

Proyecto: Detección de Árboles en Imágenes Aéreas

Sistema Funcional Final

Autores: Pablo Asensio Martínez
Vanessa Lomas García
Modelo: IEEE/ANSI 830-1998

1. Introducción	2
1.1. Descripción general del sistema funcional	2
1.2. Descripción de la instalación.....	2
2. Despliegue.....	3
3. Funcionamiento del sistema.....	5

1. Introducción

En este documento se va a presentar el sistema funcional final de visión artificial que se ha desarrollado para detectar árboles en imágenes aéreas.

1.1. Descripción general del sistema funcional

La aplicación sigue una arquitectura de cliente/servidor en la que el cliente manda una imagen al servidor (contenedor docker) y el servidor devuelve la imagen de entrada con la detección de árboles realizada, donde cada árbol detectado se ha marcado con una *bounding box* circular y su correspondiente centro. Dicha imagen además de mostrarse por pantalla, también se guarda en disco.

En la siguiente figura se muestra un ejemplo del resultado que se obtiene a la salida de la aplicación cuando se le pasa una imagen como entrada:



Figura 1. Ejemplo de detección del sistema funcional final

Cabe mencionar que las dimensiones de la imagen de entrada pueden ser las que se desee ya que internamente la imagen se irá recorriendo en pequeñas ventanas sobre las que se irá realizando la detección, y que finalmente se combinarán para dar como resultado la detección completa sobre la imagen.

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/pasensio97/AIVA_2021-imagenes_aereas

- Imagen en DockerHub.

https://hub.docker.com/r/pasensio97/tree_detector_image

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

1. Instalar [Docker](#).
2. **Aceleración de contenedores docker con GPU:**
 - **En Linux:** instalar [NVIDIA Container Toolkit](#) (nvidia-docker) que permite acelerar la ejecución de aplicaciones en contenedores Docker mediante el uso de GPU.
 - **En Windows:** omitir este paso ya que nvidia-docker no está disponible. En Windows por defecto se utilizará la CPU en vez de la GPU, por lo que es recomendable ejecutar la aplicación en Linux.
3. **Descargarse la imagen docker** mediante el siguiente comando:

```
docker pull pasensio97/tree_detector_image
```

4. **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. Comando para lanzar el contenedor docker:

- **En Linux:**

```
docker run -it -p 8000:8000 --rm --gpus=all  
pasensio97/tree_detector_image python manage.py
```

- **En Windows** (se omite la opción --gpus=all)

```
docker run -it -p 8000:8000 --rm pasensio97/tree_detector_image python  
manage.py
```

5. Para mandar peticiones como cliente se utilizará el archivo *client.py*, que se puede descargar desde el [repositorio](#) de GitHub. Este archivo permite dos argumentos de entrada:
 - --input: ubicación de la imagen sobre la que se quiere realizar la detección.
 - --output: ubicación en la que se quiere guardar la imagen de salida. Si no se indica ninguna ruta se guarda por defecto en el directorio desde el que se haya ejecutado el archivo *client.py*.

Ejemplos de comandos para ejecutar client.py son:

```
python client.py --input img.png
```

```
python client.py --input input_image.png --output output_img.png
```

2. Despliegue

El diagrama de despliegue UML de la Figura 2 representa la arquitectura servidor/cliente en la que se basa la aplicación desarrollada.

