Proyecto: Detección de Árboles en Imágenes Aéreas

Documento de Diseño

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

1. Introducción

En este documento se va a describir el funcionamiento principal del sistema de visión artificial, mostrando ejemplos de funcionamiento del sistema y explicando los siguientes diagramas UML: estático de clases y el de secuencia.

El repositorio donde se va a desarrollar el sistema es el siguiente: https://github.com/pasensio97/AIVA_2021-imagenes_aereas

1.1. Descripción del Funcionamiento

La aplicación, una vez obtiene la imagen que se debe procesar, junto con el marco de coordenadas WGS84, recorre la imagen en pasos de 400 píxeles hacia la derecha y abajo. Las sub-imágenes son procesadas por una red neuronal de detección (RetinaNet,), la cual devolverá un listado de cada árbol (coordenadas en pixeles sobre la imagen original) que ha encontrado en cada sub-imagen. Seguidamente el listado de los árboles se procesará para obtener más información sobre su geolocalización y tamaño.

1.2. Ejemplos

A continuación, se muestran los resultados que se han obtenido sobre algunas de las imágenes de prueba que se han empleado.

Figura 1. Ejemplo de detección 1

Figura 2. Ejemplo de detección 2

Figura 3. Ejemplo de detección 3

Cabe mencionar que el tiempo medio que se tarda en realizar la detección sobre una imagen de este tamaño (400 x 400) ronda los 0.07 segundos.

Observaciones:

Generalmente la red neuronal es capaz de localizar los árboles siempre y cuando no estén demasiado juntos. En el primer ejemplo, en la zona de la derecha predice más árboles de los que hay. Así mismo, en las zonas en las que el color del suelo o césped es similar al de los árboles, predice más de los que hay, como ocurre en el último ejemplo. Los arbustos no son detectados salvo si tienen un tamaño lo suficientemente grande para ser considerados árboles desde la vista aérea.

2. Diseño de la Solución

En esta sección se explicará cómo se ha diseñado la solución, que irá acompañada de distintos diagramas UML (estático de clases, de secuencia y de actividad).

