



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

Tutores: José Francisco Diez Pastor y Pedro
Latorre Carmona

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	v
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Desarrollo temporal	1
A.3. Estudio de viabilidad	4
Apéndice B Especificación de Requisitos	9
B.1. Introducción	9
B.2. Objetivos generales	9
B.3. Catálogo de requisitos	9
B.4. Especificación de requisitos	10
Apéndice C Especificación de diseño	15
C.1. Introducción	15
C.2. Diseño de datos	15
C.3. Diseño procedimental	17
C.4. Diseño arquitectónico	21
Apéndice D Documentación técnica de programación	23
D.1. Introducción	23
D.2. Estructura de directorios	23
D.3. Manual del programador	25

D.4. Compilación, instalación y ejecución del proyecto	27
D.5. Pruebas del sistema	30
Apéndice E Documentación de usuario	41
E.1. Introducción	41
E.2. Requisitos de usuarios	41
E.3. Instalación	41
E.4. Manual del usuario	42
Bibliografía	49

Índice de figuras

B.1. Diagrama de casos de uso de la aplicación	10
C.1. Diagrama de clases	17
C.2. Diagrama de secuencia para la cargar una imagen	18
C.3. Diagrama de secuencia para la predicción y muestra de defectos	19
C.4. Diagrama de secuencia para la muestra de la información de los defectos	20
C.5. Diagrama de secuencia para la muestra de las máscaras de los defectos	20
D.1. Directorios del proyecto	24
D.2. Imágenes y sus máscaras con los defectos	31
D.3. Imagen con el cuadro delimitador calculado a partir de la máscara	32
D.4. El código para la visualización de la figura D.3	32
D.5. Imagen redimensionada a (1024x1024)	33
D.6. El código para la visualización de la figura D.5	34
D.7. Imagen de prueba con máscaras correspondientes	35
D.8. Imagen de prueba con los <i>bounding box</i> de esas máscaras	35
D.9. El código para la visualización de las figuras D.7 y D.8	36
D.10.Imagen de prueba con sus mini máscaras correspondientes	36
D.11.Imagen de prueba con los <i>bounding box</i> de esas mini máscaras	37
D.12.El código para la visualización de las figuras D.10 y D.11	37
D.13.Imagen con sus anclajes	39
D.14.El código para la visualización de la figura D.13	40
E.1. Carga de una imagen en la aplicación	42
E.2. Detectar defectos de una imagen	43
E.3. Visualizar las máscaras de los defectos de la imagen	44

E.4. Máscaras de los defectos de la imagen	44
E.5. Visualizar la información de los defectos de la imagen	45
E.6. Información de los defectos de la imagen	46
E.7. Problema al cargar la imagen	47
E.8. El archivo cargado no es una imagen	47
E.9. No existe el directorio seleccionado	47
E.10. No hay una imagen cargada para detectar defectos	47
E.11. No hay defectos detectados en la imagen	47

Índice de tablas

A.1. Costes de personal	5
A.2. Coste Total	6
A.3. Licencias bibliotecas <i>Python</i>	8
B.1. Caso de uso 0: Cargar imagen.	11
B.2. Caso de uso 1: Detectar defectos.	12
B.3. Caso de uso 2: Visualizar máscaras.	12
B.4. Caso de uso 3: Visualizar información de los defectos.	13
D.1. Configuración de las anclas	38
D.2. Forma del mapa de características	39

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En esta sección se describirá el desarrollo temporal del proyecto y un estudio de viabilidad.

En el desarrollo temporal se divide el trabajo por fases o etapas y el estudio de viabilidad se puede dividir en dos apartados: viabilidad económica y viabilidad legal.

A.2. Desarrollo temporal

Las fases del desarrollo temporal de este proyecto son:

- **Fase 1:** Investigación del estado del arte sobre aplicación de redes neuronales a la detección de defectos en imágenes (incluyendo las de rayos-X): revisión de algoritmos y de implementaciones.
- **Fase 2:** Configuración de la red neuronal escogida (modelo, impresión, etc.).
- **Fase 3:** Entrenamiento de una red neuronal con imágenes de radiografías: configuración, adaptación, evaluación de parámetros, etc.
- **Fase 4:** Creación de una aplicación para la utilización dinámica de la red: aplicaciones amigables (web o escritorio) para poder utilizar el sistema inteligente en la Fase 2.

Para llevar a cabo estas fases he seguido una metodología *Scrum* donde se realizan *sprints* que tienen unos objetivos. La duración de cada *sprint* depende de las tareas asignadas a él que se irán creando y realizando a lo largo del proyecto.

Se ha utilizado *GitHub* para el seguimiento del proyecto. El repositorio del proyecto, y las *issues* realizadas se encuentra en el <https://github.com/nuf1001/XRayDetector>

Sprint 0

Tareas realizadas:

- Investigar sobre segmentación de imágenes de rayos X.
- Mirar convocatoria prototipos orientados al mercado.
- Investigar sobre la documentación en L^AT_EX.

En este sprint se realizaron las primeras investigaciones sobre el tema del proyecto.

La convocatoria prototipos orientados al mercado tiene como objetivo desarrollar actividades de transferencia de conocimiento entre el colectivo de estudiantes de la Universidad de Burgos mediante la materialización y desarrollo de un prototipo, a través de proyectos fin de grado o fin de máster.

Sprint 1

Tareas realizadas:

- Probar código correspondiente al artículo [2] en Windows.
- Probar código correspondiente al artículo [2] en Linux.
- Empezar introducción de la documentación.

En este sprint se prueba el código de un artículo relacionado con nuestro proyecto [2] y se inicia con la documentación.

Este artículo es que hemos utilizado de base para empezar a desarrollar el proyecto. Funcionaba en los dos sistemas operativos, pero se acabó desarrollando en Windows 10 por comodidad.

Sprint 2

Tareas realizadas:

- Empezar con el código de la app de escritorio.
- Entrenar la red en el ordenador Alpha de la universidad.
- Investigar para la app de escritorio.
- Crear diseño de la app de escritorio.

En este sprint se empieza con el diseño y el código de la aplicación de escritorio. También se entrena la red con un ordenador más potente al que me dio acceso la universidad. Esta última tarea tardó bastante más de lo esperado en realizarse por problemas con la red y el ordenador.

Sprint 3

Tareas realizadas:

- Averiguar/investigar cuál es la salida de la red al evaluar una imagen.
- Hacer gráficas del *loss* del entrenamiento.

En este sprint se evalúa la salida y se crean las gráficas con los valores obtenidos. Estas gráficas son las que podemos ver en el apartado 5.2 de la memoria.

Sprint 4

Tareas realizadas:

- Primer boceto de la memoria.
- Memoria - Conceptos teóricos.
- Memoria - Técnicas y herramientas.
- Memoria - Aspectos relevantes del desarrollo del proyecto.
- Memoria - Trabajos relacionados.
- Memoria - Conclusión.

En este sprint se empieza con la documentación final de la memoria.

Sprint 5

Tareas realizadas:

- Primer boceto de los anexos.
- Anexos - Especificación de diseño.
- Anexos - Documentación técnica de programación.
- Anexos - Documentación de usuario.

En este sprint se empieza con la documentación final de los anexos.

Sprint 6

Tareas realizadas:

- Entrega del TFG.
- Subir código.

Este es el último sprint y en él terminamos con toda la entrega del proyecto.

A.3. Estudio de viabilidad

En este apartado se abordará la viabilidad económica y legal de este proyecto.

Viabilidad económica

En este apartado, simularemos el coste del proyecto que tendría en una empresa, o en una venta al público.

Coste personal

Este proyecto cuenta con dos **profesores contratados** durante 6 meses para este proyecto. Por cada profesor se asignan 0.5 créditos, por lo que el coste¹ por tutor consideramos que es el siguiente:

¹https://www.ubu.es/sites/default/files/portal_page/files/pdi_laboral_2019.pdf

- Sueldo ayudante doctor: 26445.24 euros anuales. Más 2 trienios más el complemento de mejora: $26445.24 + 1260.58 + 1110.71 = 28816.52$. En total, imparte 24 créditos por lo que el coste anual por crédito es de 1200.68 euros. El coste total es el siguiente:

$$1200.68 \cdot 0.5 \text{ créditos} = 600.34 \text{ €}$$

- Sueldo medio anual de un titular: 42220.34 euros anuales. En total imparte 24 créditos por lo que el coste anual por crédito es de 1759.18 euros. El coste total es el siguiente:

$$1759.18 \cdot 0.5 \text{ créditos} = 879.59 \text{ €}$$

Por último, el desarrollador del proyecto. Supondremos un salario bruto de 2000 euros para el desarrollador. Habrá que tener en cuenta los gastos de seguridad social.

- Cotización por parte de la empresa: 23.6 %
- Cotización por parte del empleado: 4.7 %
- **Total: 28.3 %**

Por lo tanto, el coste del desarrollador será:

Concepto	Coste
Salario mensual neto	1217.25
Retención IRPF (15 %)	216.75
Seguridad social (28.3 %)	566
Salario mensual bruto	2000
Coste total (6 meses)	12000

Tabla A.1: Costes de personal

Coste informático

El coste informático es mínimo. Se ha utilizado un único portátil personal. Su coste fue de 750€ y se supone una amortización de 4 años. Para ello se contabilizará únicamente el coste amortizado.

$$\frac{750}{12 \cdot 4} \cdot 6 = 93,75e$$

Ingresos

Para este proyecto se ha contado con la concesión de la beca prototipos de la Oficina de Transmisión de Información de la Universidad de Burgos. La cuantía de la beca ha sido de 1000 euros, por lo que estos serán los ingresos del proyecto.

Coste Total

El coste total del proyecto será:

Concepto	Coste
Coste desarrollador	12000
Costes Tutores(15 %)	1479.93
Hardware	93.75
Beca prototipos	-1000
Coste total (6 meses)	12573.68

Tabla A.2: Coste Total

Viabilidad legal

Todo el código utilizado es de dominio público. Las bibliotecas de *Python* utilizadas tienen distintas licencias por lo que hay que comparar su compatibilidad.

Bibliotecas Python

En primer lugar, vamos a ver cuáles son las licencias de las bibliotecas de *Python* usadas en nuestro proyecto, ver tabla A.3:

Todas las licencias son compatibles entre sí por lo que todo el código fuente de la aplicación se ha licenciado bajo BSD (*Berkeley Software Distribution*).

Imágenes GDXRay

El conjunto de imágenes es el descrito en [4], en este artículo la viabilidad legal se describe como: “(...)Las imágenes están organizadas en una base de datos pública llamada GDXray que puede ser utilizado de forma gratuita, pero solo para fines educativos y de investigación(...)(p.1).

Por lo tanto, para la realización de este proyecto no habría ningún problema con la utilización de dichas imágenes.

