



## Informe Técnico - Económico

### Taller de Proyectos II

Autores:

Inés Varona Peña, David Manso Fernández, Óscar Martín Casares y Daniel Sirgo Rodríguez

Universidad de Valladolid  
Valladolid, España

8 de mayo de 2023

# Índice general

<b>1. Introducción</b>	<b>4</b>
<b>2. Planteamiento inicial</b>	<b>5</b>
<b>3. Solución</b>	<b>6</b>
3.1. Infraestructura . . . . .	6
3.2. Detección de señales de tráfico . . . . .	6
3.3. Normativa . . . . .	6
<b>4. Costes</b>	<b>7</b>
<b>5. Rendimiento: Validación del servicio</b>	<b>8</b>
<b>6. Anexo I: Manual del desarrollador</b>	<b>9</b>
6.1. Infraestructura de la red . . . . .	9
6.2. Despliegue IA . . . . .	10
6.2.1. Detección y seguimiento de señales . . . . .	10
6.2.2. Herramienta de etiquetado . . . . .	13
6.2.3. Medición de rendimiento . . . . .	16
6.2.4. Creación automática de datasets . . . . .	17
<b>7. Anexo II: Planificación</b>	<b>19</b>
<b>8. Referencias</b>	<b>20</b>

# Índice de figuras

2.1. Mapa de la autovía A-62 y las torres de comunicación existentes . . . . .	5
6.1. Detección de una señal . . . . .	11
6.2. Detección de una señal real capturada por nosotros . . . . .	12
6.3. Detección de una señal captada por la cámara . . . . .	12
6.4. Ejemplo de rendimiento con $medirRendimientoRed = True$ . . . . .	13
6.5. Etiquetado de una señal . . . . .	14
6.6. Selección del modelo . . . . .	15
6.7. Cuadros delimitadores . . . . .	15
6.8. Fichero con la información de etiquetado . . . . .	16
6.9. Medida de rendimiento . . . . .	17
6.10. Dataset descargado con los ficheros TXT . . . . .	18

# Índice de tablas

# Capítulo 1

## Introducción

!!!!!! REVISARRRRR !!!!

La empresa para la que trabajamos ha sido adjudicataria de un contrato con el objetivo de desarrollar un servicio de vehículo conectado consistente en el envío de información de video desde el vehículo a un servidor en el cloud. El video será procesado en el cloud para la detección automática, mediante técnicas de inteligencia artificial, de señales de tráfico en la carretera.

El objetivo de este proyecto es desarrollar una infraestructura de red 4G que permita la transmisión de video desde los vehículos conectados al servidor en el cloud. Esta infraestructura permitirá a la empresa ofrecer un servicio de predicción de señales de tráfico en la carretera, lo que ayudará a mejorar la seguridad y la eficiencia en el tráfico. Esto se logrará desarrollando un demostrador previo al despliegue de la red para probar la capacidad de la empresa para desplegar el servicio, y presentando una memoria técnicoeconómica que detalle el despliegue de la red en el tramo de autovía A-62 desde el kilómetro 158 hasta el kilómetro 231. Además, la empresa tendrá que demostrar contar con los permisos legales pertinentes para el despliegue del servicio, que tendrá una duración máxima de 6 meses.

Los componentes que la empresa pone a nuestra disposición son herramientas esenciales para el desarrollo de proyectos en infraestructura de telecomunicaciones y en inteligencia artificial.

En primer lugar, contamos con ordenadores de sobremesa y memorias USB, que nos permiten trabajar en el desarrollo de algoritmos de inteligencia artificial y en la programación de los módems Huawei y las tarjetas SIM de Vodafone S.A.U. para la emulación de la red 4G.

Asimismo, disponemos de un sistema SDR (Software Defined Radio) BladeRF 2.0 xa9, que es un dispositivo que permite la recepción y transmisión de señales de radio en un amplio rango de frecuencias. Este sistema se controla mediante el software abierto srsRAN y OpenAirInterface, que nos permiten configurar y gestionar las comunicaciones.

En cuanto a las frecuencias de banda 7, que van de 2540 a 2550 MHz y de 2650 a 2670 MHz, las tenemos disponibles en el laboratorio de pruebas y en la zona de despliegue. Estas frecuencias se utilizan para probar y validar el funcionamiento de las comunicaciones.

Para emular la transmisión de vídeo, la empresa nos proporciona vehículos Azon DeepRacer Evo dirigidos por control remoto, junto con la información básica de su manejo. Estos vehículos nos permiten simular las condiciones reales de transmisión de vídeo en diferentes entornos.

Además, contamos con un conjunto modular de pistas y señales que nos permiten montar diversos circuitos para probar y validar diferentes escenarios de comunicación.

Por último, la empresa nos proporciona bibliotecas Python para aprendizaje automático y recursos de computación en la nube para ejecutar los algoritmos de aprendizaje automático. Estas herramientas nos permiten desarrollar y probar modelos de inteligencia artificial para mejorar la eficiencia y la seguridad de las comunicaciones.