2.1. Diagrama Estático de Clases

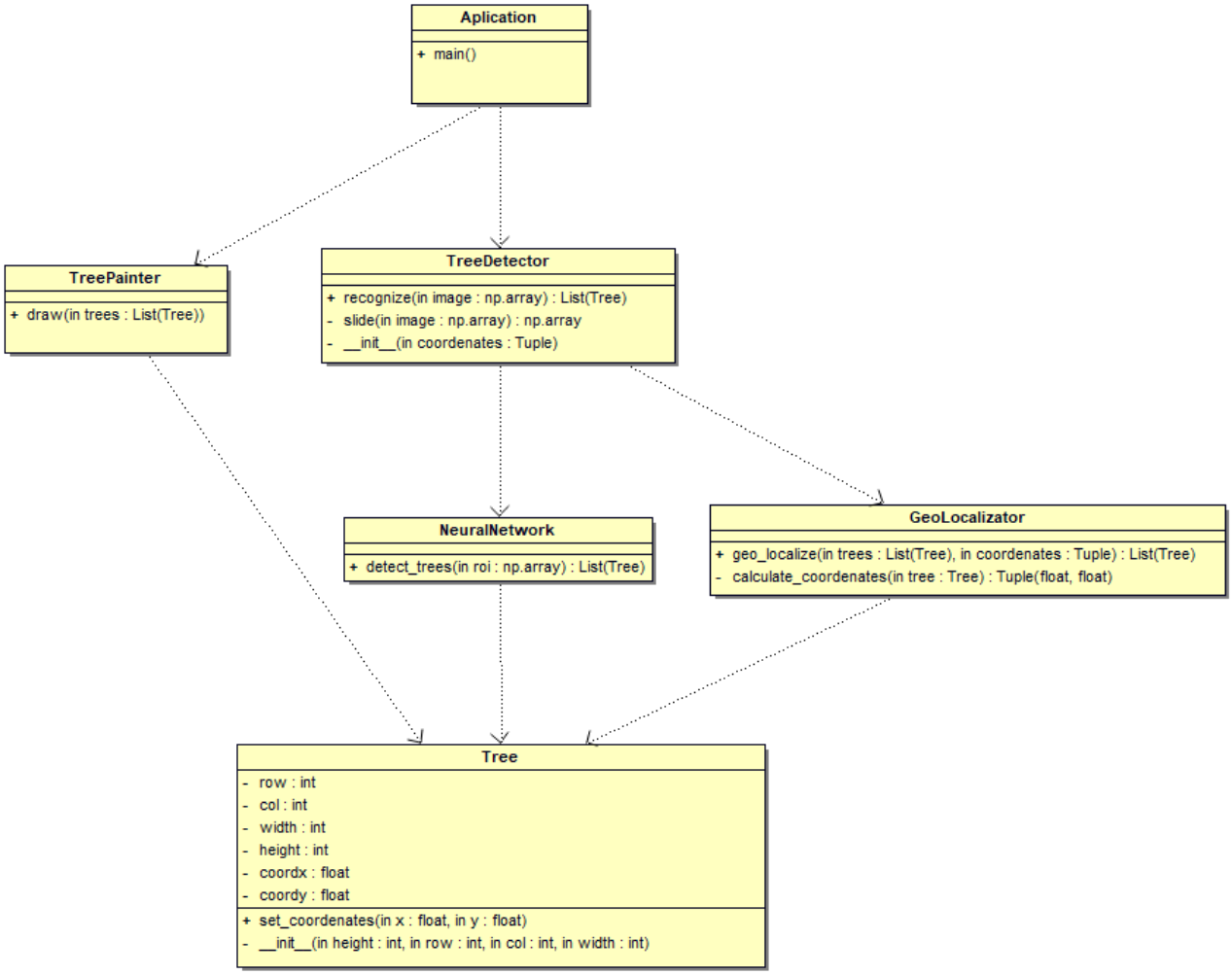

Figura 4. Diagrama Estático de Clases

2.2. Descripción de las Clases

2.2.1. Aplication

Clase general encargada de leer la imagen de entrada y el fichero donde estén guardadas las coordenadas geográficas de la imagen y ordenar el procesamiento de estos. Métodos:

- **main().** Método público que llama a TreeDetector, que es la clase encargada de realizar la detección de árboles sobre la imagen y la geolocalización.

2.2.2. TreeDetector

Clase que recibe la imagen aérea completa sobre la que se quiere realizar la detección de árboles. Es necesario construir la clase pasándole las coordenadas de la imagen. Métodos:

- **recognize(img).** Método público que da como salida una lista de Tree con todos los árboles detectados en la imagen completa.
- **slide(img).** Método privado que divide la imagen aérea completa recibida en subimágenes más pequeñas que serán pasadas a la red neuronal (clase NeuralNetwork). Explicación del algoritmo:

  Este método va recorriendo la imagen por filas y columnas en saltos de 400 píxeles. Cada vez que recorta un cuadrante de 400 x 400 píxeles, lo va pasando a la clase NeuralNetwork, hasta que no quedan cuadrantes.

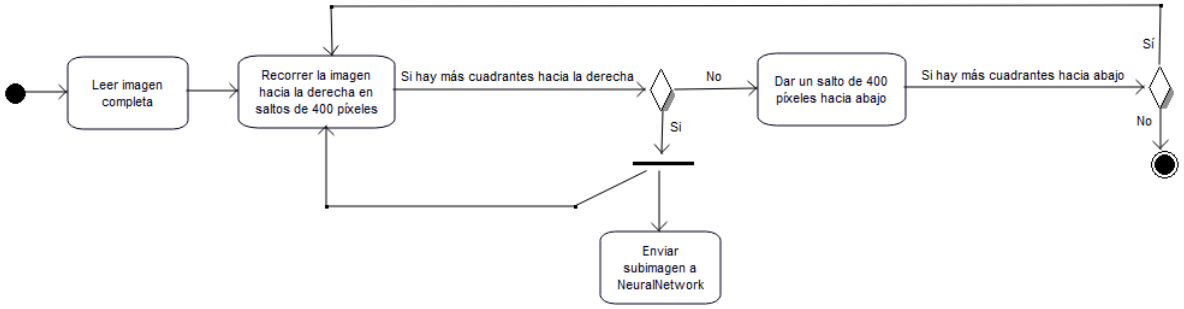

Figura 5. Diagrama de actividad del método slide(img)

2.2.3. Tree

Clase que se crea por cada árbol detectado y que almacena las coordenadas donde se encuentra en la imagen (row, col, width, height), y sus coordenadas geográficas (coordx, coordy). Métodos:

- **set_coordenates(Tuple).** Método público que asigna las coordenadas de este.

2.2.4. NeuralNetwork

Clase encargada de realizar la detección de árboles sobre la subimagen que recibe como entrada. Da como salida una lista de Tree. Métodos:

- **detect_trees(img).** Método público que se encarga de realizar la detección sobre la imagen. Explicación del algoritmo:

  En primer lugar, se selecciona la GPU que se va a utilizar para la ejecución del método. Luego, se carga tanto el modelo entrenado como el modelo preentrenado que se utiliza como *backbone*. Se carga la imagen y luego se preprocesa. Una vez preprocesada la imagen, ya se puede pasar por la red neuronal, que dará como salida una lista de *bounding boxes*, cada una con su respectiva puntuación o *score*, cuyo valor va de 0 a 1. Si el valor de *score* de una determinada bounding box es menor que 0.5, no se creará un objeto de clase *Tree*. En caso contrario, sí se creará y se enlistará.

