acometer esta tarea, requiere de una arquitectura adecuada, y no muy profunda, ya que una mayor profundidad implica un mayor número de parámetros y capacidad descriptiva, y esto aumenta la tendencia a un sobre-ajuste. Por lo tanto, al no necesitar tanta capacidad descriptiva, se reduce la profundidad para evitar el sobre-ajuste.
- Detectar defectos requiere de un *backbone* principal, pero también requiere de algoritmos de propuestas de regiones.
- La relación entre la imagen de entrada y el mapa de características codificado por el *backbone* es importante, pues el mecanismo de detección depende en su totalidad de esta correspondencia.

En esta sección no se pretende explicar el código implementado paso a paso, sino más bien, explicar brevemente el recorrido de la imagen desde que entra a la red, hasta que sale del algoritmo de propuestas de regiones, de esta manera, se comprenderá mejor la estructura del código.

Para implementar la arquitectura estudiada en la sección 3, se ha partido de un código, que ha permitido comprender mejor todas las capas por las que pasa la imagen, así como el proceso de entrenamiento, además de ayudar a afianzar los conceptos relativos al *stride*, el *padding* y los *anchors*. Este código puede encontrarse en el siguiente repositorio de GitHub: [Faster R-CNN for Open Images Dataset by Keras](#).

Es importante destacar que este código ha sido desarrollado para entrenar una red neuronal con el fin de detectar, personas, coches, y teléfonos móviles. Aunque el autor, posteriormente crea más clases, como; “tijeras”, “Apple pencil”, “mono durmiente” y “cacao”. El autor de este código modificado (RockyXu66) se basó en un código del usuario de GitHub, Yhenon (ver este enlace: [Yhenon’s repository](#)).

Los resultados obtenidos son buenos, con probabilidades finales de entre el 75 % y el 99 %, pero como se ha mencionado anteriormente, este tipo de arquitecturas trabajan bien reconociendo objetos de grandes proporciones. A lo largo de esta sección, se verá como surgen problemas de cara al entrenamiento de la red.

### 4.1. Recogida de datos

Para que la red pueda ir cogiendo imágenes, se le ha de pasar una estructura de datos que contenga el directorio donde se encuentra la imagen, e información relativa a la misma, como su ancho (w), su alto (h) y las coordenadas de los *bounding boxes* que recuadren los defectos que pueda haber en ella. Para acometer esta tarea, se ha implementado un algoritmo de búsqueda, que va directorio por directorio memorizando todas las imágenes contenidas en él así como su *groundtruth*. Aunque pueda parecer tarea fácil, no lo es, ya que aunque se sepa programar en Python, uno puede desconocer muchas de las funciones que ayudan a implementar una búsqueda iterativa. A esto se le añade la dificultad de que en el *groundtruth* no aparece el nombre de las imágenes, sino el número,

por lo que se ha tenido que extraer de los títulos de las imágenes, los números asociados, que son los mismos que aparecen en el *groundtruth*.

El algoritmo que he implementado, puede entenderse mejor con el siguiente pseudocódigo;

---

#### **Algoritmo 1** Recopilación de datos

---

```

procedure BÚSQUEDA ITERATIVA
    dir = /Usuario/Documentos/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

```

---

La siguiente parte ha consistido en dibujar los *bounding boxes* en las imágenes originales. El código correspondiente a esta tarea, puede verse en el repositorio del proyecto, cuyo enlace está en la sección 8.

Para dibujar los *bounding boxes* se ha implementado un código que recorre todos los directorios y sus correspondientes *groundtruths*, para leer las coordenadas de cada defecto correspondiente a cada una de las imágenes. Posteriormente, todas esas coordenadas se guardan en una lista que luego es transformada en un tensor para poder pasársela a la función encargada de pintar los *bounding boxes* esas coordenadas, y la imagen dónde ha de pintarlas. Con dicha implementación, se ha conseguido representar las imágenes vistas anteriormente en la figura 14.

Una vez que fueron representados los *bounding boxes* en la imagen original, es momento de crear la estructura que alberga los datos correspondientes a cada una de las imágenes que se introducirán en la red. Para acometer dicha tarea, se ha creado una lista de diccionarios, donde se recoge el nombre de la imagen, su alto, su ancho, y coordenadas de los *bounding boxes* correspondientes a los defectos de la imagen, así como la clase del defecto, que resulta ser la misma para todos los casos.

Cabe destacar, como el diccionario que alberga toda esta información para cada una de las imágenes del directorio *Castings* es muy grande, pues en total hay 2367 defectos entre todas las imágenes. Esto supone un inconveniente, porque al contener tantas imágenes, resulta imposible cargar toda esa información de una sola vez, porque el ordenador no tiene suficiente memoria, la solución a este

inconveniente, viene dada por los generadores de datos.

Los generadores de datos permiten iterar a lo largo de un conjunto muestral grande, en este caso, se itera sobre el diccionario que contiene la ruta de cada una de las imágenes y su información correspondiente. Al igual que una función habitual puede devolver parámetros tras su ejecución, cuando uno implementa un generador de datos, este retorna los valores dentro de una **función** cualquiera con un **yield(x)**, tras llamar a esa **función**, en la siguiente linea se hace uso del **next(x)** para que se continue iterando, véase el ejemplo de abajo:

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

## 4.2. Procesado previo

Una vez la red es capaz de coger las imágenes y su correspondiente información del diccionario, se procede con el procesado de la información.

Las imágenes que aparecen en el diccionario, están ordenadas tal y como aparece en el directorio **Castings**, por lo tanto, para evitar que el entrenamiento de la red pueda verse afectado por el orden de las imágenes, se introduce una semilla aleatoria para que la red vaya cogiendo las imágenes fuera de orden.

Cuando la red coge una imagen, a esta imagen se le aplican distintas transformaciones, como por ejemplo, rotaciones, pero también se le puede dar la vuelta por arriba o por abajo. El objetivo de estas transformaciones es el de aumentar el conjunto muestral, pues aunque el directorio **Castings** contiene 2727 imágenes, estas podrían no ser suficientes para entrenar la red, por lo tanto, se aplican las modificaciones mencionadas antes, a las imágenes, a fin de “enriquecer” el conjunto muestral. Como consecuencia de estas transformaciones, en [Tea] se observa un aumento del rendimiento en algunos modelos.

No obstante, al modificar una imagen, las coordenadas de los *bounding boxes* del *groundtruth* ya no coinciden con los defectos en la imagen modificada, por lo que en este apartado también se calculan las nuevas coordenadas para luego volcarlas en el diccionario.

Una vez la imagen se ha rotado o dado la vuelta, esta es re-escalada a un tamaño de  $\mathbb{R}^{300 \times 400}$  con el fin de que el entrenamiento del modelo sea más rápido y así ahorrar tiempo.

## 4.3. Cálculo de regiones

A estas alturas, la red ya ha aplicado las transformaciones necesarias y ha re-escalado la imagen para adaptarla al entrenamiento. A partir de estos nuevos datos, se calculan las regiones de la imagen.

Como ya se mencionó en la sección 3.1.3, habrá  $450 \times 9 = 4050$  *anchors* si la geometría del mapa de características es  $\mathbb{R}^{18 \times 25}$  y si cada *anchor* tiene  $3 \times 3 = 9$  *boxes* correspondientes en la imagen original (véase la figura 11).

Inicialmente, todos los *anchors* son categorizados como “negativos” pero dependiendo del resultado de la **IoU** entre los *anchors* y los *bounding boxes* estos pueden tornarse “positivos”. Esta

información es recogida en los valores que la función encargada de proponer regiones devuelve, i.e., `calc.rpn`. (ver cuadro 2)

Los valores que retorna la red son los siguientes:

- `y_rpn_cls = lista(num_bboxes, y_is_box_valid + y_rpn_overlap)`
- `y_rpn_regr = 0 o 1 (0: el 'box' no es válido y 1: el 'box' es válido)`
- `y_rpn_overlap = 0 o 1 (0: el 'box' no contiene defecto y 1: el 'box' contiene defecto)`
- `y_rpn_regr = lista(num_bboxes, 4*y_rpn_overlap + y_rpn_regr)`
- `y_rpn_regr = x1,y1,x2,y2 (coordenadas de los bounding boxes)`

A continuación, se enuncian las dimensiones correspondientes a las estructuras de datos que se muestran en la lista de arriba:

- `y_rpn_cls` es un tensor de la forma  $y_rpn\_cls \in \mathbb{R}^{1 \times 18 \times 25 \times 18}$ .<sup>32</sup> En naranja se destaca la dimensión del mapa de características, y en cian se representa el producto de  $9 \times 2$ , pues cada punto en el mapa de características tiene 9 anchors, y cada anchor tiene 2 valores, uno para `y_is_box_valid` y otro para `y_rpn_overlap` respectivamente.
- `y_rpn_regr` también es un tensor, pero con las siguientes dimensiones  $y_rpn\_regr \in \mathbb{R}^{1 \times 18 \times 25 \times 72}$ . En este caso, en cian se representa el producto entre los 9 anchors por los 4 valores para `tx`, `ty`, `tw` y `th`, que tiene cada anchor, así mismo, esto hay que multiplicarlo por los dos valores posibles que pueden tener `tx`, `ty`, `tw` y `th`, i.e., `y_is_box_valid` y `y_rpn_overlap`. De este producto se obtienen 72 posibles valores ( $9 \times 4 \times 2$ ).

Ya se tiene la propuesta de regiones, ahora, para comprobar el anchor final que la red ha tomado, se representa en las figuras 15 y 16, un conjunto de figuras donde en todas ellas se representa la misma imagen, pero para diferentes escalas de anchors (ver el segundo párrafo de la sección 3.1.3)

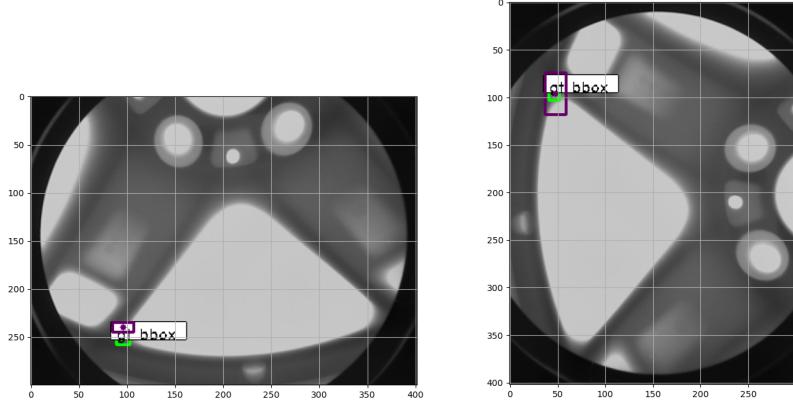
Como se puede observar en las figuras 15 y 16 según se aumenta el tamaño de la escala de los anchors, el espacio que cubre es mayor y esto no es bueno, ya que decrementa el resultado de la IoU, pues el defecto es muy pequeño.

Además, si el tamaño del anchor respecto al tamaño del bounding box es muy pequeño, entonces el resultado de la IoU será bajo, y esto tampoco es bueno de cara al entrenamiento, ya que a pesar de que el anchor cubra un defecto, este podría quedar descartado debido a su baja calificación tras efectuar la intersección sobre la unión. Algunas medidas de la IoU para las imágenes 15 y 16, pueden verse en el cuadro 5.

Como se puede observar en el cuadro 5, según aumentan las escalas, el valor máximo de la intersección sobre la unión se ve reducido, pero también cabe mencionar que para escalas mayores, hay menos resultados de IoU que sean 0.0, mientras que para las escalas más pequeñas, el número de resultados de IoU que resultan ser cero son considerablemente más que aquellos distintos de cero. Esto es lógico, ya que para escalas de los anchors más pequeñas, la IoU será mayor que para escalas mayores, porque la mayoría del anchor se solaparía con gran parte del bounding box.