## Capítulo 2

# Planteamiento inicial

### Plantramiento Inicial

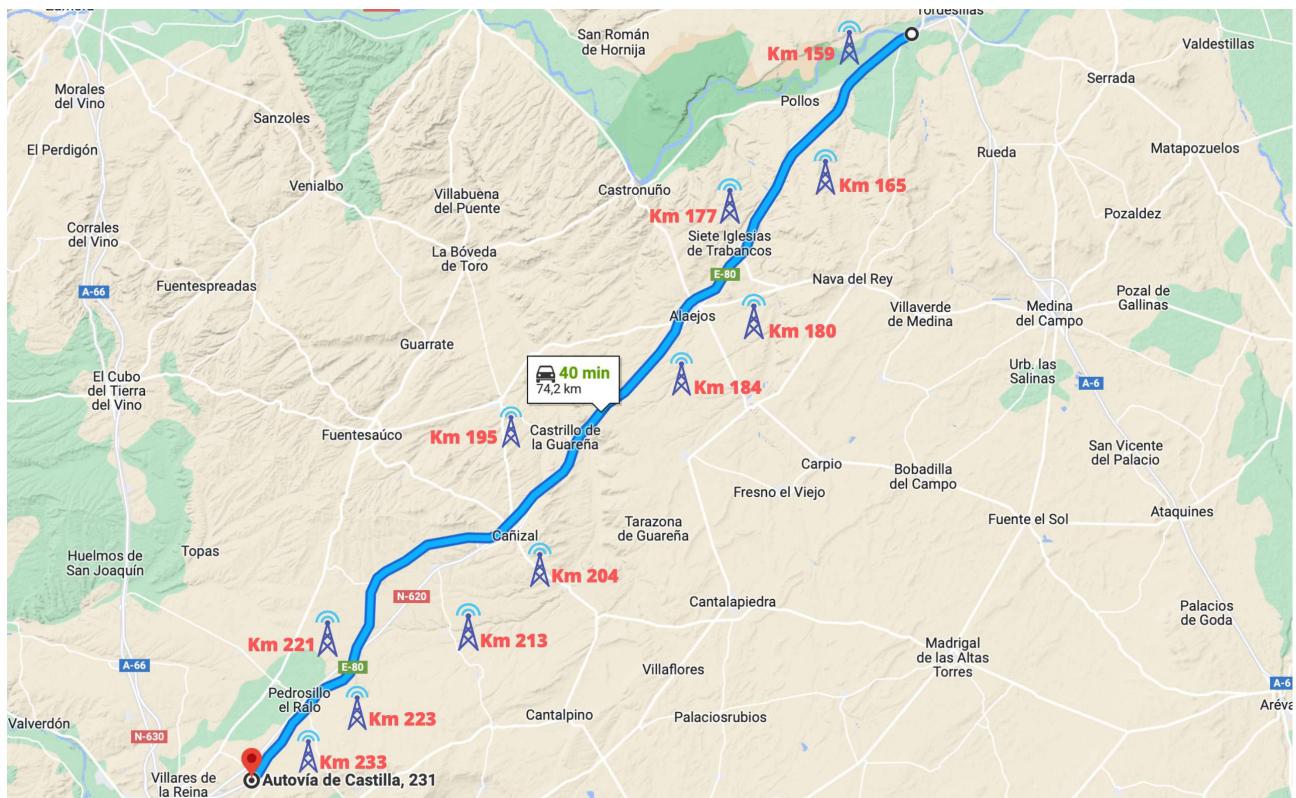


Figura 2.1: Mapa de la autovía A-62 y las torres de comunicación existentes

# Capítulo 3

## Solución

### 3.1. Infraestructura

Hablar de manera resumida y sencilla la forma en la que se va a implementar la infraestructura

### 3.2. Detección de señales de tráfico

Hablar de manera resumida y sencilla la forma en la que se va a implementar la detección de señales

### 3.3. Normativa

Para obtener los permisos necesarios para la instalación y operación de una infraestructura de red 4G mediante una SDR, es necesario seguir los siguientes pasos:

Pago de tasas, modelo 790, importe de 70,53€ que se pagan en efectivo. Con eso te dan un justificante, que se adjunta en el informe xml

junto al informe, se adjunta la declaración de no inhabilitación de competencia profesional y datos adicionales no estructurados del proyecto para solicitudes de título habilitante para el uso del dominio público radioeléctrico, servicio móvil y fijo de banda estrecha.

El modelo 990, (**FALTA MIRAR**) que es el coste de continuación, o pago recurrente.

# **Capítulo 4**

## **Costes**

## **Capítulo 5**

# **Rendimiento: Validación del servicio**

# Capítulo 6

## Anexo I: Manual del desarrollador

### 6.1. Infraestructura de la red

En el proceso de creación de la red 4G, se siguieron los siguientes pasos:

1. Descarga de la última versión de **Ubuntu, 22.04 LTS**.
2. Grabación de la ISO en un pincho a través de **UNetbootin** y posterior instalación de Linux en el ordenador.
3. Instalación de los drivers de la *Blade* siguiendo el manual de *GitHub*:

```
https://github.com/Nuand/bladerf/wiki/Getting-Started:-Linux
sudo add-apt-repository ppa:nuandllc/bladerf
sudo apt-get update
sudo apt-get install bladerf
```

4. Instalación de **srsran** siguiendo el manual de GitHub y las librerías **boost** y **libboost**:

```
https://docs.srsran.com/projects/4g/en/latest/general/source/1\_installation.html
https://docs.srsran.com/projects/4g/en/latest/getting\_started.html
git clone https://github.com/srsRAN/srsRAN_4G.git
cd srsRAN_4G
mkdir build
cd build
cmake ../
make
make test
sudo make install
srsran_4g_install_configs.sh user
```

5. Modificación de los archivos de configuración que se encuentran en la ruta *root/.config/srsran/*:

- **epc.conf**
- **enb.conf**
- **user\_db.csv**

En el archivo **enb.conf** se cambiaron los valores de **MCC** y **MNC** que están disponibles en la SIM, y se establecieron los valores correspondientes al **MCC** y **MNC** utilizados en el proyecto (**901-70**) para que se correspondan con el IMSI de las SIM. También se modificó el **dl\_earfcn** utilizando una página web y se estableció el ancho de banda **n\_prb** en **50**.

