



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

1 de junio de 2023

Índice general

| | |
|---|-----------|
| 1. Resumen | 4 |
| 2. Introducción | 5 |
| 3. Planteamiento inicial | 6 |
| 4. Solución | 8 |
| 4.1. Infraestructura | 8 |
| 4.2. Detección de señales de tráfico | 11 |
| 4.3. Normativa | 25 |
| 4.3.1. Pago tasas del modelo 790 | 26 |
| 4.3.2. Envío de la solicitud | 26 |
| 4.3.3. Pago tasas del modelo 990 | 27 |
| 5. Costes | 29 |
| 6. Rendimiento: Validación del servicio | 31 |
| 6.1. Kernel de Baja Latencia | 34 |
| 7. Conclusiones | 36 |
| 8. Anexo I: Manual del desarrollador | 37 |
| 8.1. Infraestructura de la red | 37 |
| 8.2. Despliegue IA | 41 |
| 8.2.1. Detección y seguimiento de señales | 42 |
| 8.2.2. Herramienta de etiquetado | 48 |
| 8.2.3. Medición de rendimiento | 50 |
| 8.2.4. Creación automática de datasets | 51 |
| 9. Anexo II: Legislación | 53 |
| 10. Anexo II: Planificación | 54 |

Índice de figuras

| | |
|--|----|
| 3.1. Mapa de la autovía A-62 y las torres de comunicación existentes | 6 |
| 4.1. Distribución de la cobertura y estudio de cobertura | 10 |
| 4.2. Vehículo Amazon DeepRacer utilizado | 12 |
| 4.3. Primer ejemplo de detección de señales miniatura en vídeo en circuito | 13 |
| 4.4. Segundo ejemplo de detección de señales miniatura en vídeo en circuito | 13 |
| 4.5. Detección de señales reales en vídeo en carretera real | 14 |
| 4.6. Gráficas del rendimiento obtenido | 15 |
| 4.7. Ventajas de las diferentes tecnologías | 16 |
| 4.8. Estructura general de las matrices de confusión | 18 |
| 4.9. Fórmulas de cálculo | 18 |
| 4.10. Matriz de confusión para la categoría peligro | 19 |
| 4.11. Matriz de confusión para la categoría obligación | 19 |
| 4.12. Matriz de confusión para la categoría prohibición. | 20 |
| 4.13. Matriz de confusión para la última categoría. | 20 |
| 4.14. Resultados de la prueba (1) | 21 |
| 4.15. Resultados de la prueba (2) | 22 |
| 4.16. Resultados de la prueba (3) | 23 |
| 4.17. Resultados de la prueba (4) | 24 |
| 4.18. Fallo de detección. | 25 |
| 6.1. Esquema de la red. (Se ha usado <i>iproute</i> para saber las direcciones IP de las diferentes interfaces). | 31 |
| 6.2. Salida del comando <i>ping</i> | 32 |
| 6.3. Salida del comando "iperf" en el lado cliente. | 33 |
| 8.1. Dataset descargado con los ficheros TXT | 38 |
| 8.2. Capturas de pantalla de la aplicación móvil MyMQTT para cliente suscrito | 40 |
| 8.3. Publicación de comandos desde el móvil | 41 |
| 8.4. Detección de una señal | 43 |
| 8.5. Detección de una señal real capturada por nosotros | 44 |
| 8.6. Detección de una señal captada por la cámara | 44 |
| 8.7. Ejemplo de rendimiento con <i>medirRendimientoRed = True</i> | 45 |
| 8.8. Ejemplo de ejecución sobre el propio coche | 46 |
| 8.9. Ejemplo sobre la cámara frontal del ordenador | 47 |
| 8.10. Matrices de confusión | 47 |
| 8.11. Etiquetado de una señal | 48 |
| 8.12. Selección del modelo | 49 |
| 8.13. Cuadros delimitadores | 49 |
| 8.14. Fichero con la información de etiquetado | 50 |
| 8.15. Medida de rendimiento | 51 |
| 8.16. Dataset descargado con los ficheros TXT | 52 |

Índice de tablas

| | |
|---|----|
| 4.1. Cobertura en función de la modulación. | 9 |
| 4.2. Capacidades del enlace de bajada y de subida | 9 |
| 4.3. Cantidad de usuarios disponibles en función de la velocidad del vehículo | 10 |
| 5.1. Costes Fijos | 29 |
| 5.2. Costes mensuales | 29 |
| 5.3. Medida de vehículos que circulan por el tramo | 29 |
| 5.4. Tarifas y porcentaje de usuarios estimados | 30 |
| 5.5. Ingresos en función de las estimaciones | 30 |
| 5.6. Amortizaciones en función de las estimaciones | 30 |

Capítulo 1

Resumen

Se presenta un proyecto en el cual nuestra empresa ha sido encargada de implementar una infraestructura de red 4G con el propósito de brindar servicios a vehículos conectados en una autopista específica en Castilla y León. El tramo de carretera seleccionado abarca 73 kilómetros, donde se aprovechará la infraestructura de red 4G existente y se instalarán equipos adicionales en las torres de telefonía ya disponibles a lo largo de la carretera. El objetivo principal de este proyecto es ofrecer un servicio ininterrumpido con un ancho de banda suficiente para transmitir vídeo captado por cámaras en los vehículos, lo cual permitirá a los usuarios detectar señales de tráfico, mejorar la seguridad vial, cumplir con las regulaciones de tráfico, así como optimizar el consumo de combustible y reducir las emisiones contaminantes. Para lograr estos objetivos, se propone el uso de técnicas avanzadas de procesamiento de imágenes y aprendizaje automático, como el modelo YOLO, con el fin de lograr una detección precisa y eficiente de las señales de tráfico, contribuyendo así a la seguridad vial y ofreciendo soluciones innovadoras.

Se han considerado varias decisiones y aspectos relacionados con la infraestructura de red. Se ha optado por alquilar torres existentes y utilizar componentes propios para diseñar una red que cumpla con los requisitos económicos y proporcione cierto grado de control en caso de problemas. Se han explorado opciones como la construcción de torres propias, el funcionamiento como operador virtual o la adquisición e instalación de equipos propios. Se destaca la selección de una estación base de la empresa china Baicells y una antena de la empresa irlandesa Alpha Wireless, junto con sus características técnicas relevantes. Además, se menciona el uso de la herramienta Xirio-online para simular la infraestructura y realizar la planificación y el diseño de las redes de telecomunicaciones. Asimismo, se plantea la opción de alquilar servicios de AWS en lugar de adquirir una gran cantidad de servidores para el procesamiento de datos. Se enfatiza la importancia de establecer una conexión dedicada de alta velocidad y confiabilidad para gestionar eficientemente el tráfico de datos en la red 4G.

Para la detección de objetos, se utiliza el algoritmo YOLO (You Only Look Once), el cual permite una detección rápida y precisa en una imagen completa. Para evaluar el rendimiento de la detección, se utilizan métricas como la precisión y la recuperación.

Además, se ha explicado el proceso burocrático necesario para enviar la solicitud y realizar el pago de las tasas correspondientes mediante los modelos 790 y 990, requisitos esenciales para llevar a cabo el proyecto. También se ha elaborado una tabla con los costos y amortizaciones asociados al proyecto.

Por último, se ha realizado un análisis del rendimiento de la red utilizando la versión estándar de Linux y aplicando un kernel de baja latencia.

Capítulo 2

Introducción

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 vídeo desde el vehículo a un servidor en la nube. El vídeo será procesado en *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 vídeo desde los vehículos conectados al servidor *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 técnico-económica 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*), en concreto una BladeRF 2.0 xa9, 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, que nos permite configurar y gestionar las comunicaciones.

En cuanto a las frecuencias, se usan las de banda 7. En concreto el rango que va de 2540 a 2550 MHz y de 2650 a 2670 MHz. 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 3

Planteamiento inicial

Partimos de la base de que, como se ha comentado anteriormente, 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 3.1. 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 costes asociados a los componentes y considerar opciones con precios competitivos.

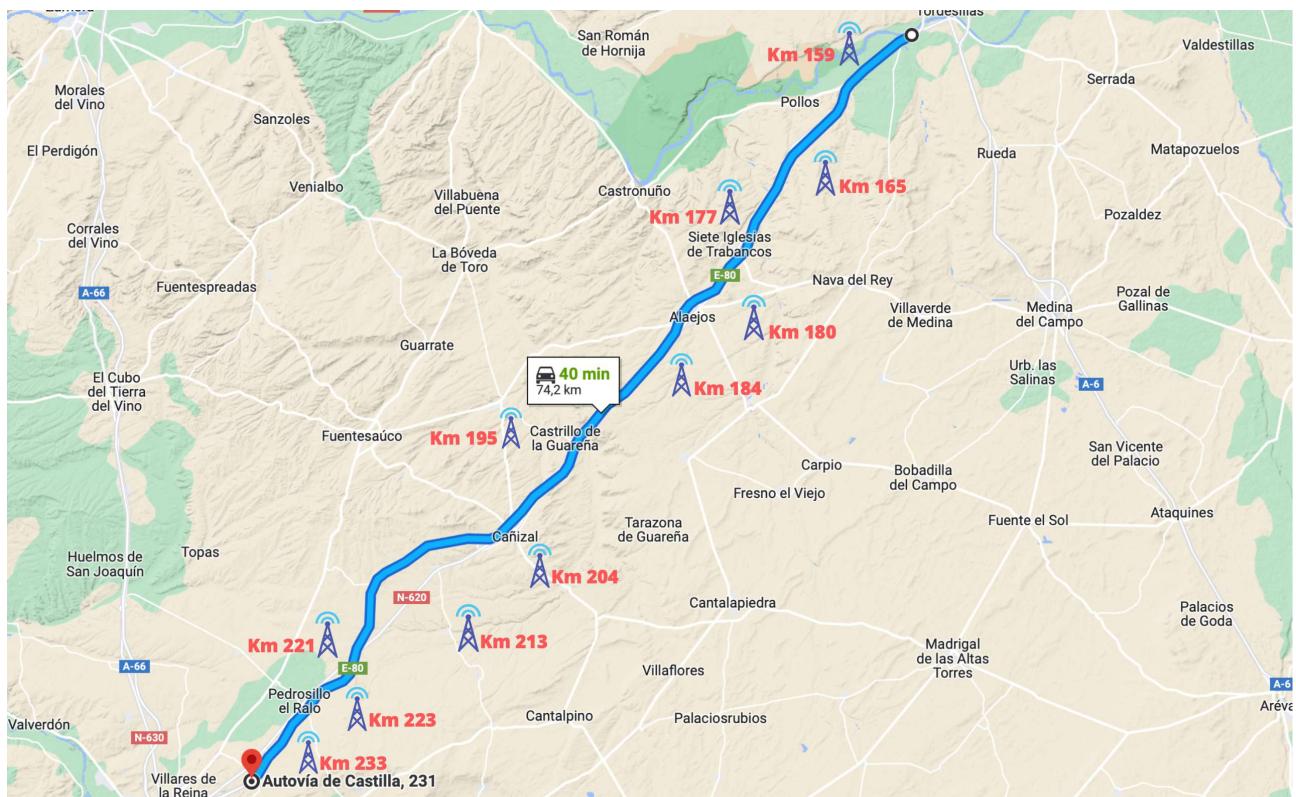


Figura 3.1: Mapa de la autovía A-62 y las torres de comunicación existentes

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

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 en general. De igual forma, puede fomentar el cumplimiento de las normativas y regulaciones de tráfico, disminuir la cantidad de sanciones, o puede ayudar a optimizar el consumo de combustible y moderar las emisiones de gases contaminantes procedentes de dichos vehículos.

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, desconocimiento de la misma 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 permitiendo 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 eso, 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. Uno de los más avanzados y precisos es YOLO. Lo que le 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 4

Solución

4.1. Infraestructura

En primera instancia, se debe decidir cómo se quiere plantear la infraestructura de red, si vamos a construir nuestras propias torres, 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.

Además, 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 3.1.

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, este 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 4.1: Cobertura en función de la modulación.

Dependiendo de la configuración se proporcionará un rendimiento u otro, en nuestro caso nos hemos decidido por la configuración 6. Esta 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*.

| Modulación | DownLink | UpLink |
|------------|----------|---------|
| 256 QAM | 348 Mbps | 92 Mbps |
| 64 QAM | 264 Mbps | 70 Mbps |
| 16 QAM | 70 Mbps | 53 Mbps |
| QPSK | 53 Mbps | 53 Mbps |

Tabla 4.2: Capacidades del enlace de bajada y de subida

La antena seleccionada es fabricada por la empresa irlandesa Alpha Wireless, una compañía fundada en 2006 dedicada a la fabricación de antenas. El modelo seleccionado es el AW3864-E-F, una antena sectorial de 4G que nos proporciona un ancho de haz de 65deg y una ganancia de 18 dB en las frecuencias de interés.

Con estos datos nos encontramos en disposición de simular la infraestructura, para ello hemos utilizado la herramienta de planificación y diseño de redes de telecomunicaciones Xirio-online. Una plataforma desarrollada una empresa española llamada Xirio, especializada en soluciones de ingeniería para telecomunicaciones. Con ésta se pueden crear mapas de cobertura y realizar simulaciones de propagación de señal con redes de telefonía móvil o televisión digital entre otras, así como evaluar del rendimiento de la red.

A través de un estudio de multicobertura e introduciendo todos los parámetros ya mencionados en la herramienta, hemos dispuesto las antenas a lo largo de tramo de autovía en las distintas ubicaciones en las que previamente localizamos las torres de telecomunicaciones. Como se puede apreciar en la figura 4.1 teniendo en cuenta la cartografía del terreno, podremos cubrir la totalidad de la carretera a través de 7 estaciones base y 12 antenas. Asimismo, dado que no hemos logrado averiguar la altura exacta de cada una de las torres utilizadas, hemos supuesto una altura de 30 metros para todas ellas, ya que la altura media se encuentre en 15 y 50 metros.

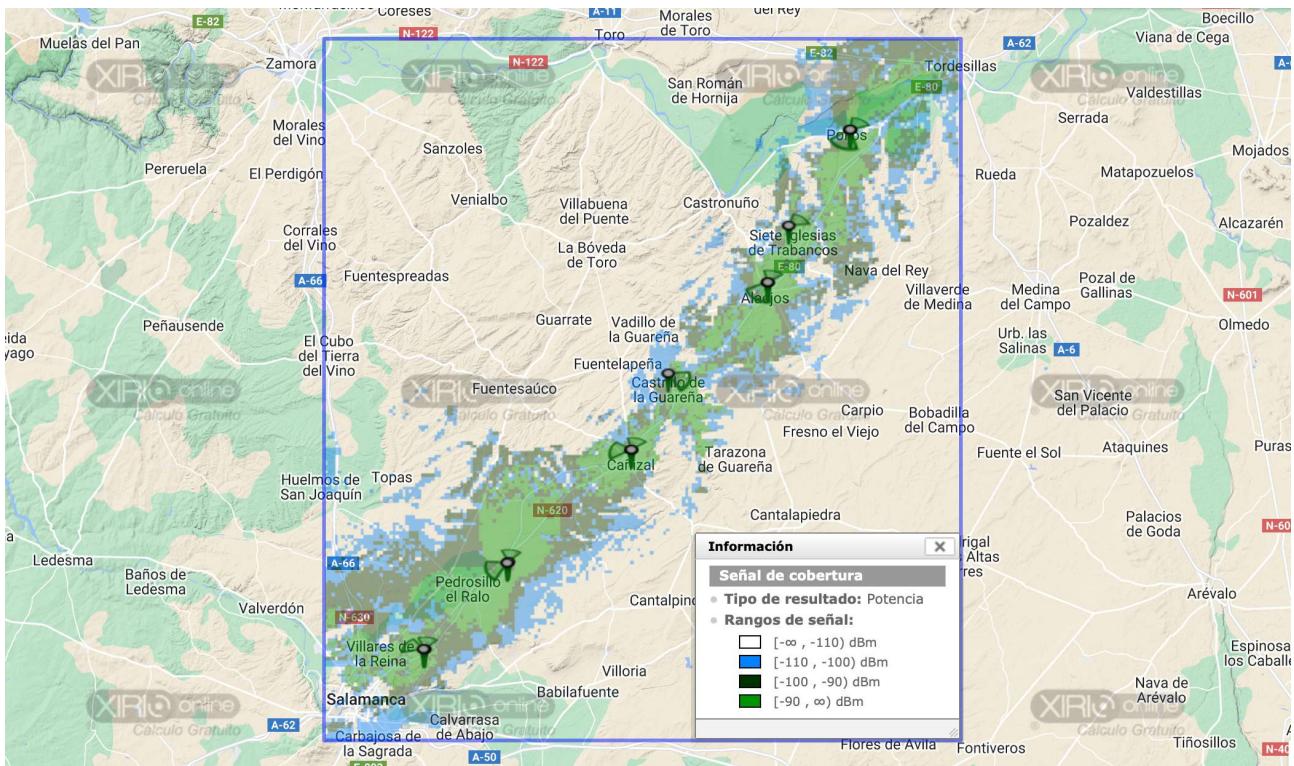


Figura 4.1: Distribución de la cobertura y estudio de cobertura

Un detalle muy importante del proyecto es conocer cuál es la cantidad de usuarios a la que se podrá prestar el servicio de detección de señales de tráfico. Tal y como hemos distribuido las antenas y estaciones base, en prácticamente todo el tramo de autovía se proporciona una potencia suficiente como para que los usuarios operen a través de la modulación 256 QAM. De este modo, podremos calcular la cantidad de usuarios a la cual podremos prestar servicio en función de las distintas velocidades que nos podríamos encontrar. Dividiendo la capacidad del enlace de subida entre la cantidad de Mbps necesarios para transmitir por cada usuario, podremos conocer el número de usuarios posible de manera simultánea, teniendo en cuenta además cuál es la distancia y fotogramas por segundo adecuados para poder detectar la señal con fiabilidad.

| Distancia | Velocidad | Frames/Sec | Mbps | Cantidad Usuarios |
|-----------|-----------|------------|----------|-------------------|
| 20 metros | 80 km/h | 12 f/s | 2 Mbps | 46 usuarios |
| 20 metros | 100 km/h | 15 f/s | 2,5 Mbps | 36 usuarios |
| 20 metros | 120 km/h | 18 f/s | 3 Mbps | 30 usuarios |

Tabla 4.3: Cantidad de usuarios disponibles en función de la velocidad del vehículo

La interconexión entre las estaciones base y el núcleo de la red 4G se establecerá utilizando una topología de estrella, donde el núcleo se situará en Madrid junto con el centro de operaciones (CTO). Esta elección permite centralizar la gestión completa del servicio en una ubicación, lo que brinda ventajas logísticas en caso de necesitar intervenciones. La elección de Madrid se basa en su ubicación geográfica estratégica y en la disponibilidad de abundantes recursos.