Biblioteca	Versión	Licencia
Keras	2.1.3	Licencia MIT (<i>Massachusetts Institute of Technology</i>) o licencia X11. Es una licencia de software libre permisiva, es decir, impone muy pocas limitaciones en la reutilización.
TensorFlow	1.3.0	Licencia Apache. Es una licencia de software libre permisiva, es decir, requiere la conservación del aviso de derecho de autor y el descargo de responsabilidad, pero no es una licencia copyleft, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.
NumPy	1.18.1	Licencia BSD (<i>Berkeley Software Distribution</i>). Es una licencia de software libre permisiva, es decir, impone muy pocas limitaciones en la reutilización.
SciPy	1.1.0	Licencia BSD (<i>Berkeley Software Distribution</i>). Es una licencia de software libre permisiva, es decir, impone muy pocas limitaciones en la reutilización.
Matplotlib	3.1.3	Tiene su propia licencia libre. Es una licencia que usa código compatible con BSD (<i>Berkeley Software Distribution</i>), y su licencia se basa en la licencia de PSF (<i>Python Software Foundation</i>) que es una licencia de software libre permisiva (cumple con los requisitos OSI para ser declarada licencia de software libre).
PIL	7.0.0	Licencia de <i>Python Imaging Library</i> . Es una licencia de software libre permisiva
Scikit-image	0.16.2	Licencia BSD (<i>Berkeley Software Distribution</i>). Es una licencia de software libre permisiva, es decir, impone muy pocas limitaciones en la reutilización.
OpenCV	4.1.0	Licencia BSD (<i>Berkeley Software Distribution</i>). Es una licencia de software libre permisiva, es decir, impone muy pocas limitaciones en la reutilización.
Distutils	3.6.10	Licencia BSD (<i>Berkeley Software Distribution</i>). Es una licencia de software libre permisiva, es decir, impone muy pocas limitaciones en la reutilización.

Tabla A.3: Licencias bibliotecas *Python*

Apéndice B

Especificación de Requisitos

B.1. Introducción

En esta sección se abordarán los diferentes objetivos y requisitos del proyecto. Se presentarán tanto los requisitos globales del proyecto, como los requisitos funcionales y casos de uso de la aplicación.

B.2. Objetivos generales

Entrenar una red neuronal para obtener un modelo bien entrenado que sepa diferenciar correctamente las imperfecciones en las imágenes de rayos-x.

Crear una aplicación amigable y sencilla de utilizar para el usuario que cargue el modelo obtenido y te enseñe donde se encuentran los defectos, en el caso de que los haya.

B.3. Catálogo de requisitos

En esta sección se enumerarán los requisitos funcionales de nuestra aplicación.

- **RF-1** La aplicación tiene que permitir detectar los defectos de una pieza metálica a partir de una imagen de rayos-x.
 - **RF-1.1** El usuario podrá elegir la imagen que quiera predecir.
 - **RF-1.2** El usuario podrá ver la imagen con los defectos destacados con rectángulos.

- **RF-2** La aplicación tiene que permitir cargar imágenes para trabajar con ellas.
 - **RF-2.1** El usuario podrá elegir la imagen de cualquier ruta que quiere cargar en la aplicación.
- **RF-3** La aplicación tiene que permitir enseñar las máscaras de la imagen que ha sido evaluada.
 - **RF-3.1** El usuario podrá visualizar todas las máscaras generadas por el programa.
- **RF-4** La aplicación tiene que permitir ver información sobre los defectos de la imagen que ha sido evaluada.
 - **RF-4.1** El usuario podrá visualizar el *Bounding Box*, la clase y la marca de cada defecto detectado.

B.4. Especificación de requisitos

En esta sección se mostrar y desarrolla el diagrama de casos de uso resultante.

Diagramas de caso de uso

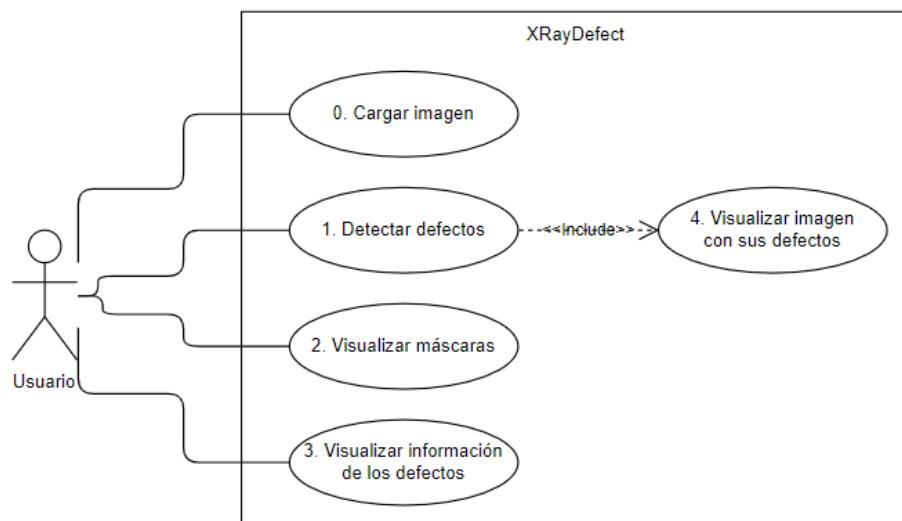


Figura B.1: Diagrama de casos de uso de la aplicación

Caso de uso 0: Cargar imagen.

Descripción	Permite al usuario cargar una imagen de su equipo a la aplicación.										
Requisitos	RF-2 RF-2.1										
Precondiciones	Ninguna										
Secuencia normal	<table border="1"> <thead> <tr> <th>Paso</th><th>Acción</th></tr> </thead> <tbody> <tr> <td>1</td><td>El usuario pincha el botón “Cargar imagen”.</td></tr> <tr> <td>2</td><td>Se despliega un menú de búsqueda de archivos.</td></tr> <tr> <td>3</td><td>Se eligen la imagen y se da a “Abrir”.</td></tr> <tr> <td>4</td><td>Se cargan la imagen dentro de la aplicación.</td></tr> </tbody> </table>	Paso	Acción	1	El usuario pincha el botón “Cargar imagen”.	2	Se despliega un menú de búsqueda de archivos.	3	Se eligen la imagen y se da a “Abrir”.	4	Se cargan la imagen dentro de la aplicación.
Paso	Acción										
1	El usuario pincha el botón “Cargar imagen”.										
2	Se despliega un menú de búsqueda de archivos.										
3	Se eligen la imagen y se da a “Abrir”.										
4	Se cargan la imagen dentro de la aplicación.										
Postcondiciones	La imagen aparece en pantalla.										
Excepciones	Error al cargar la imagen. El archivo no es una imagen. El directorio no existe.										
Importancia	Alta										
Urgencia	Alta										

Tabla B.1: Caso de uso 0: Cargar imagen.

Caso de uso 1: Detectar defectos.

Descripción	Permite al usuario obtener la predicción de los defectos de la imagen cargada.						
Requisitos	RF-1 RF-1.1 RF-1.2						
Precondiciones	Tener una imagen cargada						
Secuencia normal	<table border="1"> <tr> <td>Paso</td><td>Acción</td></tr> <tr> <td>1</td><td>El usuario pincha el botón “Detectar defectos”.</td></tr> <tr> <td>2</td><td>Aparece la predicción en pantalla.</td></tr> </table>	Paso	Acción	1	El usuario pincha el botón “Detectar defectos”.	2	Aparece la predicción en pantalla.
Paso	Acción						
1	El usuario pincha el botón “Detectar defectos”.						
2	Aparece la predicción en pantalla.						
Postcondiciones	La imagen aparece en pantalla con los defectos marcados con rectángulos.						
Excepciones	No hay una imagen cargada.						
Importancia	Alta						
Urgencia	Alta						

Tabla B.2: Caso de uso 1: Detectar defectos.

Caso de uso 2: Visualizar máscaras.

Descripción	Permite al usuario visualizar las máscaras de la imagen creadas.						
Requisitos	RF-3 RF-3.1						
Precondiciones	Tener una imagen cargada y que ya se haya hecho la detección de defectos.						
Secuencia normal	<table border="1"> <tr> <td>Paso</td><td>Acción</td></tr> <tr> <td>1</td><td>El usuario pincha el botón “Máscaras”.</td></tr> <tr> <td>2</td><td>Aparece las máscaras en una nueva ventana.</td></tr> </table>	Paso	Acción	1	El usuario pincha el botón “Máscaras”.	2	Aparece las máscaras en una nueva ventana.
Paso	Acción						
1	El usuario pincha el botón “Máscaras”.						
2	Aparece las máscaras en una nueva ventana.						
Postcondiciones	Aparecen las máscaras creadas.						
Excepciones	No hay defectos en la imagen.						
Importancia	Media						
Urgencia	Media						

Tabla B.3: Caso de uso 2: Visualizar máscaras.

Caso de uso 3: Visualizar información de los defectos.							
Descripción	Permite al usuario visualizar el <i>Bounding Box</i> , la clase y la marca de cada defecto detectado.						
Requisitos	RF-4 RF-4.1						
Precondiciones	Tener una imagen cargada y que ya se haya hecho la detección de defectos.						
Secuencia normal	<table border="1"> <thead> <tr> <th>Paso</th><th>Acción</th></tr> </thead> <tbody> <tr> <td>1</td><td>El usuario pincha el botón “Información”.</td></tr> <tr> <td>2</td><td>Aparece la información en una nueva ventana.</td></tr> </tbody> </table>	Paso	Acción	1	El usuario pincha el botón “Información”.	2	Aparece la información en una nueva ventana.
Paso	Acción						
1	El usuario pincha el botón “Información”.						
2	Aparece la información en una nueva ventana.						
Postcondiciones	Aparece la información de los defectos que se hayan detectado.						
Excepciones	No hay defectos en la imagen.						
Importancia	Media						
Urgencia	Media						

Tabla B.4: Caso de uso 3: Visualizar información de los defectos.

Apéndice C

Especificación de diseño

C.1. Introducción

En esta sección se describe cómo están implementados los datos de esta aplicación, cuáles son los procedimientos más relevantes y cómo se organizan los proyectos.

C.2. Diseño de datos

En esta sección, explicaremos como están organizados tanto el conjunto de imágenes obtenidas, cómo son y como se organizan las características extraídas de las imágenes, y el diagrama de clases de la aplicación.

Conjunto de imágenes

El conjunto de imágenes es el descrito en [4]. Estas imágenes están disponibles en un repositorio público: https://drive.google.com/drive/folders/143893UA1c7TB_ZiTGl9S9w4rp3wH1ji. Este repositorio contiene 5 grupos de carpetas una para cada grupo: *Castings*, *Welds*, *Baggage*, *Nature* y *Settings*. Estas imágenes son almacenadas en formato de escala de grises de 8 bits “png” (*Portable Network Graphics*). También tenemos metadatos adicionales para cada serie en un archivo ASCII llamado Xssss_readme.txt incluido en la subcarpeta Xssss, donde “X” es la inicial del grupo y “ssss” es el número de serie del grupo de imágenes. Para este proyecto sólo hemos utilizado el grupo *Castings*.

