UNIVERSIDAD EUROPEA MIGUEL DE CERVANTES

ESCUELA POLITÉCNICA SUPERIOR

TITULACIÓN:

MÁSTER EN GESTIÓN Y ANÁLISIS DE GRANDES VOLÚMENES DE DATOS: BIG DATA

TRABAJO FIN DE MÁSTER

Detección de objetos en imágenes mediante técnicas de Deep Learning: Señales de tráfico

Autor:

Antonio José Aroca Aguilar

Tutor:

Javier del Pozo Velázquez

Valladolid, noviembre de 2022

Contenido

1.- Introducción

La visión por computador, y particularmente la detección y reconocimiento de objetos, ha venido adquiriendo cada vez más importancia en los últimos años, con una gran cantidad de aplicaciones en distintas materias. En la ilustración 1 se pueden observar tres escenarios que ejemplifican distintos usos de la visión por computador hoy en día. Algunos de estos usos, como pudiera ser la conducción autónoma, suponen un cambio radical en la sociedad y esto hace que su implantación suponga un proceso largo y costoso. Sin embargo, existen otras muchas aplicaciones que, sin ser tan sonoras mediáticamente, suponen un gran avance y son una importante ayuda en la realización de distintos trabajos.

Ilustración 1: Ejemplos de uso de la visión por computador en la sociedad y la industria. Fuentes: [1], [2] y [3] respectivamente.

Precisamente nos encontramos en esta última situación. En este proyecto se persigue la implementación de un sistema soporte para el mantenimiento de señales de tráfico mediante técnicas de visión por computador. De esta forma, se pretende facilitar y agilizar el trabajo de aquellas personas, y organismos, dedicados a esta labor tan tediosa como importante. Una rápida identificación de aquellas señales en mal estado puede permitir que sean reparadas o sustituidas sin demora, lo que a su vez podría evitar accidentes y salvar vidas.

1.1.- Objetivos del trabajo

La finalidad principal de este trabajo de fin de máster es el desarrollo de un sistema para el reconocimiento de señales de tráfico a partir de imágenes. Al mismo tiempo, también se pretenden alcanzar otra serie de objetivos más concretos como los que se exponen a continuación:

- Estudiar el estado del arte de los distintos modelos empleados en la actualidad para la detección y el reconocimiento de imágenes y su funcionamiento.
- Estudiar el estado del arte de los distintos sistemas y técnicas realizadas para el mantenimiento de señales de tráfico en la actualidad.
- Buscar, tratar y procesar datasets de código abierto relacionados con señales de tráfico.
- Implementar y/o utilizar algoritmos de aprendizaje automático y visión por computador para la detección y el reconocimiento de objetos a partir de imágenes.
- Testear los algoritmos implementados ejecutándolos con distintos parámetros y distinta cantidad de etapas en el entrenamiento de la red.
- Estudiar la eficacia del sistema desarrollado mediante distintas métricas ampliamente utilizadas en problemas de visión por computador y detección de objetos.
- Liberar el código en GitHub para contribuir a la comunidad y que pueda ser estudiado y utilizado por cualquier usuario.

1.2.- Análisis de la situación

En España, y en cualquier otro lugar del mundo, las carreteras se encuentran llenas de todo tipo de señales. Estas nos indican a qué velocidad podemos ir, en qué dirección o direcciones podemos circular, qué precauciones tomar en todo momento, etc. *"Una señal de tráfico tiene una función fundamental y prioritaria para la seguridad vial. Los conductores recibimos en torno al noventa o noventa y cinco por ciento de la información por la vista." (Elena de la Peña, subdirectora general Asociación Española de la Carretera)*. Sin embargo, en algunas ocasiones estos elementos pueden deteriorarse o encontrarse ocultos por el paisaje, lo que supone un grave problema para la seguridad vial.

En nuestro país existen alrededor de 4.000.000 de señales sólo en carreteras. A esta cifra habría que sumarle aquellas ubicadas en pueblos y ciudades. El informe *Necesidades de Inversión en Conservación (Asociación Española de la Carretera, 2019)* asegura que más del 70% de las señales de tráfico están en mal estado o alguno de sus componentes ha superado la fecha de caducidad. Del mismo modo, también considera necesaria la sustitución o renovación de al menos 374.000 señales de código, obviando aquellas que son informativas. Contrario a este informe, Interempresas publica en [4] que, entre los años 2009 y 2019 se redujeron los recursos destinados a conservación y seguridad vial en un 76% por parte del Ministerio de Transportes, Movilidad y Agencia Urbana.

El mantenimiento de señales de tráfico se puede definir como *"el conjunto de actividades que se realiza para conservar de manera funcional y en buen estado todos los dispositivos utilizados para regular la circulación vehicular." (Manual para el mantenimiento de la red vial secundaria, Ministerio de Transportes de Colombia y Universidad Pontificia Javeriana)*. Este mantenimiento debe garantizar que la transmisión del mensaje que la señal representa se pueda realizar de manera clara, fácil y correcta. Según la legislación española, sólo la autoridad responsable de una vía puede modificar o sustituir las señales de dicha vía, y con datos del Ministerio de Fomento de 2014, tan sólo el 51,2% de los kilómetros totales de carretera están gestionados por la administración central. Sin embargo, en casos de emergencia, las autoridades están capacitadas para colocar señalización de carácter provisional. Finalmente, cabe mencionar que la iniciativa ciudadana tiene un papel muy importante en este mantenimiento mediante la comunicación y la denuncia de aquellas señalizaciones que no estén en un estado óptimo.

Aunque puede sufrir variaciones en función del país, mayoritariamente las actividades de mantenimiento de la señalización vial se clasifican en tres grupos distintos: periódicas, rutinarias y funcionales *(Manual centroamericano de mantenimiento de carreteras, Ciudad de Guatemala, 2004; Manual internacional de conservación de*

carreteras, Madrid, 1997; Mantenimiento rutinario de caminos con microempresas. Manual técnico, Lima, 2003). Estas se diferencian entre ellas principalmente en el alcance y la frecuencia con la que se realizan las distintas actividades pertenecientes a cada tipo. En la ilustración 2 se observan tres ejemplos de señales de tráfico que requieren mantenimiento, cada una de un tipo distinto.

Ilustración 2: Ejemplos de los distintos tipos de mantenimiento de señales de tráfico. Fuentes: [5], [6] y [7] respectivamente.

Las actividades funcionales deben realizarse mensualmente y su objetivo es detectar fallos ocurridos por eventos aleatorios. Estas actividades están pensadas para corregir aquellos deterioros provocados por accidentes de tráfico, vandalismo, o desastres naturales entre otros eventos.

Los trabajos de mantenimiento rutinarios deben hacerse al menos dos veces al año. Estos tienen como finalidad ofrecer al usuario una correcta visibilidad de la señalización. Es por ello por lo que las actividades más habituales dentro de este grupo son la limpieza de señales, ya sea por vandalismo, polvo o barro, carteles publicitarios, etc., pero también corregir aquellas cuya orientación o inclinación no permita una correcta visibilidad desde la calzada, y el retiro de vegetación y maleza que potencialmente oculten completa o parcialmente la señalización.

Finalmente, las actividades periódicas deben realizarse con frecuencia anual y engloban a todas aquellas labores cuyo propósito sea evitar el deterioro causado por el tiempo y mantener la integridad de las señales de tráfico. Este grupo también abarca aquellas tareas que garantizan que la señal se corresponda con las condiciones de tránsito de la vía. *"La señal cumple una función primordial para la seguridad de la carretera. Su duración por deterioro puede oscilar entre los diez y los veinte años." (Juan Cruz Martínez, director API Fabricación)*. Esta variación de tiempo está motivada por las condiciones medioambientales a las que es sometida cada señalización vial, así el salitre propio de las zonas costeras o la nieve en zonas montañosas, por ejemplo, tienden a reducir la vida útil de los materiales con los que estas son construidas. Por ello, para garantizar el estado idóneo de la señalización, estas actividades requieren unos estándares de calidad muy elevados.

La realización de todas estas tareas de forma regular implica la recopilación y análisis manual de datos. Esto unido a la gran complejidad de la infraestructura de transporte, su supervisión y gestión supone mucho personal y un alto coste tanto en infraestructura y materiales, como en mano de obra. Además, es necesario mantener información actualizada sobre el estado y la ubicación de cada señal de tráfico. Sin embargo, debido a la limitación de tiempo y presupuesto, junto a las consideraciones de seguridad a tener en cuenta en la recopilación manual de datos, los distintos organismos encargados del mantenimiento de las señales de tránsito necesitan una forma más eficiente de extraer información acerca del estado de estas.

Por ello, en este trabajo se propone una solución basada en la inteligencia artificial, *machine learning*, y visión por computador. Se pretende detectar e identificar las señales de tráfico captadas por una cámara ubicada en ciertos vehículos. De esta forma, y con ayuda de un sistema GPS que ubique la posición de estos automóviles en todo momento, se podrá registrar qué señales de tráfico se han detectado e identificado correctamente, y cuáles no.

Si el sistema presenta unos porcentajes elevados de eficacia, se podría suponer que en aquellas situaciones en las que se ha cometido un error de identificación, el fallo es motivado por una señalización que se encuentra en mal estado. Por otro lado, si el error cometido se corresponde con una no detección de la señal, el motivo podría ser que esta se encuentra derribada, tapada por cualquier motivo, o simplemente que no está orientada como debería. En cualquier caso, sería necesaria su revisión y mantenimiento correspondiente, siempre tras una pasada previa para poder identificar las señales. De esta manera, se tendría la referencia para saber si en ese punto geográfico en el que se debería haber detectado una señal, ha ocurrido algo.

Este sistema no pretende, en ningún caso, sustituir a todas las tareas de mantenimiento que se han comentado en párrafos anteriores, y otras muchas que son igualmente necesarias, ni siquiera a las de un grupo concreto. Simplemente se postula como un sistema soporte para detectar con mayor celeridad algunos casos, así como agilizar todo el proceso necesario para su tratamiento y rehabilitación. Dicho esto, en un principio, esta técnica permitiría localizar mayoritariamente casos pertenecientes al mantenimiento rutinario y funcional, los dos más frecuentes y costosos.

2.- Implementación

2.1.- Obtención, procesado y almacenamiento de datos

Uno de los aspectos más importantes a tener en cuenta, si no es el que más, cuando se realizan tareas de *machine learning* es seleccionar un *dataset* adecuado para el entrenamiento del modelo, o modelos a utilizar. Así como realizar un correcto preprocesado de estos datos de entrada. Referente a la selección, en todo problema existen dos opciones, crear un nuevo *dataset* desde cero, específico para el caso que pretendemos solucionar, o utilizar un conjunto de datos ya existente.

La primera de las dos alternativas puede resultar la más idónea puesto que se obtendrán los datos en función de las necesidades específicas requeridas. Sin embargo, la creación de un *dataset* es un proceso largo, tedioso y complicado si no se tienen los medios adecuados. En primer lugar, hay que tener en cuenta la cantidad de datos, fotografías en este caso, a recopilar. Esto no siempre es posible. Además, estos datos tienen que ser variados y no estar sesgados para evitar problemas de generalización. También, es necesario considerar las especificaciones del equipo utilizado para recopilar la información. No proporcionan imágenes de la misma calidad una cámara profesional que la de un teléfono móvil, por ejemplo.

Además, no sólo hay que preocuparse de obtener las imágenes, sino que también sería necesario realizar el etiquetado de las señales de tráfico que en ellas aparezcan. A pesar de que hoy en día existen numerosas herramientas que permiten establecer los límites de los *bounding boxes* de forma cómoda, repetir el proceso para cada señal de todas las imágenes puede convertirse en una tarea tediosa considerando el tamaño del *dataset*. También es imprescindible tener una gran precisión al marcar los límites, ya que este es un aspecto clave y, en caso contrario, se estaría perjudicando enormemente el proceso de aprendizaje.

En el caso de este proyecto, debido a las limitaciones de tiempo y recursos existentes, se ha considerado conveniente la utilización de un *dataset* ya existente. Sin embargo, este debía presentar una serie de características concretas, las cuales serán explicadas a continuación:

I. Al tratarse de un proyecto académico, el conjunto de datos debe ser gratuito y de uso libre.
II. Existen numerosos *datasets* de señales de tráfico que cumplen la primera condición, sin embargo, en su inmensa mayoría están pensados para

problemas de clasificación de imágenes. Es recomendable utilizar uno que esté diseñado para problemas de detección y reconocimiento de objetos.

III. El número de datos debe ser elevado. A mayor número de imágenes, mejor será el entrenamiento y el proceso de generalización.

IV. No pueden ser imágenes muy pesadas porque la herramienta a utilizar es Google Colaboratory, cuya versión gratuita ofrece recursos limitados. Aunque trabajar con fotografías de menor peso y calidad puede afectar negativamente a los resultados del experimento, se considera que, si la finalidad es que el modelo trabaje con imágenes captadas por una pequeña cámara ubicada en un vehículo en movimiento, estas tampoco van a ofrecer grandes parámetros de calidad.

V. Las imágenes deben cumplir ciertos estándares en cuanto a nitidez. El conjunto de datos a utilizar no debe presentar imágenes en las que no se aprecien con claridad todos los objetos de la escena. Se deben evitar aquellas fotografías distorsionadas por el movimiento de la cámara o aquellas en las que el sol esté de cara, entre otras.

VI. El etiquetado de las imágenes y la información de los *bounding boxes* debe estar dispuesto en formato YOLO.

Dicho esto, de entre todos los conjuntos de datos encontrados que cumplían con todas, o al menos la mayor parte de las condiciones descritas, y tras estudiar detalladamente cada uno de ellos, se ha optado por utilizar el *dataset* llamado *Road Sign Detection*. Este conjunto de imágenes creado por Larxel, un científico de datos senior de reputada experiencia que trabaja en el Hospital Israelita Albert Einstein en Sao Paulo, se trata de un conjunto de datos disponible en Kaggle ([14]), siendo de uso público bajo la licencia *CC0: Public Domain*. Esto quiere decir que se trata de un fichero sin derechos reservados, el creador renuncia a todos sus derechos de autor y estos pasan al dominio público.

Ilustración 3: Imagen de portada del dataset Road Sign Detection. Fuente: [14].

Estamos ante un *dataset* que presenta un total de 877 imágenes en las que podemos encontrar señales de tráfico pertenecientes a cuatro categorías distintas, *traffic light* (semáforo), *stop* (señal de *stop*), *speed limit* (señal de límite de velocidad), y *crosswalk*

(señal de paso de peatones). Sin embargo, estas no están representadas con la misma proporción. Mientras que tres de ellas sí aparecen en cantidades similares, en torno a 150 o 200 presencias en el conjunto de datos por cada clase, se encuentran límites de velocidad hasta en 788 ocasiones. Aquí aparece reflejado que en ciertas imágenes podemos encontrar representadas más de una única señal, lo cual es algo de gran importancia considerando el propósito final de este proyecto, al ser más representativo de la realidad. De hecho, las fotografías que constituyen este *dataset* han sido tomadas desde el salpicadero de distintos vehículos.

Tener una clase desbalanceada, con una representación hasta cuatro o cinco veces mayor que el resto, puede constituir un problema para el entrenamiento del modelo. Además, provocar una mayor predisposición a clasificar una señal como de esta clase. Sin embargo, este desbalance es motivado, en parte, porque no se hace distinción del límite establecido por la señal. Esto, unido al gran número de señales que hay de este tipo en todas las carreteras y ciudades españolas en contraste con el resto de marcas viales, ha hecho que no se considere conveniente ningún tratamiento en un principio. Será una vez empezada la experimentación, en caso de detectarse este problema, cuando se realicen las transformaciones pertinentes sobre los datos. Esta decisión se ha tomado así para mantener una representación más fidedigna de la realidad y porque, además, en la mayoría de las ocasiones, al eliminar una imagen con uno o varios límites de velocidad también se elimina alguna de las otras señales que se detectan.

Por otro lado, esto no quiere decir que no haya sido necesario ningún preprocesado ni tratamiento de estos datos. Todo lo contrario, esta es una parte muy importante de cualquier proyecto de *machine learning*, y como tal se le ha dedicado especial atención. Para dicho procedimiento se ha utilizado Roboflow, una plataforma para la implementación y el uso de técnicas de visión artificial de forma rápida y sencilla. Según datos de su propia web, ofrecen más de 66 millones de imágenes ya anotadas, 90.000 *datasets*, muchos de ellos públicos, o 7.000 modelos preentrenados.

Desarrollado por la empresa estadounidense con la que comparte nombre, Roboflow ofrece un marco de trabajo de visión por ordenador, proporcionando una gran variedad de técnicas que cubren todo el proceso, desde la recopilación de datos hasta el despliegue del modelo. También incluye la posibilidad de que los usuarios carguen sus propios conjuntos de datos, e incluso incorpora una herramienta gráfica para el etiquetado de estas imágenes y los objetos que contengan. Además, aunque ofrece una gran variedad de modelos preentrenados como se ha comentado anteriormente, incluye opciones de implementación, permitiendo trabajar eficientemente con bibliotecas como EfficientNet, Tensorflow o

Pytorch. Igualmente, presenta opciones de visualización y despliegue, y facilita la coordinación y comunicación de un equipo de desarrollo, organizando el trabajo en proyectos e incluyendo incluso un control de versiones.

En lo referente al preprocesado de datos, la función para la cual se ha utilizado esta herramienta, Roboflow permite la realización de una gran cantidad de tareas, entre las que se encuentran la reorientación de imágenes u objetos contenidos en ellas, cambiar el tamaño de estas, variar el contraste y el brillo, distintas opciones de *data augmentation*, etc. Entre estas tareas destaca que ha sido necesario equiparar el tamaño de las imágenes ya que en un principio no había homogeneidad en todo el *dataset*. Por ello, se han establecido unas dimensiones de 416x416x3 para todas las ilustraciones, siendo esta última magnitud el número de canales, es decir, en este proyecto se va a trabajar con fotografías a color. Este tamaño de imagen ha sido determinado en base a la observación de las distintas fotografías que componen el *dataset* y a nuestra experiencia. Se ha creído que este era un tamaño adecuado, que no iba a distorsionar las escenas representadas y, que no alteraría en exceso las características y proporciones de las imágenes.

Otro aspecto clave llevado a cabo con esta herramienta es la creación de los subconjuntos de entrenamiento, validación y testeo. Seleccionar correctamente cada agrupación es de vital importancia pues esto será determinante para el entrenamiento del modelo, y consecuentemente, para sus prestaciones. En este caso, gracias a Roboflow, tan solo es necesario especificar qué porcentaje del *dataset* incluir en cada grupo. Será la propia herramienta la encargada de aleatorizar los datos para evitar sesgos y problemas de generalización. Finalmente, se trabajará con un conjunto de entrenamiento que contendrá el 70% de las imágenes, unas 613 ilustraciones; mientras tanto, el de validación estará formado por 175 fotografías, un 20%; y el de testeo por el 10% restante, 87 imágenes.

Para la realización de este trabajo se ha utilizado la versión gratuita de esta herramienta desde la web, la cual permite el uso de la mayoría de sus funciones básicas. Sin embargo, sus desarrolladores ofrecen distintos planes de pago personalizados en función de las necesidades y los requerimientos de cada usuario. Estos incluyen servicios avanzados como modo *offline*, la elaboración de *reviews*, personal de mantenimiento exclusivo, distintas integraciones de TI, o acceso a la VPC entre otras funciones. Adicionalmente a estos servicios, Roboflow ofrece de manera gratuita acceso a una gran cantidad de foros y blogs oficiales en los que se publican las últimas novedades. Del mismo modo, su documentación oficial también está disponible.

