



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

17 de mayo de 2023

Índice general

1. Introducción	4
2. Planteamiento inicial	5
3. Solución	7
3.1. Infraestructura	7
3.2. Detección de señales de tráfico	8
3.3. Normativa	14
3.3.1. Pago tasas del modelo 790	14
3.3.2. Envío de la solicitud	15
4. Costes	17
5. Rendimiento: Validación del servicio	18
6. Anexo I: Manual del desarrollador	19
6.1. Infraestructura de la red	19
6.2. Despliegue IA	20
6.2.1. Detección y seguimiento de señales	20
6.2.2. Herramienta de etiquetado	23
6.2.3. Medición de rendimiento	26
6.2.4. Creación automática de datasets	27
7. Anexo II: Planificación	29
8. Referencias	30

Índice de figuras

2.1. Mapa de la autovía A-62 y las torres de comunicación existentes	5
3.1. Vehículo Amazon DeepRacer utilizado	10
3.2. Primer ejemplo de detección de señales miniatura en vídeo en circuito	11
3.3. Segundo ejemplo de detección de señales miniatura en vídeo en circuito	12
3.4. Detección de señales reales en vídeo en carretera real	13
3.5. Gráficas del rendimiento obtenido	14
6.1. Detección de una señal	21
6.2. Detección de una señal real capturada por nosotros	22
6.3. Detección de una señal captada por la cámara	22
6.4. Ejemplo de rendimiento con <i>medirRendimientoRed = True</i>	23
6.5. Etiquetado de una señal	24
6.6. Selección del modelo	25
6.7. Cuadros delimitadores	25
6.8. Fichero con la información de etiquetado	26
6.9. Medida de rendimiento	27
6.10. Dataset descargado con los ficheros TXT	28

Índice de tablas

3.1. Cobertura en función de la modulación.	8
3.2. Capacidades del enlace de subida y bajada.	8

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

Partimos de la base que somos una empresa a la que se la ha adjudicado el despliegue de infraestructura de red 4G, con el objetivo de prestar servicio a vehículos conectados. Dicho servicio queremos que se implemente en la autovía que conecta la ciudad de Salamanca con Tordesillas, en Castilla y León.

El tramo de carretera seleccionado comprende 73 kilómetros, en concreto desde el kilómetro 158 al kilómetro 231 de la A-62. En dicho tramo nos encontramos con que ya existe una buena infraestructura de red 4G prestando servicio, por ello vamos a aprovechar parte de ella. En concreto, haremos uso de las torres de telefonía ya disponibles, disponemos de 11 a lo largo de la carretera, tal y como podemos observar en la figura 2.1. En las torres que sean necesarias, que no tienen por qué ser todas, instalaremos nuestros propios equipos, como estaciones base y antenas. Es importante evaluar cuidadosamente los costos asociados a los componentes y considerar opciones con precios competitivos.

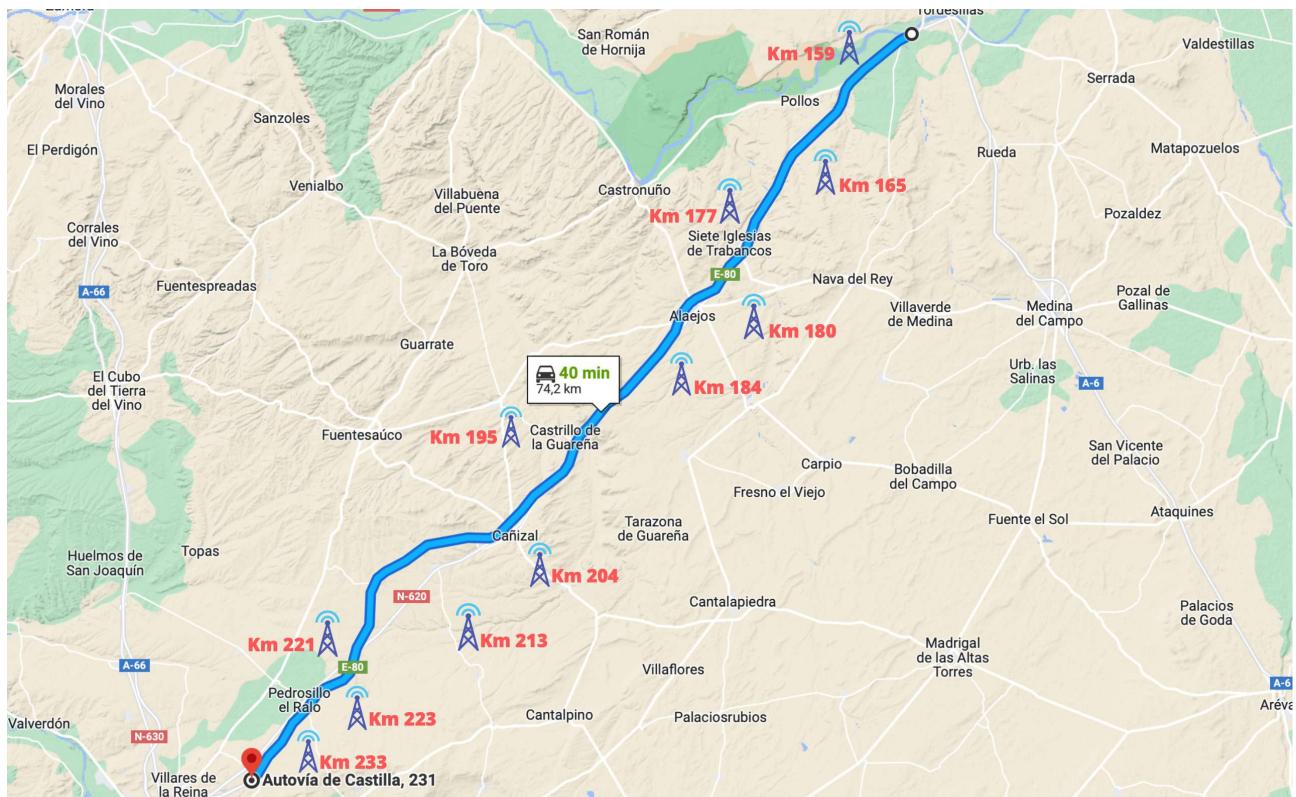


Figura 2.1: Mapa de la autovía A-62 y las torres de comunicación existentes

Queremos que un número elevado de coches pueda hacer uso de dicho servicio de manera ininterrumpida, es decir, se requiere que cada coche tenga capacidad suficiente para enviar el vídeo captado por la cámara durante todo el recorrido. Por ello, es necesario disponer de un ancho de banda de al menos varios Mbps dependiendo de la calidad del vídeo.

Prestando este servicio, los vehículos de los usuarios tendrán la capacidad de detectar las señales de tráfico aumentando así la seguridad vial, ya que así los vehículos pueden adaptar su velocidad y comportamiento en la carretera, lo que puede reducir el riesgo de accidentes y mejorar la seguridad vial en general. De igual forma, puede fomentar el cumplimiento de las normativas y regulaciones de tráfico, reduciendo así la cantidad de sanciones, o puede ayudar a optimizar el consumo de combustible y reducir las emisiones de gases contaminantes.

La seguridad en las carreteras es uno de los temas más importantes en el mundo actual. Una de las medidas para garantizar la seguridad es la señalización vial, la cual indica a los conductores las condiciones y normas de circulación. Sin embargo, a menudo las señales pueden ser difíciles de identificar debido a factores como su ubicación o la distracción del conductor.

