Proyecto:

Clasificación y localización de defectos en superficies de metal

Sistema Funcional Final

Autores:
Natalia Lourdes Pérez García de la Puente
Vicente Gilabert Maño
Luis Rosario Tremoulet
Modelo: IEEE/ANSI 830-1998

Contenidos

1.Introducción

Se va a presentar el documento del sistema funcional final del sistema de visión artificial para la clasificación y localización de defectos sobre superficies metálicas.

1.1. Descripción general del sistema funcional

La aplicación sigue una arquitectura de cliente/servidor en la que el cliente (aplicación en C) manda una imagen al servidor mediante HTTP y el servidor devuelve un fichero con los resultados de la clasificación (bounding box y tipo de defecto).

Existe la opción de guardar los resultados dibujados sobre la imagen y se guardan en disco. En la siguiente imagen se muestra un ejemplo del resultado:

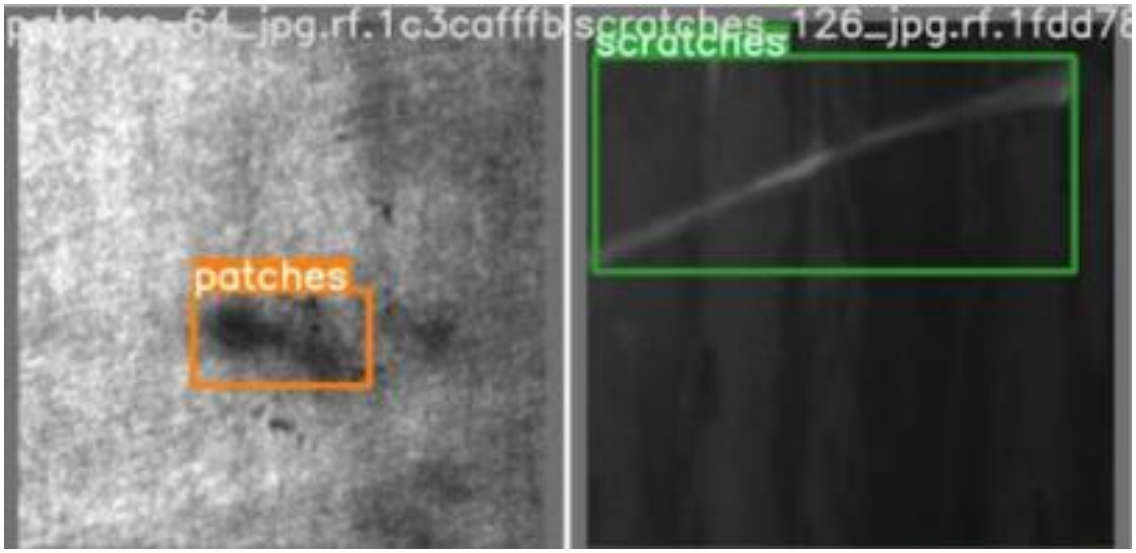

Figura 1. Ejemplo de una clasificación con localización.

Observaciones:

De forma general, la red neuronal es capaz de detectar los tres tipos de imperfecciones en el conjunto de *test*. En ciertas ocasiones predice dos tipos de imperfecciones en la misma imagen. Para dar solución a esta problemática se utiliza una lógica para quedarse con el tipo de defecto que más confianza aporta.

1.2. Descripción de la instalación

En primer lugar, los enlaces de interés relativos al proyecto son los siguientes:

- Repositorio de GitHub donde se ha desarrollado el proyecto:

  https://github.com/Luisrosario2604/AIVA_2022_METAL

- Imagen del proyecto en DockerHub.

  https://hub.docker.com/repository/docker/luisrosario04/aiva_2022_metal

A continuación, se detallan los pasos necesarios para poner la aplicación en funcionamiento:

1. **Instalar** Docker.
2. **Descargarse la imagen docker** mediante el siguiente comando:

   ```
   docker pull luisrosario04/aiva_2022_metal
   ```

3. **Comenzar un contenedor docker** con la imagen descargada y dejarlo ejecutando de fondo.

   El contenedor docker actúa como el servidor y se mantiene a la espera de recibir peticiones, en este caso imágenes, por parte de un cliente. El comando para lanzar el contenedor docker:

   ```
   docker run -p 8000:5000 luisrosario04/aiva_2022_metal
   ```

4. **Para mandar peticiones como cliente se ha utilizado POSTMAN**. Esta plataforma permite enviar un GET simulando al cliente y manda una imagen en base64 al servidor mediante HTTP para posteriormente visualizar la respuesta del servidor.