---

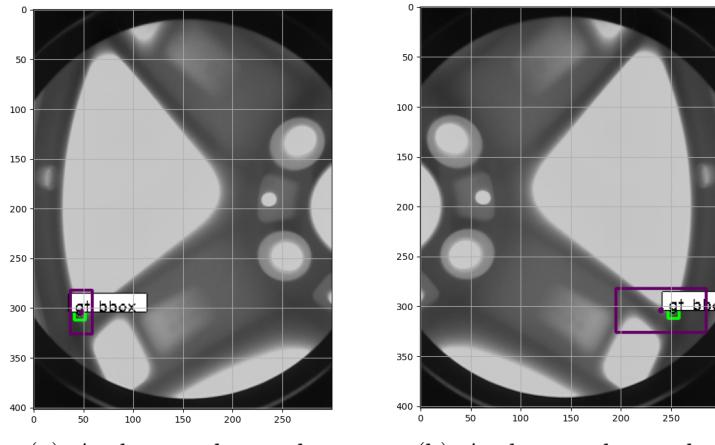
<sup>32</sup>La primera dimensión del tensor  $y_rpn\_cls \in \mathbb{R}^{1 \times 18 \times 25 \times 18}$  hace referencia al batch. El batch es el número de imágenes que se le pasa al algoritmo en cada iteración del entrenamiento, como en este caso es 1, ello significa que en cada iteración a la red se le pasa una sola imagen.



(a) *Anchor* resultante de escala  $\{8, 16, 32\}$

(b) *Anchor* resultante de escala  $\{16, 32, 64\}$

Figura 15: Resultados obtenidos tras la propuesta de regiones, los recuadros en verde representan los *bounding boxes* del *groundtruth* y en morado los *anchors*.



(a) *Anchor* resultante de escala  $\{32, 64, 128\}$

(b) *Anchor* resultante de escala  $\{64, 128, 256\}$

Figura 16: Resultados obtenidos tras la propuesta de regiones, los recuadros en verde representan los *bounding boxes* del *groundtruth* y en morado los *anchors*.

Escalas	$\text{IoU}_{min}$	$\text{IoU}_{max}$
$\{8, 16, 32\}$	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

Cuadro 5: Clasificación de *anchors* según su IoU con los *bounding boxes*

Es importante destacar otro aspecto fundamental, que es la posición del centro del *anchor* respecto al *bounding box* en la imagen. Si bien la premisa del párrafo anterior a la figura 15 es correcta, no se puede dar por hecho que el centro del *anchor* calculado en la propuesta de regiones siempre vaya a caer en el centro del *bounding box* correspondiente. Ahora hay que entender porque

no siempre coinciden los centros de los *anchors* y el de los *bounding boxes*, para esto, si uno revisa la sección 3.1.3 y observa de nuevo las imágenes 12 y 13, puede notar como la posición de los *anchors* está directamente ligada con la resolución de la imagen, por lo tanto matemáticamente se tendría la siguiente relación de probabilidades.

Si  $\mathbf{X}$  es una imagen cualquiera.

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

Y el centroide<sup>33</sup> de cualquier *bounding box* es  $\bar{\beta}$  y el centroide de cualquier *anchor* es  $\bar{\alpha}$ , entonces:

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

Con la relación 4.3 se quiere ilustrar mas intuitivamente el concepto de que para una imagen  $\mathbf{X}$  de mayor resolución (i.e.,  $n \times m$ ), la probabilidad de que  $\bar{\alpha}$  y  $\bar{\beta}$  sean iguales es mayor, que para una imagen  $\mathbf{X}$  de menor resolución (i.e.,  $\nu \times \mu$ ) simplemente porque la cantidad de *anchors* en una imagen de dimensiones  $n \times m$  es mayor, y por ello, la probabilidad de que los dos centroides coincidan es mayor.

También hay que tener en cuenta, que el defecto puede caer directamente debajo de la posición de un *anchor*, en este caso la **IoU** será mayor que si el defecto no coincidiese con el *anchor*. En la figura 15 para una escala de *anchors* de  $\{8, 16, 32\}$ , puede observarse como el *anchor* generado, no coincide al completo con el *bounding box* del *groundtruth*.

#### 4.4. De propuestas de regiones a ROIs

Una vez la **Fast R-CNN** (ver figura 10) ha propuesto las regiones correspondientes y ya ha generado el conjunto de “cajas” que resaltan dichas regiones, se ha de transformar estas coordenadas que se encuentran en formato **rpn**, en coordenadas que se ajusten a la imagen modificada de tamaño  $300 \times 400$ .

Aunque conceptualmente ya se ha explicado esta capa en la sección 3.1.4, a continuación se va a poner el foco en las operaciones y se explicará a alto nivel los pasos seguidos en esta etapa.

La función **rpn\_to\_roi** es la que se encarga de transformar aquello que la red ha generado, en regiones de interés que puedan representarse a modo de *bounding box* en la imagen de  $300 \times 400$ . Esta función recibe como argumentos los siguientes parámetros:

- **rpn\_layer** denota la predicción realizada por el modelo de red, de las regiones calculadas por la función **calc\_rpn** explicada en la sección anterior (4.3). Las dimensiones de este argumento son las mismas que **y\_rpn\_cls**, que ya se ha visto en la sección anterior (4.3). Por lo tanto, el modelo predictor no altera ni debe alterar las dimensiones de la variable que retorna **calc\_rpn**.
- **regr\_layer** denota la salida tras aplicar una regresión de los recuadros que destacan las regiones. Al igual que **rpn\_layer**, las dimensiones de este argumento no se ven alteradas por el modelo predictor, y estas pueden verse en la sección 4.3.

---

<sup>33</sup>El centroide de un objeto es el centro geométrico del subespacio formado por el objeto, e.g., el centroide de un rectángulo de dimensiones  $b \times h$  tendrá su centroide en  $\frac{b}{2}$  o  $\frac{h}{2}$  dependiendo si este es calculado respecto al eje x o eje y, aunque en los dos casos si se traza una linea recta, siempre se pasa por el centro, por ello la referencia no es un problema.

- `max_boxes` especifica el número máximo de *bounding boxes* que se dibujarán. No obstante, este dato se usará en la etapa de cribado de *bounding boxes* al final de esta función.
- `overlap_thresh` determina el solapamiento mínimo de un *bounding box* para que este sea contemplado y posteriormente representado. En este caso, al ser los defectos muy pequeños, este límite será pequeño,  $\approx (0,2 - 0,4)$ .

Para poder transformar esas coordenadas en elementos representables de la imagen, se ha de iterar a lo largo de todos los tamaños de *anchors* (i.e., 9) y luego, para cada *anchor* hay que contemplar cada *aspect ratio*<sup>34</sup>. El propósito de esta transformación es importante, pues se pretende obtener una lista de *bounding boxes*, donde cada *bounding box* es un vector de 4 coordenadas. No obstante, no se tiene una lista, sino 2 matrices de varias dimensiones, de manera que la posición de cada elemento, su coordenada en cada dimensión determina el *anchor* al que se refiere mediante la información de su escala, *aspect ratio* y la posición de su centro.

Cada *anchor* es re-escalado contemplando su tamaño, los *aspect ratios* correspondientes y el *stride* de 16<sup>35</sup>. Posteriormente, la dimensión de `regr_layer` pasa de  $\mathbb{R}^{1 \times 18 \times 25 \times 18} \rightarrow \mathbb{R}^{4 \times 18 \times 25}$ , quedándonos solo con el mapa de características y manteniendo los 4 valores de cada *anchor* (i.e., `tx`, `ty`, `tw` y `th`). Hecho esto, se crea una malla del tamaño del mapa de características, y a partir de ella se calculan la posición y tamaño de cada uno de los *anchors* (i.e. `x`, `y`, `w` y `h`). Por último y para afinar los valores de la posición y el tamaño, se aplica una regresión.

Además, en el código se evita que cualquier *anchor* sobrepase el alto y ancho. Seguidamente, se procede con la traducción de coordenadas, de  $(x, y, w, h)$  a  $(x_1, y_1, x_2, y_2)$ <sup>36</sup>. Por último, y siguiendo la dinámica al principio de este párrafo, se evita el dibujo de *bounding boxes* fuera del mapa de características.

Una vez se tienen los *bounding boxes* en una lista, se aplica la función `non_max_suppression_fast` que se encarga de cribar esa lista de *bounding boxes* con el fin de mantener aquellos recuadros que cumplen con los parámetros de `max_boxes` y `overlap_thresh`, este tamizado es implementado de la siguiente manera;

1. Ordenamiento de la lista de probabilidades.<sup>37</sup>
2. Búsqueda de la probabilidad más grande de la lista (la última) y guardarla en la lista de selección.
3. Cálculo de la **IoU** con la última “caja” y el resto de “cajas” de la lista. Si la **IoU** es mayor que el umbral de solapamiento (i.e., `overlap_thresh`), entonces esta casilla es eliminada de la lista.
4. Repetición de los pasos 2 y 3, a fin de que no haya ningún elemento en la lista de probabilidades.

A la salida de esta función, se obtiene la lista final de *bounding boxes*, con la lista de probabilidades asociadas a cada uno de los *bounding boxes*. A partir de estos datos, uno ya puede representar la imagen transformada, con los *bounding boxes* encima, esto es lo que se hace en la sección 6.

---

<sup>34</sup>Los *aspect ratios* vistos en la sección 3.1.3 (i.e., `anchor_box_ratios` =  $[[1, 1], [\frac{1}{\sqrt{2}}, \frac{2}{\sqrt{2}}], [\frac{2}{\sqrt{2}}, \frac{1}{\sqrt{2}}]]$ ).

<sup>35</sup>Se recuerda que en este proyecto, el *stride* es 16. No obstante, que el nombre **VGG-16** lleve el número 16 es coincidencia, pues el 16 del nombre del *backbone* determina el número de capas.

<sup>36</sup>Las coordenadas son transformadas al mismo formato de coordenadas que sigue el *groundtruth*.

<sup>37</sup>Hay que recordar que cada *bounding box* tiene una probabilidad asociada.

## 4.5. Capa final, la *ROI Pooling*

El cálculo de regiones y su traducción en datos representables ya está hecho, ahora falta preparar la red para el entrenamiento. Pero antes, queda describir la última capa del modelo, la **ROI Pooling layer** que ya se ha introducido en la sección 2.3, pero que se explica en esta sección.

Tal y como escriben Jifeng Dai et al. [DQX<sup>+</sup>17] la **ROI Pooling layer** es una capa que opera en un espacio de dos dimensiones, y que se utiliza en la mayoría de los métodos de propuesta de regiones basados en detección de objetos.

Esta capa realiza la labor de conversión de una región rectangular de entrada de tamaño arbitrario en características de tamaño fijo. En otras palabras, esta es la función que procesa los *rois* para obtener una salida de tamaño específico mediante una capa *max pooling*. Cada *roi* de entrada se divide en varias subceldas, a las que se les aplica una capa *max pooling*, donde el número de subceldas debe ser de la dimensión de la forma de salida. Yinghan Xu et al. [Xu]

Este proceso puede entenderse mejor con la siguiente figura.

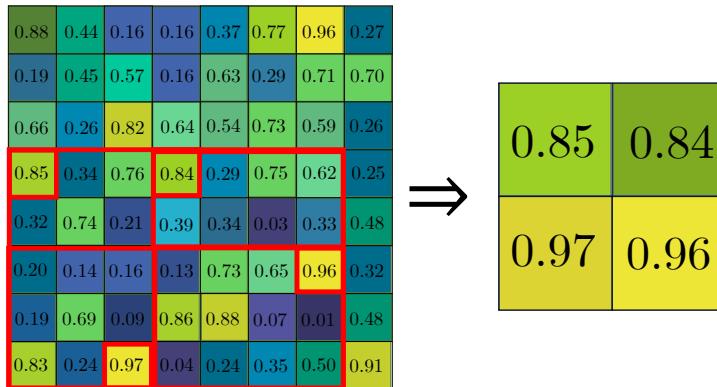


Figura 17: Ejemplo de la acción llevada a cabo por la capa **ROIConv**. Figura creada a partir de una ilustración en [Xu].