El core de la red 4G, esencial para proporcionar servicios de conectividad y gestionar el tráfico de datos, se implementará utilizando la solución Telrad BreezeWAY2020 de Telrad Networks. Telrad Networks, una empresa líder en el sector de las telecomunicaciones desde 1951, ofrece esta solución que tiene la capacidad de atender simultáneamente a 10.000 usuarios. Además, cuenta con una Arquitectura de Clúster Distribuido que permite un escalado gradual y prácticamente ilimitado en términos de capacidad y rendimiento a nivel de red.

El centro de operaciones se encargará de la detección de señales de tráfico utilizando modelos de inteligencia artificial, como YOLO v3. Dado el alto requerimiento computacional de este servicio, ha surgido el debate sobre si es más conveniente adquirir una gran cantidad de servidores o bien optar por servicios en la nube ofrecidos por grandes empresas tecnológicas como AWS (Amazon) o Azure (Microsoft).

En el contexto de los vehículos autónomos, se ha observado una tendencia creciente hacia la realización del cálculo computacional en el propio vehículo. Esto se debe a que delegar un mayor acceso y control externo aumenta los riesgos de seguridad. Por lo tanto, resulta más seguro realizar el procesamiento de datos en el vehículo mismo.

En línea con esta consideración, se ha decidido alquilar los servicios de AWS, ya que una inversión inicial significativa no sería viable a largo plazo, dado que en el futuro estos servicios podrían no utilizarse para este propósito.

El núcleo de nuestra red establece su conectividad a Internet mediante un enlace de conexión dedicado, el cual puede ser suministrado por un proveedor de servicios de Internet (ISP) o a través de una conexión de fibra óptica. Es fundamental que este enlace posea una velocidad y fiabilidad adecuadas para gestionar eficientemente el tráfico de datos generado por los usuarios de la red 4G.

4.2. 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 [yol, b].

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 [yol, a].

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 4.2, así podremos manejar el vehículo y acceder al contenido de su cámara.



Figura 4.2: 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 4.3 y 4.4, 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.

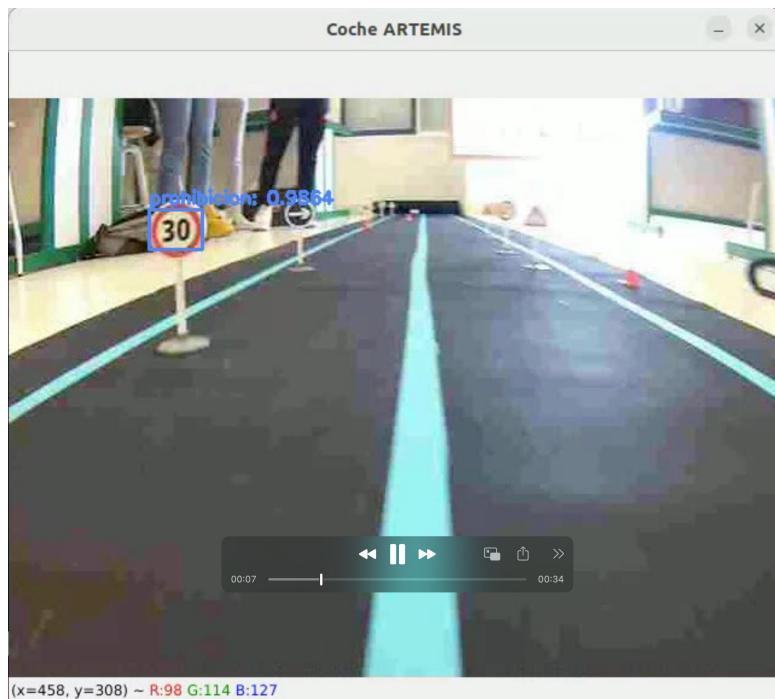


Figura 4.3: Primer ejemplo de detección de señales miniatura en vídeo en circuito

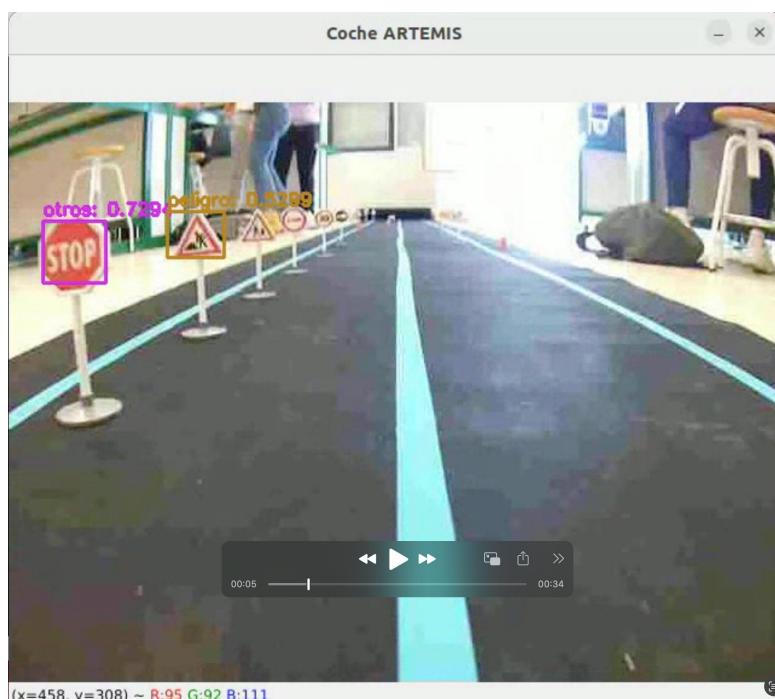


Figura 4.4: Segundo ejemplo de detección de señales miniatura en vídeo en circuito

De igual forma, podemos probar con imágenes de una vía normal, podemos visualizar el procesado de nuestro algoritmo en la figura 4.5



Figura 4.5: 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.git> 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 8.15 que para cada categoría contemplada de señales de tráfico obtenemos una precisión media (AP) diferente. . Establecida una probabilidad mínima de detección de 0,5 y threshold de 0,3, tenemos 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.

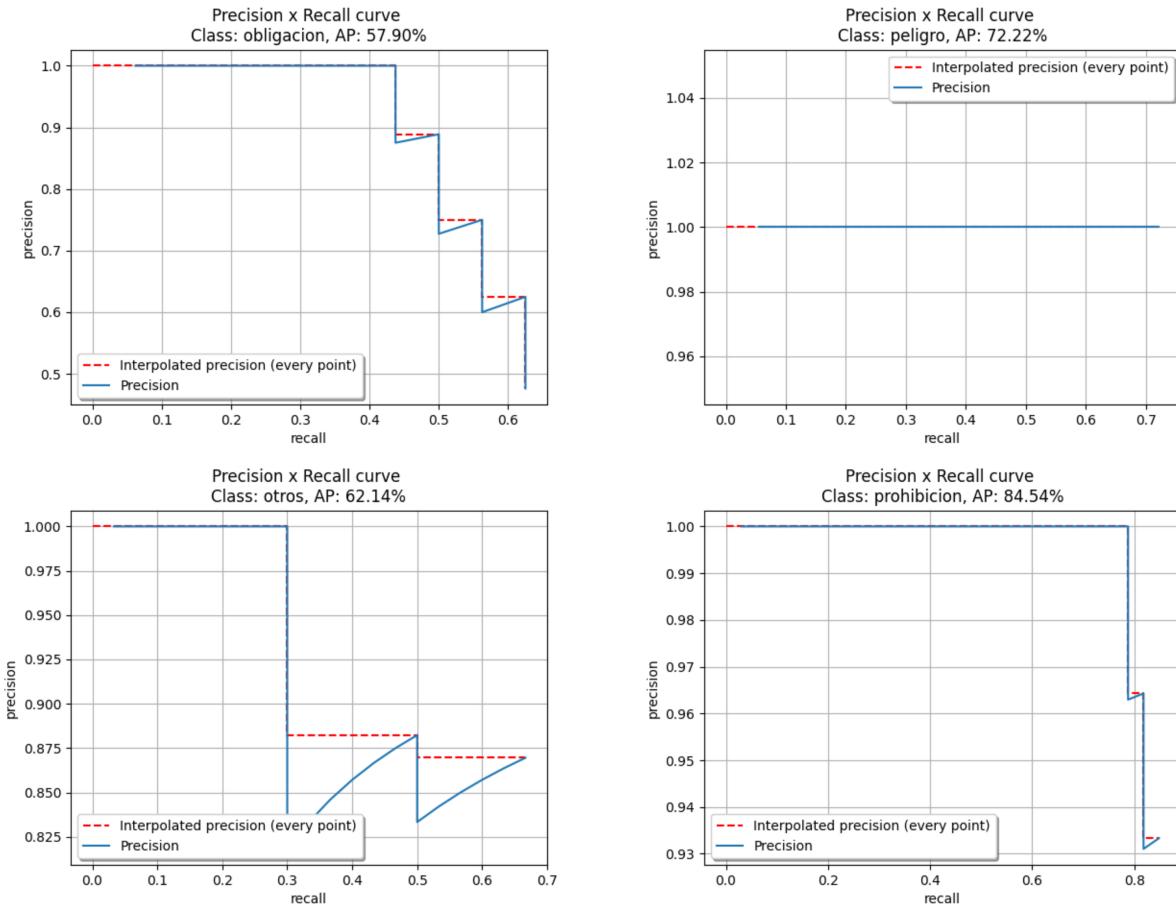


Figura 4.6: Gráficas del rendimiento obtenido

En primer lugar, en conocimiento de que la mayor precisión la hemos obtenido para la clase prohibición con una probabilidad de casi el 85 %, podemos ver que está cerca de parecerse a la curva ideal. La curva ideal que desearíamos sería una curva que se acerque lo máximo posible a la esquina superior derecha, es decir, alta precisión y alto recall. Asimismo, podemos relacionar la curva ideal con el área bajo la curva de precisión, el valor de precisión media obtenido representa el área bajo dicha curva, por lo que si la curva fuese ideal sería claro que al área sería del 100 %.

Podemos concluir entonces que la mejor precisión se ha obtenido en la detección de señales de prohibición. Además, destacar el resultado obtenido para la clase peligro, el cual puede llamar la atención por resultar una línea recta en todo momento. Esto se debe a que para todas las imágenes que se han contemplado para medir el rendimiento, el algoritmo ha sido capaz de detectar todas las señales que se han procesado. Este resultado quizás no sea del todo realista, por lo que se podrían añadir un mayor número de imágenes de cada clase para lograr objetivos más fehacientes con la realidad.

Una vez montada la estructura principal del sistema de inteligencia artificial, se procedió a intentar implementar nuevas funcionalidades con el objetivo de mejorar las prestaciones. Se decidió entonces dar un paso más allá en la clasificación de las señales, para intentar detectar de qué tipo de señal concreta se trata.

Como ya se ha mencionado, entrenar un algoritmo de IA requiere de una capacidad computacional muy alta. Por ello, una forma sencilla de implementar cierta inteligencia que sea capaz de detectar las señales tras procesarlas mediante YOLO, sería la incorporación de una red CNN.

Una CNN o Convolutional Neural Network (Red Neuronal Convolucional) es un tipo de arquitectura de red neuronal especialmente diseñada para procesar datos, como imágenes o audio. Ésta destaca por su rendimiento en visión artificial gracias a su capacidad para capturar características locales y patrones espaciales en imágenes, es decir, detección y clasificación de objetos.

La consecución de ambas tecnologías, tal y como se ejemplifica en la figura 4.7, nos permite ciertas ventajas

en la consecución de ambas tecnologías:

- 1. Eficiencia: YOLO destaca por su eficiencia en la detección de objetos, por lo que hacer una detección inicial nos permitirá identificar rápidamente las ubicaciones aproximadas de las señales de tráfico.
- 2. Localización precisa: Una vez que YOLO detecta las señales de tráfico, la ubicación de la señal se utiliza para extraer una región más pequeña, que es enviada a las CNN correspondientes para una clasificación más precisa. Al hacer esto, te aseguras de que solo se envíen las regiones relevantes a las CNN, en lugar de toda la imagen, lo que ayuda a mejorar la eficiencia y precisión del proceso de clasificación.
- 3. Especialización: Al entrenar las CNN separadas para cada categoría específica de señales de tráfico, logramos una especialización en la clasificación. Como cada CNN se entrena con imágenes específicas, permite una mayor capacidad para aprender características distintivas y patrones asociados a cada tipo de señal.

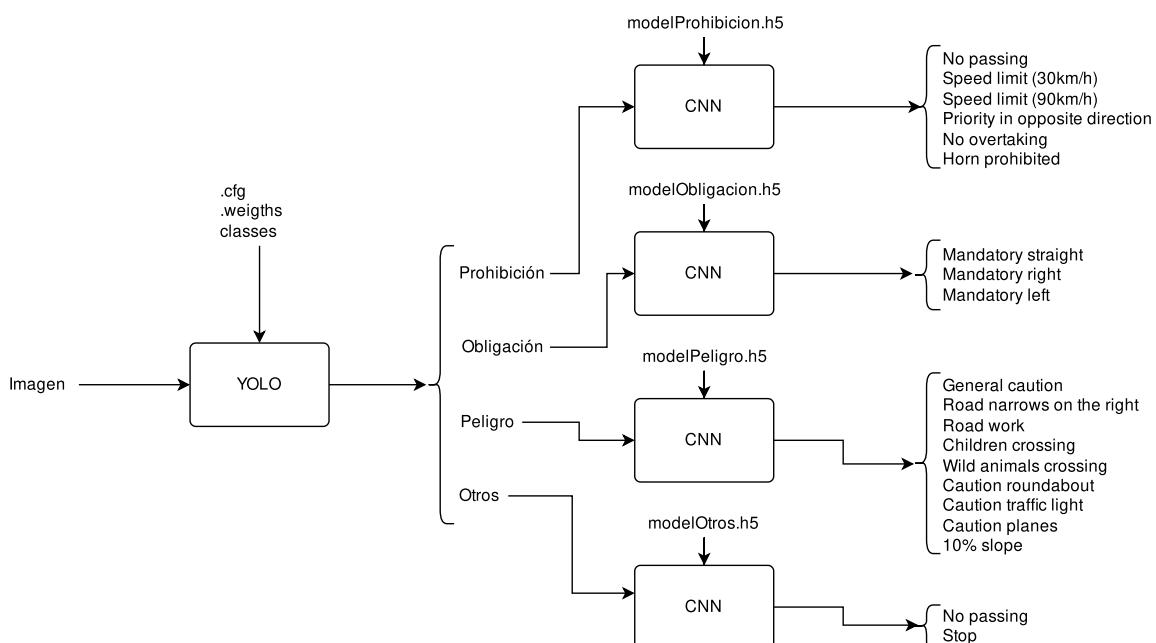


Figura 4.7: Ventajas de las diferentes tecnologías

El modelo CNN definido consta de varias capas. La estructura de capas que contiene el modelo son las siguientes:

- 1. Capa de convolución 2D con 16 filtros y tamaño de kernel 3x3, con función de activación ReLU.
- 2. Capa de convolución 2D con 32 filtros y tamaño de kernel 3x3, con función de activación ReLU.
- 3. Capa de MaxPooling 2D con tamaño de pool 2x2.
- 4. Capa de normalización por lotes (Batch Normalization).
- 5. Capa de convolución 2D con 64 filtros y tamaño de kernel 3x3, con función de activación ReLU.
- 6. Capa de convolución 2D con 128 filtros y tamaño de kernel 3x3, con función de activación ReLU.
- 7. Capa de MaxPooling 2D con tamaño de pool 2x2.
- 8. Capa de normalización por lotes (Batch Normalization).

- 9. Capa de aplanamiento (Flatten).
- 10. Capa densa (fully connected) con 512 neuronas y función de activación ReLU.
- 11. Capa de normalización por lotes (Batch Normalization).
- 12. Capa de Dropout con una tasa de dropout de 0.5.
- 13. Capa densa (fully connected) con 2 neuronas y función de activación softmax.

En total, las CNNs están formadas 13 capas, incluyendo las capas de convolución, capas de pooling, capas de normalización por lotes, capas densas y capas de dropout. Cada una de estas capas realiza las siguientes funciones:

- 1. Capa de convolución 2D: Realiza la convolución de la imagen de entrada con un conjunto de filtros para extraer características importantes de la imagen. Utiliza una función de activación ReLU para introducir no linealidad en la red.
- 2. Capa de convolución 2D: Similar a la capa anterior, realiza otra convolución con un conjunto de filtros diferentes para extraer más características de la imagen. También utiliza la función de activación ReLU.
- 3. Capa de MaxPooling 2D: Reduce la dimensión espacial de la salida de las capas anteriores mediante la reducción de la resolución espacial. Ayuda a disminuir la cantidad de parámetros y a aprender características más generales.
- 4. Capa de normalización por lotes (Batch Normalization): Normaliza los valores de activación de la capa anterior para acelerar el entrenamiento y mejorar la estabilidad de la red.
- 5. Capa de convolución 2D: Realiza otra convolución con más filtros para extraer características más complejas de la imagen.
- 6. Capa de convolución 2D: Realiza otra convolución con aún más filtros para capturar características más específicas de la imagen.
- 7. Capa de MaxPooling 2D: Reduce la dimensión espacial de la salida de las capas anteriores.
- 8. Capa de normalización por lotes (Batch Normalization): Normaliza los valores de activación de la capa anterior.
- 9. Capa de aplanamiento (Flatten): Transforma la salida de la capa anterior en un vector unidimensional, preparándola para la entrada a las capas densas.
- 10. Capa densa (fully connected): Capa de neuronas completamente conectadas donde cada neurona está conectada a todas las neuronas de la capa anterior. Utiliza la función de activación ReLU.
- 11. Capa de normalización por lotes (Batch Normalization): Normaliza los valores de activación de la capa anterior.
- 12. Capa de Dropout: Apaga aleatoriamente un porcentaje de las neuronas durante el entrenamiento para evitar el sobreajuste y mejorar la generalización del modelo.
- 13. Capa densa (fully connected): Capa final de clasificación con 2 neuronas y función de activación softmax. Produce las probabilidades de pertenencia a cada una de las subclases.