Sin embargo, no solo es necesario realizar las transformaciones que se consideren pertinentes sobre las imágenes, sino que también hay que modificar el etiquetado de los

datos. En esta ocasión se va a utilizar la red neuronal YOLO como modelo para reconocer las distintas señales de tráfico. Lamentablemente, este requiere que las etiquetas y la información de los *bounding boxes* de cada objeto a identificar estén dispuestos en un formato específico, el cual no coincide con la estructura en la que vienen dispuestos en el *dataset* seleccionado. Esto quiere decir que *Road Sign Detection* no cumple con la sexta condición que se indicó anteriormente en esta misma sección. Aun así, se seleccionó este *dataset* por delante de otros que sí cumplieran dicha restricción porque este es un problema que se puede solventar, y el resto de los conjuntos de datos tampoco cumplían con los seis requisitos en su totalidad. Este era el mejor que se ha podido encontrar.

Una solución a este problema sería etiquetar las imágenes nuevamente con ayuda de la herramienta Roboflow. Sin embargo, se ha pensado que una mejor alternativa consistiría en diseñar un *script* que modificara las anotaciones originales. Para ello, previamente se debe conocer con detalle cómo son los dos formatos con los que se va a operar, YOLO y Pascal VOC.

Pascal VOC es el formato en el que se encuentran las anotaciones del *dataset* seleccionado. Este está basado en una serie de etiquetas xml que proporcionan tanto información acerca de la imagen como acerca de los *bounding boxes* contenidos en ella. En la ilustración 4 se puede observar un ejemplo de este tipo de anotaciones.

```
<annotation>
    <folder>images</folder>
    <filename>road754.png</filename>
    <size>
        <width>300</width>
        <height>400</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>speedlimit</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>85</xmin>
            <ymin>100</ymin>
            <xmax>163</xmax>
            <ymax>178</ymax>
        </bndbox>
    </object>
    <object>
        <name>speedlimit</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>180</xmin>
            <ymin>99</ymin>
            <xmax>254</xmax>
            <ymax>176</ymax>
        </bndbox>
    </object>
</annotation>
```

Ilustración 4: Ejemplo de anotaciones en formato Pascal VOC de una de las imágenes del dataset Road Sing Detection.

Las dimensiones del *bounding box* se indican de manera absoluta en este tipo de anotaciones mediante los tags “xmin”, “ymin”, “xmax” e “ymax”. Por ello es por lo que se especifica también el tamaño de la imagen. Esto quiere decir que los recuadros que contienen los objetos se localizan indicando la posición de los dos puntos que constituyen la diagonal principal de dicho rectángulo. También es identificada la clase a la que pertenece mediante el campo “name”.

Por otro lado, el formato YOLO no presenta información alguna de la imagen en particular, solamente muestra los datos referentes a los *bounding boxes*, utilizando una línea por cada objeto que se encuentre en la imagen, y separando la información de un mismo elemento por un único espacio en blanco. Además, hay que tener en cuenta que, al no presentar las dimensiones de la imagen original, las coordenadas son tratadas de manera relativa, siendo la esquina superior izquierda la (0,0) y la esquina inferior derecha la (1,1).

Ilustración 5: Formato en que YOLO considera las imágenes y sus anotaciones. Fuente: [24].

En las ilustraciones 5 y 6 se puede observar justo lo que se había mencionado anteriormente. También queda reflejado en dichas imágenes cómo deben especificarse el tamaño y la posición de los *bounding boxes*. Por lo tanto, cada línea referente a un objeto debe contener la siguiente información: en primer lugar, un número entero comprendido entre 0 y n-1, siendo n el número de clases con las que se trabajan; a continuación, aparecerá la coordenada “X” del centro del *bounding box*, seguido de su coordenada “Y”; en cuarta posición aparecerá al ancho del recuadro; y finalmente su altura.

```
2 0.413333 0.347500 0.260000 0.195000
2 0.723333 0.343750 0.246667 0.192500
```

Ilustración 6: Ejemplo de anotaciones en formato YOLO de la misma imagen representada en la ilustración número cuatro.

Conocidas las particularidades que presenta cada formato, ya se podía crear un *script* de Python para leer el archivo xml con las anotaciones en Pascal VOC y escribir los archivos en formato ".txt" con sus correspondientes anotaciones en formato YOLO. El código de este *script*, llamado *parser.py*, se puede encontrar en el anexo 1 de este mismo documento.

Adicionalmente a este fichero, se ha implementado otro ejecutable de Python para mostrar por pantalla una imagen con sus respectivas anotaciones, *bounding boxes*, comprobando así que la conversión se ha llevado a cabo correctamente, y que no existen errores por trabajar con números flotantes. Hay que recordar que YOLO considera tamaños relativos entre el (0,0) y el (1,1), lo cual puede ser una fuente de error. Afortunadamente esto no fue así, todas las anotaciones fueron correctamente convertidas de un formato a otro. Este *script*, llamado *mostrar_anotaciones.py*, también se halla en el primer anexo de esta memoria.

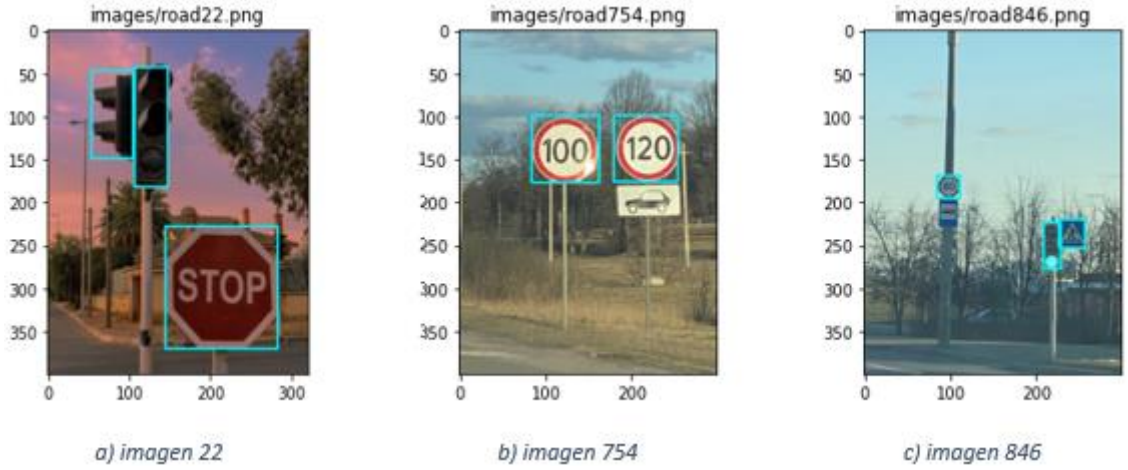

Ilustración 7: Ejemplos de imágenes del dataset con sus respectivos bounding boxes tras realizar la conversión de estos a formato YOLO.

En cuanto al almacenamiento del conjunto de datos, se va a utilizar Google Colaboratory como IDE de desarrollo, como ya se había comentado anteriormente en alguna ocasión. Por ello, las imágenes van a ser guardadas en un directorio de Google Drive. Además, el formato YOLO exige una disposición muy concreta tanto de las ilustraciones como de sus respectivas anotaciones. En primer lugar, los elementos pertenecientes a cada uno de los subconjuntos de entrenamiento, validación y testeo deben ubicarse en tres carpetas diferentes. Y dentro de cada una de estas, se dispondrán en dos directorios distintos las imágenes y las anotaciones.

Finalmente, es preciso crear un archivo de configuración con extensión ".yaml", en el cual hay que disponer cierta información utilizando unas etiquetas concretas. En las tres primeras líneas, se indicarán las rutas a los directorios que contienen las imágenes de cada uno de los tres subgrupos, utilizando para ello las palabras clave "train", "val" y "test"

seguidas de dos puntos y de los distintos *paths*. En el hipotético caso en que alguno de estos subconjuntos no se fuera a utilizar o no hubiera sido creado, podría omitirse dicha línea con total libertad. A continuación, en una nueva línea debe especificarse con la etiqueta "nc" el número total de clases que hay, siendo cuatro las estudiadas en este trabajo. Para terminar, en "names" se puede dar una lista indicando el nombre de cada una de las distintas clases. En realidad, este archivo de configuración puede ser mucho más complejo y aportar una gran cantidad de información. Sin embargo, estas secciones comentadas son imprescindibles, sin las cuales no se podría comenzar el proceso de entrenamiento de la red, mientras que el resto de los campos serían opcionales.

En resumen, el preprocesado de datos es una de las tareas más importantes y a las que más tiempo debe dedicarse en cualquier problema de *machine learning*. Aunque en visión por computador, por el hecho de trabajar con imágenes, no existe tanto juego a la hora de hacer ingeniería de características con respecto a otro tipo de tareas con distintas variables y columnas numéricas, por ejemplo, eso no quiere decir que no sea necesario analizar adecuadamente los datos de los que disponemos. Por ello, se ha hecho uso de la herramienta Roboflow para realizar distintas labores de preprocesado de imágenes. Además, también se ha utilizado esta herramienta para la generación de subconjuntos, y ha sido necesario realizar una conversión de formato en las anotaciones.

En un principio, no se creyó necesario realizar ninguna modificación extra sobre las imágenes o el conjunto de datos, además de las ya realizadas. No obstante, siempre queda a expensas de los resultados mostrados en el entrenamiento de la red, el que haya que hacer un balance de clases o aumentar la cantidad de datos mediante técnicas de *data augmentation*, entre otras posibles transformaciones.

2.2.- Análisis exploratorio

En esta sección, análisis exploratorio, se debería realizar un estudio y analizar cómo se distribuyen los datos. Esta exploración sería imprescindible y tendría una gran repercusión en el proyecto si se estuviera trabajando en problemas de clasificación, regresión, series temporales, procesamiento del lenguaje natural, *clustering*, etc. No obstante, no es así en visión por computador, donde los datos que se manejan son imágenes principalmente. Por ello, este apartado se ha dedicado a la realización de un estudio del estado del arte de la inteligencia artificial y la visión por computador, así como a investigar los principales algoritmos y modelos existentes que son susceptibles de ser utilizados en este trabajo.

El aprendizaje automático, también llamado *machine learning*, es un término que comprende todo el proceso mediante el cual un ordenador, o una máquina en general, es capaz de mejorar con la experiencia, aprender a partir de datos. Esto quiere decir que, cuanto mayor sea la cantidad de datos de la que dispone una computadora, mejor será su proceso de aprendizaje y, consecuentemente, sus prestaciones.

De forma más precisa, se puede definir el *machine learning* como aquella *"rama de la inteligencia artificial que crea modelos analíticos automáticamente, de manera autónoma y sin la intervención de un programador humano, mediante la identificación de patrones en un conjunto de datos" ([26])*. El objetivo principal de este campo es realizar predicciones sobre un conjunto de datos completamente nuevo, distinto al utilizado para obtener los patrones de los que se ha hablado anteriormente.

Dentro de esta disciplina se han desarrollado multitud de algoritmos y técnicas en los últimos años. Por su interés e importancia en este trabajo cabe destacar las conocidas como redes neuronales convolucionales, las famosas CNNs (*Convolutional Neural Networks*). Sin embargo, no se puede hablar de ellas sin conocer previamente qué es una red neuronal artificial.

Nombradas en la literatura como *Neural Network* o NN, una red neuronal no es más que un conjunto de neuronas artificiales conectadas entre sí, y que comparten información entre ellas. Siendo a su vez una neurona artificial, una única unidad de cómputo que intenta imitar el proceso de razonamiento humano. En función de los datos de entrada que reciba, las distintas capas de la red extraerán los patrones usados para dar una información de salida cuando se le presenten nuevos datos no conocidos.

Tradicionalmente, las redes neuronales estaban constituidas por tres capas únicamente: la de entrada, una intermedia, y la de salida. Sin embargo, su capacidad de aprendizaje era limitada. Por ello, años más tardes empezaron a surgir redes con un número

mayor de capas intermedias, lo que suponía unas prestaciones superiores. La utilización de estas redes con más de tres capas es lo que se conoce como *deep learning*, o aprendizaje profundo, y es precisamente lo que se ha utilizado en este proyecto.

Dicho esto, este trabajo podría ubicarse en el campo de la visión por computador, CV. Esta rama de la inteligencia artificial tiene como objetivo dotar a las computadoras de la capacidad de "ver", "observar" y "comprender" las imágenes. En realidad, lo que se hace es dar a los ordenadores la facultad de obtener información significativa e interpretarla para realizar predicciones sobre imágenes. Aquí se puede apreciar una relación muy estrecha entre la visión por computador y el *machine learning*, particularmente con las redes neuronales que se han visto anteriormente, ya que estas son las herramientas que permiten extraer e interpretar información de las imágenes. Este vínculo entre ambos es tal, que en muchos casos las distintas disciplinas llegan a entrelazarse y es difícil encontrar la frontera que separa una de la otra.

Esta no es la única rama de la que se nutre la visión por computador, como se aprecia en la ilustración 8, pero sí que es la de más interés por la naturaleza de este trabajo. Al ser un proyecto puramente académico y teórico no se va a montar y probar un sistema robótico funcional incorporado a un vehículo. Simplemente se va a utilizar un *dataset* ya existente, y se recopilarán imágenes de distintas fuentes para testear el modelo implementado utilizando un ordenador.

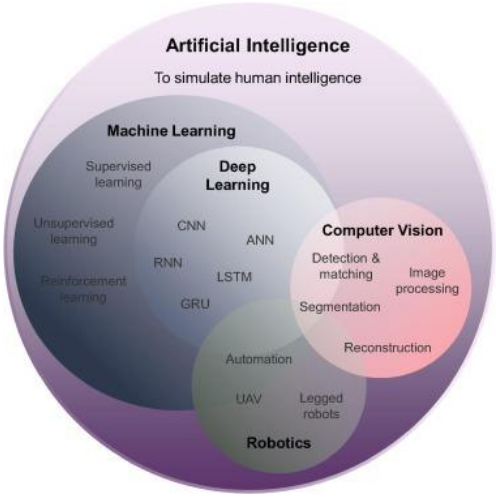

Ilustración 8: Diagrama de Venn que permite observar la relación existente entre la visión por computador y otros campos de la inteligencia artificial. Fuente: [27].

Hasta ahora, se había comentado que una NN permite obtener patrones a partir de unos datos de entrada. No obstante, las imágenes que se pretende tratar tienen una mayor dimensionalidad y complejidad que un simple vector de números. Para solucionar este problema surgen precisamente las redes neuronales convolucionales. Una CNN es una red

neuronal cuya estructura se ha modificado con el fin de hacerla más eficaz y eficiente ante este tipo de datos de entrada. A lo largo de los años estas estructuras han evolucionado rápidamente creando una gran variedad de modelos distintos. Será responsabilidad de cada trabajador, estudiante o investigador el elegir un prototipo u otro en función de las necesidades de cada problema.

Llegados a este punto se ha comentado qué existen herramientas que permiten recopilar información a partir de imágenes, pero ¿con qué propósito? Aunque, en un principio, el objetivo de aplicar técnicas de CV puede ser obtener cualquier tipo de información a partir de una o varias ilustraciones de entrada para cualquier fin, generalmente son cuatro las aplicaciones básicas de la visión por computador.

- **Clasificación de imágenes.** El objetivo es etiquetar una imagen, o lo que es lo mismo, clasificarla en un grupo determinado. Básicamente se pretende describir en una palabra lo que representa una ilustración.
- **Detección de objetos.** El objetivo es localizar en una imagen los objetos que haya representados en ella y que sean de interés. Para ello se utiliza una marca especial llamado *bounding box*, consistente en un recuadro que rodea a cada elemento destacándolo sobre el fondo de la imagen.
- **Reconocimiento de objetos.** El objetivo ahora no es únicamente detectar los distintos objetos de la imagen, sino identificar qué es cada uno de estos. En la literatura es bastante común considerar la detección y el reconocimiento de objetos como una única tarea, sin diferenciarlas. En la ilustración 9 se puede apreciar gráficamente la diferencia entre ambas.
- **Segmentación de imágenes.** El objetivo no es sólo detectar y reconocer cada objeto dentro de una ilustración, sino que se persigue indicar a qué clase pertenece cada píxel.

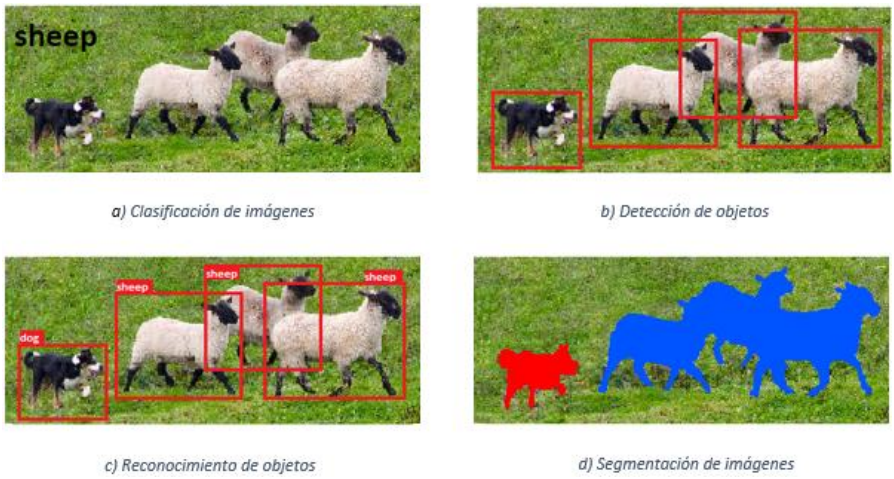

Ilustración 9: Principales aplicaciones de la visión por computador. Fuente: [38].

Aunque el nombre de este trabajo de fin de máster indica que la tarea a realizar consiste en una detección de objetos, se considera que para el caso de uso en el que está enfocando, es decir, un sistema de ayuda para el mantenimiento de señales de tráfico, tiene un mayor interés no sólo detectar las señales, sino también ser capaces de reconocerlas. De esta forma se podrá identificar, presuntamente, más casos de deterioro en la señalización vial. Además, tal y como se ha mencionado anteriormente, el reconocimiento y la detección de objetos tienden a llamarse de la misma forma en la literatura. Por ello, tal y como se explicará en secciones posteriores, se ha decidido diferenciar dos etapas distintas dentro de este proyecto. Una primera en la que simplemente se realiza la detección de las señales que se identifican en el *dataset*, sin diferenciar qué tipo de señales son; y una segunda etapa en la que se pone el foco en el reconocimiento de estas señales, no sólo su detección.

Las técnicas existentes para reconocimiento de objetos suelen dividirse en dos grupos en función del número de pasadas que estas realicen sobre los datos ([34]). En primer lugar, existen los modelos y arquitecturas cuyo funcionamiento requiere de dos etapas o barridos de datos. Los modelos y arquitecturas pertenecientes a este grupo descubren cuáles son las regiones en las que existe una posibilidad elevada de encontrarse un objeto en el ciclo inicial. Finalmente, en la segunda fase, con la información obtenida en la etapa anterior, deciden si realmente se halla un objeto en las zonas potenciales mencionadas, y en caso positivo, de qué objeto se trata.

En [34] se destaca que entre los algoritmos más conocidos pertenecientes a este conjunto se encuentra R-CNN, un modelo consistente en aplicar una red neuronal convolucional para clasificación, VGG-16 por ejemplo, a cada una de las regiones potenciales con el fin de extraer las características de estas. En cuanto al método utilizado para identificar regiones de interés, existen distintas aproximaciones como puede ser Edge Boxes, que aplica filtros a las imágenes para encontrar cambios de textura. Posteriormente surgieron otras versiones de esta estructura, Fast R-CNN y Faster R-CNN, que no suponen un incremento sustancial de prestaciones, pero sí conllevan una mejora importante en tiempo.