Tras esta capa, se encuentra una etapa de clasificación, que constituye el final de la arquitectura de la *Faster R-CNN*, y se encarga de predecir el nombre de la clase para cada *anchor* de entrada y la regresión de su “caja” correspondiente.

Como ya se mencionó en la sección 2.4, en este proyecto no hay clases de defectos, pero existen dos clases de detecciones que son, defecto y fondo. Esta función predice el nombre de una clase<sup>38</sup> y aplica otra regresión que refina las coordenadas de las “cajas”.

Dicho esto, los argumentos que recibe esta función son los siguientes:

- `base_layers` este parámetro denota el modelo de red neuronal, i.e., el *backbone* de la arquitectura, que como se sabe por la sección 2.4, se trata de la **VGG-16**.

<sup>38</sup>Esto se refiere a defecto o a fondo.

- **input\_rois** las dimensiones de este argumento son, `(1, num_rois, 4)` y contiene los *rois* calculados en la función anterior.
- **num\_rois** representa el número máximo de *rois* que la red es capaz de procesar en cada iteración. En este proyecto, este parámetro se fija a 4, a fin de no aumentar drásticamente la carga computacional a la hora de entrenar a la red y evitar que el proceso sea aún más lento.

A su salida, la red expedirá una lista con dos variables:

- **out\_class**. Este parámetro denota la salida de la capa de clasificación.
- **out\_regr**. Este otro parámetro denota la salida de la capa de regresión.

Como se vió en la sección 3.1.4, de la clasificación se ocupaba la rama que implementaba la función *softmax*, mientras que en la otra rama, se procedía con el cálculo de regresión de las “cajas” finales.

La red ya ha hecho su trabajo, el siguiente paso es entrenarla con el fin de que logre detectar esos pequeños defectos en imágenes, como los que se ven en la figura 14, aunque esto ya se discute en la sección 5.

## 5. Entrenamiento

El objetivo de esta sección no es mostrar el resultado de los distintos entrenamientos, puesto que esto ya se hace en la sección 6, sino más bien, explicar los parámetros que intervienen en dicho proceso y posteriormente explicar las métricas que hacen posible la representación de los resultados de manera gráfica y analítica.

Pero antes de describir ningún parámetro, hay que tener clara la idea que hay detrás del entrenamiento de una red neuronal. Según se escribe en [Xer], el entrenamiento de una red neuronal consiste en ajustar cada uno de los pesos<sup>39</sup> de las entradas de todas las neuronas que forman parte de la red neuronal, para que las respuestas de la capa de salida se ajusten lo más posible a los datos ya conocidos<sup>40</sup>.

Ahora bien, para poder entender que sucede en el entrenamiento, uno debe conocer los parámetros que lo hacen posible, así como las variables que influyen en el resultado del mismo.

### 5.1. Parámetros

Como se ha podido observar a lo largo de esta memoria, delegar en una inteligencia artificial la tarea de detección que ocupa a esta tesis, requiere la comprensión de muchos conceptos y herramientas. La etapa de entrenamiento de una red neuronal, no es una excepción, pues en ella intervienen multitud de métricas y conceptos, no obstante, en esta sección se pretende mencionar y explicar exclusivamente aquellos conceptos y parámetros que influyen en mayor medida en el proceso de entrenamiento.

Los parámetros más relevantes que influyen en el proceso de entrenamiento son los siguientes:

- **La profundidad.** Como es lógico, el tiempo de entrenamiento de una red profunda será mayor que el correspondiente a una red con pocas capas convolucionales.
- El **batch**, es un hiperparámetro que define el número de muestras con las que se va a trabajar antes de actualizar los parámetros internos del modelo. En otras palabras, y como ya se mencionó en una de las anotaciones de la sección 4.3, el *batch* en términos prácticos, es el número de imágenes que se le pasa al algoritmo en cada iteración del entrenamiento, como en este caso es 1, significa que en cada iteración a la red se le pasa una sola imagen.

Es importante mencionar, que si uno quiere reducir el tiempo de entrenamiento, lo razonable es pasarle a la red una imagen detrás de otra y no varias al mismo tiempo.

- El número de ***rois***, es el número de regiones de interés que la red tendrá en cuenta simultáneamente, como ya se mencionó, este parámetro se ha fijado a 4. Un mayor número de *rois* implica un entrenamiento más largo.
- El **tamaño** del conjunto muestral (i.e., las imágenes con las que se entrena la red). Como es lógico, si el conjunto muestral es grande ( $\sim 10000$  imágenes) entonces el entrenamiento tomará bastante tiempo, y si además se incrementa el número de *rois* y el *batch*, entonces un entrenamiento ininterrumpido, podría durar casi una semana.

---

<sup>39</sup>En el anexo A.1, puede encontrarse información acerca de la definición de los pesos y otros parámetros de las redes neuronales.

<sup>40</sup>En el caso de este proyecto, los datos que se conocen, son las coordenadas de los defectos contenidos en el *groundtruth* para cada una de las imágenes de entreno.

- El ***epoch***, según explica Jason Brownlee en [Bro], es un hiperparámetro que define el número de veces que el modelo a entrenar trabajará a través de todo el conjunto de datos de entrenamiento.

Un *epoch* significa que cada muestra del conjunto de datos de entrenamiento ha tenido la oportunidad de actualizar los parámetros internos del modelo. Además, este parámetro está compuesto por uno o más *batches*.

- La **resolución** de las imágenes entrantes. En el caso de que estas fuesen de alta resolución, su procesado a lo largo de las capas convolucionales ralentizaría la etapa de entrenamiento considerablemente, ya que, una mayor resolución implica más píxeles, y más píxeles implican más *anchors* que se traducen en más regiones de interés que calcular, aunque esto podría mejorar el entrenamiento a nivel de resultados, ya que una imagen de mayor resolución ayudaría a preservar en mayor medida el defecto ya de por si pequeño.
- El número **máximo** de *boxes*. Como se explicó en la sección 4.4 este parámetro determina el número máximo de “cajas” finales que se representarán, por lo tanto si este parámetro aumenta, se generarán más recuadros, lo que tomará más tiempo, ya que han de calcularse sus coordenadas y la probabilidad que lleva asociada.
- El ***overlap threshold*** establece el umbral de solapamiento, que las “cajas” han de cumplir para que sean tomadas en consideración.

## 5.2. Métricas

Conociendo los parámetros que caracterizan e influyen el proceso de entrenamiento, a continuación, se estudian las métricas que permiten juzgar si la red se ha entrenado correctamente. Aunque no sea la única métrica, la función de pérdidas del modelo posee un protagonismo considerable, pues determina si la red se ha entrenado razonablemente. No obstante, a partir de ella no se puede concluir si el resultado del entrenamiento es el correcto, por eso, este protagonismo es compartido con el parámetro **mAP** que se explica posteriormente.

Con el fin de entrenar el modelo RPN, a cada *anchor* se le asigna una etiqueta de clase binaria, que determina si es un defecto o no. Ahora bien, a un *anchor* se le puede asignar una etiqueta positiva si alguna de las dos condiciones se cumplen:

- El *anchor/anchors* con la mayor **IoU** se solapa con un *bounding box* del *groundtruth*.
- El *anchor* con una **IoU** superior a 0,7 se solapa con cualquier *bounding box* del *groundtruth*.

Tal y como se menciona en [RHGS15], un único *bounding box* puede asignar etiquetas positivas a varios *anchors*. Normalmente, la segunda condición es suficiente para determinar las muestras positivas, aunque se sigue contemplando la primera condición, por si acaso la segunda no encuentra ninguna muestra positiva. Se le asigna una etiqueta negativa a un *anchor* no positivo si su **IoU** es inferior a 0,3 para todos los *bounding boxes* del *groundtruth*. Ahora bien, en el caso de haber *anchors* que no son ni positivos ni negativos, estos no contribuyen al entrenamiento de la red y quedan descartados.

Dicho esto, la función objetivo que se minimiza en la **Fast R-CNN** de este proyecto para una imagen, viene definida en [RHGS15] como sigue:

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

Donde  $i$  denota un *anchor* dentro del *batch*  $\mathcal{B}$  (i.e., un *anchor* dentro de una imagen) y  $p_i$  es la probabilidad de que dicho *anchor* sea un objeto (i.e., `y_is_box_valid`). Luego,  $p_i^* = 1$  si el *anchor* es positivo y 0 si el *anchor* es negativo.

El parámetro  $t_i$  es un vector que contiene las 4 coordenadas del *bounding box* generado por la red, mientras que  $t_i^*$  contiene el mismo tipo de coordenadas, pero para el *bounding box* del *ground-truth*, asociado a un *anchor* positivo.

Como se sabe por las secciones 3.1.4 y 4.5 la **Faster R-CNN** retorna una lista que contiene la salida de la capa de clasificación y la salida de la capa de regresión, i.e., `out_class` y `out_regr` respectivamente. De ahí los parámetros  $L_{cls}$  y  $L_{reg}$ .

- El parámetro  $L_{cls}$  denota las pérdidas en escala logarítmica fruto de la comparativa “hay o no un defecto”.
- Mientras que  $L_{reg}$  representa las pérdidas asociadas a la regresión.

Por último, el término  $p_i^* L_{reg}$  activa las pérdidas por regresión solo cuando  $p_i^* = 1$  (i.e., solo para *anchors* positivos) ya que  $p_i^* \in \{0, 1\}$ . Los términos *cls* y *reg* mencionados en la sección 3.1.1 están representados por  $p_i$  y  $t_i$  respectivamente. Estos dos parámetros correspondientes a la salida de las capas de regresión y de clasificación, son normalizados por los términos  $N_{cls} = 2^{41}$  y  $N_{reg} = 4096^{42}$  respectivamente, posteriormente, estos son ponderados por el parámetro  $\lambda = 10$ .

En este proyecto se han aplicado dos funciones de pérdidas, una al modelo RPN y otra al modelo clasificador. Pero además, como la capa de propuesta de regiones expedía dos valores, el asociado a la clasificación de la detección<sup>43</sup>, y el correspondiente a la regresión de las coordenadas de los *bounding boxes*. Por esta razón, hay dos tipos de funciones de pérdidas, la correspondiente a la salida de la etapa de clasificación y la que va asociada a la etapa de regresión. Esto hace un total de cuatro funciones de pérdidas que se explican brevemente a continuación:

- `loss_rpn_cls`. Esta función se corresponde con las pérdidas del resultado de clasificación del modelo RPN.
- `loss_rpn_regr`. De manera similar a la anterior, esta también se corresponde con una salida de la RPN, pero esta vez, esta va asociada a la regresión de las coordenadas de los *bounding boxes*.
- `loss_class_cls`. Esta función y la siguiente, se corresponden con la salida del modelo clasificador, esta en concreto hace referencia a las pérdidas asociadas a la salida de la etapa clasificadora.
- `loss_class_regr`. A diferencia de la anterior, esta se corresponde con la salida de la etapa de regresión de los *bounding boxes*.
- `curr_loss`. Esta última función de pérdidas suma los valores de las cuatro funciones anteriores, y normalmente tiene una tendencia decreciente.

---

<sup>41</sup>Este parámetro indica el número de clases, en el caso de este proyecto  $N_{cls} = 2$ , pues se tiene, defecto y fondo.

<sup>42</sup>Se recuerda que 4096 era el número de *anchors* en la imagen.

<sup>43</sup>Si se trata de un defecto o no.