Fruto del entrenamiento hemos obtenido las matrices de confusión de cada CNN, en una matriz de confusión las filas representan las clases reales y las columnas representan las clases predichas por el modelo, donde cada celda de la matriz muestra la cantidad de ejemplos que pertenecen a una clase específica. La estructura general de este tipo de matrices es la siguiente 4.8 [cm,]

| | | Predicted Number | | | |
|---------------|---------|------------------|----------|-----|-----------|
| | | Class 1 | Class 2 | ... | Class n |
| Actual Number | Class 1 | x_{11} | x_{12} | ... | x_{1n} |
| | Class 2 | x_{21} | x_{22} | ... | x_{2n} |
| | . | . | . | . | . |
| | . | . | . | . | . |
| | . | . | . | . | . |
| Class n | | x_{n1} | x_{n2} | ... | x_{nn} |

Figura 4.8: Estructura general de las matrices de confusión

Donde el número total de falsos negativos (TFN), falsos positivos (TFP), verdaderos negativos (TTN) y verdaderos positivos son respectivamente (TTP) 4.9.

$$\begin{aligned}
 TFN_i &= \sum_{\substack{j=1 \\ j \neq i}}^n x_{ij} \\
 TFP_i &= \sum_{\substack{j=1 \\ j \neq i}}^n x_{ji} \\
 TTN_i &= \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{\substack{k=1 \\ k \neq i}}^n x_{jk} \\
 TTP_{all} &= \sum_{j=1}^n x_{jj}
 \end{aligned}$$

Figura 4.9: Fórmulas de cálculo

La categoría peligro se divide en 9 subclases: General caution, Road narrows on the right, Road work, Children crossing, Wild animals crossing, Caution roundabout, Caution traffic light, Caution planes y 10 % slope. En su matriz de confusión, encontrada en la figura 4.10, detectamos que no se ha producido ninguna detección falsa y que el número de verdaderos positivos es elevado, es decir, el modelo entrenado para las clases de peligro está teniendo un rendimiento muy bueno en términos de clasificación.

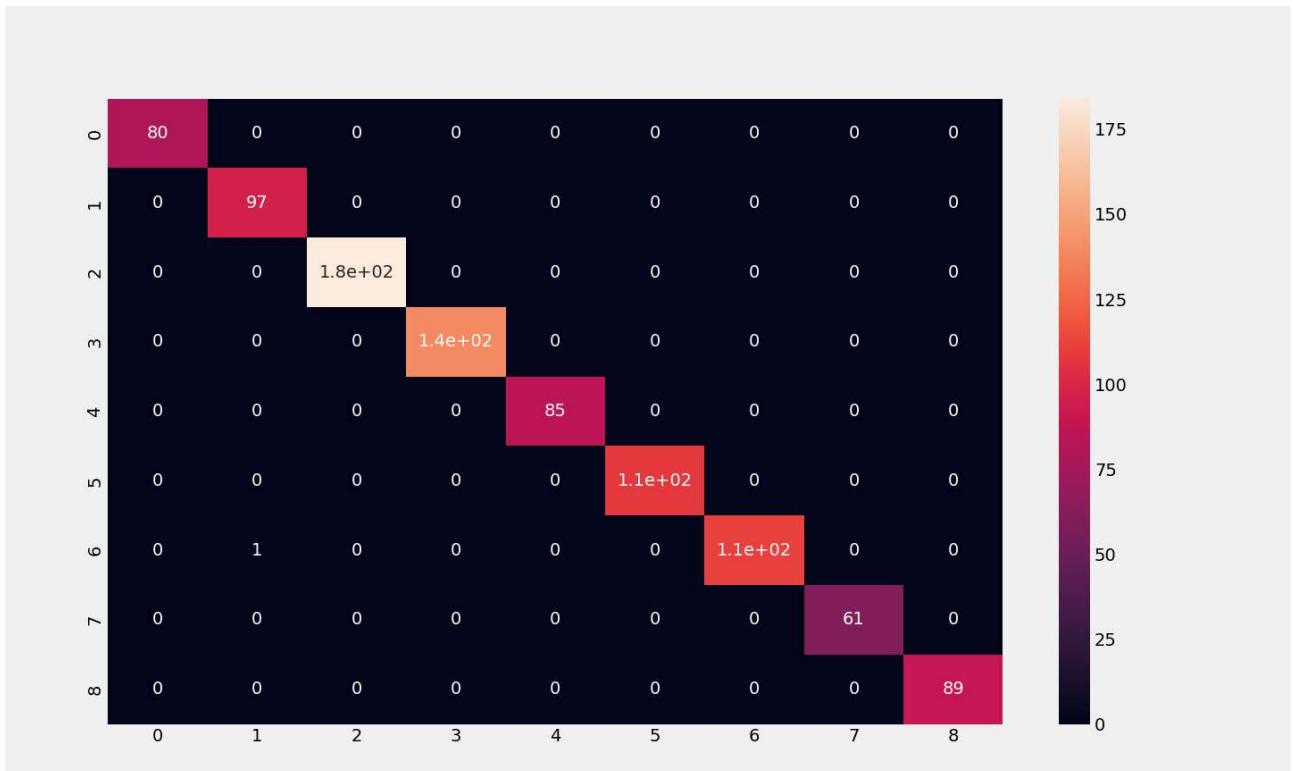


Figura 4.10: Matriz de confusión para la categoría peligro

La categoría obligación se divide en 3 subclases: Mandatory straight, Mandatory right y Mandatory left. En la figura 4.11 podemos ver su matriz de confusión, los resultados son análogos a los observados en las clases comentadas para la categoría peligro.

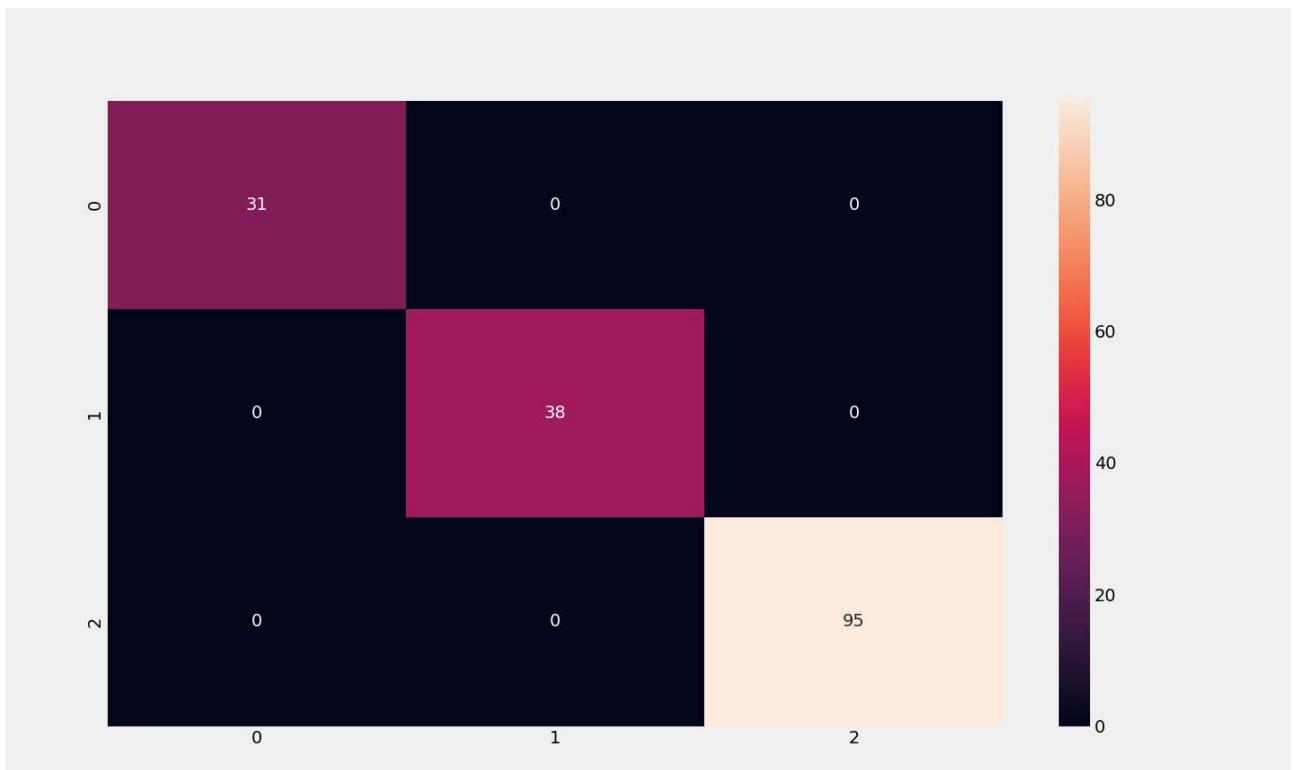


Figura 4.11: Matriz de confusión para la categoría obligación

La tercera de las clases que tenemos tras la detección en YOLO es prohibición, que se divide en 6 subclases: No passing, Speed limit (30km/h), Speed limit (90km/h), Priority in opposite direction, No overtaking, Horn

prohibited. Analizando los resultados de la figura 4.12 nos damos cuenta de que por primera vez nos encontramos detecciones falsas, sin embargo, la cifra es anecdótica comparada con las detecciones verdaderas.

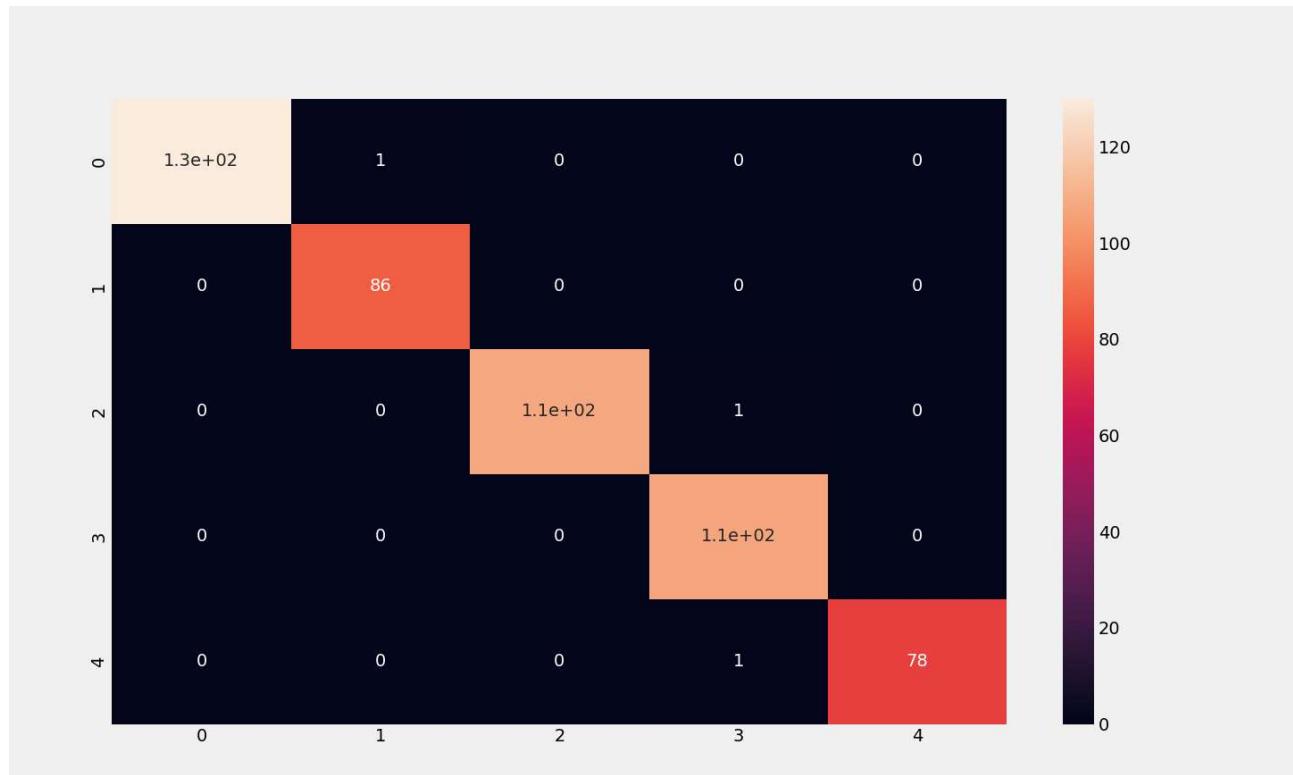


Figura 4.12: Matriz de confusión para la categoría prohibición.

Finalmente, la última clase otros se divide en 2 subclases: No passing y Stop y los resultados son equivalentes a los tres anteriores 4.13.

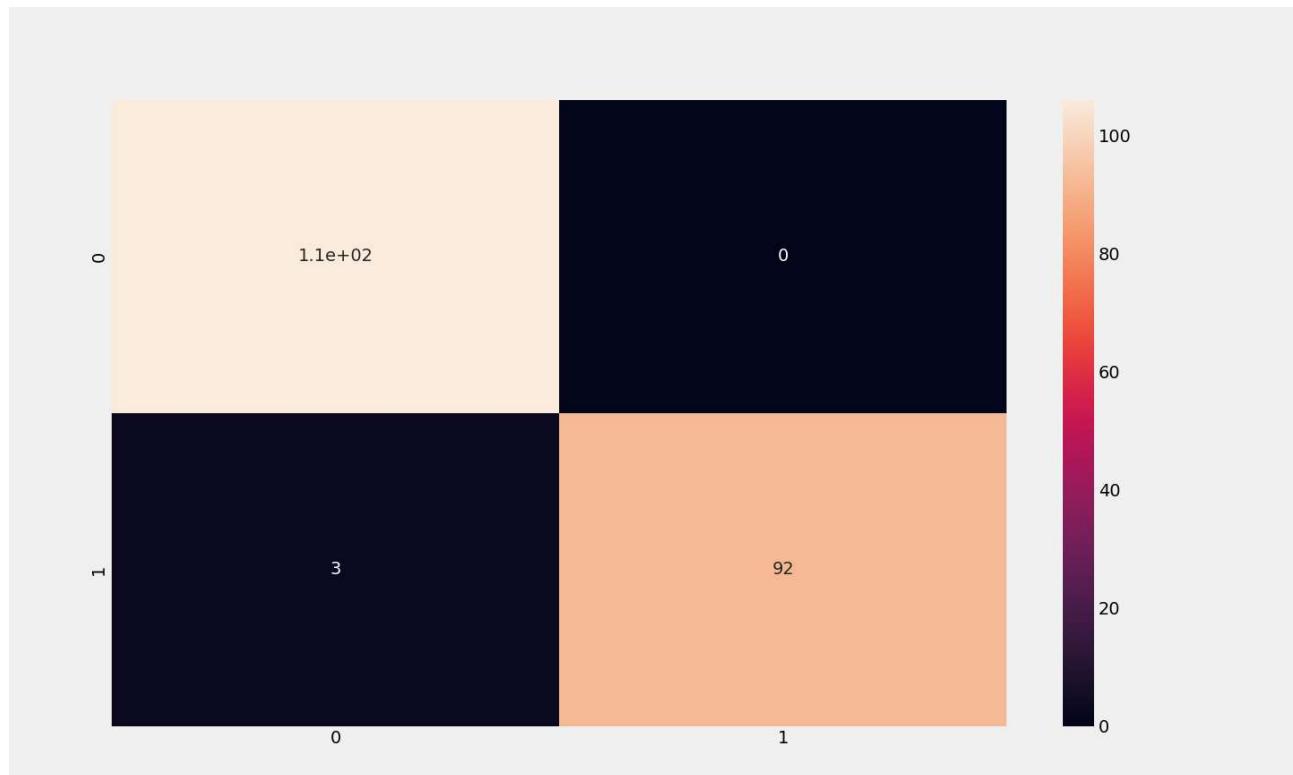


Figura 4.13: Matriz de confusión para la última categoría.

Llevando a la práctica la incorporación de las redes CNN veremos realmente si son fiables, por ello, procederemos a ver cómo se desenvuelve en el propio vehículo de Amazon. Montando una prueba con el coche y varias señales nos encontramos con los siguientes resultados 4.14, 4.15, 4.16, 4.17.

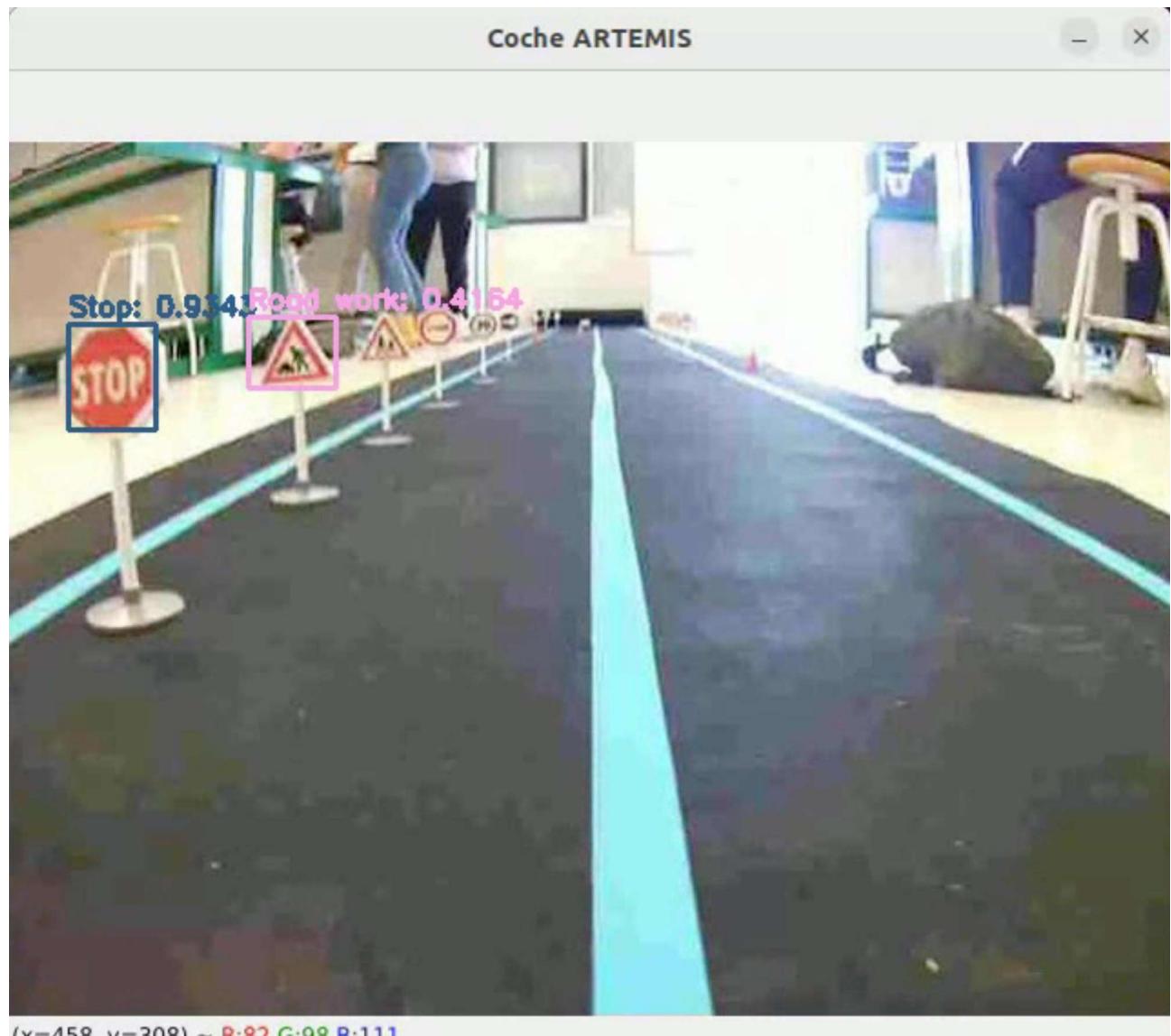


Figura 4.14: Resultados de la prueba (1).