En el archivo **epc.conf** se modificaron los valores de **MCC** y **MNC** y se añadieron los nombres de la red con:

```
full_net_name= NOMBRE
short_net_name= NOMBRE
```

En el archivo **user\_db.csv** se creó un usuario nuevo con la siguiente información:

```
nombre, mil (Auth), IMSI (aparece en las hojas de las sims),
KEY (aparece en las hojas de las sims), opc,
OPC(aparece en las hojas de las sims), 9000,
sqn (se pone automáticamente, pero pon números, por ejemplo todo a 0),
7 (QCI), dynamic (IP_alloc)
```

6. Para crear la red y que funcione, se siguieron los siguientes pasos:

```
srepc_if_masq.sh enp0s25  
srsep epc.conf  
srsenb enb.conf
```

7. Una vez obtenida la conexión a internet con la red 4G, nos bajamos los ficheros *python* de control del coche para crear el servidor cloud e instalamos el servidor **MQTT Mosquitto**.

## 6.2. Despliegue IA

En el presente documento se explicará cómo hacer uso de las diferentes herramientas que se han utilizado a lo largo del desarrollo del sistema inteligente de detección de señales. Destacar que el desarrollo ha sido llevado a cabo en un ordenador con sistema operativo *MAC OS/Linux*, es decir, en caso de trabajar con un ordenador *Windows* habrá que prestar especial al código. Tendrás que modificar las líneas en las que se hace referencia a directorios o ficheros, ya que se hace de forma diferente en dichos sistemas operativos, en *MAC OS/Linux* se hace con ‘/’ y en *Windows* con ‘\’. Asimismo, los directorios están referenciados respecto a nuestro ordenador personal, por lo que habrá que adecuarlos al tuyo propio.

Para hacer uso del sistema, debemos descargarnos el proyecto de nuestro repositorio de *GitHub*, suponiendo que tenemos la herramienta de línea de comandos de *git* instalada, podemos hacer:

```
git clone https://github.com/OscarMartinn/TallerDeProyectos2.git
```

En primer lugar, para poder trabajar con todo proyecto debemos crear nuestro propio entorno virtual en el cual instalaremos todas las librerías necesarias para ejecutar el código.

```
python3 -m venv 'nombre_entorno'
```

Una vez creado el entorno debemos activarlo, por ejemplo, en un ordenador **MAC**:

```
source nombre_entorno/bin/activate
```

Todas las librerías necesarias junto con sus versiones concretas se han guardado en el fichero *requirements.txt*, por ello instalaremos automáticamente cada una de ellas:

```
pip install -r requirements.txt
```

Tras realizar todos los pasos anteriores ya nos encontramos en disposición de ejecutar todos los algoritmos. Desglosaremos el proyecto en tres partes distintas:

1. Detección y seguimiento de señales
2. Herramienta de etiquetado
3. Medición del rendimiento
4. Creación automática de datasets.

### 6.2.1. Detección y seguimiento de señales

Accediendo a la primera carpeta **1\_YOLOV3** nos encontraremos con todos los scripts encargados de ejecutar el algoritmo. Tenemos cinco ficheros **Python**:

1. **imagen\_yolov3.py**: script encargado de detección de señales en una imagen individual.
2. **video\_yolov3.py**: script encargado de detección de señales en un video.
3. **camara\_yolov3.py**: script encargado de detección de señales a través de la cámara frontal o webcam del ordenador.
4. **automatic\_imagen\_yolov3.py**: script encargado de detección de señales en una imagen individual, pero el nombre de la imagen a procesar se introducirá mediante línea de comandos y no dentro del fichero.
5. **automatizacion\_imagenes.py**: script encargado de leer todas las imágenes de un directorio y mandárselas a través de línea de comandos al fichero *automatic\_imagen\_yolov3.py* para poder procesar varias imágenes ininterrumpidamente.

Por la propia estructura de **YOLOV3**, todos los scripts encargados de la detección de señales se nutren de tres ficheros básicos, los que se encuentran dentro de la carpeta **yolo\_data**:

- **classes.names**: fichero que contiene todas las clases de señales con las que ha sido entrenado el modelo y que va a ser capaz de detectar.
- **yolov3\_ts\_test.cfg**: fichero que contiene la configuración de **YOLO**.
- **yolov3\_ts.weights**: fichero que contiene los pesos obtenidos como resultado del entrenamiento del modelo.

En el script **imagen\_yolov3.py** deberemos modificar la variable *imageName* con el nombre de la imagen en formato JPG, que ha de encontrarse dentro del directorio destinado a las imágenes *images*. Con el siguiente comando podemos visualizar un ejemplo, figura 6.1

```
python3 imagen_yolov3.py
```

