

# Proyecto CarDensityAI

## Despliegue y Funcionamiento del Sistema

**Autora:** Blanca Rodríguez González

**Autor:** Francisco Vázquez Donaire

## Índice

<b>1. Introducción</b>	<b>2</b>
1.1. El sistema: CarDensityAI . . . . .	2
1.2. Instrucciones de instalación . . . . .	3
<b>2. Despliegue</b>	<b>4</b>
<b>3. Funcionamiento del Sistema</b>	<b>5</b>

## 1. Introducción

Este documento tiene como objetivo recoger el despliegue así como el funcionamiento del sistema de visión artificial propuesto para la detección y recuento de vehículos a partir de imágenes aéreas.

Con este fin, se presentará, de forma resumida, el sistema desarrollado y se esbozarán las instrucciones de instalación del mismo para su uso general.

### 1.1. El sistema: CarDensityAI

Como solución al problema de estimación de tráfico planteado por el Excmo. Ayuntamiento contratador, los desarrolladores de la empresa *AISolutions* diseñan un sistema de visión artificial basado en técnicas de aprendizaje profundo para la detección de vehículos. En este sentido, tras una etapa de preprocesado de la imagen a analizar en la que se genera subconjuntos de imágenes, se hace inferencia sobre cada una de ellas y, posteriormente, se reagrupan, guardando los resultados finales.

El sistema ***CarDensityAI*** se plantea, por tanto, siguiendo una arquitectura cliente/-servidor en la que el cliente enviará una imagen aérea de una ciudad determinada al servidor mediante un protocolo TCP/IP. La totalidad de este sistema estará recogida en un contenedor docker. La aplicación devolverá la detección en dos formatos: una imagen formato *png* en la que se recuadran los vehículos detectados y un fichero *csv* en la que se recogen las coordenadas de las detecciones así como el número total de vehículos contados. Esta información se guardará en un directorio de salida.

Es importante mencionar que, el cliente solicitará el análisis de la imagen desde un portal web en el que se seleccionará la imagen de interés y, tras el tiempo pertinente de procesado, se descargará el resultado en un directorio temporal del directorio donde se encuentre el proceso.

La Figura 1 muestra un ejemplo de detección del sistema prototipo planteado.



Figura 1: Resultado del sistema CarDensityAI

El código fuente del sistema así como otra información adicional se encuentra en un repositorio de GitHub creado con este propósito. Además, el sistema se encapsula en una imagen docker en DockerHub.

## 1.2. Instrucciones de instalación

Habiendo introducido la esencia del sistema propuesto, se presentan a continuación las instrucciones de puesta en marcha del mismo por primera vez.

1. Instalar docker. Para ello, se accederá a la página [web oficial de docker](#) y se instalará la versión apropiada de la máquina donde se ejecute (disponible para Linux, Windows y MacOS).  
Una vez instalada, la aplicación se mantendrá deberá mantener abierta siempre que se ejecute la aplicación CarDensityAI.
2. Descargar el archivo *docker-compose.yml* del [repositorio de GitHub](#) destinado al proyecto.
3. Abrir una terminal del sistema sobre el directorio que guarde el fichero mencionado anteriormente. Para este paso podemos abrir la terminal genérica y buscar la carpeta en cuestión o abrir directamente la terminal desde el explorador de archivos.
4. Ejecutar el siguiente comando sobre ella para proceder a la descarga de la [imagen docker de la aplicación desde DockerHub](#) así como su ejecución.

```
docker compose up
```

Este paso tardará unos minutos la primera vez que se ejecute la aplicación.

5. Manteniendo el contenedor iniciado, se abrirá un navegador web y se accederá a la dirección <http://localhost:8000>. Esto dará acceso directo a la aplicación web.
6. Una vez en la aplicación web, esta se podrá usar en un entorno amigable, seleccionando la imagen en nuestro ordenador que se deseé analizar. Una vez el proceso haya terminado, se guardará en una carpeta temporal llamada *results* los resultados obtenidos (imagen original, imagen de salida, fichero de información csv). Es importante mencionar que, en caso de que se ejecute de nuevo la aplicación los resultados se sobre-escribirán, por lo que el usuario deberá, manualmente, mover los resultados a otro directorio para evitar la pérdida de información.

Para futuros usos de la aplicación, se repetirá el proceso indicado encima de estas líneas desde el apartado 3; es decir, se abrirá una terminal sobre el directorio contenido los archivos del proyecto y se ejecutará el comando *docker compose up*. Esto pondrá en marcha el sistema desarrollado, permitiendo al usuario procesar, desde un navegador web, la imagen de interés.

## 2. Despliegue

Una vez se ha descrito el sistema, así como la puesta en marcha del mismo, se discutirá, de forma más detallada, los aspectos físicos del sistema para entender las comunicaciones entre los elementos del mismo. Con este propósito se presenta el diagrama de despliegue UML en la Figura 2.

