UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

Desarrollo de aplicación basada en CNN para algoritmos de visión en coches autónomos

Developing CNN-based applications for vision algorithms in autonomous cars

Sergio Gil Gavela

Doble grado en Matemáticas e Ingeniería Informática

Curso académico 2020-21

Director: Carlos García Sánchez

RESUMEN

Hablar de conducción autónoma es sinónimo de hablar de reconocimiento de imágenes y detección de objetos. En el paradigma actual las redes neuronales convolucionales han emergido como la alternativa más efectiva para emular la visión humana y detectar los objetos de la calzada.

El propósito de este trabajo es evaluar el funcionamiento de algunas de las redes neuronales más actuales empleadas en detección de objetos si fuesen integradas en el dispositivo de reconocimiento de objetos de un coche autónomo. En concreto, el trabajo se centrará en evaluar los siguientes tres puntos relevantes en los dispositivos de detección de objetos:

- **Precisión**
- **Velocidad de inferencia**
- **Coste energético**

Para ello se llevará a cabo un reentrenamiento de una red neuronal generalista en Darknet para convertirla en una red orientada a conducción autónoma. La posterior evaluación se divide en dos partes

- **Precisión:** Se evaluará atendiendo a la métrica mAP sobre tres datastes distintos: MS COCO, BDD100k y un pequeño dataset de elaboración propia.
- **Velocidad de inferencia y coste energético:** Serán evaluadas utilizando el kit de herramientas OpenVINO de Intel.

PALABRAS CLAVE

Detección de objetos, conducción autónoma, red neuronal convolucional, MS COCO, BDD100k, YOLOv4, YOLOv3, mAP, Darknet, OpenVINO.

ABSTRACT

Talking about autonomous driving is synonymous with talking about image recognition and object detection. Nowadays, Convolutional Neural Networks have emerged as the most efficient alternative to emulate human vision and to detect objects around the streets.

This project's purpose is to evaluate some of the most popular existing neural networks if they were integrated in an autonomous car. In particular the project will focus in the evaluation of this three relevant measures for a neural network:

- **Precision**
- **Inference speed**
- **Energy cost**

To do so, we will retrain a generalist neural network in Darknet in order to turn it into an autonomous-driving-oriented neural network. Afterwards, the evaluation will be divided into two separate parts:

- **Precision:** It will be evaluated attending to mAP over three different data sets: MS COCO, BDD100k and a small self-made data set.
- **Inference speed and energy cost:** They will be evaluated with Intel's OpenVINO toolkit.

KEY WORDS

Object detection, autonomous driving, Convolutional Neural Network, MS COCO, BDD100k, YOLOv4, YOLOv3, mAP, Darknet, OpenVINO.

Índice

1. Introducción

1.1. Contexto y antecedentes

Hablar de conducción autónoma es sinónimo de hablar de reconocimiento de imágenes y detección de objetos. La motivación principal detrás de la conducción autónoma es que un vehículo pueda circular por la vía sin necesidad de una persona al volante. Con este fin, es necesario una cámara integrada en el vehículo junto con un procesador que reconozca los elementos de la calzada tales como peatones, señales, otros vehículos, etc y tome decisiones sobre cómo debe comportarse en función de lo que tiene delante.

La idea de la conducción autónoma data de 1939 cuando Norman Bel Geddes desarrolló un vehículo eléctrico que era controlado por un circuito eléctrico embebido en el pavimento de la carretera. No obstante, no fue hasta 1980 cuando Mercedes-Benz introdujo por primera vez la visión por computador en un vehículo que consiguió recorrer a más de 100km/h calles sin tráfico. [1]

A medida que se tiene un entendimiento más completo de las imágenes, no sólo es deseable clasificarlas, además se busca una identificación precisa del tipo y localización de los objetos contenidos en la imagen. Los principales avances en detección de objetos se han llevado a cabo gracias a la mejora en los modelos de machine learning. Hasta hace no mucho el mejor desempeño había sido obtenido utilizando modelos de aprendizaje superficial, es decir, utilizando una o dos capas de representación para los datos. Sin embargo, en los últimos años las Redes Neuronales Profundas, en concreto las Redes Neuronales Convolucionales (CNN) han emergido como el modelo más eficiente para la detección de objetos. [2]

1.2. Motivación y objetivos

El propósito de este trabajo es evaluar el funcionamiento de algunas de las redes neuronales más actuales empleadas en detección de objetos si fuesen integradas en el dispositivo de reconocimiento de objetos de un coche autónomo. En concreto, el trabajo se centrará en evaluar los siguientes tres puntos relevantes en los dispositivos de detección de objetos:

- **Precisión**
- **Velocidad de inferencia**
- **Coste energético**

Para hacer una evaluación exhaustiva y obtener resultados consistentes se han fijado los siguientes objetivos:

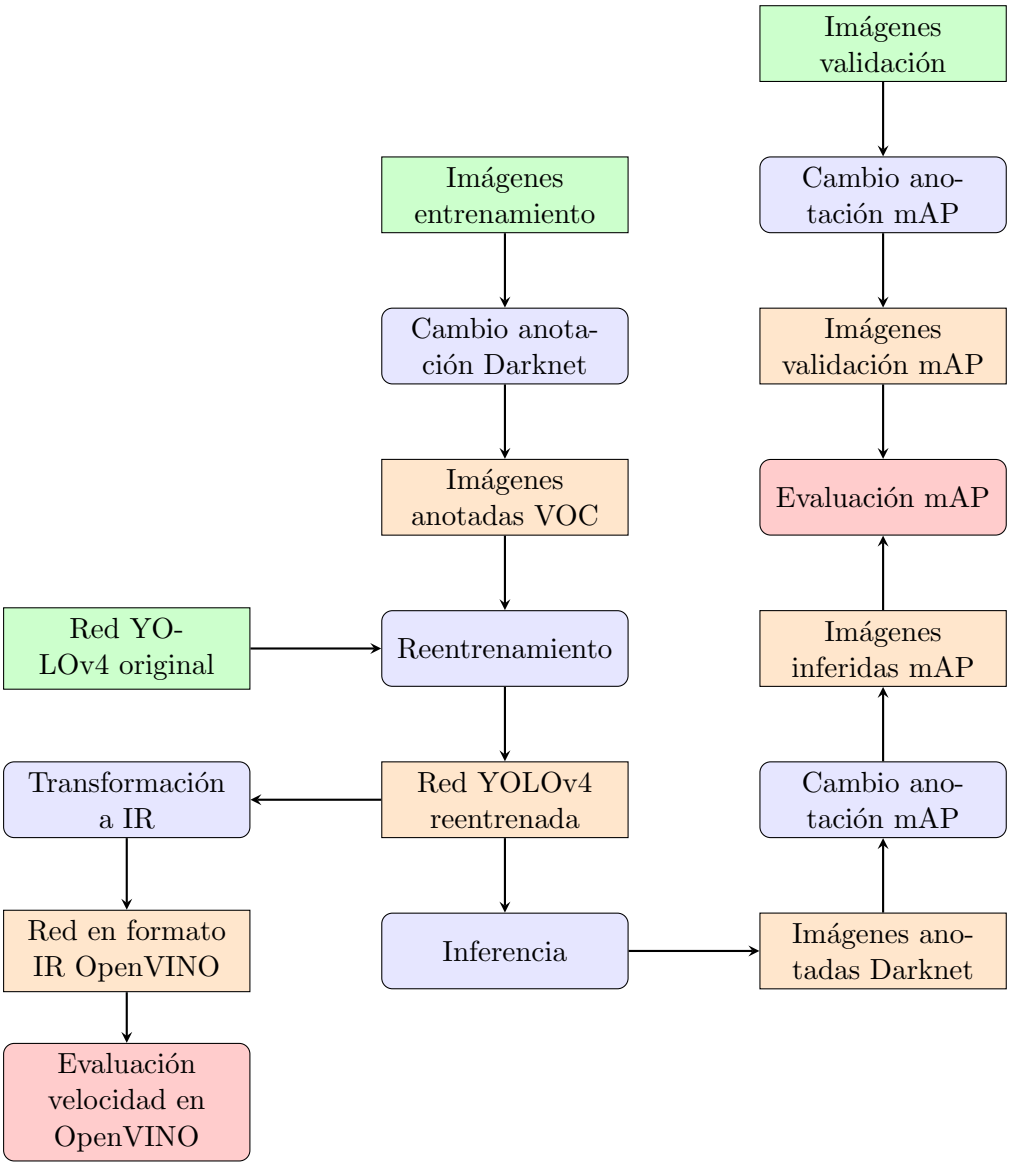
- **Evaluar la precisión de las redes YOLOv3 y YOLOv4** sobre varios datasets. Estas dos redes forman parte del paradigma actual de detección de objetos.
- **Reentrenar YOLOv4** con un dataset orientado a conducción autónoma y evaluar la precisión de esta nueva red. YOLOv4 es una red generalista por lo que en un principio está entrenada únicamente para la detección de objetos cotidianos como sofás, televisores o personas. El objetivo es convertirla en una red orientada a conducción autónoma.
- **Determinar**, atendiendo a las métricas que se definirán más adelante **qué red es más precisa.** La métrica principal que se va a usar es mAP.

- **Para la red más precisa evaluar su velocidad de inferencia y su coste energético** sobre distintos hardwares.
- **Concluir qué red y qué hardware de los evaluados son los más apropiados** para integrar en un coche autónomo.

Para llevar a cabo los objetivos anteriormente expuestos se han empleado principalmente tres frameworks

- **Darknet:** Empleado para el reentrenamiento e inferencia orientada a evaluar la precisión.
- **OpenVINO:** Inferencia orientada a evaluar la velocidad.
- **Repositorio mAP:** Scripts empleados para evaluar la precisión.

A pesar de que año a año surgen nuevas redes orientadas a visión artificial, este campo está todavía en desarrollo. Por este motivo, las tecnologías y los frameworks empleados en detección de objetos son a menudo inconexos y requieren distintas versiones de las librerías o emplean formatos distintos para anotar los objetos que hay en las imágenes.



En el diagrama de la anterior página se pueden observar los procesos que han sido necesarios para integrar los frameworks entre sí y llevar a cabo la evaluación de la precisión y la velocidad de inferencia.

Recorrer el entramado del diagrama e integrar las funcionalidades de los frameworks no es en absoluto trivial. Para hacer un proceso completo desde un dataset en "crudo" hasta una evaluación exhaustiva de una red reentrenada han sido necesarios numerosos scripts, varios de elaboración propia, y un control minucioso de las versiones de las librerías necesarias para cada framework. Estos scripts sirven para convertir entre fomatos de red y formatos de anotaciones con el objetivo de adaptarse a los requerimientos de cada framework (ver cuadro 1). Más allá de la evaluación de la red, esta integración entre frameworks ha sido la base sobre la que se sostiene este trabajo y la principal aportación personal.

Formatos de red	
Darknet	Archivo .weights
OpenVINO	IR (archivo .bin + archivo .xml)

Formatos de anotaciones	
MS COCO	COCO format
BDD100k	Scalabel format
Darknet retrain	VOC format
Darknet inference	Darknet format
OpenVINO	COCO format
Repositorio mAP	mAP format

Cuadro 1: Formatos de red y anotaciones en los distintos frameworks empleados.

Para concluir la introducción, en la figura 2 se pueden observar algunos resultados similares a los que se espera obtener. En la gráfica se compara la precisión y la velocidad de algunos de las redes de detección de objetos más punteras como YOLOv4 o YOLOv3 evaluadas sobre diferentes hardwares.

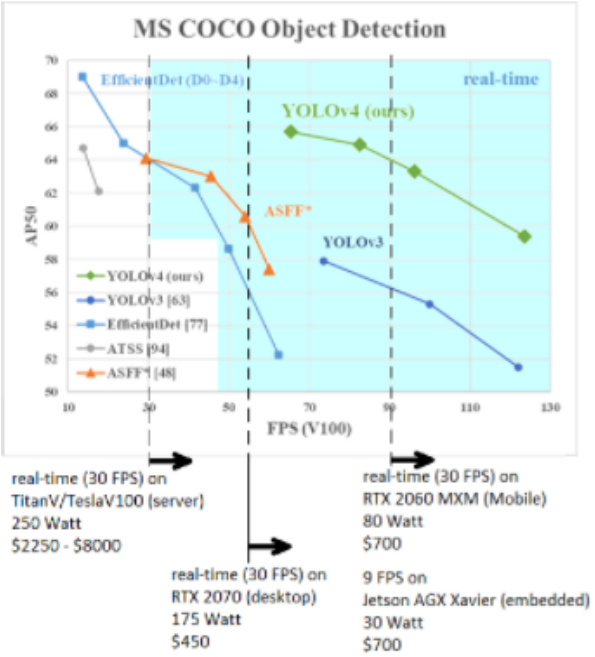


Figura 1: Algunas de las redes más punteras evaluadas sobre el dataset MS COCO. [19]

2. Introduction

2.1. Context and background

Speaking about autonomous driving is almost as speaking about image recognition and object detection. The main motivation underlying autonomous driving is that a vehicle could circulate without a person behind the steering-wheel. To achieve this, an autonomous vehicle needs a camera and a processor to detect the relevant objects in the street such as pedestrians, traffic signs or other kind of vehicles in order to make decisions on how to behave.

The first autonomous driving ideas date from 1939 when Normal Bel Geddes developed an electrical vehicle which was controlled by an electric circuit. Nevertheless, it was in 1980 when Mercedes-Benz introduced for the first time computer vision in the CPU of a vehicle which was able to circulate at 100km/h around empty streets. [1]

Nowadays there is a deeper understanding of images and it is not only desirable to classify them but also to identify precisely the kind of objects in the images and their locations. The main breakthroughs in object detection have been made when machine learning algorithms were improved. Until a few years ago superficial learning algorithms, which use one or two representation layers, were the ones with best performance. However, during the past five years Deep Neural Networks have proved to be the most efficient object detection model. [2]

2.2. Motivation and goals

This project's purpose is to evaluate some of the most popular existing neural networks if they were integrated in a self-driving car. In particular the project will focus in the evaluation of this three relevant measures for a neural network:

- **Precision**
- **Inference speed**
- **Energy cost**

In order to do an exhaustive evaluation and obtain consistent results the following goals have been set:

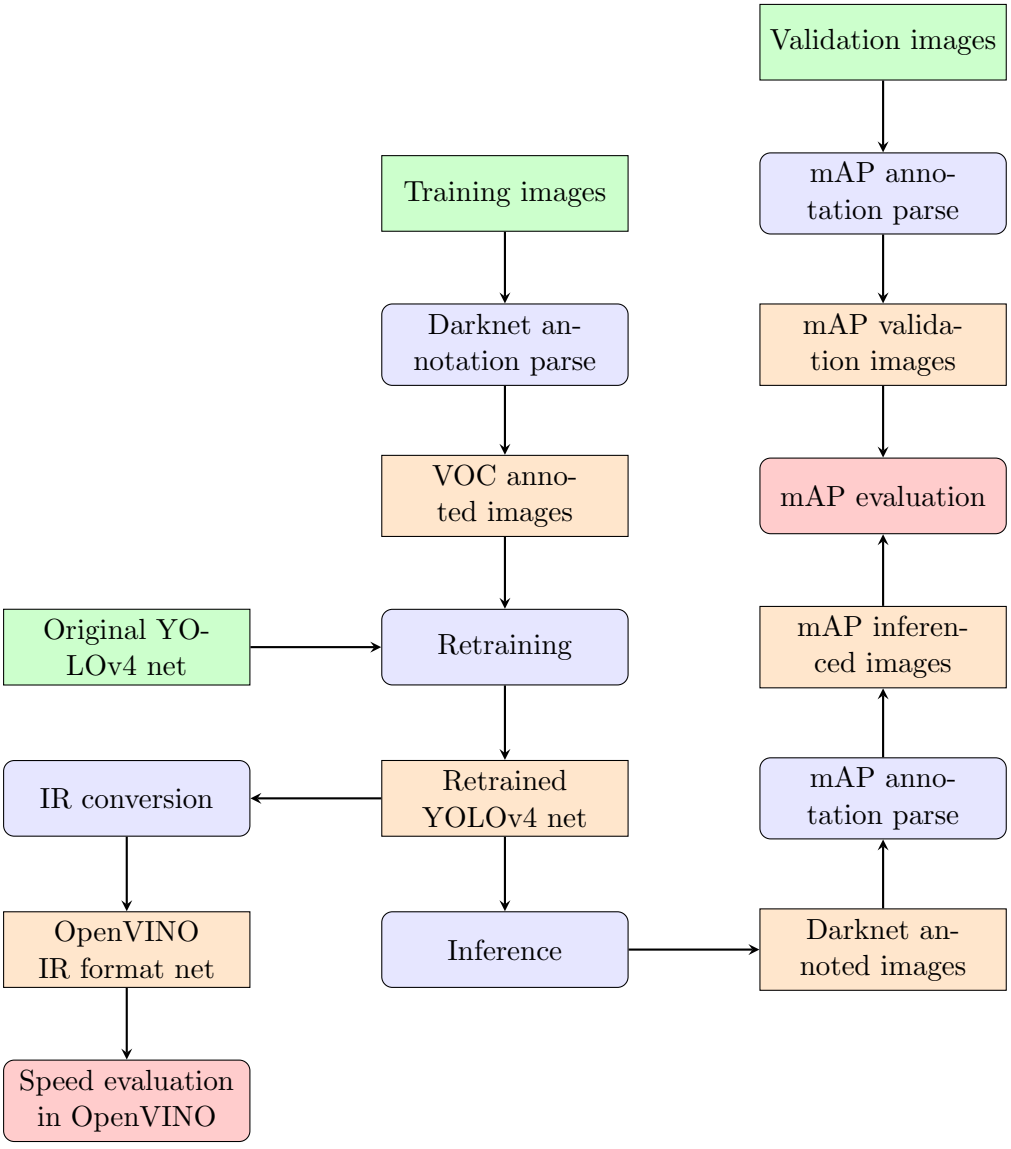
- Evaluate YOLOv3 and YOLOv4 precision over several data sets.
- Retrain YOLOv4 using an autonomous-driving-oriented data set and evaluate the the retrained network precision.
- Using appropriate metrics, determine which of the neural networks is the most precise.
- Evaluate the most precise network inference speed and energy cost in three different hardwares.
- Conclude which network and which hardware are the best ones to be integrated into an autonomous vehicle.

To achieve the fixed objectives mainly three different frameworks have been used:

- **Darknet:** Used for retraining and inference in order to evaluate precision.
- **OpenVINO:** Used to perform inference in order to evaluate speed.
- **Repositorio mAP:** Several scripts used to evaluate precision.

Despite each year new artificial vision oriented networks appear, this is a developing field. For this reason, the technologies and frameworks which are used in object detection are often disconnected, have very different library requirements and use distinct annotation formats for the images.

In the following diagram it can be seen what processes have been necessary in order to evaluate precision and inference speed.



Going through the diagram above and integrating the different frameworks functionalities is not trivial at all. The full process from a a raw data set to an exhaustive speed and precision evaluation requires several scripts, some of which are self-made, and a meticulous libraries version control. These scripts purpose is to be able to change between annotation and network formats (see frame 2). Beyond the network evaluation, this integration among frameworks is the foundation of this project and the main personal contribution.

Network formats	
Darknet	.weights file
OpenVINO	IR (.bin file + .xml file)

Annotation formats	
MS COCO	COCO format
BDD100k	Scalabel format
Darknet retrain	VOC format
Darknet inference	Darknet format
OpenVINO	COCO format
mAP repository	mAP format

Cuadro 2: Network and annotation formats required by the frameworks used in this project.

To conclude the introduction, in figure 2 some results as the ones expected are shown. The graphic compares precision and speed of some leading edge object detection CNN such as YOLOv4 or ATSS over different hard-wares.