Para resolver este problema, se puede utilizar el aprendizaje automático para diseñar modelos capaces de detectar de manera precisa y eficiente las señales de tráfico. Estos modelos pueden ayudar a mejorar la seguridad en las carreteras al permitir que los conductores reciban información clara y precisa en todo momento.

En este contexto, nuestra empresa ha sido encargada de diseñar e implementar un modelo de aprendizaje automático para la detección de señales de tráfico. Para ello, debemos utilizar técnicas avanzadas de procesamiento de imágenes. El objetivo final es contribuir a la seguridad en las carreteras y ofrecer soluciones innovadoras y eficaces.

Con el objetivo de poder prestar dicho servicio, se debe elegir qué modelo de inteligencia artificial se va a seleccionar para llevarlo a cabo. En el campo de la detección de objetos en imágenes y videos, existen varios modelos de aprendizaje automático que se pueden utilizar. Sin embargo, uno de los más avanzados y precisos es YOLO. Lo que lo distingue de otros modelos es su capacidad para detectar múltiples objetos en una sola imagen de manera eficiente y precisa, utilizando redes neuronales convolucionales (CNN) y técnicas de filtrado de cajas delimitadoras.

Capítulo 3

Solución

3.1. Infraestructura

En primera instancia, se debe decidir cómo se quiere plantear la infraestructura de red, si vamos a construir nuestras propias torres o no, si vamos a funcionar como una operadora virtual o si por el contrario vamos a comprar e instalar nuestros propios equipos. En nuestro caso, nos hemos decantado por alquilar las torres de telecomunicaciones existentes, en las cuales vamos a instalar nuestros propios componentes. De esta manera, lograremos diseñar una red que cumple con compromisos económicos y que a la vez nos proporcionan cierto control en caso de averías o problemas.

Luego, debemos conocer dónde se encuentran las torres de telecomunicaciones existentes. Existen varias páginas web a través de las cuales podemos conocer dicha información, como Infoantenas <https://geoportal.minetur.gob.es/VCTEL/vcne.do>, un servicio desarrollado por parte del Ministerio de Asuntos Económicos y Transformación Digital, o AntenasGSM <https://antenasgsm.com>. A través de dichas páginas hemos podido conocer la localización de las torres, mostradas en la figura 2.1.

Sin embargo, tras seleccionar el modelo de estación base y antena que vamos a utilizar, en función de sus distintas características técnicas hemos propuesto un diseño de infraestructura, en la cual no va a ser necesario utilizar todas las torres de telecomunicaciones disponibles. Existen 11 torres desplegadas y utilizaremos 7 de ellas. En origen las distintas torres pertenecían a las propias empresas operadoras que prestan el propio servicio, o empresas filiales de las mismas. No obstante, la mayoría de ellas han vendido una gran cantidad de las torres a otras empresas, como pueden ser Cellnex Telecom, una empresa española, o American Tower Corporation, empresa estadounidense, ambas se dedican a la construcción y gestión de infraestructuras de telecomunicaciones. [Ref: Telefónica vende a ATC las torres de Telxius por 7.700 millones de euros — Empresas (elmundo.es) y Orange vende 1.500 torres a Cellnex para usarlas en régimen de alquiler — Empresas (elmundo.es)]. Se estima que el alquiler de estas torres se encuentra en torno a los 4.000 y 20.000 euros en zonas urbanas, y 1.000 y 15.000 euros en zonas rurales, costo que habrá que tener en cuenta a la hora de presupuestar la infraestructura.

Para el diseño del proyecto se ha contactado con los principales distribuidores en España de estos componentes, como pueden ser Ericsson, Nokia, Kathrein o Moyano Telsa, con el objetivo de lograr una red lo más similar a lo que nos podemos encontrar en las grandes operadoras españolas. Sin embargo, ninguna de estas empresas nos ha respondido ni nos ha facilitado un catálogo de sus productos, por ello, hemos procedido a diseñar la red con equipos de los cuales si hemos podido encontrar información.

En primer lugar, hemos seleccionado una estación base proporcionada por una empresa china llamada Bai-cells, una compañía fundada en 2014 con sede en cinco continentes que ofrece productos de tecnología inalámbrica 4G y 5G. El modelo concreto es el Nova846, éste nos proporcionará una potencia de transmisión entre 0 y 37 dBm, una distancia máxima de 60km y una sensibilidad diferente en función del rango de distancias en el que se encuentra el usuario gracias a la modulación adaptativa.

Esquema de modulación	RSRP [dBm]	Distancia cubierta [km]
QPSK	-120 < RSRP < -110	Entre 40 y 60
16 QAM	-110 < RSRP < -100	Entre 10 y 40
64 QAM	-100 < RSRP < -85	Entre 4 y 10
256 QAM	RSRP > -85	Menor a 4

Tabla 3.1: Cobertura en función de la modulación.

Dependiendo de la configuración proporcionará un rendimiento u otro, en nuestro caso nos hemos decidido por la configuración 6. Ésta nos brindará un rendimiento máximo en el enlace de subida, que nos interesa que sea máximo para que el mayor número de usuarios pueda enviar el vídeo en streaming. Lograremos tener las distintas capacidades mostradas para el enlace de downlink y uplink.

DL 256 QAM	DL 64 QAM	DL 16 QAM	DL QPSK	UL 256 QAM	UL 64 QAM	UL 16 QAM	UL QPSK
348 Mbps	264 Mbps	70 Mbps	53 Mbps	92 Mbps	70 Mbps	53 Mbps	40 Mbps

Tabla 3.2: Capacidades del enlace de subida y bajada.

3.2. Detección de señales de tráfico

YOLO significa "You Only Look Once" <https://pjreddie.com/darknet/yolo/>. Se le puso ese nombre debido a que la red neuronal solo se aplica una vez a la imagen completa, lo que permite una detección rápida y precisa de objetos. Una de las ventajas clave de YOLO es que puede detectar múltiples objetos en una sola imagen, prediciendo las etiquetas de clase y las ubicaciones de los objetos al mismo tiempo. La red neuronal divide la imagen en regiones, predice cajas delimitadoras o bounding boxes y probabilidades para cada una de ellas, y filtra las cajas delimitadoras con una técnica llamada supresión no máxima para eliminar aquellas con baja confianza o superpuestas con otras de mayor confianza. [YOLOv3: An Incremental Improvement, Joseph Redmon, Ali Farhadi University of Washington].

El origen de YOLO se remonta al año 2015, fue desarrollado por Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhadi mientras trabajaban en la Universidad de Washington. La primera versión de YOLO se presentó en un artículo titulado "You Only Look Once: Unified, Real-Time Object Detection." en la conferencia de visión por ordenador CVPR (Computer Vision and Pattern Recognition) en 2016.

El éxito de YOLO se debió en gran parte a su capacidad para detectar objetos en tiempo real, con una velocidad de procesamiento mucho más rápida que otros algoritmos. Además, su precisión en la detección de objetos en imágenes fue muy alta en comparación con otros.

En 2016, Joseph Redmon fundó una empresa llamada "YOLO: You Only Look Once Inc." para desarrollar la tecnología de detección de objetos YOLO en un producto comercial. Sin embargo, en 2018, Redmon anunció que se retiraba de la investigación en inteligencia artificial debido a sus preocupaciones éticas sobre el uso de la tecnología de IA en aplicaciones militares.