A parte de la función de pérdidas<sup>44</sup> explicada antes, hay otras métricas implementadas en este TFG, que también permiten analizar el resultado del entrenamiento. Estas son:

- **class\_acc.** Determina entre 0-1 la exactitud de que un recuadro contenga o no un defecto.
- El **mAP**. Este parámetro es descrito por Jonathan Hui en [Huia], como una métrica muy popular y comúnmente utilizada para medir la precisión de detectores de objetos como la **Faster R-CNN**. Este parámetro representa la media de la AP (*Average Precision*). En el contexto de esta tesis, la precisión mide la exactitud de las predicciones, es decir, el porcentaje de predicciones correctas. La precisión puede calcularse con la siguiente formula:

$$Precision = \frac{TP}{TP + FP} \text{ donde } \begin{cases} (TP) & : True Positive, \text{ positivos verdaderos} \\ (FP) & : False Positive, \text{ falsos positivos} \end{cases} \quad (7)$$

A continuación, se introduce el término *recall*, que mide lo bien que se encuentran todos los positivos. Este parámetro puede calcularse de la siguiente manera:

$$Recall = \frac{TP}{TP + FN} \text{ donde } \begin{cases} (TP) & : True Positive, \text{ positivos verdaderos} \\ (FN) & : False Negative, \text{ falsos negativos} \end{cases} \quad (8)$$

Es importante mencionar la estrecha relación que existe entre el **mAP** y la **IoU**, pues como se explicó en la sección 3.1.3, para cada *bounding box* se mide el solapamiento entre el *bounding box* predicho por la red, y el del *groundtruth*, mediante la **IoU**. Por lo tanto, para la tarea de detección de defectos de este proyecto, la precisión es calculada usando el valor de la **IoU** dado un umbral de solapamiento. Shivy Yohanandan et al. [Yoh].

La explicación anterior puede entenderse mejor con el siguiente ejemplo extraído de [Yoh]. Si se toma como ejemplo un umbral de solapamiento (i.e., umbral de la **IoU**) de 0,5 y una **IoU** que vale 0,7 para una detección en específico, entonces la predicción queda clasificada como *True Positive*<sup>45</sup> (TP). Ahora bien, si la **IoU** es 0,3, la detección es clasificada como *False Positive*<sup>46</sup>(FP)

Al poner un umbral de solapamiento, solo permanecen aquellas detecciones que pasen ese umbral. Si se decide aumentar este umbral, los falsos positivos (FP) disminuyen, mejorando así la precisión, pero empeorando el *recall*, pues se pueden perder detecciones. En el caso de disminuir el umbral, los falsos positivos aumentan y consecuentemente, la precisión disminuye.

Una vez explicadas las métricas y los parámetros más influyentes, en la siguiente sección se exponen los resultados relativos al entrenamiento, dónde además se justifican los valores obtenidos con las métricas explicadas en esta sección.

---

<sup>44</sup>En el anexo A.3 se profundiza acerca del porque se busca minimizar esta función y se describen conceptos y herramientas estrechamente relacionados.

<sup>45</sup>En castellano, positivo verdadero.

<sup>46</sup>En castellano, falso positivo.

## 6. Resultados y análisis correspondientes

Una vez explicadas las métricas en la sección anterior, (5) en esta parte de la memoria se exponen los resultados de los cuatro entrenamientos que se han efectuado, así como sus análisis correspondientes. Luego de exponer estos resultados, se muestra la salida de `rpn_to_roi` para diferentes imágenes, a fin de observar las propuestas de regiones que la red ha llevado a cabo, y así justificar los resultados de los entrenamientos.

### 6.1. Resultados de los entrenamientos

En esta sección se exponen los resultados de los cuatro entrenamientos que pueden encontrarse en las figuras 18, 19, 20 y 21.

En el primer entrenamiento, el número máximo de *bounding boxes* se estableció en 50, y el *aspect ratio* de los *anchors* fue de (128, 256, 512). En el segundo entrenamiento se estableció un máximo de 400 *bounding boxes* y un *aspect ratio* para los *anchors* de (16,32,64). Posteriormente, para el tercer y cuarto entrenamiento, los *aspect ratios* fueron de (16,32,64) y (32,64,128) respectivamente, mientras que el número máximo de *bounding boxes* se mantuvo en 400.

Como puede observarse en los dos primeros entrenamientos, y especialmente en el primero, las distintas funciones de pérdidas tienen muchas oscilaciones y parece no haber ningún comportamiento que las permita converger. Esto es lógico, ya que en el primer entrenamiento, los pesos dentro de la red se están regulando por primera vez. Por ello, en el segundo entrenamiento ya se observa una tendencia menos oscilatoria (ver figura 19), aunque no es hasta el cuarto entrenamiento que las funciones de pérdidas convergen al final del proceso (ver figura 21). También influye el tamaño de los *aspect ratios*, pues en el primer entrenamiento son mucho mayores que en el resto, y esto ha podido afectar negativamente al entrenamiento de la red por lo que se comentó en la sección 4.3.

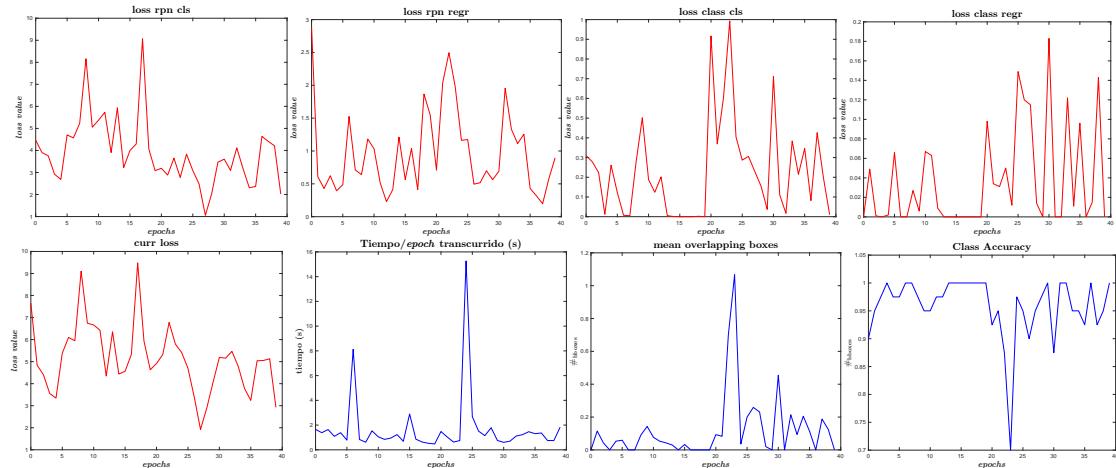


Figura 18: Resultados del primer entrenamiento.

Para establecer una diferencia entre los tipos de métricas, en rojo se representan los resultados correspondientes a las distintas funciones de pérdidas, mientras que en azul se representan el resto de métricas que pueden leerse en el título de cada una de las gráficas. Además, cabe destacar, como el orden de los resultados de los entrenamientos expuestos en las figuras 18, 19, 20 y 21, es relevante, ya que en el primer entrenamiento, los valores relativos de las funciones de pérdidas son mayores

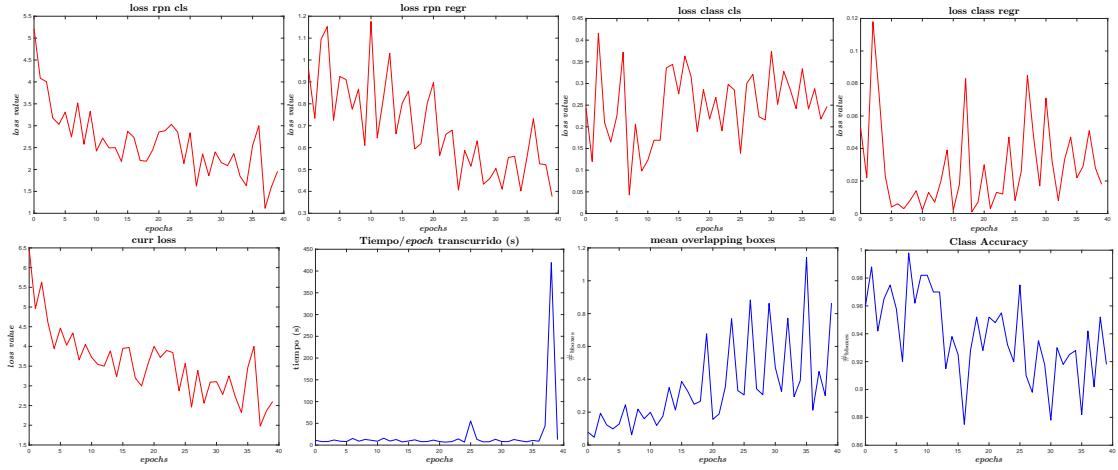


Figura 19: Resultados del segundo entrenamiento.

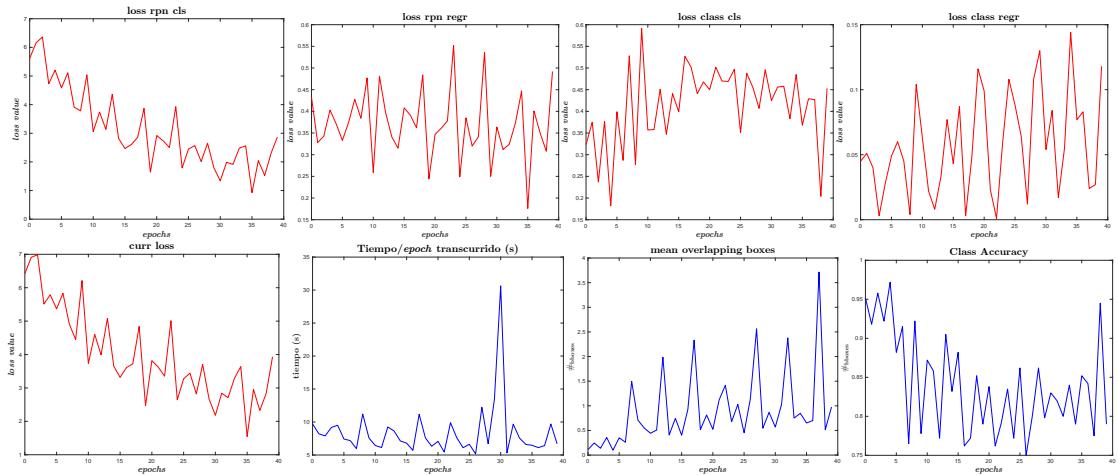


Figura 20: Resultados del tercer entrenamiento.

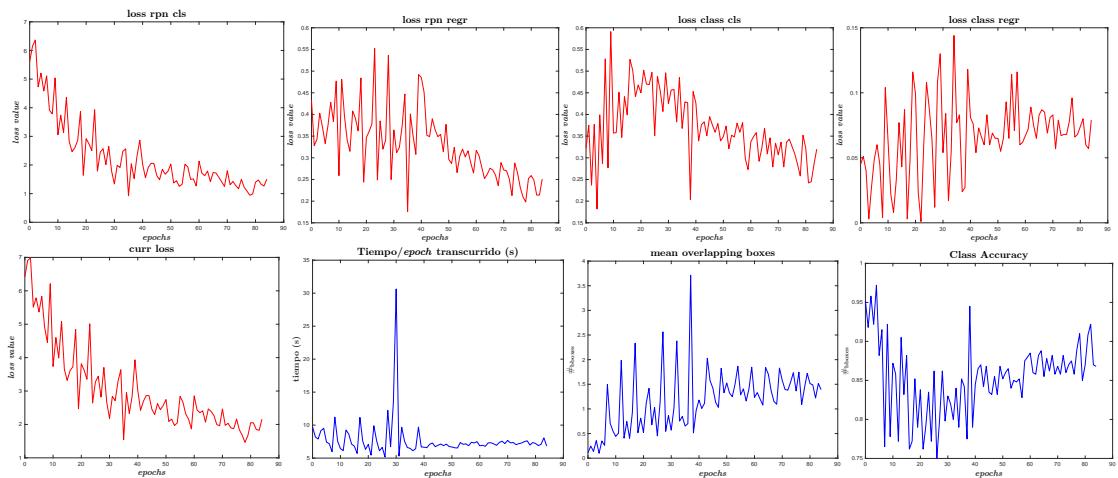


Figura 21: Resultados del cuarto entrenamiento.

respecto al tercero y cuarto, esto significa que los entrenamientos posteriores van a alcanzar mejores resultados, a no ser que los parámetros que se le especifique a la red sean radicalmente diferentes.