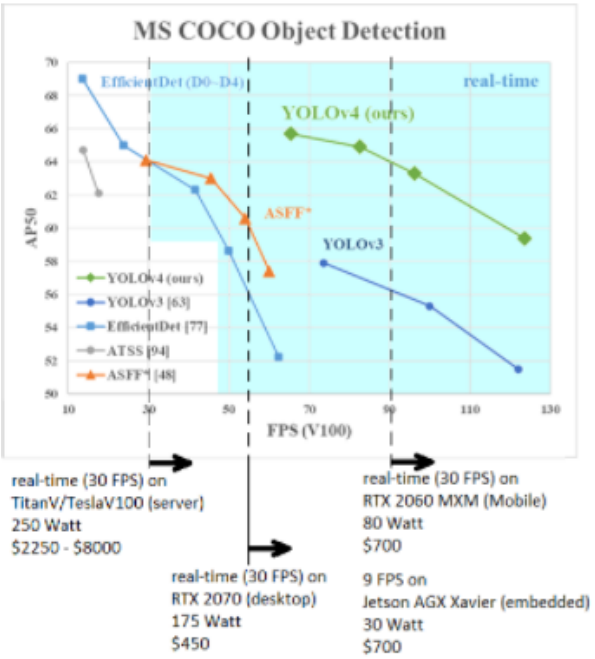


Figura 2: Some of the most leading edge networks evaluated over MS COCO data set. [19]

3. Fundamentos teóricos

El propósito de esta sección es ofrecer una visión general del funcionamiento de las herramientas que posibilitan el reconocimiento de imágenes, es decir, las redes neuronales y los algoritmos de detección de objetos.

3.1. Qué es una red neuronal

Para ahondar en cómo funciona una CNN es preciso realizar primero una descripción más general de qué es una red neuronal.

Una red neuronal recibe un vector de valores de entrada. Cada uno de los elementos del vector llega a un nodo llamado neurona. Cada neurona posee una función, que modifica la entrada recibida (esta función es una suma ponderada de los valores de entrada). Tras modificar la entrada, la neurona emite una salida que es recibida por una neurona de la siguiente capa de la red neuronal. Este proceso se propaga por las distintas capas de la red neuronal hasta llegar a la capa de salida que emite una predicción [3]. La Figura 3 muestra el entramado de conexiones entre las neuronas de distintas capas.

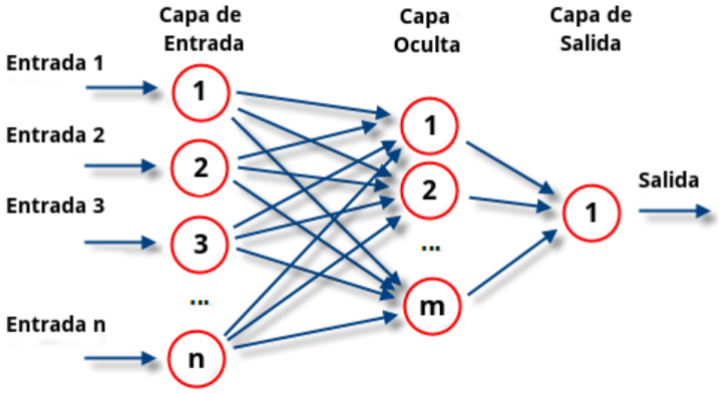


Figura 3: Una red neuronal consta de una capa de entrada, una o varias capas ocultas y una capa de salida. [3].

Para conseguir que la red neuronal funcione es imprescindible entrenarla. Para entrenar una red neuronal se deben introducir una serie de casos de entrenamiento cuyo resultado es conocido. El entrenamiento de una red neuronal consiste en modificar las funciones que alteran los valores de entrada de cada neurona. Para ello, en función del resultado final obtenido y el resultado final esperado la propia red modifica estas funciones teniendo en cuenta qué neuronas han contribuido más al resultado y si el resultado ha sido acertado. Este método se denomina "Backpropagation".

La idea clave detrás de que la red neuronal posea varias capas es que cada una de ellas reconozca elementos de la imagen más complejos que la capa anterior. Para aclarar este concepto se puede usar el ejemplo de una red neuronal encargada de reconocer dígitos manuscritos. Si uno se fija en las imágenes de la figura 4 su cerebro identifica en todas ellas el número 3, no obstante, los píxeles en negro y en blanco de cada imagen son muy distintos entre sí.

Figura 4: Las tres imágenes son fácilmente reconocibles como un tres pero los píxeles en color negro de cada una son muy distintos [3].

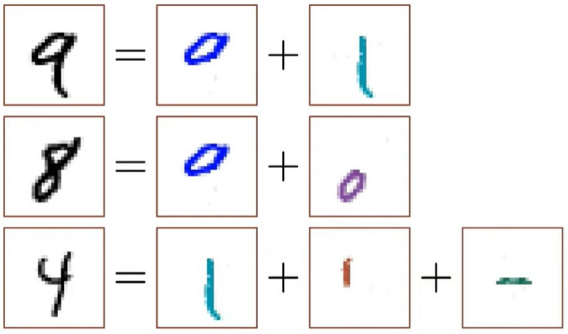


Figura 5: Tanto el cerebro como las redes neuronales descomponen los dígitos en partes más pequeñas para identificarlos [3].

El motivo por el que el cerebro reconoce un 3 en todas las imágenes es que descompone el dígito 3 en dos semicírculos abiertos por la izquierda uno encima del otro. En la misma línea que el cerebro humano, una red neuronal busca identificar los rasgos que caracterizan cada dígito. Por ejemplo, un 9 consta de un círculo completo y una línea vertical o un 8 consta de dos círculos uno encima del otro como se puede ver en la figura 5.

El proceso que realiza la red neuronal para reconocer un dígito se ilustra en la figura 6: la primera capa de la red recibe un vector en el que cada elemento representa el valor de un píxel. El valor de cada píxel es 0 si es blanco, 1 si es negro o un valor entre 0 y 1 para la escala de grises entre el negro y el blanco. De esta manera, una imagen de 28x28 se traduce en un vector con 784 entradas. En la siguiente capa, cada neurona representa pequeños segmentos que pueden estar presentes o no en la imagen, en la siguiente capa los patrones se vuelven más complejos y una neurona puede representar por ejemplo un círculo en la parte superior de la imagen o un segmento vertical largo en la parte derecha de la imagen.

Un punto clave a tener en cuenta es que la neurona que identifica un círculo en la parte superior de la imagen se activa si en la capa anterior se han activado las neuronas que reconocen los elementos más simples que componen un círculo. Finalmente, la última capa tendrá tantas neuronas como posibles salidas, en este caso una por cada dígito. La neurona que se activa es la del dígito que se ajusta a los patrones de la penúltima capa (en la figura 6 se activa el 9 ya que se ajusta a un círculo en la parte superior de la imagen y un segmento largo en el lado derecho).

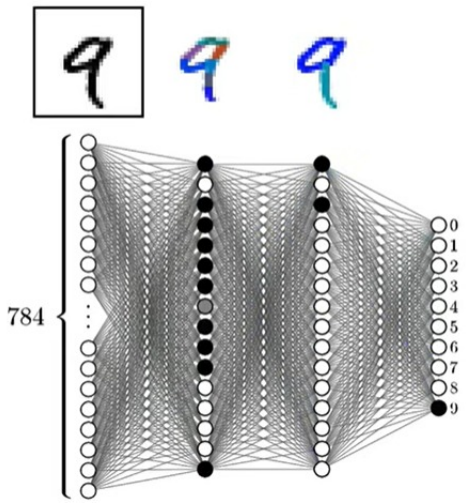


Figura 6: Cada capa de la red identifica pequeños fragmentos de la imagen que se van uniendo hasta formar un dígito en la capa final [3].

Cabe destacar que la asignación que se está estableciendo entre una neurona y una parte de la imagen es ficticia, es decir, no tiene por qué ser exactamente como se ha descrito. De hecho, es la propia red mediante el entrenamiento la que asigna a las neuronas estas características (bordes, rellenos, colores, etc.) propias del elemento que se quiere reconocer. De esta manera, mediante un entrenamiento apropiado, una red neuronal puede extraer las características que identifican un dígito, un vehículo o una expresión facial.

3.2. Cómo funciona una Red Neuronal Convolucional

Una vez aclarado el concepto de red neuronal se puede pasar a explicar qué es una red neuronal convolucional (CNN). El proceso que distingue a una CNN de otro tipo de red neuronal son las convoluciones. Las convoluciones consisten en tomar grupos de píxeles cercanos de la imagen e ir operándolos matemáticamente contra una pequeña matriz *nxn* que recorre todas las neuronas de entrada y genera una nueva matriz de salida que será la siguiente capa de neuronas. A esta pequeña matriz se la denomina "kernel".

Sobre la primera convolución del kernel se aplica la función de activación que nos permite quedarse con determinadas características de la imagen. La función de activación más utilizada se llama ReLu (Rectifier Linear Unit) y se aplica de la siguiente manera sobre el valor de cada píxel: *f(x)=max(x,0)*. [4] Esto se ilustra en la figura 7.

En la figura 7 se utilizan unos valores para el kernel que identifican las líneas verticales de la imagen. Se puede ver que tras aplicar la convolución y la función Relu, en la última matriz de la figura 7 afloran los segmentos verticales. Si los 0s en lugar de estar ubicados en la segunda columna del kernel lo estuviesen en la segunda fila, estaríamos identificando segmentos horizontales de la imagen.

En la realidad, no se aplica un solo kernel a la imagen, sino que se aplican muchos filtros. Por ejemplo, con 32 filtros distintos se obtienen 32 matrices de salida distintas. A este conjunto de matrices se le denomina "mapa de características" y dibujan características (segmentos, colores, texturas) de la imagen original que son las que harán posible distinguir un vehículo de un peatón.

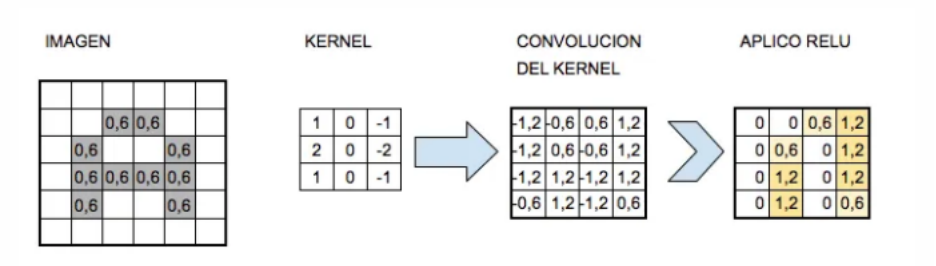


Figura 7: El kernel se itera sobre la imagen. Los valores del kernel de la imagen están elegidos para encontrar patrones verticales. [4]

Al aplicar 32 filtros distintos, la segunda capa de una red neuronal encargada de procesar una imagen de 28×28 pixels tiene nada menos que $32 \times 28 \times 28 = 25088$ neuronas, un número que en las siguientes capas se puede volver inmanejable.

La solución a este problema es un proceso llamado "subsampling" que consiste en reducir la cantidad de neuronas antes de realizar una nueva convolución. La idea es reducir el tamaño de las imagenes filtradas haciendo prevalecer las características más importantes detectadas por cada filtro. El método de subsampling más utilizado es el "Max-pooling", que reduce la imagen a la mitad y se queda con el píxel de mayor valor como se puede ver en la figura 8.

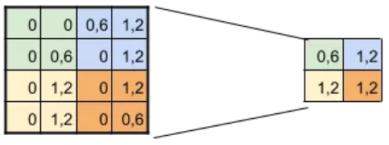


Figura 8: De cada cuadrícula 2×2 el "Max-pooling" se queda con el píxel de mayor valor. [4]

.

El proceso de filtrar, aplicar la función de activación y hacer subsampling se repite sucesivas veces. Una se han hecho varias convoluciones sobre la imagen de entrada, la salida se conecta con una red neuronal tradicional multicapa oculta sobre la que se aplica una función llamada "Softmax" encargada de normalizar la salida, es decir, de asignar una probabilidad para cada tipo de objeto candidato a ser detectado (por ejemplo *[perro:0.1, gato:0.05, pájaro:0.85]*). En la figura 9 se muestra la arquitectura completa de una CNN.

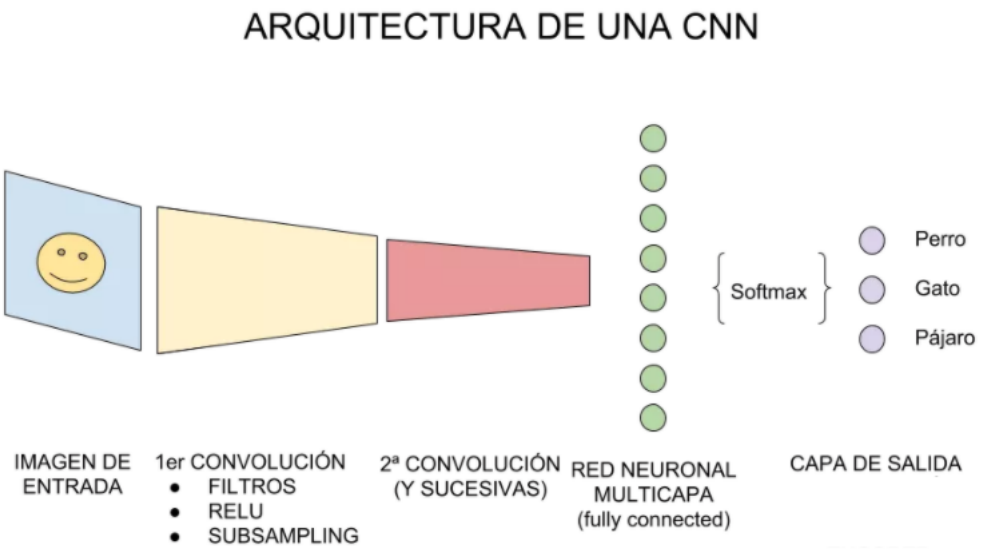


Figura 9: Arquitectura completa de una CNN. [4]

3.3. Algoritmos de detección de objetos

Dado que el propósito de este trabajo es evaluar y reentrenar los algoritmos más actuales para el reconocimiento de imágenes, es necesario aclarar cuales son los principales objetivos del reconocimiento de imágenes y cual es el método que usan los algoritmos de machine learning para alcanzar estos objetivos.

Un algoritmo de Machine Learning encargado de detectar objetos deberá:

- Detectar y clasificar múltiples tipos de objetos (peatón, coche, bici, etc.)
- Segmentar los objetos, es decir, determinar qué píxeles de la imagen forman parte del objeto.
- Dar posición a los objetos, esto es, determinar su posición (se puede dibujar un rectángulo alrededor del objeto).

Por tanto, cuando se entrena una red neuronal cuyo propósito es reconocer elementos de una imagen se debe indicar en cada imagen del conjunto de entrenamiento, qué objetos están presentes en la imagen y cual es la posición de estos. De esta manera, la salida ya no será un dígito como en el ejemplo de la sección anterior sino que serán tantos objetos como detecte el algoritmo y la posición y la dimensión de cada uno de ellos.

Para que un algoritmo de detección de objetos funcione, primero se debe de tener una CNN entrenada para reconocer determinados tipos de objetos, por ejemplo, perros y gatos. La cuestión ahora es cómo recorrer la imagen en busca de los objetos que se quieren detectar. Una posible solución sería iterar una ventana deslizante, no obstante esto plantea varios inconvenientes:

- ¿De qué tamaño será la ventana?
- ¿Cuántos píxeles se deslizará la ventana en cada paso?
- ¿Si se detecta un objeto en la ventana se puede asegurar su posición?

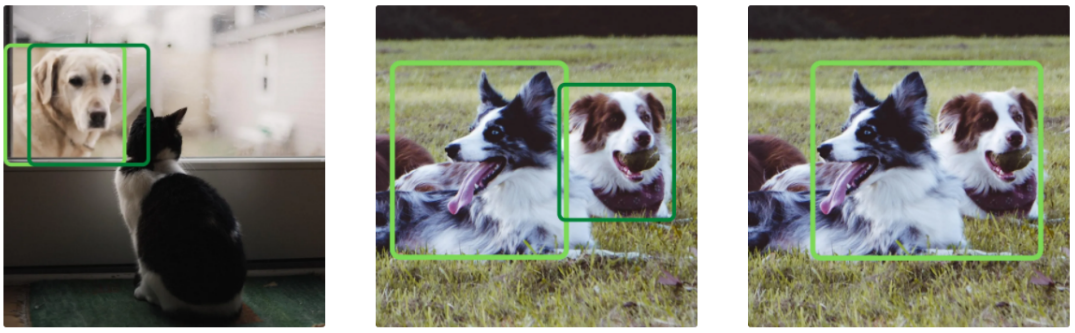


Figura 10: Desplazamientos de la ventana muy grandes o muy pequeños pueden provocar que no se detecten objetos o que se detecte dos veces el mismo objeto [6].

Estas cuestiones no son triviales ya que dependiendo del tamaño de la ventana o el desplazamiento, el tiempo de cómputo puede alargarse mucho (hay que tener en cuenta que cada nueva posición de la ventana implica una nueva clasificación con la CNN). Por otra parte, si se desplaza poco, la ventana podría detectar varias veces el mismo elemento y si se desplaza demasiado podría no detectar elementos demasiado cercanos como se puede ver en la figura 10. [6]

En las siguiente subsecciones se revisarán las propuestas más usadas actualmente para abordar estos problemas que entraña la detección de objetos.

3.3.1. R-CNN

Las R-CNN (Region Based Convolutional Neural Networks) surgen en 2014 y su idea principal es determinar regiones de interés dentro de la imagen y posteriormente emplear una red pre-entrenada para detectar objetos en esas regiones.

La clave en este caso es el algoritmo usado para seleccionar las regiones de interés. Uno de los más empleados es la búsqueda selectiva cuyo pasos son:

1. Generar una segmentación inicial de la imagen con muchas regiones candidatas. Esta segmentación inicial se basa en criterios cómo "áreas del mismo color", detección de bordes en la imagen, cambios de contraste o brillo, etc.

2. Usar un algoritmo voraz para combinar recursivamente algunas regiones y convertirlas en regiones más grandes.

3. Las regiones generadas serán las candidatas a contener uno de los objetos que se quieren identificar y se las aplicará una red convolucional (ver figura 11).

A pesar de las mejoras que se han intentado implementar, este algoritmo sigue siendo demasiado costoso a nivel de tiempo: detectar objetos en una sola imagen puede llevar más de 20 segundos. Por tanto, es poco viable emplear R-CNN en un vehículo autónomo que requiere de tiempos de reacción mucho más pequeños.

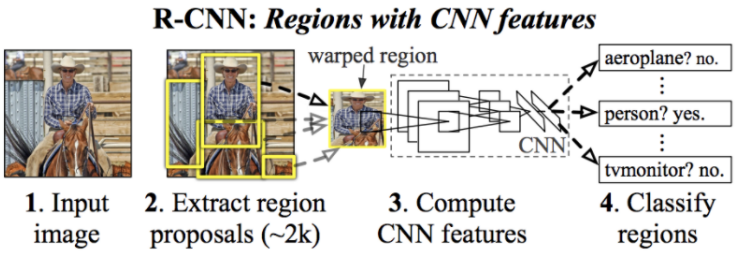


Figura 11: Las R-CNN extraen regiones de interés que posteriormente son evaluadas por una red neuronal. [7]

3.3.2. Fast R-CNN

Para intentar subsanar los prolongados tiempos de ejecucción que requiere R-CNN surgen otros dos algoritmos: Fast R-CNN y Faster R-CNN.
En R-CNN se ejecutaba un algoritmo de búsqueda selectiva que encontraba regiones de interés que eran procesadas por la CNN completa, es decir, convoluciones y red entrenada. En Fast R-CNN la imagen inicial pasa por varias convoluciones para obtener un mapa de características y una vez se tiene el mapa de características se identifican las regiones de interés sobre el mismo.

La razón por la que este algoritmo es más rápido que R-CNN es que en lugar de suministrar más de 1000 regiones candidatas a la red que tienen que ser convolucionadas y pasadas por la red entrenada, la operación de convolución solo se realiza una vez sobre la imagen inicial. El recorte en los tiempos se ver reflejado en la gráfica de la figura 12.

