



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

XRayDetector

**Detección de defectos en
piezas metálicas a partir de
imágenes de radiografía**



Presentado por Noelia Ubierna Fernández
en Universidad de Burgos — 20 de julio
de 2020

Tutor: José Francisco Diez Pastor y Pedro
Latorre Carmona



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. José Francisco Díez Pastor y Pedro Latorre Carmona, profesores del departamento de Ingeniería Informática, área de Lenguajes y Sistemas informáticos.

Exponen:

Que la alumna Dña. Noelia Ubierna Fernández, con DNI 71565283-R, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado “Deteción de defectos en piezas metálicas a partir de imágenes de radiografía”.

Y que dicho trabajo ha sido realizado por la alumna bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 20 de julio de 2020

Vº. Bº. del Tutor:

D. José Francisco Díez Pastor

Vº. Bº. del Tutor:

D. Pedro Latorre Carmona

Resumen

El control de calidad es un aspecto importante de los procesos de fabricación. Ha aumentado el nivel de competencia en el mercado, por lo que los fabricantes deben aumentar su tasa de producción manteniendo los límites de calidad para los productos.

Durante la creación de piezas metálicas se pueden crear pequeñas burbujas o poros que son indetectables por el ojo humano aun encontrándose en la superficie. Estas burbujas pueden hacer que la pieza se rompa durante su periodo de uso, pudiendo llegar a ser un problema crítico en el caso, por ejemplo, de las piezas de automóviles ya que están sometidas a una fatiga continua.

La finalidad del proyecto es la de aplicar un sistema de aprendizaje basado en una red neuronal, para la identificación de estos defectos en imágenes de rayos-X tomadas principalmente de partes automotrices (ruedas de aluminio y nudillos).

Descriptores

Piezas metálicas, imágenes rayos-X, burbujas, imperfecciones, detección, aplicación de escritorio.

Abstract

Quality control is an important aspect of manufacturing processes. The level of competition in the manufacturing market has increased, so manufacturers must increase their production rate while maintaining quality limits for their products.

During the creation of metal parts, small bubbles or pores can be created that are undetectable by the human eye even when they are on the surface. These bubbles can cause the piece to break during its period of use, which can become a critical problem in the case, for example, of auto parts since they are sometimes in continuous fatigue.

The purpose of the project is to apply a learning system based on a neural network, for the identification of these defects on X-ray images taken mainly from automotive parts (aluminum wheels and knuckles).

Keywords

Metal parts, X-ray images, bubbles, blemishes, detection, desktop application.

Índice general

Índice general	III
Índice de figuras	v
Índice de tablas	vii
Introducción	1
1.1. Estructura de la memoria	2
1.2. Materiales adjuntos	3
Objetivos del proyecto	5
2.1. Objetivos generales	5
2.2. Objetivos técnicos	5
2.3. Objetivos personales	6
Conceptos teóricos	7
3.1. Minería de datos	7
3.2. Nondestructive testing (NDT)	13
3.3. Segmentación de imágenes	15
3.4. <i>Average Precision</i> (AP)	18
Técnicas y herramientas	23
4.1. Python	23
4.2. Anaconda	23
4.3. Jupyter Notebook	23
4.4. Spyder	24
4.5. Bibliotecas de Python	24

4.6. Modelos de Python	25
4.7. Git	26
4.8. L ^A T _E X	26
4.9. tmux	27
4.10. Técnicas	27
Aspectos relevantes del desarrollo del proyecto	29
5.1. Introducción	29
5.2. Entrenamiento	30
Trabajos relacionados	57
6.1. Artículos científicos	57
6.2. Conferencias	58
Conclusiones y Líneas de trabajo futuras	61
7.1. Conclusión	61
7.2. Líneas de trabajo futuras	62
Bibliografía	63

Índice de figuras

3.1.	Inteligencia Artificial y subcampos	9
3.2.	Imagen de prueba	10
3.3.	Máscara de una imagen de prueba	11
3.4.	Funcionamiento esquematizado de las Redes Neuronales	13
3.5.	Ejemplo de radiografías industriales	14
3.6.	Arquitectura de red de la máscara R-CNN	15
3.7.	Arquitectura ResNet	17
3.8.	Segunda etapa	18
3.9.	Fórmula IoU	19
5.10.	Etapas de entrenamiento del código	30
5.11.	Últimas <i>epochs</i> del entrenamiento	31
5.12.	Evaluación con el modelo número 159	33
5.13.	Evaluación con el modelo número 160	34
5.14.	Imagen C0001_0018 con los <i>Bounding Box</i> del modelo 159	35
5.15.	Cuadrícula de la imagen C0001_0018 de objetos de verdad y sus predicciones del modelo 159	36
5.16.	Imagen C0001_0018 con los <i>Bounding Box</i> del modelo 160	37
5.17.	Cuadrícula de la imagen C0001_0018 de objetos de verdad y sus predicciones del modelo 160	38
5.18.	Imagen C0001_0004 con los <i>Bounding Box</i> del modelo 159	39
5.19.	Cuadrícula de la imagen C0001_0004 de objetos de verdad y sus predicciones del modelo 159	40
5.20.	Imagen C0001_0004 con los <i>Bounding Box</i> del modelo 160	41
5.21.	Cuadrícula de la imagen C0001_0004 de objetos de verdad y sus predicciones del modelo 160	42
5.22.	Imagen con anclajes positivos	45
5.23.	Clasificación de propuestas	46

5.24. ROI antes del refinamiento	47
5.25. ROI después del refinamiento	48
5.26. Detecciones después de NMS	49
5.27. Objetivos de entrenamiento para la rama de la máscara	50
5.28. Máscaras predichas	50
5.29. Gráfica del loss de las 160 <i>epochs</i>	51
5.30. Gráfica del mrcnn bbox loss de las 160 <i>epochs</i>	52
5.31. Gráfica del mrcnn class loss de las 160 <i>epochs</i>	52
5.32. Gráfica del mrcnn mask loss de las 160 <i>epochs</i>	53
5.33. Gráfica del rpn bbox loss de las 160 <i>epochs</i>	53
5.34. Gráfica del rpn class loss de las 160 <i>epochs</i>	54
5.35. Código para gradar los valores de las gráficas	55

Índice de tablas

3.1. Conjunto de datos	20
3.2. Datos finales	21
5.3. Comparación de las dos últimas <i>epochs</i>	35

Introducción

El control de calidad es un aspecto importante de los procesos de fabricación. Ha aumentado el nivel de competencia en el mercado de fabricación por lo que los fabricantes deben aumentar su tasa de producción manteniendo los límites de calidad para los productos.

Las piezas de aluminio tienen actualmente un alto interés tecnológico en diversos sectores industriales (como el del automóvil) debido, entre otras, a propiedades de interés, como:

- Su ligereza.
- Su óptima conductividad térmica y eléctrica (un conductor de aluminio pesa sólo la mitad que un conductor de cobre equivalente).
- Su alto porcentaje de reciclabilidad.

No obstante, durante su creación se pueden crear pequeñas burbujas o poros que son indetectables por el ojo humano aun encontrándose en la superficie. Estas burbujas pueden hacer que la pieza se rompa durante su periodo de uso, pudiendo llegar a ser un problema crítico en el caso, por ejemplo, de las piezas de automóviles ya que están sometidas a una fatiga continua.

Hay una serie de exámenes no destructivos (*nondestructive examination* - NDE) que son técnicas disponibles para producir imágenes bidimensionales y tridimensionales de un objeto tales como: imágenes de rayos-X, inspección ultrasónica, inspección de partículas magnéticas u otras.

Actualmente el análisis (inspección) de estas piezas se realiza de manera visual y subjetiva por parte de personal de la propia empresa. Los empleados

examinan las imágenes de rayos-X y determinan la posición de los defectos en el caso de que haya. Este proceso de análisis presenta dos grandes desventajas: el cansancio y el cambio de criterio del operario, con el paso del tiempo. Además, cada operario tiene un criterio diferente de evaluación, dificultando un seguimiento objetivo de la calidad.

La finalidad del proyecto es la de aplicar un sistema de aprendizaje basado en una red neuronal, para la identificación de estos defectos en imágenes de rayos-X tomadas principalmente de partes automotrices (ruedas de aluminio y nudillos).

El objetivo final a largo plazo sería poder comprobar la validez de una metodología automatizada que permita reducir costes y hacer el proceso más robusto y objetivo.

1.1. Estructura de la memoria

Esta memoria incluye los siguientes apartados:

- **Introducción:** Breve descripción del problema a resolver y la solución propuesta. Estructura de la memoria y listado de materiales adjuntos.
- **Objetivos del proyecto:** Exposición de los objetivos generales, técnicos y personales del proyecto.
- **Conceptos teóricos:** Breve explicación de los conceptos teóricos necesarios para la comprensión y el desarrollo del proyecto.
- **Técnicas y herramientas:** Presentación de las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto.
- **Aspectos relevantes del desarrollo:** Listado o exposición de los aspectos más importantes durante el desarrollo del proyecto.
- **Trabajos relacionados:** Breve resumen de los trabajos y proyectos vinculados con la detección de defectos en imágenes de rayos-X y el estado del arte del proyecto.
- **Conclusiones y líneas de trabajo futuras:** Conclusiones obtenidas al final del proyecto y exposición de posibles mejoras o líneas de trabajo futuro.

1.2. Materiales adjuntos

Anexos aportados junto a la memoria:

- **Plan del proyecto software:** Planificación temporal y estudio de viabilidad económica y legal del proyecto.
- **Especificación de requisitos del software:** Objetivos generales, catálogo de requisitos del sistema y especificación de requisitos funcionales y no funcionales.
- **Especificación de diseño:** Diseño de datos, diseño procedimental y diseño arquitectónico.
- **Manual del programador:** Estructura de directorios, manual del programador, compilación, instalación ejecución y pruebas (aspectos relevantes del código fuente).
- **Manual de usuario:** Requisitos de usuarios, instalación, manual de usuario.

Repositorio del proyecto accesible a través de internet <https://github.com/nuf1001/XRayDetector/> [7].

Objetivos del proyecto

2.1. Objetivos generales

- Investigar sobre técnicas del estado del arte aplicadas a la detección de defectos a través de imágenes de rayos-X con la ayuda de artículos científicos relevantes para documentar las tecnologías, herramientas y procesos utilizados.
- Desarrollar una aplicación de escritorio que permita la cómoda utilización del software de detección de defectos en imágenes de rayos-X.
- Proporcionar una interpretación de los datos obtenidos sencilla, para facilitar al usuario su entendimiento.
- Analizar el rendimiento de los modelos obtenidos, su exactitud a la hora de predecir.

2.2. Objetivos técnicos

- Aplicar el método **Mask R-CNN** para la segmentación de instancias de objetos.
- Desarrollar una aplicación de escritorio implementada en *Python*, que refleje los resultados obtenidos para una herramienta funcional.
- Utilizar *tkinter* como una herramienta para crear la aplicación.
- Utilizar *GitHub* como herramienta para el seguimiento del proyecto junto con su extensión *ZenHub*.

- Aprender a utilizar L^AT_EX, en especial la herramienta online Overleaf.
- Utilizar la metodología ágil *Scrum* adaptada a este proyecto.
- Realizar test de usabilidad y otros.

2.3. Objetivos personales

- Aplicar al máximo los conocimientos adquiridos por la Universidad.
- Aprender sobre inteligencia artificial.
- Aprender sobre el desarrollo de aplicaciones de escritorio.
- Aportar a los sectores industriales un producto que mejore la rapidez y el rendimiento dentro de las empresas.

Conceptos teóricos

3.1. Minería de datos

La minería de datos consiste en la aplicación de técnicas de la inteligencia artificial sobre grandes volúmenes de datos, teniendo como objetivo el extraer tendencias o patrones interesantes de dicho conjunto de datos y convertirlos en un formato comprensible.

Todos los días se crea una gran cantidad de bytes de datos, pero para extraer información singular de esos datos es necesario aplicar distintos métodos o técnicas tales como el aprendizaje automático. Para ello se llevan a cabo distintas etapas [14]:

- **Selección:** Recopilar e integrar los volúmenes de datos existentes. Identificar las variables más importantes. Aplicar técnicas de muestreo.
- **Exploración:** Utilizar las técnicas de análisis de exploratorio de datos. Obtener la distribución de los datos. Analizar la conexión que existe dentro de esta información.
- **Limpieza:** Detección de valores atípicos. Eliminar datos erróneos e irrelevantes.
- **Transformación:** Modificar la dimensión (aumentar, disminuir). Utilizar técnicas de discretización y numeración. Creación del escalado simple y multidimensional.
- **Minería de datos:** Utilizar técnicas predictivas: análisis discriminante, redes neuronales, árboles de decisión, regresión y series temporales,

algoritmos genéticos y métodos bayesianos. Utilizar técnicas descriptivas: Reglas de asociación y dependencia, *Clustering* y segmentación, reducción de la dimensión, escalamiento y análisis explorativo.

- **Evaluación e interpretación de resultados:** Intervalos de confianza, *bootstrap*, análisis ROC¹ (*Receiver Operating Characteristic*), evaluación de modelos.
- **Difusión y uso de modelos:** Su visualización y su simulación.

Esto tiene relación con este proyecto porque disponemos de unas imágenes de las que queremos obtener una información en concreto. Por lo tanto, en nuestro caso como ya disponemos de las imágenes solo aplicaremos las tres últimas etapas.

Ahora definiremos algunos conceptos importantes para entender mejor todo esto.

Inteligencia Artificial

La inteligencia artificial o IA es la simulación de tareas intelectuales realizadas por humanos por parte de las máquinas, es decir, es una disciplina que es capaz de crear sistemas con capacidad de aprender según su experiencia como una persona real, siendo también capaces de resolver problemas o llevar a cabo tareas lógicas.

La inteligencia artificial contiene el aprendizaje automático (*Machine Learning*) y el aprendizaje profundo (*Deep Learning*) aunque también es parte de campos que no llevan a cabo un aprendizaje como tal.

Durante los años 80 se pensaba que se podía alcanzar al nivel de inteligencia humana haciendo que los programadores crearan un gran conjunto de reglas suficiente para controlar el conocimiento y construir máquinas intelectuales. Esta idea es conocida como *IA simbólica*.

Se comprobó que este enfoque era capaz de resolver problemas que estaban bien definidos y eran lógicos, como jugar al ajedrez, pero no era funcional con problemas más complejos que necesitaban reglas explícitas para su resolución, como el reconocimiento de voz, la clasificación de imágenes, etc. Para estos problemas se trató con un nuevo enfoque, el aprendizaje automático o *machine learning*.

¹Representación de la razón o ratio de verdaderos positivos (VPR) frente a la razón o ratio de falsos positivos (FPR) según se varía el umbral de discriminación.

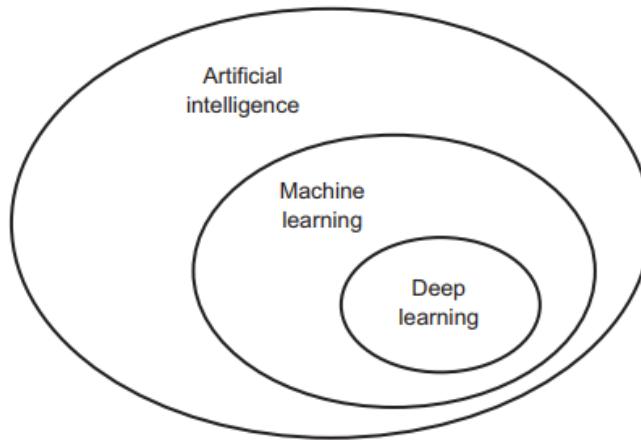


Figura 3.1: Inteligencia Artificial y subcampos [2]

Sobre la selección de las imágenes en este proyecto no hay una selección en sí de imágenes para el entrenamiento y el testeo, sino que hay un archivo tipo .txt que contiene la dirección de las imágenes que se van a entrenar y otro con las que se van a testear.

