UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Detección de defectos en piezas metálicas usando radiografías y *Deep Learning*

Presentado por Fco. Javier Yagüe Izquierdo
en Universidad de Burgos — 9 de junio
de 2021
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 D. Pedro Latorre Carmona, profesores del Departamento de Ingeniería Informática, Área de Lenguajes y Sistemas Informáticos.

Exponen:

Que el alumno D. Fco. Javier Yagüe Izquierdo, con DNI 71312406D, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 9 de junio de 2021

Firmado por DIEZ PASTOR JOSE FRANCISCO - 71278385M el día 10/06/2021 con un certificado emitido por AC FNMT Usuarios

LATORRE CARMONA PEDRO - 22569294S

Firmado digitalmente por LATORRE CARMONA PEDRO - 22569294S
Fecha: 2021.06.09 14:46:47 +02'00'

Vº. Bº. del Tutor:

D. José Francisco Díez Pastor

Vº. Bº. del Tutor:

D. Pedro Latorre Carmona

Resumen

La evolución del proceso tecnológico e industrial ha traído consigo nuevas técnicas de fabricación haciendo posible la elaboración de piezas cada vez más complejas, trayendo consigo a su vez una mayor dificultad de ejecutar una de las fases más cruciales del proceso de fabricación, el control de calidad.

La detección de defectos está convirtiéndose en un proceso difícil de llevar a cabo, en parte por el aumento de la complejidad de las formas y características visuales de los objetos a analizar, además de posibles defectos contenidos en el interior de la pieza.

La aplicación de imágenes tomadas por *Rayos X* ha facilitado en gran medida este proceso, que permite detectar no solo defectos relativamente superficiales si no además fallos de soldadura o burbujas internas que pueden afectar a la integridad física de la pieza y representar un grave problema de seguridad posterior.

El objetivo del trabajo es automatizar la fase de control con la ayuda de una red neuronal entrenada para la detección automatizada de estos defectos internos en imágenes tomadas por *Rayos X*.

Descriptores

Rayos X, deep learning, machine learning, detección, defectos, aplicación web.

Abstract

With the development of the industrial world new production techniques have emerged that have made possible to produce metal parts with more complex shapes, making it more difficult for one of the most important phases of the manufacturing process, quality control.

The emergence of metallic parts with much more complex shapes mean that the visual and superficial analysis has become even more insufficient in addition to the impossibility of detecting internal defects that the part could have.

The application of *X-Ray* images have made this process much more easy, allowing to not only detect superficial defects in a much more simple way but also to detect welding or casting defects that could represent a serious hazard for the physical integrity of the metal parts that could lead to a security problem afterwards.

This objective of the project is to automate the control phase thanks to the application of a neuronal network trained for the automatic detection of the internal defects present in *X-ray* images.

Keywords

X-Ray, deep learning, machine learning, detection, defects, web application.

Índice general

Índice de figuras

Índice de tablas

Introducción

A medida que el proceso industrial se ha desarrollado, también lo han hecho numerosas técnicas de fabricación, permitiendo reducir las fases de fabricación de piezas metálicas y no es posible realizar exámenes visuales en puntos intermedios de la fabricación.

Esto significa que el examen visual convencional, además de pasar por alto posibles defectos internos o apenas visibles, adquiere un mayor nivel de dificultad a la hora de manipular y observar la pieza, aumentando el tiempo invertido y coste invertido en esta fase. La aplicación de imágenes tomadas por *Rayos X* han revolucionado este proceso, facilitándolo en gran medida y llevándolo a un nivel de detalle superior.

Durante la fabricación de piezas metálicas en concreto, pueden darse determinados defectos no perceptibles a simple vista que pueden afectar a la integridad de la pieza en un futuro provocando su rotura o desgaste. Los más comunes son grietas de soldadura que se han ido extendiendo y burbujas que pueden tener su origen en espacio vacío o aire contenido en el propio material.

A partir de este punto pueden tomarse dos caminos, el Análisis No Destructivo, o el Destructivo, de la pieza.

Análisis No Destructivo

Se realizan con el objetivo de no alterar la pieza, ni a nivel estructural ni interno. Incluyen algunas pruebas en las que intervienen determinados fluidos en los que se puede sumergir la pieza, análisis meramente visual personal de la pieza por parte de un operario ó el método basado en radiografías que se aborda en este proyecto.

Como resultado se debería de obtener unas observaciones lo más completas posible permitiendo que la pieza objetivo pueda continuar en la línea de producción hasta su finalización y venta.

Análisis Destructivo

El objetivo es un análisis más profundo, por lo que la pieza se ve alterada de múltiples formas ya sea mediante compuestos químicos, cortes o impactos y desgaste prolongado que busca simular el uso continuado que tendrá la pieza una vez se le de el uso correspondiente.

Por ejemplo existe el *Ensayo de dureza Vickers* que consiste en perforar una pieza con el objetivo de poner a prueba la dureza del material, lo que a pesar de otorgar unos resultados fiables, significa que la pieza sobre la que se realizan las pruebas ya no podrá ser utilizada y será desechada una vez concluidas las pruebas.

A continuación se detallan las ventajas y desventajas entre los mismos:

Ventajas

- Análisis No Destructivo
  - La pieza se conserva intacta al final de las pruebas
  - Tiempo de ejecución reducido
  - Especialmente útil para piezas de alto valor
- Análisis Destructivo
  - Técnicas más experimentadas
  - Permite pruebas directas sobre el interior de la propia pieza
  - Es posible analizar la composición del metal

Desventajas

- Análisis No Destructivo
  - Se requiere experiencia para su práctica
  - Inversión inicial significativa
- Análisis Destructivo
  - La pieza es destruida

- Mayor coste a largo plazo
- Más tiempo para su ejecución

A las ventajas del *Análisis No Destructivo* se pretende añadir la detección automática de estos defectos en las imágenes que ya se han tomado gracias a los *Rayos X*, de forma que con la ayuda de una Red Neuronal entrenada se reduzcan aún más los tiempos de análisis de este tipo de fotografías durante la fase de control de calidad.

Mejora del Proyecto

Como mejora significativa del Proyecto, se ha trabajado con radiografías de piezas etiquetadas por D. José Francisco Díez Pastor y D. Pedro Latorre Carmona. Gracias a esto, el modelo se ha desarrollado para ejecutarse sobre un conjunto propio, de forma que el modelo se ha entrenado y desarrollado para funcionar sobre un conjunto objetivo propio.

También y gracias al uso de *Detectron2* ha sido posible el uso de bibliotecas de programación más potentes. Además de visualización de resultados algo más interactiva en un visor web.

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.

El repositorio de *Github* del proyecto se encuentra en: https://github.com/fyi0000/TFG-GII-20.04

Objetivos del proyecto

2.1. Objetivos generales

- Profundizar en la aplicación del *Deep Learning* y redes neuronales en el mundo industrial.
- Asegurar una ejecución eficiente y con un rendimiento adecuado.
- Análisis de los resultados, procesado y presentación sencilla de los mismos.
- Hacer posible la ejecución de la herramienta final en múltiples entornos.

2.2. Objetivos técnicos

- Introducción y aprendizaje de la librería de *Facebook Detectron 2* para la detección y segmentación de imágenes.
- Analizar la fase de entrenamiento y optimizar el modelo resultante.
- Desarrollar una aplicación web en *Flask* que permita el uso sencillo del modelo
- Facilitar la accesibilidad de la web con la ayuda de *Docker* para hacer posible su ejecución en múltiples entornos.
- Introducirse al uso práctico de *Github* y la organización del proyecto.

2.3. Objetivos personales

- Profundizar en el mundo del *Machine Learning* y conocer hasta qué punto está presente en el día a día.
- Aumentar los conocimientos ya adquiridos de *Python* con el uso de más librerías.
- Adentrarse en el mundo del desarrollo web introduciéndose tanto en *Frontend*, de cara al usuario y la presentación del diferente contenido web, como en el *Backend*, como funciona la parte del servidor y el tratamiento de la información facilitada por el usuario.
- Analizar las necesidades de un trabajador del sector para intentar aportar el mayor número de facilidades con la herramienta.

Conceptos teóricos

3.1. Machine Learning y técnicas utilizadas

La parte más importante de este proyecto recae sobre la aplicación del *Machine Learning* y una de sus ramas, el *Deep Learning*.

¿Qué es el *Machine Learning*?

El *Machine Learning*, también referido como aprendizaje automático, es el desarrollo de algoritmos que permitan el aprendizaje de determinada información por parte de las máquinas a partir de unos datos facilitados. Esto significa que este proceso se llevará a cabo de una forma autónoma y intervención directa del usuario. Los algoritmos trabajan sobre un conjunto de datos destinados al entrenamiento, que permitirán la ejecución de tareas de clasificación, por ejemplo de forma automatizada. Esto significa que un mismo algoritmo puede aprender a clasificar diferentes clases de objetos según se le proporcione un conjunto de entrenamiento sin que el programador tenga que realizar ningún cambio interno en el algoritmo.

Categorías de *Machine Learning*:

- **Aprendizaje Supervisado:** Los algoritmos de aprendizaje automático que pertenecen a esta categoría obtienen un modelo a partir de unos datos de entrada y una salida conocida, de forma que las respuestas ajustan el modelo para en el futuro hacer predicciones sobre unos datos de entrada nuevos y cuya respuesta es desconocida. Las dos técnicas más comunes para el desarrollo de los modelos mediante Aprendizaje Supervisado son *Regresión* y *Clasificación.* El sistema empleado en

este proyecto pertenece a esta categoría, siendo capaz de diferenciar dentro de la imagen regiones que son de interés de las que no.