En [4] este grupo se describe como: “(...)contiene 2.727 imágenes de rayos-x organizadas en 67 series. Las imágenes de rayos-x se tomaron principalmente de partes automotrices (ruedas de aluminio y nudillos) usando un intensificador de imágenes. (...)Es interesante resaltar que la serie C0001 contiene no solo una secuencia de 72 imágenes de rayos-x tomadas de una rueda de aluminio girando su eje central, pero también anotaciones de *Bounding Boxes* de la verdad del terreno (*ground truth*) de 226 pequeños defectos y la matriz de calibración de cada imagen que relaciona las coordenadas 3D de la rueda de aluminio con coordenadas 2D de la imagen de rayos-x” (p.4-p.5).

Diagrama de clases de la aplicación

En este apartado se comentará las clases realizadas por nosotros en el proyecto. Es decir, las clases que se han creado para realizar la aplicación. Señalamos que, únicamente se analizaran las realizadas por nosotros, toda la estructura del repositorio `metal-defect-detection` [1] no será analizada.

Clases de la aplicación

Para la aplicación se han creado las siguientes clases:

- `XrayDataset` dentro de `app.py`: Clase que contiene el conjunto de datos de las imágenes de rayos-X, tiene una sola función que crea un conjunto de datos compuesto por una imagen con su id, su directorio, su altura y su anchura también crean una clase “*Casting Defect*” que es el tipo de defectos que puede haber.
- `Detector` dentro de `app.py`: Clase que contiene toda la estructura gráfica de la ventada de la aplicación. Se inicializan todos los elementos gráficos (*frames*, botones...), las variables necesarias para cada uno, y los mediadores. También contiene todos los métodos necesarios para la visualización de las imágenes, la detección de los defectos y la visualización de las máscaras y la información adicional de los defectos. Debido a que todo se encuentra en la misma clase la mayoría de funciones no devuelve nada.
- `InferenceConfig` dentro de `app.py`: Clase que contiene únicamente dos variables para la configuración del `config`. Esta configuración hace que se ejecute la detección de defectos en una solo imagen cada vez.

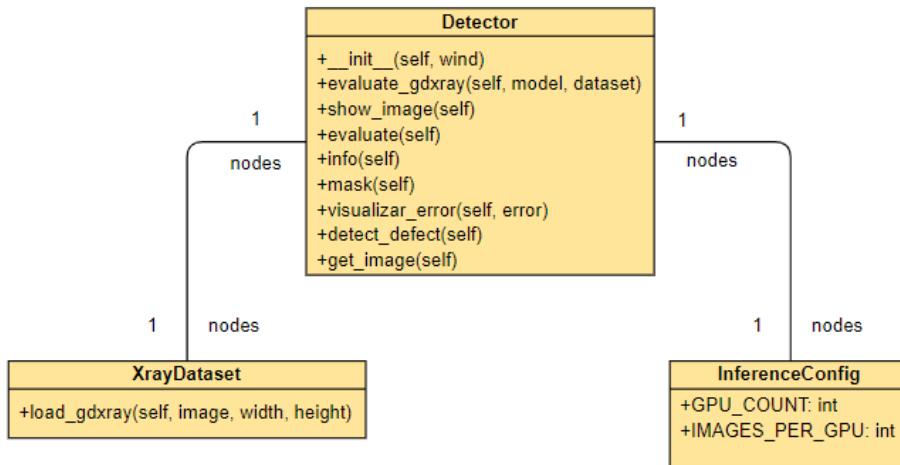


Figura C.1: Diagrama de clases

C.3. Diseño procedimental

Diagramas de secuencia

En este apartado se mostrarán los diagramas de secuencia respectivos a 4 tareas de la aplicación:

- Cargar de imágenes, ver figura C.2.
- Predecir los defectos de las imágenes y mostrar la imagen con los defectos, ver figura C.3.
- Visualizar la información de los defectos, ver figura C.4.
- Visualizar las máscaras de los defectos, ver figura C.5.