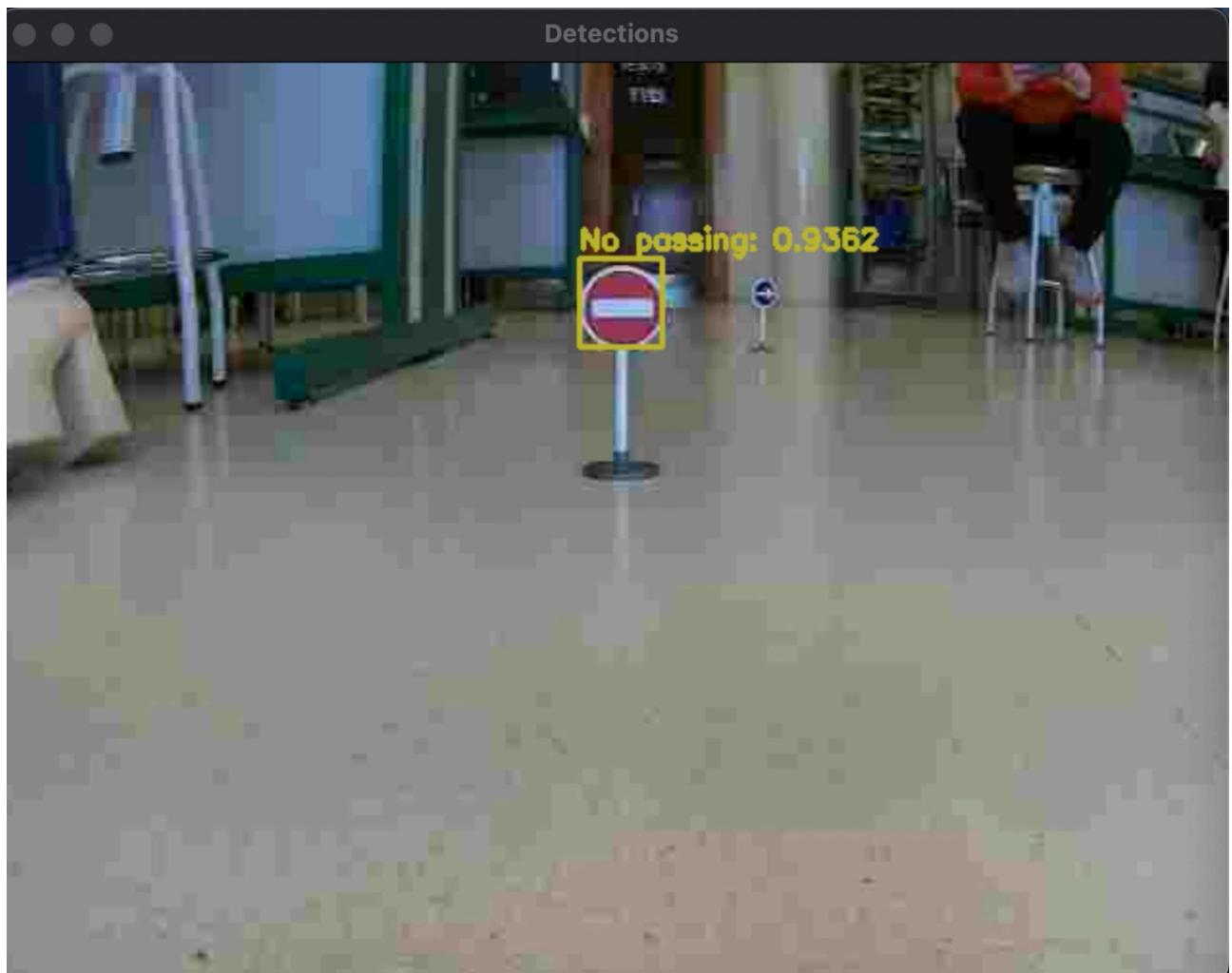


Figura 4.15: Resultados de la prueba (2).

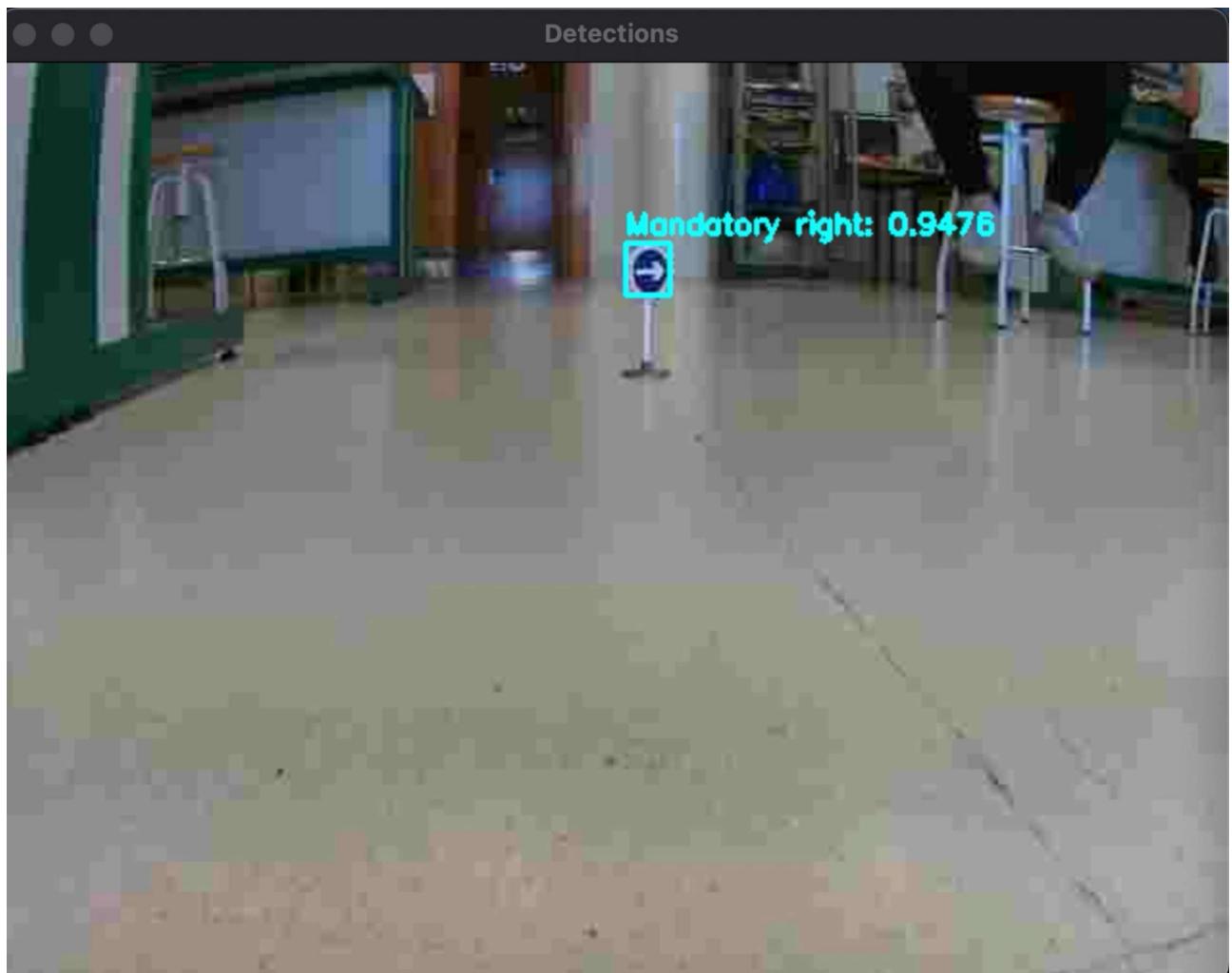


Figura 4.16: Resultados de la prueba (3).

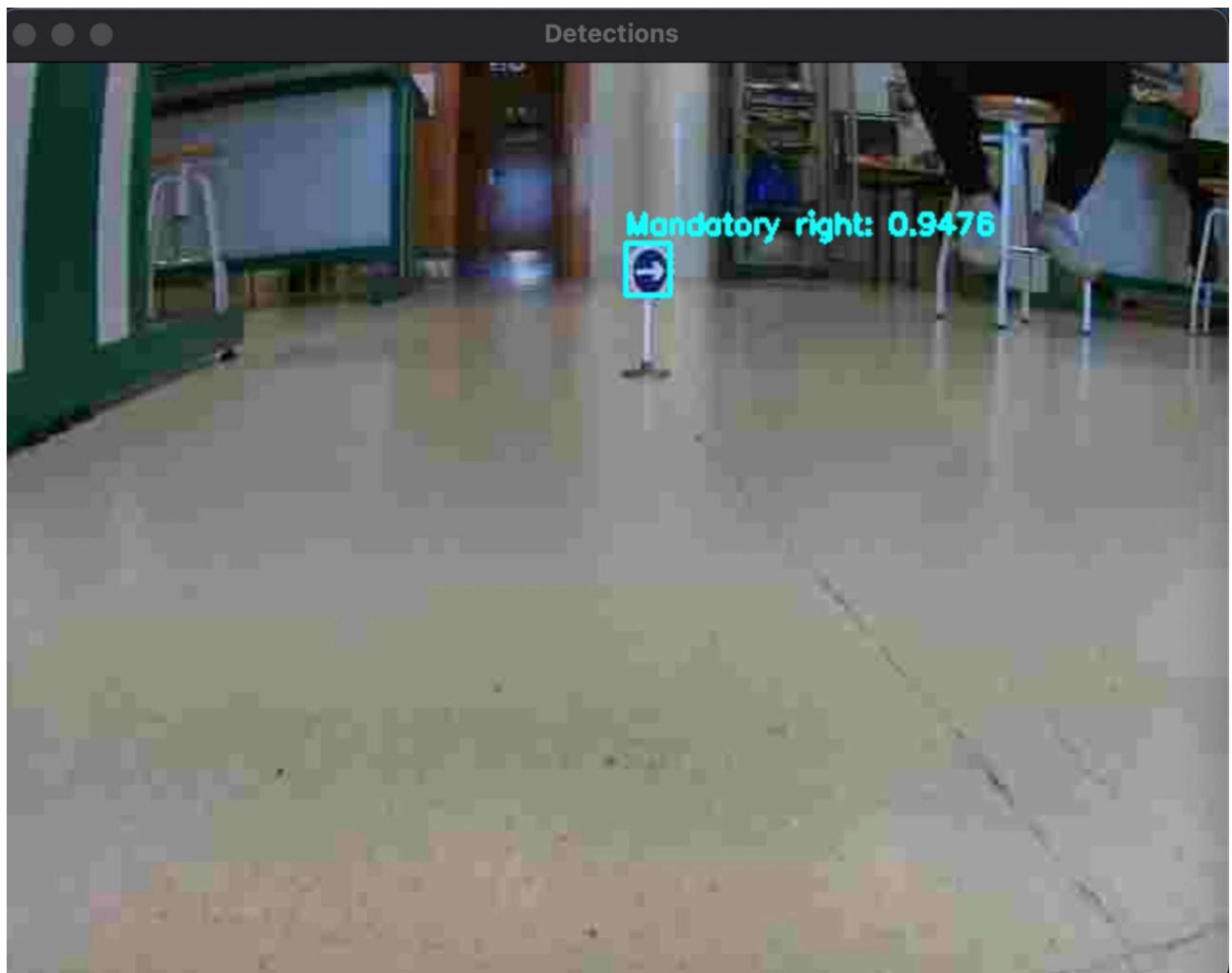


Figura 4.17: Resultados de la prueba (4).

Sin embargo, a pesar de quedar satisfechos con los resultados obtenidos, hemos encontrado ocasiones en las que no detecta bien cada señal, como podemos observar en la señal de limitación 30 km/h 4.18.



Figura 4.18: Fallo de detección.

Esto puede ser debido a que las condiciones de calidad de imagen que se obtienen de la cámara del propio vehículo no son exactamente las mismas que las imágenes con las que ha sido entrenado. De cara a líneas futuras sería interesante que el conjunto de imágenes creado para el entrenamiento, se diseñara a partir de vídeos grabados por coche.

En último lugar, según se puede constatar en el Anexo I apartado 6.2, también hemos utilizado una herramienta de etiquetado denominada LabelIMG <https://github.com/heartexlabs/labelImg>. Sería recomendable atender al manual de desarrollador incluido en dicho anexo para aprender a manejar o conocer más detalles acerca del sistema de inteligencia artificial.

4.3. Normativa

Pedir permiso para usar una banda de frecuencias es fundamental debido a varios motivos clave. En primer lugar, garantiza un uso ordenado y eficiente del espectro electromagnético, que es un recurso limitado. Al asignar permisos, se evita la interferencia y se asegura que múltiples dispositivos puedan operar sin afectarse mutuamente. Además, el control de frecuencias permite una planificación adecuada de las comunicaciones, facilitando la coexistencia de diversas tecnologías y servicios, como la telefonía móvil, la transmisión de datos y las emisiones de radio y televisión. Asimismo, la asignación de permisos ayuda a proteger la seguridad y privacidad de las comunicaciones al regular el acceso a determinadas bandas. Por último, la necesidad de pedir permiso promueve la responsabilidad y la rendición de cuentas por parte de los usuarios, fomentando un uso responsable y respetuoso del espectro electromagnético.

En nuestro proyecto, al estar ubicado en España, se ha tenido que pedir permiso al Ministerio de Asuntos Económicos y Transformación Digital.<https://sededitatid.mineco.gob.es/es-es>

Como en la tecnología LTE cada canal es de 15MHz, nos encontramos en una situación de banda estrecha. Es decir, deberemos pedir nuestros permisos mediante los pasos descritos del enlace anteriormente mencionado en el apartado Redes Radioeléctricas del Servicio Móvil y Fijo de Banda Estrecha". Este procedimiento se puede dividir por tanto en tres etapas:

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

4.3.1. 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 demandadas 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 (ver anexos), 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.

4.3.2. Envío de la solicitud

El Reglamento de uso del dominio público radioeléctrico aprobado por el Real Decreto 123/2017, de 24 de febrero, considera usos experimentales los destinados a efectuar pruebas técnicas o ensayos sobre propagación, utilización de nuevas bandas de frecuencia o demostraciones de nuevos servicios o tecnologías. Su duración está limitada a máximo seis meses. La solicitud de título habilitante (autorización individual) para el uso del dominio público radioeléctrico para la cobertura de eventos de corta duración se debe presentar ante esta Secretaría de Estado con una antelación de, al menos, diez días hábiles al comienzo del período de utilización solicitado.

Junto con la solicitud, se debe presentar una memoria técnica (fichero estructurado XML) que incluirá el período de utilización, una descripción del servicio a prestar, la red radioeléctrica, con las estaciones y equipos

que se pretenden utilizar, con indicación de sus características técnicas y ubicación. Dicha memoria técnica puede estar firmada por un técnico competente o en otro caso venir firmada electrónicamente por el titular responsable de la red o su representante legal.

Para generar la descripción de la red, se facilita la herramienta SM_GenXML, se completa el fichero descriptivo de la red y se realiza el proceso de firma del mismo.

La solicitud de título habilitante de uso de frecuencias para eventos de corta duración o para pruebas experimentales debe presentarse utilizando el procedimiento Solicitud Nuevas Redes Radioeléctricas Servicio Móvil y Fijo de banda estrecha correspondiente al conjunto de procedimientos relativos a “Redes Radioeléctricas del Servicio Móvil y Fijo de banda estrecha” disponibles en la sede electrónica: Respecto al punto “Anexado de documentos” del Manual indicado, en lo relativo al documento “Datos Adicionales No Estructurados del Proyecto” deberán facilitarse las fechas del evento, frecuencias alternativas a las grabadas en el fichero, rango de funcionamiento en frecuencias de los equipos y cualquier otra información necesaria o que pueda facilitar el estudio de la red por parte de la Administración Durante el proceso de generación de la descripción de red mediante la herramienta SM_GenXML se deberá adjuntar este fichero firmado electrónicamente.

Es decir, la solicitud, se presentará de forma telemática, deberá contener una memoria técnica en formato XML junto con la declaración responsable y el documento de datos adicionales (documentos que se han de adjuntar junto con el documento XML). Estos ficheros se pueden encontrar en los anexos.

4.3.3. Pago tasas del modelo 990

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

El artículo 3 en el que se describe la 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:

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

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.

En el presente proyecto, $B = 20 \times 10^3$ $S = 10 Km^2$, los valores de los coeficientes corresponden a los descritos dentro del escenario del apartado 1.7: Sistemas de comunicaciones móviles terrestres de banda ancha. Y por tanto:

- C1=1,25
- C2=1,375
- C3=1,6
- C4=1
- C5=0,7055

Una vez realizadas las cuentas, se obtiene un valor de importe aproximado de $T = 2332,07$ € anuales. Como es una red experimental, su uso está limitado a 6 meses, por lo que el pago de la tasa 990 es de 1166,038.

Capítulo 5

Costes

| Costes Fijos / CAPEX | | | |
|-----------------------------------|-------------|----------|---------------------|
| DESCRIPCIÓN | PRECIO | Cantidad | PRECIO TOTAL |
| ALQUILER TORRES TELEFONÍA (1 AÑO) | 15.000,00 € | 7 | 105.000,00 € |
| ESTACIONES BASE | 8.319,15 € | 7 | 58.234,05 € |
| ANTENAS | 1.571,39 € | 7 | 10.999,73 € |
| CORE | 3.697,40 € | 1 | 3.697,40 € |
| FIBRA | 6,00 € | 1288 | 7.728,00 € |
| SOBRECOSTES | | 20 % | 37.131,84 € |
| PRECIO TOTAL CAPEX | | | 222.791,02 € |

Tabla 5.1: Costes Fijos

| Alquiler de Servicios / OPEX | | | |
|------------------------------|------------|-------|--------------|
| DESCRIPCIÓN | EUROS/h | HORAS | PRECIO TOTAL |
| AWS (32vGPU) | 0,71 € | 4320 | 3.067,20 € |
| INGENIEROS | 15,65 € | 345 | 5.399,25 € |
| PRECIO TOTAL OPEX | | | |
| | 8.466,45 € | | |

Tabla 5.2: Costes mensuales

Nuestro diseño asegura una capacidad de dar servicio completo a 30 usuarios de manera simultánea por cada ENB. Esta cifra es tan reducida debido a las limitaciones del 4G en el canal de subida y además, nos ha sido imposible obtener especificaciones de otras estaciones base de otras empresas que tengan una capacidad mayor. Por este motivo se van a realizar varias estimaciones de usuarios, la primera se hace con cifra real de usuarios que somos capaces de dar servicios y el resto se harán en función del número de vehículos que recorren la autopista.