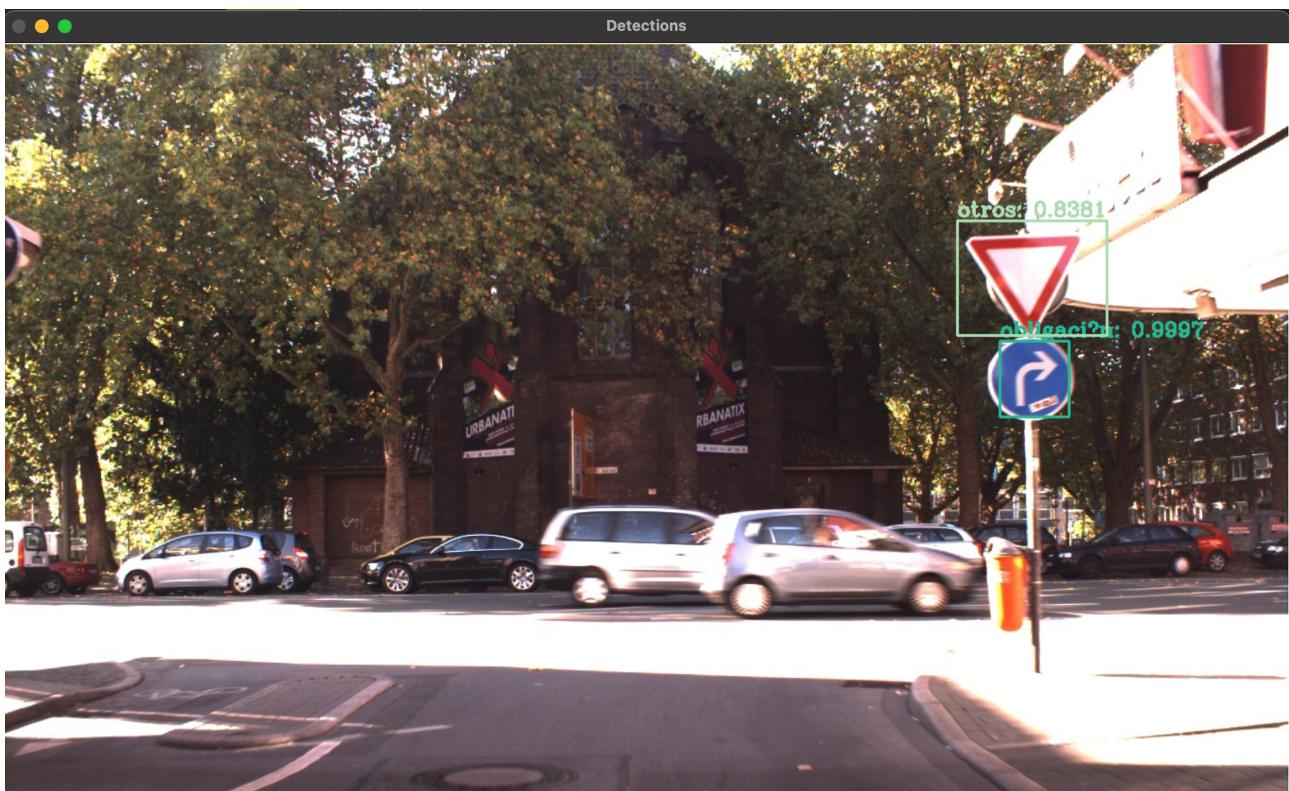


Figura 6.1: Detección de una señal

En caso de querer realizar el procesamiento de un video, lo haremos con el script **yolo-3-video.py**. De igual manera debemos modificar la variable *videoName* con el nombre del video, el cual ha de encontrarse dentro de la carpeta *videos* en formato MP4. Lo pondremos en marcha mediante, obteniendo como resultado la figura 6.2

```
python3 video_yolov3.py
```

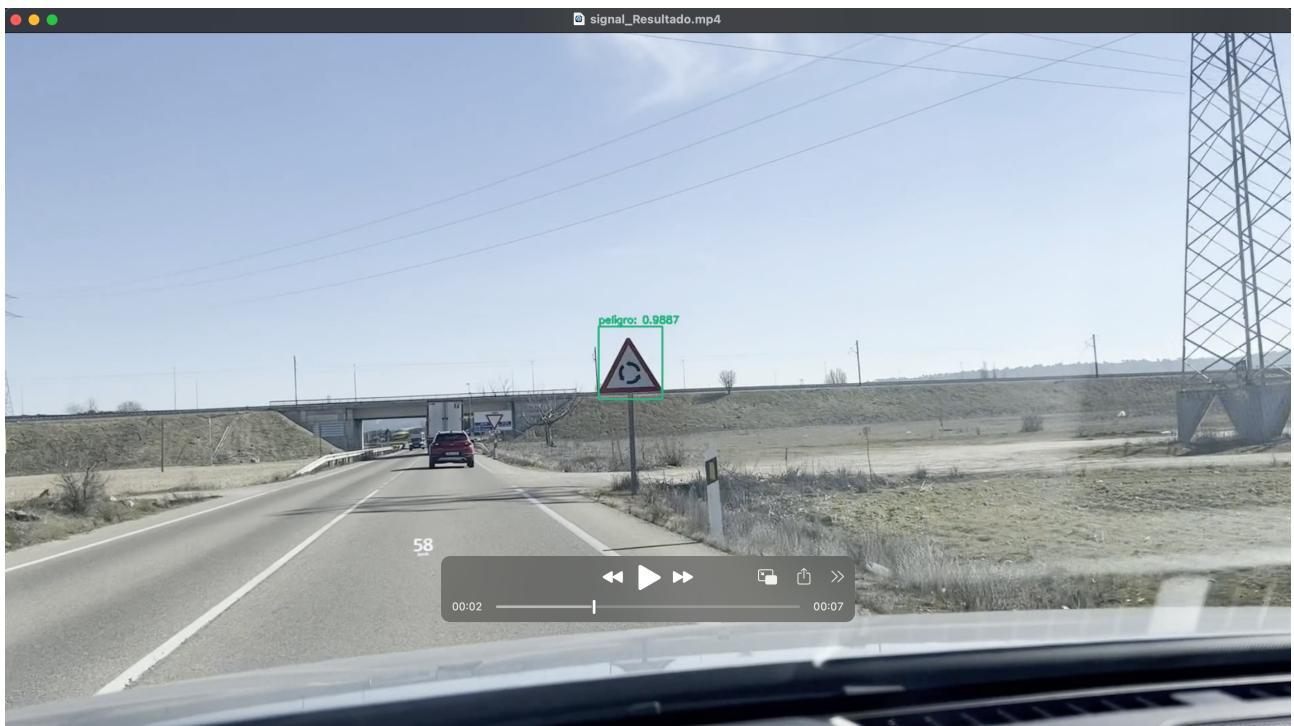


Figura 6.2: Detección de una señal real capturada por nosotros

Asimismo, podremos procesar video en tiempo real procedente de la cámara o webcam de nuestro propio ordenador, figura 6.3. A través del script **camara\_yolov3.py** podremos ponerlo en marcha:

```
python3 camara_yolov3.py
```



Figura 6.3: Detección de una señal captada por la cámara

Puede ser de utilidad poder procesar muchas imágenes de manera conjunta, por ejemplo, si quisieramos medir el rendimiento de la red. Para ello, disponemos de dos scripts que funcionan de manera conjunta, estos son **automatic\_imagen\_yolov3.py** y **automatizacion\_imagenes.py**.