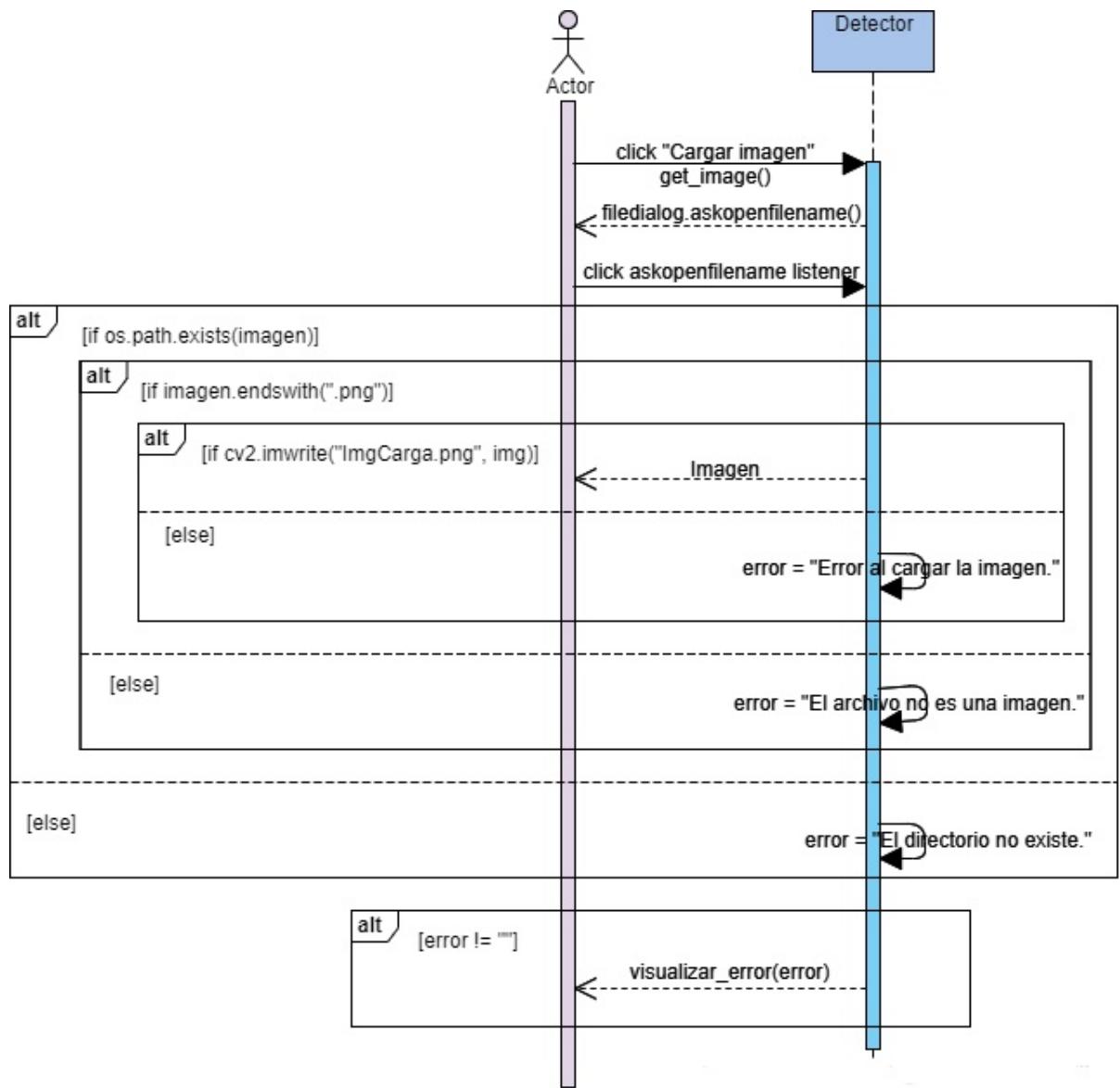


Figura C.2: Diagrama de secuencia para la cargar una imagen

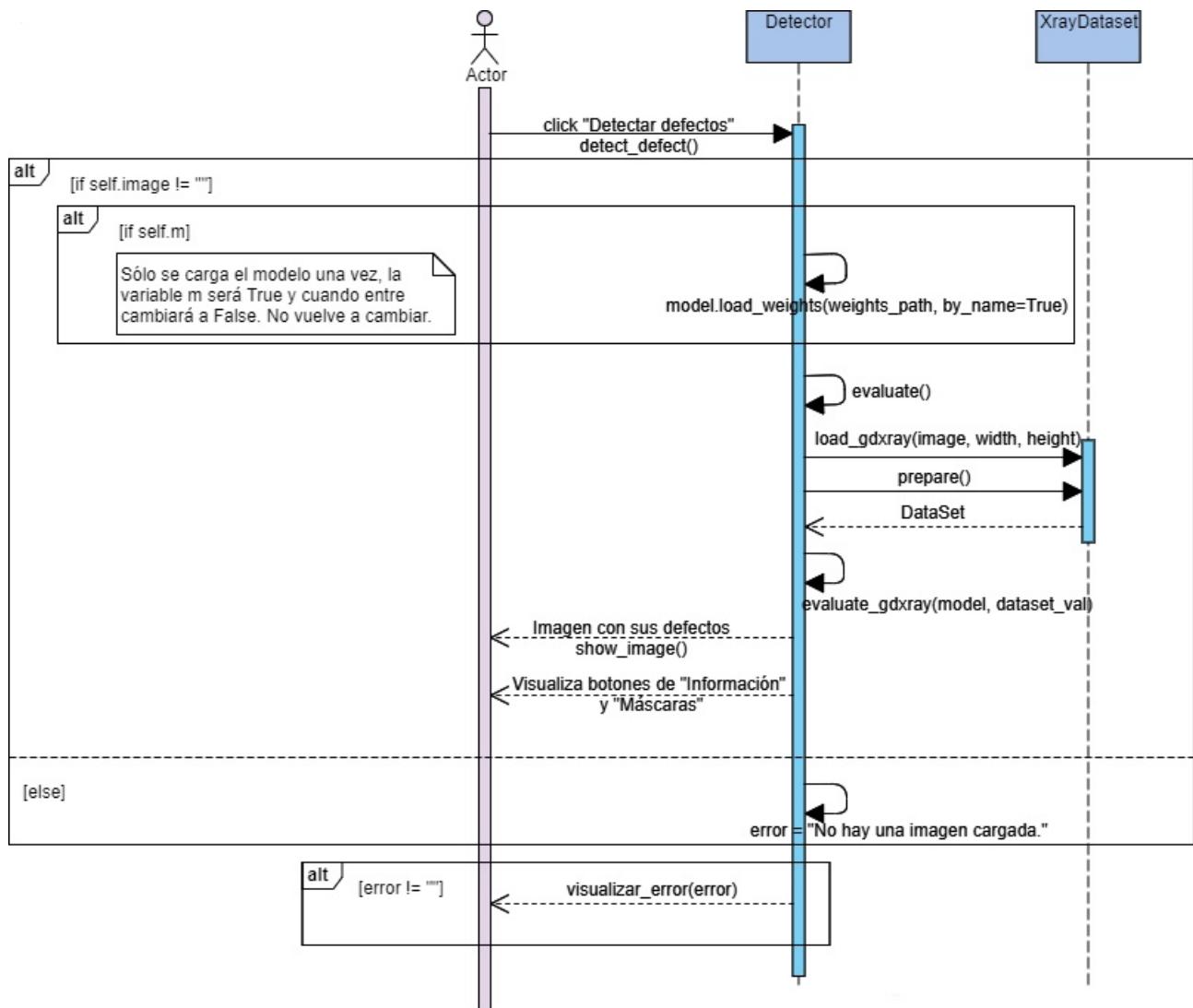


Figura C.3: Diagrama de secuencia para la predicción y muestra de defectos

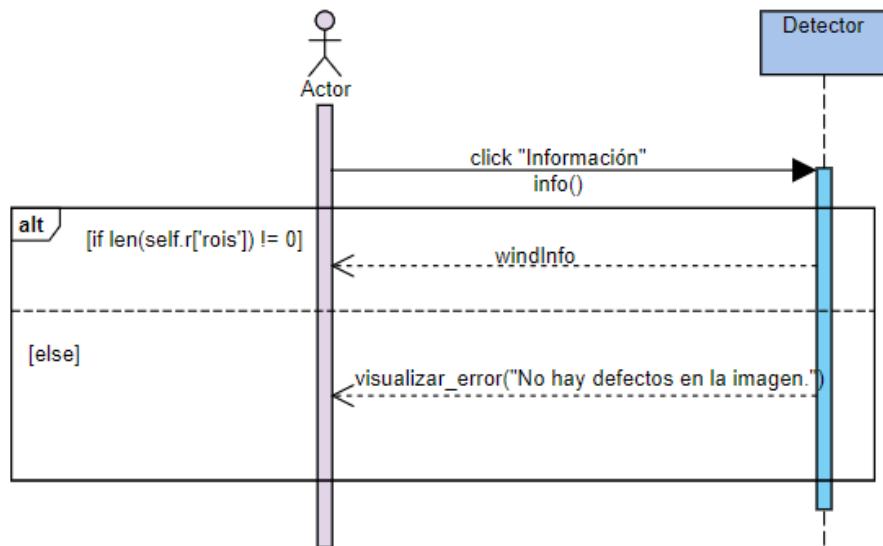


Figura C.4: Diagrama de secuencia para la muestra de la información de los defectos

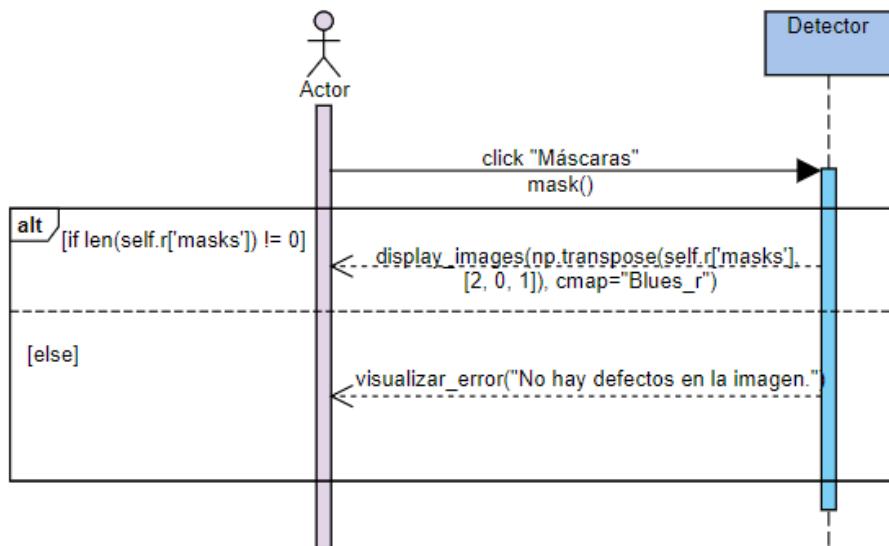


Figura C.5: Diagrama de secuencia para la muestra de las máscaras de los defectos

C.4. Diseño arquitectónico

En este apartado se definirá el diseño arquitectónico de la aplicación, la cual sigue el patrón de diseño *Singleton*.

Patrón Singleton

El patrón de diseño *Singleton* [3] (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.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En esta sección se describe la documentación técnica de programación incluyendo la instalación del entorno de desarrollo, la estructura de la aplicación, su compilación y las baterías de pruebas realizadas.

D.2. Estructura de directorios

Para ejecutar este programa, todos los archivos tienen que estar en el mismo directorio, sin importar exactamente la localización en los archivos.

Los modelos que genera se van a guardar en “directorio de los archivos/logs/gdxray” creando una carpeta por cada vez que se ejecuta el entrenamiento, el nombre de esta carpeta se compone por: gdxray + “la fecha de creación” + T + “hora de creación” (ejemplo, el archivo “gdx-ray20200508T1627” se creó el día 8 de mayo del 2020 a las 16:27 horas).

Las imágenes deben estar guardadas en el directorio “C:/Users/nombre usuario/data/GDXray/Castings/Cssss” donde “C” proviene de “*Casting*” y “ssss” es el número de serie del grupo de imágenes. Las imágenes de una serie se almacenan en el archivo Cssss_nnnn.png donde “nnnn” corresponde al número de la imagen de rayos-x de esta serie. En total hay 67 carpetas. Dentro de algunas de estas carpetas hay una carpeta llamada “mask” con las máscaras de todas las imágenes que se encuentren en la carpeta y un archivo “ground_truth.txt” con las posiciones de los errores.

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

Estas direcciones son opcionales, es decir, al entrenar tú puedes asignar la dirección donde guarde los modelos y la dirección de donde coja las imágenes, pero son las recomendadas.

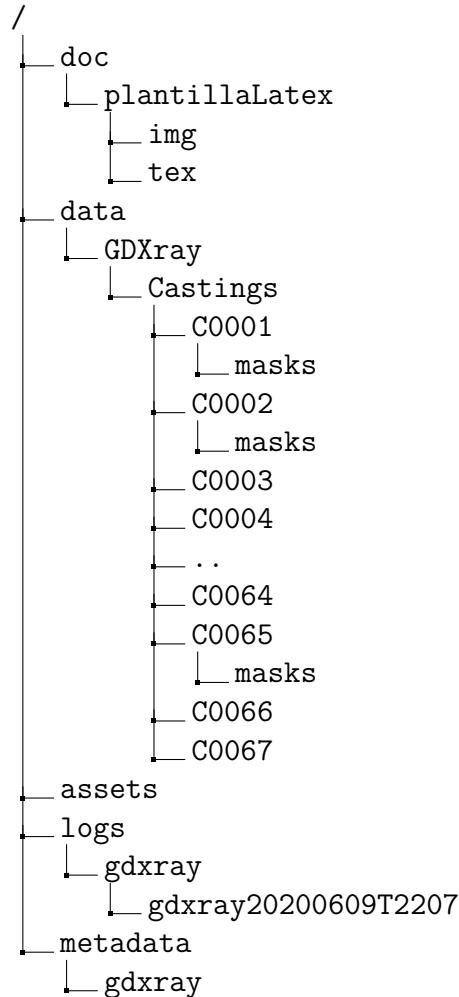


Figura D.1: Directorios del proyecto

D.3. Manual del programador

El siguiente apartado tiene como objetivo servir de referencia a futuros programadores que trabajen en la aplicación.

Entorno del proyecto

Empezamos explicando el entorno en el que está realizado el proyecto.

- **Ordenador portátil:** Aspire ES1-521. Memoria 16GB RAM. Procesador AMD A6-6310 APU con AMD Radeon R4 Graphics, 1800 Mhz. 64 bit.
 - **Sistema operativo:** Microsoft Windows. Versión 10.0.18362.752.
- **Ordenador Alpha:** Procesador Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz. 64 bit.
 - **Sistema operativo:** GNU/Linux. Versión 4.15.0-88-generic. Ubuntu 7.4.0-1ubuntu1 18.04.1.
- **Python:** Versión 3.6.5 Anaconda Custom.
- **Anaconda:** Versión 4.8.2.
- **Spyder:** Versión 3.3.6.
- Los demás requerimientos del proyecto se comentarán en la sección [D.4](#).

Python

El proyecto de que he partido está programado en *Python* por lo que yo he continuado usando este lenguaje. También *Python* es el lenguaje más popular a la hora de programar inteligencia artificial.

Spyder

Para trabajar con *Python* he utilizado el entorno *Spyder* que es un entorno de desarrollo integrado multiplataforma de código abierto (IDE).

Para realizar la aplicación he utilizado la librería *Tkinter* que es una interfaz gráfica de usuario (GUI) para *Python* muy popular.

26PÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

Archivos y notebooks

Comentaremos que se realiza en cada archivo *.py* y *notebook* del proyecto.

- **app.py**: En este archivo tenemos todo el código de la aplicación. Se crea la interfaz de la aplicación, se crea el modelo con que él se van a detectar los defectos y carga la imagen. Para ello se instalan todas las librerías necesarias como *Tkinter* o *cv2* y también se importan las clases y funciones de los archivos *utils*, *visualize*, *model*, *gdxray*.
- **check.py**: En este archivo se cargan las imágenes de entrenamiento y se muestran con los *bounding box* dado por el conjunto de datos, en otras palabras, se representan los defectos reales de las piezas.
- **coco.py**: En este archivo se carga un subconjunto del conjunto de datos COCO, explicado en la memoria en el apartado 3.3. Puedes auto descargar el conjunto de datos, entrenarlo y evaluarlo.
- **config.py**: En este archivo se declaran las variables globales que se van a utilizar en todo el proyecto.
- **environment.yml**: En este archivo es el utilizado para crear el entorno virtual de trabajo, dentro nos encontramos con el nombre que va a recibir este entorno y todas las dependencias que se van a instalar en él.
- **gdxray.py**: En este archivo se cargan las imágenes de entrenamiento o testeо, depende de la opción con la que lo ejecute. Prepara todo el subconjunto de datos para llamar al modelo (en el archivo *model.py* para entrenar la red).
- **inspect_data.ipynb**: En este *notebook* se inspecciona y visualiza el código de carga y preprocesamiento de datos.
- **inspect_model.ipynb**: Este *notebook* incluye código y visualizaciones para probar, depurar y evaluar el modelo.
- **inspect_weights.ipynb**: Este *notebook* incluye código y visualizaciones para probar, depurar y evaluar los pesos del modelo.
- **mask_rcnn_coco.h5**: Este archivo es el modelo de Mask R-CNN en Python 3 que hemos utilizado para el primer entrenamiento.

- **model.py**: En este archivo se crea el modelo con los pesos y se entrena. Este es llamado siempre por el archivo **gdxray.py** ya que es el que carga las imágenes que el modelo necesita para entrenar.
- **preprocessing.py**: En este archivo se preprocesan los datos, es decir, cambian algunos datos para que el trato con ellos sea más cómodo. Por ejemplo, la función **prepare_welding()** se recorta las imágenes y las máscaras a 768 píxeles de ancho, se renombran y se genera un archivo de metadatos con los nuevos.
- **utils.py**: En este archivo tenemos las funciones y clases de utilza comunes en el proyecto. Tenemos la clase **Dataset** que es la clase base para el conjunto de datos, en ella se crean añaden las imágenes, se crean las clases del conjunto de datos con las que se trabaja (“*Casting*”, “*Welding*” y “*Background*”) y se prepara el conjunto de datos para su uso (guarda en las variables globales los datos que se necesitan), entre otras funciones. También tenemos las funciones de **resize_mask()** y **resize_image()** que redimensionan las máscaras y las imágenes para que todas sean del mismo tamaño.
- **visualize.py**: En este archivo tenemos las funciones de visualización de las imágenes y las estadísticas de los pesos y las máscaras de una imagen.