A partir de entonces, la investigación y el desarrollo de YOLO fueron continuados por otros investigadores. Actualmente, hay cuatro versiones oficiales de YOLO: YOLOv1, YOLOv2, YOLOv3 y YOLOv4, donde cada versión ha mejorado en términos de precisión y velocidad de procesamiento, y ha incorporado nuevas técnicas de detección de objetos, como la eliminación de falsos positivos. [A Review of Yolo Algorithm Developments A Review of Yolo Algorithm Developments, Peiyuan Jiang, Daji Ergu*, Fangyao Liu, Ying Cai, Bo Ma, 2022].

En concreto, nosotros vamos a utilizar YOLO versión 3, la cual utiliza 53 capas convolucionales sucesivas. Seleccionamos esta versión ya que, debido al auge de estas tecnologías, quizás sea la versión con mayor documentación y ejemplos de los cuales aprender.

Lo normal cuando un particular se plantea desarrollar un proyecto de este tipo es trabajar a través de Google Colab o mediante sus propios recursos locales. Google Colab es una herramienta en línea que permite escribir, ejecutar y compartir código Python en un entorno de notebook en la nube con acceso a recursos hardware de alta calidad y herramientas de desarrollo preinstaladas. Sin embargo, a pesar de que nuestros ordenadores no están especialmente pensados para trabajar con IA, decidimos utilizarlos frente a la plataforma de Google. Utilizando nuestros propios recursos optamos por premiar la sencillez y facilidad de trabajo frente a la alternativa

de poseer un mejor hardware online.

Dado que estamos hablando de software abierto, existen numerosas versiones de cada YOLO hechas por particulares. No obstante, la tercera versión oficial de YOLO se nutre de tres ficheros de configuración, el fichero de clases, el fichero que contiene la configuración de YOLO y el fichero que contiene los pesos resultados del entrenamiento del modelo.

Por defecto, viene pre-entrenado con el conjunto de datos COCO (Common Objects in Context), que es un conjunto de datos de detección de objetos, que contiene más de 330.000 imágenes etiquetadas con más de 2,5 millones de instancias de objetos de 80 categorías diferentes, incluyendo personas, animales, vehículos, muebles y objetos de la vida cotidiana. Dicho pre-entrenamiento puede resultar muy útil para numerosas aplicaciones. Sin embargo, dado que nosotros queremos desplegar un sistema dedicado únicamente a la detección de señales de tráfico, trabajaremos con unos pesos especiales destinados a dicha operación.

El entrenamiento de algoritmos de detección de objetos puede ser muy costoso computacionalmente. El proceso de entrenamiento puede llevar varias horas, días o incluso semanas, dependiendo del tamaño del conjunto de datos y los recursos hardware disponibles. Por ello, decidimos partir de pesos ya pre-entrenados en detección de señales.

Los pesos utilizados están preparados para detectar cuatro grupos de señales diferentes:

- Prohibición: señales circulares con fondo blanco y borde rojo, como puede ser una prohibición de superar una determinada velocidad o la prohibición de entrada de un vehículo en vía.
- Peligro: señales triangulares con fondo blanco y borde rojo, como puede ser la señal de advertencia de una curva peligrosa o peligro de desprendimientos.
- Obligación: señales circulares azules, como puede ser la obligación de superar determinada velocidad u obligación de realizar un giro.
- Otros: resto de señales de tráfico.

Bajo estas premisas, mediante nuestra infraestructura de red 4G podemos poner en marcha el vehículo Amazon DeepRacer que tenemos en la figura 3.1, así podremos manejar el vehículo y acceder al contenido de su cámara.



Figura 3.1: Vehículo Amazon DeepRacer utilizado

Gracias a que nuestra escuela, ETSIT de la Universidad de Valladolid, poseía un pequeño circuito para trabajar con el vehículo, lo montamos y dispusimos varias señales de tráfico a lo largo de la maqueta. Así, pudimos probar el rendimiento de nuestro algoritmo de detección y los pesos de entrenamiento que posee. Tal y como podemos observar en las figuras 3.2 y 3.3, logramos la detección de las señales a lo largo de la maqueta, a pesar de la calidad del vídeo y de que las señales miniatura no sean exactamente igual a las reales.

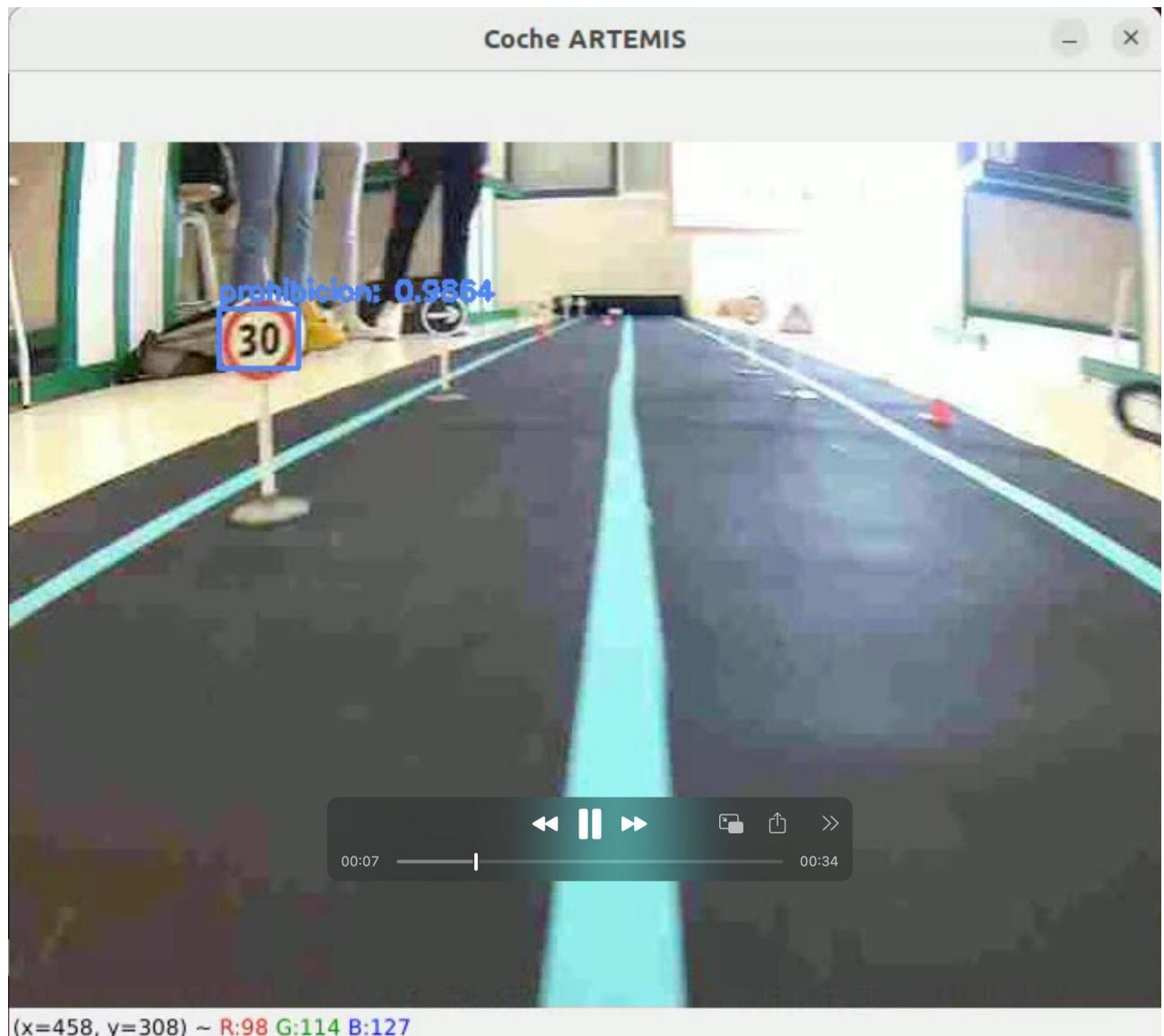


Figura 3.2: Primer ejemplo de detección de señales miniatura en vídeo en circuito