   ```
   "base64_image": "data:image/jpeg;
   base64,/9j/4AAQSkZJRgABAQEAYABgAAD/..."
   ```

2. Clases

Respecto a la arquitectura de clases empleada, finalmente se utilizan tres clases.

- La clase *System Recognition,* planteada como interfaz, que permite realizar un despliegue con capacidad de crecer y aumentar la cantidad de modelos con los que implementar la detección.
- La clase *Imperfection*, como lista de clases que describen los defectos.
- Por último, la clase *YOLOv5*, que permite realizar la llamada al modelo concreto.

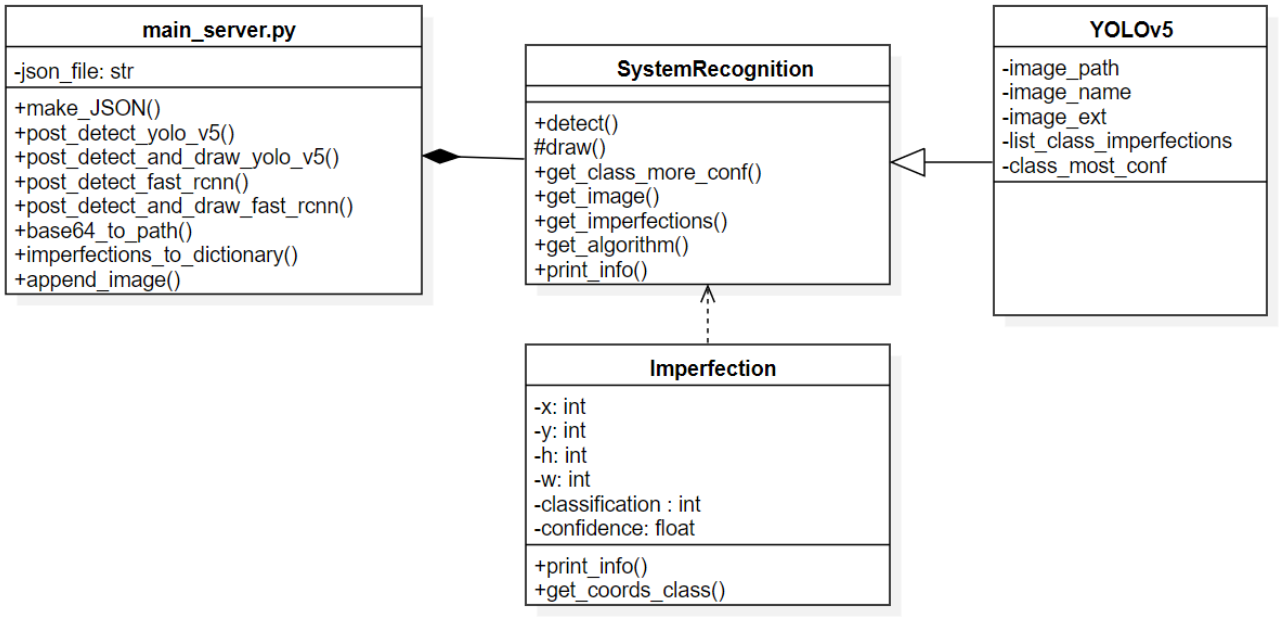

Figura 2. Diagrama de clases de la aplicación.

Estas clases están controladas a partir de un archivo *main_server.py* que es el que permite poner en funcionamiento la aplicación.

3. Despliegue

El diagrama de despliegue UML de la Figura 3 representa la arquitectura servidor/cliente en la que se basa la aplicación desarrollada.