Aprendizaje Automático o *Machine Learning*

A la mayoría de los programas se les introducen unas reglas y unos datos para que sean procesados siguiendo estrictamente estas reglas y de este modo se obtiene una o varias respuestas al final del programa, pero con el aprendizaje automático son los datos y sus respuestas esperadas, en el caso del aprendizaje supervisado, lo que se introduce como entrada con el fin de obtener las reglas que permitan hacer un mapeo efectivo. Estas reglas son las que a continuación van a ser utilizadas con un nuevo conjunto de datos para generar sus respuestas, es decir, el sistema “ha aprendido” a generar estas respuestas.

Por lo tanto, para la poder realizar un aprendizaje automático necesitamos tres componentes indispensables:

- **Datos de entrada:** En nuestro caso imágenes de rayos-X de piezas metálicas con o sin defectos.
- **Datos salida esperados:** La localización de los defectos de cada imagen. En nuestro proyecto tenemos las máscaras de las imágenes y

los *bounding box* (cuadro delimitador) donde se encuentran los errores.

Las máscaras son imágenes de ceros y unos, los unos representan el pixel donde se encuentra un defecto y los ceros el resto de la imagen.

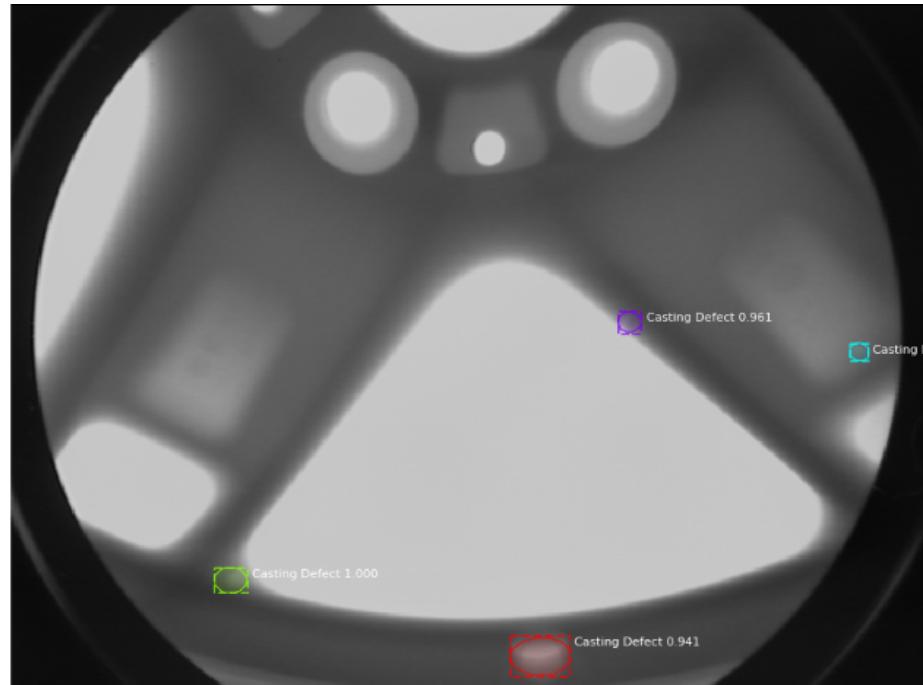


Figura 3.2: Imagen de prueba

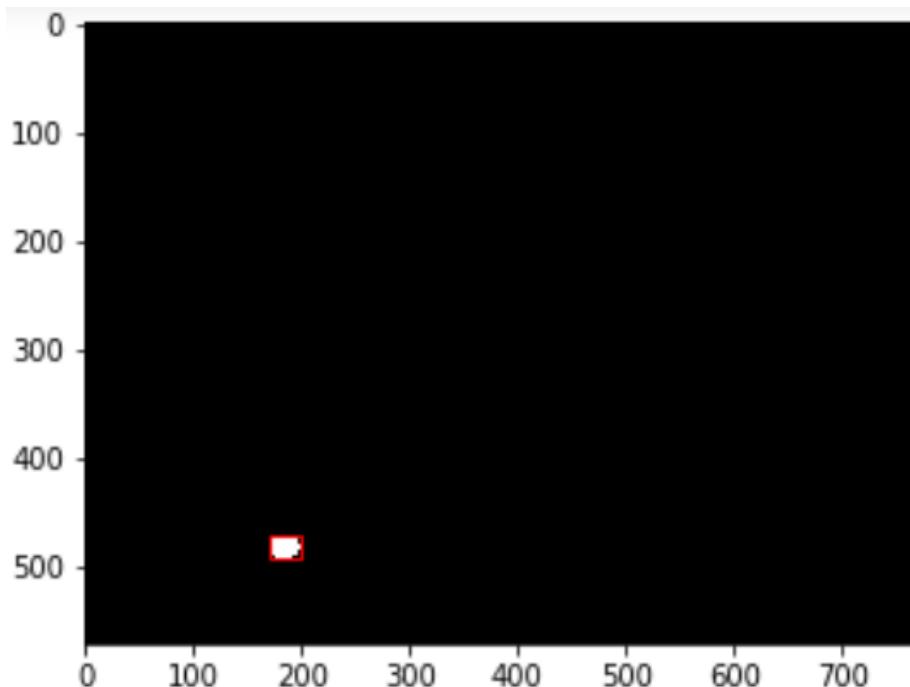


Figura 3.3: Máscara de una imagen de prueba

En la figura 3.2 tenemos una imagen con los defectos superpuestos y en la figura 3.3 se ve la máscara de uno de los defectos de la imagen, los pixeles blancos representan un defecto.

El *bounding box* de esta imagen es: [(472, 171, 493, 199)]. Estos valores representan las x y las y de donde se encuentra el defecto ([(xmin, ymin, xmax, ymax)]).

- **Un método de comprobación de la salida generada:** Imprescindibles para determinar la diferencia entre la salida generada y la salida esperada. Esto se utiliza como señal realimentación para que el algoritmo se actualice o aprenda.

Estos componentes también son los imprescindibles en el aprendizaje profundo. Los tipos de implementación son [3]:

- **Aprendizaje supervisado:** Los algoritmos disponen de datos “etiquetados” (*labelled data*) e intentan crear una función que, dados los datos de entrada, asigne una etiqueta adecuada a los datos de salida.

El algoritmo va a aprender a asignar las etiquetas gracias a un conjunto de datos pre-existente con el que se entrena.

- **Aprendizaje no supervisado:** Los algoritmos no disponen de datos “etiquetados”, solo se conoce los datos entrada, no existen las etiquetas asociadas que permiten establecer la clase a la que pertenecen. Es decir, intentan descubrir la estructura de los datos para encontrar alguna forma de organización o patrón.
- **Aprendizaje de refuerzo:** La base de este tipo de aprendizaje es mejorar la respuesta del modelo. Para ello lleva a cabo un proceso de retroalimentación, es decir, aprende del medio externo, del entorno que le rodea. Los datos de entrada son los *feedback* que obtiene como respuesta a sus acciones. El sistema aprende a base de ensayo-error.

Aprendizaje Profundo o *Deep Learning*

Este tipo de aprendizaje se basa en la creación de modelos computacionales compuestos por varias capas, es decir, el término “profundo” hace referencia a la idea de la representación continua de los datos por medio de dichas capas. La profundidad del modelo es la cantidad de capas que tenga.

Los modelos de estas capas son denominados **Redes Neuronales**, las capas están apiladas una detrás de la otra. Este término viene de la neurobiología, y aunque muchos de los conceptos del aprendizaje profundo tuvieron como inspiración el funcionamiento del cerebro humano, la verdad es que aún no hay evidencias de que el cerebro esté estructurado de esta forma ni de que aprenda usando estos modelos de aprendizaje profundo.

Redes Neuronales

Como ya hemos dicho este algoritmo está basado en el funcionamiento del cerebro humano, es decir, que entre las capas o nodos denominados neuronas artificiales se transmiten señales entre sí desde la capa de entrada (*input*) hasta la de salida (*output*). Dentro de cada neurona hay a su vez capas que poseen un peso, un valor numérico con el que modifica su entrada. La capa pasa al siguiente la salida que ha obtenido y así hasta el final.

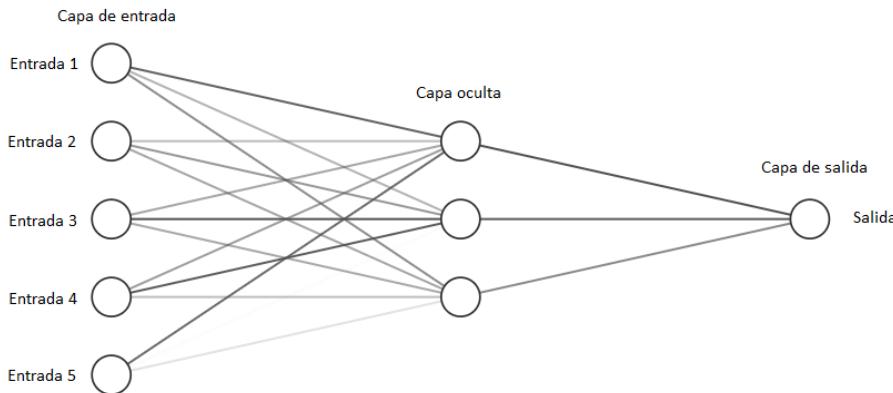


Figura 3.4: Funcionamiento esquematizado de las Redes Neuronales

3.2. Nondestructive testing (NDT)

El *Nondestructive testing* (NDT) [25] es un procedimiento que se usa para inspeccionar, probar o evaluar los materiales o componentes para detectar defectos en las características de tal manera que la estructura o diseño original de estos materiales al ser inspeccionados no sean modificados o destruidos. Al final de la inspección, la pieza no debe haber sido modificada, debe poder usarse.

Este tipo de pruebas se hacen para asegurar que la calidad de los productos alcance un cierto “estándar”.

Para las pruebas destructivas debe haber un número limitado de componentes de muestra en vez de los componentes que realmente van a salir al mercado. Estas pruebas se suelen utilizar en la fabricación para garantizar la fiabilidad del producto, reducir costes de producción y mantener un alto nivel de calidad.

Hay tres tipos de métodos de pruebas:

- **Superficiales:** Dan información sobre el estado superficial de los materiales inspeccionados. Los métodos son: Inspección Visual (VT), Líquidos Penetrantes (PT), Partículas Magnéticas (MT) y Electromagnetismo (ET).
- **Volumétricas:** Dan información sobre el estado interno de los materiales inspeccionados. Los métodos son: Radiografía Industrial (RT), Ultrasonido Industrial (UT) y Emisión Acústica (AE).

- **De hermeticidad:** Dan información del grado en que pueden ser contenidos los fluidos en recipientes, los métodos son: Pruebas de Fuga, Pruebas por Cambio de Presión (neumática o hidrostática), Pruebas de Burbuja, Pruebas por Espectrómetro de Masas y Pruebas de Fuga con Rastreadores de Halógeno.

Para este proyecto trabajaremos con radiografías industriales (*Radio-graphic Testing* o RT) que utilizan rayos-X o rayos-gamma. Las aleaciones de aluminio son ligeras y poco densas por lo que es mejor utilizar rayos-X, los rayos-gamma se suelen utilizar para metales más gruesos o densos como el plomo o el hormigón.

Radiografías Industriales (RT)

Este método de pruebas implica que un objeto de prueba va a ser expuesto a una gran radiación que pasará a través de él y genera una imagen en una placa especial.

Las placas que se suelen utilizar se llaman películas de rayos-X industriales, también se pueden utilizar algunos tipos de detectores de radiación digital. El efecto final de tener áreas más claras, donde la radiación ha penetrado menos, y áreas más oscuras, donde la radiación ha pasado más fácilmente. Si hay un defecto o vacío en una parte donde pasa más radiación se queda plasmado en la película como una mancha oscura.

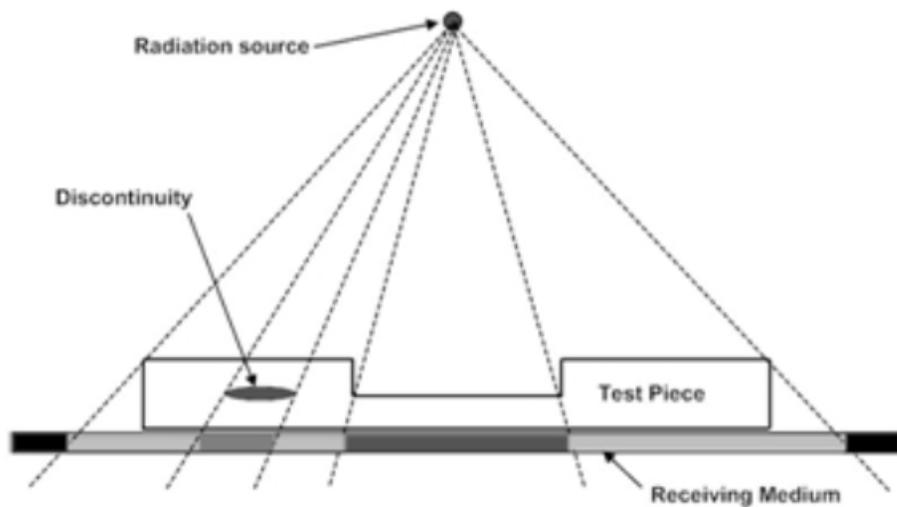


Figura 3.5: Ejemplo de radiografías industriales [25]

3.3. Segmentación de imágenes

Consiste en dividir la imagen en sus partes componentes hasta llegar a un punto de subdivisión en el que se recogen las regiones u objetos de interés. La base de estos algoritmos es o la similitud o la discontinuidad entre los niveles de gris (en las imágenes que sean en escala de grises) o entre valores RGB (en las imágenes a color) de píxeles vecinos.

Este proyecto ha utilizado el modelo de *deep learning Mask R-CNN* [11] para la detección de defectos en piezas metálicas. Esta implementación parte del *framework* realizado por Matterport Inc. [27], el cual se proporciona en código abierto.

Mask R-CNN

Las redes neuronales convolucionales son un tipo de redes neuronales artificiales donde las “neuronas” corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico.

Mask R-CNN es una red neuronal convolucional regional que se compone de un marco de dos etapas: la primera escanea la imagen y genera propuestas (áreas que probablemente contengan un objeto). La segunda clasifica las propuestas y genera cuadros delimitadores y máscaras.