- **Aprendizaje No Supervisado:** Los algoritmos de esta categoría tienen como objetivo extraer determinados patrones implícitos en un conjunto de datos, ya sea para agrupar o diferenciar unos de otros. En este tipo de aprendizaje no se utilizan respuestas predefinidas a los datos de entrada facilitados. La técnica más común en este tipo de aprendizaje es el *Clustering*. Este último método también se puede aplicar a la segmentación de imágenes aunque no ha sido el caso de este proyecto.[10]

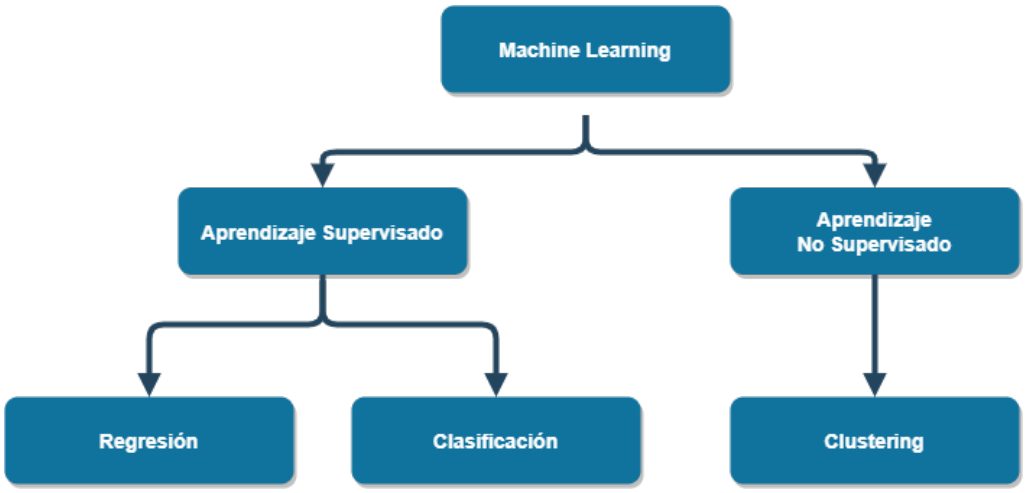

Figura 3.1: Técnicas del Machine Learning

¿Qué es el Deep Learning?

El *Deep Learning* ó Aprendizaje Profundo en castellano, es una rama del *Machine Learning* que se basa en imitar el funcionamiento de las *Redes neuronales* humanas y sus distintas conexiones entre capas. Las *Redes neuronales* se organizan en capas de entrada, ocultas y de salida. Conforme la información va siendo procesada y las capas ocultas reciben información, se generan salidas que a su vez sirven de entrada para la siguiente capa. El número de veces que este proceso se repite definirá la profundidad que tenga el modelo.

Conforme se obtienen unos resultados, se hace una comparación con una serie de resultados predefinidos antes del entrenamiento. Dependiendo de las coincidencias obtenidas entre los resultados obtenidos y los resultados, se estima como de bueno es el rendimiento y el ajuste necesario de los parámetros o *pesos* de las capas intermedias.

La principal característica que diferencia el *Deep Learning* del conjunto de métodos de *Machine Learning* es que el *Deep Learning* utiliza algoritmos para confeccionar una red neuronal que aprende la información partiendo de los datos proporcionados, generando decisiones propias. El *Machine Learning* por su parte utiliza algoritmos para extraer la información de los datos facilitados y en base a ellos generar decisiones.

Además el *Deep Learning* es capaz no solo de procesar la información etiquetada que se le proporciona, si no de etiquetar por sí mismo dicha información. Por ello, dependiendo de la aplicación que se le dé al *Deep Learning*, se podría clasificar tanto como *Aprendizaje Supervisado* ó *Aprendizaje No Supervisado*.

3.2. Faster R-CNN y Redes Neuronales Convolucionales

¿Cómo funciona la detección de objetos mediante Redes Neuronales?

Cuando se lleva a cabo la detección de objetos en imágenes con la ayuda de Redes Neuronales y más concretamente mediante *R-CNN*, convencionalmente se llevan a cabo 3 fases:

- **Generación de Regiones Propuestas:** En esta primera fase se seleccionan múltiples regiones de la imagen que podrían o no contener

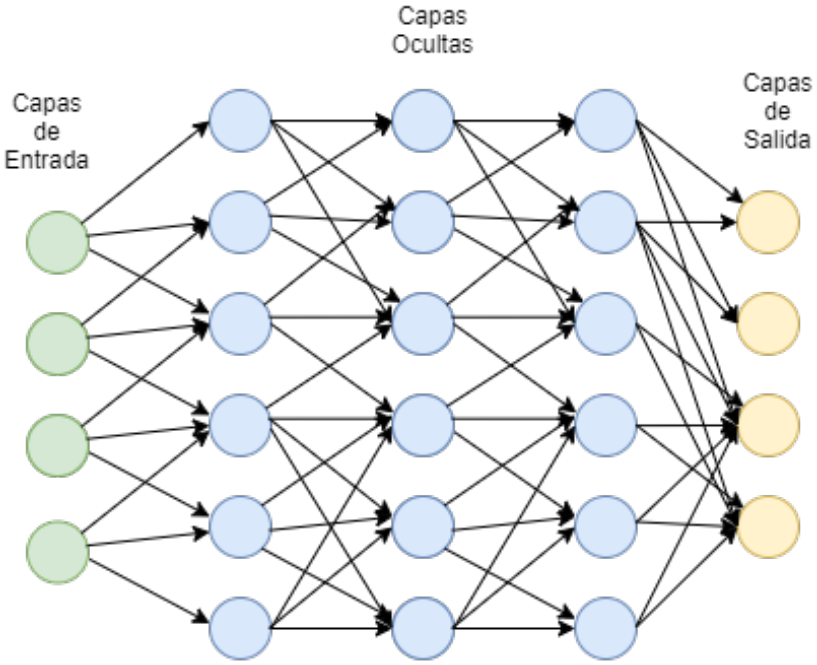

Figura 3.2: Estructura de las Redes Neuronales

un objeto a reconocer. El número de este tipo de regiones pude rondar las miles para una sola imagen. Aunque pueden ser de múltiples tamaños y estar solapadas unas sobre otras.

- **Extracción de Descriptores o Características:** Utilizando un vector de descriptores que reconocen las regiones, describen la imagen contenida en la región para poder ser reconocida posteriormente. Es la fase más importante para el correcto funcionamiento.

- **Clasificación:** Por último se clasifican las regiones según su contenido y se determina si contiene un objeto o por el contrario forma parte del fondo. Finalmente se clasifican los diferentes objetos detectados en las respectivas clases.

Redes Neuronales Convolucionales

Las *Redes Neuronales Convolucionales* ó *CNN* por sus siglas en inglés, son un tipo de redes neuronales caracterizadas por la presencia de una capa que les da nombre *Capa Convolucional* y que destacan por comportarse especialmente bien a la hora de aplicarse en el reconocimiento de imágenes.

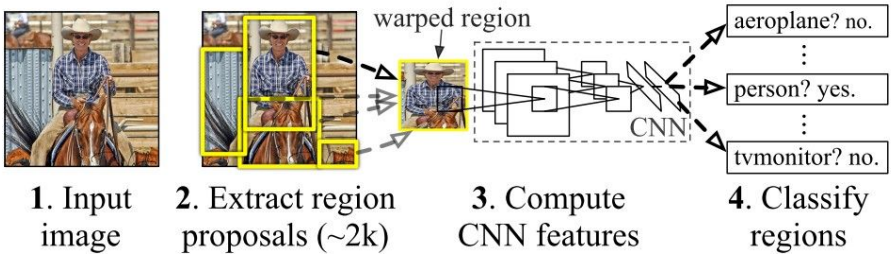

Figura 3.3: Flujo de funcionamiento de *R-CNN* [6]

El objetivo de este tipo de redes es similar al comportamiento del ojo humano, al observar un imagen, se segmenta según los diferentes componentes que presenta, como por ejemplo diferenciar personas en un parque del propio parque que forma parte del fondo de la imagen.

Este tipo de redes tienen su origen en 1980, cuando el japonés *Kunihiko Fukushima* [16] introdujo el *Neocognitron*, una red neuronal básica para el aprendizaje no supervisado que funcionaba sobre imágenes. Ese mismo año el francés *Yann LeCun*, implementó sobre este primer trabajo del japonés la llamada *LeNet* que funcionaba sobre dígitos escritos para su reconocimiento.

Este tipo de redes presentaban problemas en cuanto a la escala de imágenes, ya que el funcionamiento se degradaba conforme se incrementaban las dimensiones de las imágenes. La aparición e impulso del *Deep Learning* en 2012 promovió el desarrollo de las *CNN* y su aplicación a todo tipo de imágenes.

Componentes y Funcionamiento de las Redes Neuronales Convolucionales

Antes de comenzar con el proceso, es importante tener en cuenta que una imagen es una matriz de valores, esto significa que según el tamaño de la imagen tendremos una matriz de mayor o menor tamaño. A su vez esto significa que para el procesamiento de imagen se necesitarán un número diferente de neuronas.

También hay que tener en cuenta el tipo de imagen que se está utilizando, ya que una imagen en escala de grises es una única matriz con valores únicos, mientras que una imagen en *RGB*, los 3 canales de color, representan 3 matrices diferentes formando distintas capas que también se han de procesar.

Esto puede representar un problema si se trabaja con imágenes de grandes dimensiones ya que a medida que se incrementa el tamaño, también lo hace la capacidad de computación necesaria para procesar dicha imagen.

- ***Input Layer* ó Capa de Entrada:** El primer paso es preprocesar la imagen y como se ha comentado, se descomponen las capas que la forman, dependiendo del tipo de imagen que sea, escala de grises o en color y del tamaño de la misma.

- **Capa Convolucional:** Esta capa es similar a una capa oculta de una red neuronal corriente que realiza una operación y se la trasmite a la siguiente. La principal diferencia es que en este caso las conexiones no son totales, es decir, no todas las neuronas están conectadas entre sí. Esto recibe el nombre de conectividad local. A su vez, se realizan las operaciones convolucionales propiamente dichas. El proceso consiste en, utilizando un kernel ó una matriz de unas determinadas dimensiones, recorrer la imagen por completo y realizar un producto escalar obteniendo una nueva matriz. Esta matriz recoge las características o *features* relevantes de la imagen, como los bordes y contornos de los objetos.

  Dependiendo de la imagen, es posible que se deba aplicar la técnica llamada *padding*. Esto implica incrementar en un determinado número de filas y columnas la matriz imagen con el objetivo de que se conserve información relevante presente en los bordes.

- ***Pooling Layer*:** En esta capa, normalmente inmediatamente posterior a la Capa Convolucional, tiene como objetivo reducir las dimensiones de la matriz de características obtenida aplicando una técnica definida. Por ejemplo, una de las técnicas más usadas se basa en máximos o *max-pooling*. Se recorre la matriz con otro kernel de determinado tamaño y se conserva el máximo valor contenido en la matriz original.