Las estimaciones se han calculado con estos parámetros:

- Velocidad media: 100 km/h
- Kilómetros entre ENB: 10 km
- Minutos en cada ENB: 6 minutos
- Minutos en un día: 1440 minutos

| Máximo (vehículos · km/año) | (vehículos · tramo/año) | (vehículos · tramo/día) | Vehículos por ENB |
|-----------------------------|-------------------------|-------------------------|-------------------|
| 10593226 | 105932260 | 290225,3699 | 1209 |
| Mínimo (vehículos · km/año) | (vehículos · tramo/año) | (vehículos · tramo/día) | Vehículos por ENB |
| 2.406.816 | 24068160 | 65940,16438 | 275 |
| Media (vehículos · km/año) | (vehículos · tramo/año) | (vehículos · tramo/día) | Vehículos por ENB |
| 6500021 | 65000210 | 178082,7671 | 742 |

Tabla 5.3: Medida de vehículos que circulan por el tramo

| Tarifa | Precio | Porcentaje de Usuarios |
|---------------------|---------|------------------------|
| Detección | 15,00 € | 30 |
| Detección + Control | 40,00 € | 70 |

Tabla 5.4: Tarifas y porcentaje de usuarios estimados

| Estimación | Descripción | Clientes Totales | Ingresos Totales |
|------------|--------------------------------|------------------|------------------|
| 1 | Máximo vehículos reales | 30 | 975,00 € |
| 2 | Máximo de vehículos | 1209 | 39.301,35 € |
| 3 | Media de vehículos | 742 | 24.115,37 € |
| 4 | Mitad de la media de vehículos | 371 | 12.057,69 € |
| 5 | Mínimo de vehículos | 275 | 8.929,40 € |

Tabla 5.5: Ingresos en función de las estimaciones

| AMORTIZACIÓN | | | | | |
|----------------------|--------------|--------------|--------------|--------------|--------------|
| PARTIDA | ESTIMACIÓN 1 | ESTIMACIÓN 2 | ESTIMACIÓN 3 | ESTIMACIÓN 4 | ESTIMACIÓN 5 |
| CAPEX | 222.791,02 € | 222.791,02 € | 222.791,02 € | 222.791,02 € | 222.791,02 € |
| OPEX | 8.466,45 € | 8.466,45 € | 8.466,45 € | 8.466,45 € | 8.466,45 € |
| INGRESOS MENSUALES | 975,00 € | 39.301,35 € | 24.115,37 € | 12.057,69 € | 8.929,40 € |
| BENEFICIOS MENSUALES | - 7.491,45 € | 30.834,90 € | 15.648,92 € | 3.591,24 € | 462,95 € |
| AMORTIZACIÓN TOTAL | - | 8 MESES | 15 MESES | 63 MESES | 482 MESES |

Tabla 5.6: Amortizaciones en función de las estimaciones

Capítulo 6

Rendimiento: Validación del servicio

Antes de comenzar con el rendimiento de la red, es necesario saber cómo esta está dispuesta y cuáles son las direcciones IP tanto de las interfaces interiores de la red 4G como de las interfaces correspondientes al usuario y al router. Esto se puede observar en la figura 6.1.

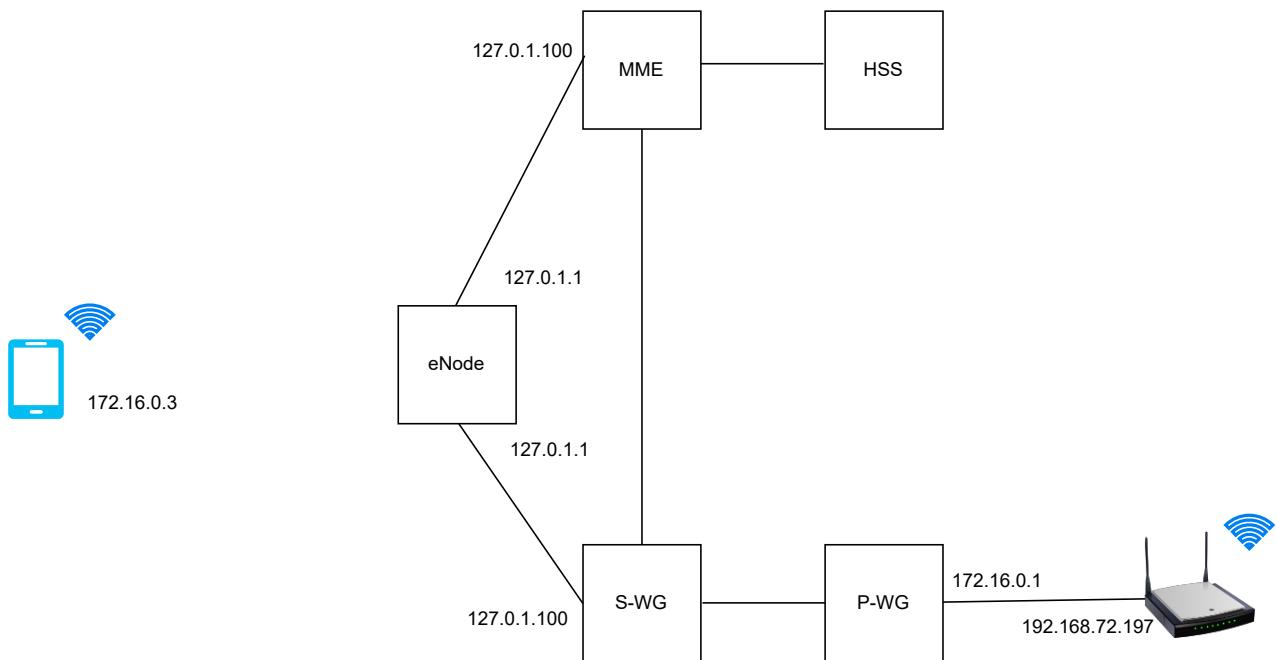


Figura 6.1: Esquema de la red. (Se ha usado *iproute* para saber las direcciones IP de las diferentes interfaces).

Una de las formas más simples de calcular el rendimiento de la red **extremo a extremo** es usando la herramienta *ping*. Esta herramienta mide la latencia enviando un paquete de datos desde un dispositivo a otro y midiendo el tiempo que tarda en ser recibido y respondido. La latencia se refiere al tiempo que tarda un paquete de datos en viajar desde un dispositivo a otro en una red.

Haciendo *ping* desde nuestro equipo de usuario ¹ (UE), con dirección IP 176.16.0.3, que corresponde a un dispositivo móvil que se encuentra en la misma sala y red que el ordenador de sobremesa en el que corre el eNodeB y el core obtenemos 6.2

¹Se ha hecho uso de la app “Ping IP”

```

Starting ...
PING 172.16.0.1 (172.16.0.1) 56(84) bytes of data.
Reply from 172.16.0.1: icmp_seq=1 ttl=64 time=66.0 ms
Reply from 172.16.0.1: icmp_seq=2 ttl=64 time=30.8 ms
Reply from 172.16.0.1: icmp_seq=3 ttl=64 time=22.0 ms
Reply from 172.16.0.1: icmp_seq=4 ttl=64 time=19.8 ms
Reply from 172.16.0.1: icmp_seq=5 ttl=64 time=20.9 ms
Reply from 172.16.0.1: icmp_seq=6 ttl=64 time=26.5 ms
----- 172.16.0.1 ping statistics -----
Packets: Sent = 6, Received = 6, Lost = 0 (0.0% loss),
Approximate round trip times in milli-seconds:
Minimum = 19.8ms, Maximum = 66.0ms, Average =
31.0ms

```

Figura 6.2: Salida del comando *ping*.

Aunque la latencia se puede ver afectada por la calidad de la red, la carga de tráfico, la distancia a la estación base, y otros factores, los valores obtenidos que se aprecian son bastante buenos. Se puede considerar que un buen valor de latencia en una red 4G se encuentra en el rango de 30 a 50 ms.

En lo que respecta al streaming de video (lo que se busca en este proyecto para que los videos sean recibidos y procesados con la menor latencia posible), una latencia inferior a 150 ms es adecuada para una experiencia de streaming fluida, mientras que una latencia superior a 200 ms puede causar retrasos en el video y una latencia superior a 300 ms puede generar interrupciones frecuentes o una experiencia de usuario negativa. En condiciones de laboratorio estas condiciones se cumplen satisfactoriamente.

Para seguir con el cálculo del rendimiento de nuestra red utilizamos la herramienta *Iperf3* para calcular el rendimiento extremo a extremo.

Iperf3 es una herramienta de línea de comandos que se utiliza para medir el rendimiento de la red, en términos de ancho de banda y calidad de servicio (QoS), entre dos dispositivos. En definitiva, *Iperf* mide la tasa de transferencia de datos entre dos dispositivos, y puede proporcionar información detallada sobre el rendimiento de la red, incluyendo la tasa de transferencia de datos, la latencia y la pérdida de paquetes. Además, utiliza un modelo cliente-servidor para dichas tareas. El cliente envía datos al servidor, y el servidor devuelve una respuesta al cliente. Durante esta comunicación, *Iperf* mide la tasa de transferencia de datos en ambas direcciones.

Esta comunicación se realiza haciendo uso del comando *iperf3* con el ordenador como servidor y el móvil (UE) con la Sim como cliente ²

```

iperf3 -s -p 5001
iperf3 -c 172.16.0.1 -P 50 -p 5001 -f m -t5

```

²Haciendo uso de la app *Aruba Utilities*

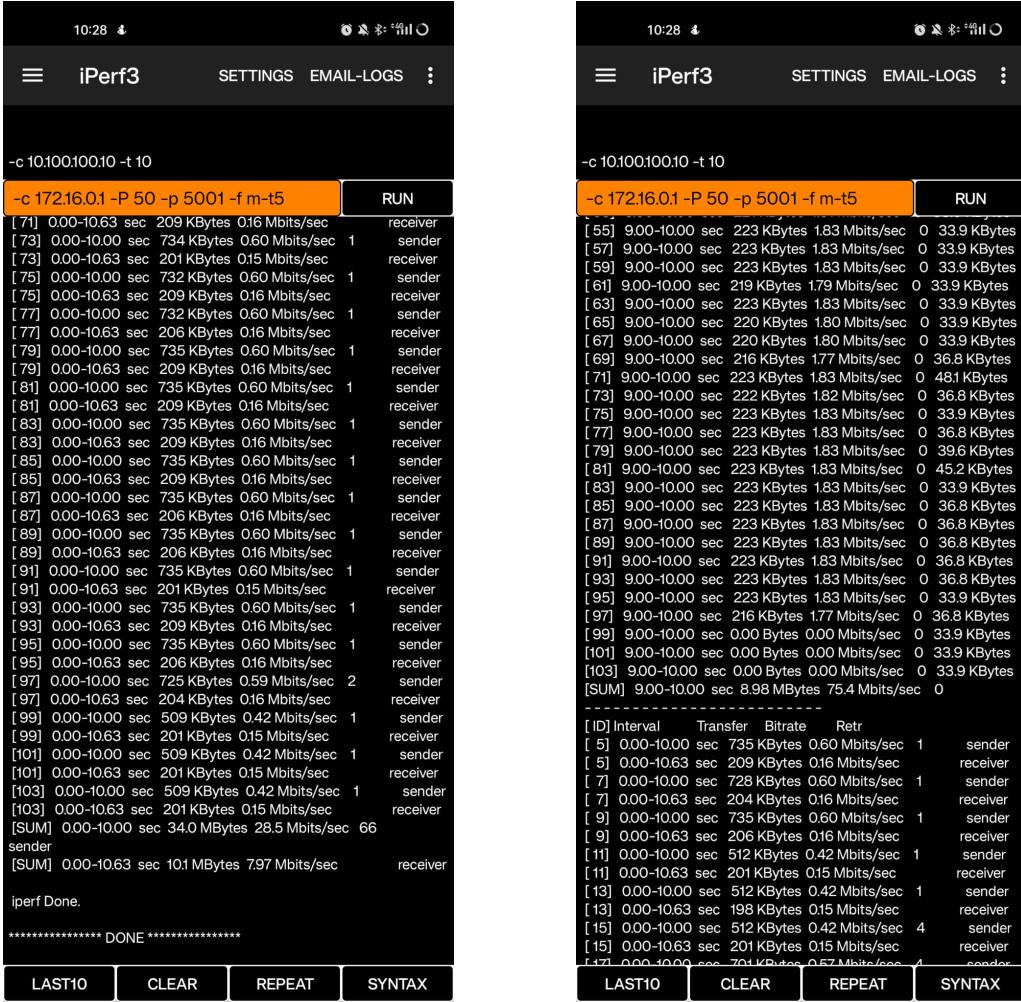


Figura 6.3: Salida del comando "iperf" en el lado cliente.

Listing 6.1: Salida del comando "iperf" en el lado servidor.

```

[SUM] 0.00-1.00 sec 684 KBytes 5.61 Mbits/sec
[SUM] 1.00-2.00 sec 766 KBytes 6.28 Mbits/sec
[SUM] 2.00-3.00 sec 1.07 MBytes 9.01 Mbits/sec
[SUM] 3.00-4.00 sec 1.07 MBytes 8.95 Mbits/sec
[SUM] 4.00-5.00 sec 1018 KBytes 8.34 Mbits/sec
[SUM] 5.00-6.00 sec 810 KBytes 6.64 Mbits/sec
[SUM] 6.00-7.00 sec 806 KBytes 6.60 Mbits/sec
[SUM] 7.00-8.00 sec 1.06 MBytes 8.92 Mbits/sec
[SUM] 8.00-9.00 sec 1.11 MBytes 9.29 Mbits/sec
[SUM] 9.00-10.00 sec 1.10 MBytes 9.24 Mbits/sec
[SUM] 10.00-10.63 sec 718 KBytes 9.28 Mbits/sec

```

La salida de *Iperf* depende de muchos factores, como la calidad de la señal, la carga de la red, la capacidad de los dispositivos involucrados, entre otros. En una red 4G, una salida razonable dependería de la configuración y las características específicas de la red y de los dispositivos utilizados. Sin embargo, como referencia general, en una red 4G se pueden esperar tasas de transferencia de datos de al menos 10-20 Mbps, y en algunos casos incluso mayores, dependiendo de la calidad de la señal y la configuración de la red. Por lo que los resultados obtenidos son buenos.

En cuanto al rendimiento de la red, según [Royuela et al., 2022], para un valor de 100 prb de download, obtenemos 7,5 bits/Hz. Extrapolando a nuestra situación donde prb tiene un valor de 25, obtenemos una tasa binaria de 18,75Mbps de forma teórica. Sin embargo, en las capturas, sólo obtenemos un valor máximo de 9,28, esto puede ser debido a varias causas, entre las que encontramos:

Sobrecarga de red: la velocidad de la red 4G puede variar según la cantidad de usuarios conectados simultáneamente en una determinada celda. Si la celda está congestionada debido a un alto número de usuarios, la velocidad de transferencia puede disminuir. Esto afecta en gran medida a nuestro caso, ya que en la misma sala están montadas 4 redes 4G a la vez.

Interferencia electromagnética: La interferencia de otros dispositivos electrónicos o estructuras físicas puede afectar la calidad de la señal y, por lo tanto, disminuir la velocidad de transferencia. Al encontrarnos en un laboratorio de docencia entre los que encontramos elementos como routers, antenas, ordenadores, etc. eso afecta considerablemente.

Continuando con el análisis del rendimiento de la red punto a punto, creamos un programa en python que utiliza el protocolo NTP para calcular el delay, el jitter y la latencia, utilizando como servidor NTP la dirección 130.206.3.166, que corresponde al servidor hora.rediris.es. En el Anexo ?? se observa el código empleado.

El protocolo NTP calcula la latencia y los otros parámetros midiendo el tiempo que tarda una petición de tiempo en ser enviada desde un cliente a un servidor NTP y la respuesta del servidor en ser recibida por el cliente, siendo los clientes las diferentes interfaces de nuestra red.

Para la interfaz s1c (127.0.1.1) obtenemos una latencia media de 0.057ms y un jitter de 0.0077ms. Para la interfaz MME (127.0.1.100) obtenemos una latencia media de 0.051ms y un jitter de 0.01369619369506836 s. Para la interfaz PGW (172.16.0.1) obtenemos una latencia media de 0.036ms y un jitter de 0.014568805694580078 s. Para la interfaz de la sim asignada (172.16.0.6) obtenemos una latencia media de 41.3ms y un jitter de 0.033ms. Obviamente estos datos son peores que los anteriores ya que las interfaces anteriores se encontraban en el mismo ordenador, mientras que esta actual se encuentra en un dispositivo móvil alejado aproximadamente un metro de la SDR.

Podemos observar que se obtiene una latencia lógica para una red 4G.