En el cuarto entrenamiento (ver figura 21) se observa como las pérdidas asociadas al modelo RPN tienen un comportamiento decreciente a lo largo de todo el proceso, aunque en las pérdidas asociadas a la etapa de regresión de los *bounding boxes*, se observa un ligero crecimiento. Además, para las pérdidas asociadas a la clasificación de la detección, se observa una rápida caída de la pendiente, lo que implicaría un aprendizaje más rápido<sup>47</sup>. Así mismo, se observa una tendencia convergente, mientras que las pérdidas de la etapa de regresión siguen disminuyendo, alcanzando un valor menor. Esto podría significar que la precisión de los defectos ya es alta en la fase inicial del entrenamiento, pero al mismo tiempo, la precisión de las coordenadas de los *bounding boxes* sigue siendo baja y necesitaría más tiempo de aprendizaje.

En las dos funciones de pérdidas aplicadas al modelo clasificador, se observa una tendencia creciente al inicio del entrenamiento y en el caso de las pérdidas asociadas a la regresión de coordenadas de los *bounding boxes* esta tendencia creciente domina todo el proceso<sup>48</sup>.

A continuación se adjunta una tabla con la media de los resultados de cada una de las funciones de pérdidas para los cuatro entrenamientos:

#	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

Cuadro 6: Valores medios de las métricas obtenidas tras los cuatro entrenamientos.

Los tiempos de media por *epoch*<sup>49</sup> han sido los siguientes para cada uno de los cuatro entrenamientos:

#	tiempo(s)
1°	1.6836
2°	22.2965
3°	8.3643
4°	7.7076

Cuadro 7: Media de tiempos para cada *epoch* de cada entrenamiento

De estas medias calculadas, y de los resultados en general, se puede inferir lo siguiente:

- El algoritmo de optimización está correctamente implementado, pues a lo largo del proceso de entrenamiento para los 4 casos, las pérdidas bajan de manera paulatina (ver figuras 18, 19, 20 y 21), esto puede verse mejor en el cuadro 6. Esto significa, que la red está siguiendo el proceso de entrenamiento normal, pero esto no quiere decir que este entrenándose adecuadamente.

<sup>47</sup>Más adelante se argumenta acerca de porque el aprendizaje efectuado no es correcto.

<sup>48</sup>La explicación de este párrafo y el anterior, se apoya en una argumentación contenida en [Xu].

<sup>49</sup>En los cuatro entrenamientos, el *epoch* es 40 (ver explicación en la sección 5.1).

De hecho, puede verse como el parámetro **mAP** se mantiene constante y nulo para todos los entrenamientos (ver sección 5.2)

- Como se concluía en el anterior punto, el parámetro **mAP**, indica que la “precisión media” es 0, echando la vista atrás y volviendo a la ecuación 5.2, se puede ver como  $\text{mAP} = \text{recall}$ , y para que el *recall* fuese cero, ello implicaba que  $TP = 0$ . Esto en términos prácticos, se traduce en una escasa llegada de regiones positivas a la red, o simplemente, estas son muy pocas ( $\sim 1\%(\text{zonas totales})$ ) mientras que casi el 99 % de las zonas son negativas. Esto significa que debido al umbral establecido, la mayoría de las detecciones han sido clasificadas como *False Positives*.

Teniendo en cuenta la conclusión final de la lista de arriba, la pregunta que puede surgir es, cómo es posible que la red esté aprendiendo y que el algoritmo de optimización funcione correctamente, si a la red se le están pasando zonas negativas. Pues bien, aunque el algoritmo de optimización consiga reducir la función de pérdidas, este lo hace encontrando mínimos incorrectos, ya que la propia función de optimización no es correcta. Esto último no significa que la ecuación 5.2 sea inválida, sino que los parámetros relativos a la propuesta de regiones dentro de la ecuación 5.2 no son del todo ciertos.

Si uno profundiza en esta problemática, puede darse cuenta como a medida que la red pasa por la etapa de entrenamiento a esta se le van otorgando una serie de premios, cada vez que detecta una zona negativa. Estos premios se traducen en un aumento de pesos<sup>50</sup>, y en una no penalización por parte de la función de pérdidas por lo que la red deja de aprender. Además, como todo lo que detecta la red son zonas negativas, ya que casi todo de lo que llega a la etapa de clasificación son zonas negativas, ya sea por una baja **IoU** (que es lo que pasa en esta tesis) o por que no solapa con ningún *bounding box* del *groundtruth*, los pesos se aumentan considerablemente, y esto no es bueno.

No obstante, aunque lo discutido en el párrafo anterior sea verdad, lo que ha ocurrido en los cuatro entrenamientos, es que al haber un desbalance de clases, i.e., hay muchas más regiones propuestas sin defecto, que con defecto, la red colapsa como consecuencia de no poder aprender debido a la falta de pérdidas que guíen el entrenamiento. Por lo tanto, la red aprende un poco a detectar regiones “negativas” pero este aprendizaje no llega muy lejos debido a lo anterior.

Si se observan los resultados expuestos en la sección 6.2, se intuye como el problema de estos resultados reside en la propuesta de regiones, ya que se le está premiando a la red por detectar zonas calificadas como negativas, por ello la función de pérdidas va disminuyendo, pero el número de regiones positivas es nulo, ya que el **mAP** permanece constante con un valor de 0. Es por esto, que la siguiente sección se centra en el estudio de las propuestas llevadas a cabo por la red.

## 6.2. Propuesta de regiones

Como la red no es capaz de aprender correctamente, se ha de poner el foco en la propuesta de regiones, por ello, antes de la capa de clasificación en el código, se han representado las regiones que propone la red, para saber si los cálculos son coherentes.

Las imágenes que se recogen en este apartado son fruto de numerosas simulaciones con diferentes variaciones del número máximo de *boxes* y del umbral de solapamiento. Véase la siguiente figura:

---

<sup>50</sup>Para profundizar más acerca de los pesos y su influencia dentro de una red neuronal ver las secciones A.1 y A.2 contenidas en los anexos.

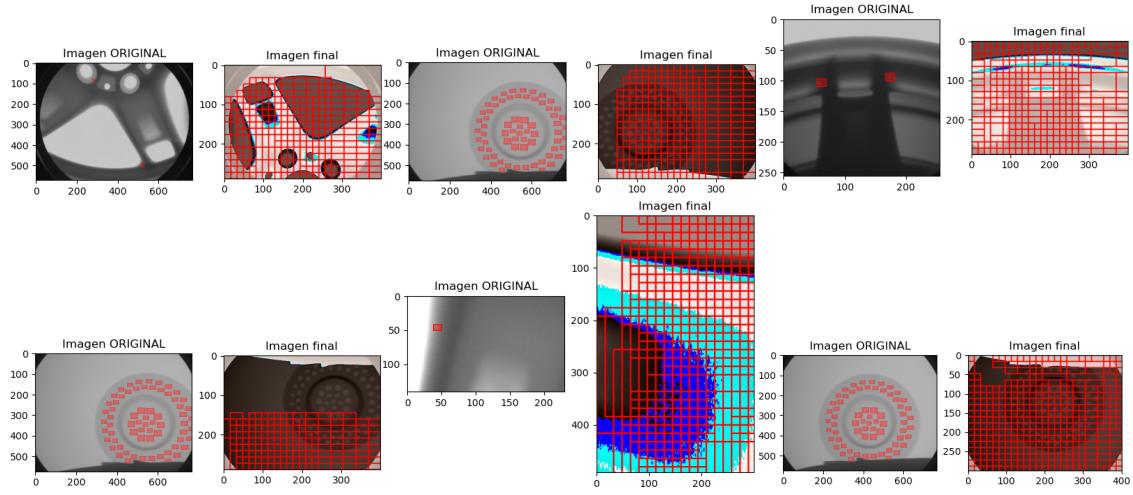


Figura 22: Algunas propuestas de regiones para algunas imágenes

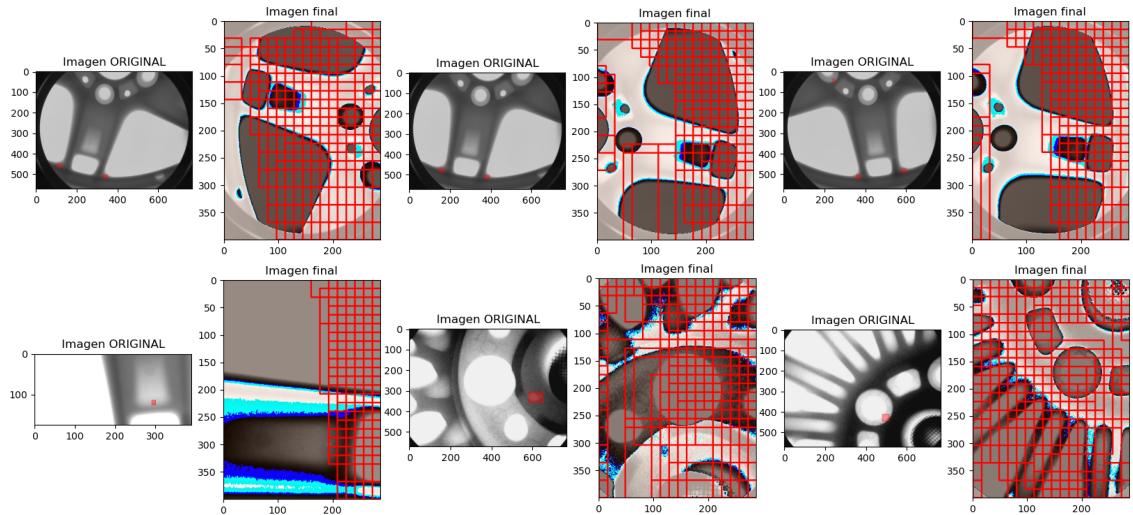


Figura 23: Algunas propuestas de regiones para algunas imágenes

Las imágenes de las figuras 22 y 23 se corresponden con un conjunto de propuestas de regiones que ha realizado la red, para diferentes imágenes de entrada. Para facilitar la tarea de comparación, se ha representado a la izquierda de cada par de imágenes la imagen original con sus *bounding boxes* correspondientes a los defectos que se indican en el *groundtruth* y a la izquierda se representa la imagen modificada con las regiones propuestas por la red. Así pues, uno puede observar si las regiones recuadradas engloban o no defectos.

Cabe destacar el efecto degradante que tienen todas las capas de convolución en la resolución de la imagen, según va pasando por las diferentes capas de la arquitectura. Aunque no pueda apreciarse, en varias imágenes se ve como el defecto desaparece por completo (respecto a la imagen original), pero esto no es por la degradación de la resolución de la imagen, sino porque en el mapa de características ese defecto desaparece tras pasar por varias capas del *backbone*, lo que provoca una propuesta errónea de regiones.

Los resultados de la figura 22 y 23 se han conseguido para un umbral de **IoU** de 0.4 y un número

máximo de *boxes* representables de 50 para unas imágenes y 400 para otras imágenes. Esta gran diferencia entre 50 y 400, se debe al deseo de observar en que posición dentro de la imagen se sitúan las “cajas”, pues si se dibujan solo 50 *bounding boxes*, estos se corresponderán con los *bounding boxes* que tengan mejor calificación y por lo tanto, serán los *bounding boxes* más precisos. De esta manera, representando pocas “cajas” se puede observar más fácilmente, cuales son las “mejores” regiones.

En muchos de los casos se observa como no se rodea el defecto, pero si algunas zonas de alrededor, para evitar esto se ha probado con distintos umbrales de solapamiento (normalmente más bajos, para que se filtre lo menos posible) y un dibujado de más cajas, pero no se ha apreciado una mejora considerable, esto puede apuntar a que el problema viene de antes.

Cabía esperar que la red no propusiese de manera perfecta las regiones dentro de las imágenes, ya que como se menciona en la sección 3 y en [RHGS15], la **Faster R-CNN** es una arquitectura destinada a la detección de objetos en imágenes a color. Y como se mencionaba en la sección 2.4, no se quieren detectar objetos, sino defectos. Además, la proporción de los defectos tampoco ayuda, pues estos son muy inferiores comparados con las de un objeto corriente. A estas dos restricciones, hay que añadirle que las imágenes están en blanco y negro, lo que impide a la red aprender de alguna característica visual para diferenciar mejor los defectos, ya que la diferencia entre el defecto y el fondo de la imagen es el tono de blanco y negro.