D.4. Compilación, instalación y ejecución del proyecto

A la hora de instalar este proyecto hay que tener instalado *Python3*. Tras haber descargado los archivos del proyecto, las imágenes y la implementación de *Mask R-CNN* en *Python 3*, *Keras* y *TensorFlow* se debe crear un entorno de *Python* para instalar las librerías necesarias en él con el comando “**conda env create -f environment.yml -n defect-detection**”, en este archivo aparecen todas las dependencias del entorno.

Este comando, aunque me creaba el entorno correctamente había algunas librerías que no se me instalaban por lo que a continuación debía entrar en el entorno con “**conda activate defect-detection**” e instalarlas manualmente. Las dos librerías que no se instalaban eran, *OpenCV* y *TensorFlow*.

Con “**pip install opencv-python**” instalé *OpenCV* y para la librería de *TensorFlow* primero compruebo si hay ya instalada alguna versión, en una

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

ocasión se descargaron varios paquetes de la librería, pero no todos, para borrarla e instalarla una que funcione con “`pip install tensorflow==1.3.0`”, esta librería debía tener una versión superior a la 1.3.

Comandos seguidos

Se debe de tener en cuenta que las palabras comprendidas entre las comillas pueden dar error al ejecutar, si es así, cambie las comillas del texto copiado y ponga la comilla simple que se encuentra en el teclado junto a signo de interrogación del final de las oraciones.

Aquí tendremos los comandos que he seguido para empezar con el proyecto y los comando para la ejecución del final de proyecto.

Descarga

Primero se clona el repositorio con los archivos de proyecto.

- `git clone https://github.com/nuf1001/XRayDetector`

También se pueden descargar del proyecto original si se tiene algún problema.

- `git clone https://github.com/maxkferg/metal-defect-detection`

Si lo que quiere es entrenar al descargar el repositorio anterior debe descargar también el modelo `mask_rcnn_coco.h5` ya que no se encuentra en él.

- `wget https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5`

A continuación, descargamos las imágenes con las que trabajaremos. Como es un archivo muy grande te pide una doble confirmación para descargarlo ya que no se puede examinar para buscar virus. Para solucionar esto se ejecutan dos comandos, el primero para pedir que descargar el archivo y guardar una autorización en el archivo `cookies.txt` y el segundo para confirmar con este archivo que quiere descargarlo. Por último, es sólo descomprimir el archivo que se ha descargado.

- wget -save-cookies cookies.txt -keep-session-cookies -no-check-certificate 'https://docs.google.com/uc?export=download&id=143893UAlc7TB_ZiTGl9S9w4rp3wh1ji' -O- | sed -rn 's/.*/confirm=([0-9A-Za-z_]+).*/Code:\n\1\n/p'
- wget -load-cookies cookies.txt 'https://docs.google.com/uc?export=download&confirm=S3q0&id=143893UAlc7TB_ZiTGl9S9w4rp3wh1ji' -O Cast
- unzip Cast

Instalación

Primero creamos el entorno virtual en el que vamos a trabajar y lo activamos.

- conda env create -f environment.yml -n defect-detection
- conda activate defect-detection

Como ya se ha explicado el entorno no se crea perfectamente, le faltan algunas librerías de instalar, tras la activación del entorno las instalamos.

- pip install opencv-python

En ocasiones **TensorFlow** ya está instalado, pero con una versión que no vale. Por lo tanto, primero debemos comprobar si está instalado o no y en el primer caso desinstalarlo para instalar una versión que sí que valga.

- python3 -c 'import tensorflow as tf; print(tf.__version__)'
- pip uninstall tensorflow
- pip install tensorflow==1.3.0

Entrenamiento

En este comando hay que comprobar que los archivos que está utilizando se encuentran en las carpetas a las que nos estamos dirigiendo, si alguno falla el entrenamiento no se llevará a cabo.

- python gdxray.py train -dataset= /data/GDXray -series=Castings -model=mask_rcnn_coco.h5 -logs=logs/gdxray -download=True

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

Evaluación

En este comando pasa lo mismo que en el anterior. Lo que más hay que tener en cuenta es la carpeta donde se va a encontrar el modelo que vamos a evaluar.

- `python gdxray.py evaluate -dataset= /data/GDXray -series=Castings -logs=logs/gdxray -model= /path/to/trained/model -limit=10`

Ejecutar la aplicación

Para ejecutar la aplicación debes estar en el entorno que hemos creado en los pasos anteriores.

- `python app.py`

D.5. Pruebas del sistema

Datos de entrenamiento

En este apartado se realizarán varias pruebas al conjunto de datos que tenemos.

Imágenes y máscaras

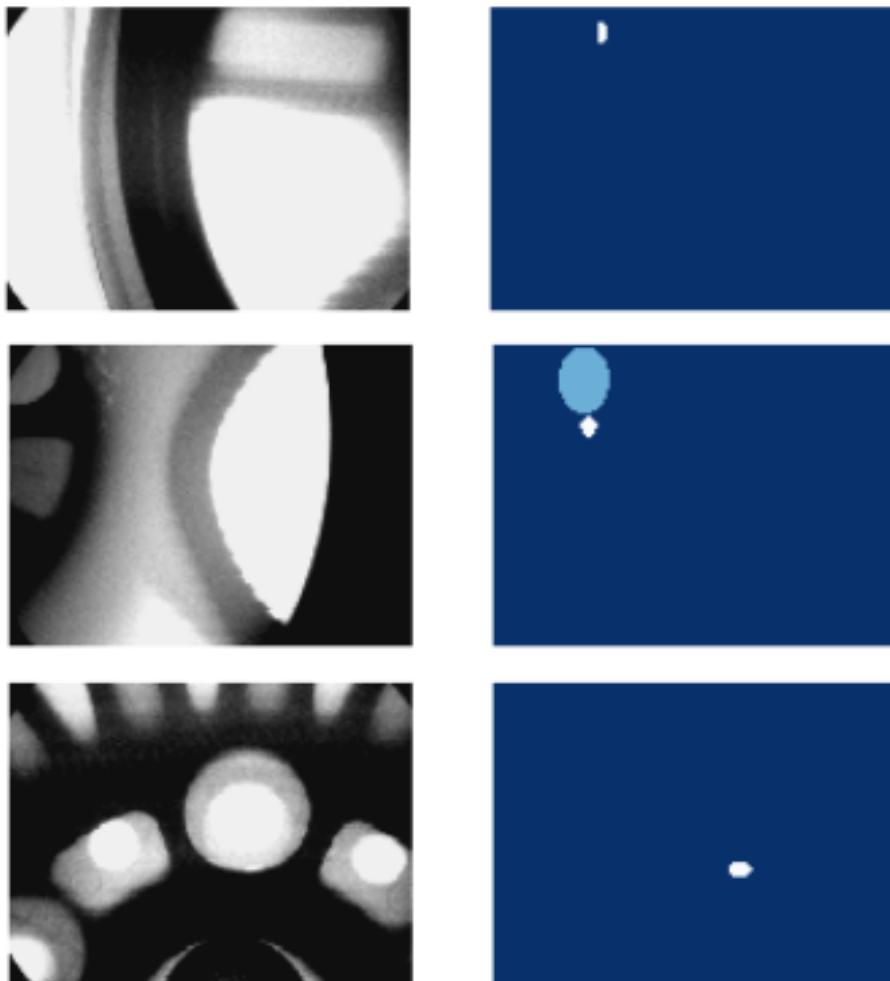


Figura D.2: Imágenes y sus máscaras con los defectos

En la figura D.2 tenemos tres ejemplos de imágenes con sus máscaras correspondientes. Las imágenes de la derecha son las máscaras, las manchas blancas o azul claro son donde se encuentran los defectos en la imagen original.

Bounding Boxes

En este apartado calculamos los cuadros delimitadores a partir de máscaras en lugar de utilizar las coordenadas del cuadro delimitador proporcionadas por los conjuntos de datos de origen. Esto nos permite manejar los

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

cuadros delimitadores de forma coherente, independientemente del conjunto de datos de origen, y también hace que sea más fácil cambiar el tamaño, rotar o recortar imágenes porque simplemente generamos los cuadros delimitadores a partir de las máscaras de actualizaciones.