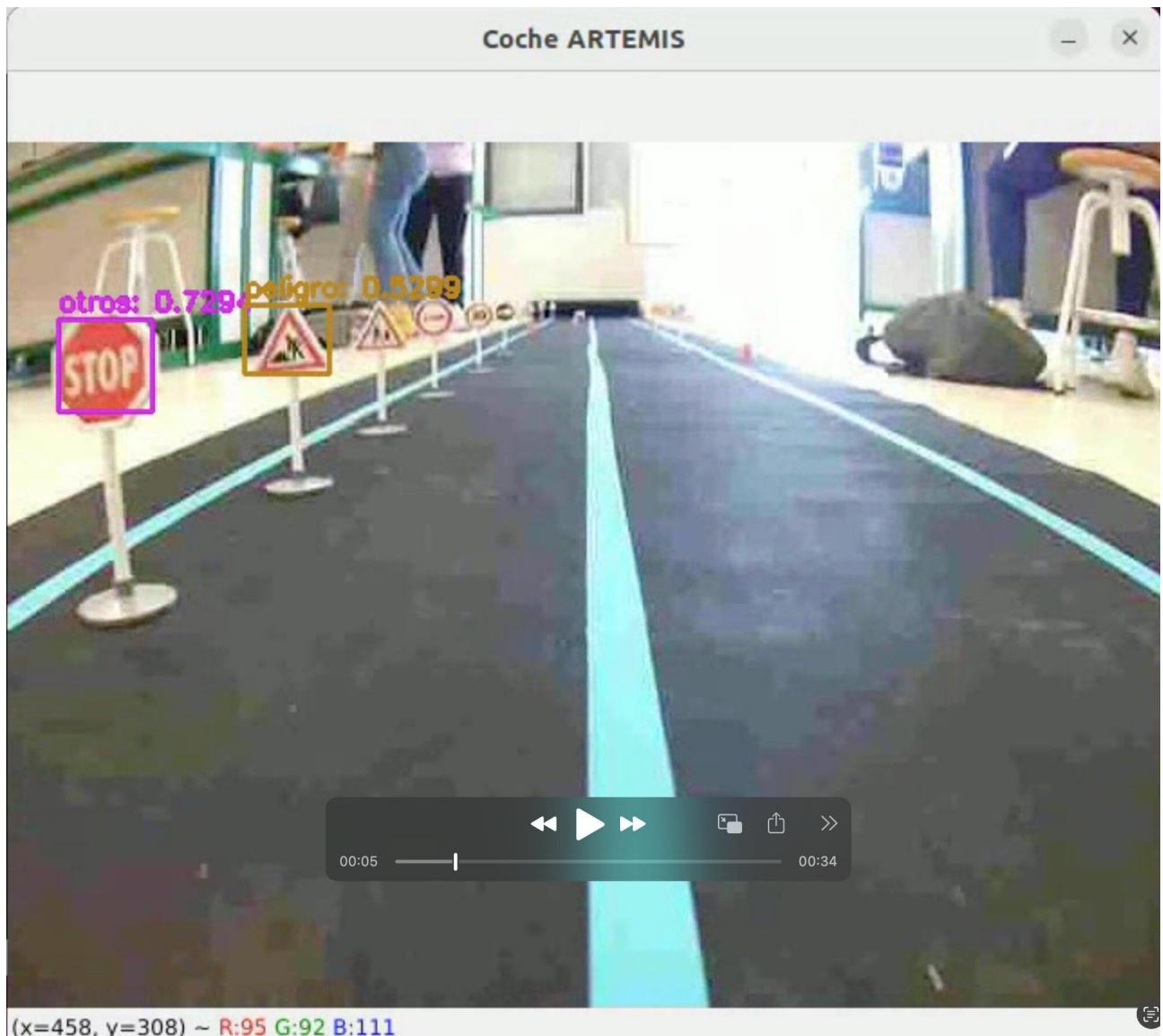


Figura 3.3: Segundo ejemplo de detección de señales miniatura en vídeo en circuito

De igual forma, podemos probar con imágenes de una vía normal, podemos visualizar el procesado de nuestro algoritmo en la figura 3.4



Figura 3.4: Detección de señales reales en vídeo en carretera real

Finalmente, a través de un popular repositorio de GitHub <https://github.com/rafaelpadilla/Object-Detection-Metrics> dedicado a medir el rendimiento de algoritmos de detección, podremos analizar nuestro sistema de detección de señales. A través de 64 imágenes etiquetadas y procesadas por el algoritmo, podremos medir el rendimiento de la red mediante sus detecciones. En concreto, podremos obtener la curva de Precisión vs Recuperación (Precision vs Recall) de nuestro modelo. La precisión es un término que describe la medida verdaderos positivos (TP) en relación con los falsos positivos (FP). Por otro lado, la recuperación se refiere a la proporción de verdaderos positivos (TP) en comparación con la suma de verdaderos positivos y falsos negativos (FN). En resumen, estos conceptos son utilizados para evaluar la efectividad de un modelo de clasificación y se pueden emplear para establecer el límite óptimo del modelo.

Para calcular la precisión y la recuperación se hace uso de las siguientes fórmulas:

$$\text{precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{recuperación} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Podemos contemplar en la figura XXX (rendimiento.pdf) que para cada categoría contemplada de señales de tráfico obtenemos una precisión media (AP) diferente. Para la clase obligación se obtiene una precisión del 57,9 %, para peligro un 72,22 %, para prohibición un 84,54 % y para el resto un 62,14 %. Destacar que entre las 64 imágenes que se han analizado, se encontraban tanto imágenes de señales reales, como señales de juguete, esto es clave porque los pesos pre-entrenados no contaban con señales falsas. Por ello, podemos notar que quizás los valores de precisión obtenidos no son suficientemente altos debido a la inclusión de señales que no se asemejan con las de entrenamiento.