Por otro lado, la alternativa se encuentra en las técnicas que únicamente requieren de una etapa para realizar todo el proceso. En sus inicios, estas técnicas eran más simples por norma general y permitían el reconocimiento de objetos en tiempo real. Sin embargo, su inconveniente principal es que presentaban un número mayor de falsos positivos. Dentro de este grupo destacan YOLO y SSD por encima del resto.

La idea básica de YOLO es dividir la imagen en un *grid* o cuadrícula de tamaño fijo. Una vez hecho esto, y por cada región, una CNN obtendrá la probabilidad de que esta celda sea el centro de un objeto, y también indicaría de qué objeto se trata. Además, se añadiría

el *bounding box* de la región respecto a la imagen global. Es decir, por cada celda del *grid* se obtiene la probabilidad de ser parte de un objeto o no, cuatro reales que permiten calcular el *bounding box* respecto a dicha casilla, y una probabilidad sobre cada una de las clases disponibles para identificar de qué objeto se trata. Esto quiere decir que la salida se corresponde con un tensor.

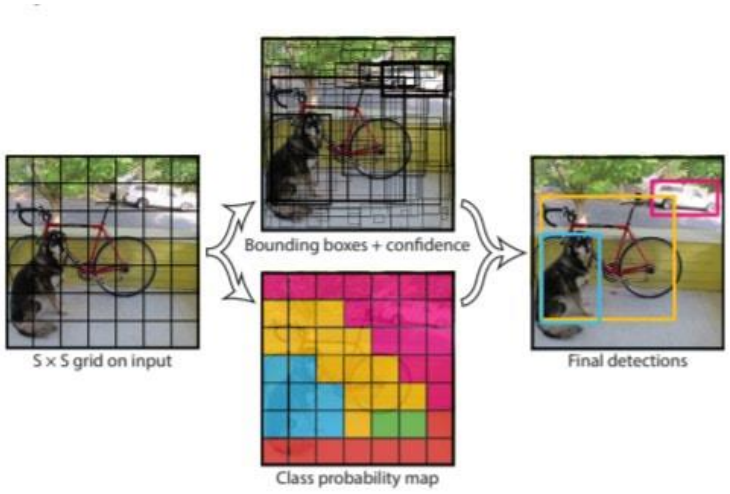

Ilustración 10: Esquema de funcionamiento de YOLO. Fuente: [30].

Algunas de las implementaciones más usadas de YOLO eran las que utilizaban las redes neuronales LeNet y DarkNet ([34]). Era un algoritmo más rápido que todos los de la fecha y podía funcionar en tiempo real. Aunque en sus inicios presentaba mayor cantidad de falsos positivos que otros algoritmos, seguía teniendo una precisión elevada en muchos casos, y aceptable en la mayoría. Además, al tratarse de una CNN exhibía una alta capacidad de aprendizaje. Sin embargo, disponía de un gran contratiempo. Al detectar únicamente un objeto por celda de la cuadrícula, le costaba identificar objetos pequeños en grupos, como puede ser una hilera de hormigas, por ejemplo. Surgido en 2015, YOLO revolucionó el mundo de la detección y reconocimiento de objetos. Fue el primero en hacerlo en una única etapa y, aunque fue superado por otros que surgieron más tarde, a partir de la tercera versión ha destacado por encima del resto.

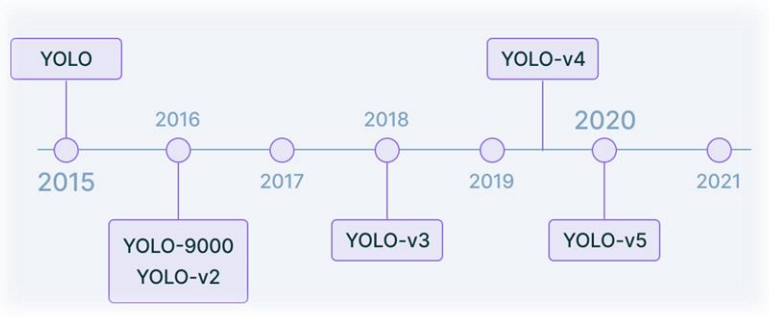

Ilustración 11: Timeline de YOLO. Fuente:[29].

Como se explica en [29], un año posterior al nacimiento de YOLO, surgió YOLOv2 con la finalidad de corregir el problema comentado anteriormente mediante el cual la red no era capaz de identificar pequeños objetos en grupo. Esto lo logra permitiendo la predicción de hasta cinco *bounding boxes* distintos por cada celda del *grid*. Este número cinco se estableció de forma arbitraria por ofrecer un buen compromiso entre complejidad y rendimiento. Además, YOLOv2 mejoró la precisión de sus predicciones ligeramente al utilizar una CNN que incluía capas de normalización por lotes.

En paralelo a YOLOv2, hizo su aparición YOLO-9000 con el objetivo de detectar todavía más clases de las 80 permitidas por la segunda versión de esta red. Aunque con una precisión inferior de media, YOLO-9000 se convirtió en un algoritmo muy interesante y poderoso por ser capaz de detectar objetos pertenecientes a más de nueve mil categorías distintas.

Como ya se ha mencionado, surgieron otros algoritmos que, si bien no superaban a las versiones comentadas en los párrafos anteriores en términos de velocidad, sí que lo hacían en precisión. Para mejorar a estos modelos, en [29] se sigue explicando que nació YOLOv3 en el año 2018 con la red neuronal DarkNet-53 como estructura principal. Esta red es mucho más profunda que las utilizadas hasta ahora en versiones anteriores y, aunque se vio reducido el número de *bounding boxes* permitido por cada celda a tres, YOLOv3 incorporaba la detección a multiescala como una de sus principales novedades. Esta posibilidad de hacer predicciones a tres escalas distintas mejoraba significativamente la detección de objetos de pequeño tamaño. De esta forma consiguieron superar a sus competidores y ser prácticamente el único algoritmo utilizado desde la fecha para detección de objetos, ya sea en tiempo real o no.

El autor de YOLO dejó su trabajo hace algunos años por lo que, a partir de esta tercera versión los siguientes algoritmos no son considerados por muchos investigadores como productos YOLO oficiales. Sin embargo, esto no significa que no se haya seguido trabajando y mejorando los modelos anteriores. YOLOv4 surge precisamente como una actualización y mejora de su predecesora.

Posteriormente, en el año 2020 se dio a conocer YOLOv5, un proyecto de código abierto mantenido por la empresa norteamericana Ultralytics. En [30] y [31] se aprecia que se trata de una familia de arquitecturas y modelos basadas en las versiones anteriores de este algoritmo y preentrenados con la base de datos COCO. Además, YOLOv5 tiene integrado *Weight and Biases*, un sistema que permite la visualización en tiempo real del proceso de entrenamiento y su registro en la nube, a parte de un registro local mediante *Tensorboard*. También existe una gran cantidad de documentación y tutoriales oficiales de libre acceso,

así como foros en los que una amplia y activa comunidad de investigadores y desarrolladores discuten sobre los últimos avances y se ayudan entre sí.

Respecto a las distintas arquitecturas que constituyen esta gran familia de modelos que supone YOLOv5, las principales son las siguientes ([31]):

- **YOLOv5n** (nano) **->** arquitectura más pequeña. Requiere un espacio de almacenamiento de 4 MB utilizando el sistema FP16, ofrece una precisión de 28,4 mAP sobre el dataset COCO, y una velocidad de 6,3 ms sobre una NVIDIA V100 b1.
- **YOLOv5s** (small) **->** arquitectura algo superior que requiere 14 MB de almacenamiento, ofrece una precisión de 37,2 mAP, y una velocidad de 6,4 ms bajo las mismas condiciones que la anterior.
- **YOLOv5m** (medium) **->** con un requerimiento de memoria de 41 MB, presenta una precisión de 45,2 mAP y una velocidad de 8,2 ms bajo las mismas condiciones.
- **YOLOv5l** (large) **->** arquitectura que requiere de un espacio de 89 MB, tiene una precisión de 48,8 mAP y una velocidad de 10,1 ms bajo las mismas condiciones.
- **YOLOv5x** (xlarge) **->** la arquitectura más profunda de todas estas. Requiere 166 MB de espacio, tiene una precisión de 50,7 mAP y ofrece una velocidad inferior a los 12,1 ms bajo las mismas condiciones.

Finalmente, en base a toda la información aquí presentada y mucha más que se ha recabado, y en base a la experiencia, se va a utilizar la familia YOLOv5. Se piensa que es la opción más interesante para este trabajo porque, aunque han surgido otras versiones más avanzadas de YOLO en los últimos años, estas no están tan optimizadas y aun hoy en día es más corriente utilizar YOLOv5, e incluso YOLOv3, que la sexta y la séptima versión de estos modelos. Además, aunque en un principio no se vaya a requerir el procesamiento en tiempo real, sí que resulta una opción muy interesante a tener en cuenta de cara a futuras actualizaciones del sistema.

2.3.- Diseño e implementación de los modelos o técnicas necesarias

Llegados a este punto, ya se dispone de un *dataset* que se ha trabajado y preprocesado para adecuarlo y optimizarlo según las necesidades requeridas por este trabajo. También se ha seleccionado ya el algoritmo, o familia de arquitecturas, a utilizar para llevar a cabo la detección y reconocimiento de señales de tráfico. Ahora bien, tal y como se había comentado anteriormente, se van a diferenciar dos etapas distintas en este proyecto en función de si el foco está puesto en la detección, o si lo está en el reconocimiento de las señales de tráfico.

En primer lugar, se hará hincapié únicamente en la detección de la señalización vial, sin entrar a ver qué representan estos objetos. Para ello, y con el fin de aprovechar el modelo YOLOv5 del que ya se ha hablado previamente, se realizó una pequeña transformación en las anotaciones de la base de datos. Esta consistía en modificar e igualar la clase a la que pertenecían todos los objetos, provocando así la existencia de una única categoría y que el modelo no tenga que aprender a diferenciar los distintos grupos. Esto se hizo mediante la implementación de un nuevo *script* de Python llamado *unificar_clases.py*. Este archivo, cuyo código se encuentra nuevamente en el primer anexo de este documento, simplemente iba leyendo y modificando todas las anotaciones encontradas en un directorio dado.

También se ha elaborado un cuaderno de Google Colaboratory llamado *YOLOv5.ipynb*, el cual va a permitir realizar el entrenamiento de una arquitectura determinada con la base de datos especificada y los parámetros que se indiquen. Este *notebook* se ha creado con el objetivo de disponer de un código limpio y reutilizable que permita realizar todas las pruebas necesarias haciendo los mínimos cambios posibles. El uso de Colab como entorno de desarrollado de este cuaderno, por delante de otros más populares como Jupyter, viene motivado porque las ejecuciones de este último se realizan en local. Por otro lado, al ser una herramienta de Google, Colab permite la ejecución en la nube utilizando los servidores de la gran compañía estadounidense. De esta forma se aprovecha la capacidad de YOLO para reducir drásticamente el tiempo de ejecución mediante el uso de la tarjeta gráfica, sin tener que dedicar tiempo y esfuerzo en la configuración y gestión de esta.

En dicho cuaderno, cuyo código se puede encontrar en el primer anexo de esta memoria, en primer lugar, se clona el repositorio de YOLOv5 de Ultralytics y se instalan las dependencias. A continuación, se inicia sesión en Tensorboard para el registro local del proceso de entrenamiento. También, se inicia sesión en weight and biases para su visualización y registro en la nube. Tras esto, se realiza el entrenamiento propiamente dicho,

y se comprueban las prestaciones del modelo resultante utilizando para ello el subconjunto de testeo. Finalmente, se dedican algunas celdas para descargar los pesos del modelo y ciertas gráficas con un resumen de sus prestaciones.

Para el entrenamiento es necesario especificar una serie de parámetros. Concretamente se está hablando del tamaño de las imágenes, 416 como ya se dijo en secciones anteriores; el tamaño de *batch* a utilizar, 32 en este caso debido a las dimensiones del *dataset* y a que un tamaño de lote demasiado grande puede hacer que el algoritmo converja hacía óptimos locales; la tasa de aprendizaje o *learning rate*, 0.01 de forma estática durante todo el experimento; y el optimizador, que en este caso se trata del SGD. Se ha seleccionado el descenso del gradiente estocástico (SGD) por ser un optimizador que suele presentar buenos resultados en general, con independencia del problema a resolver, y porque solo calcula las derivadas en una única muestra escogida aleatoriamente por cada *batch*. De esta forma se consigue reducir el tiempo y el coste computacional en la optimización de los pesos del modelo.

Otro parámetro para el cual se debe especificar un valor es el número de épocas. Se comenzó el proyecto realizando un entrenamiento que completara 100 de estas iteraciones. Sin embargo, pronto se vio que este número se quedaba corto, no daba tiempo al algoritmo a converger, provocando así un problema de *underfitting*. Por ello se decidió aumentar el número de épocas a 1000. Además, para evitar futuros problemas de sobreentrenamiento se ha utilizado una técnica conocida como *early stopping*. Esta detiene el entrenamiento llegado el caso en el que, tras un número concreto de épocas, un 10% de las totales, no se consiga mejorar los resultados de las etapas anteriores.

Existen más parámetros que también influyen en mayor o menor medida en el proceso de entrenamiento del modelo, sin embargo, en este caso se ha decidido dejar sus valores por defecto.

Hecho este cuaderno que se va a reutilizar durante todo el proyecto y modificadas las anotaciones del *dataset* para eliminar las clases, se probó las distintas arquitecturas que nos ofrece YOLOv5. Obviando la más pequeña, se optó en primer lugar por la versión *small* de este algoritmo. Después, ante los resultados presentados, se fue aumentando progresivamente la profundidad y la complejidad de la arquitectura con sus versiones *medium* y *large*. No fue necesario probar la versión *xlarge* porque con YOLOv5l ya estaban surgiendo problemas de *overfitting*, como se verá con más detenimiento en el apartado *3.- Análisis de los resultados obtenidos*.

A continuación, utilizando el mismo *notebook*, se repitió el proceso seguido anteriormente, con el objetivo en este caso de solucionar un problema de reconocimiento

de objetos, es decir, trabajando ahora sí con las cuatro clases que distingue el *dataset* originalmente.

Después, tras estudiar qué arquitectura es la que ofrece mejores prestaciones para la localización e identificación de las señales de tráfico, se intentó mejorar aún más estos resultados aplicando distintas técnicas de visión por computador y *machine learning*.

La primera de las técnicas aplicadas tiene como objetivo principal corregir los problemas de sobreentrenamiento surgidos con las arquitecturas de mayor profundidad, así como mejorar la generalización del modelo. Esta técnica, conocida como *data augmentation*, permite aumentar la cantidad de datos disponibles mediante la adición de datos sintéticos, que consisten en copias ligeramente modificadas de los originales. Estas variaciones son aleatorias para aumentar la diversidad de la información disponible. Si no fuera así, no se estaría evitando realmente el *overfitting*.

Mientras que con datos numéricos se suelen interpolar nuevos datos en base a una función determinada o simplemente duplicar los datos existentes, en el caso de imágenes los métodos varían. Para ampliar la cantidad de información disponible se han generado nuevas imágenes aplicando las siguientes transformaciones:

- **Rotación**. Se gira la imagen de manera aleatoria en un intervalo comprendido entre los -10º y los 10º. Para mantener el tamaño original se recortan las esquinas y se completa con negro o blanco las partes vacías. Por ello es necesario asegurarse previamente de que los *bounding boxes* de los objetos no estén situados en los bordes recortados de la imagen.
- **Perspectiva**. Agregando variabilidad a la perspectiva de la imagen se hace que el modelo sea más resistente a balanceos de la cámara. Se ha hecho un cambio aleatorio dentro del intervalo comprendido entre los -15º y los 15º tanto en horizontal como en vertical.
- **Saturación**. Variando la saturación se modifica la nitidez y vitalidad de los colores hasta en un 25% en positivo o negativo.
- **Brillo**. Aumentando o disminuyendo el brillo en un máximo del 25% se consigue que el modelo sea más resistente a cambios de iluminación, es decir, a la hora del día en que es tomada la imagen y a la configuración de la cámara.

Existen otras técnicas de *data augmentation* como pueden ser la conversión a escala de grises, difuminar la imagen (efecto desenfoque), cambios aleatorios en los colores, adición de ruido, etc. Sin embargo, solo se han aplicado las cuatro transformaciones explicadas por ser las más comunes y las que suponen un mayor impacto.

Ilustración 12: Ejemplo de las técnicas de data augmentation aplicadas. Fuente: [17].

Para aumentar la cantidad de datos mediante *data augmentation* se ha hecho uso de Roboflow, herramienta que ya se comentó cuando se explicó todo el preprocesado realizado sobre el *dataset* en secciones anteriores. Concretamente, se han duplicado el número de imágenes disponibles, eso sí, únicamente las del subconjunto *train*, que son las ilustraciones que sirven para entrenar. No se debería generar datos nuevos artificialmente en el subconjunto de testeo porque estos se utilizan para medir las prestaciones del modelo, y por ello deben ser tan fidedignos a la realidad como sea posible. Tras aplicar esta técnica se dispone de un total de 1226 imágenes de entrenamiento, 175 de validación y 87 de testeo.

Para terminar, la otra técnica aplicada para mejorar las prestaciones del modelo es la llamada *transfer learning*. Esta consiste en entrenar un modelo con un conjunto de datos y utilizar esta información aprendida para resolver un problema distinto, aunque similar. El hecho de entrenar una arquitectura sobre un problema y utilizarla en la resolución de otro puede sonar contradictorio en un principio, no obstante, en la práctica ha demostrado proporcionar grandes resultados aun cuando el problema a resolver se diferencia significativamente del anterior.

Cuando se aplica esta técnica, simplemente se tienen que cambiar y entrenar las últimas capas de la red, las encargadas de proporcionar la salida del modelo. En este trabajo, se ha hecho *transfer learning* utilizando la arquitectura de YOLOv5 que mejores resultados había proporcionado para la detección de objetos, una sola clase, para resolver el problema del reconocimiento de objetos, cuatro categorías. De esta forma se aprovecha su capacidad para extraer las características de la imagen, añadiendo el clasificador en las capas finales para diferenciar entre las distintas clases disponibles.

3.- Análisis de los resultados obtenidos

En secciones anteriores ya se comentó que se ha decidido hacer dos tipos de experimentos. Estos, aunque similares, están diferenciados entre sí por la naturaleza del problema. En este sentido, no tiene lógica alguna analizar de forma conjunta los resultados alcanzados por las soluciones planteadas para cada ejercicio. Por ello, se expondrán en primer lugar los correspondientes a la detección de señales de tráfico, despreciando de qué marca vial se trata, y finalmente los resultados alcanzados para el reconocimiento de objetos.

Por otro lado, con anterioridad a la exposición de estos resultados, es necesario aclarar una serie de conceptos, sin los cuales no se podrá entender correctamente dicho análisis. Se trata de las métricas empleadas para la evaluación de los distintos modelos implementados. Estas van a permitir cuantificar las prestaciones de cada uno de ellos y, por tanto, estas métricas son las que van a posibilitar comparar un modelo con otro y determinar cuál de ellos es el que mejores resultados proporciona.

Todas las medidas a utilizar ([39]), aunque proporcionan distinta información, están basadas en unas nociones más simples, las cuales se describen a continuación:

- **Verdadero positivo (TP)**. Se denotará de esta forma a toda predicción correcta realizada por el modelo, es decir, aquellos objetos cuya etiqueta y predicción coinciden.
- **Falso positivo (FP)**. Se corresponde con una predicción efectuada de manera incorrecta.
- **Falso negativo (FN)**. Representa a un *bounding box*, objeto, no detectado. La diferencia entre un falso negativo y un falso positivo radica en que, mientras que en un FP el algoritmo ha sido capaz de detectar la presencia de un objeto, pero no ha sido capaz de identificarlo o ha situado el *bounding box* en una posición significativamente incorrecta, en el caso de un FN el modelo ni siquiera ha sido capaz de detectar la presencia de dicho objeto.