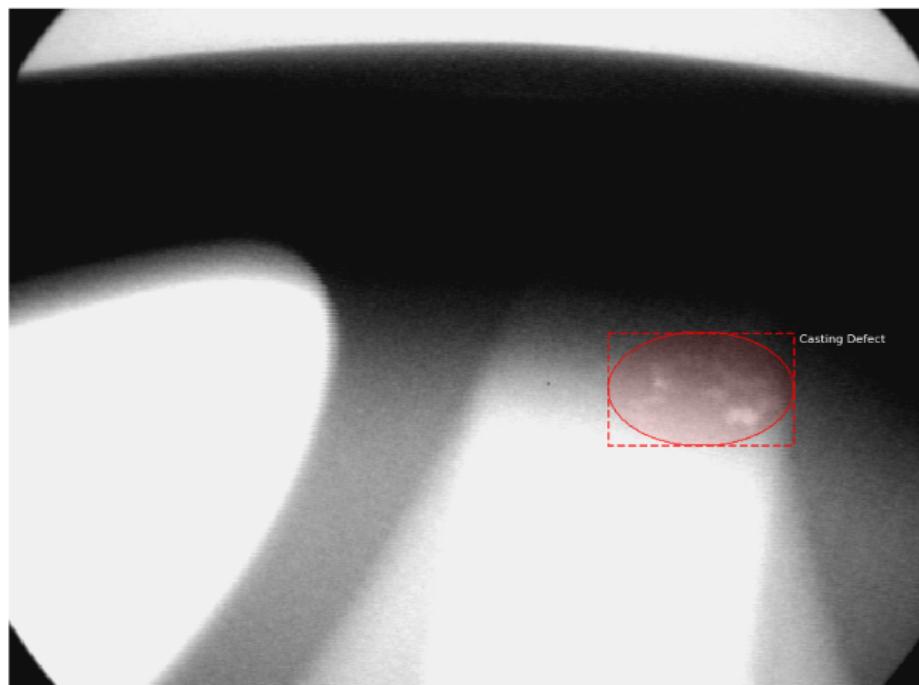


Figura D.3: Imagen con el cuadro delimitador calculado a partir de la máscara

```
# Load random image and mask.
image_id = random.choice(dataset.image_ids)
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)
# Compute Bounding box
bbox = utils.extract_bboxes(mask)

# Display image and additional stats
print("image_id ", image_id, dataset.image_reference(image_id))
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)
# Display image and instances
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

Figura D.4: El código para la visualización de la figura D.3

Cambiar el tamaño de las imágenes

Las imágenes se redimensionan a un tamaño (1024x1024), teniendo en cuenta la relación de aspecto, esta se conserva. Si una imagen no es cuadrada, se agrega relleno cero en la parte superior e inferior o derecha e izquierda, en otras palabras, se agregan márgenes negros a las imágenes donde sea necesario para que sean cuadradas.

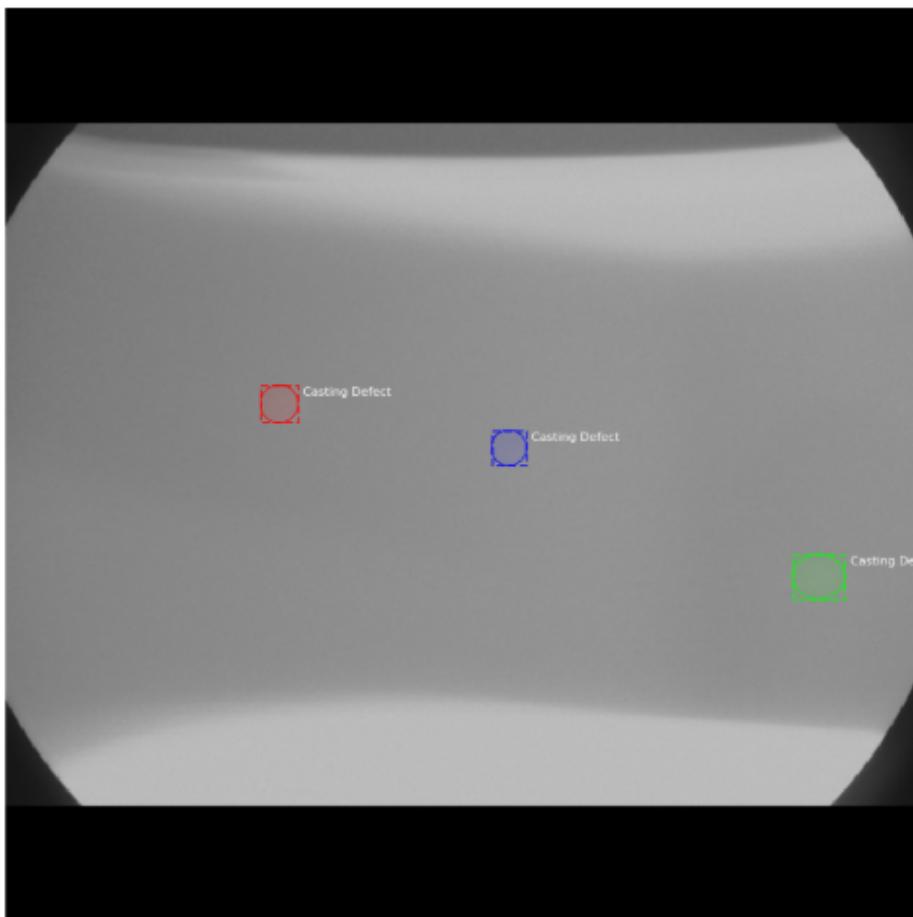


Figura D.5: Imagen redimensionada a (1024x1024)

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

```
# Load random image and mask.
image_id = np.random.choice(dataset.image_ids, 1)[0]
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)
original_shape = image.shape
# Resize
image, window, scale, padding = utils.resize_image(
    image,
    min_dim=config.IMAGE_MIN_DIM,
    max_dim=config.IMAGE_MAX_DIM,
    padding=config.IMAGE_PADDING)
mask = utils.resize_mask(mask, scale, padding)
# Compute Bounding box
bbox = utils.extract_bboxes(mask)

# Display image and additional stats
print("image_id: ", image_id, dataset.image_reference(image_id))
print("Original shape: ", original_shape)
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)
# Display image and instances
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

Figura D.6: El código para la visualización de la figura D.5

Mini Máscaras

Las máscaras binarias de instancias pueden crecer cuando se entrena con imágenes de alta resolución. Por ejemplo, si se entrena con una imagen de 1024x1024, la máscara de una sola instancia requiere 1 MB de memoria (*NumPy* usa bytes para valores booleanos). Si una imagen tiene 100 instancias, eso es 100 MB solo para las máscaras.

Para mejorar la velocidad del entrenamiento, optimizamos las máscaras mediante:

- Almacenamos píxeles de máscara que están dentro del cuadro delimitador de objetos, en lugar de una máscara de la imagen completa. La mayoría de los objetos son pequeños en comparación con el tamaño de la imagen, por lo que ahorraremos espacio al no almacenar muchos ceros alrededor del objeto.
- Cambiamos el tamaño de la máscara a un tamaño más pequeño (por ejemplo, 56x56). Para los objetos que son más grandes que el tamaño seleccionado, perdemos un poco de precisión. Pero la mayoría de las anotaciones de objetos no son muy precisas, por lo que esta pérdida es insignificante para la mayoría de los propósitos prácticos. El tamaño de `mini_mask` se puede establecer en la clase de configuración.

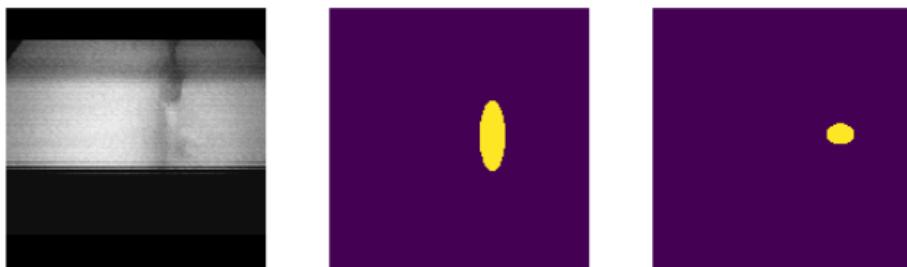


Figura D.7: Imagen de prueba con máscaras correspondientes

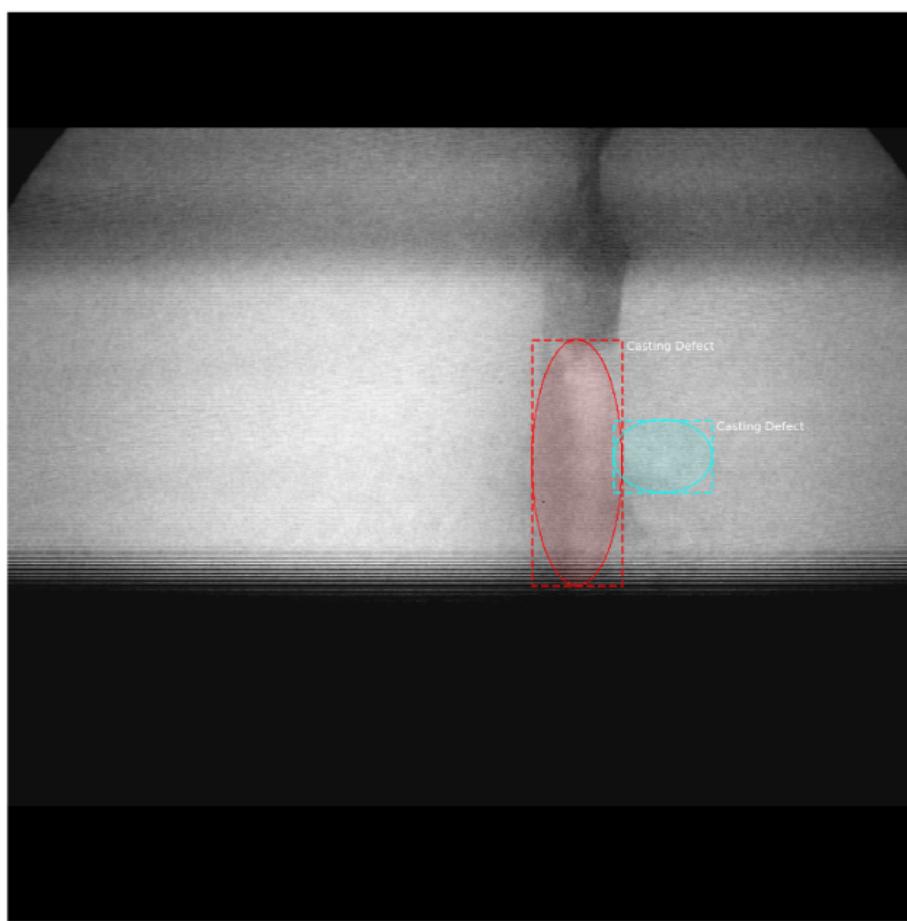


Figura D.8: Imagen de prueba con los *bounding box* de esas máscaras

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

```
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(
    dataset, config, image_id, use_mini_mask=False)

log("image", image)
log("image_meta", image_meta)
log("class_ids", class_ids)
log("bbox", bbox)
log("mask", mask)

display_images([image]+[mask[:, :, i] for i in range(min(mask.shape[-1], 7))])
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

Figura D.9: El código para la visualización de las figuras D.7 y D.8



Figura D.10: Imagen de prueba con sus mini máscaras correspondientes

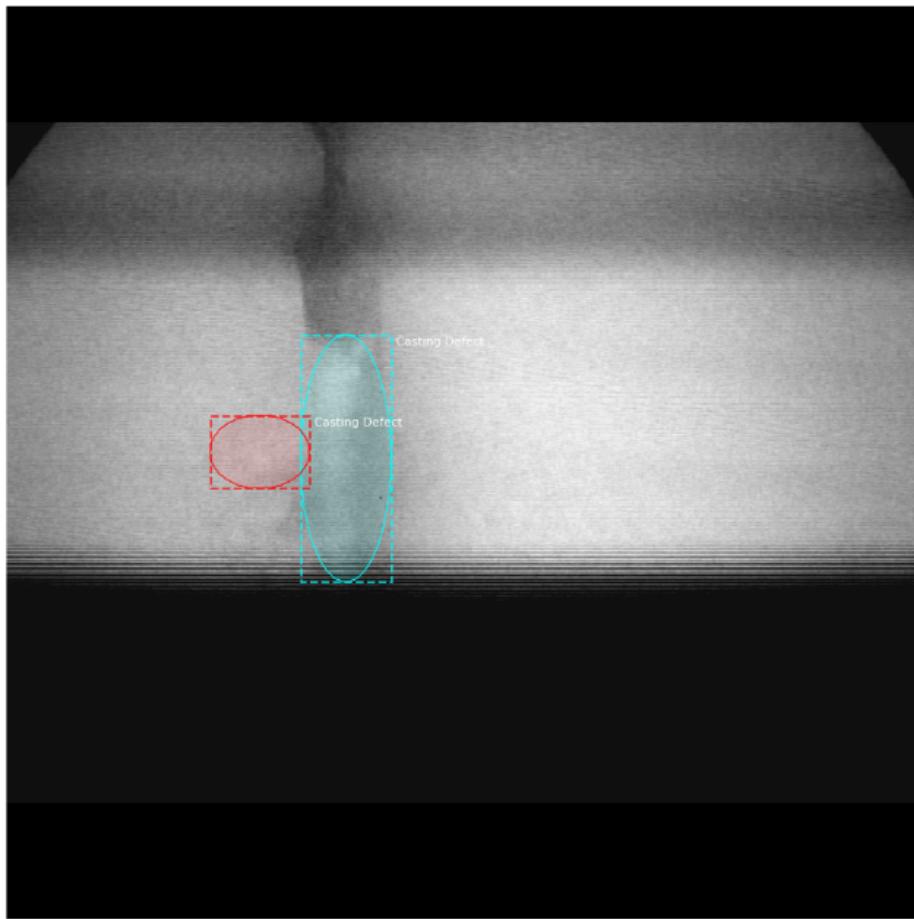


Figura D.11: Imagen de prueba con los *bounding box* de esas mini máscaras

```
# Add augmentation and mask resizing.  
image, image_meta, class_ids, bbox, mask = modellib.load_image_gt(  
    dataset, config, image_id, augment=True, use_mini_mask=True)  
log("mask", mask)  
display_images([image]+[mask[:, :, i] for i in range(min(mask.shape[-1], 7))])  
  
mask = utils.expand_mask(bbox, mask, image.shape)  
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

Figura D.12: El código para la visualización de las figuras D.10 y D.11

APÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

Anchors

Es importante usar el mismo orden de las anclas en las fases de entrenamiento y predicción. Y debe coincidir con el orden de ejecución de la convolución.