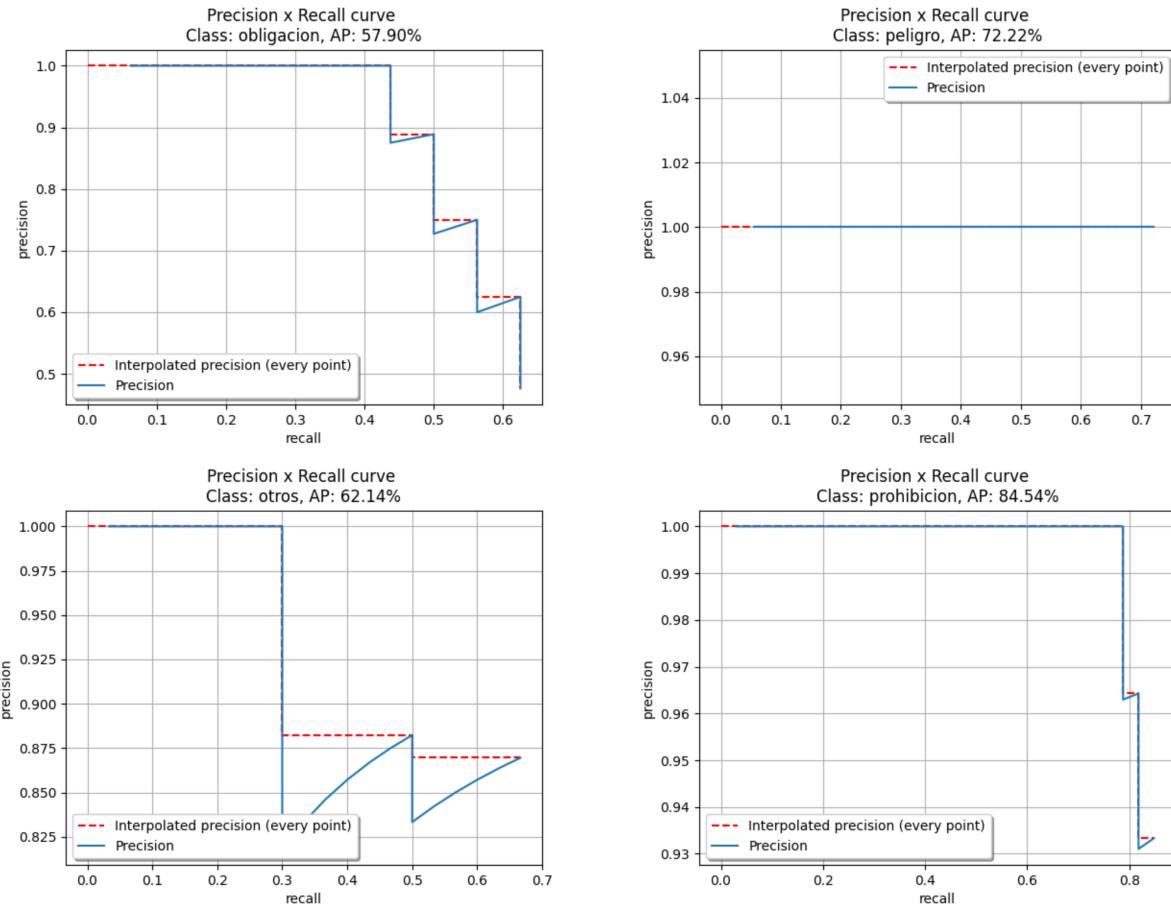


Figura 3.5: Gráficas del rendimiento obtenido

3.3. Normativa

Como en la tecnología LTE cada canal es de 15MHz, nos encontramos en una situación de banda estrecha. Ante estas situaciones podemos dividir el procedimiento en tres etapas:

- Pago tasas del modelo 790.
- Envío de la solicitud.
- Pago tasas del modelo 990.

3.3.1. Pago tasas del modelo 790

De acuerdo con la ley 39/2015, de 1 de octubre, del Procedimiento Administrativo Común de las Administraciones Públicas, y el Real Decreto 123/2017, de 24 de febrero, por el que se aprueba el Reglamento sobre el uso del dominio público radioeléctrico, la tramitación de los procedimientos relativos al espectro radioeléctrico, así como la relación con los órganos competentes del Ministerio a este respecto, se deberá llevar a cabo obligatoriamente por medios electrónicos, siempre que estén disponibles en la sede electrónica del Ministerio.

Se puede acceder a los procedimientos relacionados con el Servicio Móvil y Fijo de Banda Estrecha de la Subdirección General de Planificación y Gestión del Espectro Radioeléctrico en la sede electrónica del Ministerio.

El Real Decreto 1620/2005, de 30 de diciembre, por el que se regulan las tasas establecidas en la Ley 32/2003, de 3 de noviembre, General de Telecomunicaciones, completa y desarrolla la regulación de la referida Ley General de Telecomunicaciones, precisando las reglas y criterios aplicables para la fijación de las tasas y estableciendo el procedimiento para su liquidación. Esta resolución tiene por objeto establecer el procedimiento para la autoliquidación y las condiciones para el pago por

vía telemática de las tasas de telecomunicaciones establecidas en el anexo I, apartado 4 de la Ley 32/2003, de 3 de noviembre, General de Telecomunicaciones:

El artículo 4 anteriormente mencionado: Tasas de telecomunicaciones presenta que,

La gestión precisa para la emisión de certificaciones registrales y de la presentación de proyecto técnico y del certificado o boletín de instalación que ampara las infraestructuras comunes de telecomunicaciones en el interior de edificios, de cumplimiento de las especificaciones técnicas de equipos y aparatos de telecomunicaciones, así como la emisión de dictámenes técnicos de evaluación de la conformidad de estos equipos y aparatos, las inscripciones en el registro de instaladores de telecomunicación, las actuaciones inspectoras o de comprobación técnica que, con carácter obligatorio, vengan establecidas en esta ley o en otras disposiciones con rango legal, la tramitación de autorizaciones o concesiones demaniales para el uso privativo del dominio público radioeléctrico y la tramitación de autorizaciones de uso especial de dicho dominio darán derecho a la exacción de las tasas compensatorias del coste de los trámites y actuaciones necesarias, con arreglo a lo que se dispone en los párrafos siguientes.

La cuantía de la tasa se establecerá en la Ley de Presupuestos Generales del Estado. La tasa se devengará en el momento de la solicitud correspondiente. El rendimiento de la tasa se ingresará en el Tesoro Público o, en su caso, en las cuentas bancarias habilitadas al efecto respectivamente por la Comisión del Mercado de las Telecomunicaciones o por la Agencia Estatal de Radiocomunicaciones en los términos previstos en los artículos 47 y 48 de esta ley, en la forma que reglamentariamente se determine. Asimismo, reglamentariamente se establecerá la forma de liquidación de la tasa.

Por lo que se realiza el pago de la tasa 790, que se puede encontrar en la sede electrónica en el apartado de “Pago de tasas de tramitación de Telecomunicaciones. Modelo 790”. En “Redes Radioeléctricas del Servicio Móvil y Fijo de Banda Estrecha”.

Una vez realizado el pago de la tasa 790, se obtiene un justificante de dicho pago que se adjuntará con el paso 2, explicado a continuación.

3.3.2. Envío de la solicitud

En el anexo I, apartado 3 de la Ley 32/2003, de 3 de noviembre, General de Telecomunicaciones:

El artículo 3 anteriormente mencionado: Tasa por reserva del dominio público radioeléctrico presenta que,

La reserva para uso privativo de cualquier frecuencia del dominio público radioeléctrico a favor de una o varias personas o entidades se gravará con una tasa anual, en los términos que se establecen en este apartado.

Para la fijación del importe a satisfacer en concepto de esta tasa por los sujetos obligados, se tendrá en cuenta el valor de mercado del uso de la frecuencia reservada y la rentabilidad que de él pudiera obtener el beneficiario.