Estas tres ideas están estrechamente ligadas con otro concepto denominado *Intersection over Union* (IoU). Este valor mide el grado de superposición existente entre el *bounding box* real, establecido por el autor del *dataset*, y el recuadro predicho por el modelo. El IoU se encuentra acotado entre 0 y 1, donde uno representa una superposición perfecta y el cero quiere decir que no existe superposición alguna.

$$IoU = \frac{area\left(bb_{real} \cap bb_{predicción}\right)}{area\left(bb_{real} \cup bb_{predicción}\right)}$$

Su uso, sin embargo, se encuentra unido al de un umbral que permita determinar si una predicción es correcta o no. Es decir, en el hipotético caso en que el valor de IoU supere a dicho umbral, se tendría un verdadero positivo, y en caso contrario se trataría de un falso positivo. De aquí se intuye que el valor de este umbral va a tener un papel crucial en el cálculo de las prestaciones de un modelo determinado. Por ello, se ha optado en este trabajo por utilizar el valor por defecto establecido por los autores de YOLOv5.

Una vez se es capaz de determinar el número de predicciones correctas e incorrectas realizadas, así como los objetos que no se ha podido detectar (FN), ya se tiene todo lo necesario para calcular una serie de métricas que permitan cuantificar las prestaciones de los distintos modelos, tal y como se dijo anteriormente.

En primer lugar, está la matriz de confusión, una representación gráfica en forma de matriz del número de verdaderos positivos, falsos positivos, falsos negativos y verdaderos negativos. De estos últimos no se ha hablado anteriormente porque no son usados en detección y reconocimiento de objetos. Esto, junto con la aparición de un campo *background* que es algo más difícil de interpretar, hace que la matriz de confusión sea más útil y utilizada en clasificación de imágenes que en este tipo de problemas.

Aun así, debido a su importancia sigue siendo utilizada porque permite conocer directamente los valores de TP, FP y FN y, por tanto, permite calcular otras métricas a partir de ella. Además, cuando se tienen objetos pertenecientes a distintas categorías, muestra entre qué clases suele confundirse el modelo con mayor facilidad, es decir, aquellas clases que son más difíciles de diferenciar o identificar correctamente. Esta información es de especial utilidad porque a partir de ella se sabrá si hay que reequilibrar las distintas categorías o aplicar alguna otra técnica, o transformación, sobre el *dataset*.

En segundo lugar, se utilizará la denominada *precision* (P). Esta métrica mide el grado de exactitud de un modelo al identificar objetos. A mayor precisión, más confianza se tiene de que una muestra identificada como positiva, realmente lo es. Está métrica relaciona el número de predicciones correctas con todas las predicciones realizadas.

$$P = \frac{TP}{TP + FP} = \frac{TP}{todas\ las\ detecciones}$$

Otra métrica de gran importancia es el *recall* (R). Traducido como exhaustividad, hace referencia a la capacidad del modelo para detectar todos los objetos. Cuanto mayor valor de *recall* presente un modelo, más muestras positivas será capaz de identificar. En este caso se está relacionando las predicciones correctas con el número total de muestras positivas existentes, hayan sido detectadas por el algoritmo o no.

$$R = \frac{TP}{TP + FN} = \frac{TP}{todas\ las\ muestras\ positivas}$$

Estas dos características están muy relacionadas entre sí. Sin embargo, que un modelo tenga una precisión elevada no quiere decir que su exhaustividad también lo esté. Si un modelo presenta un gran valor de *precision* y un *recall* bajo, este clasificará pocas muestras como positivas, pero lo serán con mucha confianza. El problema radica en que no habrá sido capaz de identificar otras muchas muestras positivas. En el caso contrario, si se da una precisión baja y alta exhaustividad, el modelo clasificará correctamente la mayoría de las muestras positivas, pero se colarán, se detectarán erróneamente, muchas de las negativas.

En función del problema ante el que se encuentre cada desarrollador, estudiante o investigador, será decisión de cada uno de ellos determinar qué interesa en cada caso. Un elemento muy interesante y que supone una gran ayuda en estos casos es la curva PR. Se trata de una gráfica que enfrenta la precisión con el *recall*, y muestra el equilibrio entre estos dos valores. Permite conocer la evolución de una en función de la otra, y viceversa.

Una métrica exclusiva de los problemas de detección y reconocimiento de objetos, que no existe en la clasificación de imágenes, es el *average precision* o AP ([40], [41]). Podría definirse como el área bajo la curva PR evaluada considerando el umbral α de IoU ($AP@\alpha$). En este caso siempre interesa un valor lo más cercano posible a uno, estando esta métrica acotada entre cero y uno.

Conocido el AP, se denotará por mAP al *mean average precision*, la media del AP calculado individualmente para cada una de las clases disponibles. YOLOv5 proporciona dos valores distintos de mAP, uno calculado considerando un AP sobre el umbral del 50% ($mAP\_0.5$), y otro para el cual utiliza diez umbrales distintos, comprendidos entre el 50% y el 95%, y con un salto del 5% en cada paso ($mAP\_0.5{:}\,0.95$).

Existen otras muchas métricas que también son interesantes, como pudiera ser el valor *f1-score* entre otras, sin embargo, en un principio no serán estudiadas. Este trabajo se centrará inicialmente en las que proporciona YOLOv5, las que se han comentado en los párrafos anteriores.

3.1.- Detección de objetos

Para comenzar, se optó por probar la segunda arquitectura más simple entre todas las que incluye YOLOv5, la versión s o *small*, con una cantidad de épocas algo limitada, aunque bastante adecuada para comenzar la experimentación. Esta decisión de realizar un entrenamiento con una duración máxima de cien etapas está basada en la experiencia previa que se tiene con problemas de este tipo.

Tras completar dicho entrenamiento, la función proporciona cierta información tanto de las prestaciones alcanzadas por el modelo, como del proceso en sí. Entre esta última se encuentra el *class loss* y el *box loss*. Estos valores de pérdida indican cómo de buena es la clasificación del objeto encontrado en su clase correspondiente y cómo de similares son los *bounding boxes* predichos por el modelo y los reales respectivamente. En cualquier caso, interesa que ambos sean lo más cercanos posible al cero.

En este caso particular, en el que no se está diferenciando entre clases, el *class loss* es nulo durante todo el proceso, y por tanto no tiene interés alguno. Se obviará en todas las pruebas realizadas pertenecientes a esta sección, la detección de objetos. Por otro lado, el *box loss* sí que es una medida a tener en cuenta.

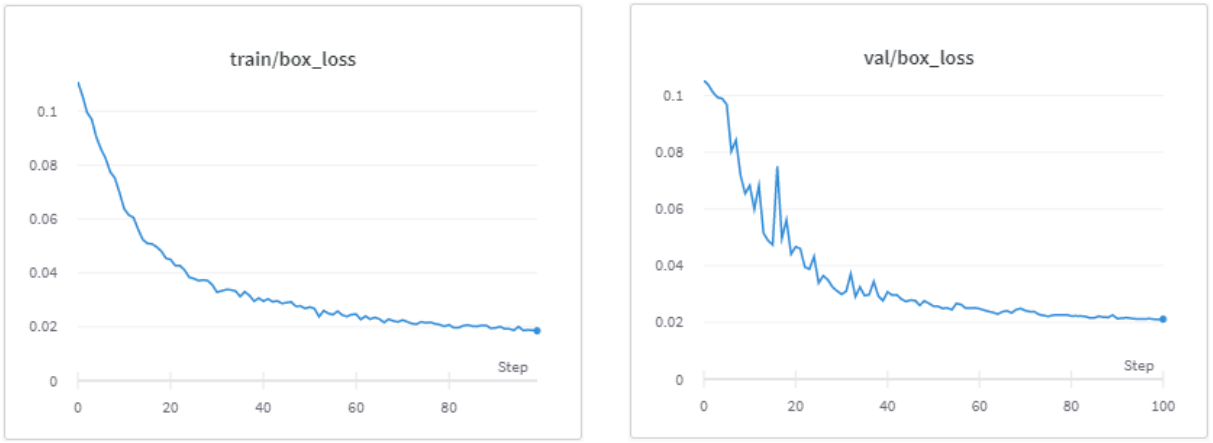

Ilustración 13: Evolución del box loss durante el proceso de entrenamiento de 100 épocas del modelo YOLOv5s sobre los subconjuntos train y validation para detección de objetos.

En las gráficas de la ilustración 13 se puede ver la evolución que ha presentado tanto en el conjunto de entrenamiento como en el conjunto de validación a lo largo de todo el proceso. Esta evolución ha supuesto un descenso de la pérdida desde un valor superior al 0.1 en las primeras etapas, a cifras que rondan el 0.02 al final del entrenamiento en ambos subconjuntos. Sin embargo, en la validación, la evolución ha sido más irregular, con subidas y bajadas repentinas más habituales y de mayor magnitud.

Además, la función de entrenamiento indica que la etapa en la que mejores prestaciones se han obtenido es la 98, prácticamente la última. Esto podría indicar que el número de épocas se ha quedado corto para la dificultad del problema a resolver, y que con más iteraciones posiblemente se hubieran conseguido mejores resultados. Aunque para afirmar esto es necesario conocer más datos acerca de dicho entrenamiento y la tendencia de cada una de las métricas presentadas.

Este modelo, ha llegado a alcanzar una precisión de 0.95008. Lo que quiere decir que se consigue un 95% de confianza de que un elemento identificado por el modelo sea realmente una señal de tráfico. Este es un porcentaje bastante elevado, más de lo esperado en una primera aproximación realizada como toma de contacto inicial. De hecho, se puede considerar que el modelo presenta una gran precisión. Por otro lado, el *recall* obtenido es de 0.86000 en su mejor iteración, lo que significa que existe un 14% de señales de tráfico que escapan a las predicciones del modelo. Aunque este valor es significativamente inferior al de la precisión, al superar ampliamente el 0.7 se puede considerar que tiene una buena exhaustividad.

En relación con el *mean average precision*, ya se ha mencionado anteriormente que YOLO presenta esta información considerando dos casos distintos. Uno en el que el umbral de IoU seleccionado es 0.5 únicamente, y otro en el que se ha calculado la media de todos los mAP trabajando con umbrales que van incrementándose progresivamente del 0.5 al 0.95. En el primer caso el valor obtenido es 0.90361, mientras que el segundo desciende hasta un 0.73913. Estos datos están indicando que con una superposición mínima del 50% entre los *bounding boxes* reales y predichos, el modelo es capaz de detectar correctamente más del 90% de las muestras. Conforme va aumentando este porcentaje de intersección mínima requerida para considerar una predicción como correcta, el porcentaje de acierto disminuye. Sin embargo, al igual que con la exhaustividad, al superar el 0.7 se considera aceptable el $mAP\_0.5{:}\,0.95$ proporcionado por el modelo.

Las prestaciones descritas se corresponden con las que brinda el modelo en la mejor etapa de entrenamiento. No obstante, observando la disposición de la evolución de estas métricas a lo largo de todo el proceso, al igual que las gráficas de pérdida mostradas en la decimotercera imagen, se intuye que todas tienen tendencia a seguir mejorando en el momento en el que acaba el entrenamiento. Por esto se piensa que el número de épocas especificado ha limitado la capacidad de la arquitectura y, consecuentemente, se repetirá el mismo entrenamiento aumentando a mil la cantidad de iteraciones permitidas incluyendo *early stopping*.

Esta técnica para dejar de entrenar en caso de observarse que el modelo no puede mejorar más ha resultado de gran ayuda puesto que se han alcanzado los mejores pesos para esta arquitectura en la iteración 832. En dicha época el *box loss* ha sido de 0.009311 para el subconjunto de entrenamiento y de 0.01776 para la validación. En ambos casos se han mejorado ligeramente los resultados anteriores. Sin embargo, se ha incrementado la diferencia entre ambos valores, lo cual no es algo que nos interese. No obstante, aunque esta distancia es lo suficientemente grande como para considerar la posibilidad de estar entrando en zona de *overfitting*, es conveniente analizar el resto de las métricas antes de llegar a esta conclusión precipitadamente.

Este segundo modelo entrenado presenta una precisión de 0.98653 y una exhaustividad de 0.88755. En ambos casos se han incrementado en más de 2.5 puntos los resultados alcanzados en el experimento anterior. Si bien esto no supone una mejora mayúscula, sí que puede llegar a ser significativa en el contexto en el que se presenta este proyecto. Con más de cuatro millones de señales de tráfico solo en carreteras, una diferencia superior al 2.5% de *recall* y *precision* entre modelos realmente puede reducir costes y reforzar la seguridad vial considerablemente.

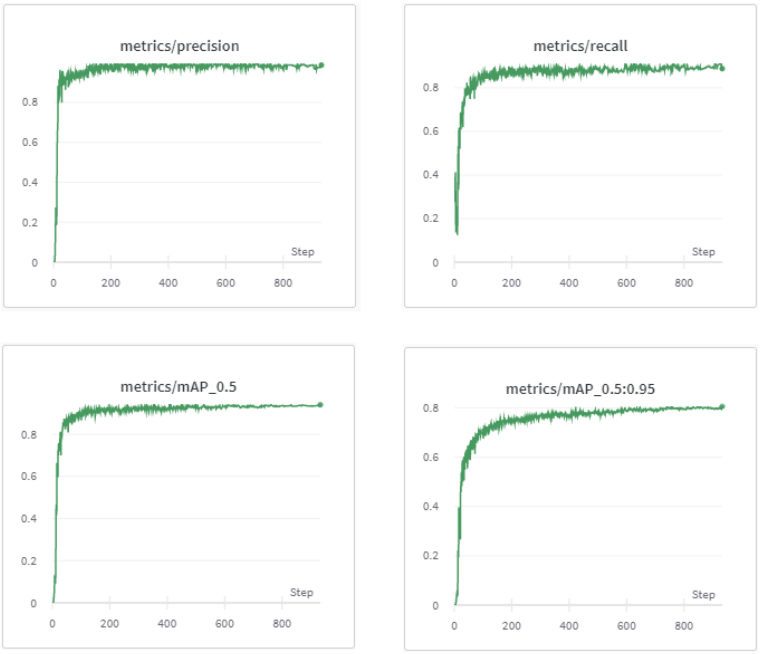

Ilustración 14: Evolución de distintas métricas durante el proceso de entrenamiento de 1000 épocas del modelo YOLOv5s sobre el subconjunto train para detección de objetos.

Además, los parámetros de $mAP\_0.5$ y $mAP\_0.5{:}\,0.95$ de este modelo marcan valores de 0.94038 y 0.80575 respectivamente. La distancia entre las marcas dadas por estas métricas con respecto al modelo anterior es aún mayor que en los casos de la precisión y exhaustividad. Concretamente, estos indicadores están diciendo que ahora el modelo es

capaz de predecir más señales de tráfico y de manera más exacta. Considerando un umbral mínimo del 50% de superposición entre *bounding boxes*, el modelo se acerca al 95% de acierto en las predicciones y, conforme va aumentando este umbral no son tantas las muestras correctas que se pierden, como sí era el caso en el primer modelo desarrollado. Ahora se supera el 80% de mAP medio utilizando umbrales que van del 50% al 95% de intersección. En realidad, se puede empezar a considerar este resultado como bueno, en lugar de simplemente aceptable.

En la ilustración 14 quedan reflejadas distintas gráficas con la evolución de las cuatro métricas estudiadas durante todo el proceso de aprendizaje. En ellas, además de verse los valores que se han estado especificando anteriormente, también se puede apreciar que estas no tienden a mejorar dichas marcas en las últimas etapas. Por tanto, aumentar aún más la cantidad máxima de épocas no va a incrementar las prestaciones del modelo. De hecho, se ha activado el *early stopping* y no se han completado la totalidad de las etapas especificadas en la ejecución. En este caso, el número de iteraciones no ha limitado la capacidad de la arquitectura.

Sin embargo, cada uno de estos valores, así como las gráficas de la decimocuarta ilustración, son correspondientes al entrenamiento del modelo. Esta información es de gran ayuda y observando su evolución se ha podido determinar que no es necesario aumentar el número de épocas de entrenamiento, pero hay que tener cuidado al analizar estos datos. No se puede afirmar que estas sean las prestaciones reales del modelo, puesto que las muestras utilizadas para calcularlas ya eran conocidas. Para ello se debe utilizar el subconjunto de testeo, muestras completamente nuevas que van a medir la capacidad de generalización del modelo y su eficacia.

Realizando predicciones sobre dicho subconjunto, los datos revelan que su precisión real es de 0.968 en lugar del 0.98653 que indicaban las muestras de entrenamiento. Aun así, sigue siendo un valor muy bueno y que no es nada desdeñable para este parámetro. Su *recall*, por el contrario, se ve potenciado hasta marcar un valor de 0.911. En definitiva, se ha conseguido un modelo cuyas prestaciones reales, medidas sobre muestras completamente nuevas y ajenas al proceso de entrenamiento, son un 96.8% de precisión y un 91.1% de exhaustividad. Ambos valores por encima del 0.9, frontera que nos permite decir que este modelo presenta muy buen rendimiento en estos dos indicadores.

Un aspecto interesante por estudiar es si seleccionando otros pesos, los de otra etapa del entrenamiento, se puede de alguna forma aumentar el *recall* para que no haya tanta diferencia entre esta medida y la precisión. Sin embargo, en la ilustración 15 se observa la curva PR de este modelo. En ella se ve como la precisión cae drásticamente al acercarse al

1 y, sobre todo, al pasar del 0.9 en exhaustividad. Esto significa que no se puede incrementar más uno de estos dos parámetros sin perjudicar al otro más de lo que se ha mejorado el anterior.

En este sentido, YOLO proporciona de manera automática los pesos en los que más equilibradas están estas dos métricas. En función del problema podría ser conveniente potenciar una por encima de la otra, sin embargo, este no es el caso que aquí se presenta. Además, como se ha dicho en el párrafo anterior, ambos indicadores superan el 90% y esto es indicativo de que se tiene un muy buen modelo.

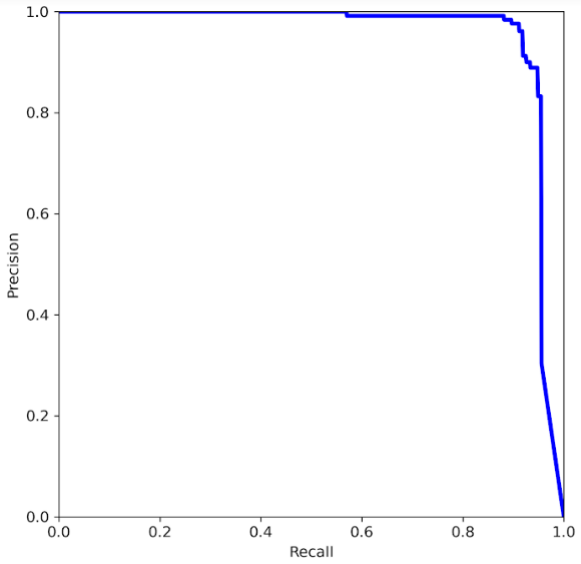

Ilustración 15: curva PR del modelo YOLOv5s para detección de objetos con un entrenamiento de 1000 épocas.