El script que deberemos ejecutar es **automatizacion\_imagenes.py**, el cual leerá cuáles son las imágenes que se encuentran en el directorio especificado en la variable *directorioAutomatic* y ejecutará individualmente **automatic\_imagen\_yolov3.py** con el nombre de cada imagen como parámetro de entrada.

Dado que nosotros hemos utilizado dichos scripts para hacernos más sencilla la tarea de medición de rendimiento de la red, tendremos la posibilidad de crear un fichero **nombre\_imagen.txt** por cada imagen, que contendrá las coordenadas del cuadro delimitador del objeto detectado y la precisión obtenida. Mediante la variable que se encuentra al comiendo del fichero **automatic\_imagen\_yolov3.py** llamada *medirRendimientoRed* podremos controlar dos modos de operación:

- Si *medirRendimientoRed* es **False**: su funcionamiento será análogo al script **imagen\_yolov3.py**, pero podremos visualizar varias imágenes de seguido, simplemente deberemos pulsar cualquier tecla para visualizar la siguiente.
- Si *medirRendimientoRed* es **True**: podremos crear el fichero **nombre\_imagen.txt** con su información correspondiente dentro del directorio *detections* para cada una de las imágenes, pero no iremos viendo en tiempo real el procesado.

Mediante la siguiente instrucción podremos ejecutarlo:

```
python3 automatizacion_imagenes.py
```

Si además tuviéramos la opción de *medirRendimientoRed* establecida a **True**, obtendremos un resultado similar al de la figura 6.4.

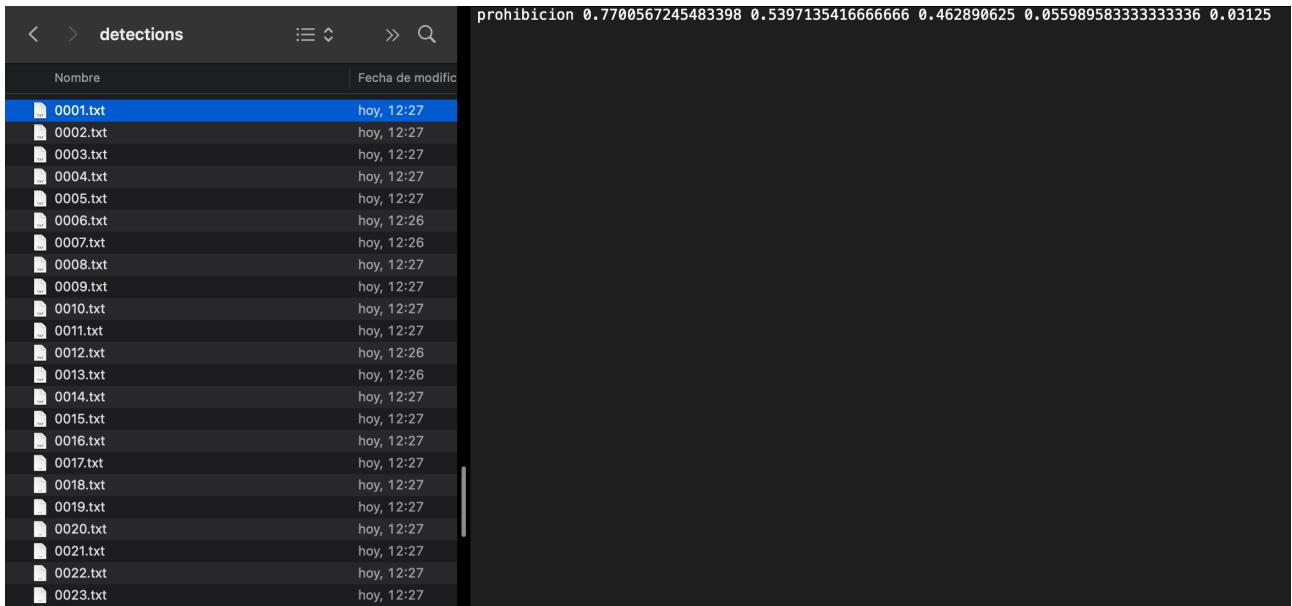


Figura 6.4: Ejemplo de rendimiento con *medirRendimientoRed = True*

### 6.2.2. Herramienta de etiquetado

Existen numerosas herramientas de etiquetado compatibles con **YOLO**, pero quizás una de las más sencillas de usar sea *LabelIMG*. Esta herramienta se encuentra disponible tanto para *Windows* como para *MAC OS/Linux*.

Para poder acceder a *LabelIMG* se puede hacer a través de su propio repositorio de *GitHub* <https://github.com/hear tex labs/labelImg>, el cual presenta las distintas opciones de instalación que se tienen dependiendo de la plataforma. En caso de necesitar instalarla en un ordenador *MAC OS/Linux*, debido a las diferentes incompatibilidades entre librerías con las que nos encontramos en su momento, recomendamos instalar las que se indican en el fichero de **requirements.txt**.