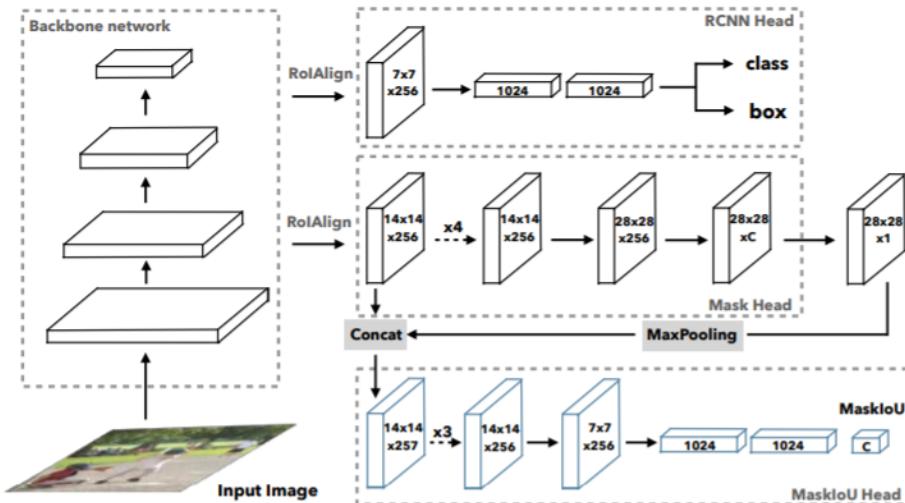


Figura 3.6: Arquitectura de red de la máscara R-CNN [29]

La primera etapa de esta arquitectura contiene tres sub-bloques:

- **Backbone:** Se encarga de extraer mapas de características de las imágenes con capas convolucionales (normalmente ResNet50 o ResNet101 [3.3](#)).
- **Feature Pyramid Network (FPN) [12]:** Permite que las características de alto nivel se encuentren a diferentes escalas y con información de diferentes niveles.
- **Region Proposal Network (RPN) [22]:** Propone regiones que tienen una alta probabilidad de contener un objeto de una de las clases entrenadas.

ResNet (*Residual neural network*) [\[15\]](#) es un tipo de red neuronal convolucional, que se utiliza frecuentemente para clasificar imágenes, también es capaz de extraer características de dichas imágenes. Los detalles del modelo ResNet son:

- **Etapa 1:** La convolución 2D tiene 64 filtros de forma² (7,7) y utiliza una zancada³ de (2,2). Su nombre es “conv1”. *BatchNorm*⁴ se aplica al eje de canales de la entrada. *MaxPooling*⁵ utiliza una ventana (3,3) y una zancada (2,2).
- **Etapa 2:** El bloque convolucional utiliza tres conjuntos de filtros de tamaño 64x64x256, *kernel size* = 3, *strides* = 1. Los 2 bloques de identidad utilizan tres conjuntos de filtros de tamaño 64x64x256, *kernel size* = 3.
- **Etapa 3:** El bloque convolucional utiliza tres conjuntos de filtros de tamaño 128x128x512, *kernel size* = 3, *strides* = 2. Los 3 bloques de identidad utilizan tres conjuntos de filtros de tamaño 128x128x512, *kernel size* = 3.
- **Etapa 4:** El bloque convolucional utiliza tres conjuntos de filtros de tamaño 256x256x1024, *kernel size* = 3, *strides* = 2.. Los 5 bloques de

²Kernel size o tamaño del grano de la imagen.

³Strides o saltos, se utilizan para reducir la carga de anclajes. Si generamos anclajes para cada otra celda en el mapa de características y tenemos una zancada de 2 el número de anclas se reducirá en 4.

⁴Es una técnica para mejorar la velocidad, el rendimiento y la estabilidad de las redes neuronales artificiales.

⁵Encuentra el valor máximo entre una ventana de muestra y pasa este valor como resumen de características sobre esa área.

identidad utilizan tres conjuntos de filtros de tamaño 256x256x1024, $kernel\ size = 3$.

- **Etapa 5:** El bloque convolucional utiliza tres conjuntos de filtros de tamaño 512x512x2048, $kernel\ size = 3$, $strides = 2$. Los 2 bloques de identidad utilizan tres conjuntos de filtros de tamaño 256x256x2048, $kernel\ size = 3$.

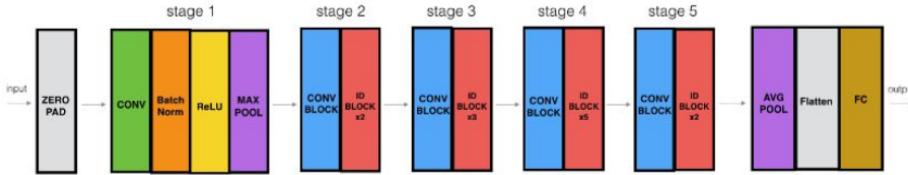


Figura 3.7: Arquitectura ResNet

En la figura 3.7 la rquitectura de ResNets de 50 capas que hemos explicado anterior mente.

El RPN genera dos salidas para cada región:

- **Clases de regiones:** Primer plano o fondo.
- **Refinamiento del cuadro delimitador:** Si una región del primer plano no está perfectamente centrada sobre el objeto, entonces, el RPN estima un porcentaje de cambio en x, y, ancho y alto para refinar el cuadro.

Después de que ya se hayan obtenido las regiones propuestas, comienza la segunda etapa. El algoritmo ROI-Align⁶ [11] se aplica para ajustar el tamaño de las regiones a la entrada del clasificador. Este algoritmo utiliza un método de interpolación bilineal⁷. Cada región propuesta se introduce en diferentes capas cabeceras de la red: un clasificador (estima la clase a la que pertenece el objeto), una cabecera (estima el cuadro delimitador del objeto) y otra cabecera formada por una *Fully Convolutional Network* (estima la máscara que segmenta el objeto).

⁶ROI o *Region of Interest* es una región propuesta a partir de la imagen original.

⁷Es una extensión de la interpolación lineal para interpolar funciones de dos variables (por ejemplo, x e y) en una malla regular de dos dimensiones.

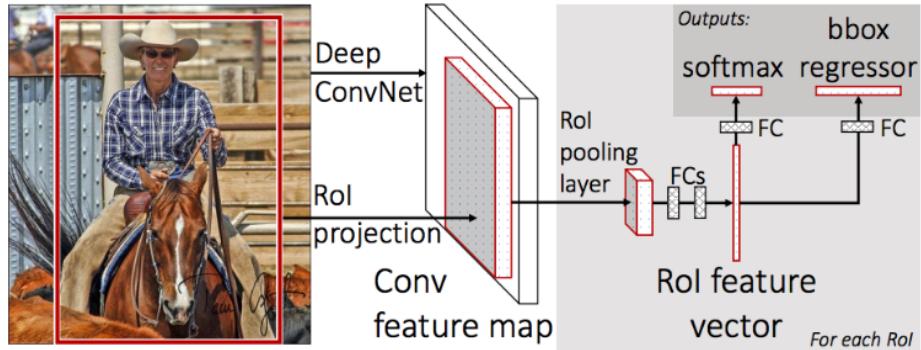


Figura 3.8: Segunda etapa [1]

The Common Objects in Context (COCO)

Los datos de entrenamiento necesarios para realizar esta tarea requieren los objetos etiquetados con sus respectivas máscaras (o plantillas). De esta manera, es posible localizar automáticamente el objeto en la imagen. No existen muchas bases de datos públicas con este tipo de características.

Microsoft COCO [13] es un conjunto de datos muy utilizado en segmentación de objetos. Contiene más de 200.000 imágenes etiquetadas y cerca de 80 clases diferentes.

Mask R-CNN [27] ya proporciona pesos pre-entrenados para Microsoft COCO para empezar. Hemos utilizado estos pesos como punto de partida para entrenar la red por primera vez.

3.4. Average Precision (AP)

Antes de poder comprender AP (*Average Precision*) o mAP (*mean Average Precision*) [24][9], primero debemos entender qué es precisión (*precision*), sensibilidad o exhaustividad (*recall*) y IoU (*Intersection over union*).

Precisión y sensibilidad

Precision: Mide cuán precisas son nuestras predicciones o la proporción de puntos de datos que nuestro modelo dice cuales son realmente relevantes. Es decir, que porcentaje de sus predicciones son correctas.

Recall: Mide qué tan bueno es el modelo para encontrar todos los puntos de datos de interés o casos relevantes.

Definiciones matemáticas:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

TP = True Positive TN = True Negative

FP = False Positive FN = False Negative

Positivo (*Positive*) o Negativo (*Negative*): Se refiere a la predicción. Si el modelo predice 1 entonces será positivo, y se predice 0 será negativo.

Verdadero (*True*) o Falso (*False*): Se refiere si la predicción es correcta o no.

Hay que tener en cuenta que, si aumentamos la precisión, la sensibilidad disminuirá y viceversa.

IoU (*Intersection over union*)

IoU mide la superposición entre 2 límites. Lo usamos para medir cuánto se superpone nuestro límite predicho con la verdad del área (el límite del objeto real cuyas coordenadas se dan en el conjunto de entrenamiento).

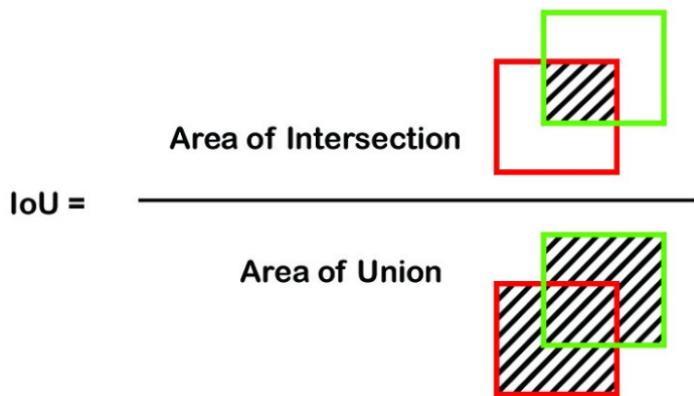


Figura 3.9: Fórmula IoU [24]

Esto nos ayuda a clasificar si la predicción es un verdadero positivo (TP) o un falso positivo (FP). Con un valor umbral de 0.5 normalmente se clasifica de tres formas:

- IoU >0.5: Verdadero Positivo (TP).
- IoU <0.5: Falso Positivo (FP).
- IoU >0.5 Pero el objeto está clasificado erróneamente: Falso Negativo (FN).

En esta clasificación no hay un verdadero negativo (TN), esto es porque se supone que el cuadro delimitador siempre tendrá algo dentro, lo que significa que un cuadro delimitador nunca estará vacío.

AP (*Average Precision*)

Para empezar, necesitamos un conjunto de datos con las coordenadas reales de los defectos y las coordenadas predichas. Por ejemplo⁸:

Imagen	Coordenadas reales	Coordenadas predichas
Img1	[123 233 183 367]	[130 253 169 370]
Img2	[276 202 454 310]	[341 209 466 295]
Img3	[378 204 456 266]	[372 211 460 268]
Img4	[309 101 429 343]	[325 157 409 361]
Img5	[71 252 163 348]	[26 275 160 371]

Tabla 3.1: Conjunto de datos

A continuación, debemos calcular la precisión (*precision*) y la sensibilidad (*recall*). Tras calcular estos tres datos todavía quedaría un dato por calcular antes de calcular mAP, la precisión interpolada (*Interpolated Precision* - IP).

Interpolated Precision: se corresponde con el valor de precisión más alto para un determinado nivel de sensibilidad. Es decir, si tenemos el mismo valor de sensibilidad para tres valores de precisión diferentes 0.87, 0.76 y 0.68, entonces la precisión interpolada para los tres valores de sensibilidad será la más alta entre estos tres valores.

$$P_{interp}(r) = \max_{r' \geq r} p(r')$$

⁸Estos datos no son datos reales de las imágenes que tenemos. Son números aleatorios, pero con sentido.

Datos finales:

Imagen	IoU	TP/FP	Precision	Recall	IP
Img1	0.550569	TP	1.000000	0.090909	1.000000
Img2	0.482510	FP	0.500000	0.090909	1.000000
Img3	0.774103	TP	0.666667	0.166667	0.666667
Img4	0.513853	TP	0.750000	0.230769	0.750000
Img5	0.430901	FP	0.600000	0.230769	0.750000

Tabla 3.2: Datos finales

Finalmente, dividimos el valor de sensibilidad de 0 a 1.0 en 11 puntos: 0, 0.1, 0.2, ... , 0.9 y 1.0. En nuestro ejemplo los valores máximo y mínimo son 0.230769 y 0.000000 respectivamente. Los valores serían: 0.000000, 0.0230769, 0.0461538, ... , 0.2076921, 0.230769. Con ello calculamos el promedio del valor de precisión máximo para estos 11 valores de sensibilidad.

$$AP = \frac{1}{11} \cdot (AP_r(0) + AP_r(0,1) + \dots + AP_r(1,0))$$

En nuestro ejemplo,

$$AP = \frac{4 \cdot 1,000000 + 4 \cdot 0,666667 + 3 \cdot 0,750000}{11} = 0,810606$$

Técnicas y herramientas

En este apartado explicaremos las técnicas y herramientas utilizadas en nuestro proyecto.

4.1. Python

Se ha utilizado el lenguaje de programación *Python*. Es un lenguaje de programación dinámicamente tipado, que soporta orientación a objetos. Posee una licencia de código abierto. Es uno de los lenguajes más utilizados para Inteligencia Artificial (IA) y *Data Science*.

4.2. Anaconda

Anaconda es una distribución de *Python* para Cálculo numérico, análisis de datos y *Machine Learning*. También es posible crear varios entornos de trabajo si estás trabajando en varios proyectos.

4.3. Jupyter Notebook

Entorno de desarrollo integrado (*Integrated Development Environment* - IDE) de programación de *Python* basado en *IPython* de código abierto. El cuaderno Jupyter es una aplicación web que le permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto narrativo. Se usa para limpieza y transformación de datos, simulación numérica, modelado estadístico, visualización de datos, aprendizaje automático, etc.

4.4. Spyder

Spyder es un IDE escrito en *Python*, para *Python*, y diseñado por y para científicos, ingenieros y analistas de datos. Ofrece una combinación de la funcionalidad avanzada de edición, análisis, depuración y creación de perfiles de una herramienta de desarrollo integral con la exploración de datos, ejecución interactiva, inspección profunda y capacidades de visualización de un paquete científico.

4.5. Bibliotecas de Python

Keras

Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en *Python*. Es capaz de ejecutarse sobre *TensorFlow*. Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de Aprendizaje Profundo. Sus fuertes se centran en ser amigable para el usuario, modular y extensible.

TensorFlow

TensorFlow es una librería de *Python*, desarrollada por Google, para realizar cálculos numéricos mediante diagramas de flujo de datos. En vez de codificar un programa, codificaremos un grafo. Los nodos de este grafo serán operaciones matemáticas y las aristas representan los tensores (matrices de datos multidimensionales).

NumPy

NumPy [16] es una biblioteca de *Python*, especializada en computación de datos. Posee gran potencial para el manejo de datos numéricos y sus operaciones, sobre todo de manera matricial, ya que contiene funciones sofisticadas y de uso simple.

SciPy

SciPy [10] es una biblioteca que contiene herramientas y algoritmos matemáticos. Su base es el objeto multidimensional de *NumPy*.

Matplotlib

Matplotlib [8] es una biblioteca que se usa para la generación y muestra de diferentes gráficos. Procesa los datos de *Python* y genera diferentes salidas. Su funcionamiento es parecido a *Matlab*.

Python Imaging Library

Python Imaging Library (PIL) [21] es una biblioteca gratuita que permite la edición de imágenes directamente desde *Python*. Soporta una variedad de formatos, incluidos los más utilizados como GIF, JPEG y PNG. Una gran parte del código está escrito en C, por cuestiones de rendimiento.

Scikit-image

Scikit-image o *skimage* [23] es un paquete de *Python* dedicado al procesamiento de imágenes y al uso de matrices *NumPy* nativas como objetos de imagen. Está disponible de forma gratuita y sin restricciones.

OpenCV

OpenCV (*Open Computer Vision*) es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Se puede aplicar a detección de movimiento, reconocimiento de objetos, reconstrucción 3D a partir de imágenes y otros.

Distutils

Distutils [20] es un paquete que proporciona soporte para construir e instalar módulos adicionales en una instalación de *Python*.

4.6. Modelos de Python

En este apartado describiremos dos modelos de *Python* que hemos utilizado con gran frecuencia.

Tkinter

Tkinter es una referencia de la biblioteca gráfica Tcl/Tk para el lenguaje de programación *Python*. Es una interfaz gráfica de usuario estándar (GUI) para *Python*.