Con respecto al mAP, se puede decir que también se han obtenidos resultados completamente satisfactorios. Estos valores, no sólo no han disminuido al cambiar de muestras, sino que en el caso del $mAP\_0.5$ se ha visto incrementado ligeramente. Se ha obtenido un *mean average precision* del 95.4% sobre dicho umbral. Por otro lado, al haber aumentado esta medida y mantenerse en el 80% el $mAP\_0.5{:}\,0.95$, ha disminuido el mAP con cotas mínimas de superposición elevadas. Esto quiere decir que los *bounding boxes* predichos son sutilmente menos exactos, algo lógico al estar utilizando imágenes nunca vistas anteriormente por el modelo.

En resumen, YOLOv5s con este entrenamiento ha resultado ser un modelo realmente bueno para la detección de señales de tráfico, teniendo en cuenta únicamente las que están etiquetadas en el *dataset* original. Este modelo presenta unas prestaciones más que satisfactorias que han sido alcanzadas con un tiempo de entrenamiento aproximado de cien minutos.

Para mejorar lo conseguido hasta ahora no tendría sentido aumentar la cantidad de épocas de entrenamiento por razones que ya se han comentado. Así que el siguiente paso será utilizar una arquitectura más profunda, YOLOv5m o *medium*, la cual permitirá la identificación de patrones más complejos dentro de las imágenes. También cabe destacar que los parámetros especificados para el entrenamiento y el testeo del nuevo modelo serán los mismos que se han utilizado en este último experimento, del mismo modo que estos serán los empleados durante todas las pruebas realizadas a partir de este momento, ya sea para detección o reconocimiento de objetos.

Con la arquitectura YOLOv5m tan solo se han requerido 271 épocas para alcanzar el pico de rendimiento. Esto hace que el tiempo de entrenamiento, a pesar de ser una estructura más compleja, se haya visto reducido con respecto al modelo anterior. En este caso han sido necesarios algo menos de 58 minutos, una hora aproximadamente. Sin embargo, al haberse incrementado la profundidad del modelo y el número de neuronas y pesos, se requiere de media un 25% más de uso de GPU y un 15% de memoria adicional respecto a la versión *small* de esta familia que supone YOLO.

Aunque las etapas necesarias para entrenar este modelo han sido un tercio de las requeridas por el modelo anterior, los resultados son muy similares. Esta nueva estructura simplemente ha convergido con mayor rapidez que la anterior. No obstante, este hecho unido a una mayor profundidad, no quiere decir que las prestaciones sean superiores. El *box loss* alcanzado en el entrenamiento es de 0.01284, mientras que en la validación su valor se ha quedado en 0.02054. Ambos superiores a la pérdida mostrada por YOLOv5s.

Por otro lado, la precisión demostrada sobre el conjunto de entrenamiento también está ligeramente por debajo con una tasación de 0.97329. En contraposición, sí que se ha obtenido un mejor *recall*, que se ha quedado justamente en el 0.90000. El punto que ha desaparecido en un parámetro se le ha añadido al otro. De esta forma se ha reducido la diferencia entre ambas y el modelo queda más compensado.

En cambio, el mAP se ve decrementado independientemente del umbral utilizado. Y esta inferioridad se ve acentuada cuanto mayor es el porcentaje de intersección requerido entre los *bounding boxes* para considerar una predicción como correcta. Dicho de otro modo, este tercer modelo no es tan exacto detectando las señales de tráfico como el anterior, al menos en el conjunto de entrenamiento. Su $mAP\_0.5$ tiene un valor de 0.93863 para estas imágenes, y un $mAP\_0.5{:}\,0.95$ de 0.77802. Aún así, son resultados buenos y aceptables para todas las métricas estudiadas.

Dicho esto, ya se ha mencionado anteriormente que las prestaciones verdaderamente importantes son las medidas sobre muestras desconocidas por el modelo, las de testeo. Sobre

estas, YOLOv5m ha sido capaz de aumentar la precisión en 0.7 puntos respecto al modelo anterior, situándose ahora la precisión en el 97.5%. Sin embargo, esto ha sido posible a costa de bajar varios puntos los otros parámetros. Concretamente, el *recall* obtenido con este nuevo modelo es de 0.879, un 3.2% inferior. Esto quiere decir que ahora se tiene un poco más de certeza de que una predicción está detectando correctamente una señal de tráfico, pero en contraposición existe un mayor número de marcas viales que escaparan a este modelo. Además, el $mAP\_0.5$ ha bajado hasta el 0.931 y el $mAP\_0.5{:}\,0.95$ hasta un 0.758, por lo que los *bounding boxes* serán menos precisos.

En definitiva, haber aumentado la complejidad y la profundidad de la estructura utilizada no ha conseguido los efectos esperados. Aunque es cierto que para el conjunto de entrenamiento existen ciertas medidas que superan a las del modelo anterior, cuando se comprueba la eficacia sobre las imágenes de testeo, esta versión mediana únicamente supera a la anterior en precisión, siendo este adelanto inferior a lo que se pierde en el resto de las métricas.

Esto descrito anteriormente puede tener varias lecturas e interpretaciones. En este caso, se considera que la más acertada es que los objetos que se pretenden detectar, las señales de tráfico, y las imágenes en las que estas aparecen no son lo suficientemente complejas como para requerir estructuras tan profundas. Estas van a tender a buscar patrones más enrevesados y complicados que pueden no ser adecuados en este caso. Otra posible interpretación sería que se entra en zona de sobreentrenamiento y por eso el modelo no es capaz de generalizar adecuadamente, a pesar de que con las imágenes ya conocidas sí mantiene las prestaciones, e incluso las mejora en ciertos indicadores. En cualquier caso, la solución pasa por la utilización de una estructura menos profunda, más simplificada, es decir, el modelo *small* utilizado anteriormente. Aun así, como la diferencia encontrada no es demasiado significativa, se piensa que es conveniente tantear la arquitectura *large* o l para estar seguros de que esto sea así.

Tras entrenar YOLOv5l, nuevamente con un máximo de 1000 épocas, los resultados obtenidos han sido bastante parecidos a los de la variante mediana o *medium*. Esto es, unas métricas sobre el entrenamiento que se mantienen con respecto a los modelos anteriores, e incluso mejoran en algunos casos, pero que no perseveran al estudiar el subconjunto de testeo.

Finalmente, para poder comparar mejor la eficacia y eficiencia de estos modelos se han creado distintas gráficas, correspondientes a las ilustraciones 16, 17 y 18. De esta forma se podrá comprobar visualmente las diferencias entre los resultados obtenidos entrenando cada una de las tres arquitecturas probadas.

En la primera imagen de este trío, la número 16, se puede apreciar un diagrama radial de cuatro lados, en el que quedan reflejadas las prestaciones obtenidas por cada modelo sobre el subconjunto de testeo. Cada vértice representa una de las métricas estudiadas. En este sentido se aprecia que no hay discusión alguna, YOLOv5s destaca por encima del resto en cada parámetro, con excepción únicamente de la precisión, siendo esta la variable más igualada. Es decir, la arquitectura *small* presenta un valor inferior en este indicador, pero no existe tanta diferencia con respecto al resto de estructuras como en los otros valores medidos, en los que sí que destaca claramente la arquitectura más simple por encima del resto. Por tanto, se considera que en términos de eficacia y exactitud haciendo predicciones en la tarea de detección de objetos, YOLOv5s es el mejor modelo encontrado hasta el momento.

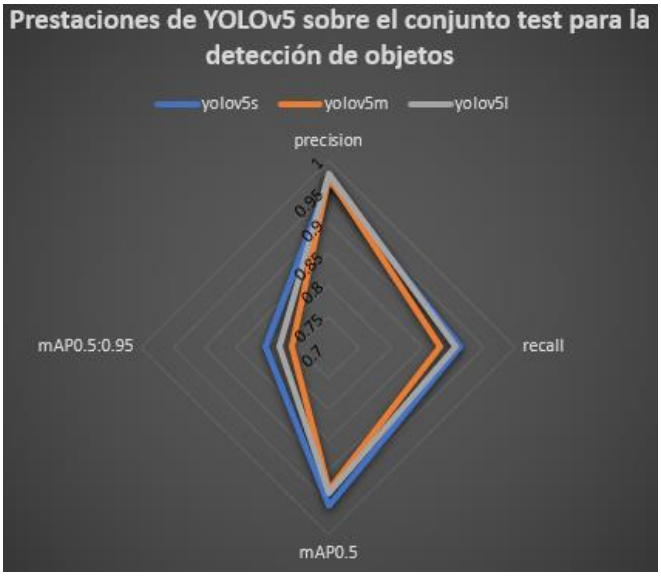

Ilustración 16: Prestaciones ofrecidas por los modelos de YOLOv5 sobre el conjunto test para la detección de objetos.

Por otro lado, se ha calculado el tiempo empleado en el entrenamiento de cada uno de estos modelos, así como el número de épocas completadas antes de la finalización de dicho proceso. Considerando el *early stopping* programado, en el que debían realizarse un 10% del total de etapas sin mejorar prestaciones antes de detener el aprendizaje del modelo, se puede suponer que se han completado cien iteraciones más del algoritmo a partir de la mejor época notificada.

En el gráfico de la ilustración 17 se observa que el modelo YOLOv5s ha requerido un total de 932 etapas de entrenamiento, más del doble de la arquitectura mediana o *medium* y superando ampliamente a la versión *large*. Aun así, su tiempo de entrenamiento es muy competitivo, superior al de YOLOv5m, pero inferior al de la estructura más compleja y profunda. Se puede afirmar sin miedo a equivocación que esto es así porque, al presentar menor complejidad y profundidad, este modelo requiere menos tiempo de entrenamiento

por época. De hecho, realizando un sencillo calculo se comprueba que YOLOv5s tan solo necesita 6.46 segundos de entrenamiento por etapa, mientras que YOLOv5m y YOLOv5l tardan 9.35 y 14.36 segundos respectivamente para finalizar una única iteración del proceso de entrenamiento.

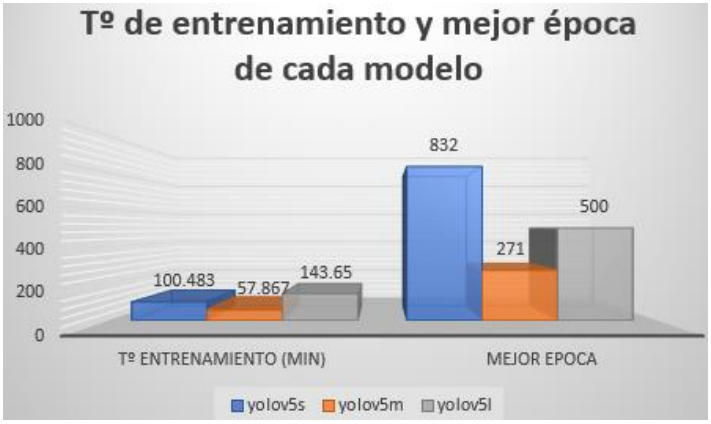

Ilustración 17: Tiempo de entrenamiento requerido por los modelos de YOLOv5 para la detección de objetos y mejor época encontrada en dicho entrenamiento.

En resumen, YOLOv5s es el modelo que menos tiempo necesita por etapa de entrenamiento para la detección de objetos llevada a cabo en este trabajo, considerando el conjunto de datos utilizado. Sin embargo, la arquitectura *medium* es la que menos tiempo ha requerido para finalizar al completo su proceso de entrenamiento. Esto debido a la rápida convergencia que ha presentado, ha necesitado aproximadamente un tercio de las épocas requeridas por YOLOv5s para alcanzar sus mejores prestaciones en este problema, con un tiempo medio un 50% más lento por iteración.

Ilustración 18: Utilización de GPU y memoria requerida por los modelos de YOLOv5 para la detección de objetos.

Además, en las gráficas de la ilustración 18, quedan reflejados ciertos aspectos relacionados con la utilización de la GPU por parte de cada modelo. En el esquema de la izquierda de esta ilustración se encuentra la utilización de dicha unidad de procesamiento

gráfico. Aquí se observa cómo, a medida que se va haciendo más compleja la arquitectura de un modelo, más utilizará esta componente. Sin embargo, mientras que el salto sí es considerable entre la versión s y m de YOLO, no ocurre lo mismo entre esta última y la arquitectura *large*. Parece ser que ambas tienen una media de utilización que ronda el 75%, no obstante, YOLOv5l tiene picos más pronunciados y YOLOv5m es más consistente, tiene menos variación con respecto a dicha media. En cambio, YOLOv5s tiene una media de utilización de GPU cercana al 50%, con una variación aún menor que la de las arquitecturas anteriores.

Por otro lado, en la gráfica de la derecha de la imagen 18 queda reflejado que, a mayor complejidad y profundidad presente el modelo, mayor será el porcentaje de memoria de gráficos que este requiera. Mientras que YOLOv5s no llega al 40% de la memoria disponible de la GPU prestada por Google, el modelo *large* consume prácticamente el doble de este recurso, si no más. Su función queda justo por debajo del límite del 80%.

Tras finalizar este estudio experimental de las distintas arquitecturas ofrecidas por YOLO, se concluye que, para el problema de detección de objetos, en el caso que aquí se presenta, YOLOv5s es el más idóneo. Esta decisión no es debida únicamente a que esta versión sea la que ofrece mejores prestaciones en general sobre el subconjunto de testeo o prueba, sino también a otra serie de factores que se han analizado a lo largo de esta sección. Entre estos, destacan el menor uso de la tarjeta gráfica, seguramente la componente hardware más costosa de todas las necesarias, y una muy buena relación entre el número de épocas requeridas para su entrenamiento y el tiempo empleado en cada una de estas.

Finalmente, en la siguiente figura (ilustración 19) se muestra un mosaico de imágenes en las que aparecen las predicciones realizadas sobre ellas por el que se ha destacado como mejor modelo de entre los estudiados, YOLOv5s. Todas estas imágenes son una muestra obtenida a partir del subconjunto de testeo. Cabe destacar que al lado de cada *bounding box* se encuentra un porcentaje que representa la certeza del modelo al afirmar que en esa posición se encuentra el objeto buscado. También es importante recordar que en el *dataset* inicial únicamente se encuentran etiquetados los semáforos, las señales de stop, las señales de paso de peatones y las señales que indican límites de velocidad. Es por esto por lo que el modelo no debería remarcar ningún otro tipo de señalización. Además, estas cuatro clases se unificaron en una gran categoría única para transformar el problema en una detección de objetos sin diferenciación entre señales, por lo que aparecerán todas marcadas con la etiqueta "*traffic sign*".

Ilustración 19: Predicciones de ejemplo del modelo YOLOv5s para detección de objetos sobre imágenes sacadas del subconjunto de testeo.

3.2.- Reconocimiento de objetos

Una vez estudiada la detección de objetos, ha llegado el momento de complicar el problema y analizar si los modelos de YOLO son adecuados y pueden obtener resultados satisfactorios en tareas de reconocimiento de objetos. Aunque esta dificultad añadida que supone identificar qué objeto es el representado va a hacer que la eficacia alcanzada por las distintas arquitecturas decaiga, se aplicarán distintas técnicas con el fin de maximizar dichas prestaciones.

Para comenzar, se empleará la estructura YOLOv5s. Se piensa que es una buena primera aproximación al problema empezar con aquella estructura que ha demostrado ser la mejor en la tarea anterior, siendo esta segunda una ampliación de la primera. Además, el entrenamiento se ha llevado a cabo utilizando los mismos parámetros que en los casos anteriores, incluido el número de etapas de entrenamiento. No tendría sentido volver a comenzar por cien iteraciones únicamente cuando anteriormente ya fue insuficiente esta cantidad y ahora el problema a resolver es más complejo. Tan sólo hay una diferencia, la ruta al archivo de configuración del *dataset*. En este caso hay que utilizar las anotaciones que se tenían antes de modificar la clase de los *bounding boxes*, es decir, se deben utilizar anotaciones que identifiquen a cada objeto por la señalización que representa.

De nuevo, al igual que cuando se analizaba la detección de objetos, YOLO proporciona como salida del algoritmo de entrenamiento distintas gráficas pertenecientes a funciones de pérdida. El modelo *small* ha obtenido, en este caso, un *box loss* de 0.01672 sobre el subconjunto de entrenamiento y un valor de 0.01989 para el de validación. En un principio, estos valores son bastante buenos, de hecho, son muy similares a los obtenidos con el mejor modelo en la sección anterior.

Además, ahora entra en juego otra función de pérdida que antes era despreciada al no estar trabajando con distintas clases, el *class loss*. Esta indica cómo de buena es la clasificación de los objetos encontrados por parte del modelo. YOLOv5s ha conseguido alcanzar una pérdida casi nula durante el entrenamiento, disminuyendo continuamente el valor de esta función hasta el 0.0008058 conseguido finalmente. Sin embargo, en el conjunto de validación, aunque también se ha obtenido un valor cercano al cero, este se encuentra un orden de magnitud por encima del anterior, el *class loss* de validación es de 0.00517.

Por otro lado, los pesos que han proporcionado un mejor rendimiento son los correspondientes a la época 322. La misma arquitectura ha requerido menos iteraciones para converger realizando una tarea más complicada que otra más sencilla. Sin embargo, esto no quiere decir que las prestaciones también sean mejores, podría darse la situación en que estas fueran nefastas. Dicho esto, este no es el caso. Es cierto que se obtienen peores

métricas que para detección de objetos, algo que ya se esperaba dada la dificultad añadida, pero estas siguen presentando valores aceptables, como se analizará a continuación.

Sobre el subconjunto de datos de entrenamiento se ha obtenido una precisión de 0.9754, prácticamente la misma que obtuvo el mismo modelo en la detección de objetos. No obstante, la precisión es el único indicador que consigue mantenerse, el resto sufren una caída de diez puntos aproximadamente. La exhaustividad ha alcanzado su máximo en 0.80534, mientras que el $mAP\_0.5$ y $mAP\_0.5{:}\,0.95$ se han quedado en 0.8555 y 0.69906 respectivamente. Aun así, se alcanza el 80% en la mayoría de los casos. Simplemente hay que tener en cuenta que los recuadros predichos no son tan precisos como anteriormente, y por eso existe una menor superposición con los *bounding boxes* reales. El modelo pierde exactitud en la ubicación del recuadro al centrarse en la identificación y el reconocimiento de cada señal. No obstante, dicha exactitud es un poco indiferente para este trabajo, lo que se persigue es detectar e identificar la señalización, no conseguir la situación perfecta del *bounding box*.

Sin embargo, todos estos datos se han obtenido sobre las imágenes de entrenamiento, y son las de testeo las que realmente importan y aportan información útil. Aquí las prestaciones medidas son: 0.924 en precisión; 0.831 en *recall*; 0.856 en $mAP\_0.5$ y 0.654 en $mAP\_0.5{:}\,0.95$. Resulta curioso observar que, mientras que algunas han disminuido ligeramente, efecto esperado al tratarse de imágenes desconocidas, otras métricas sí que se han mantenido, e incluso visto reforzadas ante estas muestras. Este hecho no hizo más que reforzar la idea preconcebida de aumentar la complejidad del modelo utilizado. Por ello, a continuación, se repitió el experimento varias veces con las mismas condiciones iniciales, salvo la utilización de arquitecturas correspondientes a modelos más profundos y pesados.

En las ilustraciones 20, 21 y 22 se puede encontrar un resumen de las prestaciones ofrecidas por cada uno de los modelos probados. Sin embargo, antes de entrar en detalle en qué es lo que se observa en cada gráfico, es importante mencionar que para completar el análisis de cada modelo se ha calculado una nueva métrica, el *f1-score*. Esta medida, aunque no tiene una interpretación directa sobre las predicciones de un modelo, resulta de gran interés por representar una media armónica entre la precisión y exhaustividad.