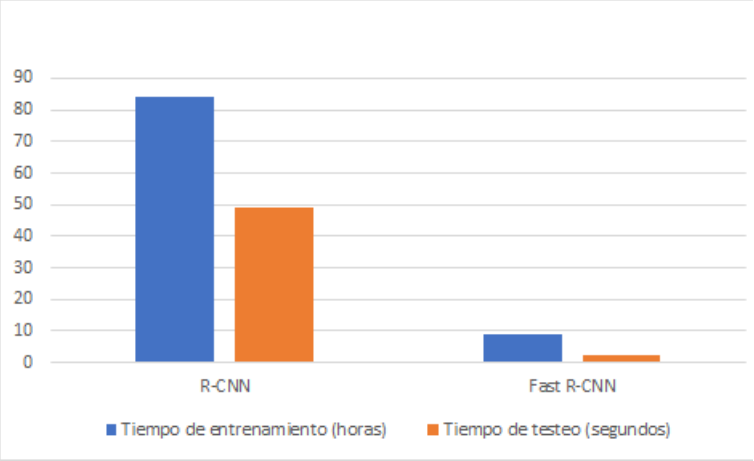


Figura 12: Fast R-CNN mejora notablemente los tiempos de entrenamiento y testeo de R-CNN. [7]

3.3.3. Yolo

Yolo (You Only Look Once) surge en 2016 con el propósito de mejorar los tiempos de ejecucción de los anteriores algoritmos de detección de objetos. Ninguno de los algoritmos anteriores observan la imagen completa sino partes de la imagen con alta probabilidad de contener un objeto, sin embargo YOLO utiliza una única red convolucional que ubica los objetos en la imagen y determina la probabilidad de pertencer a un tipo de objeto determinado.

YOLO divide la imagen en una cuadrícula $S \times S$. YOLO intentará detectar objetos sobre las celdas de la cuadrícula valiéndose de varias cajas de tamaño fijo a las que se denomina "anchors". Tomemos a modo de ejemplo una red encargada de detectar 3 categorías de objetos: peatones, coches y motos que utiliza una cuadrícula 3×3 y se vale de 2 anchors. Por cada recuadro de la cuadrícula habrá un vector que indique si alguno de los anchors ha detectado alguno de los posibles elementos que se buscan en la imagen. Si no se detecta ningún objeto para ningún anchor los elementos del vector correspondientes a los anchors serán 0 y el resto de elementos del vector darán igual ya que no se ha detectado ningún objeto. Si por el contrario uno de los anchors detecta un coche, el elemento del vector correspondiente a ese anchor será puesto a 1 (esto quiere decir que se ha detectado algún elemento) y en los siguientes elementos del vector se especificará qué tipo de objeto se ha detectado (c_1, c_2, c_3) y en qué posición (b_x, b_y, b_h, b_w) [9]. Esto se ilustra en la figura 13.

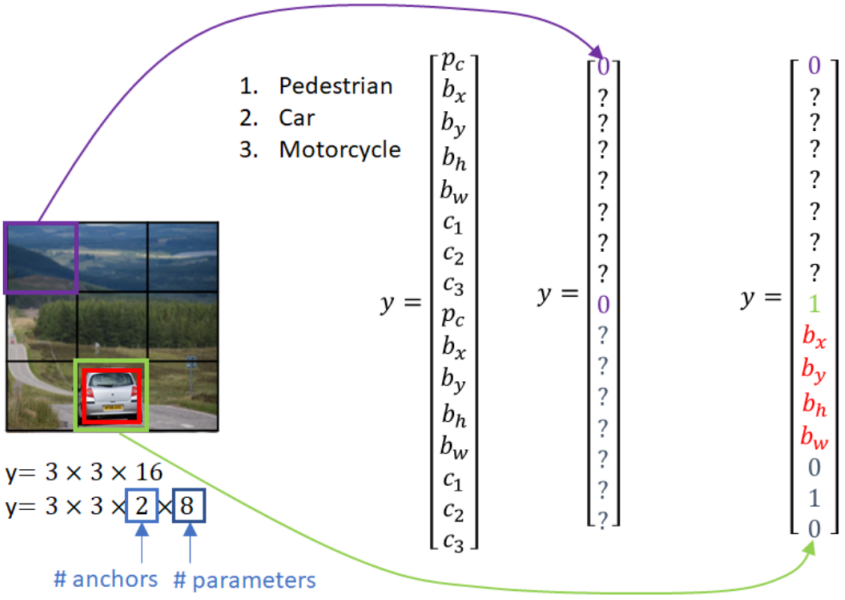


Figura 13: p_c es puesto a uno si el "anchor" corresponiente ha identificado un elemento. c_1, c_2, c_3 representan los posibles objetos a identificar y b_x, b_y, b_h, b_w identifican la posición [8].

Desde su nacimiento en 2016, YOLO se ha perfeccionado y han sido implementadas nuevas versiones (YOLOv2, YOLOv3, YOLOv4, Scaled YOLOv4, etc.). Tal y cómo se adelantaba anteriormente, el propósito de este trabajo es evaluar las dos últimas versiones de YOLO sobre distintos conjuntos de imágenes y reentrenar las últimas versiones de YOLO para mejorar su precisión a la hora de detectar elementos involucrados en la conducción cómo peatones o vehículos. Por este motivo, en la sección 5 se profundizará en la estructura de las dos últimas versiones de YOLO: YOLOv3 y YOLOv4 y se evaluará su rendimiento atendiendo a distintas métricas [1].

[1]La palabra métrica se usará como sinónimo de un indicador que mide la bondad de un modelo, en ningún caso tendrá que ver con el concepto matemático de distancia

4. Presentación de los datasets

4.1. MS COCO

MS COCO (Mictosoft Common Objects in Context) [10] es una de las bases de datos de segmentación y detección de objetos más populares. Este dataset de código abierto incluye cientos de miles de imágenes con millones de objetos ya etiquetados para el entrenamiento y evaluación de redes de deep learning. La diversidad y cantidad de imágenes que hay en este dataset lo hacen idóneo para entrenar y evaluar modelos, es por ello, que muchas de las redes más empleadas en detección de objetos han sido entrenadas en COCO (como por ejemplo YOLOv4 Darknet). La versión de este dataset lanzada en 2015 contenía 165482 imágenes de entrenamiento, 81208 imágenes en el conjunto de validación y 81434 imágenes destinadas al testeo.

Figura 14: Imagen del dataset MS COCO procesada por YOLOv4 Darknet.

El motivo por el que se ha escogido este dataset es porque las redes YOLOv3 y YOLOv4 que vamos a evaluar y reentrenar han sido entenadas en MS COCO. Por este motivo, es interesante replicar los resultados de la evaluación de estas redes sobre el conjunto de validación de MS COCO y ver si las distintas métricas que evaluan cómo de buena es una red mejoran después del reentrenamiento. Cabe remarcar que en COCO se han etiquetado más de 80 clases de objetos pero en este trabajo nos centraremos en 7 tipos de objetos que son especialmente relevantes para la conducción autónoma: personas, coches, camiones, motos, bicicletas, semáforos y autobuses. La siguiente tabla refleja la cantidad de instancias de cada objeto en el conjunto de validación de COCO sobre un total de 4952 imágenes.

Será interesante utilizar el conjunto de validación de MS COCO porque es el dataset sobre el que estaban entrenados los modelos que vamos a reentrenar, pero cómo se puede ver en el gráfico, a excepción de las clases coche y persona, el resto de clases no están muy bien representadas. Esto se debe a que COCO no es un dataset orientado específicamente a conducción autónoma y será necesario uno que sí lo sea para obtener resultados más significativos.

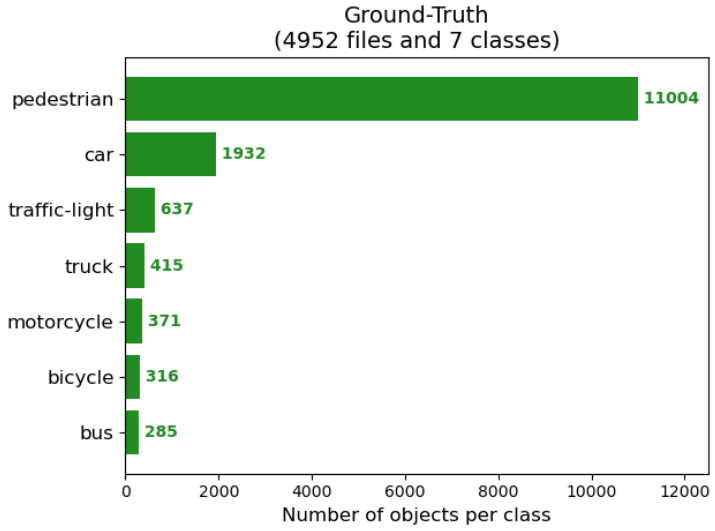


Figura 15: Instancias de cada objeto en el conjunto de validación de MS COCO.

4.2. BDD100k

BDD100k (Berkeley DeepDrive) [11] es un dataset orientado a conducción autónoma. Este dataset contiene más de 100000 imágenes tomadas en 50000 viajes a lo largo de Nueva York, San Francisco y otras regiones de Estados Unidos. En este dataset hay imágenes muy diversas desde areas residenciales y calles urbanas a grandes autovías. Además, las imagenes están tomadas en diferentes momentos del día y bajo distintas condiciones climatológicas, de esta manera al entrenar una red no se caerá en la trampa de reconocer únicamente coches a horas centrales en un día soleado. BDD contiene imágenes etiquetadas para detección de objetos, segmentación de objetos y seguimiento de objetos.

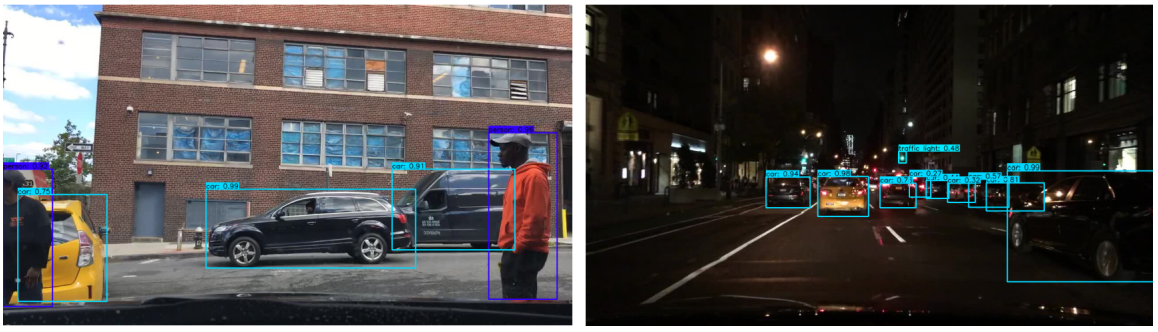


Figura 16: Dos imagenes del dataset BDD100k tomadas en condiciones distintas procesadas por YOLOv4 Darknet.

El conjunto de imágenes etiquetado para detección de objetos de BDD está dividido en 70000 para entrenamiento, 10000 para validación y 20000 para testeo. BDD es un dataset orientado a conducción autónoma y por ello, evaluar las redes YOLOv3 y YOLOv4 sobre su conjunto de validación dará una visión más precisa de cúan bien entrenadas están estas redes para ser usadas en un coche autónomo.

De las imágenes que proporciona BDD100k emplearemos las 10000 del conjunto de validación para evaluar las redes YOLOv3 y YOLOv4 antes y después del reentrenamiento. El resultado de esta evaluación será mucho más significativo que la evaluación hecha sobre MS COCO ya que serán empleadas muchas más imágenes con muchas más instancias de las clases de objetos que van a ser identificados.

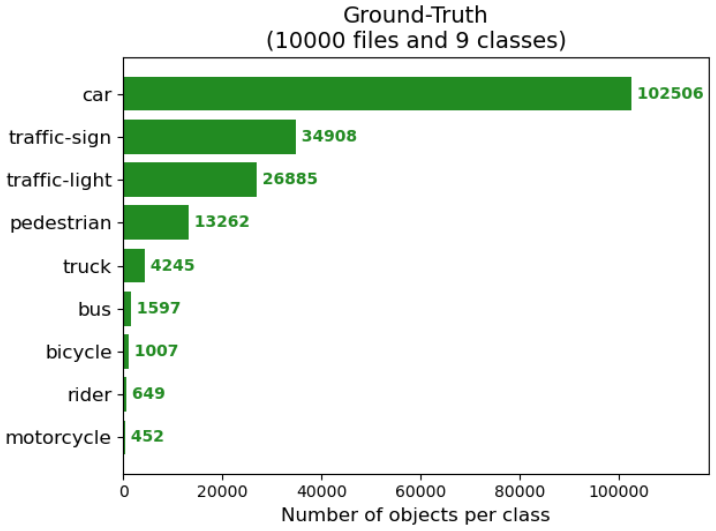


Figura 17: Instancias de cada objeto en el conjunto de validación de BDD100k.

Además se usarán las 70000 imágenes del conjunto de entrenamiento de BDD100k para reentrenar YOLOv3 y YOLOv4, que recordemos, han sido originalmente entrenadas sobre MS COCO que es un dataset generalista. Tras este reentrenamiento YOLOv3 y YOLOv4 pasarán de ser redes generalistas de reconocimiento de objetos a ser redes especialmente enfocadas a conducción autónoma. Para probar este hecho, será necesario volver a evaluaralas tras el reentrenamiento sobre el dataset de validación y comparar las métricas obtenidas antes y después del reentrenamiento.

4.3. Dataset de CIU

El dataset de Ciudad Universitaria es un dataset de elaboración propia que consta de más de 80 imágenes tomadas en los alrededores de la Facultad de Informática de la UCM en 3 días distintos. Las imágenes del dataset han sido etiquetadas manualmente empleando LabelImg para poder comparar el etiquetado manual con los resultados de YOLOv3 y YOLOv4.

El propósito de este dataset es tener un tercer conjunto de validación en un entorno local para comprobar cómo de buenos serían los modelos de detección de objetos si fuesen empleados en un coche autónomo que circula por Ciudad Universitaria. Además, es positivo conocer el proceso de etiquetado de las imágenes para un mejor entendimiento del proceso completo de detección de objetos.

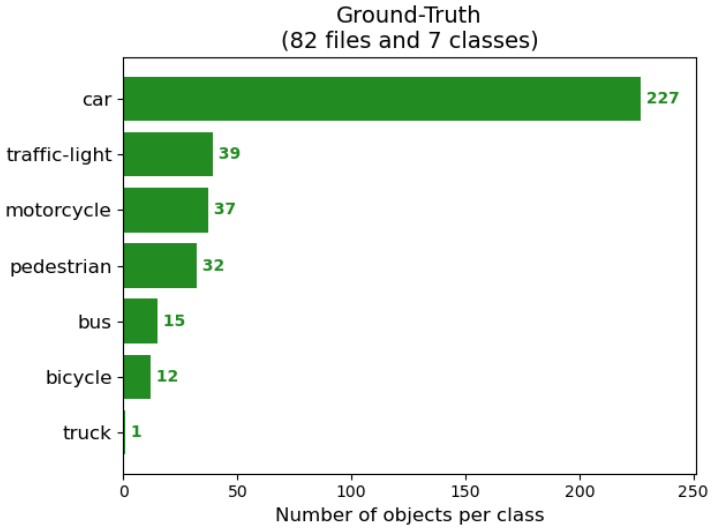


Figura 18: Instancias de cada objeto en el dataset de CIU.

Figura 19: Imagen del dataset de CIU procesada por Yolov4 Darknet.

5. Presentación de los modelos

Un modelo de machine learning es la salida que se genera cuando se entrena un algoritmo de machine learning con datos. En el caso de las redes neuronales, cuando son entrenadas lo que se modifica son las funciones que alteran los valores de entrada de las neuronas para reajustar sus pesos. Por tanto, un modelo basado en una red neuronal está formado por los pesos de las neuronas después de haber sido entrenadas.

En concreto, una red neuronal encargada de reconocer objetos debe de ser entrenada con imágenes manualmente etiquetadas. Por ejemplo, si se quisiese detectar balones de fútbol habría que suministrar a la red numerosas imagenes en las que se hayan etiquetado los balones de fútbol. De esta manera, la red neuronal puede reajustar los pesos de sus neuronas para que sean los apropiados para reconocer balones. El conjunto de todos estos pesos es lo que se denomina *modelo* cuando se habla de redes neuronales. En este experimento se usarán dos de los modelos de detección de objetos más actuales: YOLOv3 y YOLOv4, ambos entrenados sobre MS COCO. A continuación se detallan sus características.

5.1. YOLOv4

En esta sección se detallarán las características de la red neuronal YOLOv4. En general, existen dos tipos de algoritmos de detección de objetos: de una etapa o de dos etapas. La diferencia es que los algoritmos de dos etapas requieren de una etapa preliminar en la que seleccionar regiones candidatas sobre las que detectar objetos (ver sección 3.3). YOLOv4 es un detector de una etapa, es decir, prescinde de una búsqueda de regiones candidatas con el fin de llevar a cabo detecciones muy rápidas.

A continuación se enumeran los principales componentes de YOLOv4

- **Backbone:** El *backbone* de una red neuronal es la parte encargada de la extracción de características. El *backbone* de YOLOv4 tiene tres componentes diferenciados:
  - **Bag of freebies (BoF)**: Son métodos algorítmicos que aumentan el coste algorítmico del entrenamiento pero mejoran notablemente la precisión. No aumentan el coste de la inferencia. Estos métodos varían distintos parámetros de las imágenes de entrenamiento como el contraste o rotación para obtener un modelo más adaptable y preciso. YOLOv4 emplea CutMix and Mosaic data augmentation, DropBlock regularization y Class label smoothing
  - **Bag of specials (BoS)**: Son métodos algorítmicos que incrementan ligeramente el coste de la inferencia a cambio de mejorar significativamente la precisión. YOLOv4 emplea Mish activation, Cross-stage partial connections (CSP) y Multiinput weighted residual connections (MiWRC)
  - **CSPDarknet53**. [15]
- **Cuello:** El cuello de una red neuronal se encuentra entre el *Backbone* y la cabeza y se utiliza para extraer características de las imágenes en distintas fases del *backbone* (ver imagen 20). YOLOv4 hace uso de *Spacial Pyramid Pooling* (SPP) [16] y *Path Aggregation Network* (PAN) [17].
- **Cabeza:** Es la parte de la red que se encarga de la parte "real" de detección de objetos, es decir, una vez se han seleccionado las regiones candidatas o se han extraído las características

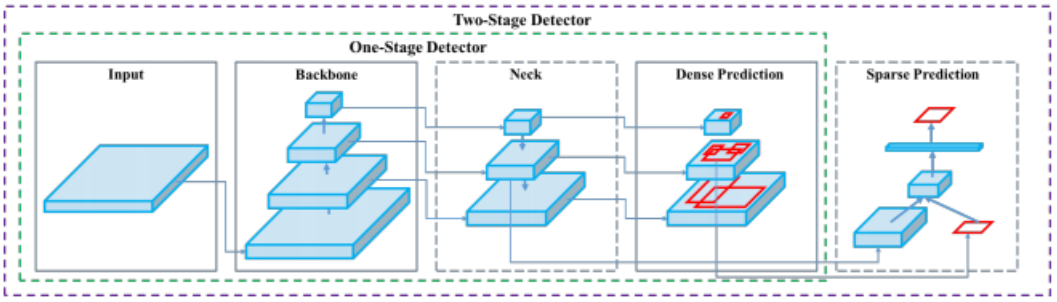


Figura 20: Detector de objetos. YOLOv4 es un detector de una etapa. [18]

de las imágenes procede a detectar objetos. En el caso de YOLO, que está basado en *anchors* (ver seccion 3.3.3), se aplicará la cabeza de la red a cada *anchor*. La cabeza de YOLOv4 es una red YOLOv3 (ver sección 5.2) [18].

5.2. YOLOv3

YOLOv3 es la red neuronal predecesora de YOLOv4 con la cual comparte varias características. El backbone de YOLOv3 también se trata de Darknet-53 [15], cuyo nombre hace referencia a las 53 capas convolucionales que tiene la red. Una de las características más destacadas que tuvo YOLOv3 es que realiza las detecciones en tres escalas distintas, para ello a las imágenes se les aplica *subsampling* (ver sección 3.2) por 32, 16 y 8 respectivamente (ver figura 21).