- ***Fully Connected Layer*:** Conforme se suceden las convoluciones se reducen las dimensiones y finalmente obtendremos una capa en la que todas las neuronas están conectadas con las entradas de la siguiente. En esta capa se dividen los distintos pesos resultantes en las clases con las que se cuenta en el modelo, obteniendo una neurona para cada una. Por ello en esta capa se aplican las funciones de activación como puede ser *SoftMax* obteniendo la probabilidad de que la entrada pertenezca a una de las clases resultantes.[5]

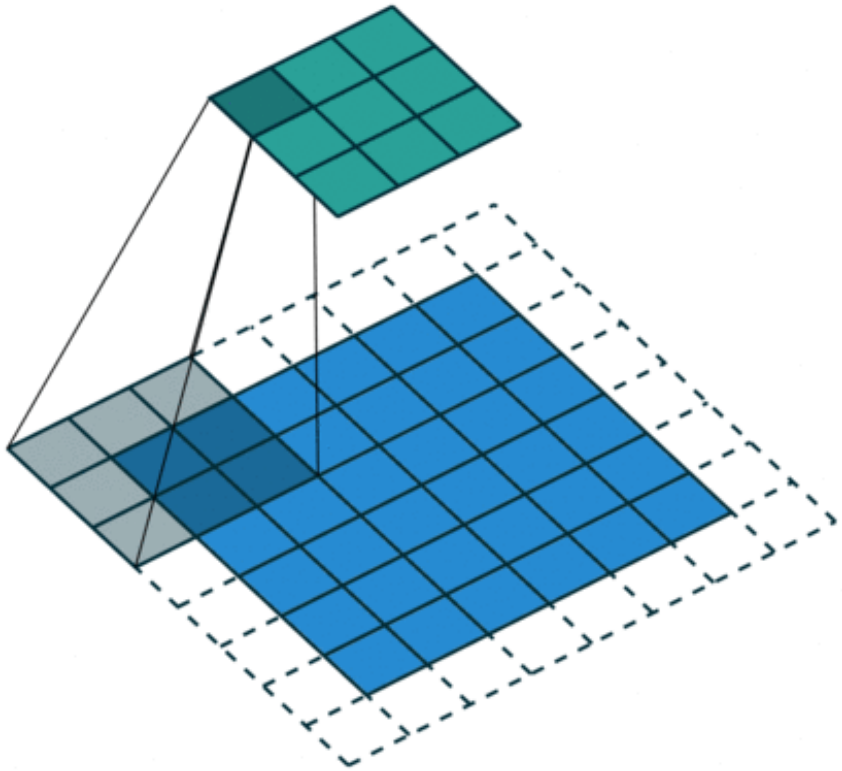

Figura 3.4: Proceso de Convolución [14]

R-CNN

Los problemas que presentaba *CNN* en cuanto a las dimensiones de las imágenes y al número de regiones que podían contener objetos de interés lastraron su desarrollo. Por ello, en 2014 *Ross Girshick*, propuso la red *R-CNN* ó *Regions with CNN features*. En definitiva los cambios que esta red presentaba eran, en primer lugar la extracción de regiones propuestas para su posterior combinación según similitud y por último establecer unas regiones propuestas candidatas a ser regiones de interés finales.[2]

Faster R-CNN

Faster R-CNN y como su nombre indica representa un paso más allá respecto a R-CNN en cuanto a velocidad y simplicidad además de diferentes cambios:

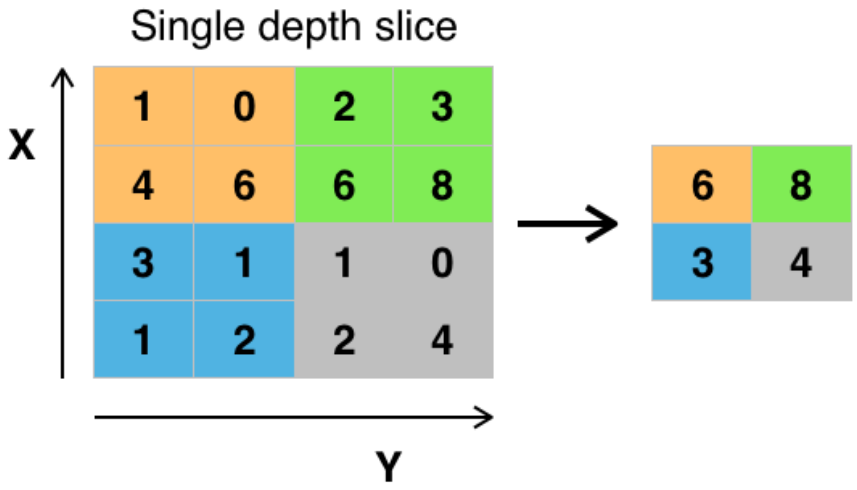

Figura 3.5: Pooling [1]

- **Nueva capa:** Llamada *ROI Pooling* extrae los vectores de características de la imagen con una misma longitud.
- **Simplicidad:** *Faster R-CNN* agrupa los 3 puntos del anterior apartado y junta las 3 fases en una única.
- **Computaciones compartidas:** Cuando procesa una *ROI* (Región de Interés) comparte mediante la *ROI Pooling* las operaciones realizadas, que pueden reutilizarse y ahorra tiempo respecto a *R-CNN*.
- **Sin Caché:** No almacena en caché las características extraídas de la imagen y por lo tanto ahorra espacio en disco.
- **RPN (Red de Regiones Propuestas):** Constituye una red neuronal completa que genera las *ROI* en distintas escalas y se lo indica a *Fast R-CNN*, una versión anterior contenida con el fin de escalar aún más la velocidad.
- **Cajas de Anclaje o *Anchor Boxes*:** En lugar de generar distintas instancias de una misma imagen con diferentes tamaños, con las *Anchor Boxes* se crea una referencia con un determinado valor que indica tamaño o escala, de esta forma puede haber distintas *Anchor Boxes* para una misma imagen optimizando la estructura. Esto sustituye a estructuras piramidales anteriores.

Finalmente el funcionamiento de *Faster R-CNN* se resumiría de la siguiente forma:

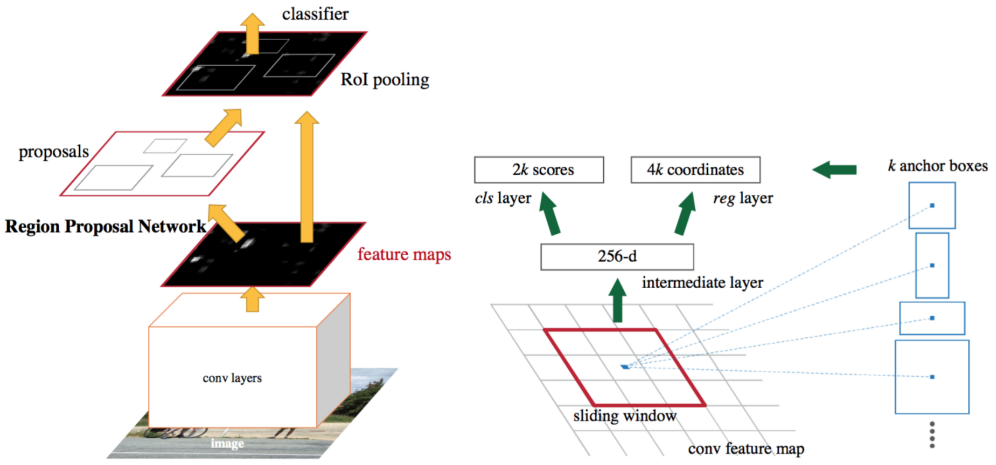

Figura 3.6: Flujo de funcionamiento de *Faster R-CNN* [11]

Primero el RPN genera las regiones propuestas de la imagen, generando para cada región un vector de características mediante el *ROI Pooling*. Posteriormente el *Fast R-CNN* clasifica estas regiones según los vectores obtenidos y se obtienen las "calificaciones" para cada objeto detectado indicando la confianza de acierto y las *Bounding Boxes*, que envuelven el objeto y su contorno.

Calificación de objetos candidatos

Se entiende por calificación de los objetos candidatos como la asignación de un valor a una región en función de la calidad de la detección, es decir, su la detección es más o menos fiable.

La calificación que recibe una determinada región propuesta depende de la *Intersection-over-Union(IoU)*.

La *Intersection-over-Union* nos permite determinar cuanto se sobreponen dos regiones determinadas, de forma que cuanto más se ajusten ambas regiones mayor será la puntuación en un rango de 0 a 1. De esta manera en la fase de entrenamiento el objetivo es que las regiones sean lo más coincidentes posibles y por tanto que el valor de la IoU se acerque lo máximo a 1.

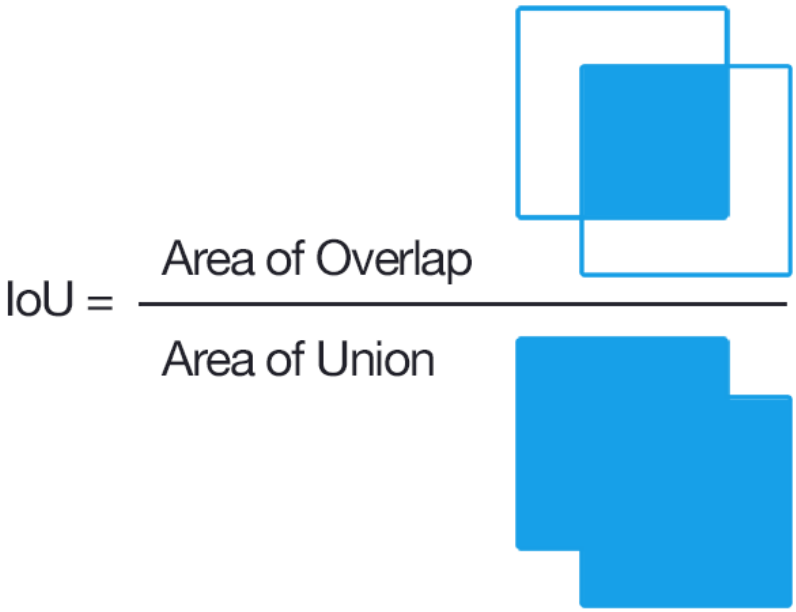

Figura 3.7: Intersection over Union [13]

La IoU se obtiene de la división entre la región coincidente de las dos áreas entre la unión de ambas y según se obtenga un valor dentro de unos límites, *RPN* otorga un positivo o negativo.

$$\text{Calificación}(IoU) = \begin{cases} Positivo & si \quad IoU \geq 0{,}7 \\ Positivo & si \quad 0{,}5 < IoU \leq 0{,}7 \\ Negativo & si \quad IoU < 0{,}3 \\ NiNegativoniPositivo & si \quad 0{,}3 \leq IoU \leq 0{,}5 \end{cases}$$

Las condiciones se resumen:

- Si el valor de IoU es mayor a 0.7, se considera un *positivo*.
- Si la región no tiene un valor de 0.7 mínimo pero alcanza el 0.5 es *positivo*.
- Con un IoU de 0.3 o inferior es *negativo*.