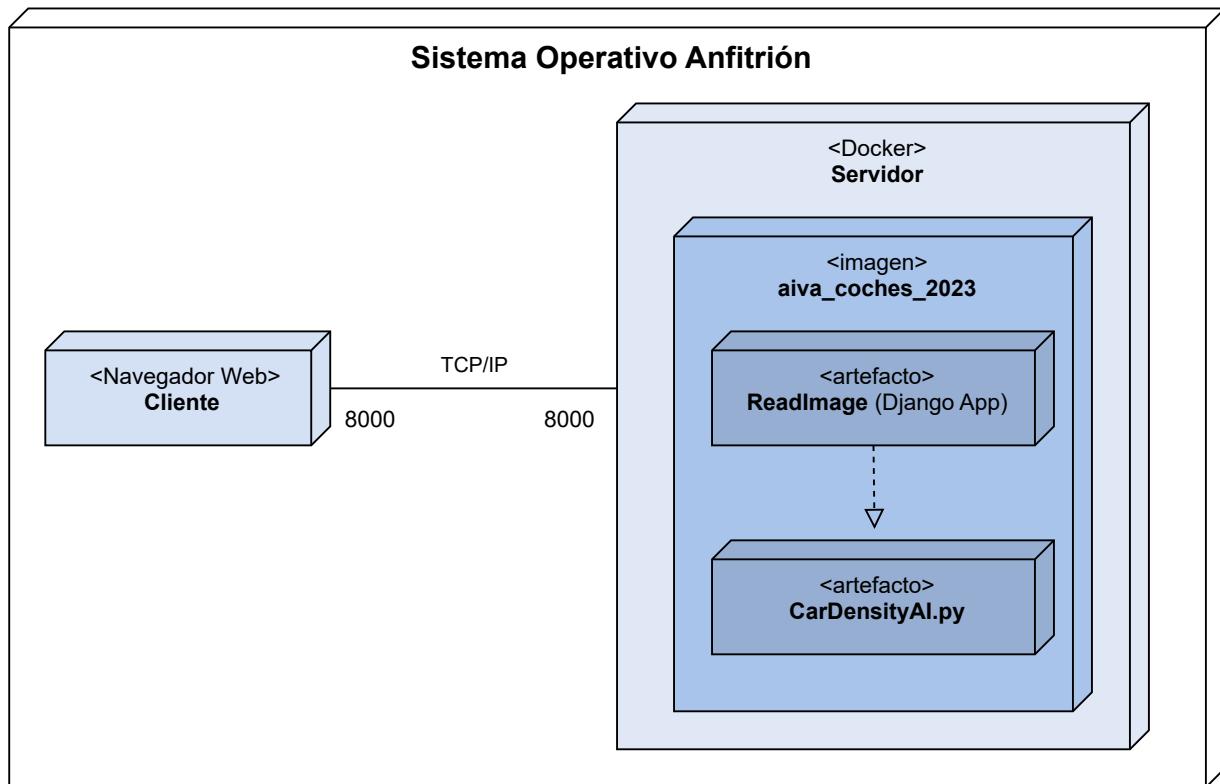


Figura 2: Diagrama de despliegue UML para la aplicación CarDensityAI.

Como se observa, el sistema CarDensityAI se encapsula en una imagen Docker, en la que se guardará la aplicación así como todas las dependencias necesarias para su ejecución. Precisamente, en ella se encuentra la aplicación web desarrollada con Django y el sistema CarDensityAI.

Una vez se ha descargado la imagen Docker, utilizando el archivo *docker-compose.yml*, se ejecutará un contenedor contenido la aplicación desarrollada. Cuando se lanza, este proceso se quedará a la escucha en el puerto 8000, y esperará las peticiones del cliente.

El cliente, por otro lado, hará las peticiones desde un navegador web. Precisamente, el usuario se conectará desde un navegador web a la dirección <http://localhost:8000> y seleccionará y subirá la imagen que se deseé analizar. Este cliente hará un *POST* al servidor encapsulado en el docker, enviándole la imagen para que la procese. Una vez finalizado el proceso de análisis de la imagen, el resultado será guardado en una carpeta results compartida entre máquina local y contenedor.

En este punto, el cliente y el servidor de nuestro sistema se alojan en la misma máquina con un sistema operativo determinado. No obstante, se está trabajando actualmente en la operación del sistema desde dos máquinas independientes.

### 3. Funcionamiento del Sistema

Por último, para la comprensión del sistema desarrollado se presenta el diagrama de secuencia UML mostrado en la Figura 3. En el, se hace una descripción esquemática del comportamiento dinámico de la aplicación.