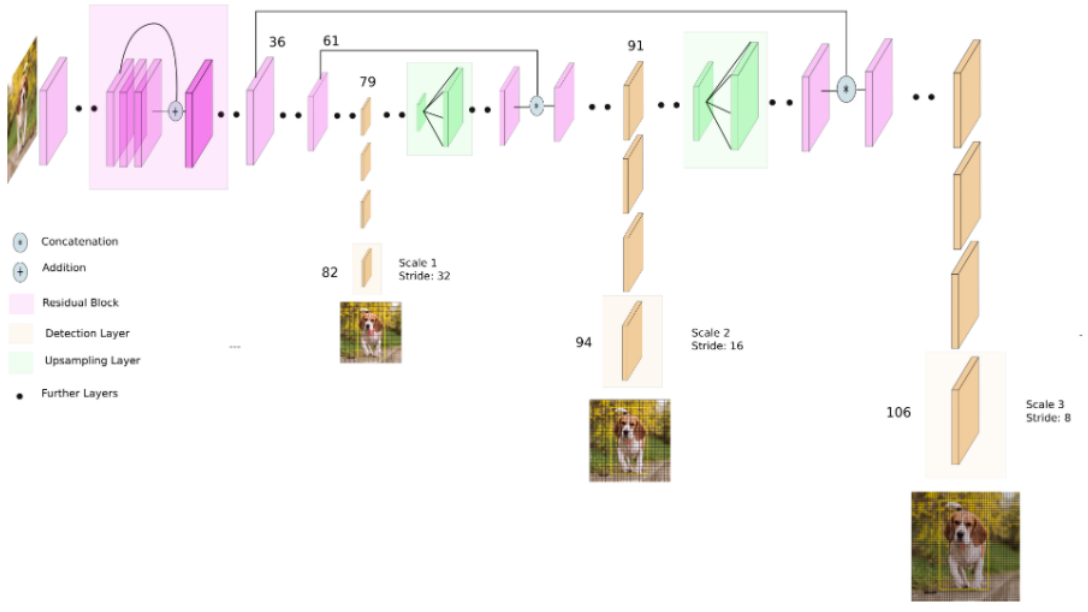


Figura 21: Arquitectura de una red YOLOv3. [24]

Por ejemplo, en una imagen 416×416 YOLOv2 hubiese predecido $13 \times 13 \times 5 = 845$ *boxes* sobre las que realizaría inferencia, en cambio YOLOv3 al hacer detecciones en tres escalas multiplica por 10 el número de *boxes* sobre los que realiza inferencia detectando de esta manera objetos más pequeños en las imágenes [23][24].

6. Métricas

En esta sección se presentan las métricas que se van a usar para valorar cómo de bueno es un modelo. Las métricas proporcionan un marco objetivo para comparar los resultados de dos modelos distintos y poder hacer una valoración de cual es el mejor.

6.1. Precisión, recall y AP

Los conceptos precisión y recall nacen de una serie de indicadores que contabilizan si las predicciones de un clasificador binario, es decir, positivo o negativo, se han realizado correctamente:

- **TP (True positives):** Casos en los que la predicción fue positiva y acertó.
- **TN (True negatives):** Casos en los que la predicción fue negativa y acertó.
- **FP (False positives):** Casos en los que la predicción fue positiva y falló.
- **FN (False negatives):** Casos en los que la predicción fue negativa y falló.

	Positivo	Negativo
Predicción positiva	TP	FP (Error de tipo I)
Predicción negativa	FN (Error de tipo II)	TN

Estas cuatro métricas sirven para calcular los ratios de precisión y recall:

- **Precisión:** El ratio de precisión indica cuantos positivos del total de positivos predichos han sido correctos. La precisión sirve para entender cómo de acertado es un modelo cuando predice un positivo. Una precisión cercana a 1 indica que el modelo acierta casi siempre que predice un positivo.

$$precision = \frac{TP}{TP + FP} \tag{1}$$

- **Recall:** El ratio de recall, también llamado sensibilidad, es el cociente entre los positivos bien predichos y el total de positivos. Un recall cercano a 1 indica que el modelo acierta en casi todos los casos en los que la predicción correcta es positivo.

$$recall = \frac{TP}{TP + FN} \tag{2}$$

Precisión y recall no son dos métricas independientes entre sí, existe un equilibrio entre ambas. En concreto, si el clasificador aumenta la precisión, disminuirá el recall y viceversa como se puede ver en la Figura 22. Esto se debe a que si el clasificador es muy preciso, es decir, solo predice positivo cuando está extremadamente seguro de que se trata de un positivo, entonces se predicirá negativo para muchas imágenes que tenían que haber sido predichas positivo (FN). Si por el contrario el clasificador siempre predice positivo el recall será 1 porque nunca fallará una predicción positiva pero su tasa de aciertos será muy baja (0.5 si hay igual de positivos reales que negativos reales en el dataset).

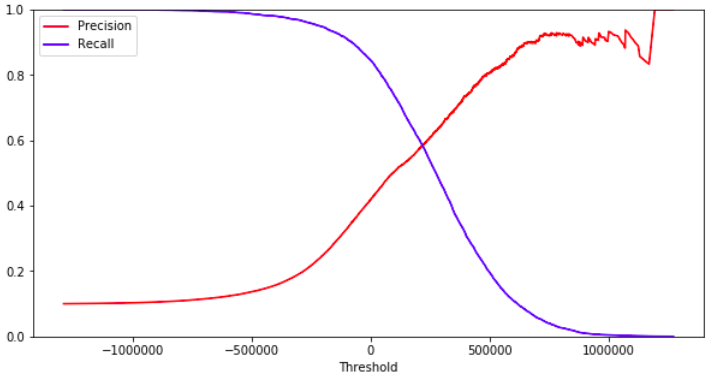


Figura 22: Equilibrio precisión-recall en función del umbral de decisión. [12]

La curva precisión-recall (PR) es el resultado de contraponer en una gráfica como varía la precisión en función del recall cuando se cambia el umbral a partir del cual el modelo considera positiva una instancia. Esta gráfica permite ver a partir de que recall la precisión se degrada. Si un modelo es bueno la curva precisión recall se acercará mucho a la esquina superior derecha, es decir aún teniendo una sensibilidad alta las predicciones son precisas. En la Figura 23 se pude ver la curva precisión-recall de YOLOv4 para la clase teléfono móvil del dataset MS COCO, la cual no es especialmente buena pero tampoco se puede considerar mala.

Encima de la Figura 23 podemos observar la siguiente anotación: *"cell-phone AP = 55.33"*, este valor es el Average Precision y hace referencia al área bajo la curva PR. Este valor sirve para evaluar y comparar el rendimiento de distintos modelos. Cuanto más próximo sea el AP a 1 mejor será el modelo.

$$AP = \int_0^1 C_{PR} \tag{3}$$

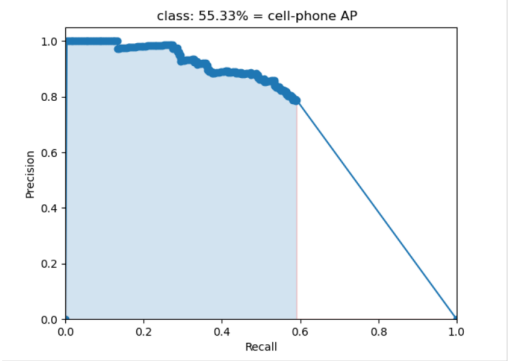


Figura 23: Curva precisión-recall de YOLOv4 para la clase teléfono móvil del dataset MS COCO.

No obstante, hay que remarcar que hasta ahora hemos hablado de métricas para clasificadores binarios, es decir, que devuelven un 0 o un 1. En detección de objetos, si queremos medir cómo de bueno es un modelo, hay que comparar la posición real de un objeto con la posición predicha por el modelo. Para que las métricas anteriores tengan sentido en un modelo de detección de objetos es necesario definir cuando se va a considerar que el modelo ha acertado o ha fallado. Esta cuestión se abordará en el siguiente apartado.

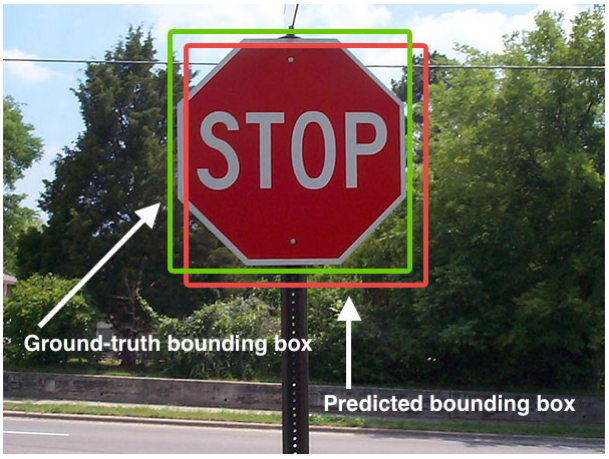


Figura 24: Ground truth y prediction en una señal de Stop. [13]

6.2. IoU

A la hora de validar las predicciones de un modelo encargado de detectar objetos hay que diferenciar entre la posición real del objeto, que se etiqueta manualmente, y la posición predicha por el modelo. Estas posiciones quedan definidas por cuatro valores *(x, y, h ,w)* donde x e y son la posición del píxel superior izquierdo del recuadro, h la altura y w la anchura. Se puede observar esta diferenciación en la Figura 24.

- **Ground truth:** Posición real del objeto. Se etiqueta manualmente en las imágenes de los conjuntos de entrenamiento y de validación.
- **Prediction:** Posición del objeto predicha por el modelo.

Los recuadros ground truth y prediction sirven para determinar el valor de IoU (Intersection over Union), que se define como el cociente entre el área de intersección y el área de unión entre ambos como se puede ver en la Figura 25. Este cociente se utiliza para marcar un umbral a partir del cual se considera que el modelo ha acertado, a este umbral se le denomina *umbral IoU*. Una observación importante sobre el umbral IoU es que cuanto más bajo sea, más positivos predecirá el modelo. En otras palabras, en función del umbral IoU una misma predicción puede ser un *False Positive* o un *True Positive* (ver figura 26), por lo que este valor afecta directamente a la curva PR y al valor AP. Un valor comunmente aceptado para el umbral IoU y que será el que usemos en este trabajo es 0,7.

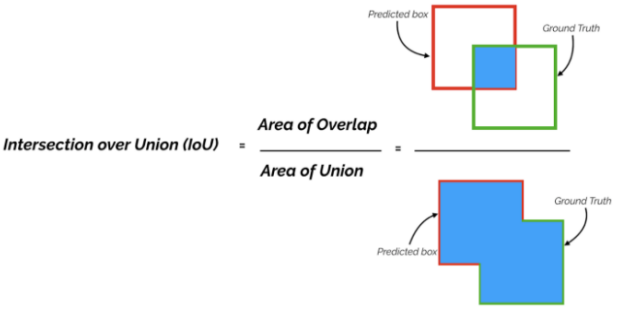


Figura 25: IoU es el ratio entre la intersección y la unión de los recuadros ground truth y prediction. [14]

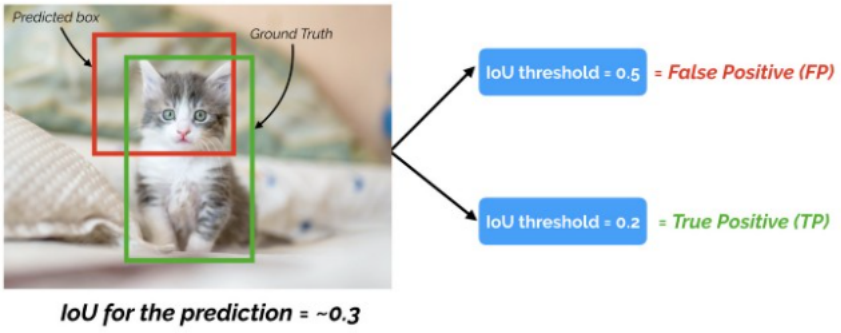


Figura 26: Una misma predicción puede ser FP o TP en función del umbral IoU. [14]

6.3. FPS y coste energético

Para un modelo de detección de objetos orientado a conducción autónoma no es suficiente ser preciso. No sirve de nada una precisión muy elevada si se tarda varios segundos en inferir los objetos en la imagen. Esto se debe a que pasado este tiempo la decisión que toma el coche autónomo ha quedado obsoleta porque el resto de coches y peatones que circulan por la vía han cambiado su posición. La unidad de medida más extendida para determinar la velocidad de la red son los FPS.

Los FPS (Frames per Second) hacen referencia a la cantidad de imágenes consecutivas que se muestran por pantalla cuando se reproduce un vídeo. Cuando se presentan imágenes que forman parte de la misma acción, a partir de 10-12 FPS el ser humano deja de ver imágenes separadas y en su mente se forma una imagen en movimiento. Por otra parte, en torno a 30 FPS es la velocidad habitual para un vídeo o una película. [25]

En el caso de un detector de objetos los FPS hacen referencia a la cantidad de imágenes que se pueden procesar en un segundo. Este valor no solamente depende del algoritmo que se esté usando sino también del hardware. Por este motivo, en la parte de experimentación, se evaluará la velocidad de inferencia sobre distintos hardwares.

Además de ser preciso y veloz, es importante que un detector de objetos no consuma una cantidad de energía excesiva ya que existe una relación directa entre velocidad y consumo. Por ejemplo, cuando se dispone de un hardware muy potente como GPUs de alta gama o CPUs con muchos hilos de procesamiento, se realiza una inferencia más rápida pero el consumo energético también es mucho mayor. El consumo de una unidad de procesamiento se mide en Watios (W) pero para poder obtener un compromiso entre velocidad y consumo será más interesante fijarse en métricas como las instrucciones por Watio o los Frames por Segundo por Watio (FPS/W). [26]

El cálculo de la energía consumida por el hardware se hará a *groso modo* empleando los consumos proporcionados por el fabricante. Cabe remarcar que se podría entrar en cuestiones más finas que influyen en el consumo energético como la cantidad de memoria RAM o la cercanía de los datos a la GPU. Asimismo, otros factores como el coste de fabricación o el volumen son importantes a la hora de diseñar un detector de objetos para un coche autónomo pero exceden los propósitos de este trabajo.

7. Metodología y frameworks

En las anteriores secciones (secciones 4 y 6) se han introducido algunos aspectos de la experimentación. El propósito de esta sección es detallar con claridad qué metodología se ha usado y qué frameworks han sido necesarios.

Cabe recordar que el propósito de este trabajo es evaluar el rendimiento de una aplicación encargada de detectar objetos viales, tanto a nivel de precisión como de coste energético o velocidad de inferencia. Con este propósito, la experimentación está dividida en dos partes diferenciadas para las cuales se han usado frameworks distintos

- **Evaluación de la precisión:** Comparativa de la precisión utilizando las métricas descritas en la sección 6 para distintas clases de objetos viales. Se compararán la red original YOLOv4, su predecesora YOLOv3 y la red YOLOv4 reentrenada con las imágenes del conjunto de entrenamiento del dataset BDD100k (ver sección 4.2).
- **Evaluación de la velocidad y coste energético:** Se tomará la red que mejores resultados ofrezca en la anterior comparativa y se compararán las métricas de velocidad de inferencia y el coste energético sobre distintos hardwares.

7.1. Reentrenamiento y evaluación de la precisión

Para reentrenar YOLOv4 y evaluar la precisión se van a utilizar tres frameworks distintos

- **Repositorio original de Darknet:** Es el repositorio original de YOLOv4 Darknet escrito por AlexeyAB, uno de los creadores de YOLO. Este repositorio provee herramientas para realizar inferencias con YOLOv3 y YOLOv4, reentrenar YOLOv4 con objetos personalizados, evaluar una red reentrenada sobre el servidor original de COCO y diversas funcionalidades más. [19]
- **Repositorio de YOLOv4 adaptado a Tensorflow:** Este repositorio, creado por *hunglc007* [20] ofrece algunas de las herramientas del respositorio original de Darknet adaptadas a tensorflow. No obstante de este repositorio solo se usará una carpeta llamada **mAP** que contiene los scripts necesarios para medir el AP dadas las anotaciones de un conjunto de validación y las predicciones hechas por Darknet. Esta carpeta se puede encontrar también en mi repositorio de Github [21].
- **Scripts para cambiar formatos de anotaciones:** Tanto el repositorio original de Darknet, como el dataset BDD100k, el dataset COCO o el repositorio de YOLOv4 adaptado a Tensorflow requieren de distintos formatos para anotar las imágenes. Estos formatos son muy específicos y no están suficientemente estandarizados por lo que se necesitan scripts que tomen una anotación en un determinado formato y lo conviertan al formato requerido por el framework que se va a usar. Estos scripts, que en su mayoría son de elabroación propia, se pueden encontrar en el repositorio de Github [21] junto a algunas instrucciones de uso.

Para evaluar las redes YOLOv4, YOLOv3 y YOLOv4 reentrenada es imprescindible tener los archivos con los pesos de las neuronas de cada red

- **Pesos de YOLOv4:** Obtenidos del enlace de descarga proporcionado en el repositorio original de Darknet [19].
- **Pesos de YOLOv3:** Obtenidos del enlace de descarga proporcionado en el repositorio original de Darknet [19].

- **Pesos de YOLOv4 reentrenada:** Obtenidos tras reentrenar en el repositorio original de Darknet la red YOLOv4. Se pueden encontrar algunos detalles de la configuración de las capas de YOLOv4 para el reentrenamiento en el apéndice A. La situación óptima hubiese sido poder probar varias configuraciones y valorar cual ha dado mejor resultados. Esto no ha sido posible ya que el reentrenamiento se realizó sobre mi GPU Nvidia GTX 1080 que a pesar de ser una gráfica aceptable, tardó 10 días en completar el reentrenamiento.

  Instalar Darknet y reentrenar una red es un proceso complejo con muchas dependencias. Para llevarlo a cabo se siguieron minuciosamente las instrucciones del repositorio de Darknet y los tutoriales de TheCodingBug [32].

Estas tres redes serán evaludas sobre los tres conjuntos de validación presentados en la sección 4 y posteriormente se compararán los resultados atendiendo a las métricas de la sección 6 para determinar cual es el mejor.

Tal y como se adelantaba anteriormente, para llevar a cabo la evaluación de la precisión es imprescindible que los datasets estén anotados ajustándose a los requerimientos de cada framework. A tal fin las siguientes anotaciones fueron necesarias:

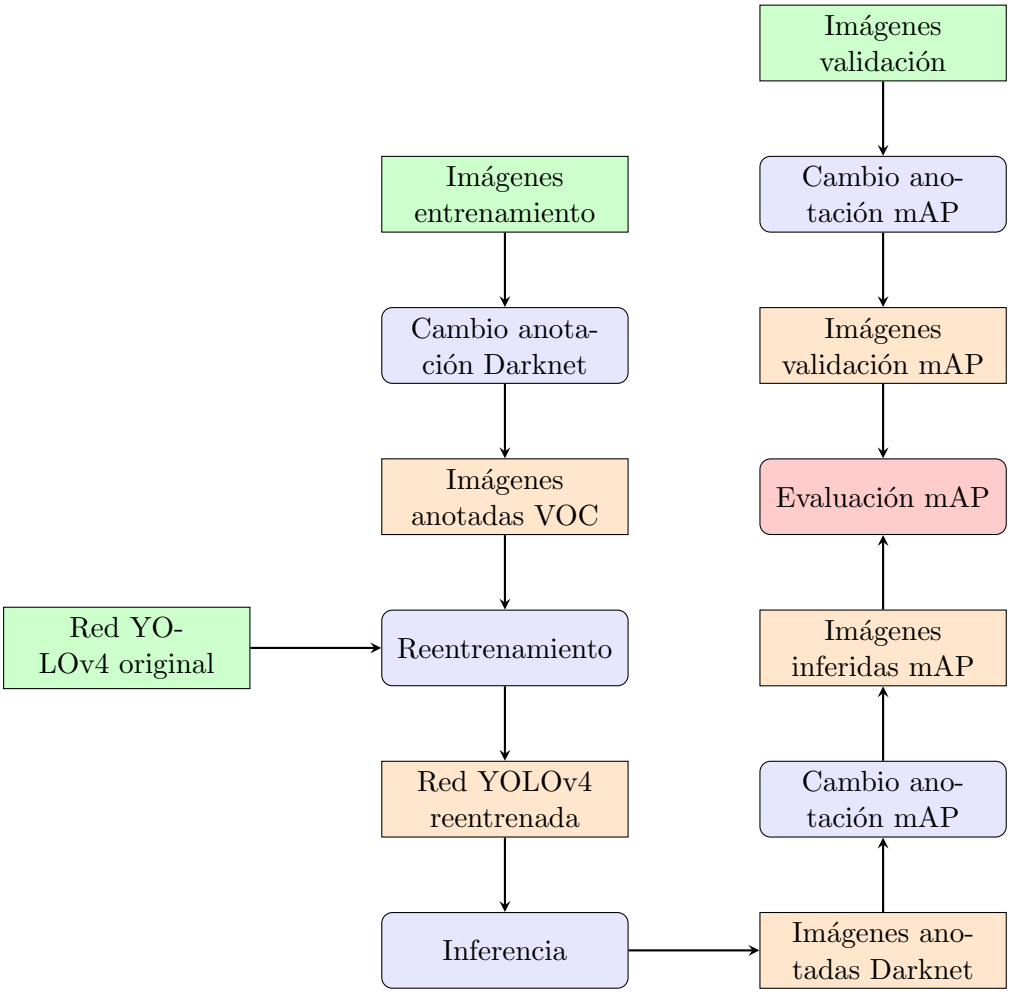
- Anotaciones originales de cada dataset en sus respectivos formatos.
- Anotaciones para testear en darknet en formato darknet.
- Archivos json inferidos por darknet en la notación de darknet.
- Archivos ground-truth anotados en el formato del repositorio YOLOv4 adaptado a tensorflow (mAP).
- Archivos prediction anotados en el formato del repositorio YOLOv4 adaptado a tensorflow (mAP).