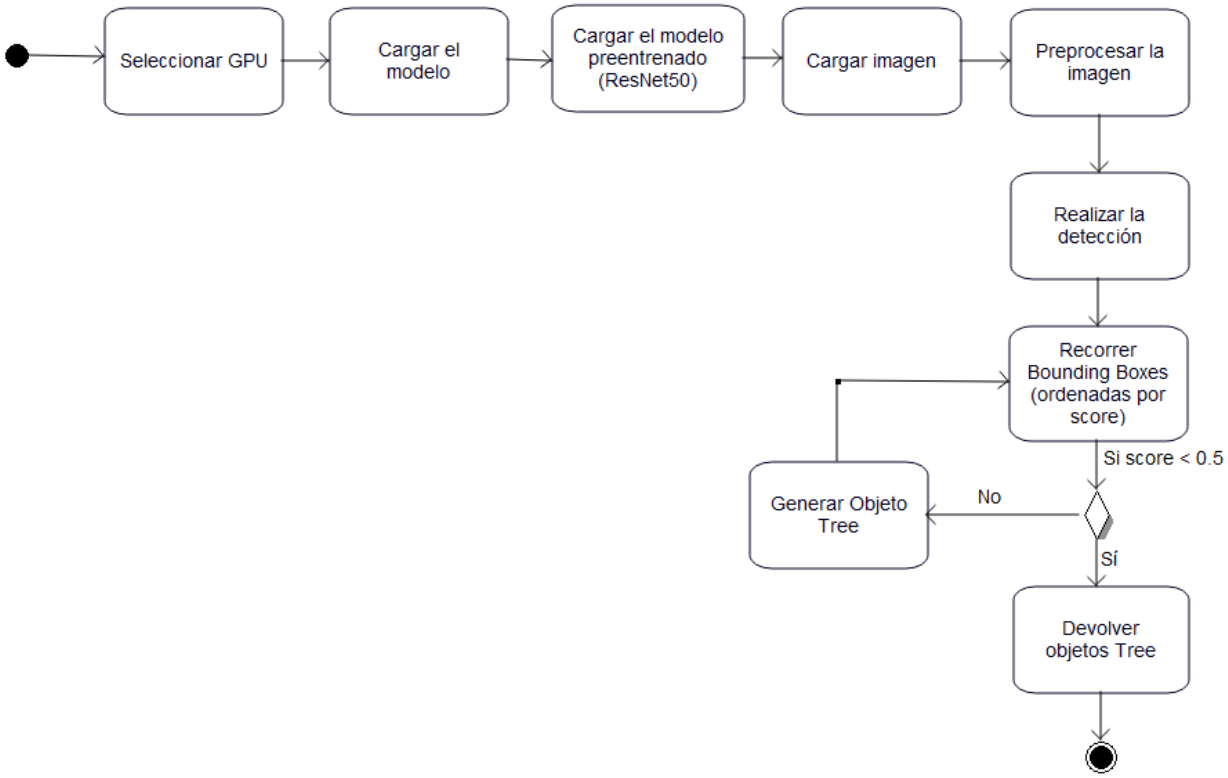

Figura 6. Diagrama de actividad del método detect_tress(img)

2.2.5. GeoLocalizator

Clase que se encarga de geolocalizar los árboles. Métodos:

- **geo_localize(List(Tree), Tuple).** Método público que manda calcular las coordenadas y las asigna a cada Tree.
- **calculate_coordenates(Tree**). Método privado que calcula las coordenadas en WGS84 del píxel central del árbol.

2.2.6. TreePainter

Clase que recibe una lista de Trees y se encarga de dibujar una bounding box circular para cada Tree, así como su punto central a partir de las coordenadas de su bounding box rectangular. Métodos.

- **draw(List(Tree)).** Método público encargado de dibujar las bounding boxes en la imagen.