Para una red FPN como la nuestra, los anclajes deben ordenarse de manera que sea fácil hacer coincidir los anclajes con la salida de las capas de convolución que predicen las puntuaciones y los cambios de los anclajes.

Primero los ordenaremos por nivel de pirámide, es decir, todos los anclajes del primer nivel, luego todos los del segundo y así sucesivamente. Dentro de cada nivel, las anclas están clasificadas por secuencia de procesamiento del mapa de características. Normalmente, una capa de convolución procesa un mapa de características que comienza desde arriba a la izquierda y se mueve a la derecha fila por fila.

Paso de anclaje: en la arquitectura FPN, los mapas de características en las primeras capas son de alta resolución. Por ejemplo, si la imagen de entrada es 1024x1024, el conjunto de características de la primera capa es 256x256, lo que genera unos 200K anclajes. Estos anclajes son de 32x32 píxeles y su zancada con respecto a los píxeles de la imagen es de 4 píxeles, por lo que hay mucha superposición.

Podemos reducir la carga significativamente si generamos anclajes para cada otra celda en el mapa de características, por ejemplo, con una zancada de 2 el número de anclas se reducirá en 4.

Vemos la configuración de las anclas de nuestro proyecto en la tabla D.1.

Número de anclas	147312
Escalas	(16, 32, 64, 128, 256)
Proporción	[0.5, 1, 2]
Anclas por celda	3
Niveles	5
Anclas en el nivel 0	110592
Anclas en el nivel 1	27648
Anclas en el nivel 2	6912
Anclas en el nivel 3	1728
Anclas en el nivel 4	432

Tabla D.1: Configuración de las anclas

Visualizaremos los anclajes de una celda en el centro del mapa de características de un nivel específico. Las formas del mapa de características de los niveles serán las reflejadas en la tabla D.2.

Nivel 0	[192 192]
Nivel 1	[96 96]
Nivel 2	[48 48]
Nivel 3	[24 24]
Nivel 4	[12 12]

Tabla D.2: Forma del mapa de características

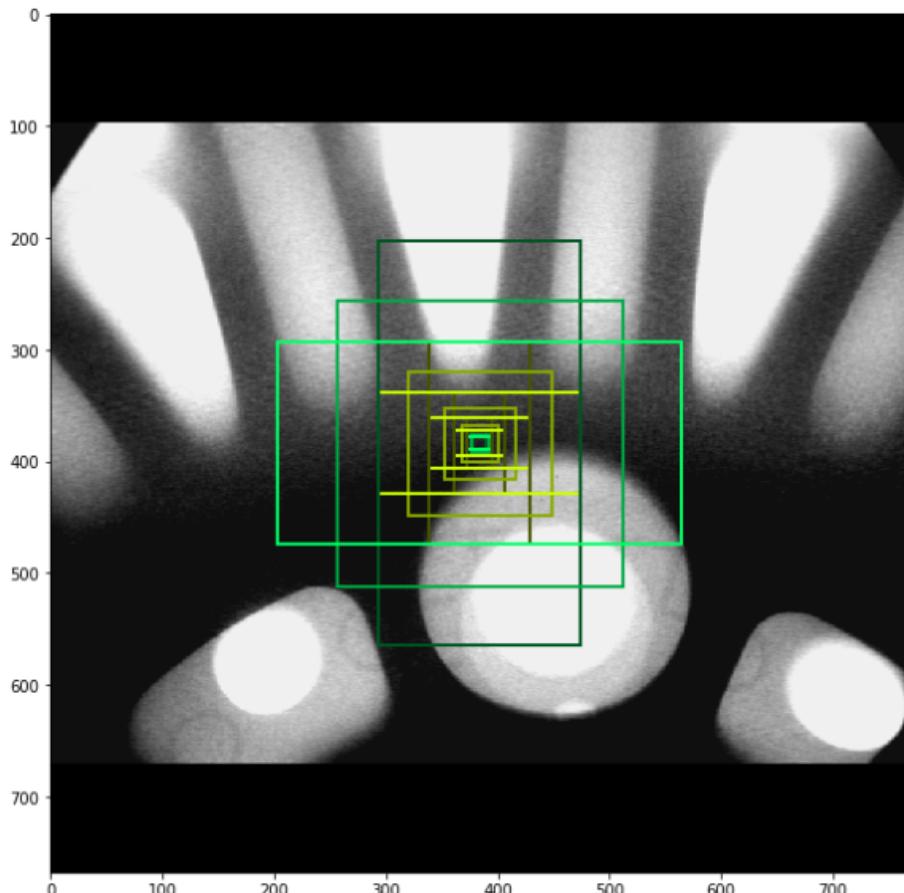


Figura D.13: Imagen con sus anclajes

AOPÉNDICE D. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

```
# Load and draw random image
image_id = np.random.choice(dataset.image_ids, 1)[0]
image, image_meta, _, _ = modellib.load_image_gt(dataset, config, image_id)
fig, ax = plt.subplots(1, figsize=(10, 10))
ax.imshow(image)
levels = len(config.BACKBONE_SHAPES)
for level in range(levels):
    colors = visualize.random_colors(levels)
    # Compute the index of the anchors at the center of the image
    level_start = sum(anchors_per_level[:level]) # sum of anchors of previous levels
    level_anchors = anchors[level_start:level_start+anchors_per_level[level]]
    center_cell = config.BACKBONE_SHAPES[level] // 2
    center_cell_index = (center_cell[0] * config.BACKBONE_SHAPES[level][1] + center_cell[1])
    level_center = center_cell_index * anchors_per_cell
    center_anchor = anchors_per_cell * (
        (center_cell[0] * config.BACKBONE_SHAPES[level][1] / config.RPN_ANCHOR_STRIDE**2) \
        + center_cell[1] / config.RPN_ANCHOR_STRIDE)
    level_center = int(center_anchor)

    # Draw anchors. Brightness show the order in the array, dark to bright.
    for i, rect in enumerate(level_anchors[level_center:level_center+anchors_per_cell]):
        y1, x1, y2, x2 = rect
        p = patches.Rectangle((x1, y1), x2-x1, y2-y1, linewidth=2, facecolor='none',
                              edgecolor=(i+1)*np.array(colors[level]) / anchors_per_cell)
        ax.add_patch(p)
```

Figura D.14: El código para la visualización de la figura D.13

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección explicaremos lo necesario para que los usuarios puedan instalar y utilizar la aplicación.

E.2. Requisitos de usuarios

El usuario, para poder utilizar la aplicación, deberá tener instalado *Python*, al menos la versión 3.6.8 y cargar el entorno virtual que se ha creado con la ayuda del archivo `environment.yml`. Idealmente, el sistema se instalará en Windows 10, ya que la aplicación está optimizada para este sistema operativo.

E.3. Instalación

Los pasos para la instalación del proyecto se detallan en [Instalación](#). Les resumimos:

- Primero compruebe que ha descargado todos los archivos como indica el apartado .
- Cree el entorno virtual y actívelo.
- Compruebe que las bibliotecas de OpenCV y TensorFlow se han instalado correctamente, si no es así instálelas en el entorno.

Para ejecutar la aplicación:

- `python app.py`

E.4. Manual del usuario

En esta sección se describe como realizar las diferentes operaciones de la aplicación.

Cargar imagen

Para cargar una imagen:

1. Hacer click en el botón “Cargar imagen”.
2. Seleccionar una imagen del dispositivo.
3. La imagen aparecerá en la aplicación.

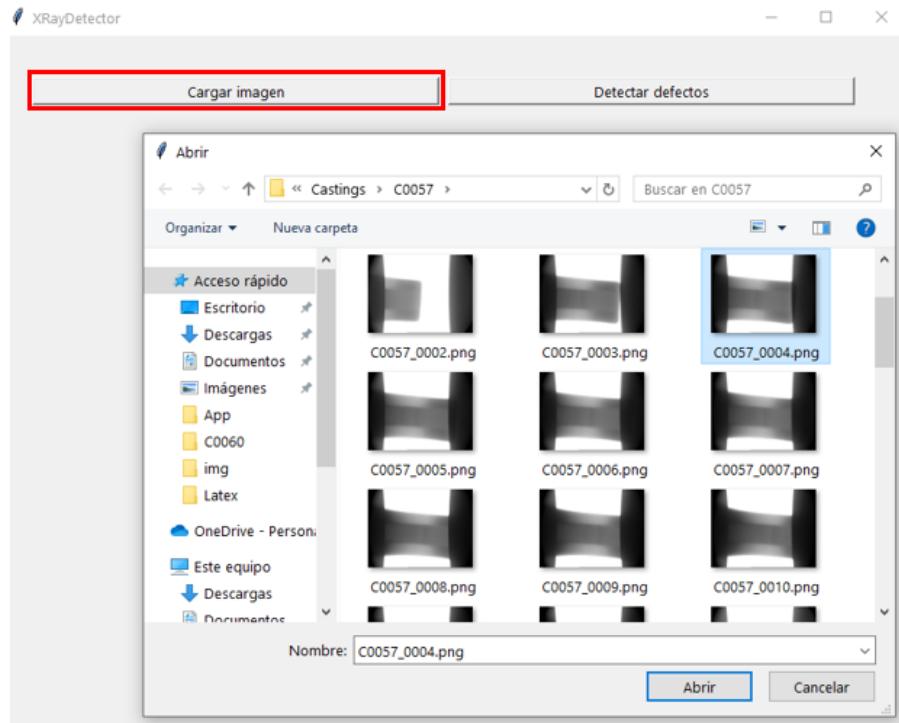


Figura E.1: Carga de una imagen en la aplicación

Detectar defectos

Para detectar los defectos de la imagen cargada:

1. Hacer click en el botón “Detectar defectos”.
2. Espere, el proceso puede tardar unos segundos.
3. La imagen con los defectos aparecerá en la aplicación.