Es importante que los archivos ground-truth y prediction de cada dataset estén correctamente anotados ya que es el *input* que necesita el repositorio de YOLOv4 adaptado a tensorflow para calcular el mAP.

Para clarificar esto y a modo de conclusión, en el diagrama de la siguiente página se puede ver el flujo de trabajo que se ha seguido para evaluar la precisión y en el cuadro 3 los scripts utilizados en cada paso, todos disponibles en mi repositorio de github [21].

	COCO	BDD100k	CIU
Cambio anotación a darknet	from_tf_to_darknet _val_annotations coco_annotation	bdd_to_coco cocobdd100k_to_ jsoncoco	Se etiquetaron directamente en formato darknet
Cambio anotación mAP tras la inferencia	yolocoord_to_box2d _not_fixed_size	yolocoord_to_box2d _fixed_size	yolocoord_to_box2d _fixed_size
Cambio anotación mAP desde el dataset	coco_ground_truth_ generator	bdd_ground_truth_ generator	ciu_ground_truth_ generator

Cuadro 3: Scripts empleados para convertir entre anotaciones



7.2. Evaluación de la velocidad de inferencia y el coste energético

En la segunda parte de la experimentación se medirá la velocidad de inferencia en FPS de los modelos y se calculará el coste energético aproximado que es necesario para realizar la inferencia.

Para medir la velocidad de inferencia se usará el framework OpenVINO. OpenVINO (Open Visual Inference & Neural Network Optimization) es un conjunto de herramientas para visión artificial ofrecidas por Intel. En OpenVINO, se pueden construir y entrenar modelos de inteligencia artificial en la nube, en frameworks populares como TensorFlow, MXNet y Caffe. Una de las herramientas más potentes que ofrece OpenVINO y que será la que se use para este trabajo es la posibilidad de desplegar distintos modelos de visión artificial en productos de Edge, es decir, realizar la fase de inferencia de la red neuronal en la nube de manera que se puede desplegar un modelo en distintos hardware (tanto CPU como GPU) de Intel. [27]

El "toolkit" OpenVINO dispone de una interfaz gráfica en la que desplegar los modelos llamada "Deep Learning Workbench" (ver Figura 27). En esta herramienta se selecciona el modelo, el dataset y el hardware sobre los que se quiere llevar a cabo la inferencia y se obtienen resultados sobre los FPS y la latencia durante la inferencia.

Figura 27: OpenVINO Deep Learning Workbench es la herramienta gráfica sobre la que se despliegan las funcionalidades del "toolkit" OpenVINO

Además de realizar la inferencia en la nube utilizando hardwares de Intel, OpenVINO ofrece la posibilidad de realizar la inferencia valiéndose de su conjunto de herramientas en el ordenador de uno mismo. Para ello se deben seguir los manuales proporcionados por Intel en la web de OpenVINO [27] e ir ejecutando los scripts apropiados en cada paso.

El mayor problema que se ha planteado para esta parte de la experimentación es que OpenVINO emplea un formato muy concreto para las redes llamado IR (Intermediate Representation). Este formato está compuesto por dos archivos un *.xml* y un *.bin* pero hasta ahora se había usado un archivo *.weights* que es el adecuado para Darknet. OpenVINO incluye un *model zoo* con archivos IR para algunas redes cómo ssd o YOLOv2, pero las herramientas que provee para transformar una red YOLOv4 reentrenada a IR no son suficientes.

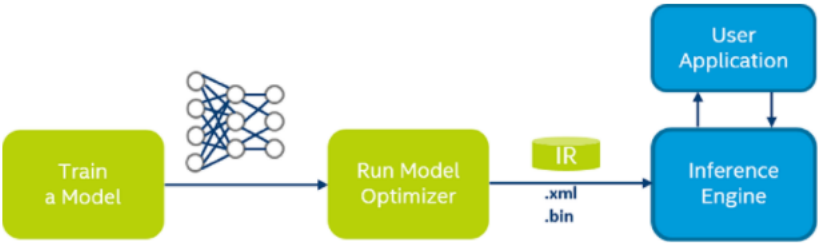
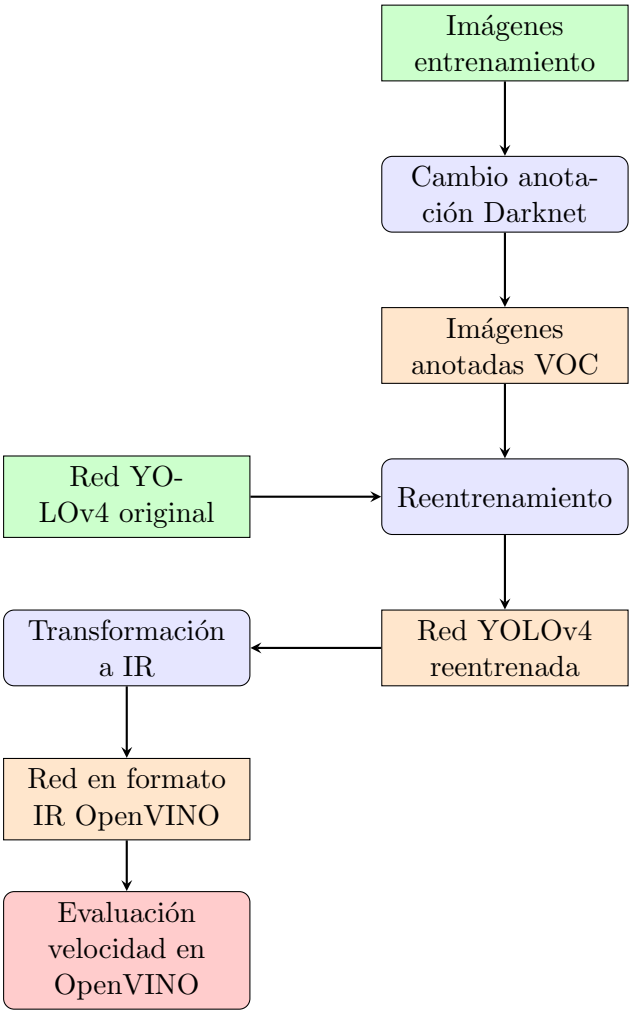


Figura 28: Flujo de trabajo de las herramientas de OpenVINO [27]

Para llevar a cabo esta conversión, en este trabajo se han usado los scripts y las instrucciones en el repositorio de TNTWEN [31] para transformar la red reentrenada a IR. Nótese que el entorno para hacer esta conversión requiere la última versión de OpenVINO pero versiones antiguas de Python y Tensorflow lo cual puede ocasionar problemas si no se configura correctamente el entorno.

En el siguiente diagrama queda claro el flujo de trabajo para la evaluación de la velocidad de inferencia en OpenVINO.



Cómo apunte final hay que mencionar que OpenVINO está orientado más allá de la evaluación de redes neuronales. OpenVINO provee las herramientas necesarias para transformar la red en formato IR a una aplicación a nivel de usuario como se puede ver en la figura 28. No obstante, esta última fase de despliegue de la aplicación no se ha explorado en este trabajo y queda como posible continuación.

8. Resultados

Como se explica en la seccion 7, la experimentación está divida en dos partes y de la misma manera estarán divididos los resultados: evaluación de la precisión de las redes y evaluación de la velocidad y el coste energético.

8.1. Evaluación de la precisión

Cabe recordar que se van a evaluar tres redes neuronales

- YOLOv4
- YOLOv3
- YOLOv4 reentrenada

sobre tres conjuntos de validación

- MS COCO
- BDD100k
- Dataset de CIU

A lo largo de la experimentación nos centraremos en las diez clases de objetos recogidas en el dataset BDD100k. No obstante, no las diez clases de BDD100k están entre las 80 clases del dataset MS COCO sobre el que han sido entrenadas YOLOv3 y YOLOv4 y por tanto estas dos redes no estarán preparadas para detectar algunas clases de interés. Sin embargo, YOLOv4 reentrenada ha sido entrenada sobre BDD100k y sí que podrá reconocer las diez clases de objetos. Cabe destacar que tampoco se puede evaluar la precisión de una red para una clase de objeto que no ha sido previamente etiquetado en el conjunto de validación.

Por ejemplo la clase *coche* es reconocida por todas las redes y está etiquetada en todos los datasets, pero la clase *señal de tráfico* no está etiquetada en COCO y por tanto YOLOv3 y YOLOv4, que son redes entrenadas en COCO no podrán reconocer semáforos o en el momento de evaluar *YOLOv4 reentrenada* sobre MS COCO no se podrá medir su precisión para la clase semáforo. En el cuadro 4 se pueden ver las clases recogidas en cada dataset.

	MS COCO	BDD100k	CIU
car	✓	✓	✓
bus	✓	✓	✓
pedestrian	✓	✓	✓
bicycle	✓	✓	✓
truck	✓	✓	✓
motorcycle	✓	✓	✓
rider		✓	
traffic-sign		✓	
traffic-light	✓	✓	✓

Cuadro 4: Clases representadas en cada dataset.

Los resultados se presentarán agrupados por dataset de esta manera será fácil comparar la precisión de cada una de las redes sobre el mismo conjunto de datos.

8.1.1. Precisión sobre MS COCO

MS COCO es el dataset sobre el que han sido entrenadas las redes YOLOv3 y YOLOv4. Sobre el conjunto de validación de COCO, AlexeyAB [19], afirma que YOLOv4 tiene un mAP entre $62,8\,\%$ y $65,7\,\%$ dependiendo de algunas configuraciones. Por su parte, en la página oficial de YOLO [22] se sostiene que YOLOv3 ha obtenido un mAP entre $51,5\,\%$ y $57,9\,\%$ dependiendo del tamaño de las imágenes. Estos resultados son muy acordes por los obtenidos para este trabajo como se puede ver en la figura 30.

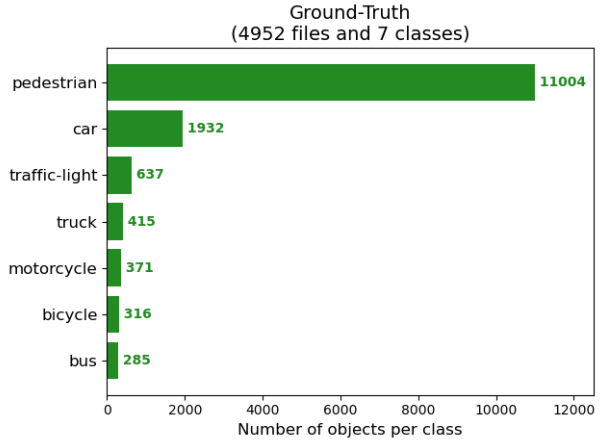
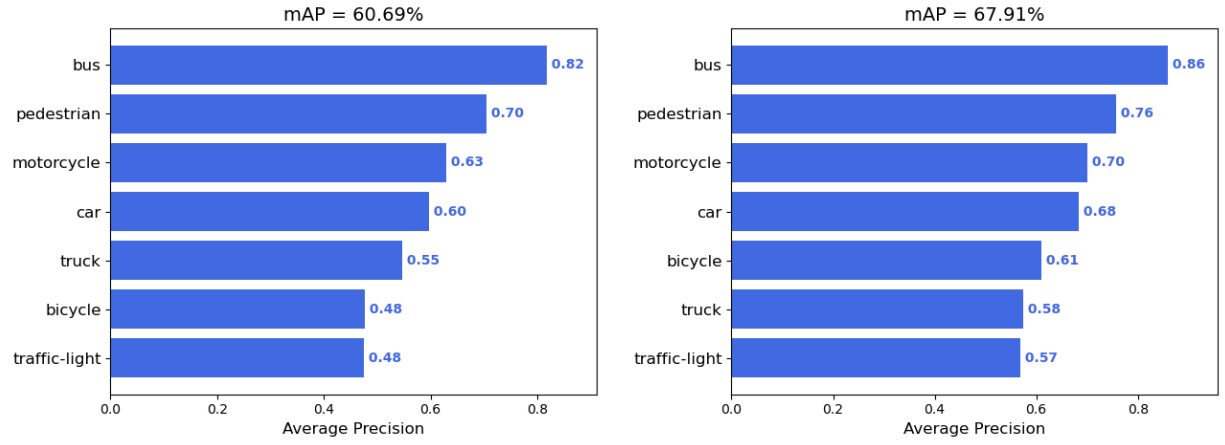


Figura 29: Número de instancias por clase en el conjunto de validación de MS COCO

Las diferencias entre los resultados anteriormente mencionados y los obtenidos se deben a que para este trabajo se ha calculado el AP promedio únicamente de las clases relacionadas con la conducción autónoma, mientras que los resultados de YOLO [22] y AlexeyAB [19] han sido calculados sobre las 80 clases de MS COCO. Pese a ello, los resultados son extremadamente similares por lo que se puede afirmar que la metodología y el uso de los frameworks para evaluar el mAP ha sido el correcto.



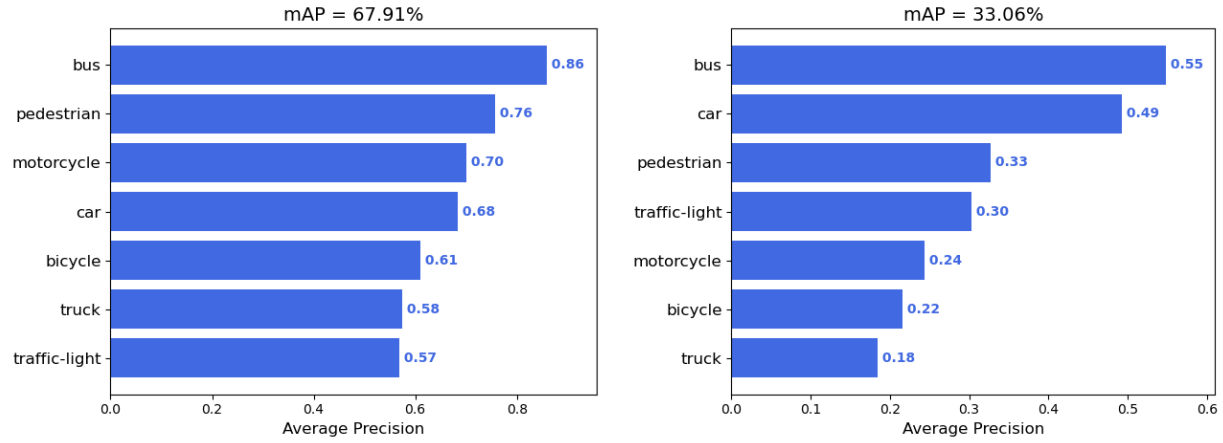
(a) YOLOv3

(b) YOLOv4

Figura 30: Resultados AP de YOLOv3 y YOLOv4 sobre MS COCO

En la figura 31 se recogen los resultados obtenidos por YOLOv4 y YOLOv4 reentrenada sobre MS COCO. Se puede observar que la precisión se ha reducido drásticamente y que por tanto, podemos considerar que sobre MS COCO el reentrenamiento de la red ha sido completamente infructuoso. No obstante, existe una explicación para este decremento en la precisión.

YOLOv4 reentrenada ha sido entrenada sobre el conjunto de entrenamiento de BDD100k, que contiene imágenes de tamaño fijo 1280×720 grabadas desde un coche, es decir, son imágenes en un contexto muy concreto como son carreteras o ciudades. Por su parte YOLOv4 ha sido entrenada sobre MS COCO, que contiene imágenes de múltiples tamaños y en contextos muy diversos. El conjunto de validación de MS COCO, siguiendo la misma filosofía que su conjunto de entrenamiento, está formado por imágenes en contextos muy diversos, por lo que YOLOv4 reentrenada no está bien preparada para reconocer los objetos en las imágenes de COCO.



(a) YOLOv4

(b) YOLOv4 reentrenada

Figura 31: Resultados AP de YOLOv4 y YOLOv4 reentrenada sobre MS COCO

En todo caso el objetivo de este trabajo es diseñar una red neuronal orientada a la conducción autónoma, por lo que aunque sea interesante evaluar *YOLOv4 reentrenada* sobre el conjunto en el que originalmente había sido entrenada *YOLOv4*, no podemos considerar estos resultados conclusivos ya que se está evaluando la precisión de la red sobre imágenes sin un contexto de conducción autónoma.

8.1.2. Precisión sobre BDD100k

BDD100k es el dataset del cual se han usado 70000 imágenes para el reentrenamiento de YOLOv4. Los resultados que se presentan a continuación son los obtenidos al evaluar las redes neuronales sobre 10000 imágenes en el conjunto de validación. En la figura 32 se puede observar que las clases relacionadas con la conducción autónoma están mucho mejor representadas que en MS COCO. Además, se incluyen dos nuevas clases: *rider* y *traffic sign*.

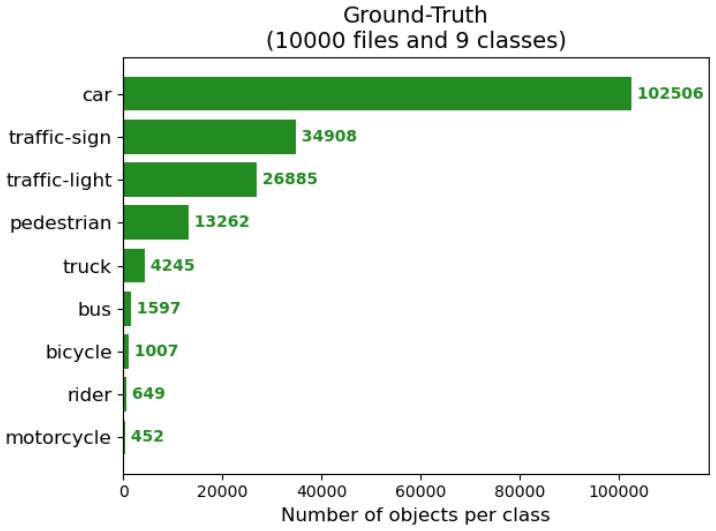
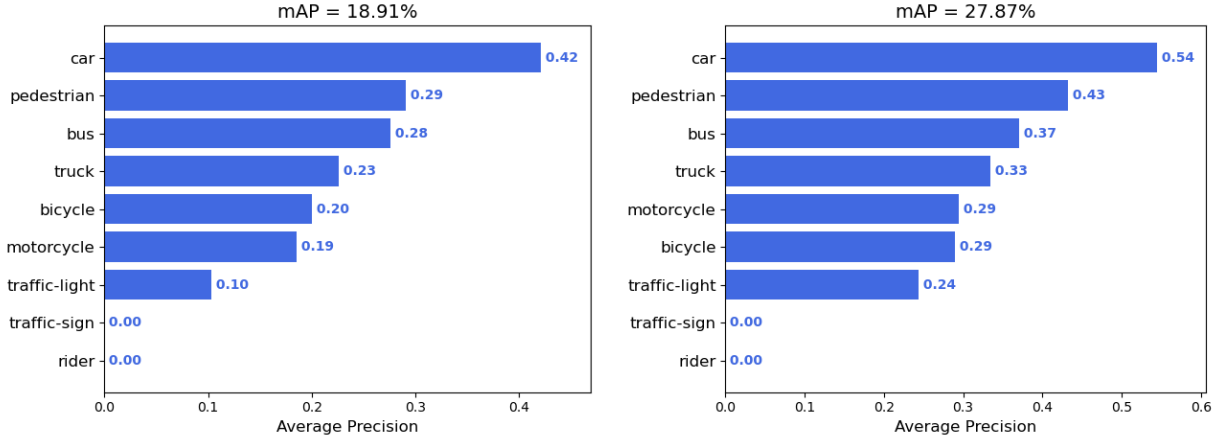


Figura 32: Número de instancias por clase en el conjunto de validación de BDD100k

Primero veamos los resultados para las redes originales YOLOv3 y YOLOv4. Como se puede ver en la figura 33, YOLOv4 supera a YOLOv3 sustancialmente. Esta mejora es consistente con los resultados obtenidos sobre el dataset MS COCO en el que YOLOv4 ya mejoraba aproximadamente en un 7 % a YOLOv3. Hay que destacar que hay dos clases, *traffic-sign* y *rider*, para las cuales se ha obtenido un AP igual a 0. La explicación es, que tal y como se detallaba al inicio de esta sección, YOLOv3 y YOLOv4 han sido entrenadas para detectar las 80 clases recogidas en COCO pero BDD100k incluye las clases *traffic-sign* y *rider* que YOLOv3 y YOLOv4 no están preparadas para detectar por no formar parte de COCO. Esto se solucionará tras el reentrenamiento.