- Por último si el valor está entre 0.3 y 0.5 no se considera ni positivo ni negativo y se descarta.

Figura 3.8: Mal resultado, buen resultado y el mejor resultado [12]

Métricas de clasificación del rendimiento

En este trabajo se utilizarán tres métricas muy utilizadas y aplicadas en la *Clasificación Binaria* para analizar los resultados obtenidos. Estas son *Precisión*, *Exhaustividad* o *Recall* en inglés y la combinación de ambas, *F1.*[7]

- **Precision:** La precisión es una métrica para poner en valor cuantos de los resultados que se han calificado como positivos, son realmente positivos. Los valores van desde 0 hasta 1, significando, por ejemplo una precisión de 0.5 que se acierta un 50 % de las veces. La fórmula es:

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

(TP = Positivo Verdadero, TN = Negativo Verdadero, FP= Falso Positivo, FN = Falso Negativo)

- **Recall:** El *Recall* o Exhaustividad contempla cuantos elementos, calificados o no, lo han sido correctamente. En nuestro caso representaría cuantos defectos del total de los presentes en la imagen se han marcado correctamente. A veces se le da una menor importancia que a la Precisión.

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

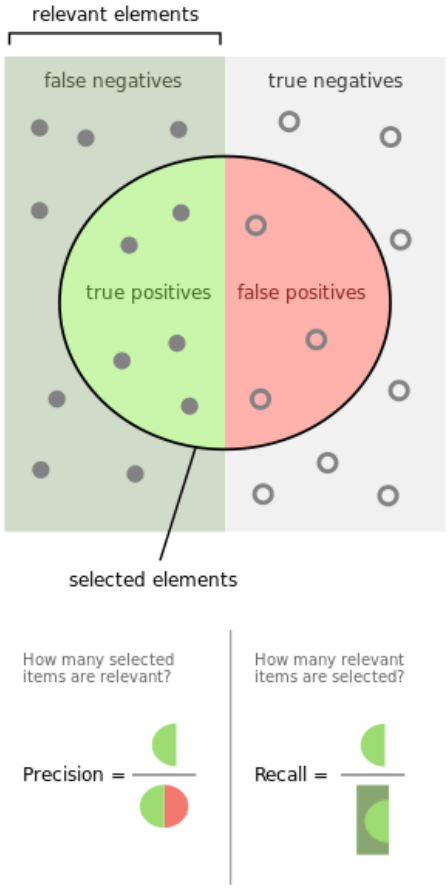

Figura 3.9: Precisión y Recall [15]

- **F1:** El F1 o *F-score* es la media armónica de las dos anteriores. La principal característica de este valor es que grandes diferencias de valores no determinan tanto su resultado como podría observarse de realizar la media aritmética, por ejemplo.

$$F1 = \frac{2 \cdot (Precision \cdot Recall)}{Precision + Recall}$$

3.3. Formato de objetos COCO: *Common Objects in COntext*

Para este proyecto y la utilización de Detectron 2, se ha trabajado con un formato de etiquetado de las imágenes basado en ficheros JSON que acompañando a una imagen original y sin necesidad de la máscara binaria es capaz de señalar la región etiquetada como objeto.

Figura 3.10: Estructura de las anotaciones JSON COCO

El fichero tiene 3 apartados que son *Images*, *Categories* y *Annotations*. Junto con las imágenes originales sirve para el registro de las imágenes tanto para el conjunto de validación ó test como para el de entrenamiento.

- **Images:** Conecta cada fichero de imagen con un identificador y dimensiones.
- **Categories:** Registra el tipo de objetos, clases, que hay en el conjunto, también con un identificador.
- **Annotations:** Contiene el identificador de un objeto, categoría a la que pertenece, si es o no un conjunto de objetos, pares de coordenadas que indican la región que ocupa, identificador de la imagen registrada al principio, área total que ocupa el objeto y *bounding box* que lo engloba.

Técnicas y herramientas

4.1. Metodología Ágil y Scrum

Para la organización y planificación del proyecto se ha seguido la metodología ágil *Scrum*, mediante reuniones en las que se reparte cada semana el trabajo para la siguiente a la vez que se revisaban los resultados del trabajo de la anterior determinando iteraciones o *Sprints*. De esta forma según se desarrollado un determinado Sprint se ha adaptado el margen temporal alargándolo o acortándolo según la situación lo ha requerido.

4.2. Control de Versiones y Gestión de Proyectos

Git y Github

Tanto Git[1] como Github[2] se han utilizado para el control de versiones y gestión del proyecto se han utilizado ambas herramientas para el versionado de los ficheros de código y su alojamiento en la nube además de la organización de *Milestones*, *Issues* entre otros.

[1]Git: https://git-scm.com/
[2]Github: https://github.com/

4.3. Herramientas

El proyecto se ha desarrollado en el lenguaje *Python* que presenta un gran dinamismo y adaptación específica para este proyecto debido a la multitud de librerías disponibles orientadas a innumerables objetivos pero específicamente al *Machine Learning*.

jQuery

De forma bastante sencilla se ha utilizado mediante *CDN* para evitar la descarga, *jQuery*[3] junto con peticiones *Ajax*. Debido a que no se tenía demasiada experiencia en este aspecto, ha sido muy interesante utilizar peticiones asíncronas para, por ejemplo, mostrar un *loader* o animación sencilla de carga sin necesidad de recargar la página.

De esta forma en lugar de recargar el fichero *HTML* cada vez que se realiza un cambio, la aplicación es algo más dinámica.

Google Colab

Google Colaboratory[4] o Colab se ha utilizado para el desarrollo y ejecución de gran parte del proyecto junto con los *Notebooks*, ficheros también utilizados en *Anaconda* para el desarrollo en *Python* que se caracterizan por su organización del código en celdas.

Similar a *Azure* de Microsoft permite la ejecución y desarrollo en la nube además de la utilización de las GPU de *Google* que son perfectas para la fase de entrenamiento en el proceso de Machine Learning, reduciendo significativamente los tiempos de ejecución, entrenamiento y descarga. También tiene disponible la TPU (Tensor Processing Unit), unidad especializada en trabajar con grandes matrices de datos que utilizan las redes neuronales.

Su uso es gratuito aunque la ejecución continuada de código tiene límites temporales.

Pycharm

Pycharm[5] es un IDE para *Python* desarrollado por *JetBrains* con múltiples funcionalidades para pruebas, gestión de entornos virtuales para *Python* e integración de herramientas para la gestión de proyectos.

[3] jQuery: https://jquery.com/
[4] Google Colab: https://colab.research.google.com/notebooks/welcome.ipynb?hl=es
[5] Pycharm: https://www.jetbrains.com/pycharm/

Utilizado bajo la licencia anual gratuita para los estudiantes universitarios.

Visual Studio Code

Visual Studio Code[6] es un editor de código ligero de *Microsoft* cuya principal ventaja es la facilidad de gestión de extensiones con múltiples funcionalidades como corrección de código, sincronización con la nube, terminal integrada e incluso desarrollo concurrente y simultáneo.

Docker

Docker[7] es un sistema de contenedores que sustituye a las máquinas virtuales convencionales, de forma que es posible crear un entorno totalmente aislado que trabaje en un sistema Unix pero ejecutado en un sistema anfitrión *Windows* ó viceversa. Esto facilita la ejecución de aplicaciones propias de un sistema concreto de forma totalmente aislada sin importar el sistema que se utiliza en el anfitrión.

Además gracias a los *Dockerfiles*, mediante un simple fichero es posible construir una imagen con unos requerimientos específicos a partir de la cual se pueden generar contenedores propiamente dichos en un tiempo muy reducido, éstos son independientes pero presentan las mismas funcionalidades base heredadas de la imagen.

4.4. Frameworks y Bibliotecas de Python

Flask

Flask[8] es un *micro-framework* orientado a creación de aplicaciones web de una forma sencilla y con una base muy sencilla pero con unas capacidades extensibles. Por ejemplo no tiene una integración directa con bases de datos pero que mediante el uso de plugins puede utilizarse rápidamente.

Es Open-Source, tiene gestión de sesiones, cookies y tiene su propio servidor para alojar la aplicación en una máquina junto con un modo desarrollador que evita que cuando se realizan cambios sea necesario relanzar la aplicación una y otra vez.

[6]Visual Studio Code: https://azure.microsoft.com/es-es/products/visual-studio-code/
[7]Docker: https://www.docker.com/
[8]Flask: https://flask.palletsprojects.com/en/1.1.x/

NumPy

NumPy[9] es una librería de *Python* que se especializa en la manipulación de variables numéricas y matriciales con múltiples funciones que facilitan enormemente la manipulación conjunta de los datos como filtrado, copia, álgebra, etc.

En este proyecto será especialmente útil para la manipulación de imágenes y las matrices que las representan.

Pandas

Pandas[10] es una librería especialmente utilizada para la manipulación de estructuras de datos y más especialmente tablas o ficheros *CSV*. Organiza los datos en *DataFrames* que permiten el uso de los datos a gran escala y con gran cantidad de registros e instancias.

OpenCV

OpenCV[11] es una librería Open-Source que incluye múltiples algoritmos de visión y reconocimiento artificial por computación mediante manipulación de información multimedia.

Pillow

Pillow[12] es una librería orientada a la manipulación de imágenes en *Python*. Permite trabajar con las distintas capas de la imagen además del guardado y carga de múltiples ficheros de imagen.

Scikit-image

Scikit-image[13] es una biblioteca de algoritmos para trabajar con imágenes aplicando distintos métodos y técnicas para, por ejemplo, el reconocimiento y segmentación de imágenes, morfología o visualización.