Os

La biblioteca *os* proporciona una forma sencilla de utilizar la funcionalidad del sistema operativo. Proporciona una interfaz para utilizar los comandos del mismo, independientemente de cuál sea éste.

4.7. Git

Sistema de control de versiones distribuido de código abierto. Estos sistemas son útiles ya que nos permiten funciones como volver a un punto anterior del proyecto, a parte de tener información detallada de los cambios.

Github

Github es un servicio *online* de código abierto que nos permite alojar nuestro repositorio del proyecto usando el control de versiones *Git*. Permite la integración de varias herramientas de ayuda al desarrollo, como puede ser *ZenHub*.

ZenHub

Zenhub es una herramienta, que se integra sobre *Github*, y que nos permite llevar un mejor control del proyecto, añadiendo elementos *Scrum* a nuestro repositorio *Github*.

TortoiseGit

TortoiseGit es una cliente de control de versiones de *Git*, que nos proporciona una herramienta de escritorio para manejar nuestro repositorio en el escritorio. Nos permite utilizar todas las herramientas de *Git* en una interfaz gráfica de manera sencilla.

4.8. L^AT_EX

L^AT_EX [28] es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica.

Overleaf

Overleaf es un editor colaborativo de L^AT_EX basado en la nube que se utiliza para escribir, editar y publicar documentos científicos. Integra variedad de herramientas para desarrollar documentos con L^AT_EX.

4.9. tmux

tmux [26] es un multiplexor de terminales, es decir, permite crear, acceder y controlar varios terminales desde una sola pantalla. *tmux* puede separarse de una pantalla y continuar ejecutándose en segundo plano, pudiendo volverse a conectar posteriormente.

Hemos utilizado este programar para poder apagar el ordenador o la terminal del ordenador Alpha de la universidad mientras entrenaba. El entrenamiento podía durar unos días y de esta manera el ordenador no tenía por qué estar encendido durante todo el proceso.

4.10. Técnicas

Patrón Singleton

El patrón de diseño *Singleton* [17] (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. No se encarga de la creación de objetos en sí, sino que se enfoca en la restricción en la creación de un objeto.

El uso del patrón *Singleton* proporciona los siguientes beneficios:

- Reduce el espacio de nombres.
- Controla el acceso a la instancia única.
- Permite el refinamiento de las operaciones y la representación.
- Más flexible que las operaciones de clases.

En este trabajo se ha usado el patrón *Singleton* para asegurar que solo se crea una instancia del modelo de detección porque este nunca cambia desde que abrimos la aplicación y el proceso de cargar el modelo tarda bastante por

lo que si sólo lo cargamos la primera vez reducimos el tiempo de detección las siguientes veces.

Aspectos relevantes del desarrollo del proyecto

5.1. Introducción

En este apartado explicamos el entrenamiento ([Entrenamiento](#)) de nuestra red y las salidas que obtenemos. Comparamos los dos últimos modelos para ver cuál de ellos lo utilizamos para la aplicación de escritorio ([Comparación modelos 159 y 160](#)) y realizamos algunas pruebas al modelo seleccionado ([Pruebas modelo 159](#)).

También veremos las gráficas con los valores obtenidos tras el entrenamiento ([Gráficas](#)).

Vocabulario

Las palabras relacionadas con este apartado son:

- **Steps:** Un *step* es un paso en el entrenamiento de la red neuronal. Tras finalizar un *step* los datos de salida se utilizan para entrenar el siguiente.
- **Epochs:** Son las etapas de nuestra red neuronal, por cada *epoch* que se ejecuta se crea un modelo válido para predecir defectos en nuevas imágenes. Una *epoch* está formada por varios *steps*, puedes asignar el número que mejor te convenga, en nuestro proyecto se ejecutan 500 *steps* por *epoch*.

- **Bounding Box:** Cuadro delimitado donde se encuentran los defectos. Son cuatro valores que representan las coordenadas (x, y) de las esquinas superior izquierda e inferior derecha del cuadro delimitador.
- **AP (Average Precision):** Este término esta explicado en [3.4](#), el máximo que se puede obtener es de 1.

5.2. Entrenamiento

Para el entrenamiento de esta red se divide en tres etapas:

- **Primera etapa:** Son 40 *epochs* entrenando la cabecera de la red.
- **Segunda etapa:** Son 80 *epochs* entrenando de la etapa 4 y superiores de la red.
- **Tercera etapa:** Son 40 *epochs* entrenando todas las capas de la red.

```

525      # Training - Stage 1
526      print("Training network heads")
527      model.train(dataset_train, dataset_val,
528                     learning_rate=config.LEARNING_RATE,
529                     epochs=40,
530                     layers='heads')
531
532      # Training - Stage 2
533      # Finetune layers from ResNet stage 4 and up
534      print("Fine tune Resnet stage 4 and up")
535      model.train(dataset_train, dataset_val,
536                     learning_rate=config.LEARNING_RATE,
537                     epochs=120,
538                     layers='4+')
539
540      # Training - Stage 3
541      # Fine tune all layers
542      print("Fine tune all layers")
543      model.train(dataset_train, dataset_val,
544                     learning_rate=config.LEARNING_RATE / 10,
545                     epochs=160,
546                     layers='all')
```

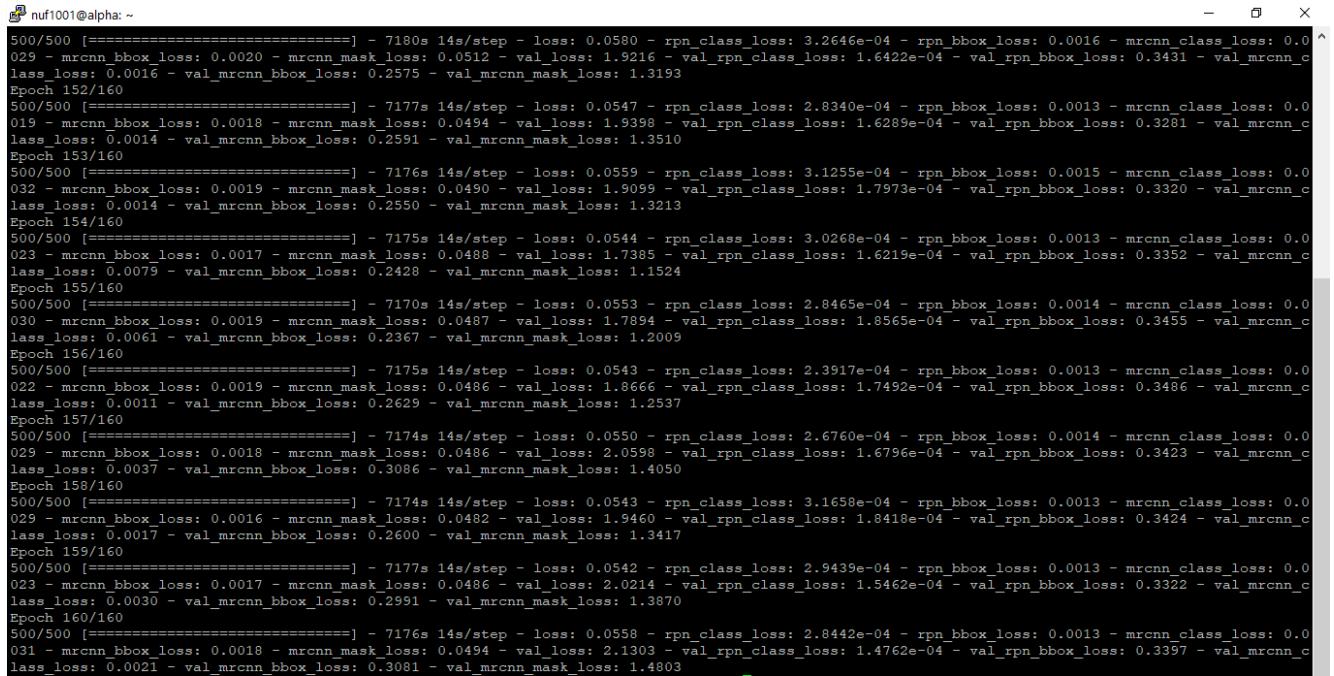
Figura 5.10: Etapas de entrenamiento del código

Esto es sólo una posible forma de entrenar la red (es la recomendada por Max Ferguson en su repositorio [\[5\]](#)), hay más formas de entrenar la red, los parámetros para seleccionar que capas quieres entrenar son:

- **heads:** Entrena la RPN, los clasificadores y los cabezales de máscara de la red.
- **all:** Todas las capas.

- **3+:** Entrena Resnet etapa 3 y superior.
- **4+:** Entrena Resnet etapa 4 y superior.
- **5+:** Entrena Resnet etapa 5 y superior.
- Una expresión regular para unir los nombres de las capas para entrenar.

Estas etapas están explicadas en [3.3](#).



```
nuf1001@alpha: ~
500/500 [=====] - 7180s 14s/step - loss: 0.0580 - rpn_class_loss: 3.2646e-04 - rpn_bbox_loss: 0.0016 - mrcnn_class_loss: 0.0
029 - mrcnn_bbox_loss: 0.0020 - mrcnn_mask_loss: 0.0512 - val_loss: 1.9216 - val_rpn_class_loss: 1.6422e-04 - val_
rpn_bbox_loss: 0.3431 - val_mrcnn_c
lass_loss: 0.0016 - val_mrcnn_bbox_loss: 0.2575 - val_mrcnn_mask_loss: 1.3193
Epoch 152/160
500/500 [=====] - 7177s 14s/step - loss: 0.0547 - rpn_class_loss: 2.8340e-04 - rpn_bbox_loss: 0.0013 - mrcnn_class_loss: 0.0
019 - mrcnn_bbox_loss: 0.0018 - mrcnn_mask_loss: 0.0494 - val_loss: 1.9398 - val_rpn_class_loss: 1.6289e-04 - val_
rpn_bbox_loss: 0.3281 - val_mrcnn_c
lass_loss: 0.0014 - val_mrcnn_bbox_loss: 0.2591 - val_mrcnn_mask_loss: 1.3510
Epoch 153/160
500/500 [=====] - 7176s 14s/step - loss: 0.0559 - rpn_class_loss: 3.1255e-04 - rpn_bbox_loss: 0.0015 - mrcnn_class_loss: 0.0
032 - mrcnn_bbox_loss: 0.0019 - mrcnn_mask_loss: 0.0490 - val_loss: 1.9099 - val_rpn_class_loss: 1.7973e-04 - val_
rpn_bbox_loss: 0.3320 - val_mrcnn_c
lass_loss: 0.0014 - val_mrcnn_bbox_loss: 0.2550 - val_mrcnn_mask_loss: 1.3213
Epoch 154/160
500/500 [=====] - 7175s 14s/step - loss: 0.0544 - rpn_class_loss: 3.0268e-04 - rpn_bbox_loss: 0.0013 - mrcnn_class_loss: 0.0
023 - mrcnn_bbox_loss: 0.0017 - mrcnn_mask_loss: 0.0488 - val_loss: 1.7385 - val_rpn_class_loss: 1.6219e-04 - val_
rpn_bbox_loss: 0.3352 - val_mrcnn_c
lass_loss: 0.0079 - val_mrcnn_bbox_loss: 0.2428 - val_mrcnn_mask_loss: 1.1524
Epoch 155/160
500/500 [=====] - 7170s 14s/step - loss: 0.0553 - rpn_class_loss: 2.8465e-04 - rpn_bbox_loss: 0.0014 - mrcnn_class_loss: 0.0
030 - mrcnn_bbox_loss: 0.0019 - mrcnn_mask_loss: 0.0487 - val_loss: 1.7894 - val_rpn_class_loss: 1.8565e-04 - val_
rpn_bbox_loss: 0.3455 - val_mrcnn_c
lass_loss: 0.0061 - val_mrcnn_bbox_loss: 0.2367 - val_mrcnn_mask_loss: 1.2099
Epoch 156/160
500/500 [=====] - 7175s 14s/step - loss: 0.0543 - rpn_class_loss: 2.3917e-04 - rpn_bbox_loss: 0.0013 - mrcnn_class_loss: 0.0
022 - mrcnn_bbox_loss: 0.0019 - mrcnn_mask_loss: 0.0486 - val_loss: 1.8666 - val_rpn_class_loss: 1.7492e-04 - val_
rpn_bbox_loss: 0.3486 - val_mrcnn_c
lass_loss: 0.0011 - val_mrcnn_bbox_loss: 0.2629 - val_mrcnn_mask_loss: 1.2537
Epoch 157/160
500/500 [=====] - 7174s 14s/step - loss: 0.0550 - rpn_class_loss: 2.6760e-04 - rpn_bbox_loss: 0.0014 - mrcnn_class_loss: 0.0
029 - mrcnn_bbox_loss: 0.0018 - mrcnn_mask_loss: 0.0486 - val_loss: 2.0598 - val_rpn_class_loss: 1.6796e-04 - val_
rpn_bbox_loss: 0.3423 - val_mrcnn_c
lass_loss: 0.0037 - val_mrcnn_bbox_loss: 0.3086 - val_mrcnn_mask_loss: 1.4050
Epoch 158/160
500/500 [=====] - 7174s 14s/step - loss: 0.0543 - rpn_class_loss: 3.1658e-04 - rpn_bbox_loss: 0.0013 - mrcnn_class_loss: 0.0
029 - mrcnn_bbox_loss: 0.0016 - mrcnn_mask_loss: 0.0482 - val_loss: 1.9460 - val_rpn_class_loss: 1.8418e-04 - val_
rpn_bbox_loss: 0.3424 - val_mrcnn_c
lass_loss: 0.0017 - val_mrcnn_bbox_loss: 0.2600 - val_mrcnn_mask_loss: 1.3417
Epoch 159/160
500/500 [=====] - 7177s 14s/step - loss: 0.0542 - rpn_class_loss: 2.9439e-04 - rpn_bbox_loss: 0.0013 - mrcnn_class_loss: 0.0
023 - mrcnn_bbox_loss: 0.0017 - mrcnn_mask_loss: 0.0486 - val_loss: 2.0214 - val_rpn_class_loss: 1.5462e-04 - val_
rpn_bbox_loss: 0.3322 - val_mrcnn_c
lass_loss: 0.0030 - val_mrcnn_bbox_loss: 0.2991 - val_mrcnn_mask_loss: 1.3870
Epoch 160/160
500/500 [=====] - 7176s 14s/step - loss: 0.0558 - rpn_class_loss: 2.8442e-04 - rpn_bbox_loss: 0.0013 - mrcnn_class_loss: 0.0
031 - mrcnn_bbox_loss: 0.0018 - mrcnn_mask_loss: 0.0494 - val_loss: 2.1303 - val_rpn_class_loss: 1.4762e-04 - val_
rpn_bbox_loss: 0.3397 - val_mrcnn_c
lass_loss: 0.0021 - val_mrcnn_bbox_loss: 0.3081 - val_mrcnn_mask_loss: 1.4803
```

Figura 5.11: Últimas *epochs* del entrenamiento

Se guarda un modelo por cada *epoch*, es decir, al final del entrenamiento tendremos 160 modelos con 500 *steps* por cada una, 80.000 *steps* en total.

En la figura [5.11](#) podemos ver la salida por pantalla del final del entrenamiento, las últimas 10 *epochs* que se han realizado. Se va ha hacer incapié en las dos últimas *epochs* la 159 y la 160 para saber cuál de los dos modelos es el mejor entrenado.

Los 5 *losses* que vemos en la figura [5.11](#) representan:

- **rpn_class_loss:** Qué tan bien la red de propuestas de la región separa el fondo con los objetos, es decir, qué tan bien se diferencia el fondo (*background* - BG) de los posibles defectos.