Listing 6.2: Salida del fichero python que calcula retardos entre interfaces.

```
tp1@tp1-desktop:~/servicios_cloud_deep_racer$ /bin/python3 /home/tp1/Escritorio/ntpyptp.py
Interfaz 127.0.1.1 - Latencia: 0.051 ms, Jitter: 0.01369619369506836 s
Interfaz 127.0.1.100 - Latencia: 0.036 ms, Jitter: 0.013817548751831055 s
Interfaz 172.16.0.1 - Latencia: 0.037 ms, Jitter: 0.014568805694580078 s
Interfaz 172.16.0.3 - Latencia: 107.0 ms, Jitter: 0.09294772148132324 s
Interfaz 172.16.0.4 - Latencia: 26.9 ms, Jitter: 0.012932538986206055 s
tp1@tp1-desktop:~/servicios_cloud_deep_racer$ /bin/python3 /home/tp1/Escritorio/ntpyptp.py
Interfaz 127.0.1.1 - Latencia: 0.055 ms, Jitter: 0.013919591903686523 s
Interfaz 127.0.1.100 - Latencia: 0.036 ms, Jitter: 0.013829231262207031 s
Interfaz 172.16.0.1 - Latencia: 0.034 ms, Jitter: 0.013852596282958984 s
Interfaz 172.16.0.3 - Latencia: 60.1 ms, Jitter: 0.04521584510803223 s
Interfaz 172.16.0.4 - Latencia: 22.2 ms, Jitter: 0.008130550384521484 s
tp1@tp1-desktop:~/servicios_cloud_deep_racer$ /bin/python3 /home/tp1/Escritorio/ntpyptp.py
Interfaz 127.0.1.1 - Latencia: 0.041 ms, Jitter: 0.01411890983581543 s
Interfaz 127.0.1.100 - Latencia: 0.039 ms, Jitter: 0.014233112335205078 s
Interfaz 172.16.0.1 - Latencia: 0.045 ms, Jitter: 0.01401066780090332 s
Interfaz 172.16.0.3 - Latencia: 63.6 ms, Jitter: 0.04976511001586914 s
Interfaz 172.16.0.4 - Latencia: 23.4 ms, Jitter: 0.009380340576171875 s
tp1@tp1-desktop:~/servicios_cloud_deep_racer$ /bin/python3 /home/tp1/Escritorio/ntpyptp.py
Interfaz 127.0.1.1 - Latencia: 0.038 ms, Jitter: 0.014064311981201172 s
Interfaz 127.0.1.100 - Latencia: 0.049 ms, Jitter: 0.014048337936401367 s
Interfaz 172.16.0.1 - Latencia: 0.045 ms, Jitter: 0.013978242874145508 s
Interfaz 172.16.0.3 - Latencia: 62.4 ms, Jitter: 0.048295021057128906 s
Interfaz 172.16.0.4 - Latencia: 23.6 ms, Jitter: 0.009762048721313477 s
```

En general, se espera que la latencia entre interfaces internas de una red 4G sea bastante baja, ya que la comunicación entre los diferentes componentes de la red es esencial para proporcionar un servicio de alta calidad a los usuarios. En términos generales, se espera que la latencia entre interfaces internas en una red 4G sea inferior a 10 milisegundos (ms).

Latencia media extremo a extremo de una red 4G son 40ms

Como última de herramienta de análisis realizamos test de velocidad de internet mediante el test disponible en google. Obtenemos una velocidad de descarga de 44.5 Mb/s y de subida de 21.1 Mb/s. Para un streaming de video en alta definición (HD) se esperan unas velocidades de bajada recomendadas de al menos 5 Mbps. y unas velocidades de subida recomendadas de al menos 5 Mbps.

6.1. Kernel de Baja Latencia

Para mejorar los resultados de rendimiento de la red instalamos el kernel de baja latencia de Linux y cambiamos las prioridades para que el núcleo tenga prioridad. Para ello utilizamos el comando:

```
sudo apt-get install -y linux-lowlatency.
```

Por último, reiniciamos el ordenador y elegimos al arrancar el Linux de baja latencia.

En general los resultados obtenidos son bastante parecidos a los obtenidos con el kernel normal, con alguna excepción en el que mejora dicha latencia y "Jitter".

Listing 6.3: Salida del fichero python que calcula retardos entre interfaces con kernel de baja latencia.

```
Interfaz 127.0.1.1 - Latencia: 0.041 ms, Jitter: 0.05975532531738281 s
Interfaz 127.0.1.100 - Latencia: 0.037 ms, Jitter: 0.05958867073059082 s
Interfaz 172.16.0.1 - Latencia: 0.036 ms, Jitter: 0.0597078800201416 s
Interfaz 172.16.0.2 - Latencia: 40.6 ms, Jitter: 0.019350528717041016 s
Interfaz 172.16.0.4 - Latencia: 45.8 ms, Jitter: 0.014258146286010742 s
```

Aún así se han realizado pruebas cuando la red no estaba muy congestionada, por lo que se espera que en una situación en la que haya más usuarios usando la red, el rendimiento sea mejor usando un kernel de baja latencia que uno normal.

Capítulo 7

Conclusiones

En conclusión, nuestro proyecto ha logrado establecer una conexión robusta y fiable mediante una red 4G, permitiendo una comunicación fluida entre el coche AWS DeepRacer y el sistema de reconocimiento de señales de tráfico basado en IA. La combinación de un rendimiento óptimo de la red, el uso de YOLO como algoritmo de IA y la capacidad de respuesta instantánea del coche ha resultado en un proyecto exitoso y prometedor.

La red 4G ha brindado una comunicación estable y confiable, mientras que el rendimiento de la red, incluyendo el ancho de banda, el jitter y la latencia mínima, ha permitido un streaming fluido y una respuesta instantánea del coche a las señales reconocidas. La elección de YOLO como algoritmo de IA ha sido acertada debido a su rendimiento sobresaliente en términos de velocidad y precisión en tiempo real. En términos de velocidad, precisión y capacidad de detección en tiempo real. Su enfoque basado en grid y su amplia adopción en la comunidad de IA respaldan su idoneidad para nuestra aplicación y nos brindan resultados confiables y efectivos en la detección de señales de tráfico. En conjunto, este proyecto destaca el potencial de la combinación de tecnologías avanzadas para mejorar la seguridad vial y la interacción entre sistemas inteligentes y el entorno físico.

Al analizar el tráfico promedio en la autopista, que es aproximadamente de 740 coches, y considerando que nuestra estación puede atender como máximo a 280 usuarios simultáneamente, se plantea la necesidad de explorar soluciones para mejorar la capacidad y el rendimiento de la red.

Otra consideración importante es el uso de edge computing. Actualmente, el procesamiento se realiza en la nube, pero es posible que trasladarlo al borde de la red (edge) pueda ofrecer mejores resultados. El procesamiento dentro del vehículo permite una respuesta más rápida y eficiente, ya que los datos se pueden analizar y actuar localmente sin tener que transmitirlos a servidores remotos. Al alquilar servidores, puedes aprovechar la infraestructura externa para realizar tareas que requieran mayor capacidad de procesamiento o almacenamiento, mientras que el procesamiento básico se puede realizar en el vehículo mismo.

En relación al alquiler de servidores, se ha tomado la decisión debido a la tendencia creciente de utilizar el procesamiento de datos dentro del vehículo. Esto significa que en el futuro es probable que los vehículos cuenten con mayor capacidad de procesamiento interno, lo que reduce la dependencia de servidores externos y permite un procesamiento más rápido y eficiente de los datos en el propio vehículo.

Capítulo 8

Anexo I: Manual del desarrollador

8.1. Infraestructura de la red

Para el proceso de creación de la red 4G, se usó el *software* srsRAN, una implementación de código abierto de una red de acceso radio (RAN) que utiliza *hardware* SDR. En nuestro caso, se ha utilizado una BladeRF 2.0 micro xA9 de Nuand con dos canales rx y tx MIMO.

Para ello, 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 un ordenador sin sistema operativo (proporcionado por la empresa).
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. Conexión de la Blade: Las antenas se deben conectar en los puertos tx1 y rx1 (configuración por defecto, sin embargo se podrían conectar a tx2 y rx2 haciendo los cambios correspondientes en los ficheros de configuración) con un grado de 90º entre ellas para evitar el efecto de *crosstalk*, como se muestra en la figura 8.1. La Blade se debe conectar a un puerto USB 3.0 que presente el ordenador ya que en un puerto USB 2.0 no se detectaría correctamente.

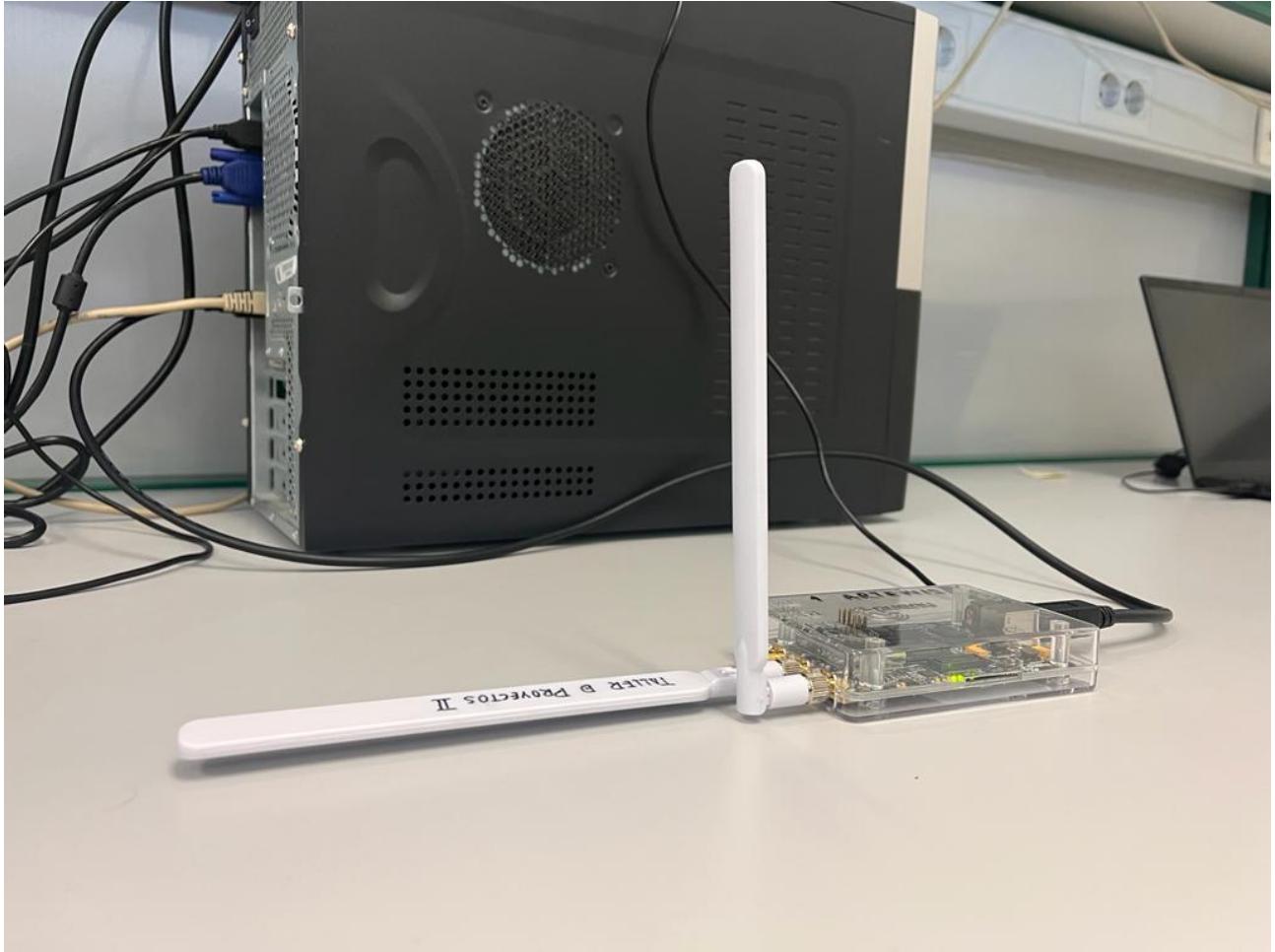


Figura 8.1: Dataset descargado con los ficheros TXT

5. 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
```

6. Modificación de los archivos de configuración que se encuentran en la ruta *root/.config/srsran/*. Los cambios se deben hacer como superusuario:

- **epc.conf**: contiene la configuración específica de un controlador de paquetes Evolved Packet Core (EPC) en una arquitectura de red LTE
- **enb.conf**: contiene la configuración específica de un nodo de banda base (eNodeB).
- **user_db.csv**: que contiene una base de datos de usuarios en formato tabular.

En el archivo **enb.conf** se cambiaron los valores de **MCC** y **MNC** que están disponibles en las hojas de datos de las SIMs (y en la propia tarjeta SIM), y se establecieron sus valores correspondientes (**901-70**) para que correspondan con el IMSI. También se modificó el valor de la frecuencia central, cambiando el valor de **d1_earfcn**, para ello se utilizó [ear,], estableciéndolo este valor 3050, lo que corresponde a un valor de frecuencia central *downlink* de 2650. Por último, se estableció el ancho de banda en 5MHz cambiando el valor de **n_prb** a **25**. Para mejorar el alcance de la red, se aumentó la ganancia de transmisor, cambiando el valor de **tx_gain** a 90.

En el archivo **epc.conf** se modificaron los valores de **MCC** y **MNC** (igual que en el caso anterior) 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 (poner todo a ceros, aunque cada vez que se levanta la red cambia automáticamente
),
7 (QCI), dynamic (IP_alloc)
```

7. Ahora que tenemos conectividad entre el enb y el epc, necesitamos que estos la tengan para el exterior, por lo que necesitamos configurar el ordenador (que ejecuta el núcleo) para que reencamine los paquetes a través de la interfaz de red. Para eso ejecutamos el siguiente comando.

```
srepcc_if_masq.sh enp0s25
```

8. Una vez está la red configurada, es hora de levantarla, ejecutamos:

```
srsepc epc.conf
srsenb enb.conf
```

9. 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**, diseñado para facilitar la comunicación entre dispositivos (en nuestro caso la SDR y el coche) intercambiando mensajes MQTT. Este servidor facilita la integración del proyecto, facilitando el monitoreo y control remoto del coche durante su funcionamiento. Para su instalación en el ordenador (que actuará como servidor), se ejecutan los siguientes comandos:

```
sudo apt-get update
sudo apt-get install mosquitto
```

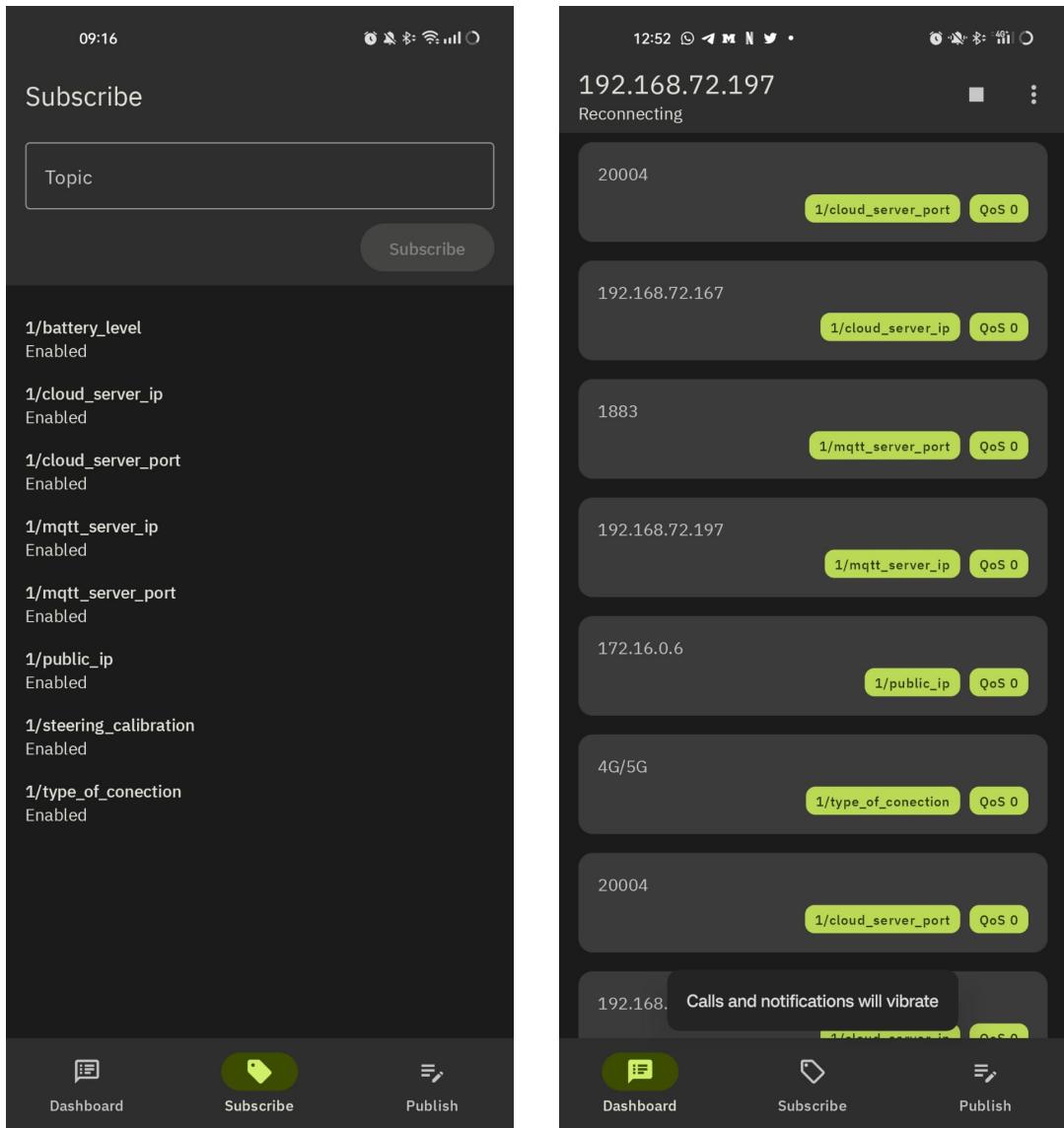
10. Modificamos el fichero de configuración /etc/mosquitto/mosquitto.conf con las siguientes líneas:

```
persistence true
persistence_location /var/lib/mosquitto/
allow_anonymous true
listener 1883 10.0.128.176
bind_interface enp0s25
log_type information
log_type warning
log_type error
```

11. Una vez configurados los ficheros, reiniciamos el servicio mosquitto para que se apliquen los cambios:

```
sudo systemctl restart mosquitto
```

12. Para instalar el servicio en el móvil (que actuará como cliente) usamos la aplicación MyMQTT. Para comprobar el tránsito de mensajes que el coche envía nos suscribimos a los tópicos básicos que se especifican en el manual de usuario del coche. Para ello, habrá que poner el ID del vehículo y el tópico al que nos queremos suscribir (Ver figura 8.2a). Si la comunicación es exitosa, se debería observar algo similar a la imagen 8.2b:



(a) Suscripciones a los tópicos desde el móvil

(b) Información mostrada al iniciar la conexión

Figura 8.2: Capturas de pantalla de la aplicación móvil MyMQTT para cliente suscrito

13. Tras configurar tanto el servidor como el cliente mosquitto, lanzamos el servicio y comprobamos su estado usando los comandos:

```
sudo systemctl start mosquitto
sudo systemctl status mosquitto
```

14. Para comunicarnos con el coche, publicaremos los tópicos que se indican en el manual de configuración del coche, como podemos ver en la imagen 8.3:

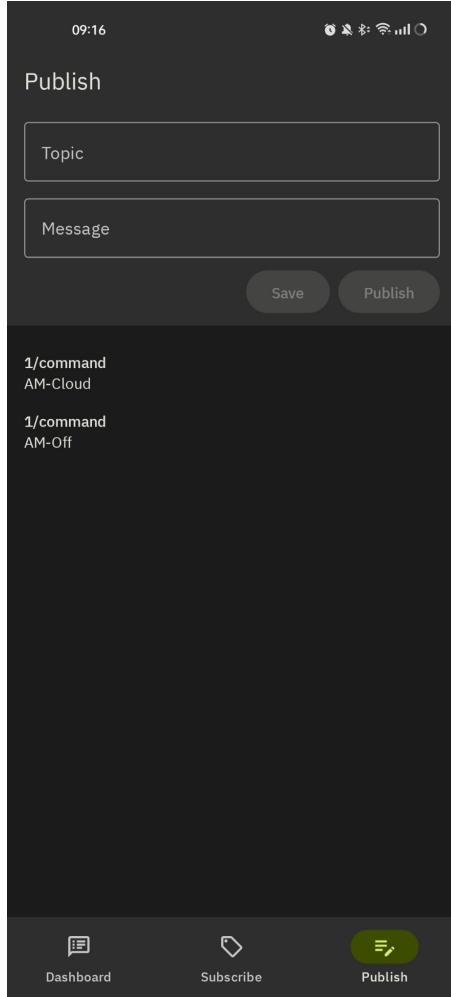


Figura 8.3: Publicación de comandos desde el móvil

8.2. 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 acceder al directorio *TallerDeProyectos2_G1* que se nos haya facilitado mediante un USB u otro medio, el cual contiene todo el software desarrollado por el equipo.