[9] NumPy: https://numpy.org/
[10] Pandas: https://pandas.pydata.org/
[11] OpenCV: https://opencv.org/
[12] Pillow: https://python-pillow.org/
[13] Scikit-image: https://scikit-image.org/

Matplotlib

Matplotlib [14] se utiliza para la visualización de contenido estático en *Python*, permite la utilización de ejes, creación de composiciones de imagen para la muestra de diferentes contenidos además de la manipulación de gráficos.

En este proyecto será utilizado para la visualización y presentación de resultados.

Plotly

Plotly[15] es una revolucionaria librería para el análisis y manipulación de datos además de la generación de gráficos y elementos de visualización de forma sencilla pero muy polivalentes. Presenta multitud de figuras que pueden personalizarse para casos concretos. Además de un dinamismo importante gracias a su integración con *JavaScript* para la creación de figuras independientes y conversores que permiten trabajar con gráficos y figuras de otras librearías como *Matplotlib*.

También será utilizado para la presentación dinámica de resultados.

Wget

Wget[16] es una herramienta gratuita utilizada para la descarga de contenido web utilizando múltiples protocolos como *HTTPS* o *FTPS*. Permite la gestión de directorios y operaciones en segundo plano.

Gdown

Gdown[17] ha sido creada por un grupo de usuarios en *Github* que permite la descarga de ficheros de un tamaño significativo alojados en *Google Drive*.

Su uso se destinará a la gestión del modelo utilizado en la red neuronal para posibilitar su actualización una vez desplegada la aplicación.

[14] Matplotlib: https://matplotlib.org/
[15] Plotly: https://plotly.com/
[16] Wget:https://www.gnu.org/software/wget/
[17] Gdown: https://github.com/wkentaro/gdown

4.5. Documentación

LaTeX

LaTeX[18] es un sistema para la generación de textos que incluye tanto texto plano como comandos para la elaboración automática documentos con formatos y presentaciones elaboradas.

Overleaf

Overleaf[19] es un editor online del sistema LaTeXque permite la creación y compilación de este tipo de documentos de forma remota además de incluir corrector ortográfico, directorio de proyectos y sistema colaborativo de forma gratuita.

Draw.io

Draw.io[20] es una aplicación para la generación de diagramas, es gratuita y contiene multitud de plantillas para hacer desde diagramas de clases hasta gráficos científicos más elaborados.

[18] LaTeX: https://www.latex-project.org/
[19] Overleaf: https://www.overleaf.com/
[20] Draw.io: https://www.diagrams.net/

Aspectos relevantes del desarrollo del proyecto

A continuación se detalla el proceso de registro y posterior entrenamiento de la red con nuestras propias imágenes. Posteriormente se explicará el proceso y evolución del modelo conforme se han ido aumentando las iteraciones en la fase de entrenamiento.

Ejemplo de imágenes con las que se cuenta y una máscara de ejemplo:

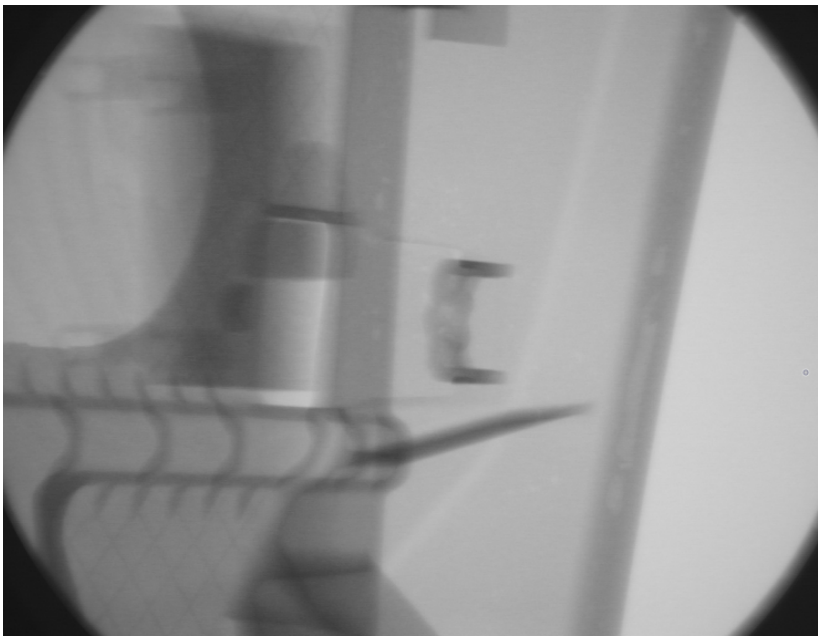

Figura 5.1: Ejemplo de imagen con defectos

Figura 5.2: Ejemplo de máscara binaria

5.1. Registro del conjunto

El primer paso que se ha llevado a cabo en este proyecto a la hora de trabajar con las imágenes propias ha sido el *registro* del conjunto de imágenes en el formato *COCO* con el que trabaja *Detectron 2*. Al final del proceso se dispondrá de un fichero en formato *JSON* que acompañará al conjunto de imágenes. La estructura se ha comentado anteriormente en la sección Formato de objetos COCO: *Common Objects in COntext*.

Pasos importantes :

1. **Organización del conjunto**: Es importante antes de comenzar a trabajar con las imágenes, disponerlas de forma que al recorrer las imágenes estén todas a un mismo nivel de directorios y con unos nombres que identifiquen claramente tanto la propia imagen como sus distintas máscaras.

   Estructura de nombres:

   - **Nombre de imagen**: "P0001_0001.png"
   - **Máscaras de la imagen**: "P0001_0001_0.png", "P0001_0001_1.png"...

2. **Imágenes:** La primera sección del fichero *JSON* suelen ser los ficheros de imagen, este apartado recoge:

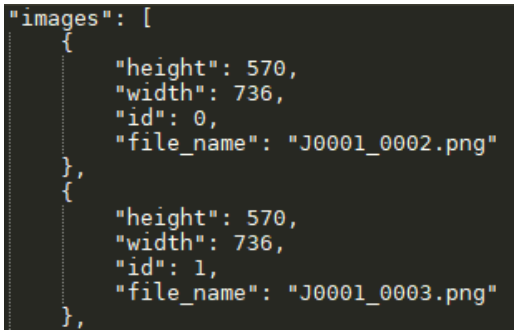

Figura 5.3: Sección de imágenes en el *JSON*

- Dimensiones de la imagen
- Identificador de la imagen
- Nombre del fichero

El nombre del fichero imagen es importante para que posteriormente puedan localizarse correctamente las imágenes.

3. **Categorías ó Clases:** Esta sección define las clases de objetos presentes en el conjunto

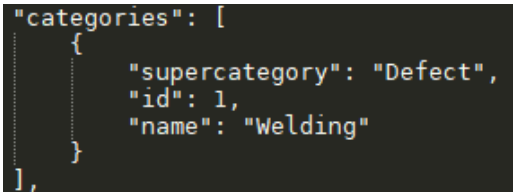

Figura 5.4: Sección de categorías en el *JSON*

- "Supercategoría" o categoría padre
- Identificador de la categoría o clase
- Nombre de la categoría

En este caso solo se cuenta con una clase *Welding*, la categoría padre es irrelevante y el nombre se tiene en cuenta sólo si se desea añadir metadatos, etiquetas en los resultados.

4. **Anotaciones:** En esta última sección se recoge la información de cada defecto recogido en las máscaras binarias, siguiendo la estructura antes vista en la sección Estructura de las anotaciones JSON COCO


```
"annotations": [
    {
        "id": 0,
        "image_id": 0,
        "category_id": 1,
        "iscrowd": 0,
        "area": 98,
        "bbox": [
            151.0,
            31.0,
            12.0,
            11.0
        ],
        "segmentation": [
            [
                159.0,
                41.998039215686276,
                150.00196078431372,
                37.0,
                154.0,
                30.001960784313727,
                160.99803921568628,
                31.0,
                162.99803921568628,
                38.0,
                159.0,
                41.998039215686276
            ]
        ],
        "width": 736,
        "height": 570
    },
```


Figura 5.5: Sección de anotaciones en el *JSON*

- ***id*:** Identificador del defecto actual
- ***image_id*:** Identificador de la imagen en la que se encuentra
- ***category_id*:** Identificador de la categoría o clase a la que pertenece
- ***iscrowd*:** Indicador de conjunto o unicidad
- ***area*:** Área total del defecto

- ***bbox*:** *Bounding box* que "enmarca" el defecto
- ***segmentation*:** Lista de pares de coordenadas 2 a 2 que recorren el contorno del efecto
- ***width y height*:** Dimensiones de la imagen de máscara

5.2. Entrenamiento

Esta fase se ha llevado a cabo en *Google Colaboratory* siguiendo las instrucciones de instalación, configuración y entrenamiento recogidas en la documentación oficial de *Detectron2*[3].

Se dispone de 21 imágenes etiquetadas de las cuales 11 se destinarán al entrenamiento y las 10 restantes a test y evaluación de resultados.

Estructura de ficheros

Organización de ficheros e imágenes que se ha seguido durante el entrenamiento. Es recomendable tener en cuenta que dentro de la carpeta *detectron2_repo* se encuentran las librerías propias a importar y para evitar avanzar y retroceder de directorio se recomienda que los ficheros recurrentes estén accesibles.

```
/
├── detectron2_repo
├── IMG
│   ├── images
│   │   ├── P0001_0000.png
│   │   └── P0001_0001.png
│   └── test
│       ├── images
│       ├── info.json
│       │   └── ..
│       └── info.json
└── output
    └── model_final.pth
```

Figura 5.6: Estructura del directorio de entrenamiento

- **detectron2_repo:** Directorio de instalación de *Detectron2*
- **images:** Directorio con las imágenes originales
- **test:** Subdirectorio con la misma estructura, imágenes de test y fichero de registro
- **info.json:** Fichero *JSON* generado durante el registro de las imágenes
- **output:** Directorio de salida de *Detectron2*, puede configurarse dentro del código
- **model_final.pth:** Fichero de pesos final que se utilizará para instanciar el objeto *Predictor* que posteriormente será el que ejecute el modelo entrenado

Registrar instancias

Una vez dispuesto el directorio lo primero es registrar las imágenes en *Detectron2*, para ello se indica el directorio donde se encuentran las imágenes y el fichero *JSON* que contiene su información.

```
register_coco_instances("weldings", {}, "./IMG/info.json", "./IMG/images")
register_coco_instances("test_weldings", {}, "./IMG/test/info.json", "./IMG/test/images")

dataset_dicts = DatasetCatalog.get("weldings")
```

Figura 5.7: Registro de las imágenes

Opcionalmente también es posible registrar en metadatos la información de etiquetas en caso de que se quisiera añadir como nombre de clases o categorías.

```
training_metadata = MetadataCatalog.get("weldings")
test_metadata = MetadataCatalog.get("weldings")
```

Figura 5.8: Registro de metadatos

En caso de haber cualquier fallo o incompatibilidad entre el fichero y las imágenes, en este punto se nos indicaría que por ejemplo el identificador de imagen de un defecto no encuentra ninguna imagen con dicho identificador o que la segmentación no es correcta.