- **rpn_bbox_loss:** Qué tan bien los RPN localizan objetos.
- **mrcnn_bbox_loss:** Qué tan bien *Mask RCNN* localiza objetos.
- **mrcnn_class_loss:** Qué tan bien *Mask RCNN* reconoce cada clase de objeto.
- **mrcnn_mask_loss:** Qué tan bien *Mask RCNN* segmenta los objetos.

Eso hace una mayor pérdida:

- **loss:** Una combinación de todos los *losses* anteriores.

Todos estos *losses* se calculan en el conjunto de datos de entrenamiento. Y los *losses* para el conjunto de datos de validación son las que comienzan con “val”.

Comparación modelos 159 y 160

```
Running GDXray evaluation on 20 images.
Image 26 AP: 1.0
Image 10 AP: 0.75
Image 162 AP: 1.0
Image 76 AP: 1.0
Image 86 AP: 1.0
Image 1 AP: 1.0
Image 127 AP: 1.0
Image 4 AP: 0.5
Image 14 AP: 1.0
Image 33 AP: 1.0
Image 157 AP: 0.6666666865348816
Image 51 AP: 1.0
Image 60 AP: 0.25
Image 144 AP: 0.5
Image 8 AP: 1.0
Image 158 AP: 1.0
Image 151 AP: 1.0
Image 146 AP: 0.5
Image 87 AP: 1.0
Image 107 AP: 1.0
Got AP=0.8583333343267441 using 20 images.
```

Figura 5.12: Evaluación con el modelo número 159

```
Running GDXray evaluation on 20 images.
Image 162 AP: 1.0
Image 138 AP: 1.0
Image 100 AP: 1.0
Image 117 AP: 1.0
Image 148 AP: 0.0
Image 3 AP: 1.0
Image 141 AP: 1.0
Image 94 AP: 1.0
Image 96 AP: 1.0
Image 132 AP: 1.0
Image 48 AP: 0.666666865348816
Image 19 AP: 1.0
Image 30 AP: 1.0
Image 98 AP: 1.0
Image 18 AP: 0.666666865348816
Image 160 AP: 0.25
Image 22 AP: 0.0
Image 118 AP: 1.0
Image 91 AP: 1.0
Image 154 AP: 0.9821428656578064
Got AP=0.8282738119363785 using 20 images.
```

Figura 5.13: Evaluación con el modelo número 160

Si comparamos los datos de la *epoch* 160 y 159 que aparecen en las figuras 5.11, 5.12 y 5.13 vemos que la penúltima tiene mejores datos, es decir, el modelo 159 tiene un *loss* menor que el modelo 160, ver en la tabla 5.3.

Como vemos en la tabla 5.3 los datos los *losses* de la *epoch* 159 son mejores que los de la 160, pero vamos a hacer algunas pruebas para corroborarlo.

La diferencia de la mAP [24] que se observa en las figuras 5.12 y 5.13 no es muy grande, hay ocasiones en las que la evaluación del modelo 160 ha sido mejor que el del 159, pero predominan las pruebas con mejor AP en el modelo 159.

Tipo de datos	<i>Epoch</i> 159	Comparación	<i>Epoch</i> 160
val_loss	2.0214	<	2.1303
val_rpn_class_loss	1.5462e-04	<	1.4762e-04
val_rpn_bbox_loss	0.3322	<	0.3397
val_mrcnn_class_loss	0.0030	>	0.0021
val_mrcnn_bbox_loss	0.2991	<	0.3081
val_mrcnn_mask_loss	1.3870	<	1.4803
mAP	0.8583	>	0.8283

Tabla 5.3: Comparación de las dos últimas *epochs***Imagen C0001_0018:**Figura 5.14: Imagen C0001_0018 con los *Bounding Box* del modelo 159

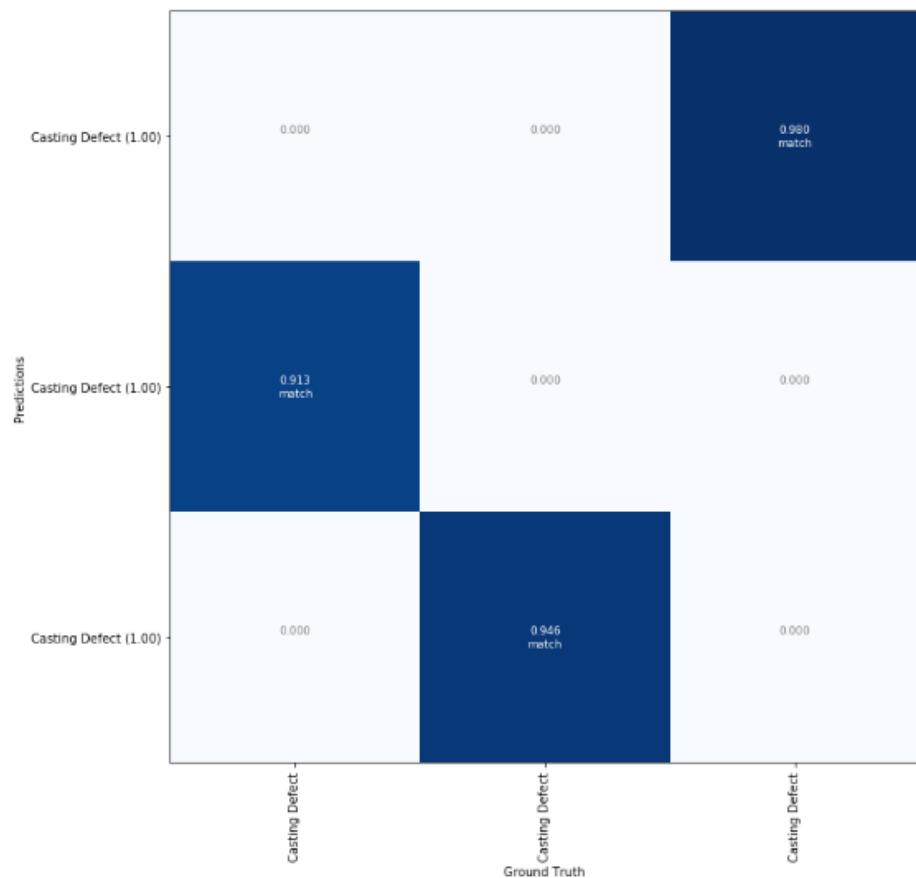


Figura 5.15: Cuadrícula de la imagen C0001_0018 de objetos de verdad y sus predicciones del modelo 159



Figura 5.16: Imagen C0001_0018 con los *Bounding Box* del modelo 160

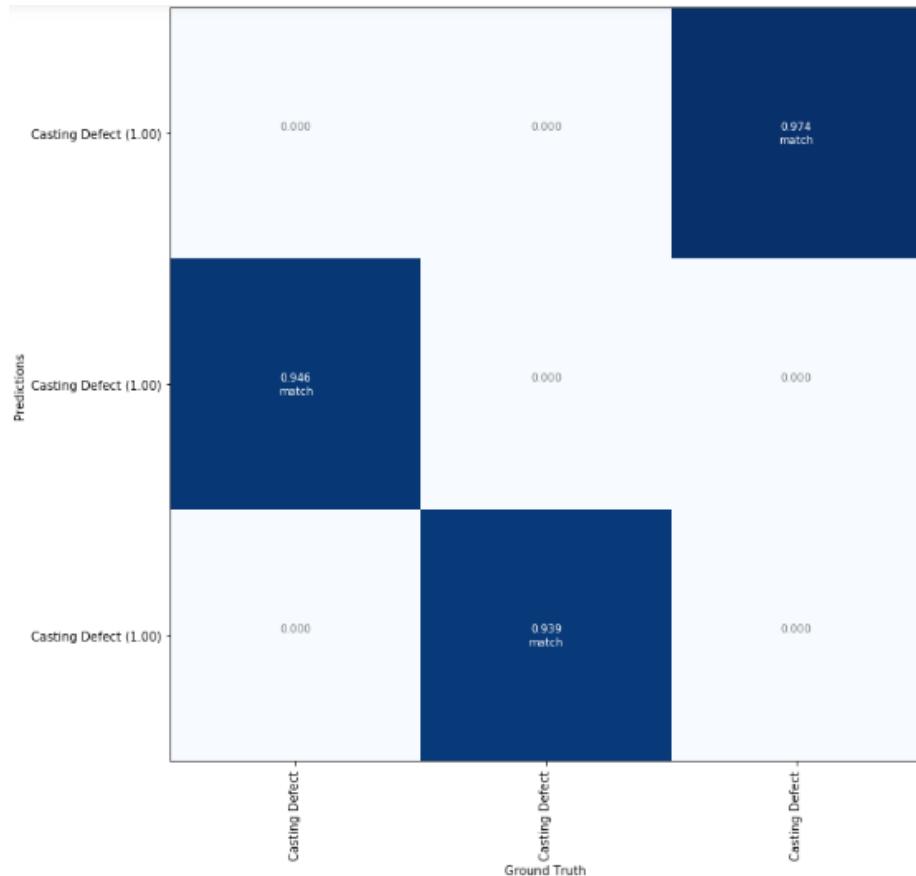


Figura 5.17: Cuadrícula de la imagen C0001_0018 de objetos de verdad y sus predicciones del modelo 160

Las figuras 5.14 y 5.16 representan la imagen C0001_0018 con los defectos detectados por los modelos 159 y 160 respectivamente. Las figuras 5.15 y 5.17 son las cuadriculas de los *ground truth* y las predicciones hechas por los modelos entrenados. Las columnas representan los defectos reales que tiene la imagen y las filas los defectos detectados por el modelo, en este caso había tres defectos y los dos modelos han detectado los tres.

Con valores de la figura 5.15 vemos que el modelo 159 tiene una tasa de aciertos de un 98 % en el primer defecto, un 91.3 % en el segundo y un 94.6 % en el tercero (media de 94.63 %). Y en la figura 5.17 vemos que el modelo 160 tiene una tasa de aciertos de un 97.4 % para el primer defecto, un 94.6 % en el segundo y un 93.9 % en el tercero (media de 95.3 %). Estos valores son la coincidencia/superposición de las predicciones del modelo y del *ground truth*. Salen de los decimales que aparecen en los cuadros de las

figuras 5.15 y 5.17.

Imagen C0001_0004:

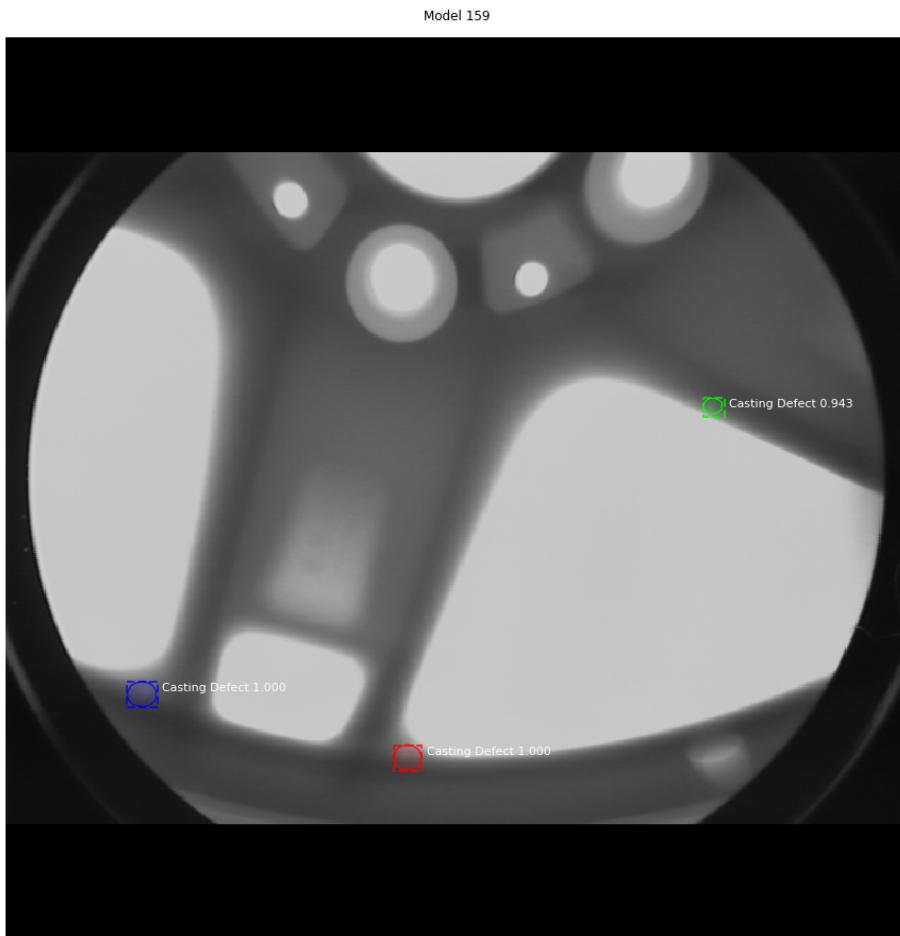


Figura 5.18: Imagen C0001_0004 con los *Bounding Box* del modelo 159

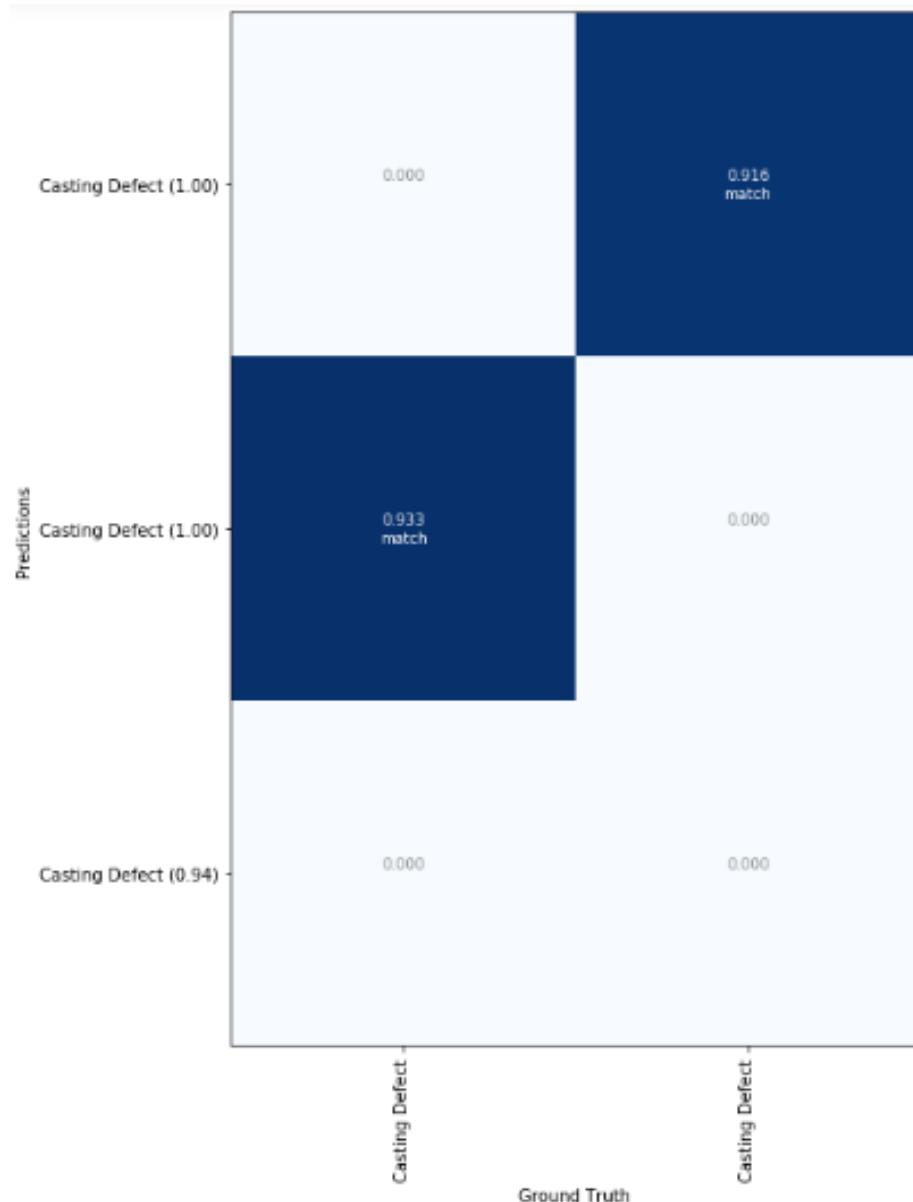


Figura 5.19: Cuadrícula de la imagen C0001_0004 de objetos de verdad y sus predicciones del modelo 159