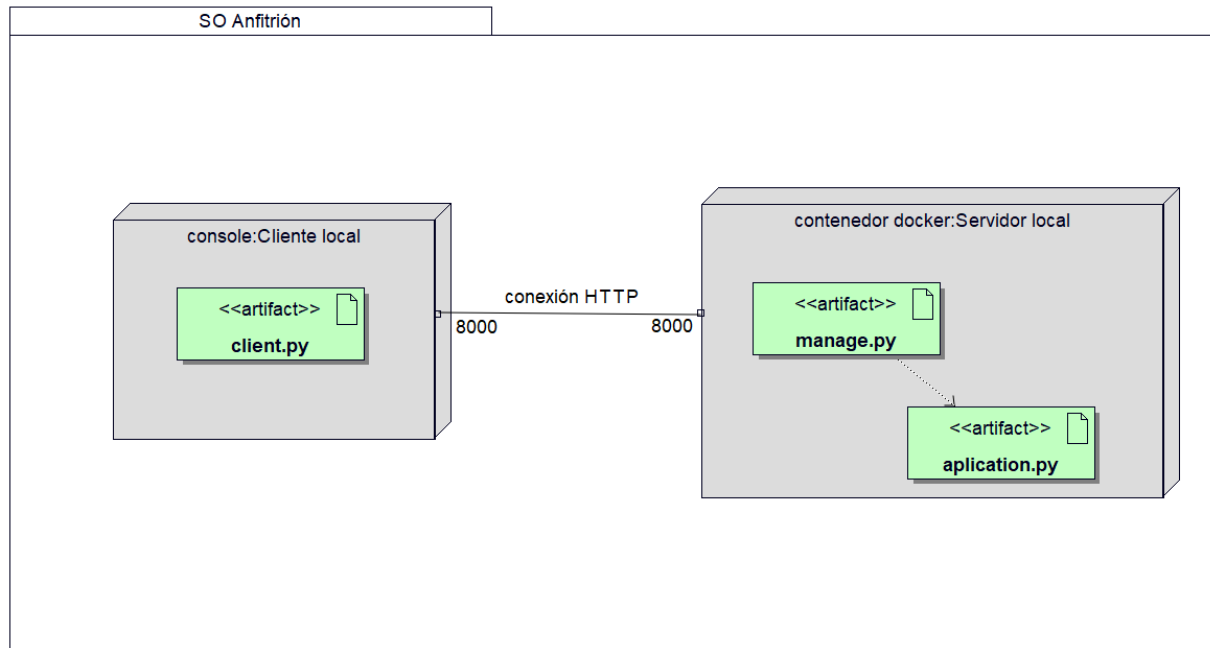


Figura 2. Diagrama de despliegue UML

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, TensorFlow, Keras, 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. Por su parte, el cliente se lanza desde una ventana de comandos en la que se llama a *client.py* y se especifica la imagen sobre la que se quiere realizar la detección. El cliente se conecta al puerto 8000, envía un POST y se mantiene a la espera de recibir una respuesta, que en este caso es la imagen con las detecciones realizadas.