Es recomendable comprobar con ayuda del objeto *Visualizer* de *Detectron2* que el registro ha sido satisfactorio y las imágenes están correctamente registradas.

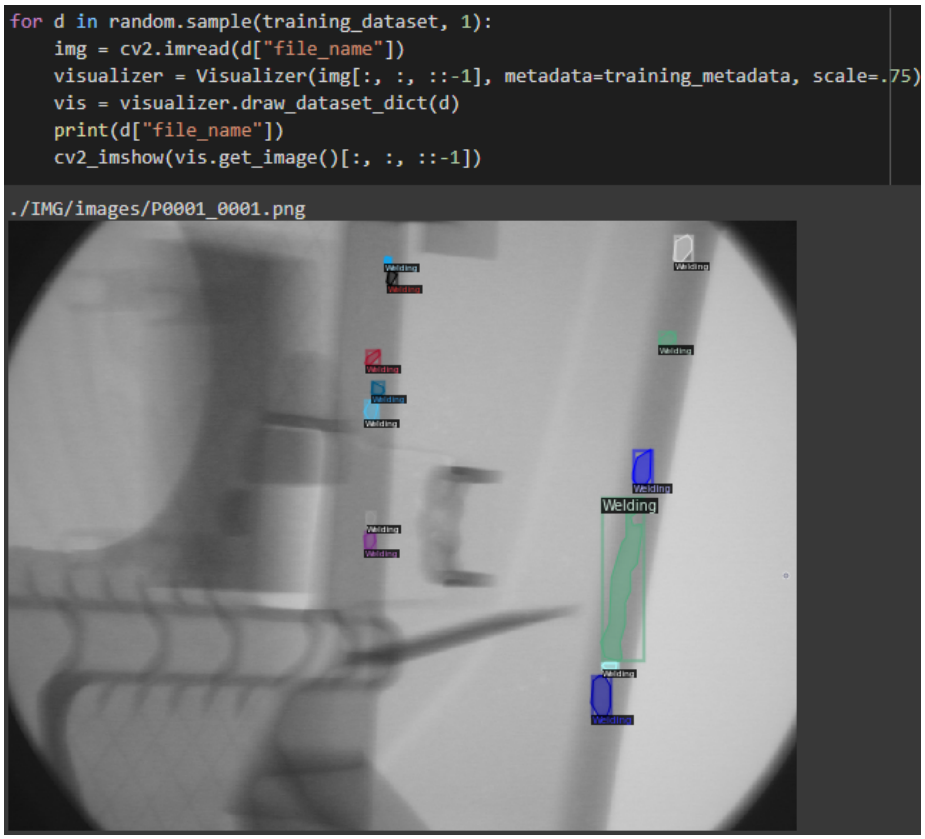

Figura 5.9: Resultado de registrar las imágenes

Configuración del entrenamiento

Después de ver que las imágenes están correctamente registradas se procede con la configuración del propio entrenamiento. Hay algunos parámetros que se recomienda dejar como aparecen en la mayoría de guías.

```
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
import os

cfg = get_cfg()
cfg.merge_from_file("./detectron2_repo/configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
cfg.DATASETS.TRAIN = ("training_dataset",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = "detectron2://COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x/137849600/model_final_f10217.pkl"
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.01
cfg.SOLVER.MAX_ITER = 500
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1  #

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
```

Figura 5.10: Configuración del entrenamiento

1. Primero se recoge en una variable la configuración vacía,
    - En este caso se va a utilizar el objeto *DefaultTrainer* que recoge configuración ya presente en otros modelos de *Detectron2* y nos permite sobreescribirla según las necesidades del caso. Otra alternativa más costosa ya que la configuración se construye desde la base es el objeto *SimpleTrainer*[4]
2. Se carga el fichero de configuración existente del modelo *R50 FPN* para a continuación modificar la configuración
3. **DATASETS.TRAIN:** Conjunto *COCO* ya registrado sobre el que se va a entrenar
4. **MODEL.WEIGHTS:** Conjunto de pesos de *model zoo*, son los pesos por defecto del conjunto incluído en *Detectron2*
5. **BASE_LR:** *Learning Rate*
6. **MAX_ITER:** Número de iteraciones
7. **NUM_CLASSES:** Número de clases presentes en el conjunto

8. Finalmente se genera un directorio de salida en este caso *output* en caso de no existir

Lanzamiento del entrenamiento

Antes de lanzar el entrenamiento y para comprobar rápidamente que *Google Colaboratory* no tiene ninguna incidencia en ese momento, mediante la función *cuda.is_available()* de la librería *Torch* se puede obtener una traza que nos confirme que todo funcione y que se tenga correctamente activada la GPU en el notebook

```
import torch, torchvision

if torch.cuda.is_available():
  print('GPU available')
  !pip show torch
else:
  print('Please set GPU via Edit -> Notebook Settings.')
  !pip show torch
```

```
GPU available
Name: torch
Version: 1.8.1+cu101
Summary: Tensors and Dynamic neural networks in Python with strong GPU acceleration
Home-page: https://pytorch.org/
Author: PyTorch Team
Author-email: packages@pytorch.org
License: BSD-3
Location: /usr/local/lib/python3.7/dist-packages
Requires: numpy, typing-extensions
Required-by: torchvision, torchtext, fastai
```