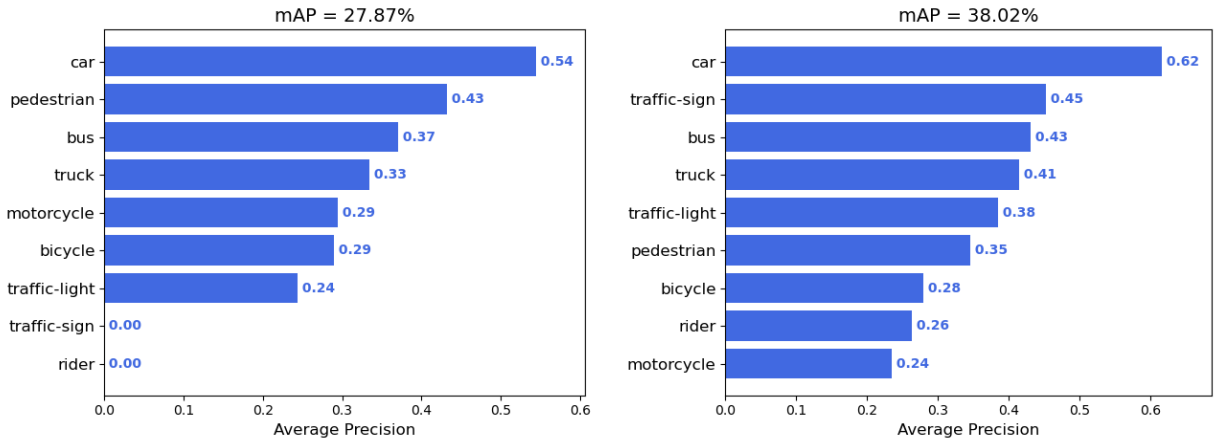


(a) YOLOv3

(b) YOLOv4

Figura 33: Resultados AP de YOLOv3 y YOLOv4 sobre BDD100k

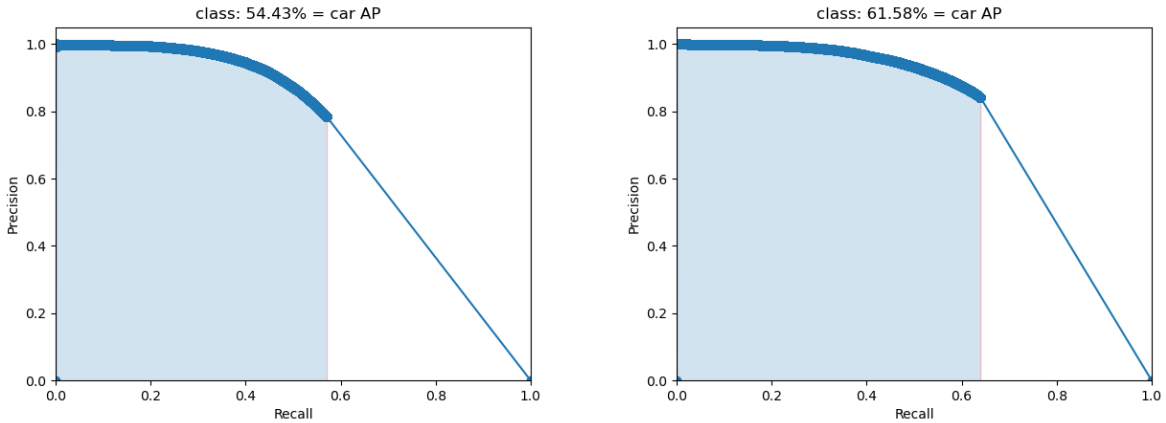
Sin embargo, mucho más interesante que la comparativa entre las dos versiones de YOLO es comparar YOLOv4 con YOLOv4 tras el reentrenamiento. Como se puede ver en la imagen 34 los resultados han mejorado en un $10,15\,\%$. No obstante, esta mejora se debe en gran medida en que la red reentrenada es capaz de detectar dos nuevas clases cuyo AP antes del reentrenamiento era 0.



(a) YOLOv4 (b) YOLOv4 reentrenada

Figura 34: Resultados AP de YOLOv4 y YOLOv4 reentrenada sobre BDD100k.

Hay que destacar la mejora en un 7 % para la clase *coche* (ver figura 35) y el notable empeoramiento de la clase *peatón* en un 8 %.



(a) YOLOv4 (b) YOLOv4 reentrenada

Figura 35: Resultados AP para la clase coche de YOLOv4 y YOLOv4 reentrenada sobre BDD100k.

En el cuadro 16 se detallan qué clases han mejorado, y cuales han empeorado su *AP*. Se puede observar que a excepción de la clase peatón, el resto de clases o bien han empeorado muy poco o bien han mejorado sustancialmente. Se puede concluir por tanto que la red reentrenada ofrece mejor precisión que la red original YOLOv4 cuando se trata de detectar objetos en imágenes orientadas a conducción autónoma como las del dataset BDD100k.

	$AP_{Reentrenada} - AP_{YOLOv4}$
Car	8 %
Pedestrian	-8 %
Bus	6 %
Truck	8 %
Motorcycle	-5 %
Bicycle	-1 %
Traffic-light	14 %
Traffic-sign	45 %
Rider	26 %
Promedio	10,3 %

Cuadro 5: Incremento entre el AP de cada clase para YOLOv4 y YOLOv4 reentrenada.

Las redes neuronales actuan como una caja negra y a menudo es complicado interpretar los resultados. La hipótesis es que la clase peatón ha disminuido su precisión debido al tamaño de los peatones en las imágenes (a menor tamaño más complicado de detectar para la red). Para comprobar esto se ha elaborado un script que se puede encontrar en el respositorio de Github [21] para calcular el tamaño promedio de cada clase en las imágenes del dataset BDD100k. Se han obtenido los siguientes resultados:

Clase	Tamaño promedio (en píxeles cuadrados)
Pedestrian	2879,52
Rider	5971,13
Car	9386,79
Truck	27412,59
Bus	33662,67
Motorcycle	7796,63
Bicycle	5413,33
Traffic-light	497,32
Traffic-sign	1193,22
Promedio	10468,13

Cuadro 6: Tamaños promedio de los objetos de cada clase

Si se observa el cuadro 6, se confirma que la clase peatón en promedio ocupa un área menor que la mayoría de las clases y mucho menor que el promedio total, lo cual puede explicar la disminución en su precisión. Por el contrario, la clase motocicleta que también redujo su precisión significativamente, tiene un tamaño similar al promedio, por lo que no se podría explicar esta reducción con la misma argumentación que en la clase peatón.

8.1.3. Precisión sobre CIU

El último dataset sobre el que se evaluarán las redes es el dataset de CIU. Este es un dataset reducido (ver figura 36) y que por tanto ofrecerá resultados poco fiables, no obstante, ofrece una perspectiva de cómo de bien funcionarían los algoritmos de detección de objetos que se están probando si fuesen integrados en la cámara un coche autónomo que circula por Ciudad Universitaria.

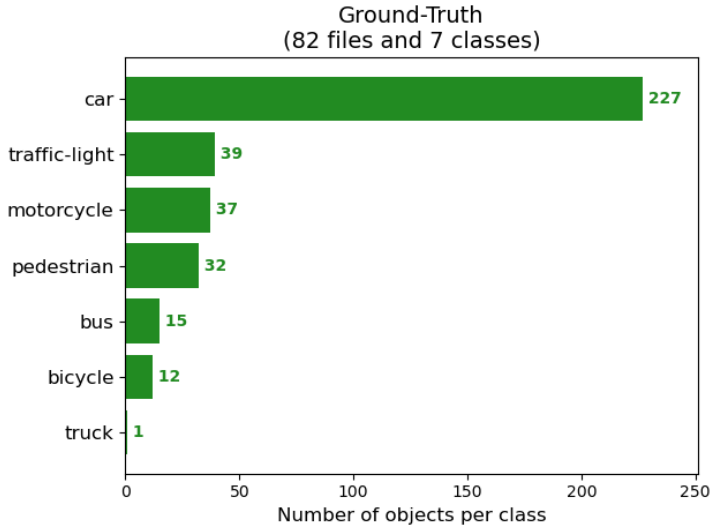
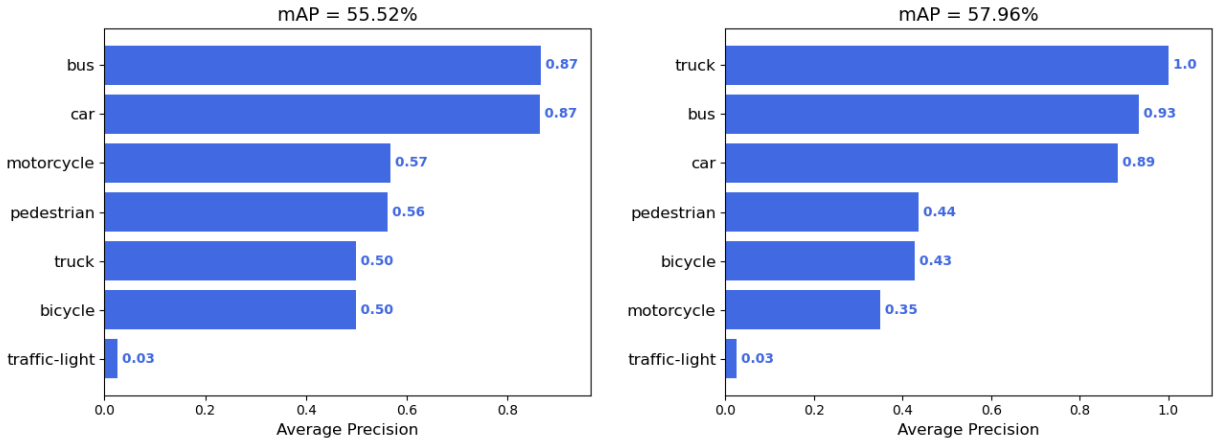


Figura 36: Número de instancias por clase en el dataset CIU

Los resultados de YOLOv3 y YOLOv4 son muy similares para este pequeño dataset y se pueden ver en el apéndice D). En cuanto a la red reentrenada se puede observar en la figura 37 como mejora ligeramente los resultados obtenidos por YOLOv4 sin reentrenar. En cualquier caso, la clase más interesante por ser la única con una representación significativa como para tener en cuenta los resultados es la clase coche, que como se puede ver en la figura 37 ha incrementado su AP en un 2 % tras el reentrenamiento.



(a) YOLOv4

(b) YOLOv4 reentrenada

Figura 37: Resultados AP de YOLOv4 y YOLOv4 reentrenada sobre el dataset de CIU.

8.1.4. Conclusiones sobre la evaluación de la precisión

Para obtener una conclusión definitiva sobre cual de las redes es la más precisa primero hay que hacer una reflexión sobre cual es el propósito de la red. Recordemos que en este trabajo se está diseñando una red neuronal que pueda ser integrada en la cámara de un coche autónomo por lo que debe de ser una red neuronal orientada a detectar imágenes en contextos de conducción. Por este motivo, aunque haya resultado de interés observar el comportamiento de la red reentrenada

sobre el conjunto MS COCO en el que originalmente está entrenada YOLOv4, no se tendrán en cuenta estos resultados ya que no se trata de imágenes grabadas desde un coche.

Por otra parte, tanto BDD100k como el dataset de CIU son imágenes grabadas desde un coche. No obstante, el dataset de CIU está formado únicamente por 82 imágenes y a excepción de la clase coche el resto de clases tienen menos de 40 instancias que las represente. Por esta razón tampoco se tendrán en cuenta los resultados obtenidos para el dataset de CIU.

Finalmente, queda el dataset BDD100k, el cual está formado por imágenes tomadas desde un vehículo y con un conjunto de validación de 10000 imágenes. Será por tanto BDD100k el conjunto cuyos resultados se tendrán en cuenta. En la sección 8.1.2 se ha visto que YOLOv3 ofrece claramente una peor precisión que YOLOv4 por lo que descartamos YOLOv3 como la red más precisa. En cuanto a *YOLOv4 reentrenada* no solamente ofrece una mejora en la precisión de la mayoría de las clases sino que es capaz de detectar dos clases más que YOLOv4 original por lo que se puede concluir que, aunque la mejora no es muy grande, la red más precisa a la hora de detectar objetos relacionados con la circulación vial es *YOLOv4 reentrenada*.

8.2. Evaluación de la velocidad de inferencia y el coste energético

El propóstio de esta sección es evaluar en "OpenVINO Deep Learning Workbench" (ver sección 7.2) el rendimiento en cuanto a velocidad de inferencia que ofrece YOLOv4 reentrenada. Esta evaluación se fijará en las siguientes métricas:

- **FPS:** Número de imágenes procesadas por segundo (ver sección 6.3).
- **Latencia:** Tiempo necesario para procesar una imagen.

OpenVINO Deep Learning Workbench pone a disposición de los desarrolladores distintos hardwares sobre los que desplegar los modelos. En este trabajo valoraremos la velocidad de inferencia sobre los siguientes hardwares:

- **Intel(R) Gen9 HD Graphics (iGPU):** Novena generación de los procesadores gráficos integrados fabricados y producidos por Intel.

Unidades de ejecución	24 unidades
Hilos hardware	168 hilos
Instancias concurrentes en el kernel	5376 instancias
Tamaño de la caché de nivel 3	512 Kb
Máxima memoria compartida localmente	192 Kb
Tamaño de la caché de último nivel	2-8 Mb
TDP (Potencia de diseño térmico)	15 W

Cuadro 7: Especificaciones Intel(R) Gen9 HD Graphics. [28]

- **Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz:** Microprocesador de la familia Intel(R) Xeon(R) Scalable Processors enfocada a servidores PC y Macintosh.

Cantidad de núcleos	14 núcleos
Cantidad de subprocesos	28 subprocesos
Frecuencia básica del procesador	2,2 GHz
Frecuencia turbo máxima	3,20 GHz
Tamaño caché de nivel 3	19,25 Mb
Cantidad de enlaces UPI	2
TDP	105 W

Cuadro 8: Especificaciones de Procesador Intel(R) Xeon(R) Oro 5120. [29]

- **Intel Movidius Myriad X VPU:** Unidad de la línea Movidius VPU Intel(R) Movidius™ que permiten cargas de trabajo exigentes de visión informática. Están diseñadas para lograr un equilibrio entre eficiencia energética y rendimiento informático.

Frecuencia básica del procesador	700 MHz
TDP	1 W

Cuadro 9: Especificaciones Intel(R) Movidius™ Myriad™ X Vision Processing Unit 0GB. [30]

Tras convertir el modelo a la representación "Intermediate Representation" (IR) con la que trabaja OpenVINO estos son los resultados obtenidos:

Target	Runtime Precisions	Best Throughput (FPS)	Respective Latency (ms)
GPU	FP32	2,7	740,05
CPU	FP32	9,54	3349,48
MYRIAD	FP16	6,32	1545,72

Cuadro 10: Resultados obtenidos para cada procesador tras la inferencia en OpenVINO.

En el cuadro 10 se puede observar como el procesador Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz es el que mejor velocidad de inferencia ha ofrecido. No obstante "Deep Learning Workbench" ofrece la posibilidad de optimizar los modelos mediante un algoritmo de calibración para obtener una velocidad mayor. Tras este proceso de calibración sobre el modelo para el procesador Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz se han obtenido los siguientes resultados:

Optimized	Target	Runtime Precisions	Best Throughput (FPS)	Respective Latency (ms)
No	CPU	FP32	9,54	3349,48
Yes	CPU	FP32, INT8	15,09	65,09

Cuadro 11: Resultados sobre el procesador Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz después del proceso de optimización del modelo.

Se puede ir un paso más allá: existen dos parámetros que se pueden modificar para mejorar la velocidad de inferencia en OpenVINO Deep Learning Workbench

- **Batch size:** Número de imágenes lanzadas a la vez a la nube para realizar la inferencia.
- **Stream:** Número de peticiones simultáneas al procesador.

Los anteriores resultados han sido obtenidos con la configuración $Batch = 1$ y $Stream = 1$. Si se varían estos parámetros para el procesador Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz y el modelo optimizado se obtiene el siguiente cuadro:

Stream	Batch size	Best Throughput (FPS)	Respective Latency (ms)
1	1	15,09	65,09
1	2	15,93	123,41
1	4	17,04	467,53
1	8	16,31	489,64
2	1	16,42	120,62
2	2	16,64	238,88
2	4	16,94	468,95
2	8	16,89	945,96
4	1	16,81	236,64
4	2	17,04	467,53
4	4	17,00	936,31
4	8	16,97	1874,54
8	1	16,42	120,62
8	2	15,0	65,33
8	4	14,54	2174,67
8	8	14,41	4285,10

Cuadro 12: Resultados sobre el procesador Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz después del proceso de optimización del modelo para distintos valores de "Stream" y "Batch size".

Las conclusiones que se pueden sacar del cuadro 12 son dos

- A mayor número de *streams* se obtienen mejores FPS (al menos hasta $stream = 4$).
- A mayor *batch size* la latencia aumenta muy rápidamente, especialmente para valores altos de *stream*. Este aumento en la latencia no es deseable.

El cuadro 12 queda recogido en el gráfico de la figura 38 donde se puede ver que los valores para los que se obtiene un mejor compromiso entre FPS y latencia son $stream = 4$ y $batchsize = 2$.

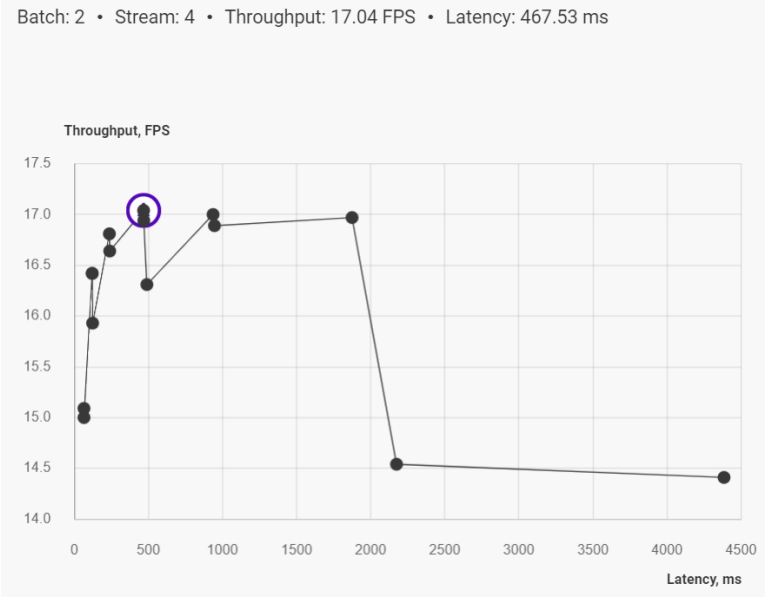


Figura 38: Datos del cuadro 12 recogidos en una gráfica generada por DL Workbench.