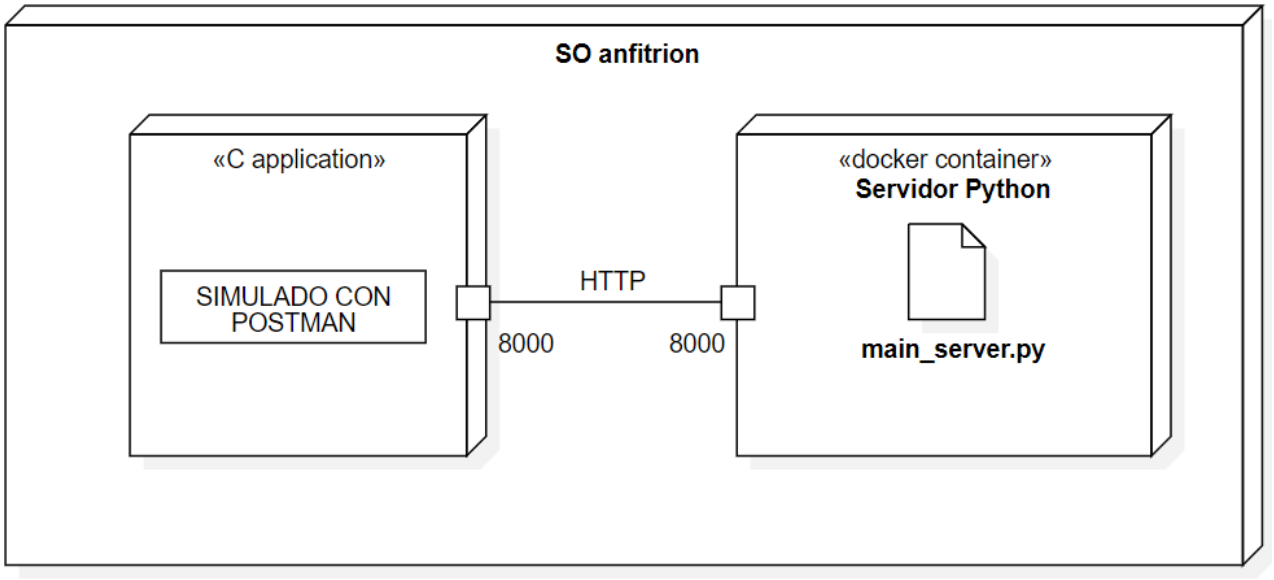

Figura 3. Diagrama UML de despliegue.

Como se puede observar, existe un cliente local en C (simulado con la plataforma POSTMAN) y un servidor en Python para recibir la imagen y realizar la clasificación.

Ambos emplean los puertos 8000 para la conexión. Tanto el servidor como el cliente se encuentran en la misma máquina/ordenador, -esto no tiene por qué ser así si se define un IP estática para el servidor-.

El contenedor docker actúa como el servidor y en su interior tiene todas las dependencias necesarias (Python, OpenCV, Torch, etc.) para ejecutar la aplicación. Cuando se lanza el contenedor docker, éste comienza la conexión HTTP y se conecta al puerto 8000, a la espera de recibir peticiones por parte del cliente.

El cliente (POSTMAN) envía una petición con la imagen en base64 al puerto 8000 y se mantiene a la espera de recibir una respuesta, que en este caso es la imagen con las detecciones realizadas.

4. Funcionamiento del sistema

El funcionamiento del sistema se muestra en el siguiente diagrama de secuencia:

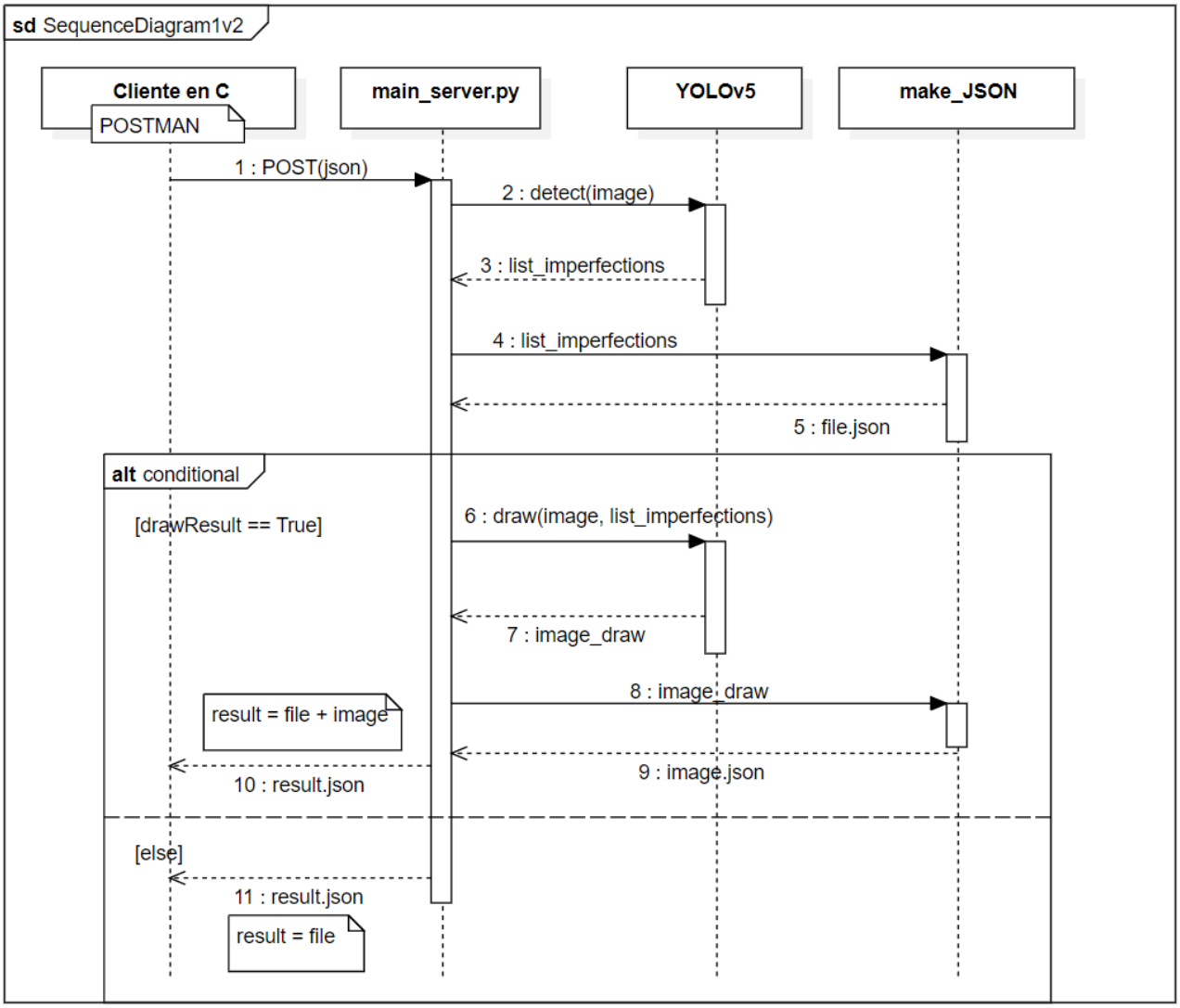

Figura 4. Diagrama de secuencia del funcionamiento del sistema.

A continuación, se describe el funcionamiento del sistema.

En primer lugar, la imagen codificada en base64 sobre la que se quiere realizar la detección es enviada por el cliente. Esta imagen debe ser procesada correctamente por el servidor. Por otra parte, cuando el servidor se lanza se carga la red neuronal y se mantiene a la espera de recibir imágenes por parte del cliente.

A continuación, al recibir una imagen, se realiza la detección con el modelo y se devuelve una lista de imperfecciones. Esa lista se envía a un creador de ficheros JSON, que es el elemento devuelto por el servidor con los resultados de la clasificación (*bounding box* y tipo de defecto). Existe la opción de enviar también la imagen resultado (en base64) con los defectos pintados con su clasificación.

A continuación, se puede ver el resultado de un archivo JSON recibido después de la clasificación por parte del servidor.


```
{"Algorithm":"Yolo V5",
"Class":0,
"Imperfections":
[
{"Class":0,"confidence":0.404082,"height":0.16,"width":0.085,"x":0.6225,"y":0.235},
{"Class":0,"confidence":0.472998,"height":0.155,"width":0.075,"x":0.1975,"y":0.0975},
{"Class":0,"confidence":0.62408,"height":0.43,"width":0.13,"x":0.615,"y":0.785},
{"Class":0,"confidence":0.655571,"height":0.215,"width":0.13,"x":0.635,"y":0.4575}
],
"Result image":"iVBORw0KGgoAAAANSUhEUgAAAMgAAADICAIAAAAiOjnJAAAgAElEQVR4AZTB2ZJeVZYl6jHmXPt3EBBEJEE81
}
```


Figura 5. Resultado archivo JSON.

La imagen “Result image” es la imagen resultante en base64 con los defectos pintados sobre ella y con el porcentaje de confianza de la predicción.