Figura 5.20: Imagen C0001_0004 con los *Bounding Box* del modelo 160

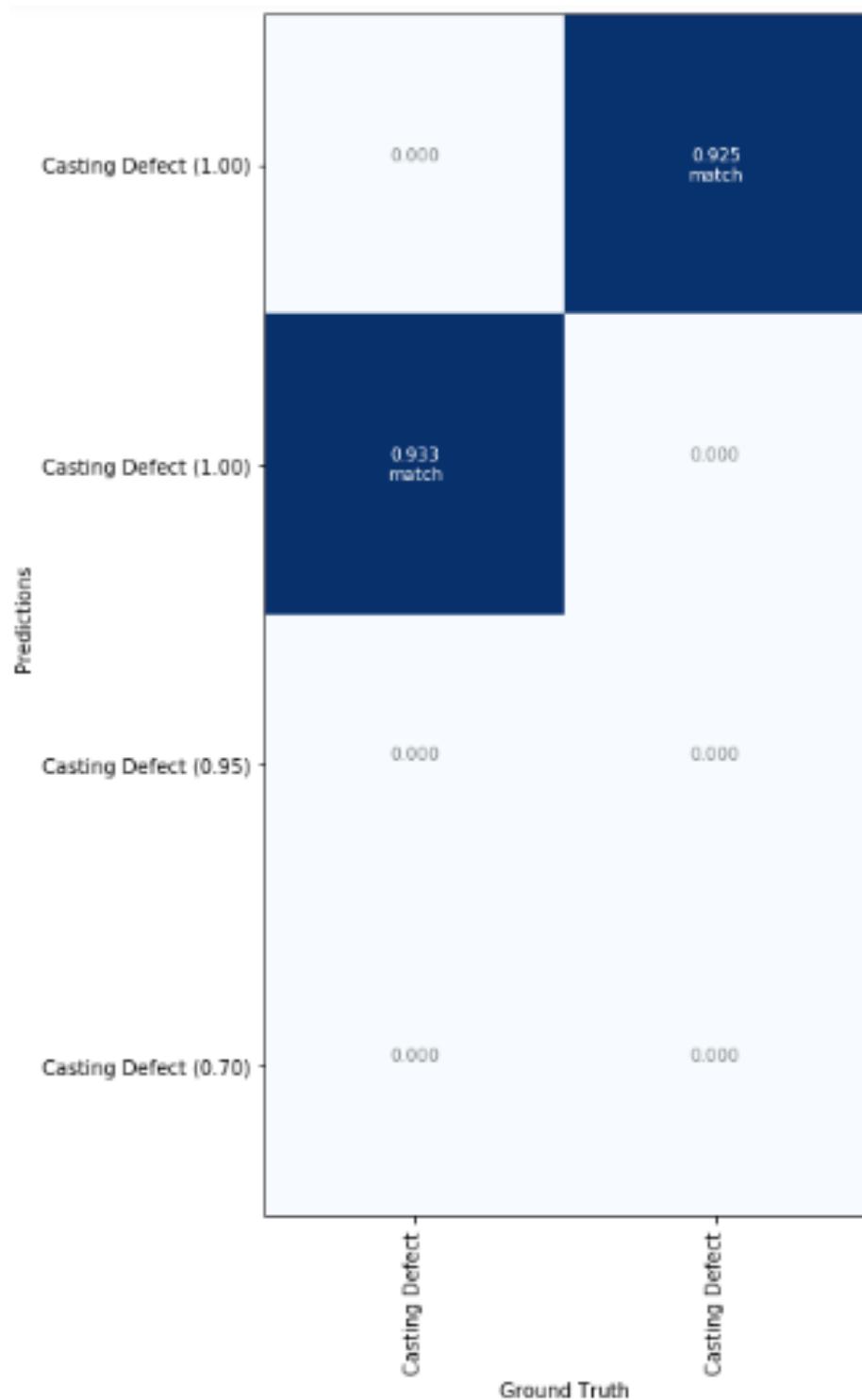


Figura 5.21: Cuadrícula de la imagen C0001_0004 de objetos de verdad y sus predicciones del modelo 160

Las figuras 5.18 y 5.20 representan la imagen C0001_0004 con los defectos detectados por los modelos 159 y 160 respectivamente. Y las figuras 5.19 y 5.21 son las cuadriculas de los *ground truth* y las predicciones hechas por los modelos entrenados. Las columnas representan los defectos reales que tiene la imagen y las filas los defectos detectados por el modelo, en este caso había dos defectos y el modelo 159 ha detectado tres y el modelo 160 cuatro.

Con valores de la figura 5.19 vemos que el modelo 159 tiene una tasa de aciertos de un 91.6 % en el primer defecto y un 93.3 % en el segundo (media de 61.63 %). Y en la figura 5.21 vemos que el modelo 160 tiene una tasa de aciertos de un 92.5 % para el primer defecto y un 93.3 % en el segundo (media de 46.45 %).

En esta imagen sólo hay dos defectos que son los que se encuentran en la parte inferior izquierda de la imagen. El modelo 159 ha detectado 3, el tercero lo ha detectado en el centro a la derecha, pero el modelo 160 ha detectado uno más en la parte inferior derecha. En estos dos errores vemos que no tienen asignado en un 1.000, esto indica que no es seguro al 100 % de que ahí se encuentre un error. Esto no indica que siempre que el valor no sea 1.000 no vaya a haber un error, sino que no que el programa no puede confirmar al 100 % que eso sea un defecto o parte de la imagen.

En conclusión, teniendo en cuenta los datos de la tabla 5.3, y la cantidad de defectos detectados por los modelos en la imagen C0001_0004 (aunque los dos modelos han detectados defectos de más, el modelo 159 sólo ha detectado uno de más en cambio del modelo 160 han sido dos) he decidido seleccionar el modelo 159 para la app.

Pruebas modelo 159

Evaluaremos la predicción de este modelo con tres pasos:

1. *Region Proposal Network.*
 - a) Objetivos RPN.
2. Clasificación de propuestas y detección.
 - a) Clasificación de propuestas.
 - b) Detección paso a paso.
3. Generación de máscaras.
 - a) Objetivos de las máscaras.
 - b) Máscaras predichas.

Añadiré un apartado con visualizaciones adicionales del mapa de funciones de la red troncal, los histogramas de *bounding box* de el RPN y la distribución de las coordenadas y , x de las propuestas generadas.

Estas pruebas se realizarán con la última imagen mostrada en el apartado anterior, la figura 5.18

1. *Region Proposal Network*

El *Region Proposal Network* (RPN) ejecuta un clasificador binario⁹ en muchos cuadros (anclas¹⁰) sobre la imagen y devuelve puntuaciones de objeto/no objeto. Las anclas con un alto puntuación de “objetividad” (anclas positivas) se pasan a la etapa dos para su clasificación.

Cuando los anclajes positivos no cubren los objetos por completo el RPN también retrocede un refinamiento que se aplicará a los anclajes para cambiarlo y cambiar su tamaño un poco a los límites correctos del objeto.

1.a Objetivos RPN

Los objetivos RPN son los valores de entrenamiento para el RPN. Para generar los objetivos, comenzamos con una cuadrícula de anclas que cubren la imagen completa a diferentes escalas, y luego calculamos la IoU de las anclas con el objeto de *ground truth*. Los anclajes positivos son aquellos que tienen un $IoU \geq 0.7$ con cualquier objeto de verdad fundamental, y los anclajes negativos son aquellos que no cubren ningún objeto en más de 0.3 IoU. Las anclas intermedias (es decir, cubren un objeto por $IoU \geq 0.3$ pero < 0.7) se consideran neutrales y se excluyen del entrenamiento.

Para entrenar el RPN, también calculamos el cambio y el redimensionado necesarios para que el ancla cubra completamente el objeto de verdad del suelo.

⁹En clasificador binario es el que podemos dividir en dos clases: “Positiva” y “Negativa”.

¹⁰Múltiples regiones posibles basadas en uniones espaciales de dimensiones fijas.

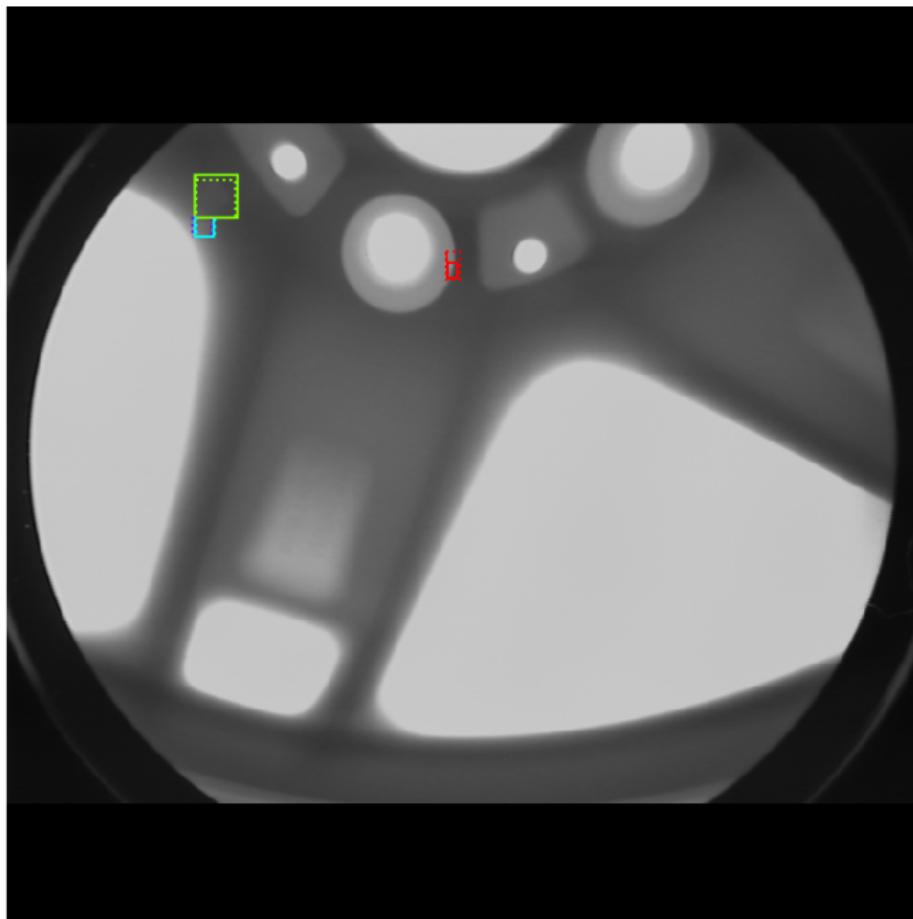


Figura 5.22: Imagen con anclajes positivos

En la figura 5.22 se representan los anclajes positivos antes del refinamiento con la línea de puntos y los anclajes después del refinamiento con la línea sólida.

2. Clasificación de propuestas y detección

Este paso se toman las propuestas de región de la RPN y se clasifican.

2.a Clasificación de propuestas

Ejecute los cabezales clasificadores en las propuestas para generar probabilidades de clase y regresiones de los *bounding boxes*.

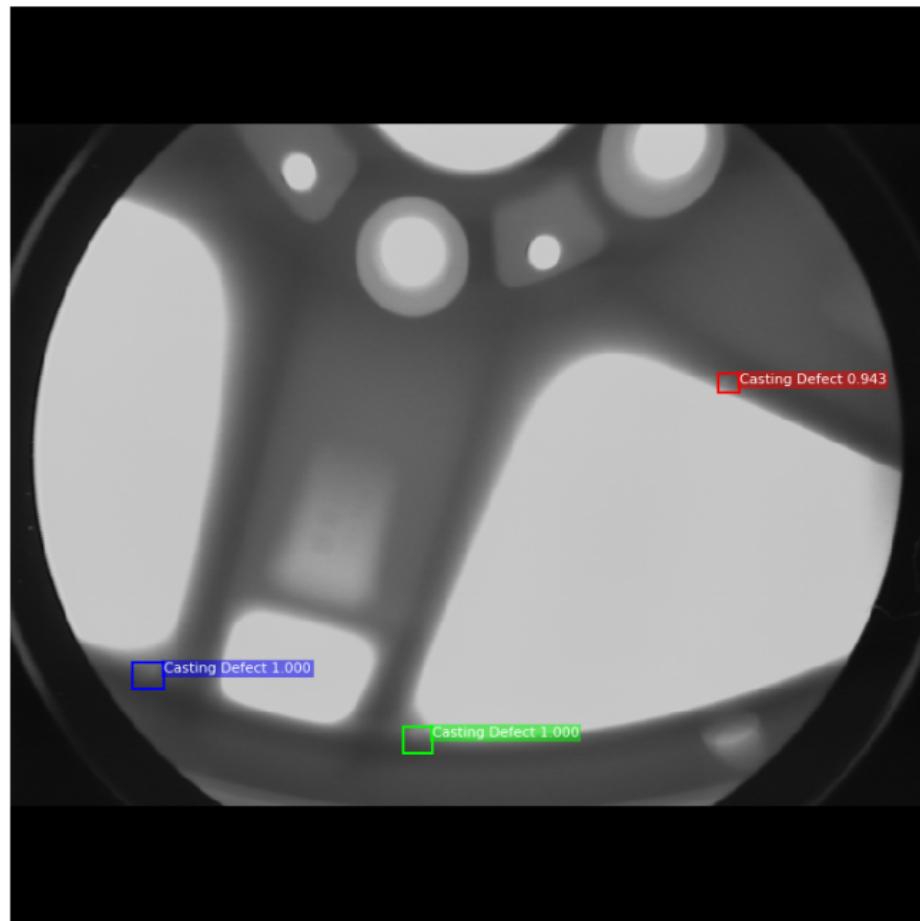


Figura 5.23: Clasificación de propuestas

2.b Detección paso a paso

Mostraremos una muestra aleatoria de propuestas (200). Las propuestas clasificadas como *background* están representadas con líneas de puntos, y el resto muestra su clase y puntuación de confianza.

En este caso tenemos 993 propuestas para *Background* y 7 para *Casting Defect*.