En definitiva, la red está intentando proponer regiones a partir de imágenes en blanco y negro, de baja resolución con defectos y proporciones muy pequeños. Esto significa que al extraer el mapa de características de la imagen, el defecto se pierde en el *backbone* debido a los inconvenientes expuestos en el anterior párrafo y el número de capas convolucionales del *backbone*. Aunque la elección del *backbone* no es perfecta, haber evitado una arquitectura *Resnet* ha sido buena idea, ya que esta tiene un mayor número de capas, lo que reduciría aún más la posibilidad de que llegase algún defecto a la etapa de clasificación.

A pesar de que todo apunte a que el mapa de características no está incluyendo muchos de los defectos de las imágenes, puede verse como en la figura 22 , más concretamente en el segundo par de imágenes de la fila de arriba y en el último par de imágenes en la fila de abajo, cuando la imagen contiene muchos defectos y el fondo es monótono, la red suele conservar gran parte de esos defectos, pues posteriormente estos aparecen recuadrados como regiones propuestas. Por ello, en el resto de las imágenes se ve como aunque no haya defectos en algunas zonas de las imágenes, la red propone regiones debido a los cambios de contraste o de color debido a las características de cada una de las piezas de fundición.

Dicho lo anterior, sería erróneo e injusto, culpar únicamente al *backbone* de desechar los defectos y no plasmarlos en el mapa de características, ya que actualmente no hay limitación en cuanto a la modificación del funcionamiento del *backbone* y es por esto, que se podrían introducir mejoras en el mismo<sup>51</sup>.

En la siguiente sección (7.2) se mencionan algunas técnicas y mejoras en la arquitectura implementada de cara a un trabajo a futuro.

---

<sup>51</sup>Estas mejoras se introducen en la sección 7.2.

## 7. Conclusiones y trabajo a futuro

Como se ha podido comprobar en la sección 6, la red no ha podido modelar la etapa de clasificación correctamente, debido a una propuesta de regiones que no es lo suficientemente precisa.

En esta sección se pretende concluir el razonamiento de los resultados obtenidos desde una perspectiva constructiva, con el fin de introducir mejoras en el proceso de detección para entrenar en un futuro una red robusta.

### 7.1. Conclusiones

Aunque no se ha conseguido entrenar el modelo de la **Faster R-CNN** para la detección de defectos, tras los diferentes análisis y estudios que se han llevado a cabo en las secciones 6 y 4 se ha logrado comprender al detalle, los detonantes de un mal aprendizaje y las características de la arquitectura que entorpecen el proceso de entrenamiento. También se ha realizado el análisis con el objetivo de optimizar el funcionamiento de la **Faster R-CNN** para adaptar su etapa de propuesta de regiones para una tarea de detección de defectos.

Aunque el conocimiento más teórico acerca de redes neuronales no es obligatorio para desarrollar este proyecto, si que ha servido para cimentar muchas de las explicaciones y suposiciones llevadas a cabo, además de aclarar conceptos relativos a la dinámica interna de una red neuronal y el efecto que tienen los parámetros más esenciales dentro de ella.

Se ha podido comprobar como incluso con una arquitectura de red no tan profunda, no se logra detectar defectos de proporciones pequeñas dentro de una imagen. No obstante, no se puede culpar solo a la red, pues como posteriormente se verá en la sección 7.2, existen mejoras complementarias que el programador puede implementar para mitigar esas dificultades. Es importante saber, que la red aprende lo que se le pide que aprenda, por ello, la dificultad de un proyecto como este, reside en saber como hay que darle esa información a la red, y en como conducir a la red hacia la realización de una tarea de naturaleza desbalanceada<sup>52</sup> sin que colapse el entrenamiento.

Como se ha corroborado en la sección 4.3, los parámetros que más influyen en la propuesta de regiones, son, las escalas de los *anchors*, los tamaños de las imágenes, y la resolución de estas, ya que todos estos parámetros determinan el mapa de características de la imagen y su posterior entrenamiento. A su vez, esto afecta al entrenamiento de la propuesta de regiones y de la clasificación, y como se ha podido comprobar en la sección 6, un buen entrenamiento de la propuesta de regiones parte de un buen mapa de características.

Adicionalmente a lo mencionado en el párrafo anterior, para acometer un buen entrenamiento, el conjunto muestral ha de ser grande y variado, no ha de haber determinismo en el orden de las imágenes que van entrando a la red, el conjunto muestral ha de tener anotaciones y características importantes que la red pueda tener en cuenta y el tamaño del *batch* debe de ser considerable<sup>53</sup>. Cabe destacar además, que normalmente un entrenamiento más largo implica mejores resultados, siempre y cuando la propuesta de regiones sea correcta.

Finalmente, se ha visto como la implementación de un modelo de detector de objetos, realiza las tareas básicas para detectar defectos pequeños, pero este no logra dar resultados precisos, porque se

---

<sup>52</sup>Este problema se explicaba en el penúltimo párrafo de la sección 6.1.

<sup>53</sup>Al hacer el *batch* más grande, el tiempo de entrenamiento será mayor.

necesita más precisión de detección<sup>54</sup> y técnicas que permitan reutilizar conocimientos adquiridos por la red durante entrenamientos para realizar tareas más fáciles, como el *transfer learning*, para posteriormente entrenar a la red a fin de que esta realice tareas de detección más complejas.

En definitiva, este proyecto pone en valor la necesidad de técnicas adicionales a la hora de detectar defectos con técnicas de detección de objetos, y evidencia las grandes diferencias que existen entre detectar objetos y detectar defectos pequeños.

## 7.2. Trabajo a futuro

El análisis realizado, ha permitido la identificación de estrategias con las que se podría mejorar el modelado del detector de defectos, estas estrategias se explican a continuación:

- Imágenes de mayor resolución. Para mejorar los resultados de cualquier entrenamiento, se pueden utilizar imágenes de mayor resolución, pues como se explicó en la sección 4.3, a mayor resolución, mayor solapamiento entre el *bounding box* del *groundtruth* y el del *anchor*.
- *Transfer learning*. Esta es una técnica, que se centra en tomar un mayor control en el entrenamiento. Esta técnica consiste en la reutilización de un modelo pre-entrenado en un nuevo problema. Actualmente es muy popular en el aprendizaje profundo porque puede entrenar redes neuronales profundas con relativamente pocos datos. Niklas Donges [Don].

Esta técnica aplicaría a este trabajo de fin de grado, de la siguiente forma. Primero se entrena la red para que detecte defectos mucho más grandes en imágenes en blanco y negro<sup>55</sup>, como los defectos de la imagen 9 vista en la sección 2.4. De esta manera, la red se acostumbraría a las imágenes en blanco y negro y tras un largo entrenamiento, evitaría resaltar otros detalles de las imágenes, e.g., las costillas, las clavículas, y otros huesos que puedan contenerse.

Así pues, los pesos de las primeras capas de la red, se mantendrían para un entrenamiento posterior, ya que las primeras capas de la red se centran en extraer texturas, contornos, siluetas y variaciones generales del fondo de la imagen. Posteriormente, la red ya se entrena con las imágenes de la base de datos GDXray, aprovechando el conocimiento previo que tiene la red acerca de las características generales de las imágenes en blanco y negro.

- Implementación de la **Mask R-CNN**. Esta arquitectura ya se mencionó en la sección 2.3, la ventaja principal que tiene, viene dada por una mayor precisión de análisis, ya que es capaz de analizar una imagen con una precisión de fracciones de píxeles, algo que resulta de especial interés debido al pequeño tamaño de los defectos. Ahora bien, el precio a pagar es una mayor complejidad, ya que aunque es muy similar a la **Faster R-CNN**, cuenta con una etapa más para la segmentación, por ello se creyó oportuno hacer funcionar primero las etapas básicas previas a la segmentación, que son las que tiene la **Faster R-CNN**. No obstante, una vez realizado este trabajo de fin de grado, esta arquitectura se podría abordar con más seguridad y por ello esta opción es considerada como la principal sustituta de la **Faster R-CNN**.
- La **FPN**. Como se ha visto en este TFG, detectar defectos pequeños en imágenes en blanco y negro es una tarea complicada. Para superar gran parte de las complicaciones que conlleva este tipo de detección, se puede crear una estructura piramidal que contenga la imagen que se quiere estudiar en diferentes escalas, de arriba a abajo. Pero, esto implica una gran cantidad

---

<sup>54</sup>Como la ofrecida por la arquitectura **Mask R-CNN**.

<sup>55</sup>O incluso, imágenes de otros tipos.

de memoria y tiempo, consecuencia de procesar imágenes en diferentes escalas. Para reducir la cantidad de memoria, en vez de trabajar con imágenes de múltiples escalas, en su lugar se puede trabajar con sus mapas de características asociados.

Jonathan Hui, en [Huib], describe esta red como un extractor de características diseñado para el concepto de pirámide introducido antes. El objetivo de este modelo es generar múltiples capas de mapas de características (mapas de características multiescala) con información de mejor calidad que la pirámide de características normal para la detección de objetos (imagen de la izquierda en la figura 24). Este modelo puede entenderse mejor con la siguiente figura (24).

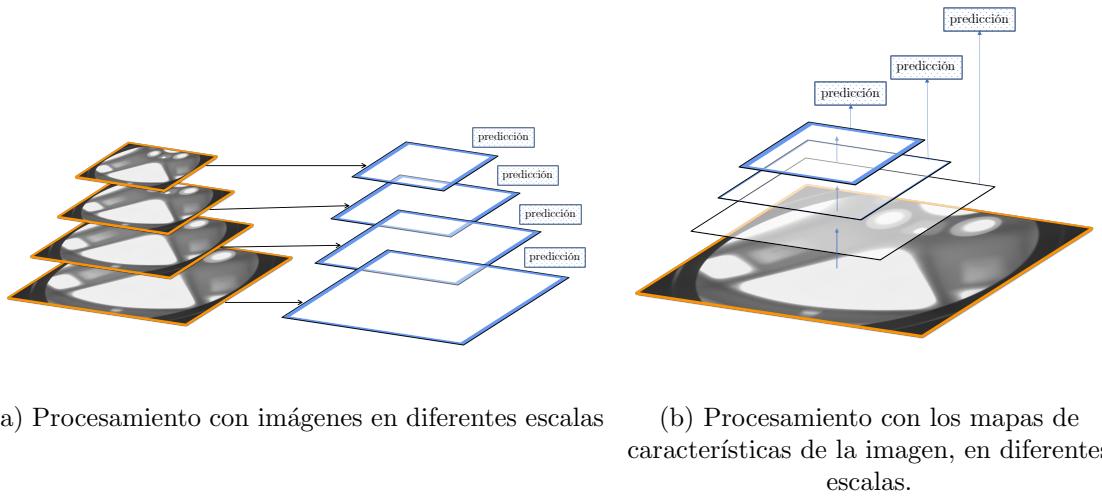


Figura 24: Dinámica de la **FPN**. Figurada creada a partir de una ilustración en [Huib].

No obstante, Jonathan Hui, en [Huib], aclara que la **FPN** no es un detector de objetos en sí mismo, sino un extractor de características que funciona con detectores de objetos, como el implementado en este proyecto, la **RPN**. Esto es una ventaja, pues la **VGG-16** extraería los mapas de características, que posteriormente se introducirían en la **FPN** para generar una pirámide con ellos. Por último, estos mapas a modo de pirámide, son introducidos en la **RPN** con el fin de proponer regiones a partir de ellos, siguiendo la misma dinámica que ya se ha explicado en la sección 3.1.3.

Cabe destacar, que la técnica del *transfer learning* puede complementarse con la arquitectura **Mask R-CNN** así como, con la implementación de la **FPN** en la **Faster R-CNN**.

## 8. Códigos

En esta sección puede encontrarse el enlace al repositorio de **GitHub**, donde se encuentran todos los códigos implementados, incluido el desarrollado en **Matlab**, para representar los resultados de los entrenamientos.

