Proyecto: Detección de Árboles en Imágenes Aéreas

Sistema Funcional Final

Autores: Pablo Asensio Martínez

Vanesa Lomas García

Modelo: IEEE/ANSI 830-1998

[1. Introducción 2](#_Toc71469490)

[1.1. Descripción general del sistema funcional 2](#_Toc71469491)

[1.2. Descripción de la instalación 2](#_Toc71469492)

[2. Despliegue 4](#_Toc71469493)

[3. Funcionamiento del sistema 4](#_Toc71469494)

[4. Rendimiento del sistema y conclusiones 5](#_Toc71469495)

# Introducción

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

## Descripción general del sistema funcional

La aplicación sigue una arquitectura de cliente/servidor (ambos en la misma máquina) en la que el cliente manda una imagen al servidor (imagen 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:

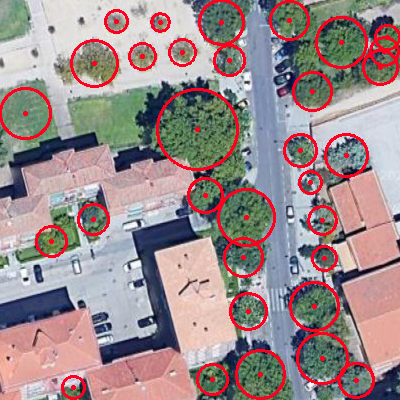
 

Figura 1. Ejemplo de detección del sistema funcional final

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.

## 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 os pasos necesarios para poner la aplicación en funcionamiento:

1. Instalar [Docker](https://docs.docker.com/get-docker/).
2. **Aceleración de contenedores docker con GPU**:

* **En Linux**: instalar [NVIDIA Container Toolkit](https://github.com/NVIDIA/nvidia-docker) (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.

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

docker pull pasensio97/tree\_detector\_image

1. **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 el comando –gpus=all)

docker run -it -p 8000:8000 --rm pasensio97/tree\_detector\_image python manage.py

1. Para mandar peticiones como cliente se utilizará el archivo **client.py**, que se puede descargar desde el [repositorio](https://github.com/pasensio97/AIVA_2021-imagenes_aereas) de GitHub. Este archivo permite dos argumentos de entrada:

* --path\_input\_image: para indicar la ubicación de la imagen sobre la que se quiere realizar la detección.
* --path\_output\_image: para indicar la 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