Para la determinación del citado valor de mercado y de la posible rentabilidad obtenida por el beneficiario de la reserva se tomarán en consideración, entre otros, los siguientes parámetros:

- a) El grado de utilización y congestión de las distintas bandas y en las distintas zonas geográficas.*
- b) El tipo de servicio para el que se pretende utilizar la reserva y, en particular, si éste lleva aparejadas las obligaciones de servicio público recogidas en el título III.*
- c) La banda o sub-banda del espectro que se reserve.*
- d) Los equipos y tecnología que se empleen.*
- e) El valor económico derivado del uso o aprovechamiento del dominio público reservado.*

En el artículo 85, apartado 3 de la Ley 31/2022, de 23 de diciembre, de Presupuestos Generales del Estado para el año 2023 se especifica el importe correspondiente al pago de la tasa 990,

La tasa por reserva de dominio público radioeléctrico establecida en el apartado 3 del anexo I de la Ley 11/2022, de 28 de junio, General de Telecomunicaciones (en adelante Ley General de Telecomunicaciones), ha de calcularse mediante la expresión:

$$T = \frac{[N \times V]}{166,386} = \frac{[S(km^2) \times B(kHz) \times F(C1, C2, C3, C4, C5)]}{166,386}$$

En donde:

T = importe de la tasa anual en euros.

N = número de unidades de reserva radioeléctrica (URR) calculado como el producto de S x B, es decir, superficie en kilómetros cuadrados de la zona de servicio, por ancho de banda reservado expresado en KHz.

V = valor de la URR, determinado en función de los cinco coeficientes Ci, establecidos en la Ley General de Telecomunicaciones, y cuya cuantificación, de conformidad con dicha Ley, será establecida en la Ley de Presupuestos Generales del Estado.

F (C1, C2, C3, C4, C5) = esta función es el producto de los cinco coeficientes indicados anteriormente.

Donde, $N = 9,1 \times 10^3$

El proyecto presente se encuentra dentro de la situación descrita en el apartado 1.1.2: Servicio móvil asignación fija/frecuencia compartida/zona de alta utilización (ya que la red se encuentra en un área metropolitana)/autoprestación. Y por tanto:

- C1=1,25
- C2=1,25
- C3=1,1
- C4=1,3
- C5=0,4590

Una vez realizadas las presentes cuentas, se obtiene un valor de importe aproximado de $T = 616,3848 \text{ €}$ anuales. Sin embargo, como es una red experimental, su uso está limitado a 6 meses, por lo que el pago de la tasa 990 es de 308,1924.

Capítulo 4

Costes

Capítulo 5

Rendimiento: Validación del servicio

Rendimiento

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
```

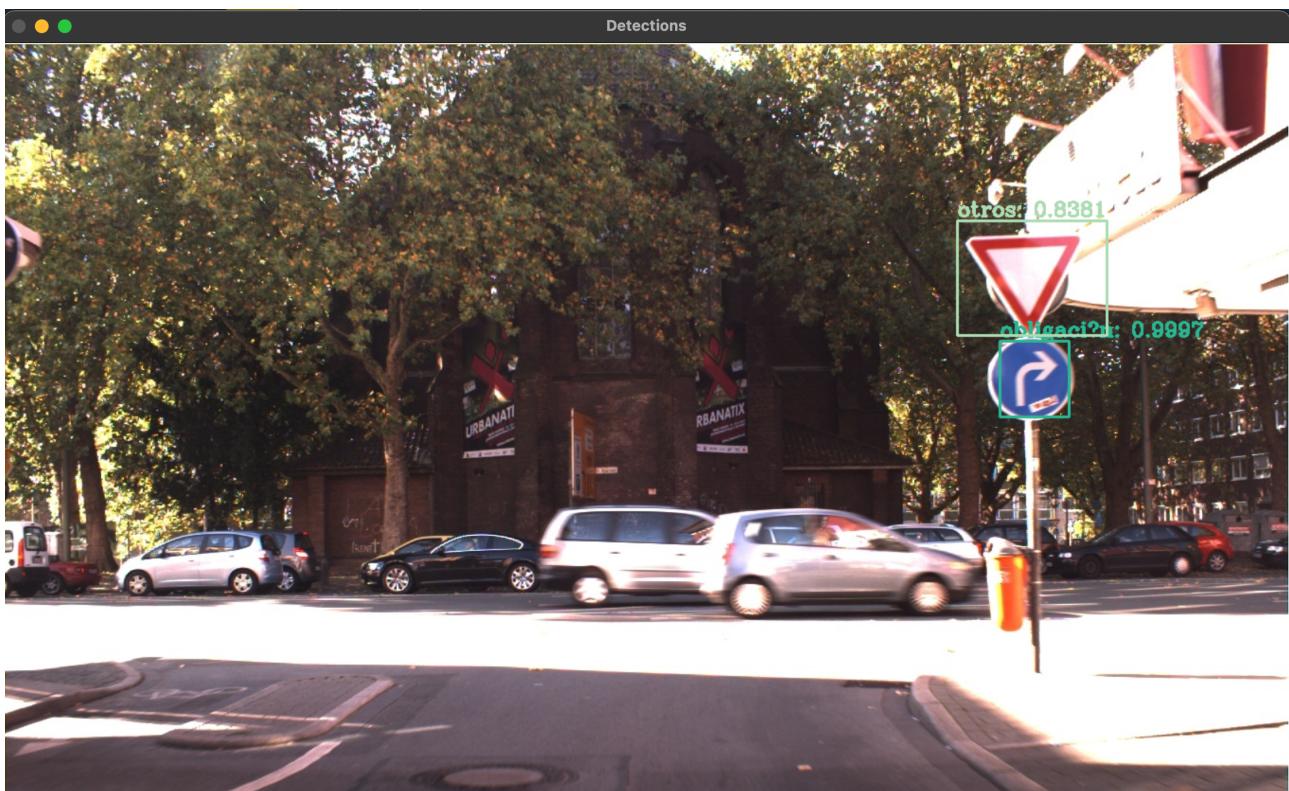


Figura 6.1: Detección de una señal

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
```