Además, en el mismo repositorio se han incluido gran parte de las imágenes producidas a lo largo de este TFG, el código en Matlab que se ha implementado para representar los resultados de los cuatro entrenamientos y los archivos resultantes de los cuatro entrenamientos en formato **.csv**.

El enlace al repositorio es el siguiente, <https://github.com/jcsombria/tfg-psr>.

## 9. Bibliografía

### Referencias

- [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<sup>+</sup>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 learning 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](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<sup>+</sup>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<sup>+</sup>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](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 entrena.  
<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>[Online; 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].

# Anexos

## A. Fundamento teórico

Aunque el objetivo de este proyecto, no es el estudio teórico de las redes neuronales, se cree pertinente añadir un anexo como este, que explique brevemente los conceptos teóricos más importantes que han ayudado al desarrollo de esta tesis. Por ello, en esta sección se expone el fundamento teórico del trabajo así como algunos avances en tareas de clasificación.

### A.1. Introducción a las redes neuronales

Antes de tratar temas más avanzados, es pertinente introducir la teoría que justifica el funcionamiento de lo más elemental relativo a esta tesis, una red neuronal simple, hasta el funcionamiento de algo elaborado como el de la arquitectura implementada en este trabajo.

Una red neuronal puede describirse como un sistema computacional capaz de realizar una tarea para la cuál ha sido entrenada, a partir de una entrada de datos y un conjunto de variables con valores iniciales que cambian de valor dependiendo de lo bien o mal que vayan realizando la tarea. El primer modelo matemático de una red neuronal, fue introducido por Walter Harry Pitts, un autodidacta matemático y lógico estadounidense que trabajó en el campo de neurociencia computacional. Posteriormente, junto a Warren Sturgis McCulloch, neurólogo y cibernetista estadounidense, llevarían a cabo diferentes ensayos que permitieron avances en los campos de la biofísica y las matemáticas.

Tras numerosos avances por su parte como por parte de otras personalidades científicas, se logró establecer y explicar el concepto de **neurona** como unidad elemental de una red. Una **neurona artificial** puede entenderse mediante la siguiente figura:

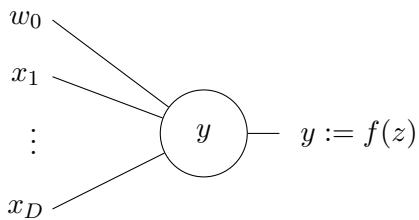


Figura 25: Una sola unidad de procesamiento y sus componentes. La función de activación viene denotada por  $f$  y se aplica a la entrada  $z$  de la “neurona” para formar su salida  $y = f(z)$ . El conjunto de parámetros  $x_1, \dots, x_D$  conforman un vector, i.e.,  $x = (x_1, \dots, x_D)$  donde  $x \in \mathbb{R}^D$ . Estos valores representan la entrada de otras unidades dentro de la red. El parámetro  $w_0$  se denomina sesgo y representa una entrada externa a la unidad. Todas las entradas se asignan a la entrada  $z$  mediante la regla de propagación. David et al. [Stu]

Más adelante, en la figura 26 se muestra un perceptrón. Un perceptrón es un sistema lineal de la siguiente forma:  $y_i = \mathbf{w}^\top \mathbf{x} + \mathbf{b}$  dónde  $\mathbf{x}_0 = 0$  para incluir  $w_0$ , que suele ser denotado por  $\mathbf{b}$ . Al parámetro  $\mathbf{w}$  se le llama **peso**, y representa la intensidad de interacción entre cada neurona.

capa de entrada

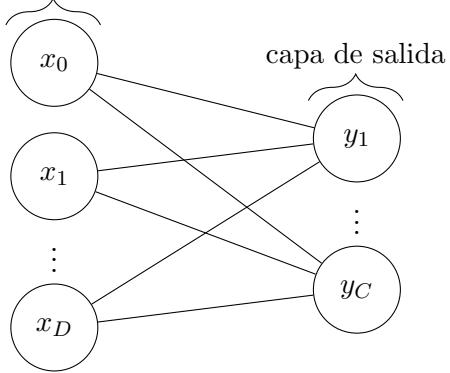


Figura 26: El perceptrón consta de  $D$  unidades de entrada y  $C$  unidades de salida. Todas las unidades se etiquetan según su salida:  $y_i = f(z_i)$  en el caso de las unidades de salida, y  $x_i$  en el caso de las unidades de entrada. Los valores de entrada  $x_i$  se propagan a cada unidad de salida mediante la regla de propagación de suma ponderada:  $y_i = \mathbf{w}^\top \mathbf{x} + \mathbf{b}$ . El valor de entrada adicional  $x_0 := 1$  se utiliza para incluir los sesgos como pesos. David et al. [Stu]

Aspectos importantes a destacar de la figuras 25 y 26:

- ❖ **Función de activación.** Es la función que se aplica a la entrada de cada una de las unidades. El propósito de una función de activación es añadir no linealidad a la red neuronal. Esta no linealidad es muy importante, ya que asegura que la salida no sea un “múltiplo de la entrada” y por lo tanto, que la salida no guarde una relación lineal con la entrada. Cuanto menos lineal sea la función de activación, más complejas serán las decisiones que tome la red y por lo tanto, se obtendrán resultados más interesantes y aptos con los que trabajar.
- ❖ **Entradas ( $\mathbf{x}$ ).** Las redes neuronales pueden tener propósitos muy variados. Algunas de las aplicaciones más populares de las redes neuronales son la visión por ordenador, el reconocimiento de voz y el procesamiento del lenguaje natural. Por lo tanto, dependiendo de la aplicación, el formato de los datos de entrada variará. En este caso, como la intención es detectar defectos en imágenes, las entradas representarán la imagen que se quiere analizar<sup>56</sup>.
- ❖ **Regla de propagación.** Comúnmente conocido en inglés como: “*propagation rule*”, engloba dos términos muy importantes: “*forward propagation*” y “*backwards propagation*”
  - La “*forward propagation*”. En castellano: “propagación hacia delante” es la forma de pasar de la capa de entrada (izquierda) a la capa de salida (derecha) en la red neuronal.
  - La “*backward propagation*” o “*back propagation*”. En castellano: “propagación hacia atrás” es el proceso de desplazamiento de derecha a izquierda, es decir, de la capa de salida a la de entrada. La mejor forma de corregir o ajustar los pesos ( $w$ ) para conseguir minimizar la función de pérdidas<sup>57</sup> es mediante la propagación hacia atrás.

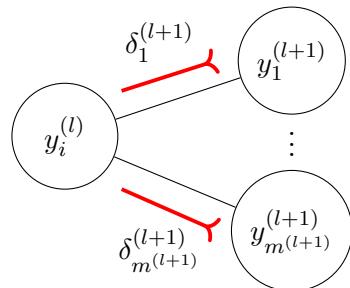


Figura 27: Una vez los errores  $\delta_i^{(L+1)}$  han sido evaluados en todas las unidades de salida, estos son propagados hacia atrás. David et al. [Stu]

<sup>56</sup>Es importante destacar el procesado previo al que se someten cada una de las imágenes antes de ser introducidas en la red (ver sección 3)

<sup>57</sup>De esto se habla más adelante (ver ecuación 9)

Conocidos los parámetros esenciales que determinan el funcionamiento de un conjunto de neuronas, se pasa al “estudio” de una red completa, con diferentes “*hidden layers*”<sup>58</sup> y “*hidden units*”<sup>59</sup>. A continuación se adjunta la figura 28 que ilustra una red neuronal arbitraria, dónde la nomenclatura que se utiliza es la siguiente;

$y_{\#}^{(\nu)}$  donde  $\left\{ \begin{array}{l} (\nu) : \text{determina el número de la capa en la que se encuentra la neurona } \# \\ \# : \text{determina el número de la neurona dentro de la capa } (\nu). \end{array} \right.$

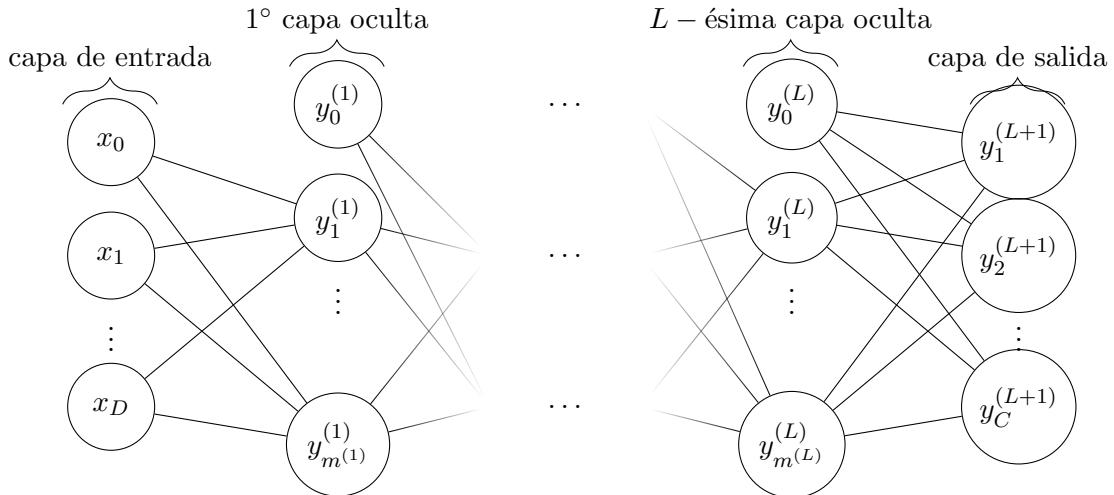


Figura 28: Gráfica de un perceptrón de  $(L + 1)$  capas con  $D$  unidades de entrada y  $C$  unidades de salida. La capa oculta  $l$  – éSIMA contiene  $m^{(l)}$  unidades ocultas. David et al. [Stu]

Las capas de entrada no efectúan ningún cálculo, simplemente toman los valores de entrada y los propagan a las capas ocultas, donde se aplica la función de activación correspondiente. Normalmente, todas las neuronas de una capa aplican la misma función de activación.

Siguiendo la jerarquía, se llega a las “*hidden layers*” i.e., capas ocultas. Estas capas, introducen la abstracción necesaria para realizar y tomar tareas y decisiones complejas, mediante la realización de numerosas operaciones sobre la entrada que reciben.

Finalmente, la capa de salida, conduce el resultado, fruto de propagar la información desde la primera capa oculta, hasta la última, a la salida. Normalmente el *output* de esta capa es una probabilidad, e.g., “*La probabilidad de que haya un defecto, en la imagen introducida es  $\mathbb{P}$* ”.

## A.2. Método de clasificación

Comprender el método de traducción de los resultados obtenidos en cada una de las neuronas a un valor probabilístico<sup>60</sup>, resulta un tanto abstracto e incluso complicado. Por ello, en esta sección se pretende explicar de manera concisa, este proceso.

El algoritmo por excelencia que se encarga de relacionar las características de una imagen con un resultado probabilístico en particular, es el de la Regresión Logística, del inglés; *Logistic Regression*. Poniendo otro ejemplo, Jurafsky et al. [Mar] escriben lo siguiente, “*La regresión logística*

<sup>58</sup>En castellano: “capas ocultas”

<sup>59</sup>En castellano: “unidades ocultas” o “neuronas ocultas”

<sup>60</sup>Se refiere a: “¿Cuál es la probabilidad de que haya un defecto en la imagen introducida?”

puede utilizarse para clasificar una observación en una de dos clases (como sentimiento positivo y sentimiento negativo)”. Esto es aplicable para el tema de este TFG, por lo tanto, la regresión logística puede utilizarse para clasificar una observación en una de dos clases como, hay defecto y no hay defecto.

Este es un algoritmo de clasificación de tipo discriminativo, esto significa que se basa en diferenciar las clases sin aprender mucho de ellas, e.g., saber diferenciar entre un perro o un gato, porque en todas las fotos de perros, estos llevan collares y los gatos no. Jurafsky et al. [Mar]. Teniendo en cuenta esto, el modelo aprenderá a asignar un peso  $\mathbf{w}$  elevado, a las características de la imagen que mejoren su capacidad de discriminar entre posibles clases.