2.3. Diagrama de Secuencia

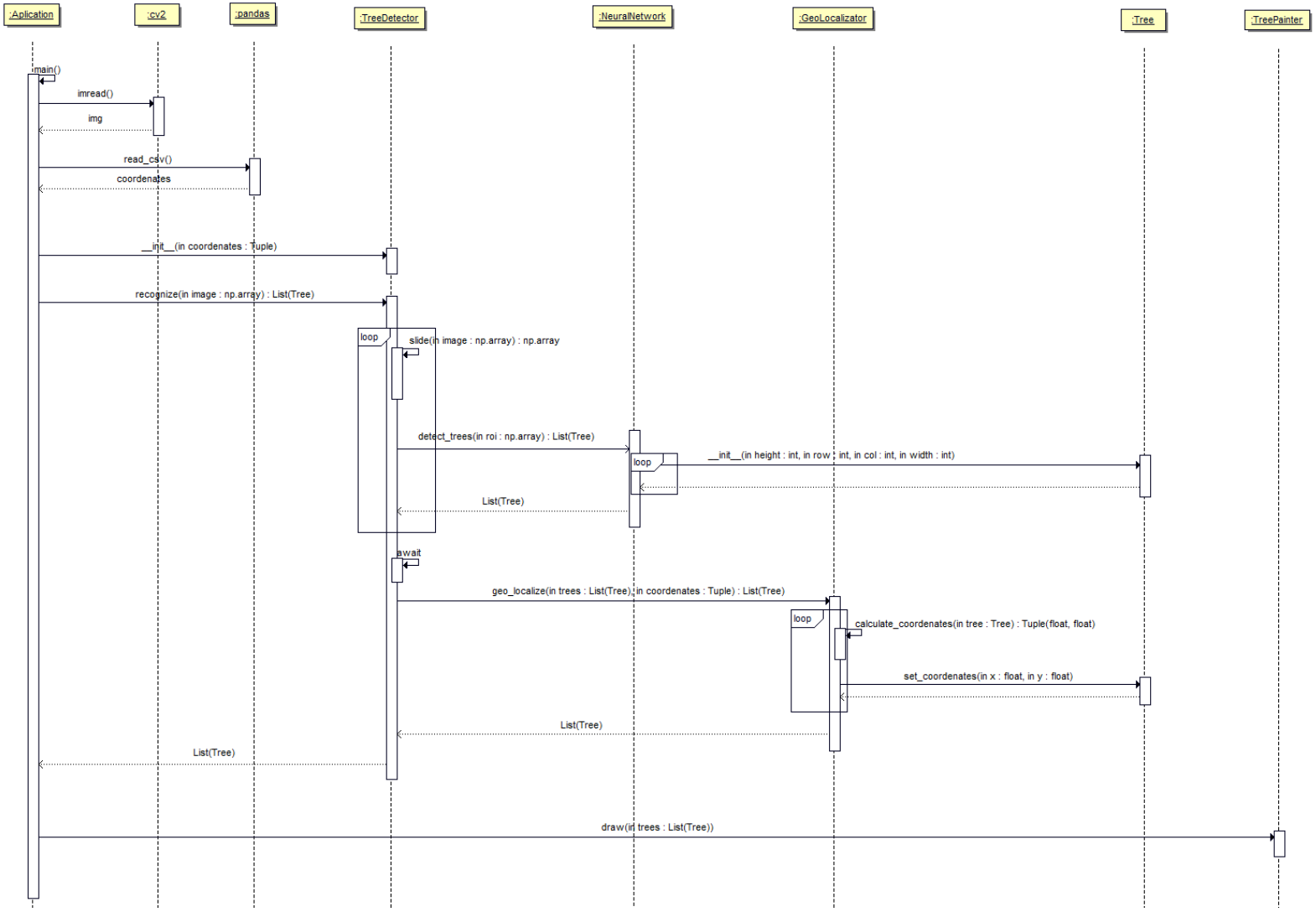

Figura 7. Diagrama de secuencias

2.3.1. Descripción del Diagrama de Secuencia

Al ejecutarse el método *main* de la clase *Aplication*, lo primero que se hace es obtener la imagen, y sus coordenadas, sobre la que desea trabajar. Una vez obtenida esta información general, A*plication* crea un objeto de la clase *TreeDetector* el cual necesita las coordenadas límites de la imagen para realizar el trabajo correctamente. En la siguiente imagen se muestra un ejemplo de una posible sección georreferenciada.

Figura 8. Ejemplo de sección georreferenciada en un mapa

Seguidamente, se ejecuta el método *recognize* de *TreeDetector* sobre la imagen que se desea procesar. Dicho método realizará un barrido por la imagen (*slice*) sobre el cual se obtendrán sub-imágenes. En la siguiente imagen se ve una sub-imagen de una imagen original.

Figura 9. Ejemplo de sub-imagen obtenida de la imagen completa aérea

Dicha sub-imagen es analizada por el método *detect_trees* de la clase *NeuralNetwork*, la que se encarga de devolver, de forma asíncrona, un listado con todos los objetos *Tree* que se han detectado. Para ello, sobre cada bounding box que detecta, se crea un objeto de la clase *Tree*.

Una vez que se ha terminado de procesar la imagen y se tiene el listado completo de todos los árboles, estos se procesarán junto con la información de la geolocalización de la imagen, por el método

geo_localize de la clase *GeoLocalizator*, para obtener las coordenadas de los árboles en el sistema de referencia WGS84.

De esta forma, el método *recognize* de *TreeDetector* devolverá un listado de todos los árboles que se han encontrado en la imagen original con los datos que se requería.

Finalmente, se representan las cajas/círculos donde se encuentran los árboles con el método *draw* de la clase *TreePainter.*