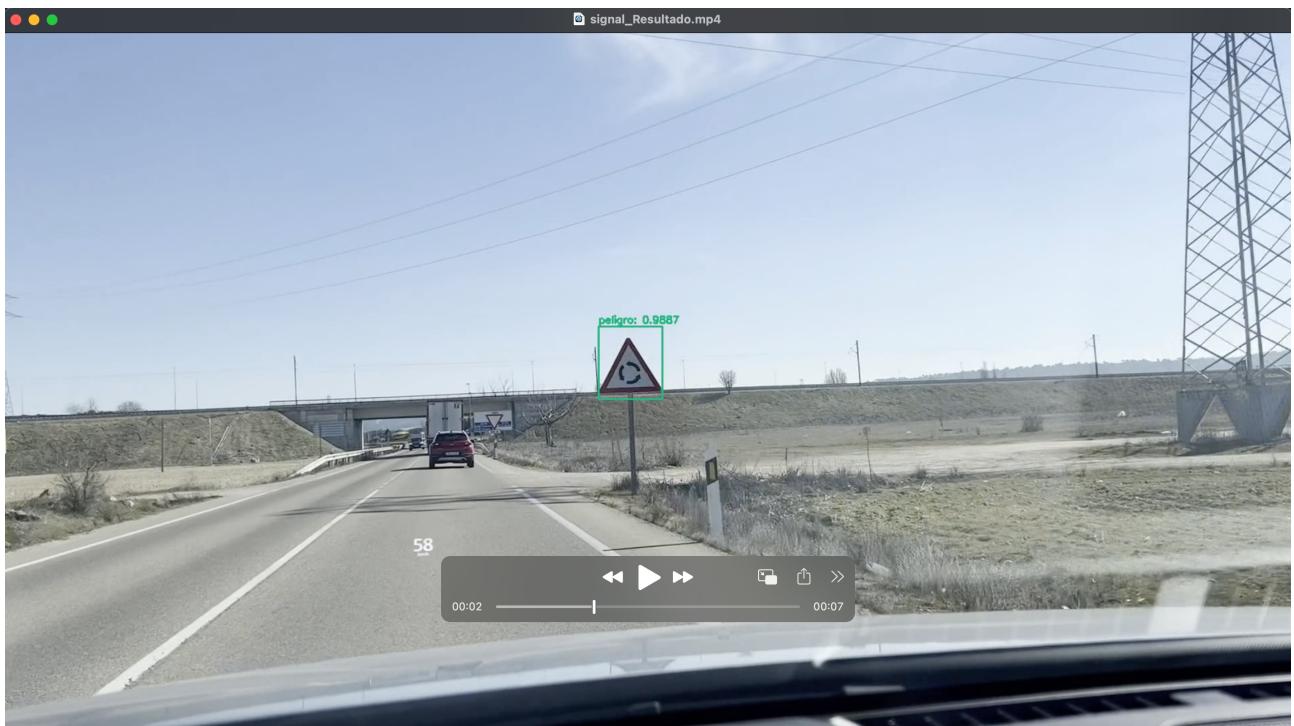


Figura 6.2: Detección de una señal real capturada por nosotros

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.

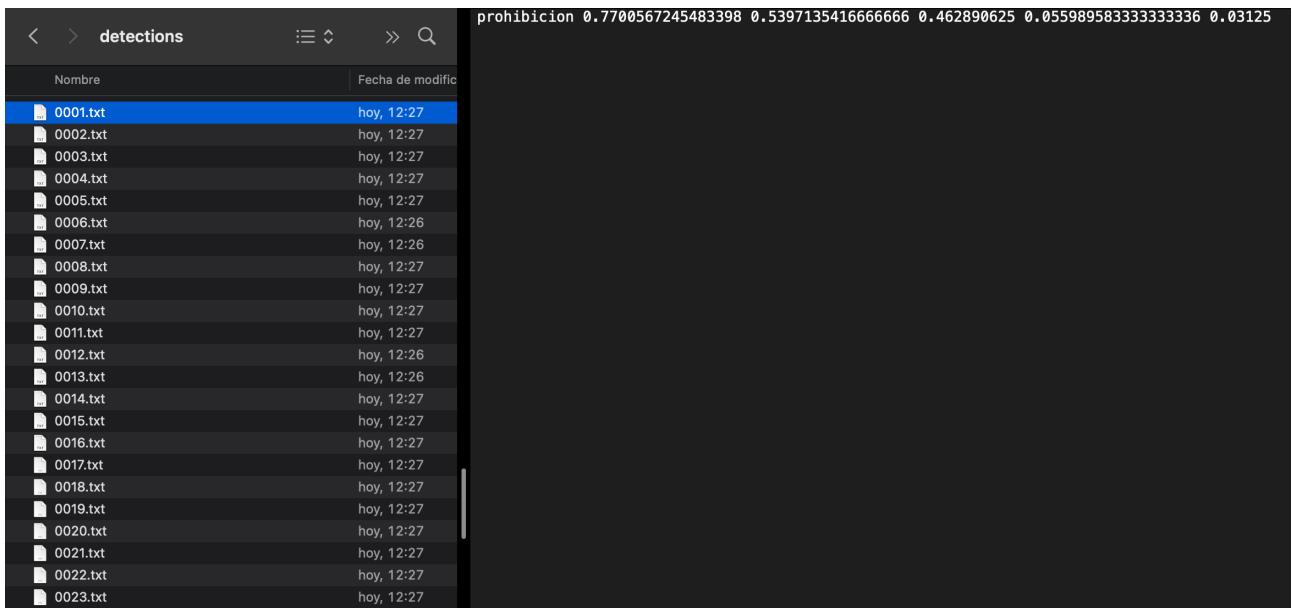


Figura 6.4: Ejemplo de rendimiento con *medirRendimientoRed = True*

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:

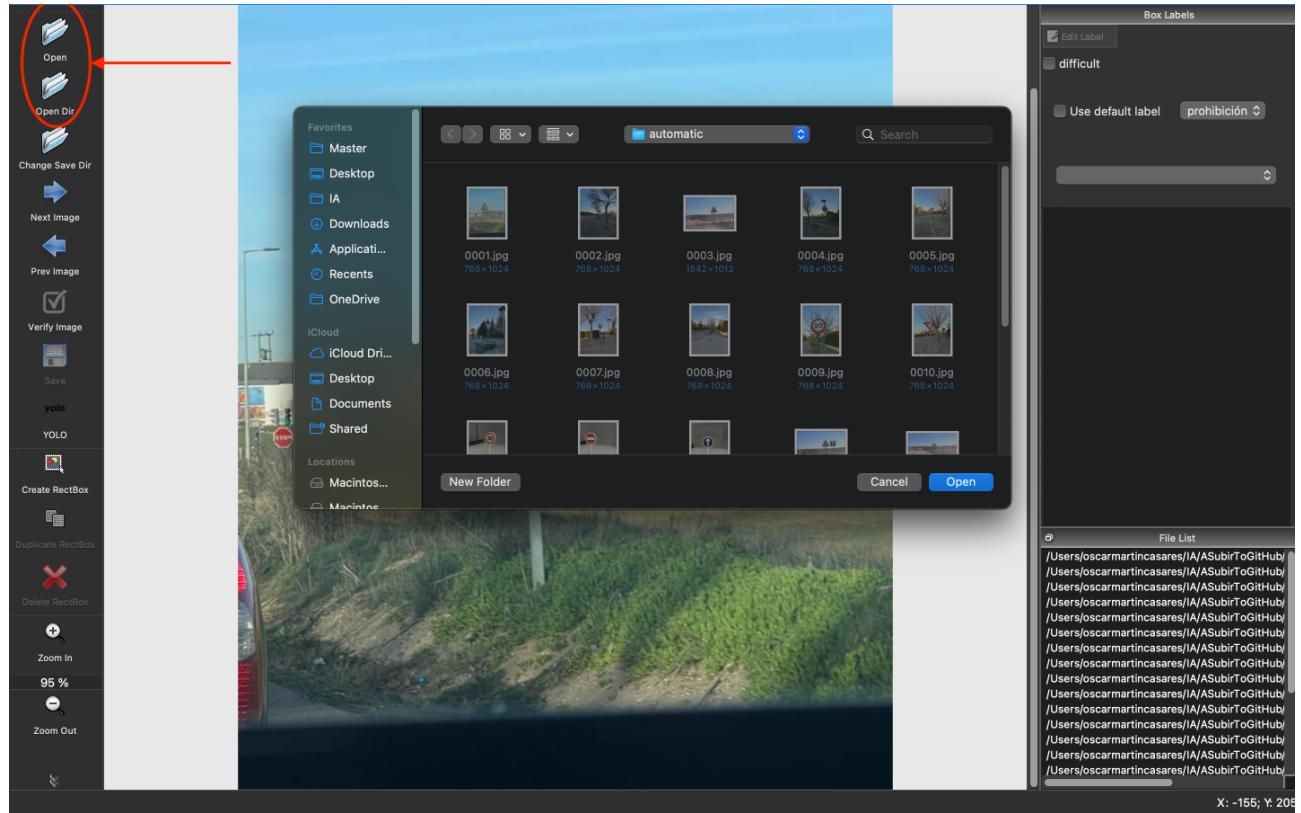


Figura 6.5: Etiquetado de una señal

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**.