Para hacer uso de dicha herramienta simplemente debemos inicializar su fichero base, el cual lanzará una interfaz con la que interactuaremos para realizar el etiquetado. Para ello, accediendo a la segunda carpeta de nuestro repositorio denominada *2\_Etiquetado*, podremos arrancarla:

```
python3 labelImg.py
```

Internamente podemos modificar cuáles queremos que sean las clases que por defecto tenemos para etiquetar, en nuestro caso tenemos las diferentes clases de señales: prohibición, peligro, obligación y otros. Si se quisiera modificarlo porque se fuera a utilizar para otra aplicación, se podría modificar mediante el fichero **predefined-classes.txt** disponible dentro de la carpeta **data**.

Se puede trabajar con una imagen individual o con un conjunto de ellas, a través de los botones indicados a continuación podremos abrir las imágenes:

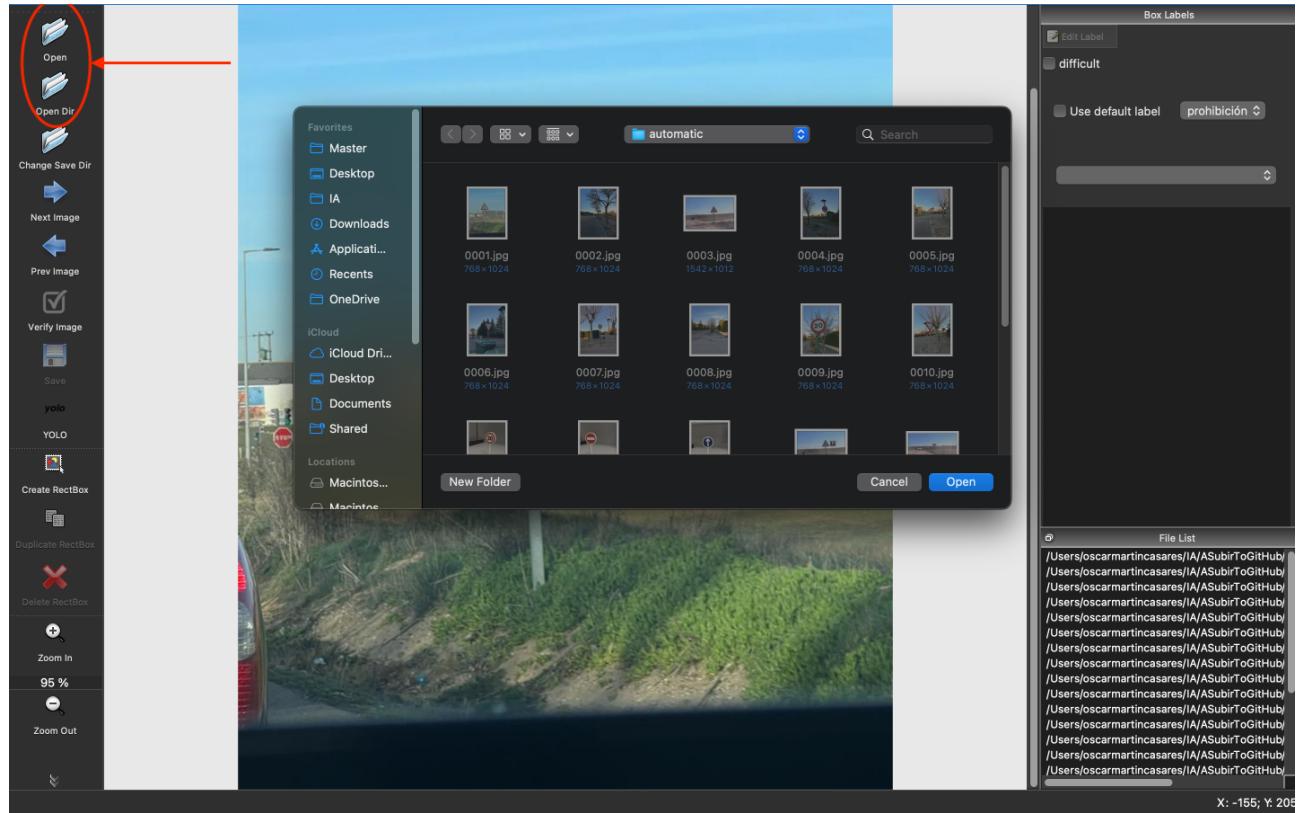


Figura 6.5: Etiquetado de una señal

Debemos asegurarnos de que el formato en el que se va a producir el etiquetado debe ser únicamente **YOLO** (ver figura 6.6). Pulsando sobre el icono mostrado podremos ir intercambiando entre diferentes formatos, ya que esta herramienta es compatible con varios.



Figura 6.6: Selección del modelo

Y mediante la opción *Create RectBox* podremos crear los cuadros delimitadores o *bounding boxes* indicando de qué tipo de señal se trata. Ver en la figura 6.7



Figura 6.7: Cuadros delimitadores

Al guardar la imagen se nos creará un fichero en formato **TXT** que contendrá la información de etiquetado (Figura 6.8) en el directorio que contenga la imagen en cuestión, con idéntico formato **YOLO** a como se nos mostraba en detección con la opción *medirRendimientoRed* a **True**.