En la primera ilustración perteneciente al trio nombrado en el párrafo anterior, la 20, se puede ver que YOLOv5m no consigue alcanzar una precisión superior a la de la versión *small*. De hecho, este modelo se ve superado en este indicador tanto por la arquitectura inmediatamente inferior como por la superior, *large*, que mantiene unos niveles de precisión

prácticamente idénticos al modelo pequeño. No obstante, la diferencia entre la estructura mediana y las otras dos en este parámetro es únicamente de medio punto, un 0.5%.

Sin embargo, esta pérdida que sufre el modelo mediano en precisión se ve recompensada con una ganancia aún mayor en exhaustividad, se encuentra una distancia hasta de dos puntos en este apartado. Esto hace que el *f1-score* de YOLOv5m, como medida que se ve afectada por igual tanto por el *recall* como por la precisión, tenga un valor más elevado que el de los otros dos modelos probados. La arquitectura *large*, por su parte, sigue con unas prestaciones parejas a la versión *small* también en estas dos métricas.

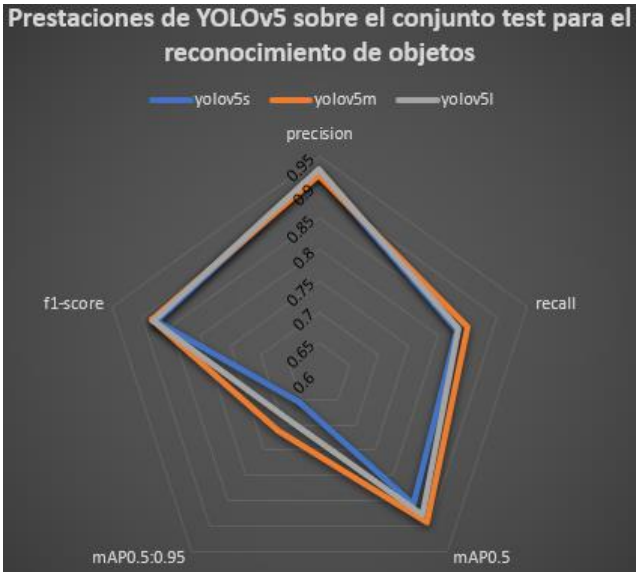

Ilustración 20: Prestaciones ofrecidas por los modelos de YOLOv5 sobre el conjunto test para el reconocimiento de objetos.

Por otro lado, observando los dos mAPs, la versión *medium* es claramente superior a las otras dos arquitecturas. Además, esta diferencia entre modelos aumenta en el $mAP\_0.5{:}\,0.95$, lo que está significando que los *bounding boxes* predichos por la estructura m son mucho más exactos. YOLOv5l, aunque sí es capaz de superar al modelo *small* en estos apartados, supone un retroceso con respecto al mediano. Por tanto, en base a esta información que aporta la ilustración 20, se considera que no conviene seguir aumentando la profundidad y complejidad de los modelos porque lo que se observará será una tendencia decreciente en cuanto a prestaciones.

Analizando ahora la gráfica de la siguiente imagen, la ilustración 21, llama la atención como el modelo más simple es el que menos épocas de entrenamiento ha necesitado, tan solo 422. Cabe recordar en este momento que según el *early stopping* especificado, cada modelo ha entrenado cien etapas más a partir de la que es marcada como "mejor época". Por su parte, YOLOv5m ha requerido más iteraciones de entrenamiento para converger y alcanzar su óptimo que un modelo más complejo como es el *large*. No obstante, esta

diferencia de etapas es prácticamente despreciable, únicamente 28 en un entrenamiento de más de 700 épocas.

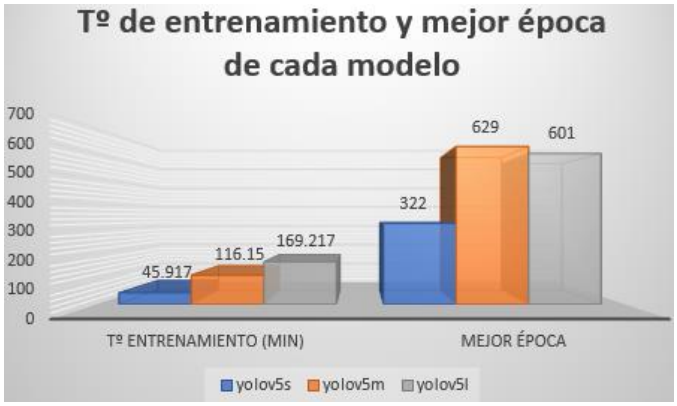

Ilustración 21: Tiempo de entrenamiento requerido por los modelos de YOLOv5 para el reconocimiento de objetos y mejor época encontrada en dicho entrenamiento.

Aun así, los tiempos de entrenamiento sí que aparecen escalonados de tal forma que el modelo más simple ha requerido un menor tiempo, y el más complejo, uno mayor. De todas formas, más interesante aún es conocer el tiempo requerido de media por cada etapa en función del modelo. En este aspecto, la versión *small* tarda 6.52 segundos en completar una única iteración del proceso de entrenamiento. El modelo *medium* aumenta este tiempo hasta los 9.55 segundos, y el *large* aún más, hasta los 14.48 segundos de media.

Observando que este crecimiento no es lineal, no parece descabellado suponer que, en caso de utilizar una estructura más compleja, el tiempo necesario para entrenarla adecuadamente sería más elevado de lo que parece razonable. Esto no hace más que confirmar la decisión tomada de no seguir aumentando la profundidad de la estructura porque se estaría entrenando un modelo tan complejo que, a priori, no se va a adaptar bien a las características del problema, sus prestaciones van a ser incluso peores que la de los modelos más simples, y el tiempo requerido para completar dicho entrenamiento sería superior al deseable.

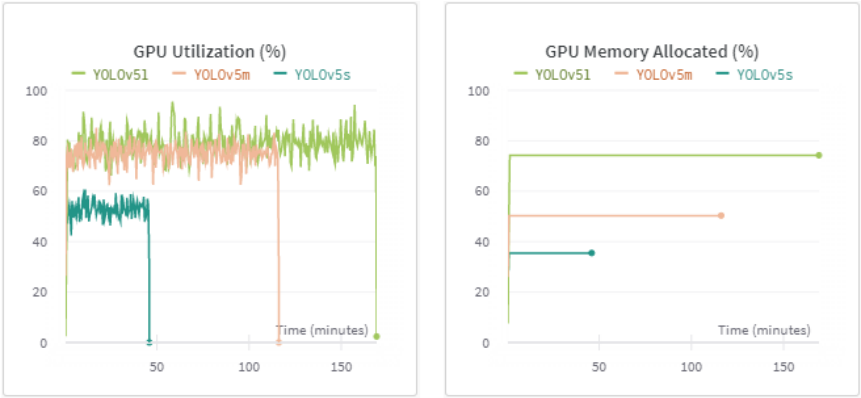

Ilustración 22: Utilización de GPU y memoria requerida por los modelos de YOLOv5 para el reconocimiento de objetos.

Además, con un modelo más complejo de los ya utilizados, según muestra la gráfica de ocupación de memoria de la ilustración 22, se rozaría el 100% de utilización de la memoria disponible. Esto, si no desborda, algo no descartable viendo la tendencia y las distancias entre los modelos estudiados.

Con la información presente en estas dos últimas gráficas se puede concluir que el mejor modelo en este caso es el YOLOv5m. No es solo que tenga los mejores valores para las métricas y estimadores medidos, y que presente una proporción razonable entre el número de épocas requeridas y el tiempo de entrenamiento necesario, sino que también hace un uso adecuado de la GPU, aprovechando sus características sin demandarla en exceso de manera que apure hasta el límite sus capacidades.

En resumen, si solamente se detecta una única clase, *traffic sign*, es decir, si el problema que se pretende solucionar es la detección de objetos, el mejor modelo obtenido es YOLOv5s. En caso contrario, si el objetivo es identificar las distintas señales de tráfico en lugar de detectarlas únicamente, si el problema a afrontar es el reconocimiento de objetos, el mejor modelo conseguido es YOLOv5m.

Una vez llegada a la conclusión de que YOLOv5m es el mejor modelo entre los estudiados para el reconocimiento de objetos, es conveniente mostrar sus características y especificaciones exactas. Con este fin se incluye la siguiente tabla (tabla 1):

-	Run summary						Train		Val		Test			
Modelo	Nº total épocas	Mejor época	$mAP\_0.5$	$mAP\_0.5:0.95$	Precision	Recall	Box loss	Class loss	Box loss	Class loss	$mAP\_0.5$	$mAP\_0.5:0.95$	Precision	Recall	F1-score
YOLOv5m	729	629	0.86711	0.71775	0.94429	0.8237	0.00936	0.00051	0.01933	0.00098	0.894	0.712	0.919	0.849	0.8826

Tabla 1: Prestaciones del modelo YOLOv5m para reconocimiento de objetos.

Observando sus prestaciones, se aprecia que no existe una diferencia significativa entre los valores mostrados durante el entrenamiento y los obtenidos utilizando los datos de testeo. Esto indica, en un principio, que el modelo no ha sufrido problemas ni de *overfitting* ni de *underfitting* y, consecuentemente, que la complejidad de su estructura es adecuada para el problema a resolver y se adapta adecuadamente a los datos de entrada.

No obstante, con el fin de detectar sus fortalezas y debilidades para intentar mejorar sus prestaciones, puede resultar de gran interés analizar su curva PR y su matriz de

confusión. Así como el análisis de sus prestaciones (precisión, exhaustividad, $mAP\_0.5$ y $mAP\_0.5{:}\,0.95$) medidas en función de cada clase por separado.

A diferencia de la curva PR mostrada en la decimoquinta imagen, la gráfica de la ilustración 23 contiene distintas funciones. Esto debido a que en este caso se está tratando con distintas clases y YOLO es capaz de calcular este valor para cada clase por separado. De esta forma se puede comprender mejor el comportamiento del modelo.

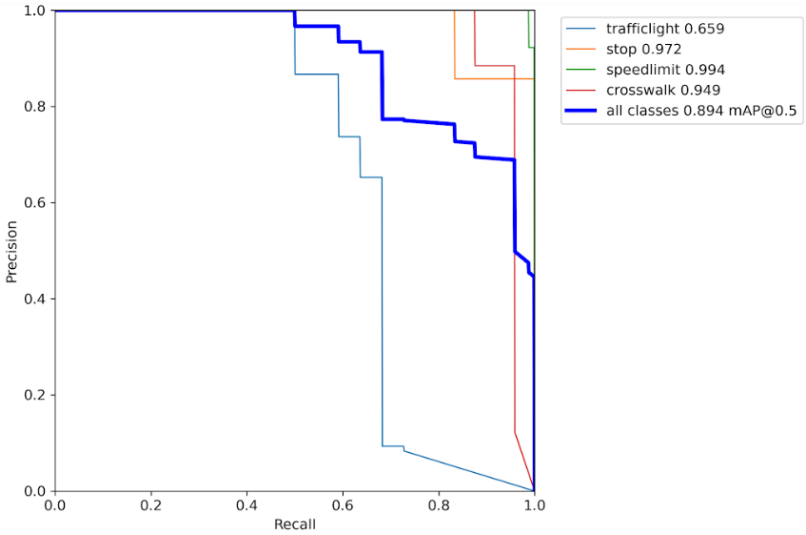

Ilustración 23: curva PR del modelo YOLOv5m para reconocimiento de objetos con un entrenamiento de 1000 épocas.

Aparentemente, según la curva PR, el modelo es capaz de obtener una alta precisión a la par que un *recall* muy elevado en todas las clases, con la única excepción de los semáforos. Aquí, el valor de una métrica disminuye drásticamente a poco que la otra comienza a aumentar y viceversa. De hecho, esta clase tira del resto y hace que la media se vea ampliamente perjudicada. De no ser así, se obtendría una curva general muy buena ya que tanto las señales de *stop* como los límites de velocidad y la señalización de paso de peatones son identificadas por el modelo con gran precisión y exhaustividad al mismo tiempo.

Ahora bien, a priori, no se indica en ninguna parte si el modelo entrenado ha seleccionado por defecto una configuración con gran precisión o, por el contrario, ha decidido potenciar el *recall* para la clase problemática. Para averiguar esto, pues tiene su importancia de cara al problema que se intenta resolver, hay que obtener el desglose de prestaciones por categoría.

En dicho informe, presente en la ilustración 24 de este documento, se aprecia que YOLO está configurado de tal forma que priorice la precisión en estos casos. Mientras que el modelo obtenido presenta un valor de 0.896 en dicha métrica, tan solo tiene un *recall* de 0.447 en cuanto a identificación de semáforos se refiere. Las otras clases, como ya se había

deducido anteriormente, presentan muy buenos datos en ambos estimadores, con valores por encima del 0.9 en todos los casos, e incluso llegando al 1 en ciertas categorías.

```
YOLOv5m summary: 290 layers, 20865057 parameters, 0 gradients
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95:
                 all        175        250      0.944      0.824      0.867      0.718
        trafficlight        175         47      0.896      0.447      0.547      0.351
                stop        175         18      0.944      0.935      0.987      0.868
          speedlimit        175        150          1      0.999      0.995      0.899
           crosswalk        175         35      0.938      0.914       0.94      0.753
```

Ilustración 24: desglose de prestaciones por clase del modelo YOLOv5m para reconocimiento de objetos con un entrenamiento de 1000 épocas.

Una vez obtenida esta información, habrá que decidir si esto es lo que realmente interesa o no de cara al problema que se pretende solucionar. Es sabido que un modelo con baja exhaustividad y alta precisión se caracteriza por no ser capaz de detectar todos los casos, habrá muchas muestras que se escapen al modelo, pero cuando se detecte una, se hará con mucha seguridad de que la predicción es correcta. La situación contraria es que en la mayoría de las ocasiones se sea capaz de identificar los semáforos, a cambio de haber predicho erróneamente que muchos otros objetos lo son cuando no es así.

En el caso presentado en este proyecto es conveniente la primera situación, la que ya se tiene. Este trabajo de fin de máster tenía el propósito de que una señalización no detectada mediante estas técnicas fuera porque ha sufrido algún tipo de deterioro y, por tanto, requiera de mantenimiento. Entonces, interesará que cuando se haga una predicción, se haga con la máxima certeza posible, se pretende maximizar la precisión por encima del *recall*. Para todos estos casos que escapen al modelo ya existe el mantenimiento manual hecho periódicamente, que en ningún caso puede ser sustituido en su totalidad por este tipo de técnicas de visión por computador.

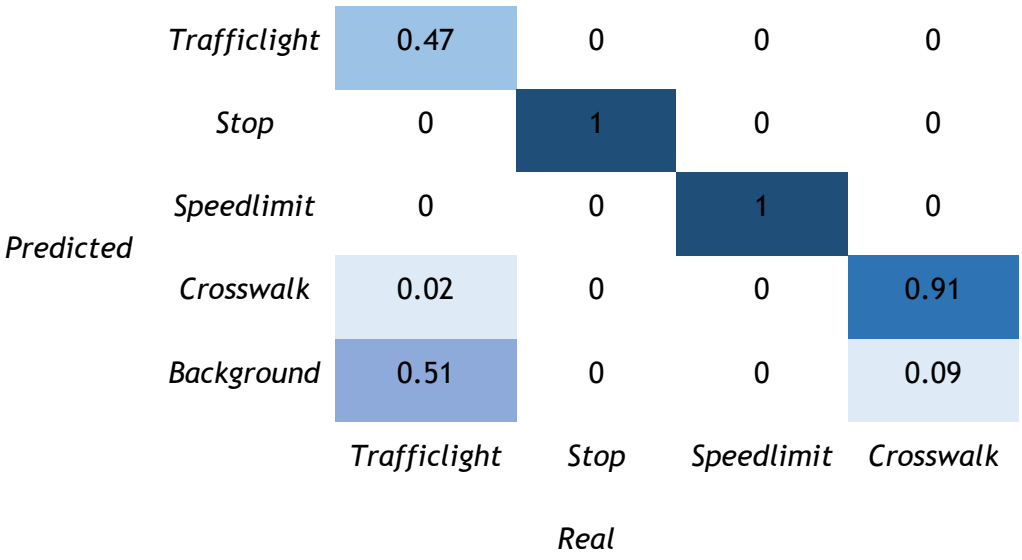

Tabla 2: matriz de confusión del modelo YOLOv5m para reconocimiento de objetos con un entrenamiento de 1000 épocas.

Finalmente, para completar esta información se puede visualizar la matriz de confusión, la tabla 2. Obviando el *background*, se observan los mayores valores, números cercanos a uno, en la diagonal principal. Sin embargo, *trafficlight* será una excepción por su bajo *recall*. En este caso se aprecia como en más de la mitad de las ocasiones en las que hay un semáforo, el modelo no lo ha detectado, es decir, predice *background*.

Además, ha sorprendido gratamente esos dos unos que aparecen en las señales de *stop* y límites de velocidad. No era esperado que el modelo fuera capaz de clasificar tan bien las señales entre sí, únicamente confunde un semáforo con una señal de paso de peatones en pequeñísimas ocasiones, un 2% de los casos.

A continuación, se pretendía mejorar aún más los resultados obtenidos. Sin embargo, aumentar la profundidad de la arquitectura y probar con el modelo xl no era una opción válida, por los motivos que ya se han comentado. Había que buscar otras alternativas para mejorar las prestaciones y, si la solución no estaba en el propio modelo, habría que modificar otros factores que también intervinieran, como los datos de entrada o las especificaciones del entrenamiento.

Una primera opción sería balancear las clases. En secciones anteriores de este documento, donde se hablaba en más detalle del *dataset*, se vio que existía un grupo de señales que estaba mucho más representada que otros. Esto puede llegar a ser perjudicial en ciertas situaciones por hacer que el modelo únicamente centre su atención en esta categoría, sin embargo, este no parece ser el caso. YOLOv5m ha mostrado excelentes resultados incluso en aquellos grupos menos representados. Por ello no se cree que este sea el origen del problema y no se realizará un balanceo de clases.

Una segunda alternativa podría ser la aplicación de técnicas de *data augmentation* para aumentar la cantidad de imágenes de entrenamiento disponibles. De esta manera el algoritmo tendría más muestras de semáforos, la categoría con la que se tiene peores resultados, y podría aprender mejor sus patrones y, consecuentemente, mostraría más facilidad a la hora de detectarlos correctamente. En un principio no se piensa que esta técnica vaya a ser efectiva porque su utilización está más relacionada con problemas de *overfitting*, no siendo este el caso aquí presente.

No obstante, se ha decidido probarla. Para ello, tras la generación de nuevos datos, se entrenará nuevamente el modelo YOLOv5m. Esta arquitectura es la elegida puesto que ha resultado ser la mejor estudiada hasta el momento para el reconocimiento de objetos con estos datos. También se realizará una prueba utilizando la versión l en lugar de la *medium* para comprobar si, al aumentar la cantidad de muestras de entrenamiento en un

100%, este modelo más complejo entra en funcionamiento y mejora las prestaciones actuales.

Adicionalmente, una tercera alternativa que puede funcionar es el conocido *transfer learning*. Esta es, a priori, la que más opciones tiene de surtir efecto puesto que, observando la matriz de confusión se vislumbra que el modelo consigue identificar muy bien los objetos. No confunde unas clases con otras. Sin embargo, tiene problemas a la hora de detectar y ubicar en el espacio de la foto los elementos de la categoría *trafficlight*. Aplicando esta técnica, realizando transferencia de aprendizaje a partir de uno de los modelos entrenados anteriormente para la tarea de detección de objetos, se pretende inculcarle esos conocimientos de detección al modelo, de forma que solo tenga que preocuparse de aprender a reconocer cada tipo de objeto, tarea que ya ha demostrado ser capaz de realizar sin problemas.