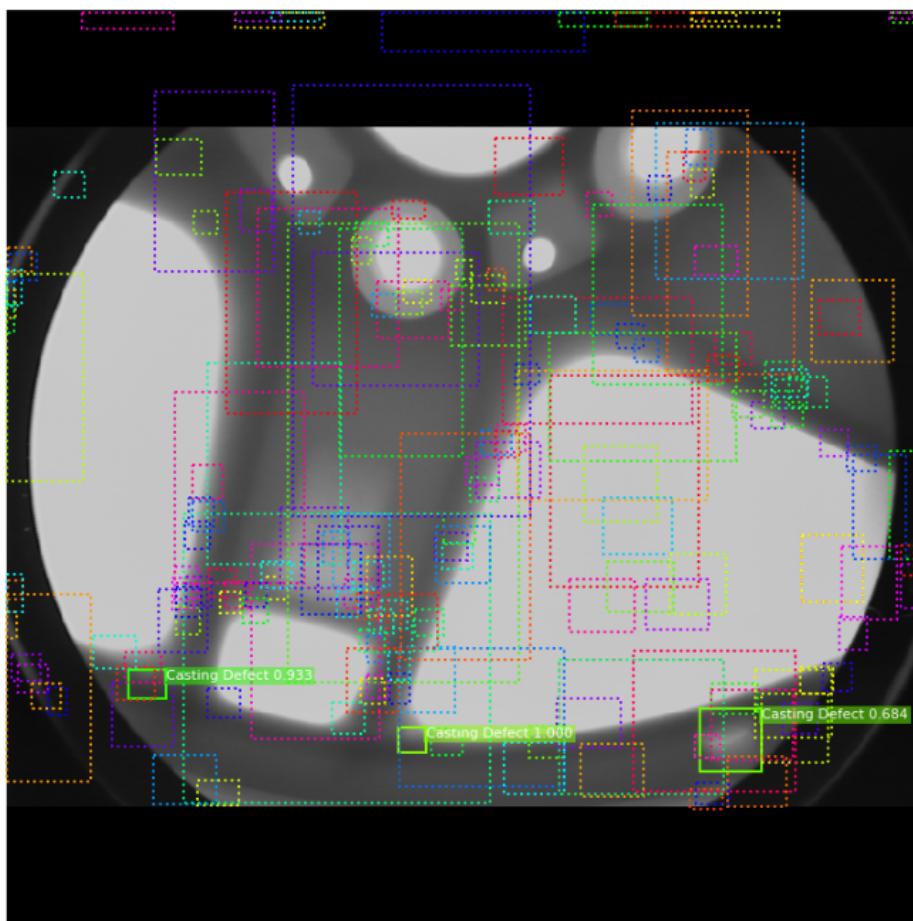


Figura 5.24: ROI antes del refinamiento

Ahora aplicamos el refinamiento del *bounding boxes*. Ahora solo se visualizan las propuestas positivas.

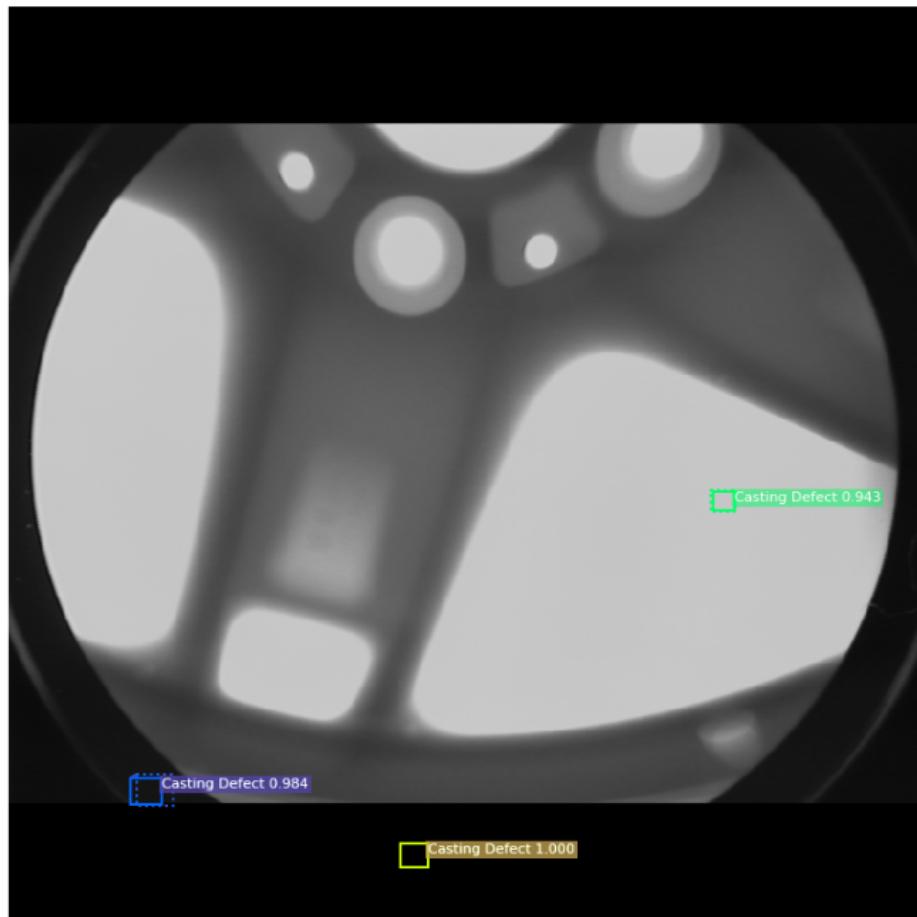


Figura 5.25: ROI depués del refinamiento

Por último, se realiza una supresión no máxima (*Non Max Suppression* - NMS) que es una técnica que filtra las propuestas en función de algunos criterios como la puntuación de confianza más alta o el IoU. Esto devuelve indicaciones de *boxes* guardadas. Los *boxes* que se devuelven son los que tienen un IoU por encima del umbral (el umbral es 0.3).

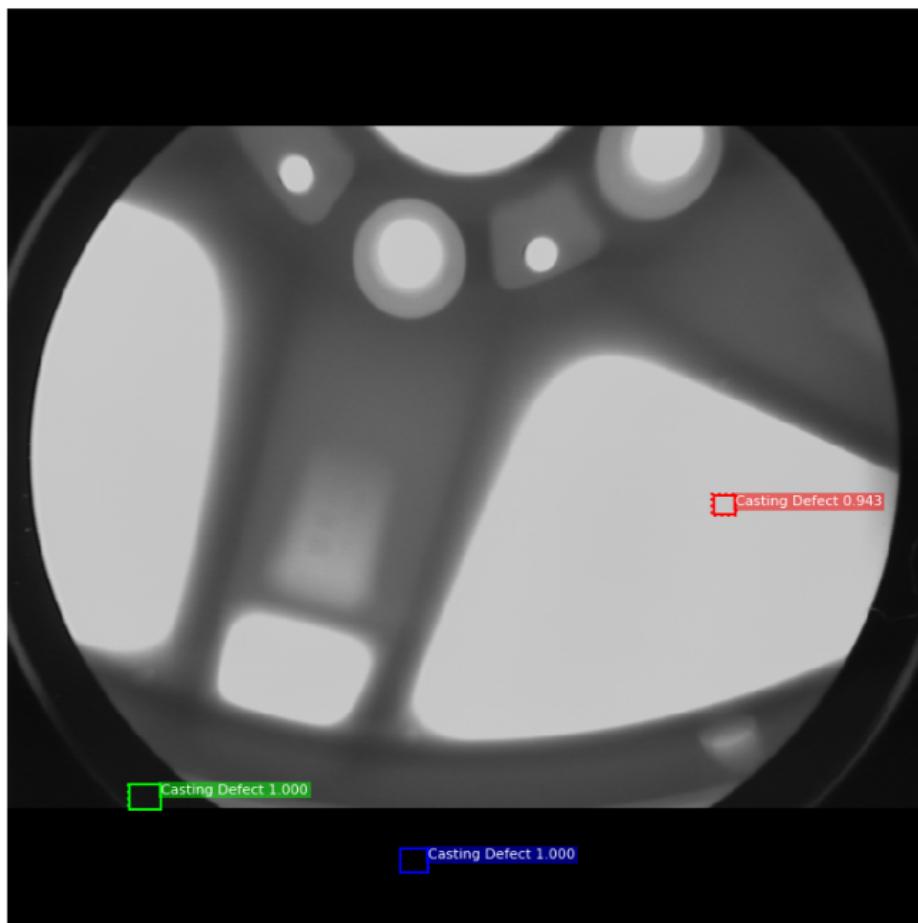


Figura 5.26: Detecciones después de NMS

3. Generación de máscaras

Esta etapa toma las detecciones (cuadros delimitadores refinados e ID de clase) de la capa anterior y ejecuta el cabezal de la máscara para generar máscaras de segmentación para cada instancia.

3.a Objetivos de las máscaras

Estos son los objetivos de entrenamiento para la rama de la máscara¹¹.

¹¹El fondo es negro para que se ve el contraste ya que casi toda la imagen es casi blanca.



Figura 5.27: Objetivos de entrenamiento para la rama de la máscara

3.b Máscaras predichas



Figura 5.28: Máscaras predichas

Gráficas

Las gráfica 5.29 a 5.34, que se muestran a continuación, representan la evolución de la figura de mérito de “pérdida” (*loss*) en función del número de pasos (*steps*). las variables asociadas a estas gráficas están definidas en la Sección 5.2.

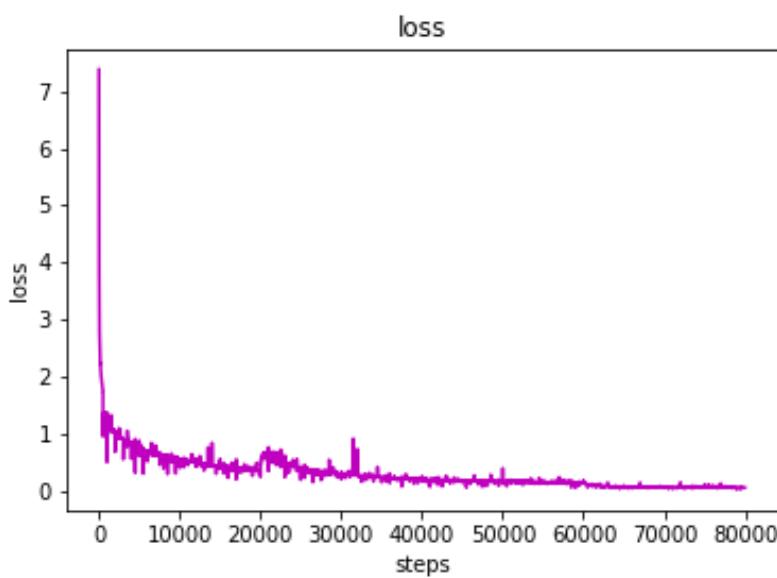


Figura 5.29: Gráfica del loss de las 160 *epochs*

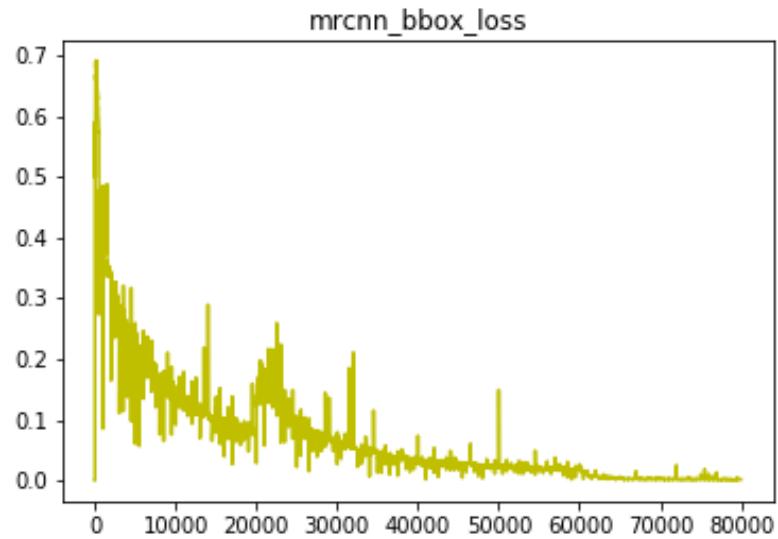


Figura 5.30: Gráfica del mrcnn bbox loss de las 160 *epochs*

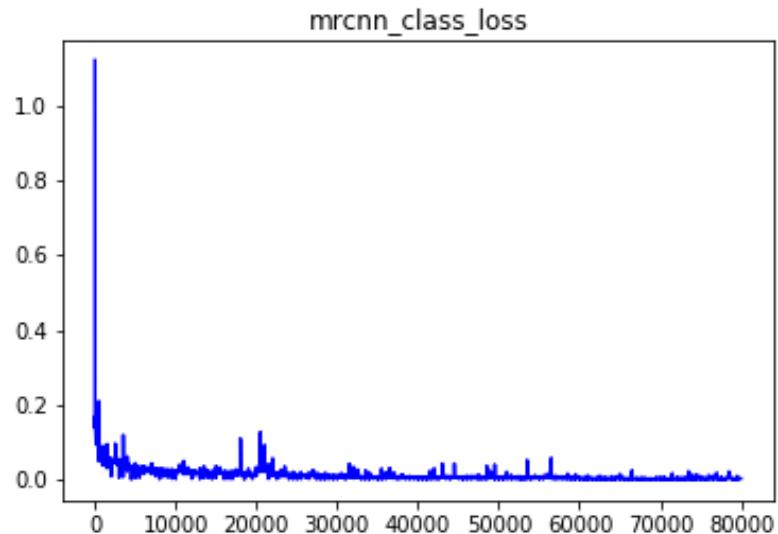


Figura 5.31: Gráfica del mrcnn class loss de las 160 *epochs*

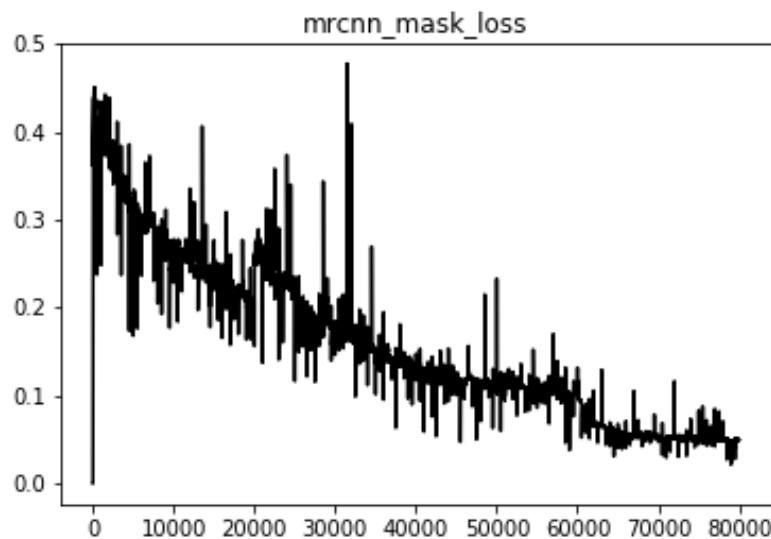


Figura 5.32: Gráfica del mrcnn mask loss de las 160 *epochs*

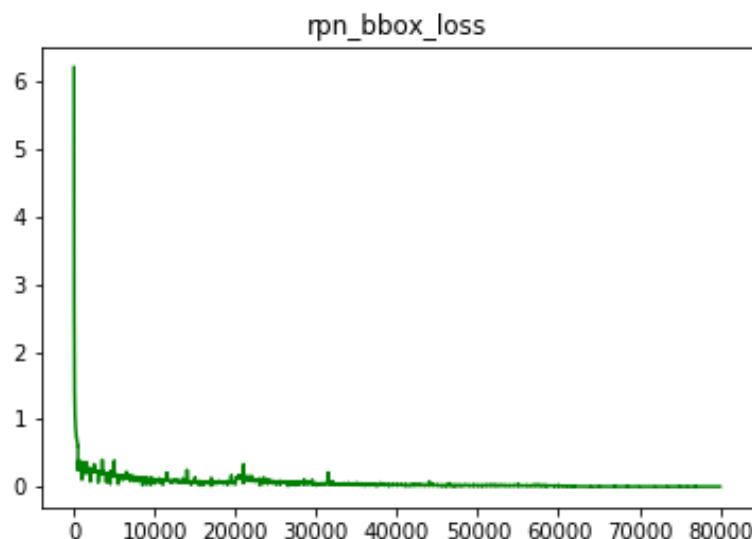


Figura 5.33: Gráfica del rpn bbox loss de las 160 *epochs*

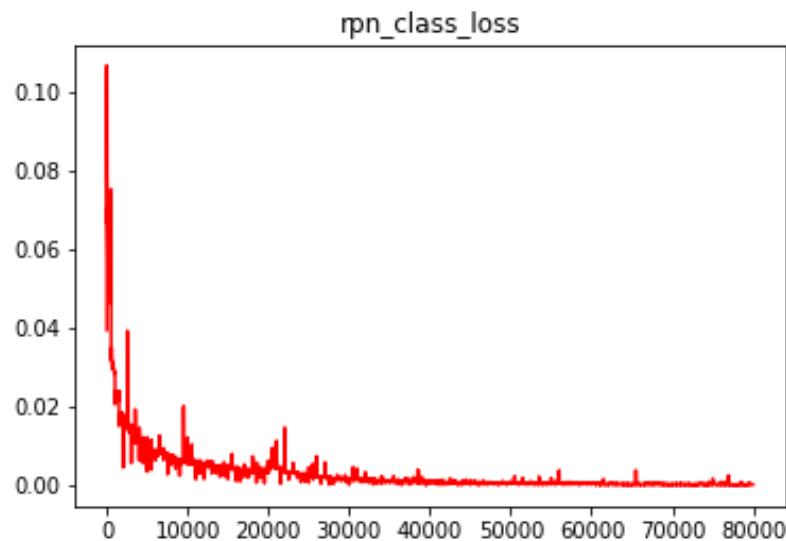


Figura 5.34: Gráfica del rpn class loss de las 160 *epochs*

Los picos más significativos son los que salen cada 500 *steps*. Ese es el momento en el que se termina un *epoch* y empieza una nueva. Al principio hay una mejora bastante significativa en los datos, pero al final mejora más lentamente.