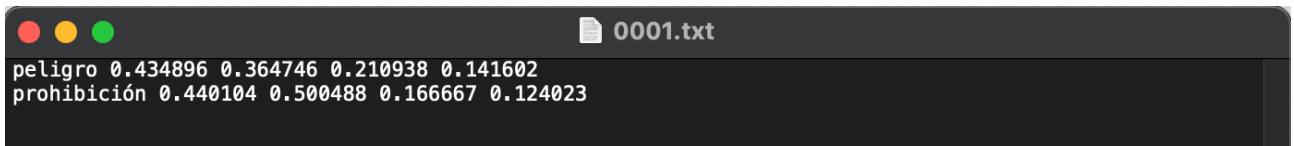


Figura 6.8: Fichero con la información de etiquetado

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:

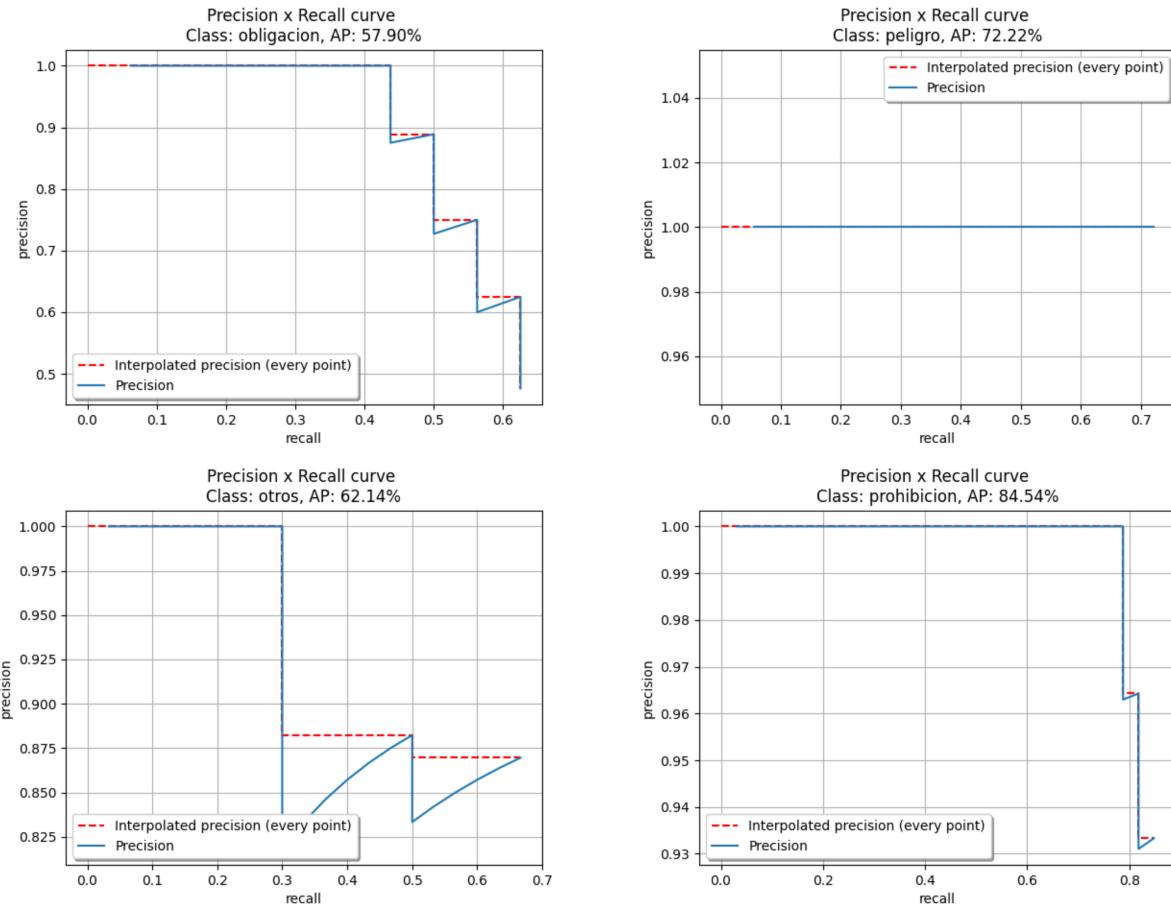


Figura 6.9: Medida de rendimiento

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 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*:

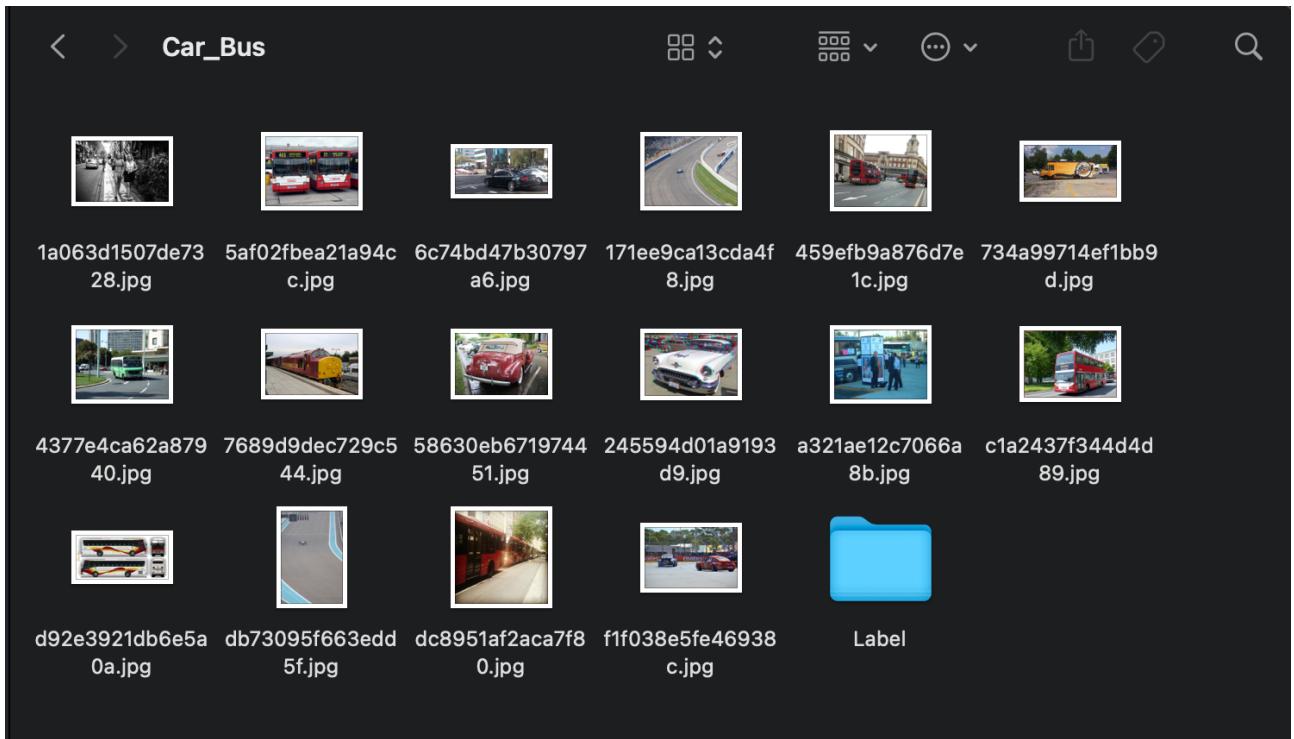


Figura 6.10: Dataset descargado con los ficheros TXT

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