Los resultados para la velocidad de inferencia en Deep Learning Workbench son más bajos de lo esperado (especialmente en el caso de la CPU y Myriad). Además Deep Learning Workbench es un entorno de caja negra en el que no se pueden ver qué scripts están siendo ejecutados, por lo que para constatar que los resultados son fiables se ha realizado la inferencia de forma local en mi propio ordenador. Se han utilizado dos dispositivos hardware de Intel par inferir con OpenVINO:

- **GPU:** Intel(R) UHD Graphics for 8th Generation Intel(R) Processors
- **CPU:** Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz

La ejecución local de los scripts de OpenVINO es extremadamente enrevesada por lo que lo mejor que se ha podido hacer es inferir la red reentrenada YOLOv4 sobre un mismo vídeo en formato *.mp4* y comprobar los FPS del vídeo tras la inferencia (ver figura 13).

Para mayor seguridad de que los resultados son fiables también se ha realizado la inferencia de manera local en Darknet (sin usar OpenVINO) sobre la siguiente GPU:

- **GPU:** NVIDIA GeForce GTX 1050 with Max-Q Design

Los resultados obtenidos se recogen en el cuadro 13.

	Velocidad de inferencia
Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz	3,15 FPS
Intel(R) UHD Graphics for 8th Generation Intel(R) Processors	4,40 FPS
Nvidia GeForce GTX 1050 with Max-Q Design	4,80 FPS

Cuadro 13: Resultados locales de la inferencia en OpenVINO

(a) CPU (b) GPU

Figura 39: Resultados inferencia de un vídeo mp4 sobre hardwares locales.

Los resultados nuevamente vuelven a ser más bajos de lo esperado, pero son consistentes con los obtenidos en la nube de Deep Learning Workbench. Tras este pequeño experimento se tiene la certeza de que los resultados de Deep Learning Workbench aunque no sean los esperados, son fiables.

Tras evaluar la velocidad de inferencia y optimizarla lo máximo posible utilizando las herramientas de DL Workbench resulta interesante evaluar el coste energético de la inferencia. El coste energético es relevante ya que un coche autónomo no dispone de un suministro constante de energía por lo que un hardware que consumiese demasiada energía, además de generar más gastos que uno que consumiese menos, podría llegar a no ser viable.

Para calcular el gasto energéticos nos fijaremos en el valor de TDP (Potencia de diseño térmico). El TDP representa la energía promedio en watts que el procesador disipa cuando opera en una frecuencia básica con todos los núcleos activos, en una exigencia de alta complejidad [33].

Teniendo en cuenta tanto el TDP como las velocidades de inferencia calculadas para cada unidad de procesamiento se obtiene el siguiente cuadro en el que también se ha incluido el precio de mercado para tener una visión más completa

Unidad hardware	Tipo	Velocidad	TDP	FPS/W	Precio
Intel(R) Xeon(R) Gold 5120	CPU	17,04 FPS	105 W	0,162	4200€
Intel(R) Movidius™ Myriad™ X	MYRIAD	6,32 FPS	1 W	6,320	90€
Intel(R) Gen9 HD Graphics	GPU	2,7 FPS	15 W	0,180	400€

Cuadro 14: Comparativa entre velocidad y coste energético de las unidades hardware.

Se puede observar que la CPU Intel(R) Xeon(R) Gold 5120 a pesar de ser la unidad que mejor

velocidad de inferencia ofrece su coste energético es tan elevado que su ratio FPS por Watio es el menor de las 3 además de ser con diferencia el hardware más caro. Hay que remarcar que Intel(R) Movidius™ Myriad™ X es una unidad enfocada a la visión por ordenador e inteligencia artificial diseñada para tener un consumo muy bajo de energía y por ello es la que mejor resultado ofrece en el ratio FPS por Watio muy por encima de Intel(R) Xeon(R) Gold 5120.

9. Conclusiones

El propósito de esta sección es emplear los resultados obtenidos para dar respuesta a los objetivos que se propusieron para este trabajo. Para hacerlo de manera ordenada se verá objetivo por objetivo cuales son las conclusiones pertinentes:

- **Evaluación de la precisión de YOLOv3 y YOLOv4 sobre varios datasets**
  Para cumplir con este objetivo se han tomado tres datasets:

  - MS COCO: El dataset sobre el que YOLOv3 y YOLOv4 fueron entrenadas.
  - BDD100k: Un dataset a gran escala orientado a conducción autónoma.
  - CIU: Un dataset reducido de elaboración propia con imágenes tomadas desde un coche en Ciudad Universitaria.

  La evaluación de YOLOv3 y YOLOv4, como se vió en la sección 8.1, se puede considerar exitosa ya que se han obtenido resultados similares a los publicados por los propios desarrolladores de YOLO. En cuanto a la comparativa entre estas dos redes, en los tres datasets sobre los que fueron evaludas YOLOv4 obtuvo una precisión ligeramente mayor que YOLOv3 (ver anexos B, C y D), algo lógico y esperable dado que YOLOv4 es una red que implementa varias mejoras sobre su predecesora YOLOv3.

- **Reentrenamiento y evaluación de YOLOv4 para convertirla en una red orientada a conducción autónoma**
  A pesar de que YOLOv4 es una de las redes más punteras de visión artificial, ha sido entrenada sobre MS COCO que es un dataset generalista por lo que no se puede considerar una red entrenada para conducción autónoma. En este punto nos planteamos si es posible reentrenar YOLOv4 para obtener mejores precisiones para objetos relevantes para un vehículo autónomo como semáforos, peatones, coches o camiones.

  Este reentrenamiento ha sido llevado a cabo en Darknet [19] utilizando las configuraciones especificadas en el anexo A. Para el reentrenamiento se utilizó el conjunto de entrenamiento de BDD100k formado por 70000 imágenes tomadas desde un vehículo en las que estaban etiquetadas 9 clases de objetos distintos

  - Coche
  - Peatón
  - Autobús
  - Camión
  - Motocicleta
  - Bicicleta
  - Semáforo
  - Señal
  - Ciclista

  La evaluación de YOLOv4 reentrenada fue hecha sobre los mismos datasets que YOLOv3 y YOLOv4 para poder obtener una comparativa fideligna.

- **Comparación de los resultados de precisión de las tres redes**
  Para comparar la precisión de las redes se ha utilizado como métrica la mAP (ver sección 6). Los resultados obtenidos han sido ligeramente distintos para cada dataset pero se ha tomado como resultados de referencia los del dataset BDD100k por ser el único de los tres con una gran cantidad de imágenes y que estas sean orientadas a conducción autónoma.

  Como se puede ver en las gráficas del anexo C, YOLOv4 reentrenada es capaz de reconocer dos nuevas clases de objetos viales y además mejora la mayoría de las precisiones de YOLOv4.

	$AP_{Reentrenada} - AP_{YOLOv4}$
Car	8 %
Pedestrian	-8 %
Bus	6 %
Truck	8 %
Motorcycle	-5 %
Bicycle	-1 %
Traffic-light	14 %
Traffic-sign	45 %
Rider	26 %
Promedio	10,3 %

Cuadro 15: Incremento entre el AP de cada clase para YOLOv4 y YOLOv4 reentrenada sobre el dataset BDD100k.

  Por este motivo se concluye que YOLOv4 reentrenada mejora la precisión de YOLOv4 y por tanto, es de las tres redes la más apropiada para integrar en un coche autónomo.

- **Evaluación de la velocidad de inferencia y el coste energético**
  La evaluación de la velocidad de inferencia se ha realizado en la herramienta OpenVINO DL Workbench (ver sección 7.2). Se ha comparado la velocidad en tres unidades hardware distintas:

  - **Intel(R) Gen9 HD Graphics:** Novena generación de los procesadores gráficos integrados fabricados por Intel.
  - **Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz:** Microporcesador de la familia Intel(R) Xeon(R) Scalable Processors.
  - **Intel Movidius Myriad X VPU:** Unidad de la línea Movidius VPU de Intel diseñadas para lograr un equilibrio entre eficiencia energética y rendimiento informático.

  La mejor velocidad de inferencia (medida en FPS) ha sido obtenida por la unidad CPU Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz. Esta velocidad en un principio era de 9,54 FPS pero tras optimizarla con las herramientas que provee OpenVINO se ha conseguido aumentar a 17,04 FPS.

  No obstante, se ha evaluado también el coste energético de estas mismas unidades atendiendo al valor TDP (Potencia de diseño térmico) proporcionado por Intel en Watios (W). Tras esta evaluación se ha calculado el ratio FPS/W de cada unidad y la que mejor compromiso ha ofrecido entre velocidad y eficiencia ha sido Intel Movidius Myriad X con 6,320 FPS/W muy por encima del resto de dispositivos.

Hay que remarcar que los resultados para la velocidad de inferencia en general han sido más bajos de lo esperado. Esto no se debe a un error en la metodología ya que se ha llevado a cabo la evaluación de la velocidad de inferencia en la nube de Intel, localmente en OpenVINO y localmente sobre una GPU Nvidia. La conclusión es que YOLOv4 es una red muy grande como para ofrecer una velocidad de inferencia alta en hardwares de uso cotidiano. Existen versiones con menos capas como *YOLOv4 tiny* que quizás puedan mejorar estos resultados.

- **Conclusión sobre qué red y hardware serían los más apropiados para integrar en un coche autónomo**
  El objetivo último de este trabajo era determinar que red y qué hardware de los evaluados serían los más apropiados para integrar en un coche autónomo, por lo que este punto no es sino un resumen del resto de conclusiones

  - La mejor red atendiendo a su precisiónde las tres evaluadas es YOLOv4 reentrenada.
  - El mejor hardware en cuanto a velocidad de inferencia es la CPU Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz. Sin embargo, si se busca un compromiso entre eficiencia y velocidad el mejor hardware es Intel Movidius Myriad X VPU.

9.1. Posibles continuaciones para el proyecto

Las redes orientadas a visión artificial es un campo en el que queda mucho por explorar y que está en constante evolución. En Abril de este mismo año ha salido la red YOLOv5 destinada a ser la sucesora de YOLOv4 y que implementa algunas mejoras respecto de esta. Por otra parte, el reentrenamiento de YOLOv4 para este trabajo fue muy costoso en cuanto a tiempo (aproximadamente 9 días) por lo que un solo fallo en el reentrenamiento provocaba un retraso importante en el desarrollo del trabajo. Con un procesador más potente que el que se usó (Nvidia GeForce GTX 1050) como por ejemplo una unidad gráfica de alta gama podría recortarse este tiempo y podrían probarse distintas configuraciones para el reentrenamiento con el fin de obtener una red aún más precisa.

Por otro lado, una línea a explorar son nuevas optimizaciones para la velocidad de inferencia. En este trabajo se ha empleado la optimización ofrecida por OpenVINO, pero podrían hacerse nuevos experimentos como inferir en varios procesadores en paralelo.

Finalmente, una última línea sería explorar las posibilidades que ofrece OpenVINO para transformar la red en formato IR a una aplicación que puede ser descargada y empleada por un usuario.

10. Conclusions

Last section purpose is to summarize all the results obtained and determine if the goals that were set at the beginning of this project have been achieved. To do so in an orderly way, we will have a look to each objective relevant conclusions:

- **Evaluation of YOLOv3 and YOLOv4 precision over several data sets.**
  To achieve this goal three data sets have been used:

  - MS COCO: This is the data set where YOLOv3 and YOLOv4 were originally trained.
  - BDD100k: This is a large scale dataset oriented to autonomous driving.
  - CIU: This is a small self-made dataset with Ciudad Universitaria photos taken from a car.

  As seen in section 8.1, YOLOv3 and YOLOv4 evaluations can be considered successful because we have obtained similar results to the ones provided by YOLO developers. Regarding the comparison between these two networks, in the three data sets YOLOv4 achieved a slightly better precision than YOLOv3 (see appendix B, C and D). This is quite logic as YOLOv4 implements several improvements over its predecessor YOLOv3.

- **Retraining and evaluation of YOLOv4 in order to convert it into an autonomous-driving-oriented network**
  In spite of being a leading edge neural network, YOLOv4 has been trained over MS COCO, a generalist data set, therefore it cannot be considered an autonomous-driving-oriented network. At this point, it is questionable if its possible to retrain YOLOv4 so that it gets better precision results for relevant objects in autonomous driving such as pedestrians, traffic lights, cars or trucks.

  This retraining has been made in Darknet [19]. We have used the configurations detailed in appendix A. To perform retraining, 70000 images from BDD100k training set have been used. This images have been taken from a vehicle and 9 distinct object classes are labeled in them:

  - Car
  - Pedestrian
  - Bus
  - Truck
  - Motorcycle
  - Bicycle
  - Traffic light
  - Traffic sign
  - Rider

  Retrained YOLOv4 evaluation has been performed over the same data sets than YOLOv3 and YOLOv4. In this way we can get a reliable comparison between the three networks.

- **Precision results comparison among the three networks**
  To compare the networks precision we have used mAP measure (see section 6). The results have been slightly different for each data set but BDD100k results are the ones that have been taken as reference. The reason for this is that BDD100k is the only large scale data set which images are autonomous-driving-oriented.

  As it can be seen in appendix C figures, retrained YOLOv4 is not only capable to recognize two more classes of objects, it also improves most of the YOLOv4 precision results.

	$AP_{Reentrenada} - AP_{YOLOv4}$
Car	8 %
Pedestrian	-8 %
Bus	6 %
Truck	8 %
Motorcycle	-5 %
Bicycle	-1 %
Traffic-light	14 %
Traffic-sign	45 %
Rider	26 %
Average	10,3 %

Cuadro 16: YOLOv4 and retrained YOLOv4 AP difference for each class of BDD100k data set.

  Putting it into a nutshell, retrained YOLOv4 improves the precision of YOLOv4 therefore it is, among the three networks evaluated, the most appropiate for autonomous driving.

- **Inference speed and energy cost evaluation**
  Inference speed evaluation was performed in OpenVINO DL Workbench tool (see section 7.2). The inference speed of three different hard ware units has been compared:

  - **Intel(R) Gen9 HD Graphics:** Ninth generation of Intel integrated graphic processors.
  - **Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz:** Intel(R) Xeon(R) Scalable Processors family microprocessor.
  - **Intel Movidius Myriad X VPU:** Intel Movidius VPU unit designed to achieve balance between energetic efficiency and computing performance.

  The best inference speed (in FPS) was achieved by CPU Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz. This speed was initially 9,54 FPS, but after optimizing the model using OpenVINO toolkit it has finally reached 17,04 FPS.

  However, energy cost of these three hard wares has been evaluated taking into account TDP value (thermic design power) provided by Intel and measured in Watios (W). After this evaluation, the ratio FPS/W was computed for each hardware. The best balance between energy cost and speed was achieved by Intel Movidius Myriad X and it was 6,320 FPS/W, much higher than the achieved by the other hard wares.

- **Final conclusion on which network and hard ware should be integrated in an autonomous vehicle**
  The final goal of this project was deciding which network and hard ware were the most appropiate for an autonomous vehicle. For this reason at this point we have to sum up all the conclusions:

  - In respect of precision, retrained YOLOv4 is the best network among the three evaluated.
  - The hard ware which achieved the best inference speed is CPU Intel(R) Xeon(R) Gold 5120 CPU @2.20GHz. However, if we are looking for a balance between efficiency and speed, the best hard ware is Intel Movidius Myriad X VPU.

10.1. Possible continuations for the project

Neural networks aplied to computer vision is a field in continuous evolution with many paths to explore. Last April YOLOv5 was introduced and its aim is to improve YOLOv4, so a path to explore is to experiment with YOLOv5. On the other hand, YOLOv4 retraining was very time consuming (approximately each try took 9 days) therefore a single mistake caused a big delay in the project. With a more powerful processor than the one that was used for retraining (Nvidia GeForce GTX 1050), such as a powerful graphic processor, retraining time can be reduced and much more different retraining configurations can be tried in order to get a more precise retrained network.

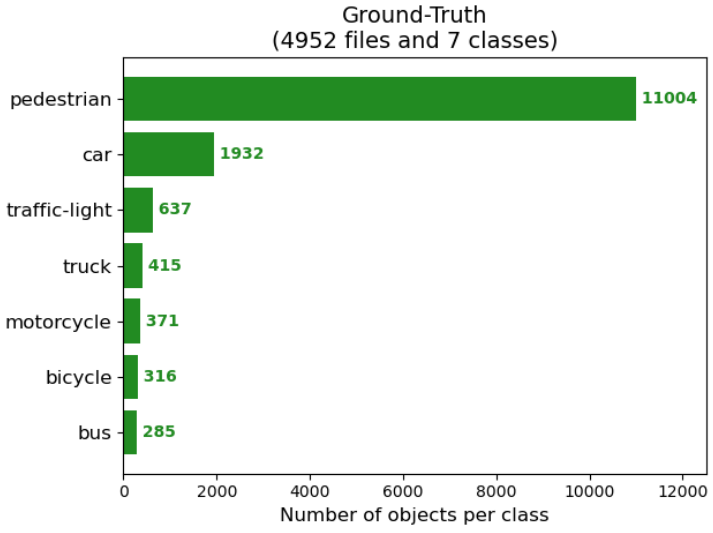
Another path that could be explored are new optimizations for inference speed. In this project only the optimizations provided by OpenVINO toolkit were used, however, new experiments such as running parallel inference could be developed to achieve better speed.

Finally, a last path that could be explored is how to convert OpenVINO's IR network into an application that can be downloaded and deployed by a user.

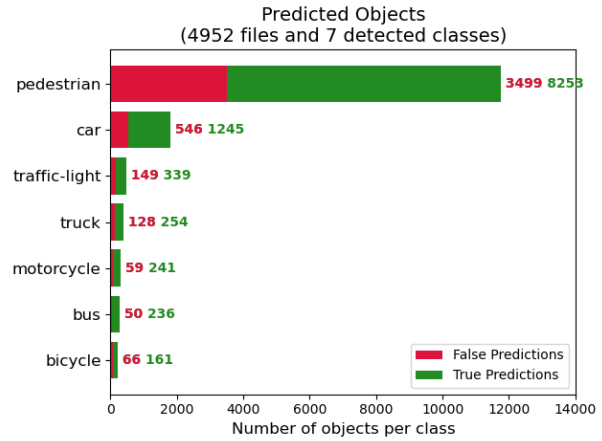
A. Detalles configuración de YOLOv4 para el reentrenamiento

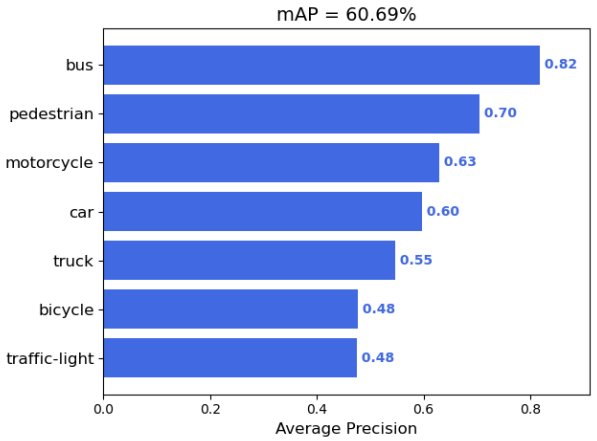
#Training	
batch	64
subdivisions	64
width	416
height	416
channes	3
momentum	0,949
decay	0,0005
angle	0
saturation	1,5
exposure	1,5
hue	.1
learning rate	0,001
burn in	1000
max_batches	20000
policy	steps
steps	16000,18000
scales	.1,.1