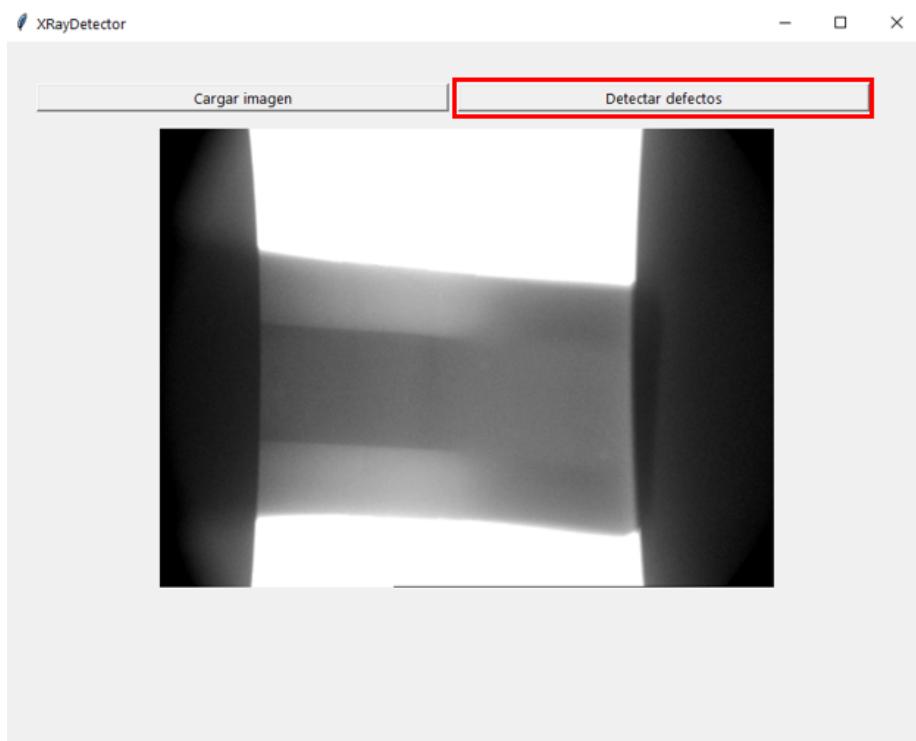


Figura E.2: Detectar defectos de una imagen

Visualizar máscaras

Para visualizar las máscaras de la imagen:

1. Hacer click en el botón “Máscaras”.
2. Se abre una segunda ventana con la/s imágenes de las máscaras.

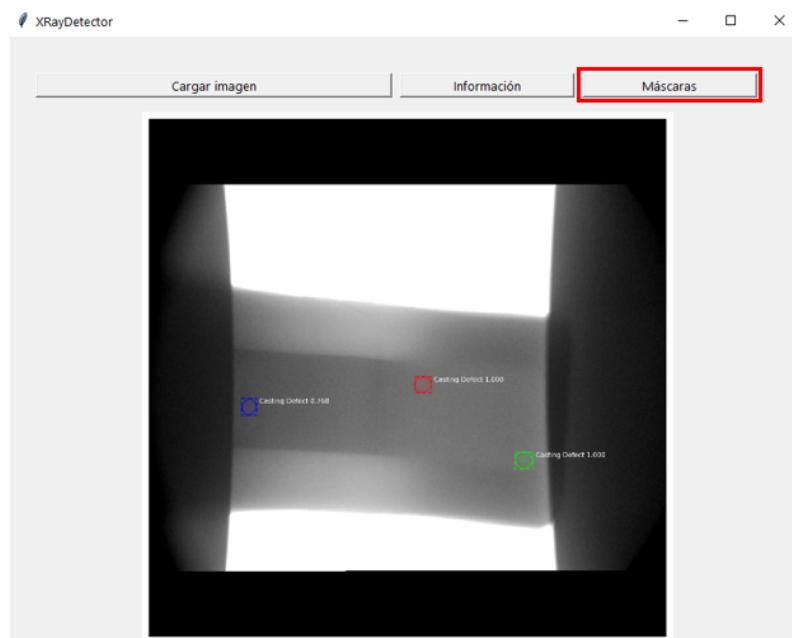


Figura E.3: Visualizar las máscaras de los defectos de la imagen

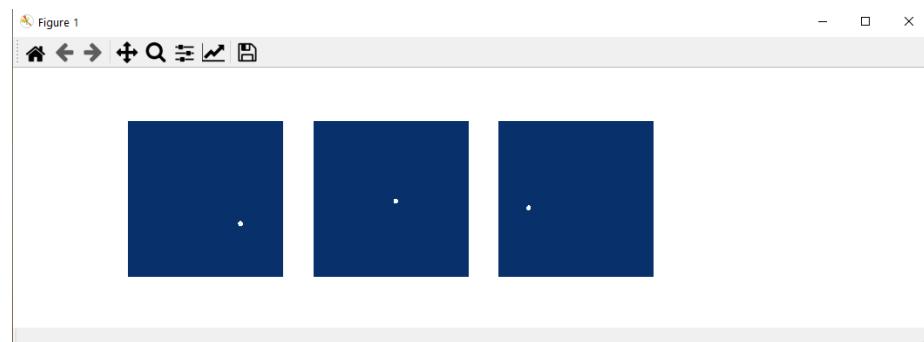


Figura E.4: Máscaras de los defectos de la imagen

Información de los defectos

Para ver la información sobre los defectos detectados en la imagen:

1. Hacer click en el botón “Información”.
2. Se abre una segunda ventana con la información de los defectos.

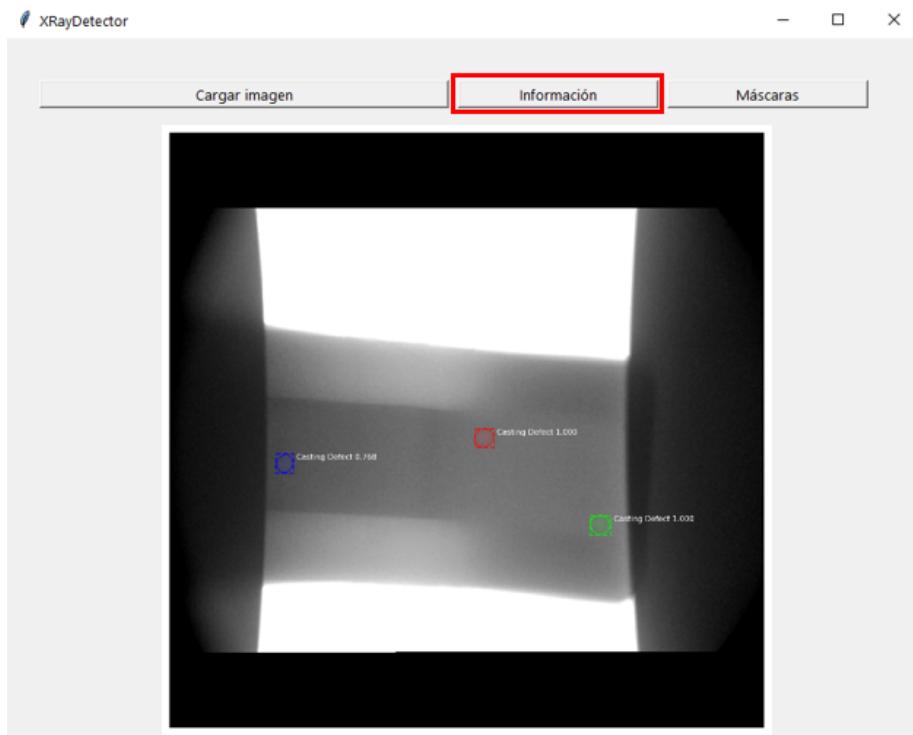


Figura E.5: Visualizar la información de los defectos de la imagen

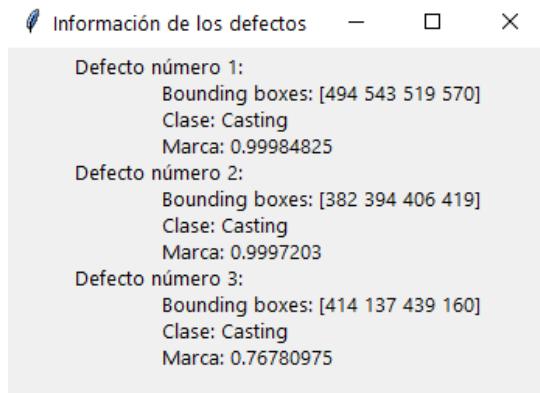


Figura E.6: Información de los defectos de la imagen

Errores

Los errores controlados que pueden salir son:

1. Se produce algún problema al cargar la imagen, mensaje: “Error al cargar la imagen.”. Ver figura E.7.
2. Se carga un archivo que no es una imagen, mensaje: “El archivo no es una imagen.”. Ver figura E.8.
3. Se selecciona un directorio que no existe, mensaje: “El directorio no existe.”. Ver figura E.9.
4. Se hace click en el botón “Detectar defectos” sin haber cargado la imagen, mensaje: “No hay una imagen cargada.”. Ver figura E.10.
5. Se produce algún problema al cargar la imagen después de detectar los defectos, mensaje: “Error al cargar la imagen.”. Ver figura E.7.
6. No haya defectos en la imagen y se hace click en el botón “Máscaras”, mensaje: “No hay defectos en la imagen.”. Ver figura E.11.
7. No haya defectos en la imagen y se hace click en el botón “Información”, mensaje: “No hay defectos en la imagen.”. Ver figura E.11.



Figura E.7: Problema al cargar la imagen

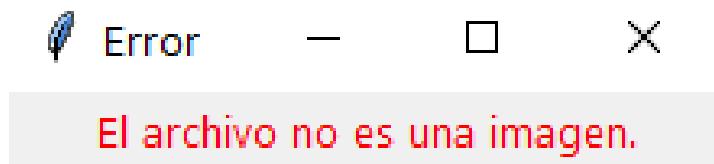


Figura E.8: El archivo cargado no es una imagen



Figura E.9: No existe el directorio seleccionado



Figura E.10: No hay una imagen cargada para detectar defectos

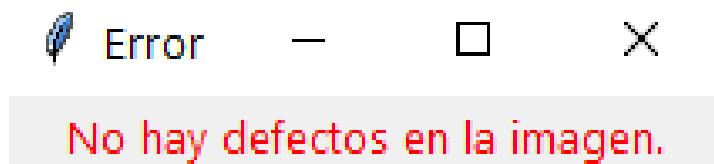


Figura E.11: No hay defectos detectados en la imagen

Bibliografía

- [1] Max Ferguson. Repositorio de detection en GitHub. <https://github.com/maxkferg/metal-defect-detection>, 2016. [Online; Accedido 21-noviembre-2019].
- [2] Max Ferguson. Automatic Localization of Casting Defects. <https://web.stanford.edu/~maxferg/automated-inspection/>, 2017. [Online; Accedido 8-mayo-2020].
- [3] Pablo Ulloa Castro. Programación orientada a objetos: Patrón Singleton. <https://medium.com/@pabulloacastro/programaci%C3%B3n-orientada-a-objetos-patr%C3%ADn-singleton-423e2755614b>. [Online; Accedido 16-julio-2020].
- [4] Vladimir Riffo, Hans Lobel, and Domingo Mery. Gdxray: The database of x-ray images for nondestructive testing. *Journal of Nondestructive Evaluation*, 05 2015.