3. Funcionamiento del sistema

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

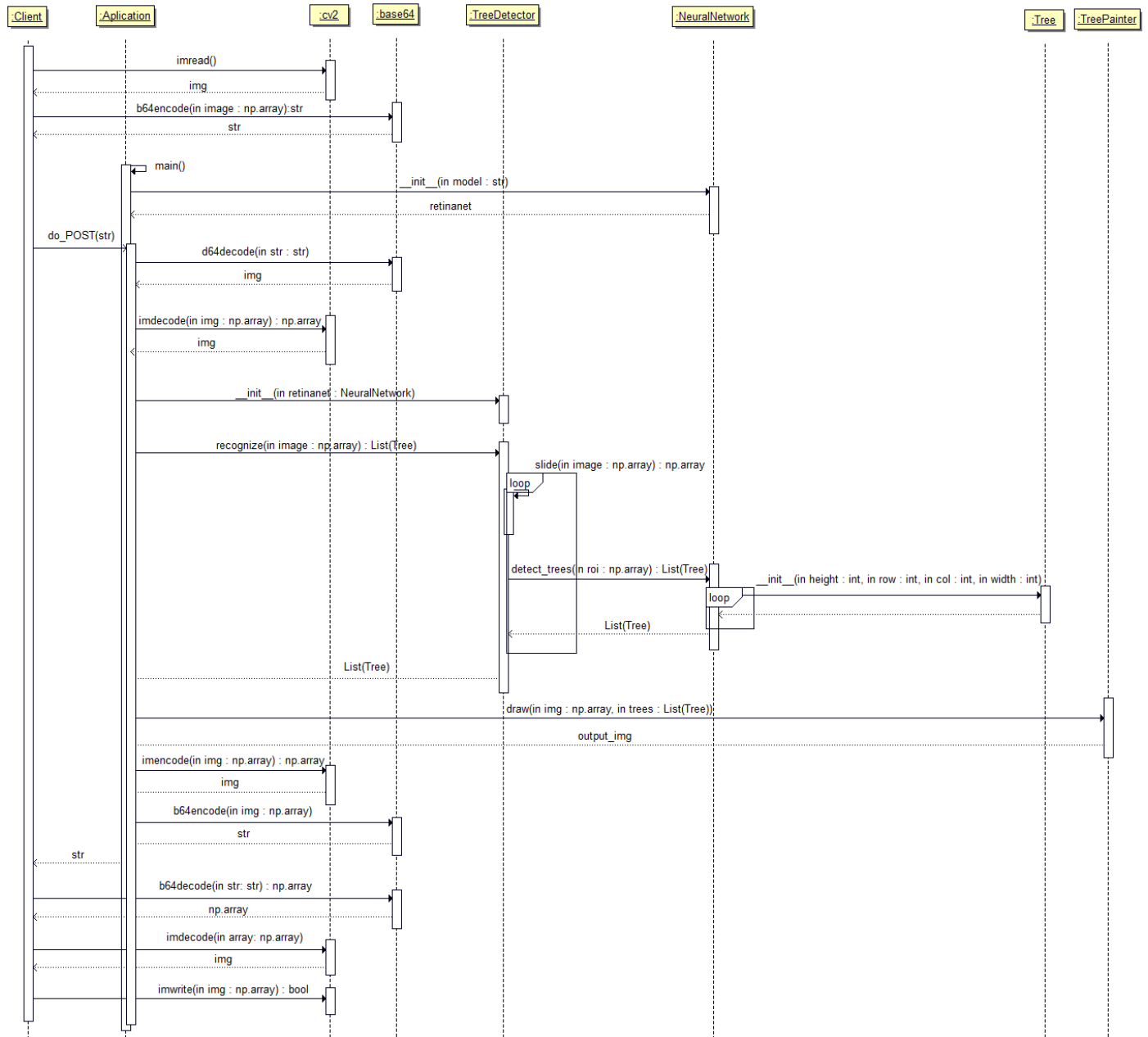


Figura 3. Diagrama de secuencia del funcionamiento del sistema

A continuación, se describe el funcionamiento del sistema:

La imagen sobre la que se quiere realizar la detección de árboles es enviada por el cliente, que previamente tiene que codificarla en base64 para procesarla correctamente en el servidor.

Por otra parte, cuando el servidor se lanza por primera vez, se carga la red neuronal y se mantiene a la espera de recibir imágenes por parte del cliente. Una vez que le llega una imagen, la decodifica hasta llegar a tenerla en formato *np.ndarray* y así poder ser procesada por la red neuronal

correctamente. A continuación, se crea un objeto de tipo `TreeDetector` y se llama a su método *recognize()*, que a su vez llama al método *_slide()* para ir recorriendo la imagen. Sobre cada subimagen se va realizando la detección mediante el método *detect_trees()* de la clase `NeuralNetwork`. Por cada árbol detectado en la imagen se genera un objeto tipo `Tree`, obteniéndose finalmente una lista de objetos tipo `Tree`, que junto con la imagen de entrada se pasarán al método *draw()* de la clase `TreeDetector` para dibujar sobre la imagen la posición de cada árbol detectado con una *bounding box* circular y su correspondiente centro. Esta imagen se codifica en base64 y se devuelve al cliente, que tendrá que decodificarla para posteriormente mostrarla por pantalla y/o guardarla en disco.