B. Resultados completos MS COCO

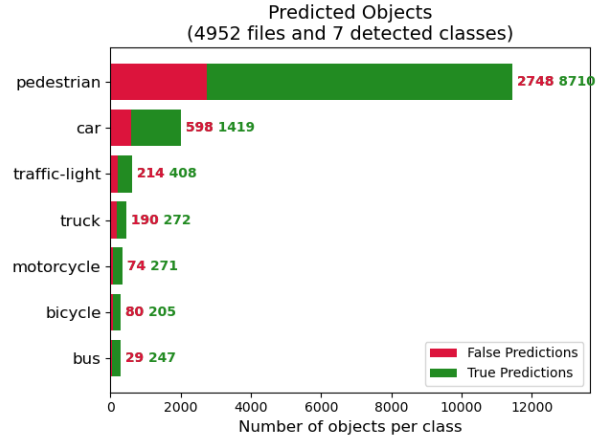


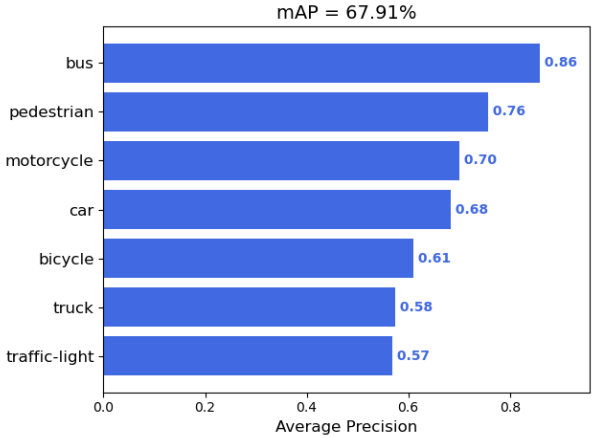
B.1. YOLOv3



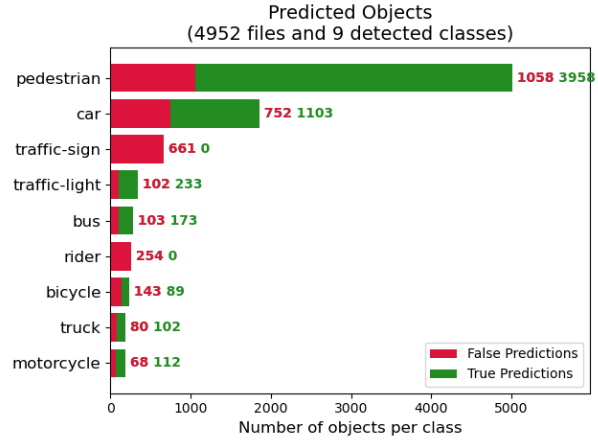


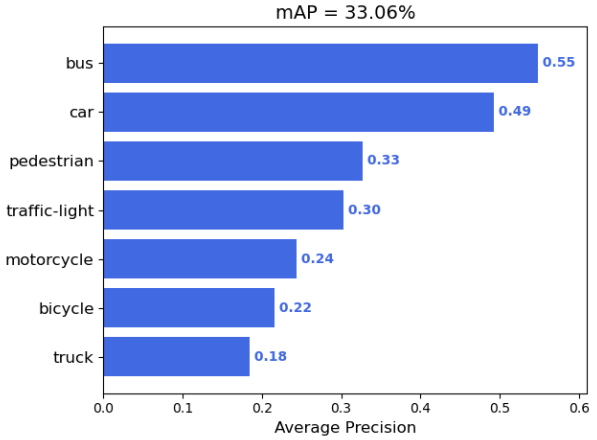
B.2. YOLOv4



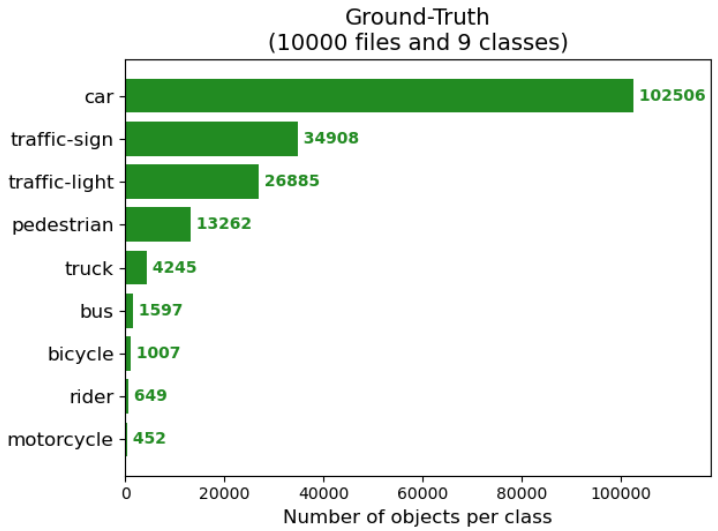


B.3. YOLOv4 reentrenada

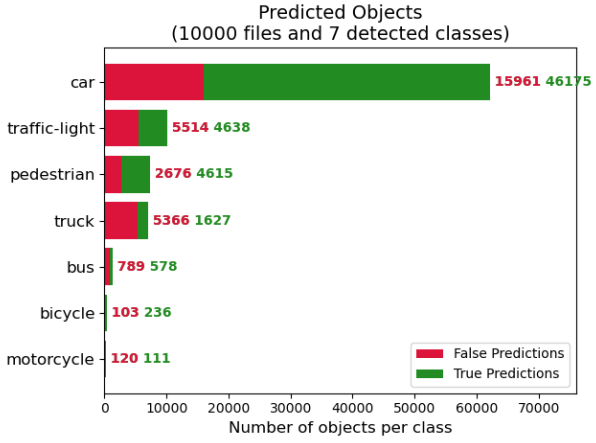


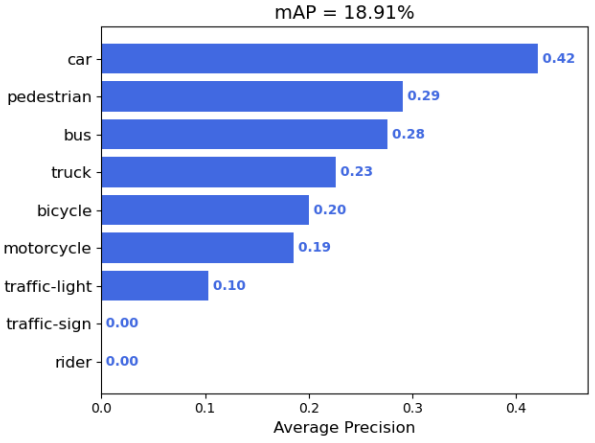


C. Resultados completos BDD100k

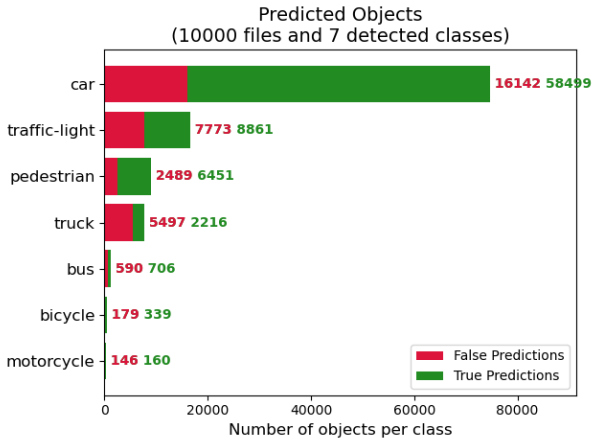


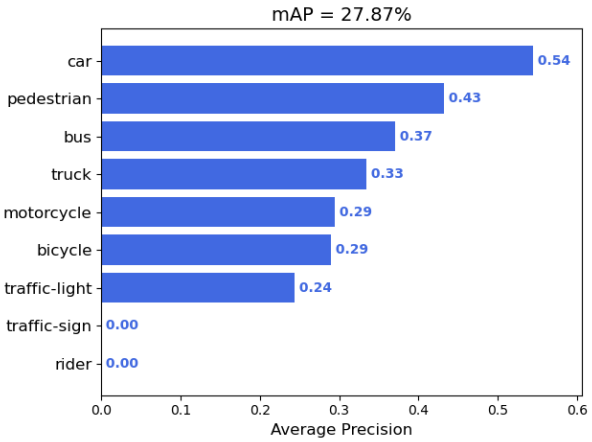
C.1. YOLOv3



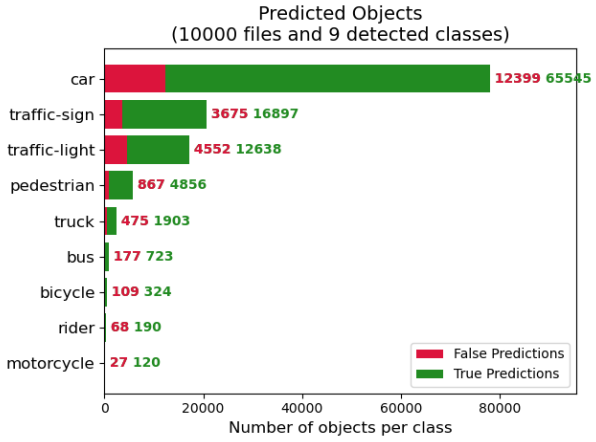


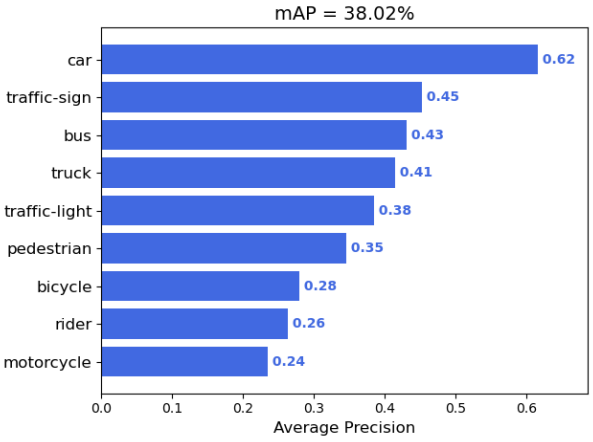
C.2. YOLOv4



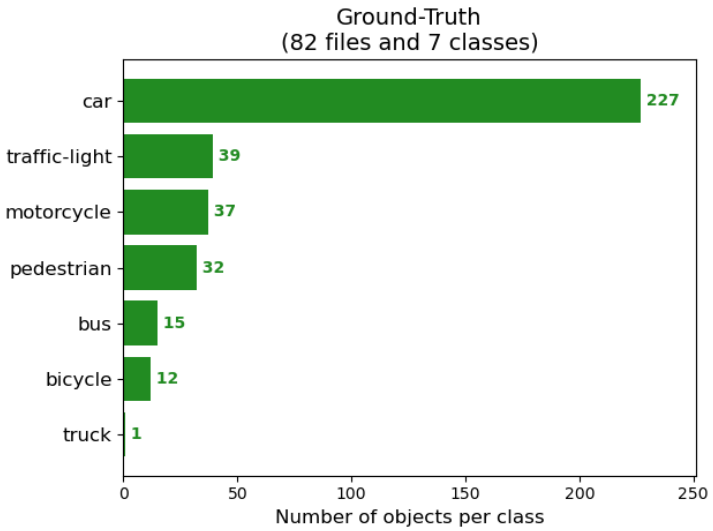


C.3. YOLOv4 reentrenada

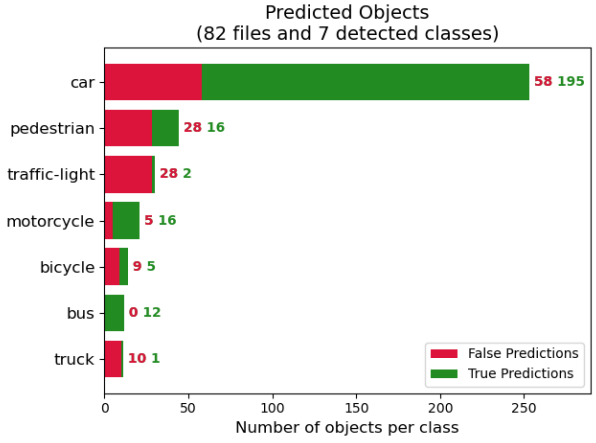


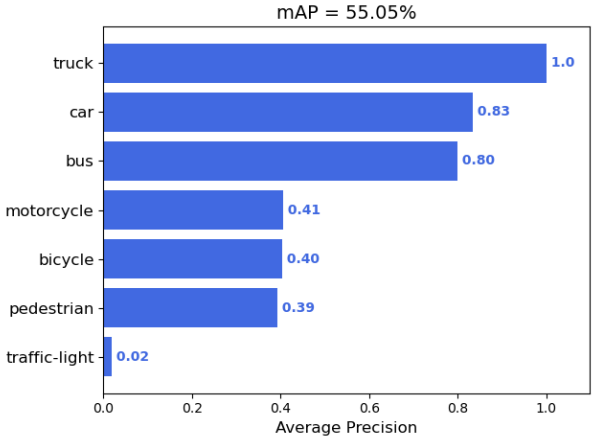


D. Resultados completos CIU

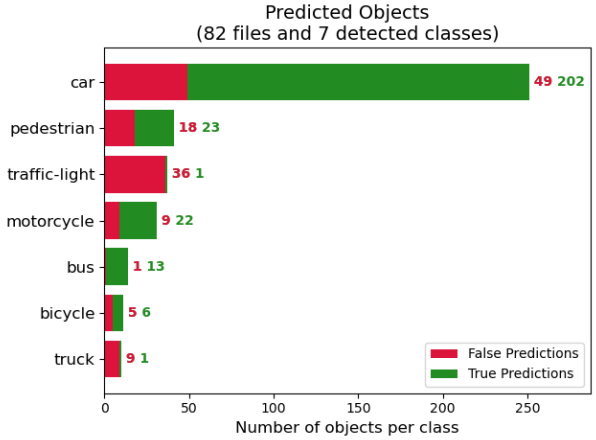


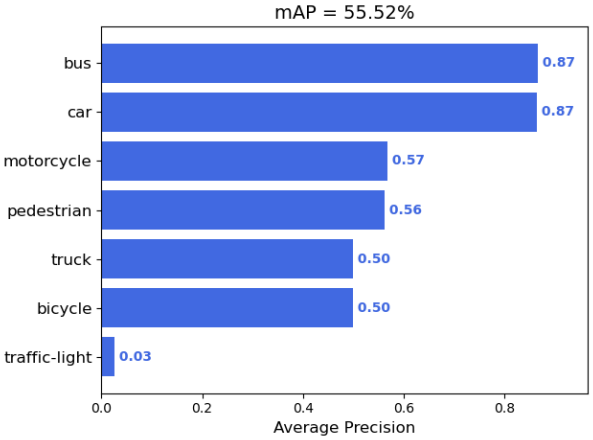
D.1. YOLOv3



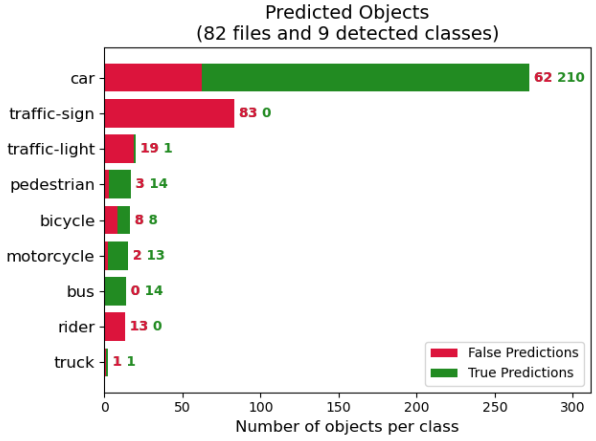


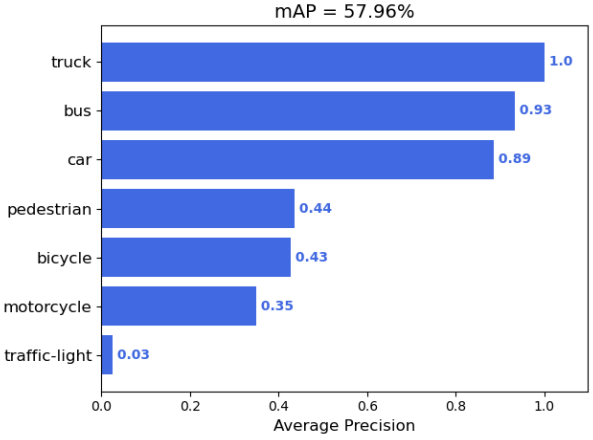
D.2. YOLOv4





D.3. YOLOv4 reentrenada





Referencias

[1] Wikipedia, *Vehículo autónomo*, https://es.wikipedia.org/wiki/Veh%C3%ADculo_aut%C3%B3nomo, s.f..

[2] C.Szegedy, A.Toshev y D.Erhan, *Deep Neural Networks for Object Detection*, Google, Inc. s.f..

[3] 3Blue1BrownEspañol, *¿Qué es una Red Neuronal? — Aprendizaje Profundo. Capítulo 1*[Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=jKCQsndqEGQ&ab_channel=3Blue1BrownEspa%C3%B1ol, 2020.

[4] Na8, *¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador*, Aprende Machine Learning, https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/, 2018.

[5] Andrej Anka, *YOLO v4: Optimal Speed & Accuracy for object detection*, Towards data science, https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50, 2020.

[6] Na8, *Modelos de Detección de Objetos*, Aprende Machine Learning, https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/, 2021.

[7] Rohith Gandhi, *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*, Towards data science, https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e, 2018.

[8] datahacker.rs, *#029 CNN Yolo Algorithm*, http://datahacker.rs/object-detection-yolo-algorithm/, 2018.

[9] DeepLearningAI, *C4W3L09 YOLO Algorithm*[Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=9s_FpMpdYW8&ab_channel=DeepLearningAI, 2018.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick y Piotr Dollar *Microsoft COCO: Common Objects in Context*, https://cocodataset.org/#home, 2015.

[11] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan y Trevor Darrell *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*, https://www.bdd100k.com/, 2018.

[12] J. Ramírez *Curvas PR y ROC*, https://medium.com/bluekiri/curvas-pr-y-roc-1489fbd9a527, 2018.

[13] A. Rosebrock *Intersection over Union (IoU) for object detection*, https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, 2016.

[14] S. Yohanandan *mAP (mean Average Precision) might confuse you!*, https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2., 2020.

[15] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. *CSPNet: A new backbone that can enhance learning capability of cnn. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop)*, 2020. 2, 7

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(9):1904–1916, 2015. 2, 4, 7

[17] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. *Path aggregation network for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pages 8759–8768, 2018. 1, 2, 7

[18] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao *YOLOv4: Optimal Speed and Accuracy of Object Detection*, 2020.

[19] Alexey Bochkovskiy *darknet* https://github.com/AlexeyAB/darknet, 2021.

[20] hunglc007 *tensorflow-yolov4-tflite* https://github.com/hunglc007/tensorflow-yolov4-tflite, 2021.

[21] Sergio Gil Gavela *annotations-and-mAP-sripts* https://github.com/sgavela/annotations-and-mAP-sripts, 2021.

[22] s.n. *YOLO: Real-Time Object Detection* https://pjreddie.com/darknet/yolo/, s.f.

[23] Joseph Redmon, Ali Farhadi *YOLOv3: An Incremental Improvement*, https://arxiv.org/abs/1804.02767, 2018.

[24] Ayoosh Kathuria *What's new in YOLO v3?* https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, 2018.

[25] Yúbal Fernández *Qué son los FPS o fotogramas por segundo, y para qué sirven en los videojuegos*, https://www.xataka.com/basics/que-fps-fotogramas-segundo-sirven-videojuegos, 2020.

[26] Josep Roca *¿Qué es la eficiencia energética (rendimiento por vatio) en una tarjeta gráfica?*, https://hardzone.es/reportajes/que-es/rendimiento-vatio-gpu/, 2020.

[27] Intel *OpenVINO™ Toolkit Overview* https://docs.openvinotoolkit.org/latest/index.html

[28] Intel *The Compute Architecture of Intel(R) Porcessor Graphics Gen9*, https://software.intel.com/content/dam/develop/external/us/en/documents/the-compute-architecture-of-intel-processor-graphics-gen9-v1d0.pdf, s.f..

[29] Intel *Procesador Intel® Xeon® Oro 5120*, https://ark.intel.com/content/www/es/es/ark/products/120474/intel-xeon-gold-5120-processor-19-25m-cache-2-20-ghz.html, s.f..

[30] Intel Movidius *Intel® Movidius™ Myriad™ X Vision Processing Unit 0GB*, https://www.intel.com/content/www/us/en/products/sku/204770/intel-movidius-myriad-x-vision-processing-unit-0gb/ordering.html, s.f..

[31] TntWen *OpenVINO-YOLOv4*, https://github.com/TNTWEN/OpenVINO-YOLOV4

[32] TheCodingBug *YOLOv4 Custom Object Detection Tutorial: Part 1 (Preparing Darknet YOLOv4 Custom Dataset)*, https://www.youtube.com/watch?v=sKDysNtnhJ4&ab_channel=TheCodingBug.

[33] Intel, *Cómo saber si el disipador térmico es compatible con el procesador Intel®*, https://www.intel.es/content/www/es/es/support/articles/000030760/processors/intel-core-processors.html.