Para guardar estos datos he añadido en el archivo `generic_utils.py` unas pocas líneas de código que me guardaban los valores en un archivo `.csv`. Este archivo se encuentra en `\Anaconda3\envs\defect-detection\Lib\site-packages\keras\utils`.

```
382     archivo = open('graficas.csv', 'a')
383
384     for k in self.unique_values:
385         info += ' - %s:' % k
386         if isinstance(self.sum_values[k], list):
387             avg = np.mean(
388                 self.sum_values[k][0] / max(1, self.sum_values[k][1]))
389             if abs(avg) > 1e-3:
390                 info += ' %.4f' % avg
391             else:
392                 info += ' %.4e' % avg
393         else:
394             info += ' %s' % self.sum_values[k]
395
396         archivo.write(str(avg))
397         archivo.write(',')
398         archivo.write('\n')
399         archivo.close()
```

Figura 5.35: Código para gradar los valores de las gráficas

Como vemos en la figura 5.35 las líneas añadidas no cambian o modifican la funcionalidad del código. Este archivo no va a tener estas líneas, ya que es un archivo del entorno que se descarga en su creación.

Trabajos relacionados

En este apartado explicaremos algunos trabajos y conferencias relacionadas a nuestro proyecto.

6.1. Artículos científicos

Automatic Localization of Casting Defects with Convolutional Neural Networks

Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, Kincho H. Law publicaron un artículo [6] donde propusieron la detección de defectos en piezas de fundición de metal mediante la utilización de redes neuronales convolucionales (CNN). Este tipo de redes han mostrado recientemente un gran rendimiento tanto en tareas de clasificación de imágenes como de localización.

El éste artículo se comparan diferentes arquitecturas de CNN para localizar los defectos. Y utilizan el aprendizaje de transferencia ara permitir que los modelos de localización CNN se capaciten en un conjunto de datos relativamente pequeño.

En un enfoque alternativo, entrenaen un modelo de clasificación de defectos en una serie de imágenes de defectos y luego usan un método clasificador de ventana deslizante¹² para desarrollar un modelo de localización simple.

¹²Este método consiste en ir escaneando una imagen mediante dicha ventana. A cada desplazamiento el clasificador predecirá ahí hay un defecto o no. Al escanear la imagen completa, esta se reduce a una cierta escala para continuar el escaneo, y realiza este proceso continuamente hasta que la imagen escaneada sea menor que la ventana de deslizamiento.

Comparan la precisión de localización y el rendimiento computacional de cada técnica.

Small Defect Detection Using Convolutional Neural Network Features and Random Forests

Escrito por Xinghui Dong, Chris J. Taylor y Tim F. Cootes [4]. El objetivo de este artículo es etiquetar los píxeles correspondientes a una pequeñas anomalías en una región con imágenes con un mínimo de falsos positivos, importante en aplicaciones como la inspección industrial. Su enfoque es ejecutar un clasificador de ventana deslizante sobre la imagen.

Las redes totalmente convolucionales (*Fully Convolutional Networks - FCN*) recientes, como U-Net, se pueden entrenar para identificar píxeles correspondientes a anomalías dado un conjunto de entrenamiento adecuado, en este artículo muestran que se pueden obtener mejores resultados reemplazando la capa final de la red con un *Random Forest* (RF) utilizando características muestradas de las capas de red anteriores.

También demuestran que, en lugar de limitar la umbral de la imagen de probabilidad resultante para identificar defectos, es mejor calcular las regiones extremas máximamente estables (*Maximally Stable Extremal Regions - MSER*).

6.2. Conferencias

Imbalanced Learning Ensembles for Defect Detection in X-Ray Images

Conferencia hecha por José Francisco Díez Pastor, César García Osorio, Víctor Barbero García y Alan Blanco Álamo en Amsterdam, Los Países Bajos en el 2013 [19].

Las imágenes utilizadas en este trabajo son muy variables (varias piezas diferentes, diferentes vistas, variabilidad introducida por el proceso de inspección, como el posicionamiento de la pieza). Debido a esta variabilidad, optaron por la técnica de ventana deslizante, un enfoque basado en la minería de datos.

Tuvieron un enfoque especial en el aprendizaje de conjuntos de dato no balanceados y llevaron a cabo experimentos con varios tamaños de ventana, varios algoritmos de selección de características y diferentes algoritmos de

clasificación. En los resultados podemos ver que el ensacado logró resultados significativamente mejores que los árboles de decisión por sí mismos.

Segmentación de defectos en piezas de fundido usando umbrales adaptativos y ensembles

Conferencia hecha por José Francisco Díez Pastor, Alvar Arnaiz González, César García Osorio y Juan José Rodríguez en Zaragoza, España en el 2014 [18].

La conferencia se centra en el desarrollo de nuevos algoritmos de construcción de *ensembles* (la combinación de varios clasificadores o regresores), sobre todo haciendo hincapié a las técnicas de incremento de la diversidad en *ensembles* homogéneos (cuando todos los miembros están construidos usando la misma técnica).

En la primera parte se presenta un estudio de los métodos más representativos de las distintas técnicas de construcción de *ensembles*, aprendizaje en conjuntos desequilibrados y breve nociones sobre validación experimental.

La segunda parte se divide en tres bloques. El primer bloque se explora como inyectar aleatoriedad en el propio algoritmo del clasificador base, para ello utilizan la fase constructiva de la metaheurística GRASP. Esta técnica, llamada “*GRASP Forest*” ha sido utilizada con éxito en árboles de clasificación y árboles de regresión. Se desarrolla un segundo método “*GAR-Forest*”, *GRASP with annealed randomness*, que parte de la idea de que los nodos que más influencia tienen en la correcta clasificación de las instancias son los nodos inferiores y hojas, mientras que los nodos que más afectan la estructura global del árbol (y por lo tanto la diversidad) son la raíz y los nodos superiores.

En un segundo bloque se aborda el problema de los *ensembles* para conjuntos desequilibrados. Se presenta un método llamado “*Random Balance*”, basado en la idea de variar aleatoriamente las proporciones entre las clases, confiando en esta heurística, se elimina la necesidad de ajustar parámetros, a la vez que se aumenta la diversidad del *ensemble*.

En la última parte aplican estas técnicas a la solución de varias aplicaciones reales.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusión

Respecto a los requisitos del proyecto que teníamos inicialmente considero que sí que se han llegado a cumplir en su mayor parte. Teniendo en cuenta que este proyecto ha sido un proyecto de investigación, hay objetivos que han podido cambiar. Los objetivos de realizar una investigación extensa y realizar una aplicación base de ejemplo se han cumplido.

El haber utilizado el lenguaje *Python* para este proyecto ha sido una ventaja en sí. Tanto para el desarrollo como para la distribución de las clases y carpetas.

En este proyecto se han realizado una serie de experimentos sobre el análisis de imágenes y la detección de objetos en ellas bastante extensos. Teniendo una gran conjunto de datos se han podido realizar muchos experimentos.

El proyecto ha abarcado gran parte de los conocimientos adquiridos durante el grado. Además, ha requerido el aprendizaje de muchos otros como la segmentación de imágenes, OpenCV, Spyder, etc.

Se han utilizado un gran número de tecnologías nuevas. Éstas han ayudado a mejorar la calidad de los procesos intermedios que se han realizado hasta llegar al producto final. Algunas han supuesto una carga importante, como la documentación, pero todo lo aprendido será muy útil en proyectos futuros.

Gracias a la parte de investigación durante la documentación se ha aprendido a familiarizarse con la búsqueda y lectura de artículos científico.

7.2. Líneas de trabajo futuras

En esta sección describimos algunos posibles caminos diferentes de investigación y detalles a mejorar en el proyecto:

- Primeramente, se sugiere que, se realicen experimentos reemplazando la capa final de la red con un *Random Forest* (RF) utilizando características muestradas de las capas de red anteriores, ya que en el artículo [4] tiene un gran beneficio para la red y se generan muy buenos resultados.
- Se puede probar cambiar la forma de entrenar la red. En este proyecto se ha realizado con tres pasos, primero las cabeceras, a continuación la capa 4 y superior y por último todas juntas. Esto se puede modificar y encontrar una mejor forma de entrenar la red.
- Para terminar, se sugiere la mejora de la aplicación. Aparte de mejoras estéticas, se sugiere la inserción de más de una imagen en la aplicación teniendo un histórico de las imágenes y los defectos que se han detectado.

Bibliografía

- [1] Waleed Abdulla. Splash of color: segmentación de instancias con máscara r-cnn y tensorflow. <https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238>, 2018. [Online; Accedido 27-junio-2020].
- [2] Marlon Cajamarca. Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo. <https://planetachatbot.com/inteligencia-artificial-aprendizaje-autom%C3%A1tico-y-aprendizaje-profundo-862ca9790bb9>, 2019. [Online; Accedido 21-junio-2020].
- [3] Paloma Recuero de los Santos. Tipos de aprendizaje en machine learning: supervisado y no supervisado. <https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/>, 2017. [Online; Accedido 24-junio-2020].
- [4] Xinghui Dong, Chris J. Taylor, and Tim F. Cootes. Small defect detection using convolutional neural network features and random forests. *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.
- [5] Max Ferguson. Repositorio de detection en GitHub. <https://github.com/maxkferg/metal-defect-detection>, 2016. [Online; Accedido 21-noviembre-2019].
- [6] Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, and Kincho H. Law. Automatic localization of casting defects with convolutional neural networks. *The IEEE International Conference on Big Data (Big Data)*, pages 176065–176086, 2019.

- [7] Noelia Ubierna Fernández. Repositorio de XRayDetector en GitHub. <https://github.com/nuf1001/XRayDetector>, 2019. [Online; Accedido 21-noviembre-2019].
- [8] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.
- [9] Jonathan Hui. mAP (mean Average Precision) for Object Detection. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173, 2018. [Online; Accedido 29-junio-2020].
- [10] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. <http://www.scipy.org/>, 2001–. [Internet; descargado 16-julio-2020].
- [11] Kaiming He and Georgia Gkioxari and Piotr Dollár and Ross Girshick. Mask R-CNN. <https://arxiv.org/abs/1703.06870>, 2017. [Online; Accedido 24-junio-2020].
- [12] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017.
- [13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755, 2014.
- [14] Cesar Perez Lopez and Daniel Santin. *Minería de datos. Técnicas y herramientas: técnicas y herramientas*. PARANINFO, 2008.
- [15] Muhammad Rizwan. Residual Networks (ResNets). <https://engmrk.com/residual-networks-resnets/>, 2018. [Online; Accedido 8-julio-2020].
- [16] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.
- [17] Pablo Ulloa Castro. Programación orientada a objetos: Patrón Singleton. <https://medium.com/@pabloulloacastro/programaci%C3%B3n-orientada-a-objetos-patr%C3%ADn-singleton-423e2755614b>. [Online; Accedido 16-julio-2020].

- [18] José Francisco Díez Pastor, Alvar Arnaiz-González, César García Osorio, and Juan José Rodríguez. Segmentación de defectos en piezas de fundido usando umbrales adaptativos y ensambles. In *XVII congreso español sobre tecnologías y lógica fuzzy, ESTYLF 2014*, pages 345–350, 2014.
- [19] José Francisco Díez Pastor, Cesar García Osorio, Víctor Barbero García, and Alan Blanco Álamo. Imbalanced learning ensembles for defect detection in x-ray images. In *26th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2013*, pages 654–663, 2013.
- [20] Python. distutils- Construyendo e instalando módulos Python. <https://docs.python.org/3/library/distutils.html#module-distutils>. [Online; Accedido 16-julio-2020].
- [21] Recursos Python. Instalar PIL/Pillow y aplicar efectos visuales. <https://recursospython.com/guias-y-manuales/installar-pil-pillow-efectos/>, 2014. [Online; Accedido 16-julio-2020].
- [22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [23] scikit-image. Image processing in Python. <https://scikit-image.org/>, 2020. [Online; Accedido 16-julio-2020].
- [24] Shubham Chauhan. Understanding mean Average Precision for Object Detection (with Python Code). <https://medium.com/analytics-vidhya/map-mean-average-precision-for-object-detection-with-simple-python-demonstrati> 2019. [Online; Accedido 29-junio-2020].
- [25] The American Society for Nondestructive Testing (ASNT). INTRODUCCIÓN A LAS PRUEBAS NO DESTRUCTIVAS. [https://www.asnt.org/MajorSiteSections/About/Introduction_to_Nondestructive_Testing.aspx#:~:text=Nondestructive%20testing%20\(NDT\)%20is%20the,part%20can%20still%20be%20used.](https://www.asnt.org/MajorSiteSections/About/Introduction_to_Nondestructive_Testing.aspx#:~:text=Nondestructive%20testing%20(NDT)%20is%20the,part%20can%20still%20be%20used.), 2011. [Online; Accedido 24-junio-2020].
- [26] tmux. Repositorio de tmux en GitHub. <https://github.com/tmux/tmux>, 2015. [Online; Accedido 16-julio-2020].

- [27] Waleed Abdulla. Mask R-CNN for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017. [Online; Accedido 24-junio-2020].
- [28] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2015. [Internet; descargado 30-septiembre-2015].
- [29] Zhaojin Huang and Lichao Huang and Yongchao Gong and Chang Huang and Xinggang Wang. Mask Scoring R-CNN. https://openaccess.thecvf.com/content_CVPR_2019/papers/Huang_Mask_Scoring_R-CNN_CVPR_2019_paper.pdf, 2018. [Online; Accedido 24-junio-2020].