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 conjuntos de imágenes.

8.2.1. Detección y seguimiento de señales

Accediendo a la primera carpeta **1_YOLOV3** nos encontraremos con todos los scripts encargados de ejecutar el algoritmo. A su vez lo tenemos dividido en dos secciones: **OnlyYolo** y **YoloAndCNN**. En la primera, **OnlyYolo**, 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.

En la segunda sección denominada **YoloAndCNN** tendremos todos los ficheros más complejos que se han desarrollado para detectar cada señal de manera específica. Estos aplican al algoritmo de la primera sección una CNN (Red Neuronal Convolucional) para, por ejemplo, precisar que la señal de otros se trata de una señal de stop. Nos encontramos con los siguientes ficheros:

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. **artemis_autonomous_car.py**: script encargado de la conducción autónoma del coche.
5. **trainingCNN_byClasses.py**: script encargado de entrenar cada una de las CNN que utiliza el algoritmo de detección. Genera cuatro ficheros model.h5 que contienen la estructura y los pesos aprendidos tras el proceso de entrenamiento.
6. **cocheRealtimeDemo.py**: script encargado de manejar el vehículo de forma manual, en el cual hará detección de clases específicas de señales de tráfico y tomará decisiones en función de cada una de ellas. Este script está preparado para hacer unas operaciones concretas para realizar una demo en la presentación del trabajo. Se podría modificar para hacer lo que uno desee en función de cada señal.

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**:

- **imagen_yolov3.py**: 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.
- Directorio **models**: directorio que contiene los ficheros en formato h5 con la estructura de entrenamiento de la CNN para las clases obligación, prohibición, peligro y otros.
- **classes.txt**: diversos ficheros en formato txt contienen la descomposición específica de cada grupo de señales.

A modo de manual, se puede comenzar poniendo en marcha los scripts encargados de utilizar únicamente YOLO, es decir, ejecución de **OnlyYolo**. 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 8.4

```
python3 imagen_yolov3.py
```

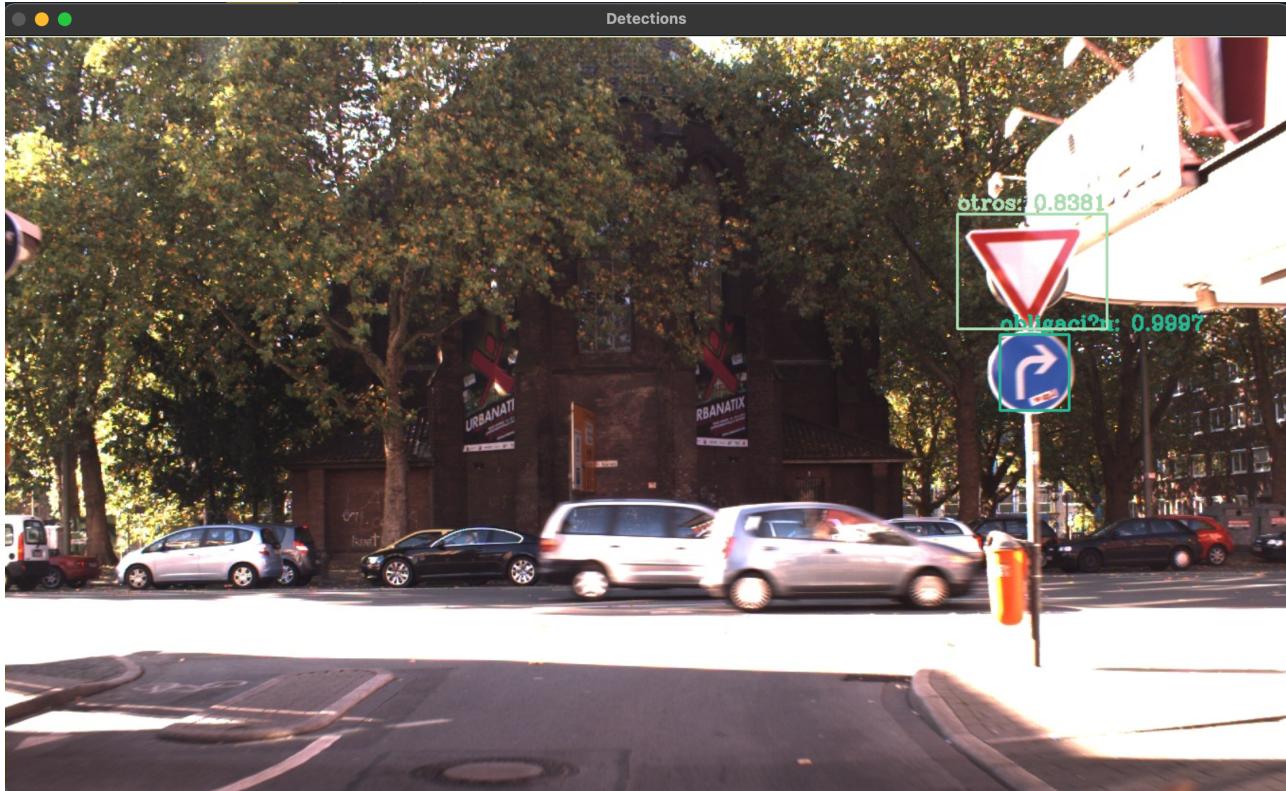


Figura 8.4: Detección de una señal

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 8.5

```
python3 video_yolov3.py
```

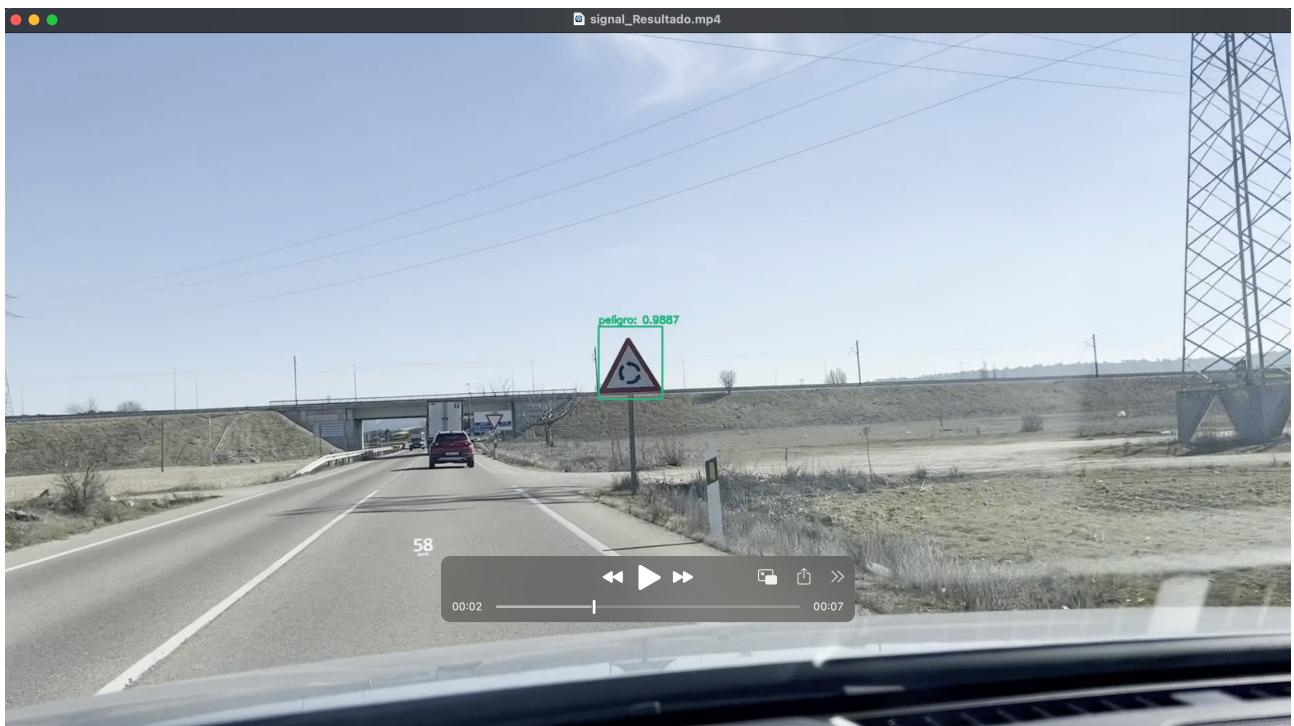


Figura 8.5: Detección de una señal real capturada por nosotros

Asimismo, podremos procesar video en tiempo real procedente de la cámara o webcam de nuestro propio ordenador, figura 8.6. A través del script **camara_yolov3.py** podremos ponerlo en marcha:

```
python3 camara_yolov3.py
```



Figura 8.6: 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 8.7.

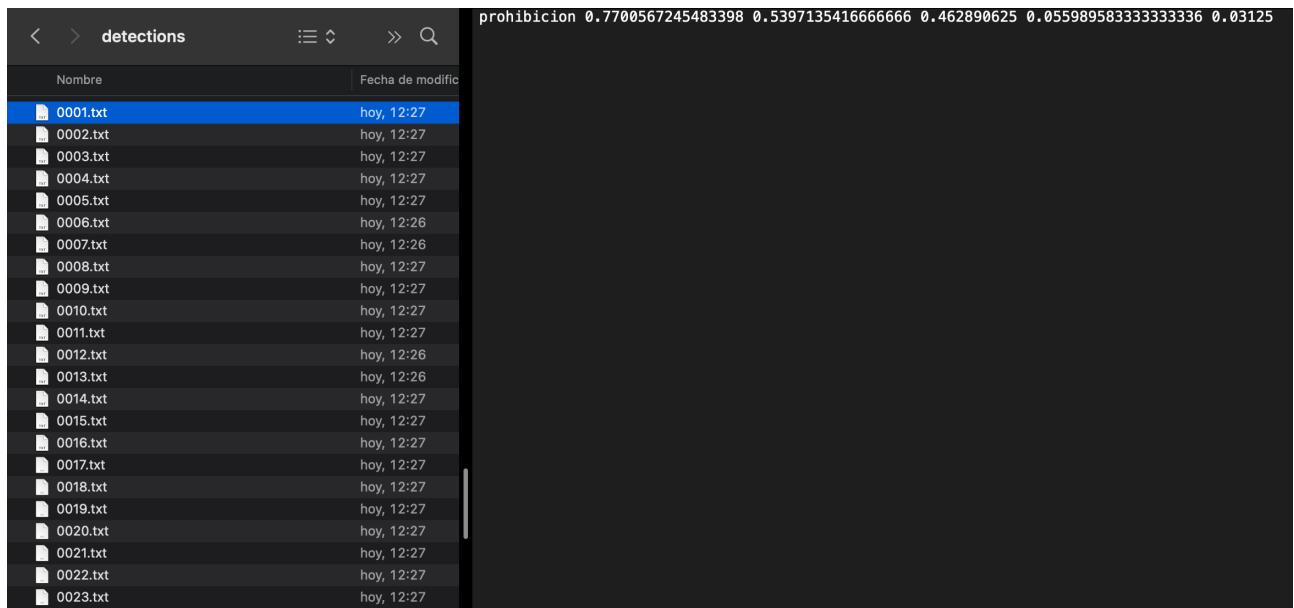


Figura 8.7: Ejemplo de rendimiento con *medirRendimientoRed = True*

De igual forma que hemos puesto en marcha los ficheros encargados de procesar imágenes, video y la cámara frontal, podríamos hacerlo para los scripts responsables de ejecutar el algoritmo de YOLO junto con la CNN, es decir, los ficheros **Imagen_yolov3CNN.py**, **video_yolov3CNN.py** y **camara_yolov3CNN.py** del directorio '**YoloAndCNN**'. A continuación, podemos visualizar un ejemplo de la ejecución sobre el propio vehículo 8.8:

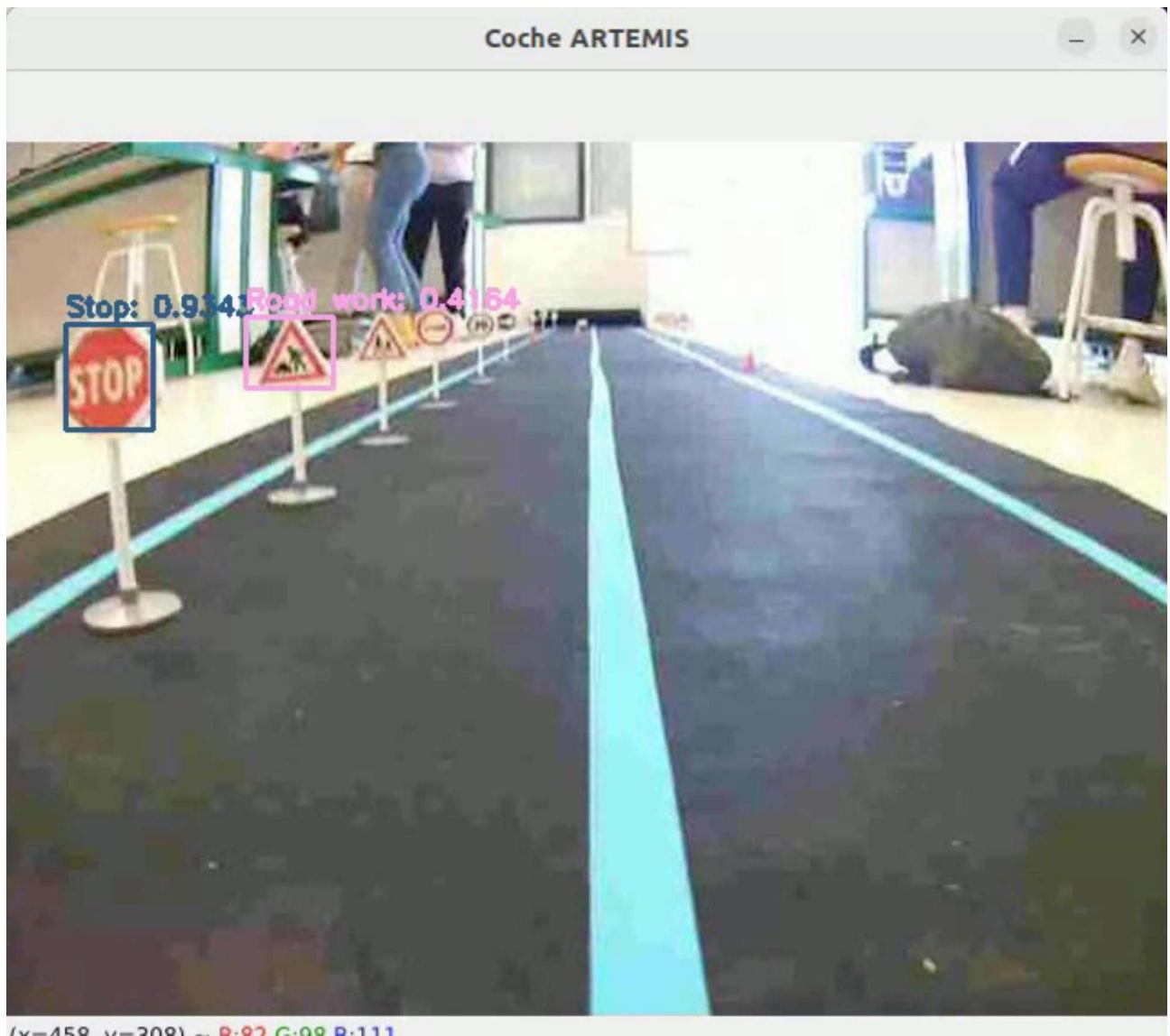


Figura 8.8: Ejemplo de ejecución sobre el propio coche

Y un ejemplo sobre la cámara frontal del ordenador en el que se muestra la detección de varias señales 8.9:

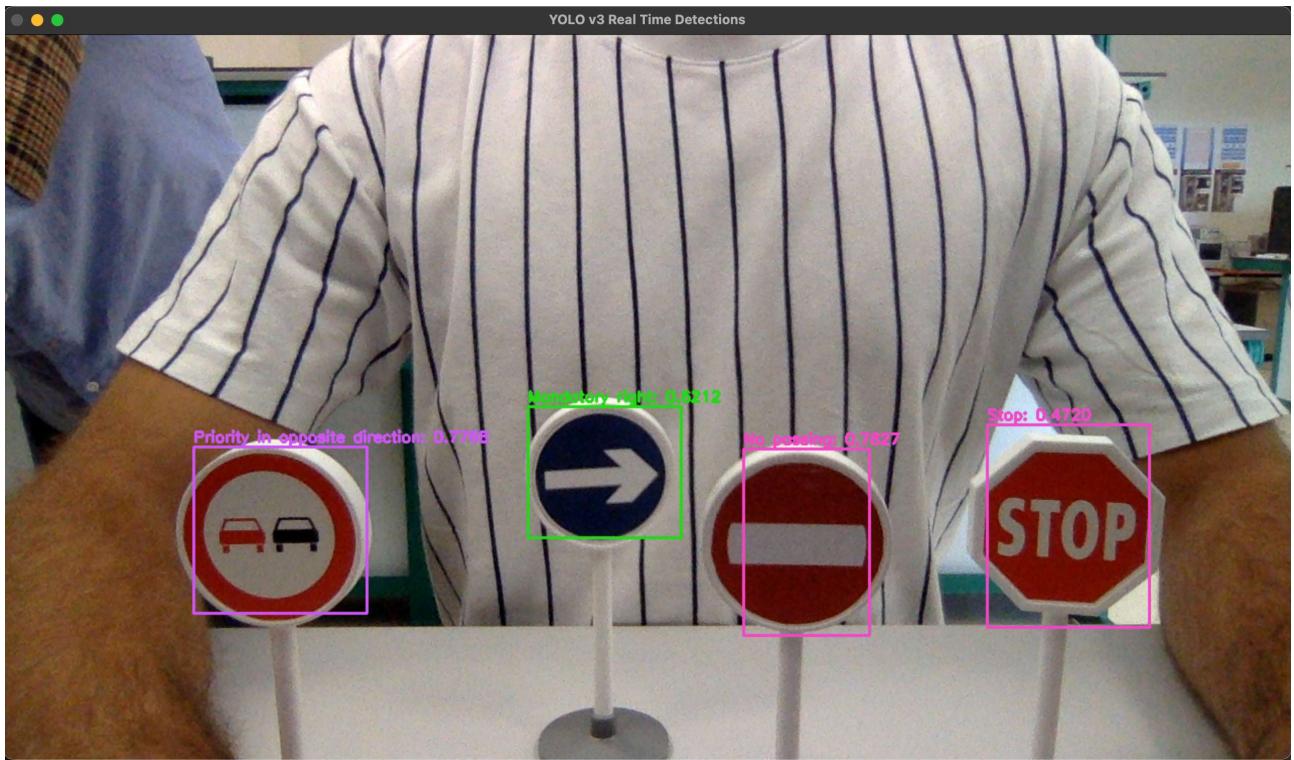


Figura 8.9: Ejemplo sobre la cámara frontal del ordenador

El script encargado de entrenar las cuatro CNNs que utilizamos en nuestro esquema de inteligencia artificial es **trainingCNN_byClasses.py**. Mediante un dataset estructurado por grupos y preparado con numerosas imágenes de cada una de las señales de tráfico de juguete con las que contamos en el laboratorio, podremos entrenar las CNNs. Obtendremos los mencionados ficheros model.h5 que se utilizan en el almacenamiento de modelos de aprendizaje automático con la biblioteca Keras de Tensorflow. Asimismo, podremos visualizar las matrices de confusión resultado de cada uno de los entrenamientos, tal y como podemos observar en la siguiente imagen 8.10:

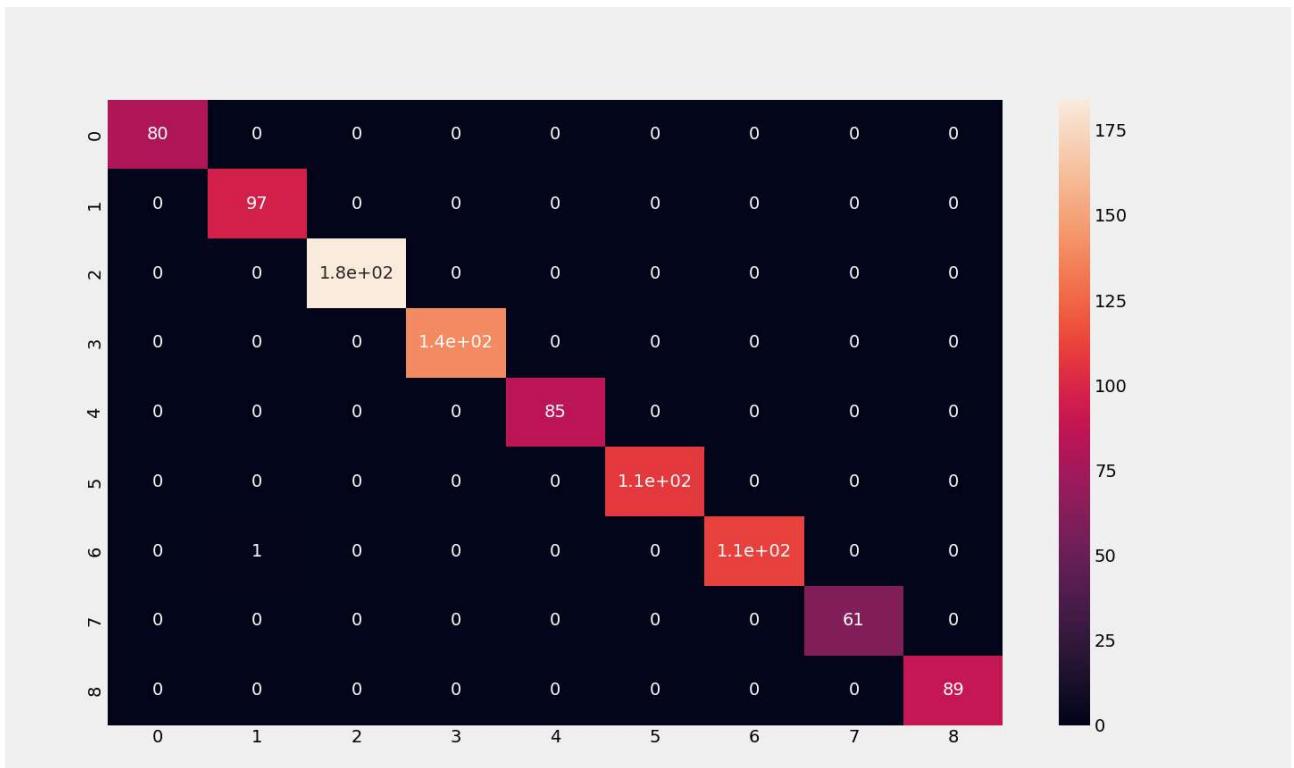


Figura 8.10: Matrices de confusión

Finalmente nos encontramos con el fichero encargado de manejar el vehículo **cocheRealtimeDemo.py**. Dicho script no ha sido desarrollado íntegramente por nosotros, sino que se nos ha proporcionado por parte del profesorado, por lo que nosotros lo hemos adaptado a nuestras necesidades. Este script requiere en primer lugar especificar la dirección IP y puerto con el que se va a conectar uno al vehículo, por ello, uno cuando proceda a ejecutar este código debe prestar atención a la variable `server_address` y modificarla según le convenga. Está preparado para la presentación de una demo del proyecto el día de la presentación, así pues, se podría configurar una acción a realizar por el coche según la detección de una señal.

8.2.2. 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/heartexlabs/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:

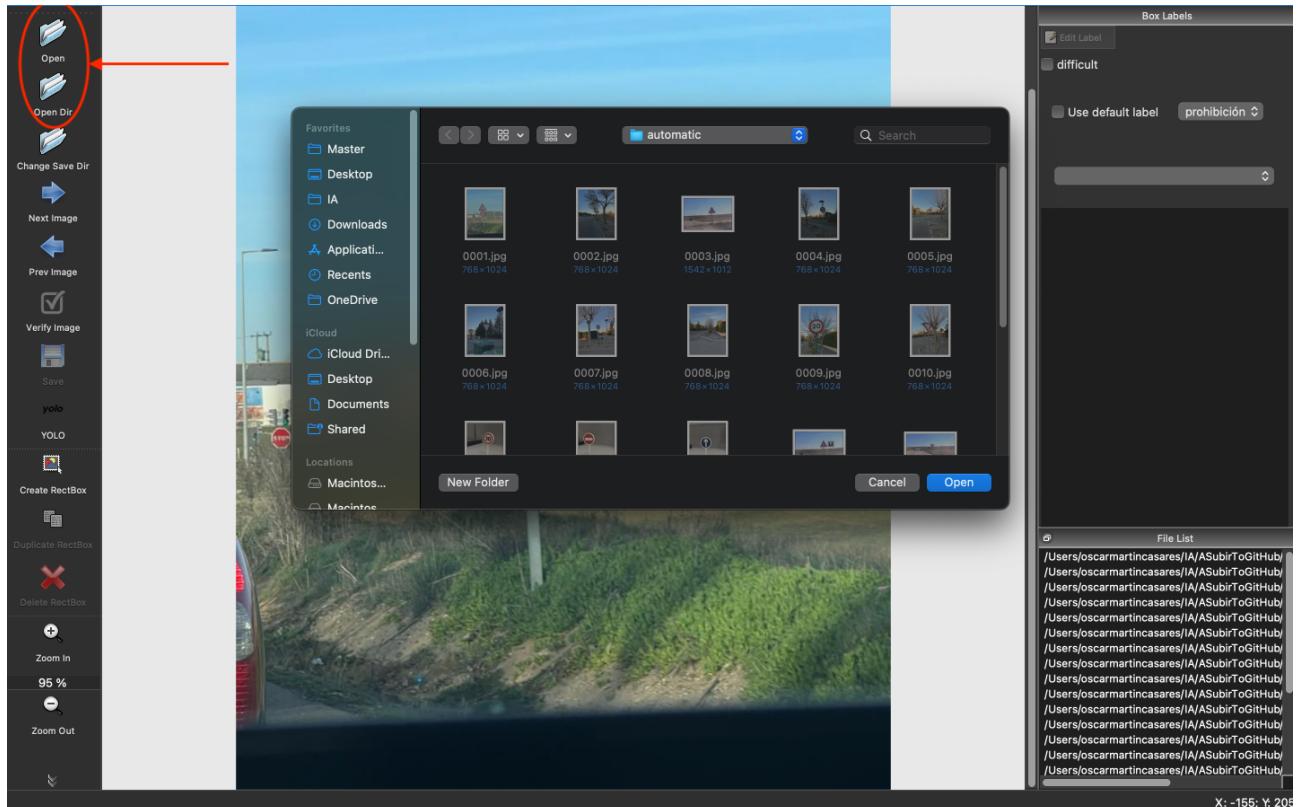


Figura 8.11: Etiquetado de una señal

Debemos asegurarnos de que el formato en el que se va a producir el etiquetado debe ser únicamente **YOLO**

(ver figura 8.12. Pulsando sobre el ícono mostrado podremos ir intercambiando entre diferentes formatos, ya que esta herramienta es compatible con varios.



Figura 8.12: 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 8.13



Figura 8.13: Cuadros delimitadores

Al guardar la imagen se nos creará un fichero en formato **TXT** que contendrá la información de etiquetado (Figura 8.14) 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**.

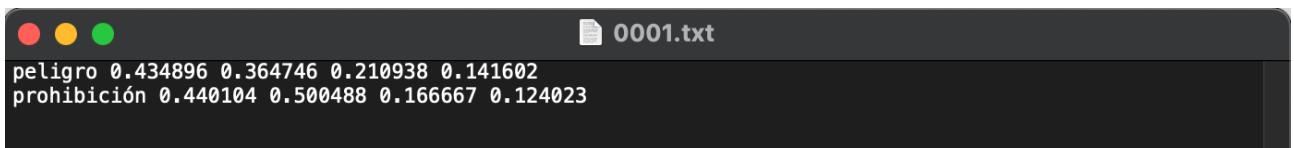


Figura 8.14: Fichero con la información de etiquetado

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
```

8.2.3. 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:

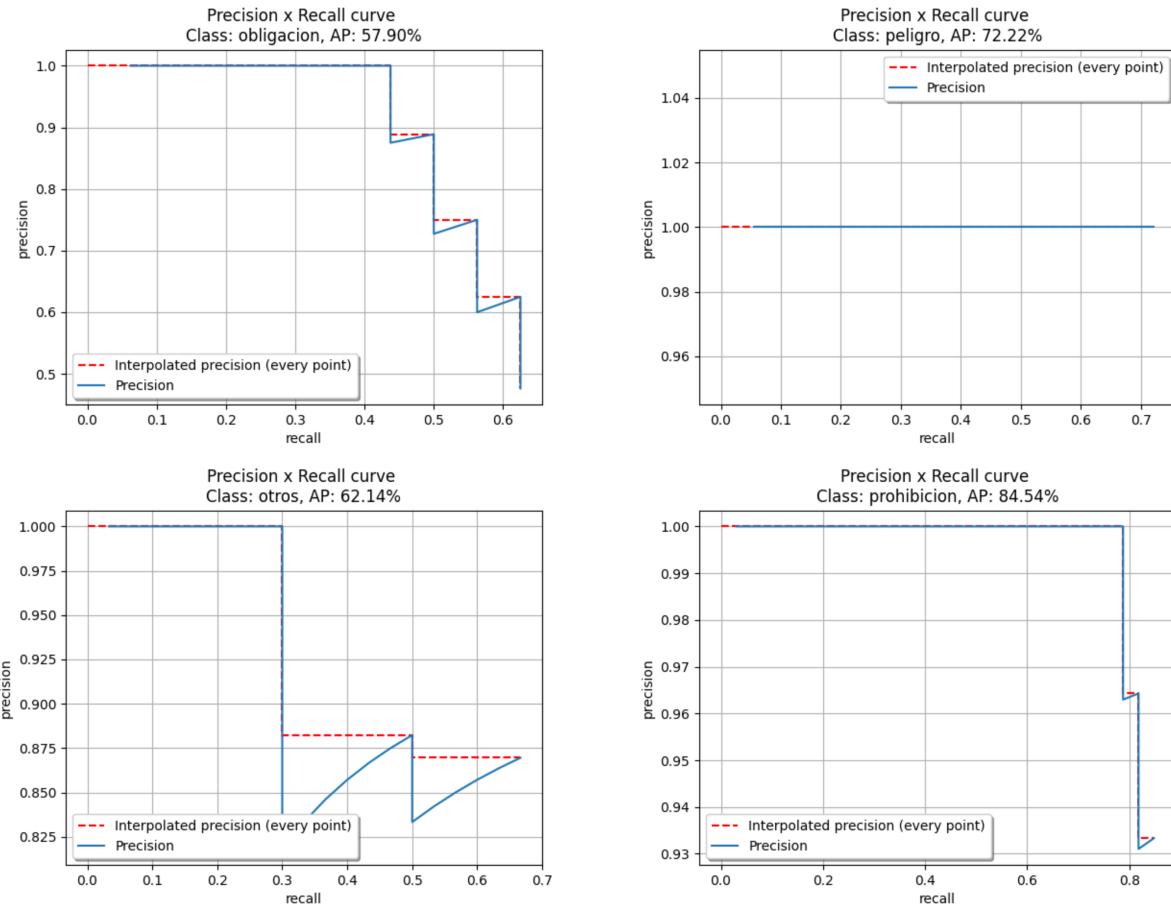


Figura 8.15: Medida de rendimiento

8.2.4. 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 8.16 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*:

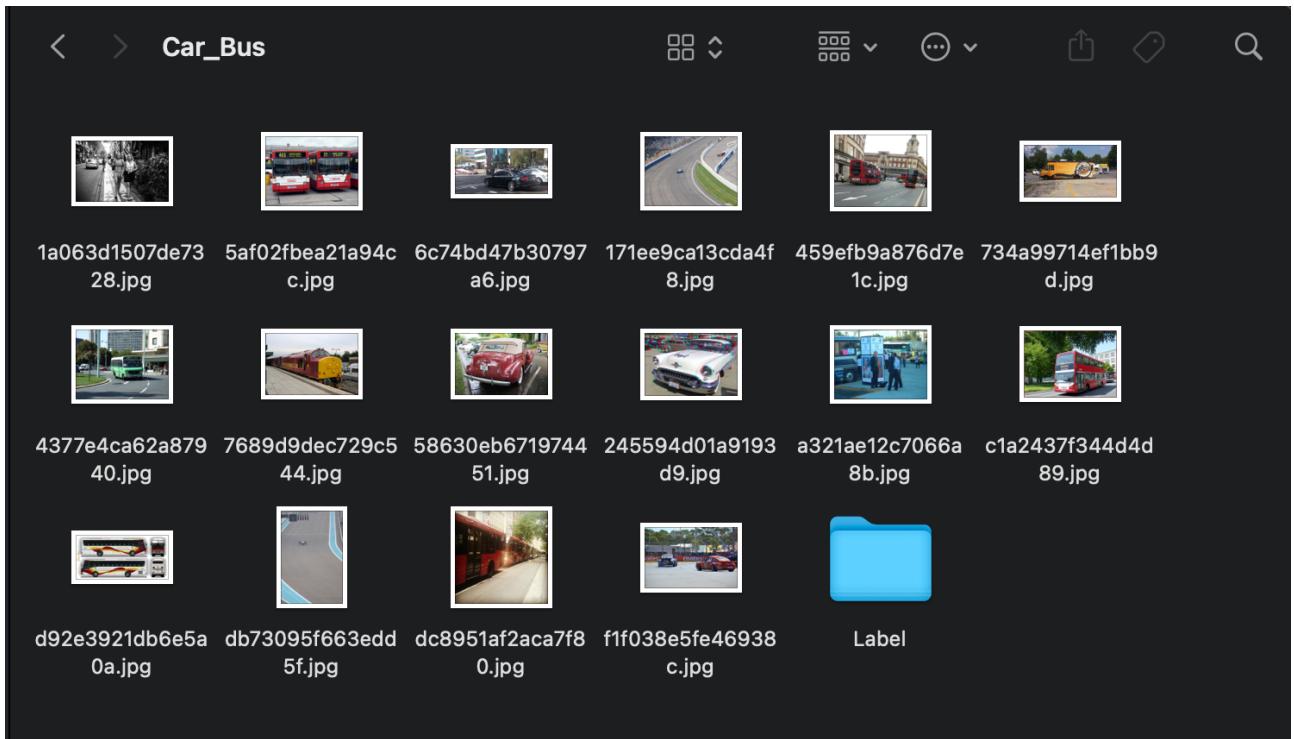


Figura 8.16: Dataset descargado con los ficheros TXT

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 9

Anexo II: Legislación

Capítulo 10

Anexo II: Planificación

Bibliografía

[ear,] 4G LTE EARFCN Calculator. <https://5g-tools.com/4g-lte-earfcn-calculator/>. Accedido el 24 de mayo de 2023.

[cm,] Generalized confusion matrix for multiple classes, garillos-manliguez, c.a. cgmanliguez@up.edu.ph november 25, 2016.

[yol, a] A review of yolo algorithm developments a review of yolo algorithm developments, peiyuan jiang, dajiergu*, fangyao liu, ying cai, bo ma, 2022.

[yol, b] Yolov3: An incremental improvement, joseph redmon, ali farhadi university of washington.

[Royuela et al., 2022] Royuela, I., Aguado, J. C., de Miguel, I., Merayo, N., Barroso, R. J. D., Hortelano, D., Ruiz, L., Fernández, P., Lorenzo, R. M., and Abril, E. J. (2022). A testbed for ccam services supported by edge computing, and use case of computation offloading. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6. IEEE.