Figura 5.11: Disponibilidad de la GPU

Una vez confirmado que todo es correcto, se instancia el objeto *Default-Trainer* con la *cfg* o configuración que se acaba de ajustar y se lanza el entrenamiento.

```
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Figura 5.12: Lanzamiento del entrenamiento

Al utilizar la máquina de *Google Colaboratory* depende del momento en el que se realice el entrenamiento para que el proceso se demore en mayor o menor medida. En caso de haber imágenes incompatibles o que algunas dimensiones no sean correctas, *DefaultTrainer* intentará corregirlas y en caso de no ser posible, descartará las imágenes.

El resultado es una traza similar a la siguiente:

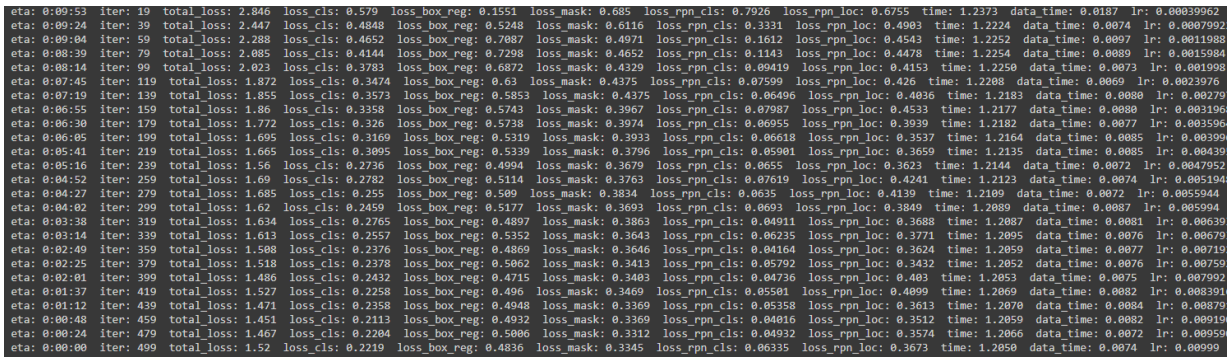

Figura 5.13: Traza del entrenamiento

Significado de los términos:

- **eta**: Tiempo restante del entrenamiento
- **iter**: Iteración, cada 100, en la que se encuentra el entrenamiento
- **loss**: Medida de rendimiento que engloba las posteriores
- **loss_cls**: Rendimiento al reconocer las clases
- **loss_box_reg**: Rendimiento al reconocer las regiones de objeto
- **loss_mask**: Rendimiento de las segmentaciones o *pred_masks*
- **loss_rpn_cls**: Rendimiento al reconocer la clase en las imágenes
- **loss_rpn_loc**: Rendimiento al localizar las clases en al imagen

Observaciones en el entrenamiento

Dos de los principales problemas en *Machine Learning* a la hora de entrenar una red neuronal son el *Overfitting* o sobreajuste, sobreentrenamiento y *Underfitting* o entrenamiento insuficiente.

Conforme se han ido haciendo pruebas con más o menos iteraciones y mayor o menor *Learning Rate* se ha realizado un entrenamiento prolongado de hasta las 1500 iteraciones para comprobar, junto con la herramienta *TensorBoard*, la evolución del *Loss* durante el entrenamiento y posterior validación con el conjuto de test.

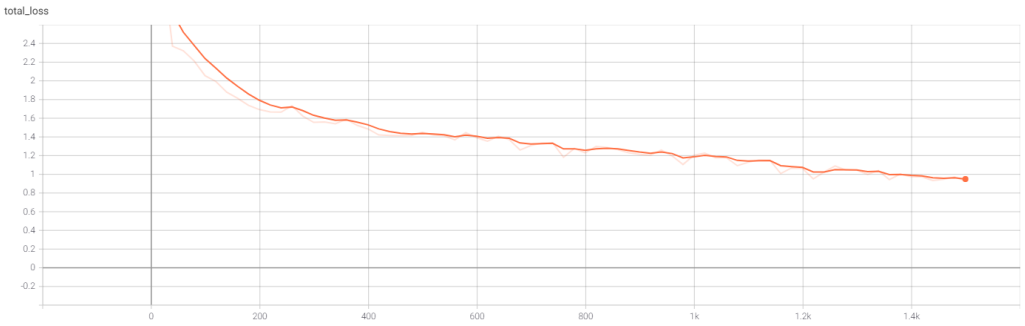

Figura 5.14: Evolución del Loss

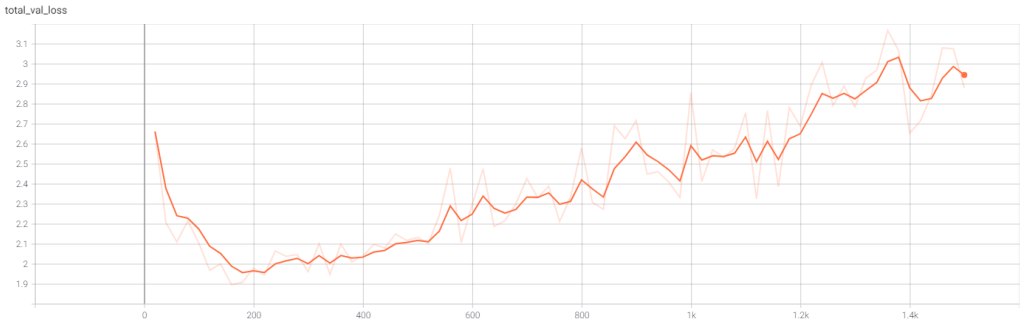

Figura 5.15: Evolución del *Loss* de Validación

Como se puede observar, alrededor de las 500 iteraciones el incremento del *Loss* nos indica que se comienza a producir el *Overfitting*. Esto lo que quiere decir es que el modelo se comportará de forma deficiente si se encuentra con objetos ligeramente diferentes a los del conjunto de entrenamiento y no los reconocerá correctamente. Si a partir de ese punto el entrenamiento se prologa y se aumenta el sobre ajuste, el resultado final que tendremos será que el conjunto de entrenamiento se está aprendiendo "en exceso", provocando que si el modelo se encuentra con imágenes con ciertas diferencias a las del conjunto de entrenamiento, el modelo fallará.

Además de estos parámetros, se han definido otras métricas que son *Precision*, *Recall* y *F1*.

Evolución en las detecciones

Para analizar la evolución del rendimiento de la red conforme avanza el entrenamiento, se han entrenado distintos modelos con un número determinado de iteraciones y ejecutado sobre las mismas imágenes de test analizando su comportamiento.

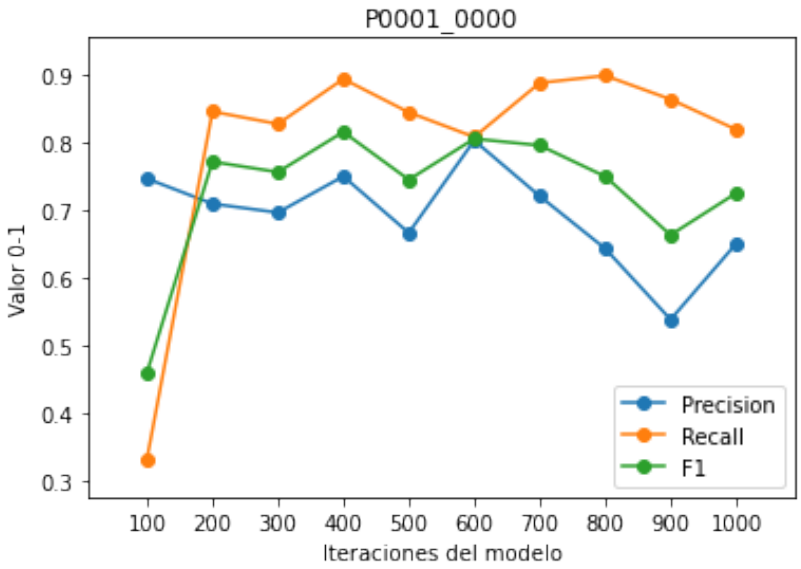

Figura 5.16: Evolución del rendimiento Caso 1

La evolución del rendimiento en todos los casos alcanza su máximo en torno a la iteración 500-600 y no se observa una mejora más allá de dichas iteraciones. Se ha contemplado también el componente de aleatoriedad durante el entrenamiento por lo que también se han evaluado modelos con las mismas iteraciones paralelamente, siendo la conclusión final la misma.

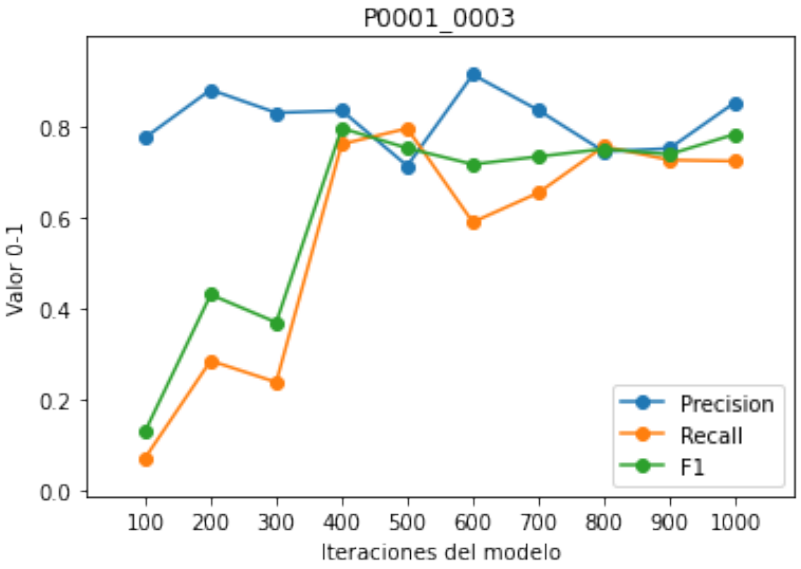

Figura 5.17: Evolución del rendimiento Caso 2

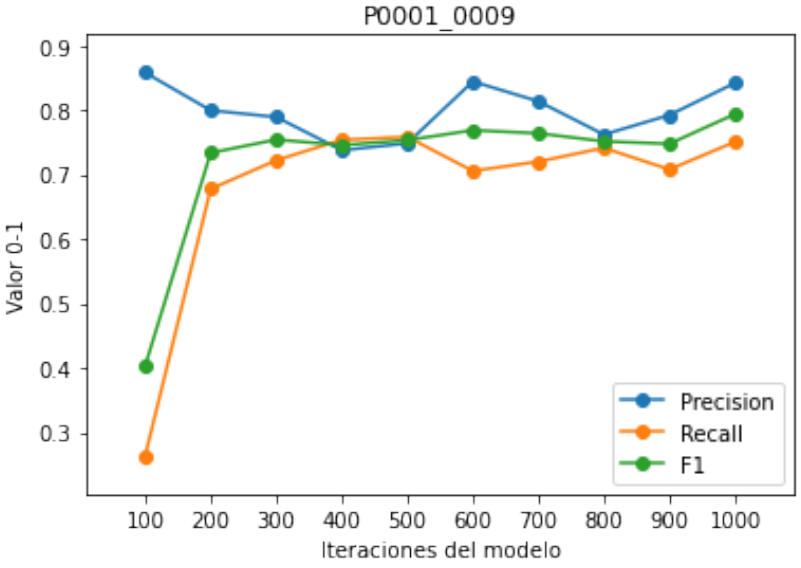

Figura 5.18: Evolución del rendimiento Caso 3

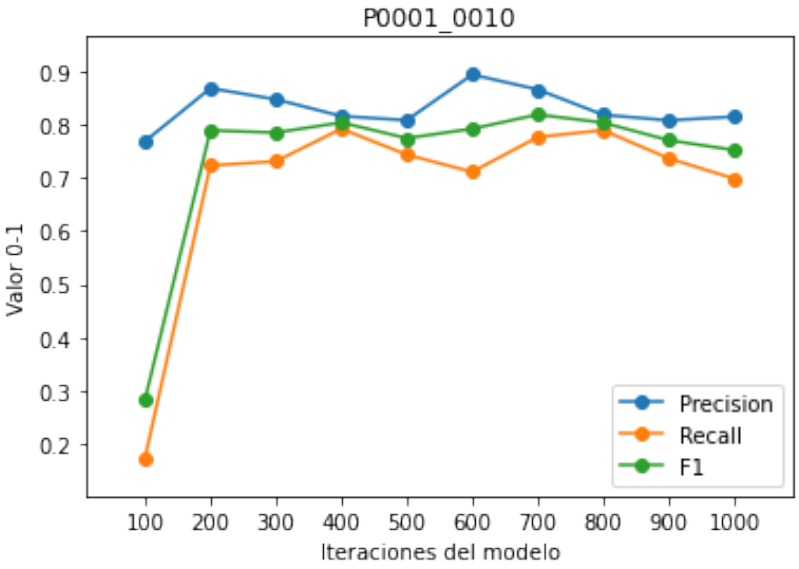

Figura 5.19: Evolución del rendimiento Caso 4

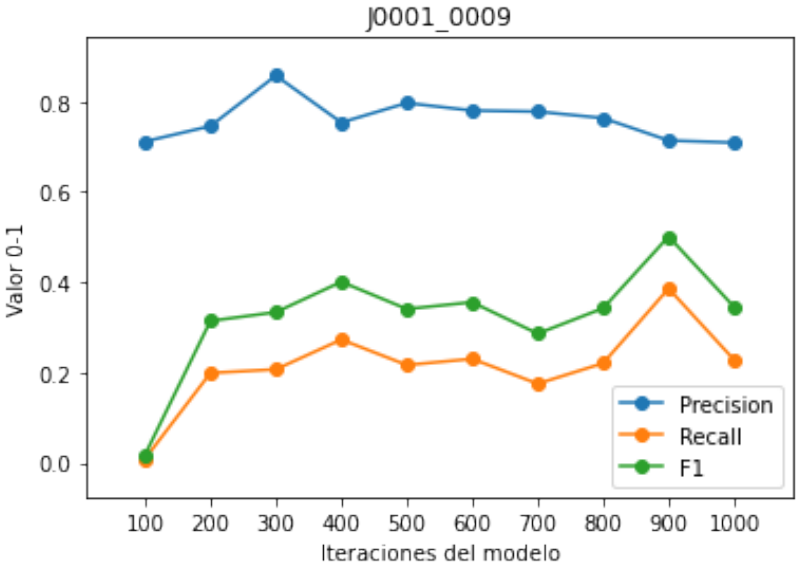

Figura 5.20: Evolución del rendimiento Caso 5

A sí mismo se han observado 2 fenómenos adicionales conforme avanzaba el sobre entrenamiento u *Overfitting*.

- **Solapamiento de las detecciones:** En determinadas ocasiones, el modelo hace 2 predicciones sobre un mismo punto en el que la confianza que recibe la detección es diferente para distintas zonas del área marcada, por lo que puede darse el caso al trabajar con imágenes como estas, que un defecto esté marcado sobre otro. Este fenómeno casi es inexistente en las primeras iteraciones y pasadas las 600 se observa de manera más frecuente.

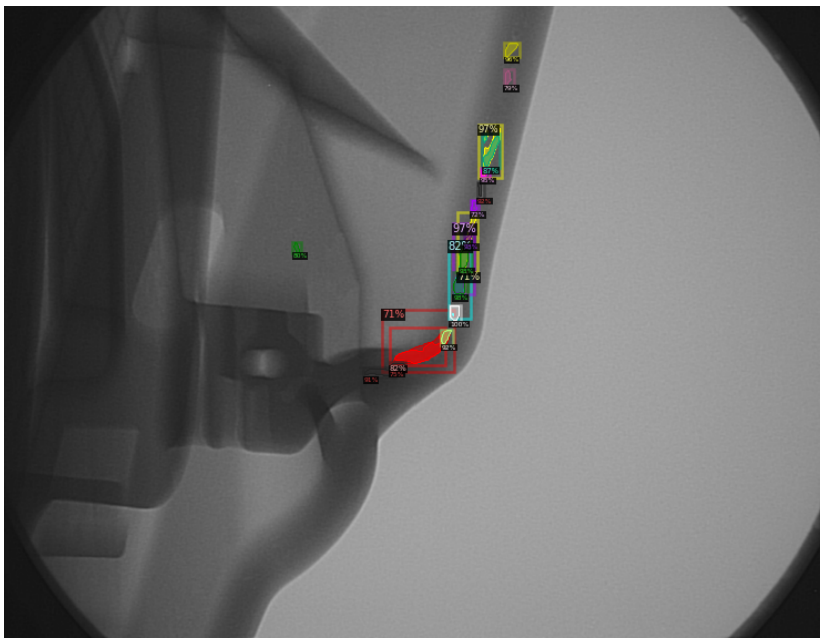

Figura 5.21: Ejemplo de superposición de los defectos

- **Incremento de los falsos positivos:** El modelo se comporta bastante bien en cuanto a falsos positivos, pero las imágenes cuentan con determinadas figuras propias de piezas metálicas, en nuestro caso cabezales o agujeros roscados, que presentan una forma circular similar a la de un defecto y que puede dar lugar a la confusión del modelo. Conforme se incrementan las iteraciones más allá de las 700 iteraciones, el incremento de este fenómeno también es notable.

Figura 5.22: Ejemplo de falso positivo

Trabajos relacionados

Research on Approaches for Computer Aided Detection of Casting Defects in X-ray Images with Feature Engineering and Machine Learning

Autores: *Du Wangzhe, Hongyao Shen, Fu Jianzhong, Ge Zhang, Quan He*

Trabajo de la Universidad de Zhejiang, año 2019 donde se contempla a nivel general las técnicas de CNN para la detección en imágenes y defectos de piezas metálicas. En el trabajo se contemplo tanto *Fast R-CNN* como *Faster R-CNN* para el funcionamiento, evaluando la *Precision* y *Recall* de ambos en un conjunto de 2236 imágenes.

Conforme se fueron observando los resultados, se observó que tanto el recorte de la propia imagen, posición de la pieza, brillo de la misma y el fondo, alteraban notablemente los resultados de las detecciones. Proponen además la utilización de histogramas para el realce de los defectos presentes en la imagen, aunque al trabajar con escala de grises, puede llevar a errores si hay presencia de partes similares a los defectos.

Concluyen que han tenido mejores resultados con *Fast R-CNN* que con *Faster R-CNN*, junto con una manipulación de las imágenes que incluía giros de 45º y recortes aleatorios de las mismas en distintas regiones. [8]

Real-Time Tiny Part Defect Detection System in Manufacturing Using Deep Learning

Autores: *Jing Yang, Shaobo Li, Zheng Wang, Guanci Yang*

En este proyecto[17] del año 2019 tenía como objetivo la aplicación del *Deep Learning* a la detección en tiempo real de defectos en agujas. En este caso se utilizó una red *SSD* [9] basada en la regresión de *Faster R-CNN*.