Como la variable objetivo es binaria, el modelo de regresión es binario, y el nombre que recibe es: “*binary logistic regression*”. Para pasar de características a un valor binario, se hace uso de la función sigmoidea, que tiene la siguiente forma:

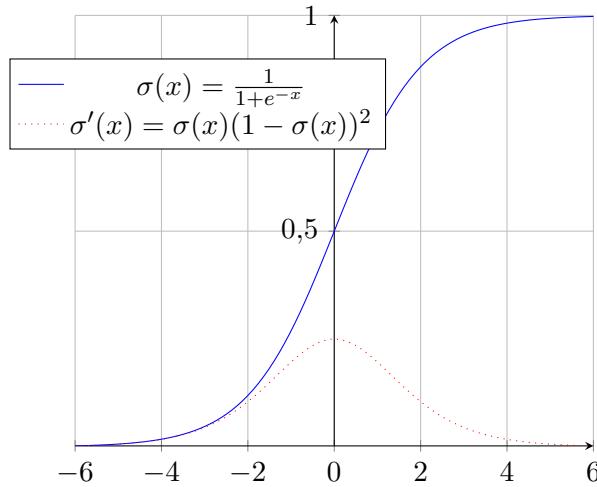


Figura 29: Representación gráfica de la función sigmoidea. StackExchange et al. [Stab]

Ahora se estudia la dinámica de las redes neuronales. Considérese un conjunto de características de la forma:  $[x_1, x_2, \dots, x_n]$  donde  $\mathbf{x} \in \mathbb{R}^n$ , y dónde la estimación llevada a cabo por el modelo clasificador se denota con  $\hat{y}$ . Cada una de las características de entrada lleva asociado un peso de la forma  $[w_1, w_2, \dots, w_{\#}]$  donde  $\mathbf{w} \in \mathbb{R}^{\#}$ . La salida del clasificador, será:  $\hat{y} = P(\mathbf{y} = 1|\mathbf{x})$  si en la imagen hay defecto(s) e  $\hat{y} = P(\mathbf{y} = 0|\mathbf{x})$  si la imagen no contiene defectos. Además, como  $\hat{y}$  representa una probabilidad, puede definirse de la siguiente manera:  $\hat{y} \in \{0, 1\}$ .

El modelo de regresión logística aprenderá gracias a la implementación de la función sigmoidea y de las características  $\mathbf{x}$  y los pesos  $\mathbf{w}$  asociados a cada una de ellas. La función sigmoidea es muy importante, pues si se retoma la siguiente expresión enunciada en la sección anterior:  $y_i = \hat{y} = \mathbf{w}^\top \mathbf{x} + b$ , se sabe que  $y_i$  puede ser negativa, o mucho mayor que 1, violando así, la definición de arriba. Pero si uno implementa la función  $\sigma(z)$  siendo  $z = \mathbf{w} \cdot \mathbf{x} + b$ <sup>61</sup> y mira su forma, puede comprobarse fácilmente como  $\sigma(z) \approx 1$  si  $z \uparrow$  debido a que  $e^{-z} \rightarrow 0$ . También se verifica que  $\sigma(z) \approx 0$  si  $z \downarrow$  ya que  $e^{-z}$  acaba siendo un número negativo de gran magnitud. Por lo que la función signmoidea, delimita los valores finales entre 0 y 1.

<sup>61</sup>El producto de dos vectores elemento a elemento se representa con la notación “.”. Además, los vectores son representados en negrita.

### A.3. Función de pérdidas y función de costes

Para entrenar el modelo de regresión, es necesario definir una función de costes. Para ello, se considera un conjunto de  $m$  muestras de entrenamiento, que se define de la siguiente manera:  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ . Para cada una de las muestras de entrenamiento, se tendrá que;  $\hat{y}^{(i)} = \sigma(z^{(i)})$  con  $z^{(i)} = w^\top x^{(i)} + b$ . Asimismo, es importante mencionar, el deseo de que  $\hat{y}^{(i)} \approx y^{(i)}$ , i.e., que la estimación sea similar al valor real. Una vez definidas las variables, la función de pérdidas se define de la siguiente manera:

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

La función de pérdidas es definida respecto a una sola muestra, i.e.,  $(x^{(i)}, y^{(i)})$  y permite saber cuánto de bien le va en el proceso de entrenamiento. Además, un buen entrenamiento vendrá determinado por 9, que al ser una función de pérdidas lo que se busca es minimizar su valor.

Definida la función de pérdidas, la función de costes, se define en la siguiente ecuación:

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

Sustituyendo 9 en 10 se tiene que:

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

Cabe destacar, que mientras que la función de costes mide el error del modelo en un conjunto, la función de pérdidas gestiona una única instancia de datos.

Ahora el objetivo es minimizar  $\mathcal{J}(w, b)$ , para ello hay que minimizar  $w$  y  $b$ . Aquí es donde entra en juego el algoritmo del “gradient descent” en castellano, el descenso del gradiente. Este algoritmo puede entenderse intuitivamente con la siguiente figura, donde el algoritmo comienza en un punto y va calculando la derivada, hasta encontrar el mínimo de la función.

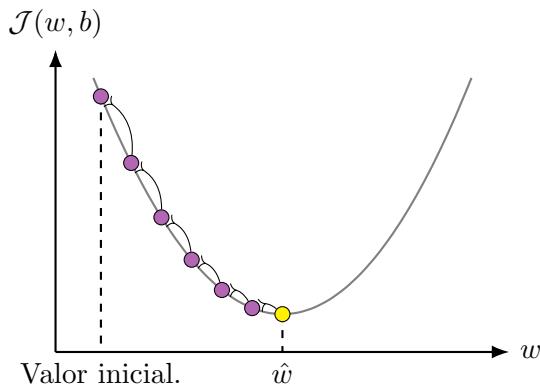


Figura 30: Ilustración gráfica del algoritmo del descenso del gradiente. StackExchange et al. [Stac]

El pseudocódigo correspondiente a este algoritmo puede encontrarse al inicio de la siguiente página.

El parámetro  $\alpha$  determina el índice o tasa de aprendizaje. La tasa de aprendizaje es un hiperparámetro (al igual que  $\mathbf{w}$  y  $\mathbf{b}$ ) utilizado para regular el ritmo al que un algoritmo actualiza o

---

**Algoritmo 2** Descenso del gradiente

---

```

procedure GRADIENT DESCENT( $\{(\mathbf{X}_i, Y_i)\}_{i \in (1, \dots, m)}$ ,  $\alpha$ ,  $\mathbf{w}_0$ ,  $\mathbf{b}_0$ )
     $\mathbf{w} \leftarrow \mathbf{w}_0$ .
     $\mathbf{b} \leftarrow \mathbf{b}_0$ .
    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

```

---

aprende los valores de la estimación de un parámetro. En otras palabras, la tasa de aprendizaje regula los pesos de la red neuronal en relación con el gradiente de pérdidas. Además, este indica la frecuencia con la que la red neuronal actualiza las nociones que ha aprendido. DeepChecks et al. [Dee].

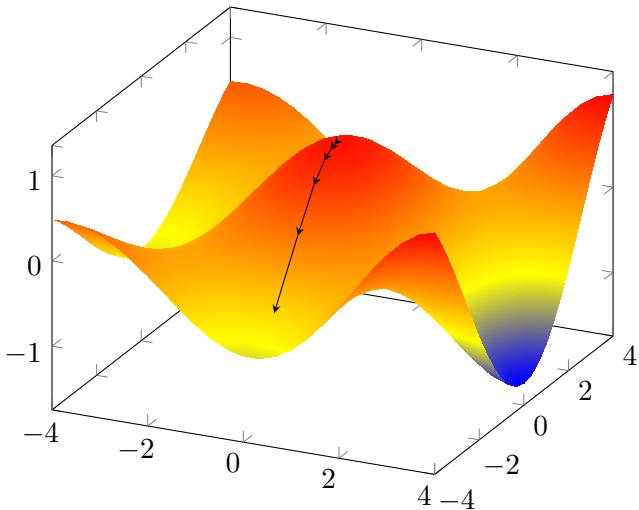


Figura 31: Función de costes arbitraria. StackExchange et al. [Stad]

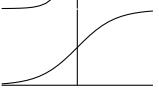
En la figura de arriba, el eje vertical representa el valor de la función de costes, i.e.,  $\mathcal{J}(\mathbf{w}, \mathbf{b})$ , mientras que los dos ejes paralelos en la parte inferior representan los hiperparámetros  $\mathbf{w}$  y  $\mathbf{b}$ . Hay que tener en cuenta que la figura representa una función de costes arbitraria y dependiendo del problema que se quiera resolver, tomará una determinada forma característica. Como se mencionó en la sección 1.3, el diseño de redes neuronales implica conocimientos de optimización, ya que uno de los objetivos finales del cómputo neuronal, es el de la reducción de costes. Esto, desde la perspectiva matemática se traduce en encontrar el mínimo de una función y ello es lo que se quiere representar con la flecha negra en la figura 31.

#### A.4. Funciones de activación

El objetivo principal de cualquier red neuronal es transformar los datos de entrada no separables de manera lineal, en características abstractas más separables linealmente, utilizando una jerarquía de capas. Las capas no lineales más populares, son las funciones de activación. Dubey et al. [DSC21].

Las funciones de activación, juegan un papel fundamental en el entrenamiento de la red, pues condicionan su aprendizaje y consecuentemente, el resultado del mismo. Es importante mencionar que no hay una función de activación que sea mejor que el resto, todo depende de la aplicación que se quiera desarrollar, y dependiendo de esto, habrá algunas funciones que potencien en mayor o menor medida el entrenamiento.

Si  $\exists x \in \mathbb{R}^{n \times m}$ , o si  $\exists x \in \mathbb{R}^n$ , entonces se pueden definir las siguientes funciones de activación no lineales:

Función	Expresión	Derivada	Figura
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
tanh	$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	
Softmax	$f(x) = \frac{e^x}{\sum_i e^x}$	$f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$	

Cuadro 8: Funciones de activación no lineales. El código asociado a esta tabla se encuentra en [Pet].

Entre todas las funciones de la tabla de arriba, la función **ReLU** ha demostrado una mayor capacidad de obtener mejores resultados tras su uso, frente al resto.

La razón real por la que se utiliza más la **ReLU** es que, según aumentan las capas en una CNN, el entrenamiento de la misma resulta más fácil y rápido, que si se utiliza la función *tanh*, o la función  $\sigma(x)$ . Esto se debe a que la función *tanh* tiene el problema del *vanishing gradient*, que consiste en que a medida que se apilan más capas, los gradientes son cada vez más pequeños y como el paso en el espacio de pesos del algoritmo de retropropagación es proporcional a la magnitud del gradiente, cuando los gradientes desaparecen, significa que la red neuronal ya no se puede entrenar. El impacto en los tiempos de entrenamiento es directo, traduciéndose en un aumento exponencial según aumenta el número de capas. Por esta razón, se le ha asignado a cada unidad de la red implementada, la función de activación **ReLU**.

Cabe destacar, como los algoritmos, funciones y técnicas explicadas a lo largo de esta sección, aplican a cualquier tipo de red, pero dada la complejidad de la tarea que se quiere realizar en este proyecto, una red neuronal no es suficiente para detectar defectos en imágenes. Para acometer una tarea de esta envergadura, se requiere de un conjunto de redes y capas que realicen tareas específicas relativas al procesamiento de las imágenes de entrada, extracción de sus características y propuesta de regiones. Esto es último es exactamente de lo que ha tratado este proyecto.

Gran parte de la teoría expuesta en este anexo, ha sido extraída de los apuntes tomados en varios cursos de *Deep Learning* impartidos en la plataforma de Coursera por parte de Andrew Ng. Estos apuntes pueden verse en [Cou] y se puede acceder al enlace de estos cursos en [AN].