# Despliegue

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

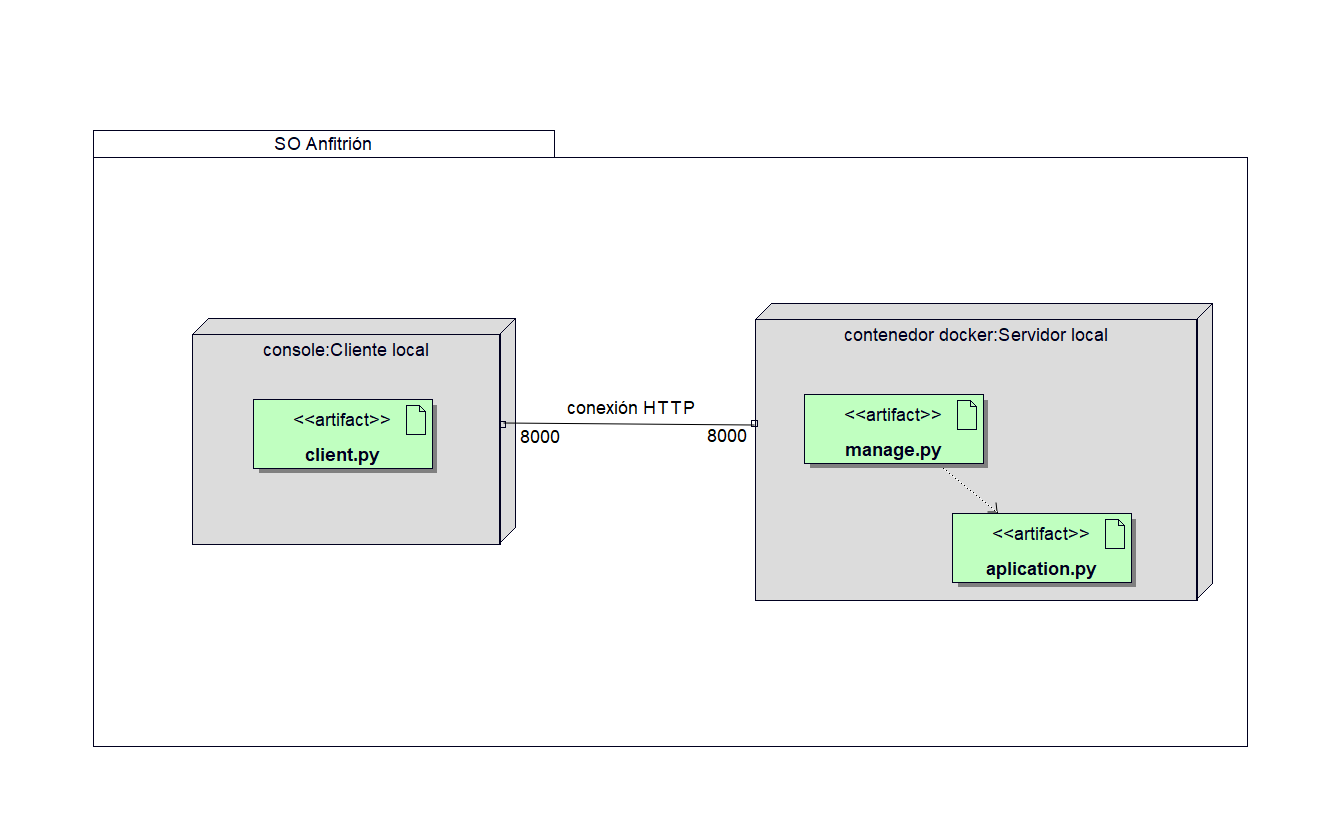


Figura 2. Diagrama de despliegue UML

Tanto el servidor como el cliente se encuentran en la misma máquina/ordenador bajo el mismo sistema operativo. 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, lanza un POST y se mantiene a la espera de recibir una respuesta, que en este caso es la imagen con las detecciones realizadas.

# Funcionamiento del sistema

# Rendimiento del sistema y conclusiones

Para evaluar el rendimiento de la aplicación desarrollada se han calculado métricas, en particular la curva Precision–Recall y el valor Average Precision (AP), que son métricas más populares que se utilizan para evaluar los modelos de detección de objetos. En concreto, se han utilizado las métricas que se utilizan en la conocida competición Pascal VOC, implementada en [1].

Por ello, ha sido necesario lo siguientes:

* Elaborar un conjunto de imágenes de test (no ‘vistas’ anteriormente por la red).
* Etiquetarlas manualmente para generar los archivos de ground truth para cada una de las imágenes de test.
* Pasar cada una de las imágenes de test por la red para obtener así los archivos con las detecciones realizadas.
* Calcular métricas a partir de los archivos de ground truth y las detecciones.

La curva Precisión – Recall obtenida se muestra en la Figura X. Esta curva lo que expresa es como varían los valores de precisión y recall al ir variando el umbral de confianza (valor de IoU). Un detector ideal es aquel para el que la precisión se mantiene alta a medida que aumenta el recall, es decir, un detector que tenga pocos falsos positivos(FP) y pocos Falsos Negativos(FN). En nuestro caso, como se puede observar, el valor de precisión va disminuyendo a medida que el valor de recall aumenta, lo que implica que para que se detecten el mayor de ;arboles posibles, el valor de falsos positivos también será mayor.

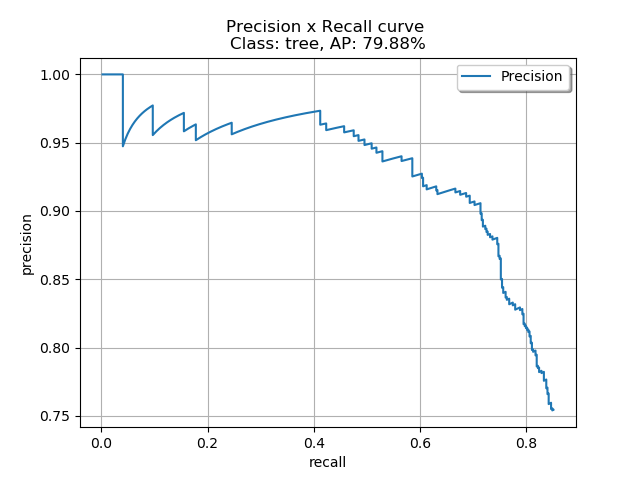


Figura 4. Curva Presision-Recall

Por otro lado, el dato cuantitativo que refleja como de bueno es el detector de árboles desarrollado, viene dado por el valor de Average Precision (AP), que representa el área bajo la curva Precision – Recall, que en este caso tiene un valor del 79,88%.

A continuación se muestras varios ejemplos donde se comparan en una misma imagen las bounding boxes de ground truth (arboles etiquetados manualmente) y las bounding boxes generadas por el modelo entrenado.

Ground Truth

Detecciones red

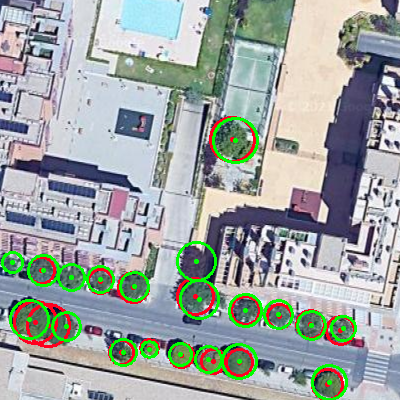
 

Figura 5. Comparativa entre ground truth y detecciones del modelo sobre imágenes de test

Como puede comprobarse, las detecciones realizadas por el modelo entrenado se aproximan bastante bien al ground truth. Sin embargo, cuando hay varios árboles juntos las detecciones no son tan precisas, como puede verse en la segunda imagen de la Figura X. Hay que tener claro que desde una vista aérea, incluso para el ojo humano, es muy difícil determinar el número exacto de árboles cuando se da la situación en la que hay varios árboles muy juntos. Aun así, la aplicación desarrollada consigue dar una buena estimación de la posición y del número de árboles que hay en una imagen aérea.