El funcionamiento era algo más complejo ya que se diseñó el sistema con una cámara de entrada de vídeo, una *Raspberry Pi 3* que ejecutaba el modelo y una pantalla de seguimiento en vivo. La cámara tomaba como entrada un *conveyor belt* o cinta transportadora que se movía a en torno a 7 metros por segundo. Se observaban las agujas recién fabricadas y se clasificaba cada una con los 4 tipos de defectos más comunes o por el contrario como una aguja correctamente fabricada.

Se compararon resultados según algoritmos utilizados y el tipo de defecto que podían presentar las agujas y se llegó a la conclusión de que la combinación de *Faster R-CNN* y *SSD* mejoraba notablemente los resultados. A pesar de todo había un tipo de defecto que presentaba una peor detección en todos los algoritmos, con un porcentaje de acierto notablemente bajo. Finalmente se apuntó a la búsqueda de una solución futura ya que no se dió con una solución.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusión

El principal objetivo del proyecto se ha cumplido correctamente. Partiendo de un número de imágenes segmentadas por los propios tutores se ha conseguido entrenar un modelo que realmente funcione y los detecte en su mayor parte.

La principal limitación era el tamaño de este debido al elevado esfuerzo que representa la segmentación manual de imágenes de este tipo. Se ha buscado balancear el conjunto de entrenamiento de tal manera que la detección cubriese el mayor tipo de defectos posibles, a pesar de que en un determinado grupo de imágenes defectos muy pequeños no siempre se detectan como se esperaba.

Si bien no siempre los resultados son negativos ya que a veces se tiene a agrupar dichos defectos en lugar de una detección individualizada, alterando las métricas empleadas. En relación a trabajos anteriores y en concreto al realizado por *Noelia Ubierna Fernández* realizado el año pasado, se observó que el conjunto utilizado presentaba las máscaras de forma diferente, presentando una imagen por defecto que luego se fusionaba para generar la máscara binaria final.

Se procedió a la separación de las máscaras propias, conteniendo cada una un defecto por separado y comprobando finalmente que los resultados eran los mismos que al utilizar una máscara etiquetada de forma única que contuviese todos los defectos.

En cuanto a las herramientas, ha sido muy interesante profundizar en el uso del lenguaje *Python* de una forma más extendida, aplicándose también para el despliegue de la herramienta en formato Web.

La apariencia y composición de la Web no es especialmente compleja, pero si representa una buena introducción tanto al desarrollo web de cara al usuario por un lado, y al servidor por otro. También ha sido muy útil la puesta en práctica de la herramienta *Docker*, que hace posible programar, probar y ejecutar herramientas en cualquier sistema sin preocuparse de incompatibilidades posteriores.

7.2. Líneas de trabajo futuras

Ideas para la mejora del proyecto en el futuro:

1. **Ampliar el conjunto de imágenes:** Si bien como ya se ha comentado es complicado el proceso de segmentación de imágenes, podría representar un punto de inflexión en el rendimiento del modelo en un futuro

2. **Explorar diferentes algoritmos:** *Detectron2* es una herramienta muy potente que además permite el uso de diferentes algoritmos. En este caso se ha usado *Faster-RCNN* por el buen desempeño que presenta en otras labores de detección en imágenes, pero podría darse el caso de que otro de los algoritmos diferentes funcionase mejor.

3. **Revisión del despliegue:** Actualmente y a pesar de que está planificado, *Detectron2* no es oficialmente compatible con *Windows*, si bien hay usuarios que han intentado compatibilizar dependencias, la instalación no siempre es segura. Por ello puede ser que en un futuro la instalación y uso de la herramienta cambiase y habría de tenerse en cuenta.

4. **Mejoras en la Aplicación Web:** *Flask* es una gran herramienta que permite el desarrollo de aplicaciones web en *Python* pero de una forma rápida, por ello presenta algunas limitaciones y no es tan polivalente como otros lenguajes y *frameworks* que podrían hacer de la aplicación web una herramienta más avanzada.

Bibliografía

[1] Aphex34. Max pooling. https://commons.wikimedia.org/wiki/File:Max_pooling.png, 2015.

[2] Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik UC Berkeley. Rich feature hierarchies for accurate object detection and semantic segmentation tech report (v5). 2014.

[3] Detectron2. Detectron2 documentation. https://detectron2.readthedocs.io/en/latest/index.html, 2020.

[4] Detectron2. Detectron2 training. https://detectron2.readthedocs.io/en/latest/tutorials/training.html#custom-training-loop, 2020.

[5] Frank Emmert-Streib, Zhen Yang, Han Feng, Shailesh Tripathi, and Matthias Dehmer. An introductory review of deep learning for prediction models with big data. *Frontiers in Artificial Intelligence*, 3:4, 2020.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[7] Google. Clasificación: Precisión y exhaustividad. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall?hl=es-419, 2020.

[8] Du Wangzhe Hongyao Shen Fu Jianzhong Ge Zhang Quan He. Approaches for accuracy improvement of the x-ray image defect detection of automobile casting aluminum parts based on deep learning. 2019.

[9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[10] MatWorks. Machine learning, tres cosas que es necesario saber. https://es.mathworks.com/discovery/machine-learning.html.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[12] Adrian Rosebrock. Intersection over union - poor, good and excellent score. https://commons.wikimedia.org/wiki/File:Intersection_over_Union_-_poor,_good_and_excellent_score.png, 2016.

[13] Adrian Rosebrock. Intersection over union - visual equation. https://commons.wikimedia.org/wiki/File:Intersection_over_Union_-_visual_equation.png, 2016.

[14] Francesco Visin Vincent Dumoulin. Convolution arithmetic - padding strides odd. https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Padding_strides_odd.gif, 2016.

[15] Walber. Precisionrecall.svg. https://en.wikipedia.org/wiki/File:Precisionrecall.svg, 2014.

[16] Wikipedia. Kunihiko Fukushima — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Kunihiko%20Fukushima&oldid=1001400557, 2021.

[17] Jing Yang Shaobo Li Zheng Wang Guanci Yang. Real-time tiny part defect detection system in manufacturing using deep learning. 2019.