Una vez realizados estos tres experimentos que se han comentado, se ha creado una serie de gráficos como los que ya se han mostrado en otras ocasiones para poder comparar los resultados obtenidos y las prestaciones devueltas por cada nuevo modelo. En este análisis se ha incluido la mejor alternativa hasta el momento porque no se puede garantizar que estas técnicas vayan a ser efectivas de antemano.

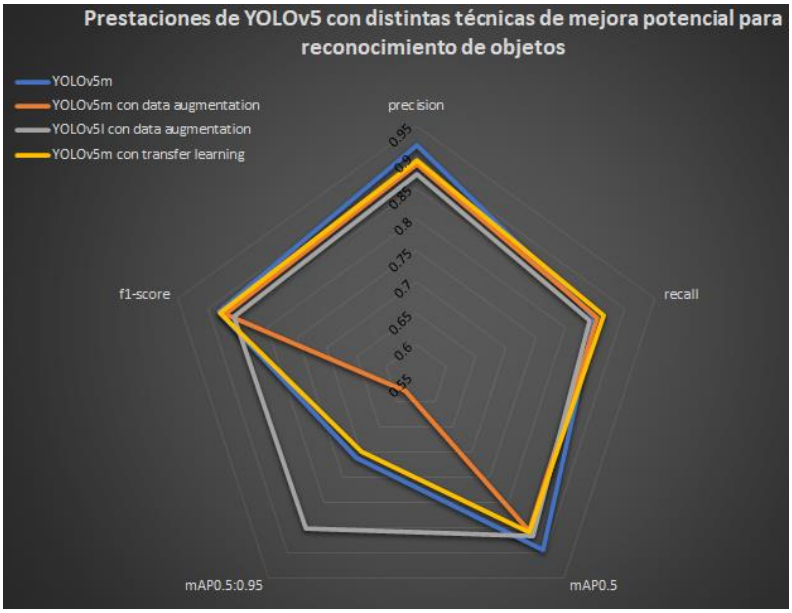

Ilustración 25: Prestaciones ofrecidas por el modelo YOLOv5m, el mejor hasta el momento, y distintos modelos probados tras aplicar ciertas técnicas para mejorar las prestaciones del anterior sobre el conjunto test para el reconocimiento de objetos.

De hecho, en el gráfico radial de la ilustración 25 se puede apreciar que las técnicas de *data augmentation* no han surtido el efecto deseado. Al trabajar con más imágenes, la estructura *medium* se ha visto perjudicada y sus prestaciones han disminuido en todos los

parámetros estudiados prácticamente, sobre todo en el $mAP\_0.5:0.95$. Sin embargo, sí que se ha mantenido en un valor aceptable cuando únicamente es considerado el umbral del 50% de superposición.

Por otro lado, YOLOv5l con *data augmentation* es el modelo más equilibrado conseguido hasta el momento, tiene todas sus métricas con valores similiares, sin una que destaque por encima del resto. No obstante, su precisión, *recall* y *f1-score* son los más bajos de todos. No sirve de nada tener un modelo muy parejo, si en las métricas que más interesan, considerando el problema que se pretende resolver, es el que menos destaca, y donde sí lo hace, es en aquellas que se podrían considerar secundarias.

Diferente resultado ha supuesto la transferencia de aprendizaje. Esta técnica sí que ha surtido el efecto deseado en el sentido de potenciar la exhaustividad de YOLOv5m. Este modelo ha aumentado el *recall* que ofrece hasta el punto de ser el mejor en este parámetro. En cambio, ha perdido bastante precisión. Observando la puntuación f1, medida que relaciona estos dos indicadores, este modelo se encuentra en cabeza junto con la arquitectura mediana de YOLO antes de la aplicación de todas estas técnicas. La diferencia entre ambos modelos radica en qué métrica es la que potencian. Sin embargo, como ya se analizó anteriormente, teniendo en cuenta la finalidad y los objetivos de este proyecto, es más interesante tener una mayor precisión que una mejor exhaustividad. Por lo tanto, se considera que YOLOv5m con el entrenamiento anterior sigue siendo el mejor modelo obtenido hasta el momento, al menos en términos de eficacia.

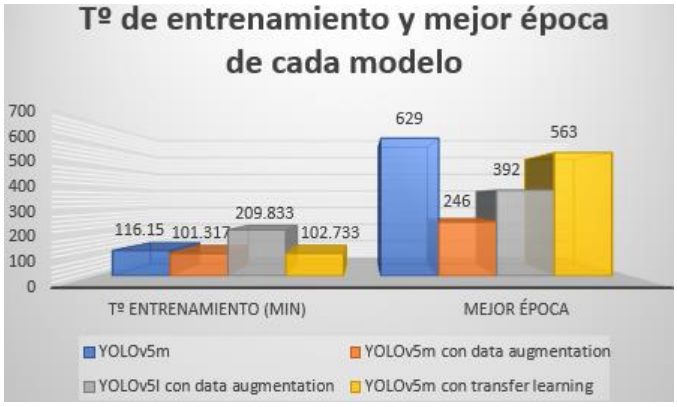

Ilustración 26: Tiempo de entrenamiento requerido por el modelo YOLOv5m, el mejor hasta el momento, y distintos modelos probados tras aplicar ciertas técnicas para mejorar las prestaciones del anterior sobre el conjunto test para el reconocimiento de objetos y mejor época encontrada en dicho entrenamiento.

En la gráfica de la ilustración 26 se observa que al aplicar la técnica de *transfer learning* el modelo requiere un menor tiempo de entrenamiento. Esto es debido probablemente a que cuando comienza dicho proceso ya parte de una base, los mejores

pesos encontrados para detección de objetos, y no de unos valores aleatorios para estas variables.

Un aspecto que sorprende es que YOLOv5m con *data augmentation* tarde tan poco, incluso menos que cuando no se aplica este recurso. Esto, a pesar de requerir un tiempo medio por etapa superior puesto que tiene que procesar el doble de imágenes (17.56 segundos/época por los 9.55 segundos/época que son requeridos sin aumento de datos). Por otro lado, YOLOv5l se comporta de la forma esperada. Su complejidad y profundidad, unido a la duplicación de la cantidad de datos que procesar, ha provocado que su tiempo de entrenamiento sea el mayor de todos, necesitando casi tres horas y media de ejecución.

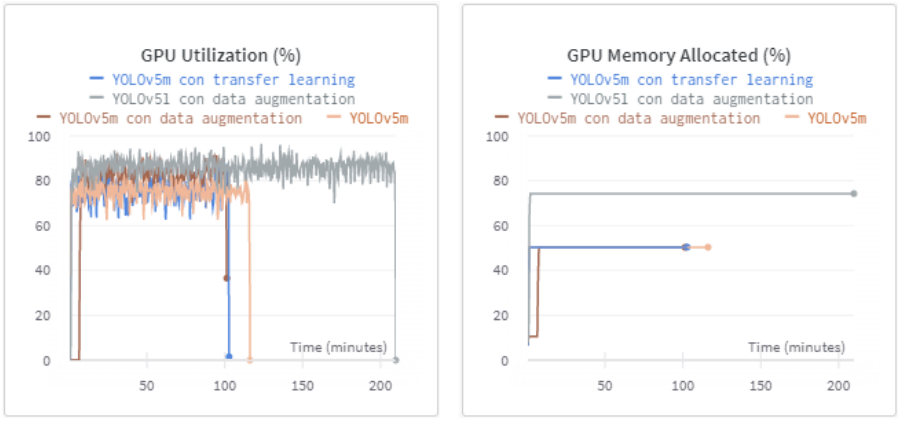

Ilustración 27: Utilización de GPU y memoria requerida por el modelo YOLOv5m, el mejor hasta el momento, y distintos modelos probados tras aplicar ciertas técnicas para mejorar las prestaciones del anterior sobre el conjunto test para el reconocimiento de objetos.

Adicionalmente, se observa en los dos gráficos de la ilustración 27 que los tres modelos que utilizan la arquitectura mediana de YOLO requieren la misma memoria gráfica, con independencia de la cantidad de imágenes con las que trabajen o los pesos de entrenamiento iniciales. Sin embargo, el que se ha entrenado con *data augmentation* sí que requiere más utilización de la GPU para procesar esa mayor cantidad de datos.

Finalmente, se concluye que YOLOv5m es el modelo que mejores prestaciones ha brindado para la tarea que se está tratando de resolver en este caso, el reconocimiento de objetos, es decir, la detección e identificación de los distintos tipos de señales de tráfico. Esta arquitectura proporciona una gran precisión, que junto a un *recall* también elevado, hace que sea la que mayor puntuación f1 tenga. Ni siquiera se ha podido mejorar su eficacia con el empleo de distintas técnicas como aumento de datos o transferencia de aprendizaje. Además, presenta una relación entre el número de épocas requeridas para alcanzar sus pesos óptimos y el tiempo necesario para entrenar cada etapa, que hace que su tiempo total de entrenamiento sea muy competitivo. Todo esto manteniendo una utilización y ocupación de memoria de la GPU que permitiría incluso realizar otros procesos simultáneamente.

A continuación, en la ilustración 28, se muestra un mosaico en el que se aprecian distintas imágenes pertenecientes al conjunto de testeo y que contienen las predicciones realizadas sobre ellas por este modelo, YOLOv5m, que se ha concluido que era el mejor para esta tarea con este conjunto de datos. Nuevamente, al igual que se veía en detección de objetos, al lado de cada *bounding box* aparecerá la probabilidad que le otorga el modelo a que dicho objeto sea realmente lo que se predice, la confianza de la arquitectura al realizar su pronóstico. La diferencia ahora es que también se encontrará encima de cada recuadro la clase asignada a cada objeto.

Ilustración 28: Predicciones de ejemplo del modelo YOLOv5m para reconocimiento de objetos sobre imágenes sacadas del subconjunto de testeo.

En esta última ilustración se intuye el punto débil de este modelo, la exhaustividad en la clase de los semáforos. Aparentemente no hay problemas al detectarlos cuando estos se encuentran de frente a la cámara. No obstante, si la perspectiva cambia y se divisa este objeto desde un lateral o de espaldas, YOLOv5m empezará a tener dificultades para hallarlos. Dicho esto, la intención en este proyecto no es detectar semáforos y otras señales de espaldas, únicamente de frente según un vehículo con cámara integrada se va acercando a ellas. Además, aun así, esta arquitectura ha demostrado tener unas excelentes prestaciones y es, sin lugar a duda, el mejor y más completo modelo de todos los estudiados.

Para terminar, se ha realizado una búsqueda de distintas imágenes de las señales de tráfico con las que se ha estado trabajando. En algunas de estas nuevas fotografías se muestran escenas en las que la señalización está deteriorada, derribada o, simplemente, en mal estado por algún motivo. A continuación, se le pasan al modelo para que este realice las predicciones oportunas sobre ellas y, comprobar así, sus resultados sobre imágenes reales.

Para llevar a cabo este proceso, fue necesaria la implementación de un nuevo *notebook* de Google Colab llamado *pruebas_finales.ipynb*, que también se puede encontrar en el anexo 1 de este documento. El *script* simplemente clona el repositorio de YOLOv5 y llama al ejecutable que este ofrece para realizar las predicciones sobre un conjunto de imágenes, especificando en dicha llamada los pesos a utilizar y la ruta de estas fotografías.

En las ilustraciones 29 y 30 se puede comprobar un ejemplo de cada una de las clases con las que se ha tratado. Aquí se aprecia que ante señales en buen estado situadas en lugares con buena visibilidad el modelo es capaz de detectarlas y reconocerlas sin error alguno, incluso ante condiciones de luminosidad cambiantes. En cambio, cuando estas señales han perdido la pintura, han sufrido algún tipo de vandalismo, están derribadas en el suelo o colgando fuera de su posición natural, el software desarrollado no podrá ni si quiera detectar que ahí se encuentra una señal de tráfico.

Ilustración 29: Predicciones realizadas sobre imágenes captadas de Internet (1/2). Fuentes: [42], [43], [44] y [45].

Ilustración 30: Predicciones realizadas sobre imágenes captadas de Internet (2/2). Fuentes: [46], [47], [48] y [49].

Con esto lo que se pretende mostrar es que realmente se ha conseguido un modelo lo suficientemente preciso y eficaz como para considerar sus predicciones como una ayuda para la temprana detección de señalización vial en mal estado y en necesidad de mantenimiento, reduciendo así el riesgo que supondría la no realización de dicho reacondicionamiento y facilitando la labor de aquellos trabajadores encargados de detectar estos casos.

4.- Conclusiones y planes de mejora

4.1.- Conclusiones

A lo largo de este proyecto se han estudiado y desarrollado algoritmos para la detección y reconocimiento de objetos centrados en las señales verticales de tráfico. Este sistema junto con un *tracker* para conseguir la posición geográfica exacta de las marcas viales, unido a una base de datos con información acerca de las señales existentes actualmente, permitiría la automatización del rastreo de aquellas marcas viales con desperfectos y necesidad de mantenimiento inmediato. De esta forma se facilitaría y modernizaría una tarea larga, tediosa y constante, pero que es de vital importancia para la seguridad de toda la sociedad hoy en día. Del mismo modo, se aumentaría la seguridad de los trabajadores encargados de detectar dichas deficiencias de forma manual.

En cuanto al cumplimiento de los objetivos principales de este experimento, que fueron planteados al inicio del mismo, a continuación, se explicará el estado en el que se encuentran actualmente, al momento de la finalización de este proyecto.

Los primeros objetivos propuestos estaban relacionados con el estudio del estado del arte tanto de los modelos actuales para detección y reconocimiento de objetos, como de los sistemas y técnicas empleados en la actualidad para el mantenimiento de señales de tráfico. Con este estudio se ha puesto de manifiesto qué es este mantenimiento, cómo se realiza y quién debe hacerlo, así como la importancia que tiene y la escasez de recursos destinados a esta tarea actualmente. También se ha realizado un repaso que cubre desde las definiciones de aprendizaje automático y visión por computador hasta qué son y cómo funcionan las CNN. Finalmente se han presentado los tipos de técnicas de detección y reconocimiento de objetos que hay, las características, ventajas y deficiencias de cada tipo, y los principales algoritmos pertenecientes a cada grupo.

El siguiente objetivo establecido se centraba en la búsqueda y el tratamiento de *datasets* relacionados con la señalización vial vertical. En este sentido, se han detectado y descrito los requisitos indispensables que debía cumplir la base de datos a utilizar, acotando así la búsqueda y filtrando los *datasets* encontrados hasta que solo quedara uno, el más idóneo. A continuación, una vez seleccionado un conjunto de datos, se realizó un estudio de la proporción de instancias pertenecientes a cada clase que tenía, se hizo un preprocesado que incluía variaciones de brillo y contraste, cambios en el tamaño de las imágenes y la creación de subconjuntos para entrenamiento, validación y testeo, entre otras acciones. Además, se adaptaron las anotaciones al formato requerido por el modelo utilizado, y se

modificaron para realizar primero la detección de objetos y luego el reconocimiento de estos.

También fue planteada la implementación y adaptación de distintos algoritmos de visión por computador y aprendizaje automático para efectuar la detección y reconocimiento de objetos de la que se ha hablado en párrafos anteriores. Así como el testeo de dichos algoritmos y el estudio de su eficacia y eficiencia. Para tal propósito se han implementado una serie de *scripts* de Python y *notebooks* de Google Colab, y se ha utilizado el algoritmo YOLO en su quinta versión y la familia de arquitecturas que este ofrece. De esta forma se ha llevado a cabo no solo el entrenamiento de distintos modelos para cada una de las dos tareas que se pretendía realizar, sino que también se han optimizado dichos entrenamientos, se han comparado estos modelos entre sí y se han intentado mejorar los resultados ofrecidos mediante la aplicación de varias técnicas de *machine learning* y *computer vision*.

Finalmente, fue propuesto un objetivo que para nosotros como estudiantes y trabajadores del mundo del dato es más una obligación ética y moral que un acto de buena voluntad, la liberación del código y documentación desarrollados para contribuir a la comunidad de *software* libre. En este sentido, el proyecto se encuentra alojado en un repositorio de GitHub ([50]). Adicionalmente, se ha elaborado una carpeta compartida en drive, disponible para todo el que la necesite en [51], con la base de datos utilizada, los pesos de los modelos entrenados y los cuadernos de Google Colab desarrollados.

4.2.- Planes de mejora

A continuación, en esta sección, se propondrán una serie de alternativas a modo de posibles mejoras futuras, algunas de las cuales ya se han mencionado brevemente en algún momento a lo largo de esta memoria.

En primer lugar, se propone la mejora del modelo conseguido en este proyecto potenciando el *recall* en la categoría *trafficlight* ya que esta exhaustividad, a pesar de no ser ni mucho menos deficiente y las magníficas prestaciones que presenta el modelo en líneas generales, es el punto en el que existe un mayor margen de mejora.

En segundo lugar, se plantea la ampliación del sistema para que sea capaz de detectar e identificar todas las señales de tráfico existentes. Puesto que no existe, o no se ha hallado, hacer esto conllevaría la previa creación de un *dataset* muy extenso y con unas características determinadas, las cuales ya se comentaron en otras secciones de este documento. Esta tarea no es nada sencilla, no obstante, la liberación del *dataset* permitiría que otros científicos puedan aprovecharse de él, y se pueda avanzar más rápidamente en una materia considerada de vital importancia. Además, habría que realizar un nuevo entrenamiento de los modelos y, posiblemente, utilizar arquitecturas más complejas, como la versión xl de YOLO, debido a la complejidad del conjunto de datos descrito. También habría que tener en cuenta los grandes requerimientos técnicos que serían demandados por dicho sistema. Se necesitaría un servidor con una GPU de gran potencia y capacidad de almacenaje.

Luego, otra opción interesante sería añadir la capacidad de realizar predicciones en tiempo real. Para ello habría que incorporar una pequeña cámara de vídeo junto con el sistema desarrollado y un *tracker* a distintos vehículos. Estos vehículos irían circulando por la carretera, y el *software* instalado iría analizando en tiempo real las imágenes captadas por la videocámara, enviando datos e información sobre el estado de la señalización vial a un servidor, al mismo tiempo que dicho *tracker* permitiría geolocalizar las distintas señales. De esta forma sí que se conseguiría realmente ayudar en las tareas de mantenimiento.

Finalmente, siempre se debe estar atento a los nuevos avances que se puedan realizar en el estado del arte y en el desarrollo de cualquier algoritmo que pueda mejorar en rendimiento o eficiencia al utilizado en el proyecto.