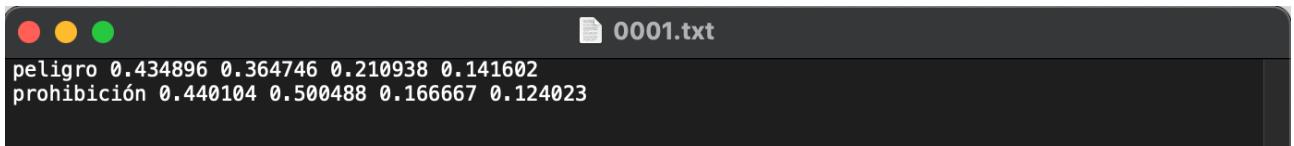


Figura 6.8: Fichero con la información de etiquetado

Además, creemos de puede ser de utilidad una herramienta que convierta un video a *frames*, para poder etiquetarlos de manera manual para entrenar o cualquier otra aplicación. Esta herramienta se llama **ffmpeg** <https://ffmpeg.org> y se puede instalar de manera muy sencilla mediante el comando:

```
pip3 install ffmpeg
```

Simplemente desde línea de comandos podremos utilizarla, indicando cuál es el video que queremos dividir, en cuántos *frames* queremos dividirlo y cómo queremos que se llamen cada una de las imágenes.

```
ffmpeg -i nombre_video.mp4 -vf fps=4 nombre_imagen-%d.jpeg
```

### 6.2.3. Medición de rendimiento

En cuestiones de medición del rendimiento, existe un repositorio de *GitHub* muy popular utilizado por la mayoría de la gente que busca medir el rendimiento de su modelo de inteligencia artificial. Este repositorio es <https://github.com/rafaelpadilla/Object-Detection-Metrics.git>, posee un fichero **README.md** de vital importancia, en el que se explica todas las métricas que podemos obtener, su explicación teórica y cómo obtenerlas. Asimismo, proporciona dos ejemplos guiados para poder realizar pruebas sencillas. En nuestro proyecto se encuentra dentro de la tercera carpeta *3\_Rendimiento*.

Sin embargo, nosotros hemos realizado nuestro propio script **precisionVSrecall.py** para poder obtener la curva de Precisión vs Recuperación (*Precision vs Recall*) que se encuentra dentro del directorio *rendimiento*. Mediante esta curva podremos evaluar el modelo. La precisión se refiere a la proporción de verdaderos positivos (TP) y falsos positivos (FP). La recuperación se refiere a la proporción de verdaderos positivos entre la suma de verdaderos positivos con falsos negativos. En definitiva, sirve para evaluar la calidad de un modelo de clasificación y se puede utilizar para determinar el umbral óptimo para el modelo.

La idea principal para obtener dicha curva es comparar la clase y cuadros de delimitadores de numerosas fotos etiquetadas y procesadas por el algoritmo de detección. Es decir, para una imagen comparar la señal real con la detectada. Se deben introducir todos los ficheros **TXT** con los datos de etiquetado en el interior de **rendimiento/images/groundtruths/** y los ficheros **TXT** con los datos de detección en el directorio **rendimiento/images/detections/**. Ejecutando entonces el script **precisionVSrecall.py** podremos obtener la curva de rendimiento:

```
python3 precisionVSrecall.py
```

A modo de ejemplo, nosotros probamos con 64 imágenes y estos fueron los resultados que obtuvimos:

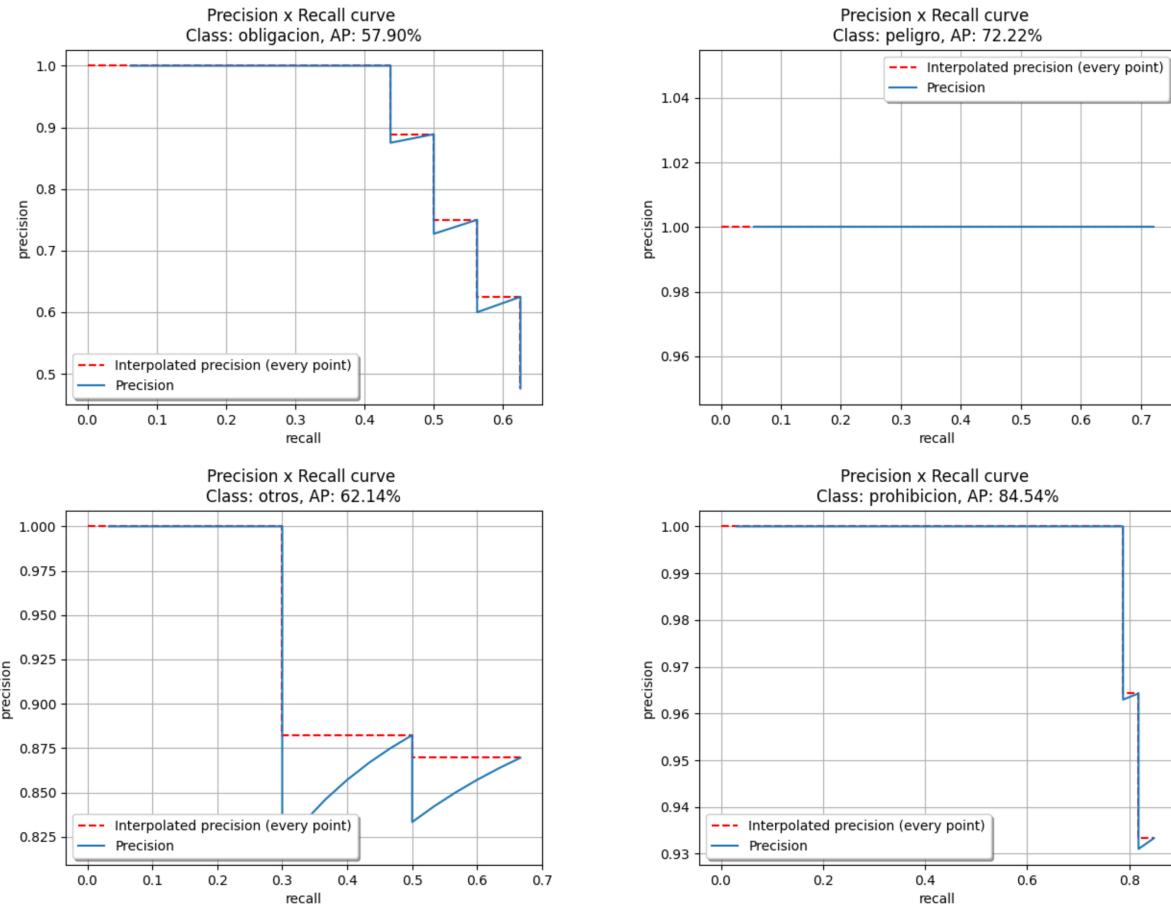


Figura 6.9: Medida de rendimiento

#### 6.2.4. Creación automática de datasets

Debido a la tarea de creación de datasets es muy tedioso, sobre todo por la tarea de etiquetado, aparte de buscar en Internet datasets realizados por terceros, podemos utilizar una herramienta que nos permite moldear uno a nuestro gusto. Esta herramienta se llama **OIDv4\_ToolKit** [https://github.com/EscVM/OIDv4\\_ToolKit.git](https://github.com/EscVM/OIDv4_ToolKit.git) y se encuentra en la cuarta carpeta *4-Crear\_Dataset*.