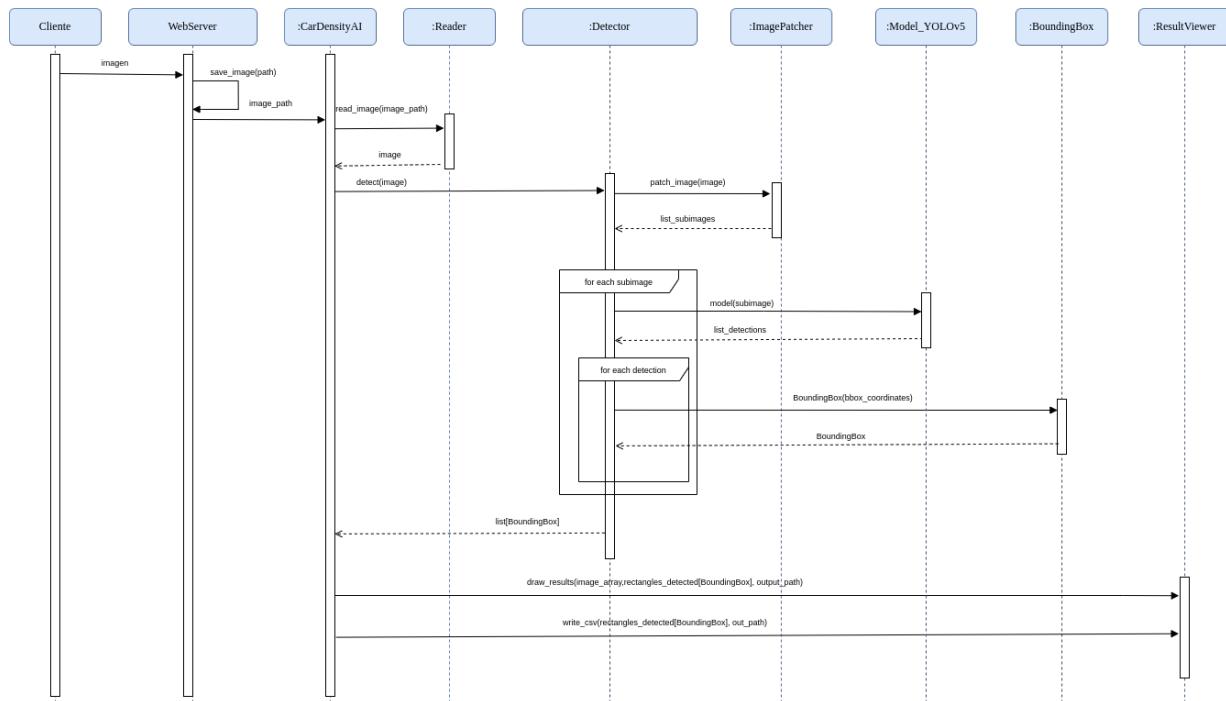


Figura 3: Diagrama de secuencia UML para la aplicación CarDensityAI.

Esta secuencia se presentó en la documentación de diseño entregada al cliente con anterioridad; sin embargo, esta ha sido adaptada a las características actuales del sistema, incluyendo, esta vez, al cliente y al servidor web diseñado (*WebServer*).

Una vez el cliente envía al servidor la imagen que se desea procesar y este la recibe, la imagen se guarda en un directorio temporal de la máquina hospedadora mediante la función *save\_image()*. Posteriormente, se guarda el directorio donde esta imagen se guarda e, introduciendo esta información, se llama al algoritmo *CarDensityAI*, entrando en el proceso desarrollado para la estimación de tráfico.

El algoritmo (*CarDensityAI*) leerá la imagen que se desea analizar mediante el uso de una clase propia, la clase *Reader* que, gracias a su método *read\_image*, devolverá la matriz de la imagen cargada si se ha indicado correctamente su ruta de almacenamiento. Con la imagen cargada, el siguiente paso es la detección de los vehículos encontrados en ella.

Con este propósito, se emplea el método *detect* de clase *Detector*; sin embargo, la imagen no entrará directamente en el proceso de detección si no que, cuando la imagen entra en este proceso, esta sufrirá un parcheo. El propio detector llamará al método *patch\_image* encontrado en la clase *ImagePatcher* el cual, dada la imagen original, devuelve una lista de imágenes de menor tamaño sobre las que trabajaremos para realizar las detecciones de manera eficiente y rápida.

Para cada una de las subimágenes generadas por el parcheador, se hace inferencia. Con este propósito, se llama al modelo pre-entrenado (YOLOv5), empleando *model()*, encontrado en una **clase externa** diseñada y encontrada en PyTorch. El modelo original arrojará una lista de cinco elementos, en el que los cuatro primeros guardarán la información de las esquinas de la caja contenedora y el quinto indicará la confianza de la detección. Esta lista se transformará a nuestro formato específico *BoundingBox*.

Este proceso se repite para todas las imágenes devueltas por el parcheador, guardando las detecciones en una global de detecciones. Cuando se procesan todas las imágenes, la lista de salida se empleará para generar los archivos de salida. Precisamente, se llamará al método *draw\_results* incluido en la clase *ResultViewer* que, tras pasar la lista de coordenadas a coordenadas globales de la imagen original, guarda, en una ruta de salida especificada una imagen png, que contendrá la imagen original pero con los vehículos detectados enmarcados. Además, se generará un fichero .csv mediante el método *writer\_csv* incluido de igual manera en la clase *ResultViewer*. Este método calcula internamente el número de vehículos detectados así como la transformación a coordenadas globales, guardando esta información línea a línea en el fichero de salida (en la ruta especificada).

La imagen y el fichero se guardarán en el mismo directorio donde el servidor web guardará la imagen original y se devolverá al cliente, en la interfaz web, un aviso al usuario de que el proceso ha terminado para que este consulte los resultados.