5.- Bibliografía

[1] David Silver, “Perception Projects from the Self-Driving Car Nanodegree Program”, *medium*.com, 26 July 2018. Available: https://medium.com/udacity/perception-projects-from-the-self-driving-car-nanodegree-program-51fb88a38ff9

[2] Intelligent video surveillance, Multitel Innovation Centre [Online]. Available: https://www.multitel.eu/expertise/artificial-intelligence/intelligent-video-surveillance/

[3] AI & Machine Learning (2020, Jan 13). AI Inspection: Machine Learning / Computer Vision for Visual Defect Detection [Online]. Available: https://www.youtube.com/watch?v=UY6xbrcViVw

[4] Canales sectoriales Interempresas (2021, May 07). La Asociación Española de la Carretera aplaude la iniciativa del Gobierno y lanza como opción a valorar el ‘Bono de Movilidad’” [Online]. Available: https://www.interempresas.net/ObrasPublicas/Articulos/350447-Asociacion-Espanola-Carretera-aplaude-iniciativa-Gobierno-lanza-opcion-valorar-Bono.html

[5] Manuel Moreno, “TO-21, kilómetro 6: la desoladora imagen de biondas y señales de tráfico rotas a la espera de una solución”, *ABC*, 07 October 2019. Available: https://www.abc.es/espana/castilla-la-mancha/toledo/ciudad/abci-to-21-kilometro-6-desoladora-imagen-biondas-y-senales-trafico-rotas-espera-solucion-201910071236_noticia.html

[6] Javier Costas, “La asignatura pendiente de las señales de tráfico: catalogación, inventario y revisión”, *motor.es*, 30 December 2021. Available: https://www.motor.es/noticias/la-asignatura-pendiente-de-las-senales-de-trafico-catalogacion-inventario-y-revision-202183859.html

[7] Ponle Freno (2020, Sep 24). Señal tapada por la vegetación en N-340 km 409 [Online]. Available: https://compromiso.atresmedia.com/ponlefreno/campanas/carreteras-senales/denuncias/senal-tapada-vegetacion-n340-409_202009245f6c660dd0cd870001f5eb16.html

[8] Zihao Wu, “Computer Vision-Based Traffic Sign Detection and Extraction: A Hybrid Approach Using GIS And Machine Learning” (2019). *Electronic Theses and Dissertations*. 2039. Available: https://digitalcommons.georgiasouthern.edu/etd/2039/

[9] Tuteorica.com (2022, Jun 09). Mantenimiento y conservación de las señales de tráfico [Online]. Available: https://www.youtube.com/watch?v=DEtp8xaq2g4

[10] Fundación Mapfre. ¿Quién debe cambiar las señales de tráfico defectuosas o mal colocadas? [Online]. Available: https://www.fundacionmapfre.org/educacion-

divulgacion/prevencion/prevencion-accidentes-mayores/quien-debe-cambiar-senales-defectuosas/

[11] Rodrigo Pareja, "Más del 70% de las señales de tráfico tiene el reflectante caducado", *Car and Driver*, 24 August 2020. Available: https://www.caranddriver.com/es/coches/planeta-motor/a33747184/mantenimiento-carreteras-senales-trafico/

[12] AEC, "Necesidades de inversión en conservación", Asociación Española de la Carretera, Madrid, España, 2014. Available: https://www.aecarretera.com/np/INFORME-NECESIDADES-DE-INVERSION-EN-CONSERVACION-ABRIL-2014.pdf

[13] Ana Sofia Figueroa Infante, Carlos Fabián Flórez Valero, María Patricia León Neira, Édgar Eduardo Muñoz Díaz, Bernardo Jacobo Ojeda Moncayo, Fredy Alberto Reyes Lizcano, Jorge Alberto Rodríguez Ordóñez, "Manual para el mantenimiento de la red vial secundaria (pavimentada y en afirmado)" *academia.edu*. 6388436. Available: https://www.academia.edu/6388436

[14] Larxel (2020). Road Sign Detection [Online]. Available: https://www.kaggle.com/datasets/andrewmvd/road-sign-detection

[15] Make ML (2020, Apr 13). Road Signs Dataset [Online]. Available: https://makeml.app/datasets/road-signs

[16] Kaggle. (Larxel profile page) [Online]. Available: https://www.kaggle.com/andrewmvd

[17] Roboflow. (Roboflow official website) [Online]. Available: https://roboflow.com/

[18] Roboflow. (Roboflow official documentation) [Online]. Available: https://docs.roboflow.com/

[19] Roboflow. (List of services from roboflow official website) [Online]. Available: https://roboflow.com/pricing

[20] Roboflow. (Roboflow universe from roboflow official website) [Online]. Available: https://universe.roboflow.com/

[21] Roboflow. (LinkedIn profile page) [Online]. Available: https://www.linkedin.com/company/roboflow-ai/

[22] Samuel Theophilus, "Roboflow: Converting Annotations for Object Detection", *medium.com*, 03 September 2021. Available: https://medium.com/analytics-vidhya/converting-annotations-for-object-detection-using-roboflow-5d0760bd5871

[23] Jayita Bhattacharyya, “Step By Step Guide To Object Detection Using Roboflow”, *developers corner*, 11 October 2021. Available: https://analyticsindiamag.com/step-by-step-guide-to-object-detection-using-roboflow/#:~:text=Roboflow%20is%20a%20Computer%20Vision,Roboflow%20accepts%20various%20annotation%20formats.

[24] Glenn Jocher, “Train Custom Data Tutorial”, *GitHub*, 03 June 2020. Available: https://github.com/ultralytics/yolov5/issues/12

[25] Srishilesh P. S., “Understanding PASCAL VOC Dataset”, *section.io*, 08 February 2022. Available: https://www.section.io/engineering-education/understanding-pascal-voc-dataset/#pascal-voc

[26] Antonio José Aroca Aguilar, “Técnicas de detección de objetos y segmentación usando retinas artificiales”, (trabajo fin de grado), Universidad de Granada, UGR, España, 2021. Available: https://github.com/nono-aroca/Deteccion-de-objetos-utilizando-retinas-artificiales

[27] M. Q. Huang, J. Ninic, Q. B. Zhang, “BIM, machine learning and computer vision techniques in underground construction: Current status and future perspectives”, *Tunnelling and Underground Space Technology*, Vol. 108, 2021. Available: https://www.sciencedirect.com/science/article/abs/pii/S0886779820306313

[28] Codebasics (2020, Dec 25). What is YOLO algorithm? | Deep Learning Tutorial 31 (Tensorflow, Keras & Python) [Online]. Available: https://www.youtube.com/watch?v=ag3DLKsl2vk

[29] Hmrishav Bandyopadhyay, “YOLO: Real-Time Object Detection Explained”, *v7labs.com*, 2022. Available: https://www.v7labs.com/blog/yolo-object-detection

[30] Grace Karimi, “Introduction to YOLO Algorithm for Object Detection”, *section.io*, 15 April 2021. Available: https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/

[31] Ultralytics, “YOLOv5”, (repositorio de YOLOv5 gestionado por Ultralytics), *GitHub*. Available: https://github.com/ultralytics/yolov5

[32] Ultralytics, “Train Custom Data”, (repositorio de YOLOv5 gestionado por Ultralytics), *GitHub*. Available: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

[33] Ultralytics. (YOLOv5 official documentation) [Online]. Available: https://docs.ultralytics.com/

[34] "Detection and Deep Segmentation", class notes for Computer Vision, Department of Computer Science and Artificial Intelligence, Universidad de Granada, Autumn 2021

[35] IBM. What is Computer Vision? [Online]. Available: https://www.ibm.com/topics/computer-vision

[36] Grupo de Inteligencia Computacional U.P.V. E.H.U., "Visión por Computador", presented at Semana de la Ciencia, Donosti, 2009. Available: https://www.ehu.eus/ccwintco/uploads/d/d4/PresentacionMundoVirtual.pdf

[37] Txema Rodríguez, "Machine Learning y Deep Learning: cómo entender las claves del presente y futuro de la inteligencia artificial", *xataka*, 16 October 2020. Available: https://www.xataka.com/robotica-e-ia/machine-learning-y-deep-learning-como-entender-las-claves-del-presente-y-futuro-de-la-inteligencia-artificial

[38] Russ Tedrake, "Robotic Manipulation, Perception Planning, and Control", (Course notes for MIT 6.420), chapter 6, 2022. Available: https://manipulation.mit.edu/

[39] Kiprono Elijah Koech, "Object Detection Metrics With Worked Example", *Towards Data Science*, 26 August 2020. Available: https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e#:~:text=Average%20Precision%20(AP)%20and%20mean,COCO%20and%20PASCAL%20VOC%20challenges.

[40] Ahmed Fawzy Gad, "Evaluating Object Detection Models Using Mean Average Precision (mAP)", *PaperspaceBlog*, 2020. Available: https://blog.paperspace.com/mean-average-precision/

[41] Deval Shah, "Mean Average Precision (mAP) Explained: Everything You Need to Know", *v7labs.com*. Available: https://www.v7labs.com/blog/mean-average-precision#h3

[42] Rebeca Álvarez, "La señal de Stop es de 1915 y otros datos de seguridad interesantes", *BBC Top Gear*, 26 June 2019. Available: https://www.topgear.es/listas/no-solo-ruedas/senal-stop-1915-otros-datos-seguridad-interesantes-445283

[43] Ponle Freno (2020, Oct 29). Señal de Stop en mal estado [Online]. Available: https://compromiso.atresmedia.com/ponlefreno/campanas/carreteras-senales/denuncias/senal-stop-mal-estado_202010295f9abff80e37c40001426bdd.html

[44] Víctor Suárez Obrero, “Estudio de los principales factores responsables de los accidentes de tráfico y sus efectos en España”, (trabajo fin de grado), Universitat Politècnica de Catalunya, UPC, España, 2021. Available: https://upcommons.upc.edu/bitstream/handle/2117/343832/ESTUDIO%20DE%20LOS%20PRINCIPALES%20FACTORES%20RESPONSABLES%20DE%20LOS%20ACCIDENTES%20DE%20TR%C3%81FICO%20Y%20SUS%20EFECTOS%20EN%20ESPA%C3%91A.pdf?sequence=1&isAllowed=y

[45] Virginia Duque Mirón, “¿Cuáles son las velocidades máximas según tipos de carretera?”, *BuscoUnCoche.es*, 05 March 2022. Available: https://buscouncoche.es/actualidad/dgt/cuales-son-las-velocidades-maximas-segun-tipos-de-carreteras/

[46] Shutterstock. Imágenes libres de regalías de Detente. ID 2185902575. [Online]. Available: https://www.shutterstock.com/es/image-photo/road-sign-has-fallen-broken-no-2185902575

[47] Facebook. (publicación del usuario Junta Municipal de los Garres y Lages). Available: https://www.facebook.com/LosGarresYLages

[48] RadioDos (2018, Oct 26). El semáforo de Ruta 12 en el ingreso por Centenario: a punto de caer [Online]. Available: https://www.radiodos.com.ar/2798-el-semaforo-de-ruta-12-en-el-ingreso-por-centenario-a-punto-de-caer

[49] HGH. (Herminio González e Hijos official website) [Online]. Available: https://www.herminiogonzalez.es/senalizacion-de-trafico/

[50] Antonio José Aroca Aguilar, “Detección de objetos en imágenes mediante técnicas de Deep Learning: Señales de tráfico”, (trabajo fin de máster), Universidad Europea Miguel de Cervantes, UEMC, España, 2022, (repositorio GitHub). Available: https://github.com/nono-aroca/Deteccion-de-objetos-en-imagenes-mediante-tecnicas-de-deep-learning

[51] Antonio José Aroca Aguilar, “Detección de objetos en imágenes mediante técnicas de Deep Learning: Señales de tráfico”, (trabajo fin de máster), Universidad Europea Miguel de Cervantes, UEMC, España, 2022, (carpeta compartida Google Drive). Available: https://drive.google.com/drive/folders/1W6VJbn8gqByqhX8y_xejGebZ7jkYuv6n?usp=sharing

Anexo 1: Código fuente

parser.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr  9 16:43:36 2022

@author: Antonio José Aroca Aguilar
"""

################################################################################
################################### IMPORTS ####################################
################################################################################

import os
from xml.dom import minidom


################################################################################
############################## CONSTANTES GLOBALES #############################
################################################################################

#path de los directorios
path = "C:/Users/Equipo/Desktop/TFM/dataset/annotations"
new_path = "C:/Users/Equipo/Desktop/TFM/dataset/parsed_annotations"

#diccionario de clases
diccionario_clases = {'trafficlight': 0, 'stop': 1, 'speedlimit': 2,
                      'crosswalk': 3}


################################################################################
########################### DEFINICION DE FUNCIONES ############################
################################################################################

def convertirCoordenadas(x_min: int, y_min: int, x_max: int, y_max: int,
                         width: int, height: int) -> tuple:
    """Permite convertir las coordenadas de un bounding box del formato xml a yolo.

    La función permite convertir los datos de un bounding box extraido a través
    del formato xml, es decir, sus cuatro esquinas, al formato requerido por
    yolo. Esto es, las coordenadas del centro del bounding box y su ancho y alto.
    Todo esto teniendo en cuenta que yolo trabaja con coordenadas delimitadas
    por el intervalo [0,1].
    """

    #Pasamos los datos a float para poder operar correctamente
    x_min = float(x_min)
```

```
    y_min = float(y_min)
    x_max = float(x_max)
    y_max = float(y_max)
    width = float(width)
    height = float(height)

    #Convertimos el bounding box al formato requerido (punto medio y tamaño)
    x = (x_min + x_max) / 2.0
    y = (y_min + y_max) / 2.0
    w = x_max - x_min
    h = y_max - y_min

    #Las hacemos relativas al tamaño de la imagen
    dw = 1.0/width
    dh = 1.0/height

    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh

    return (x, y, w, h)


def parseFile(filepath: str, path_destino: str) -> None:
    """Permite transformar anotaciones de detección de objetos de xml a formato yolo.

    La función permite convertir las anotaciones de detección de objetos de una
    imagen del formato xml a un txt en formato yolo. Para ello recibe la ruta
    del archivo xml con las anotaciones y la ruta del archivo de salida.
    """

    #Extraemos los objetos anotados y tamaño de la imagen
    informacion = minidom.parse(filepath)

    objetos = informacion.getElementsByTagName('object')

    size = informacion.getElementsByTagName('size')[0]
    width = int((size.getElementsByTagName('width')[0]).firstChild.data)
    height = int((size.getElementsByTagName('height')[0]).firstChild.data)

    #Vamos procesando cada objeto por separado y escribiendo los datos en el archivo
    with open(path_destino, "w") as f:

        for item in objetos:

            #Obtenemos la clase y las coordenadas del bounding box
            clase =  (item.getElementsByTagName('name')[0]).firstChild.data
```

```
            x_min =
((item.getElementsByTagName('bndbox')[0]).getElementsByTagName('xmin')[0]).firstChild.data
            y_min =
((item.getElementsByTagName('bndbox')[0]).getElementsByTagName('ymin')[0]).firstChild.data
            x_max =
((item.getElementsByTagName('bndbox')[0]).getElementsByTagName('xmax')[0]).firstChild.data
            y_max =
((item.getElementsByTagName('bndbox')[0]).getElementsByTagName('ymax')[0]).firstChild.data

            #Convertimos las coordenadas y la clase al formato yolo
            clase_yolo = diccionario_clases[clase]
            bounding_box = convertirCoordenadas(x_min, y_min, x_max, y_max, width, height)

            #Escribimos el nuevo archivo con los datos parseados
            f.write(str(clase_yolo) + " " + " ".join([("%.6f" % coord) for coord in
bounding_box]) + '\n')



################################################################################
################################### MAIN #######################################
################################################################################

#Obtenemos todos los archivos del directorio
files = os.listdir(path)

#Parseamos cada archivo
for i, document in enumerate(files, start=1):
    filepath = path + "/" + document
    new_filepath = new_path + "/" + document.split('.')[0] + ".txt"
    parseFile(filepath, new_filepath)
    print("Archivos parseados: {}/{} \t {}%".format(i, len(files),
                                                   round(i/len(files)*100,2)))
```

mostrar_anotaciones.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 11 20:15:46 2022

@author: Antonio José Aroca Aguilar
"""

import matplotlib.pyplot as plt
import cv2
import os

pathImg = 'images'

files = os.listdir(pathImg)

for i, document in enumerate(files, start=1):
    path_imagen = pathImg + "/" + document
    path_anotacion = 'parsed_annotations' + "/" + document.split('.')[0] + ".txt"

    img = cv2.imread(path_imagen)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    dh, dw, _ = img.shape

    fl = open(path_anotacion, 'r')
    data = fl.readlines()
    fl.close()

    for dt in data:

        _, x, y, w, h = dt.split(' ')

        nx = int((float(x) - float(w)/2)*dw)
        ny = int((float(y) - float(h)/2)*dh)
        nw = int(float(w)*dw)
        nh = int(float(h)*dh)

        cv2.rectangle(img, (nx,ny), (nx+nw,ny+nh), (0,240,251), 2)

    plt.imshow(img)
    plt.title(path_imagen)
    plt.show()
    print("Imagenes mostradas: {}/{} \t {}%".format(i, len(files),
                                                    round(i/len(files)*100,2)))
```

unificar_clases.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 14 11:28:31 2022

@author: Antonio José Aroca Aguilar
"""

import os

path = './test/labels' # introducir ruta
files = os.listdir(path)

for file in files:

    file = path + '/' + file
    contenido= list()

    with open(file, 'r') as archivo:
       for linea in archivo:
           print(linea)

    with open(file, 'r') as archivo:
       for linea in archivo:
           columnas = linea.split(' ')
           columnas[0] = '0'
           contenido.append(' '.join(columnas))

    with open(file, 'w') as archivo:
        archivo.writelines(contenido)

    with open(file, 'r') as archivo:
       for linea in archivo:
           print(linea)
    print('-----------------------')
```

YOLOv5.ipynb

Setup¶

In [ ]:

```
#imports para montar el sistema de archivos de drive
from google.colab import drive
drive.mount("/content/drive")
```

Clonamos el repositorio e instalamos las dependencias

In [ ]:

```
!git clone https://github.com/ultralytics/yolov5  # clonamos repositorio
%cd yolov5
%pip install -qr requirements.txt  # instalamos las dependencias

import torch
import utils
display = utils.notebook_init()  # comprobamos que todo este correcto
```

Train¶

Iniciamos tensorboard para el registro local del proceso de entrenamiento

In [ ]:

```
%load_ext tensorboard
%tensorboard --logdir runs/train
```

Instalamos e iniciamos weight and biases para el registro en la nube del entrenamiento, así como la visualización de dicho proceso.

In [ ]:

```
%pip install -q wandb
import wandb
wandb.login()
```

Realizamos el entrenamiento.
Es importante especificar según corresponda cada uno de los distintos parámetros de la red.

In [ ]:

```
!python train.py --img 416 --batch 32 --epochs 1000 --data /content/drive/MyDrive/T
FM/solo_señales_labels_convertidas/data.yaml --weights '' --cfg yolov5l.yaml --cach
e
```

Test¶

Comprobamos prestaciones mediante el conjunto test. Especificar en cada caso la ruta donde se encuentran los pesos del modelo entrenado, así como los distintos parámetros.

In [ ]:

```
!python val.py --weights /content/yolov5/runs/train/exp4/weights/best.pt --data /co
ntent/drive/MyDrive/TFM/solo_señales_labels_convertidas/data.yaml --img 416 --iou 0
.65 --half --task test
```

A continuación, descargamos los pesos de la red y algunas gráficas de salida que se nos proporciona como feedback del entrenamiento y el proceso de testeo.

In [ ]:

```
import os
from google.colab import files
```

```
files.download('/content/yolov5/runs/train/exp4/weights/best.pt')
```

In [ ]:

```
files.download('/content/yolov5/runs/train/exp4/weights/last.pt')
```

In [ ]:

```
for i in os.listdir('/content/yolov5/runs/train/exp4') :
    if i != 'weights':
        files.download('/content/yolov5/runs/train/exp4/'+i)
```

In [ ]:

```
for i in os.listdir('/content/yolov5/runs/val/exp4') :
    if i != 'weights':
        files.download('/content/yolov5/runs/val/exp4/'+i)
```

pruebas_finales.ipynb

Setup¶

In [ ]:

```
#imports para montar el sistema de archivos de drive
from google.colab import drive
drive.mount("/content/drive")
```

Clonamos el repositorio e instalamos las dependencias

In [ ]:

```
!git clone https://github.com/ultralytics/yolov5  # clonamos repositorio
%cd yolov5
%pip install -qr requirements.txt  # instalamos las dependencias

import torch
import utils
display = utils.notebook_init()  # comprobamos que todo este correcto
```

Detection¶

In [ ]:

```
!python detect.py --weights /content/drive/MyDrive/TFM/cuatro_clases/9/weights/best
.pt --img 416 --conf 0.25 --source /content/drive/MyDrive/TFM/cuatro_clases/9/prueb
as
```