En el propio *README.md* de la herramienta se nos explica su funcionamiento, si por ejemplo quisieramos descargar un dataset que estuviera formado por 8 imágenes de coches y autobuses podríamos hacer:

```
python3 main.py downloader --classes Car Bus --type_csv train --multiclasses 1 --limit 8
```

En la figura 6.10 podemos observar como en la carpeta *OID/Dataset/train/Car\_Bus/* se nos han descargado las 8 imágenes, incluyendo sus ficheros TXT con la información de etiquetado en el interior de la carpeta *Label*:

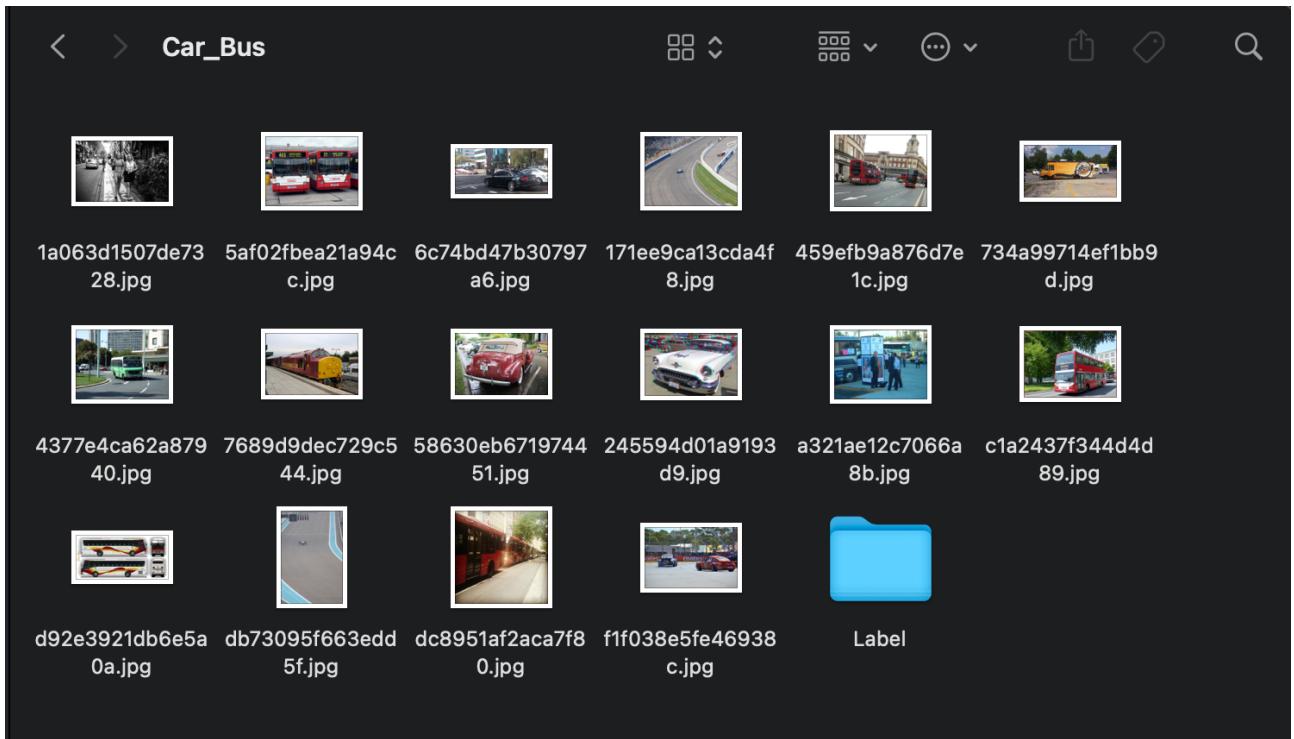


Figura 6.10: Dataset descargado con los ficheros TXT

Sin embargo, dichos ficheros con la información de etiquetado no se encuentran en formato **YOLO**, por lo que habrá que realizar la transformación. Dicha tarea se puede llevar a cabo mediante el script que se encuentra dentro de la herramienta **OIDv4\_ToolKit** llamado **convert\_to\_YOLO.py**.

En dicho *script* debemos cambiar introducir dos rutas, la que contiene las imágenes descargadas y en la que se encuentra el fichero CSV con todas las clases disponibles para descargar en la herramienta. Para obtener dichas rutas se pueden con un *script* tan sencillo como este:

```
import os
current_dir = os.path.dirname(os.path.abspath(__file__))
print(current_dir)
```

Tras ejecutar el *script* podremos observar como en el directorio en el que se encuentran las imágenes se han creado cada uno de los ficheros TXT con el mismo nombre con la información de etiquetado **YOLO**. Podríamos comprobar además si se ha realizado con éxito la transformación abriendo dicho directorio con la herramienta de etiquetado **LabelImg**.

## **Capítulo 7**

## **Anexo II: Planificación**

## Capítulo 8

# Referencias