Universidad Complutense de Madrid/Universidad de Granada (SICUE)

Facultad de Informática

TRABAJO DE FIN DE GRADO

EVALUACIÓN DE ALGORITMOS DE MACHINE LEARNING PARA CONDUCCIÓN

MACHINE LEARNING ALGORITHM EVALUATION ON ADVANCED DRIVER ASSISTANCE

Alumnos

Alejandro Simón Rodríguez

Rafael Ruz Gómez

Directores

Carlos García Sánchez

Guillermo Botella Juan

Doble Grado Ingeniería Informática y Matemáticas

2019/2020

Índice

Índice de tablas

PALABRAS CLAVE:

Visión por ordenador, conducción autónoma, red neuronal convolucional, detección de objetos, aprendizaje profundo, aprendizaje automático, Myriad, Tensorflow, Openvino, Darknet.

RESUMEN

El principal objetivo de este proyecto de investigación y desarrollo es el análisis e implementación de arquitecturas *deep learning* para la detección en tiempo real de peatones, ciclistas y vehículos en el ámbito de la conducción autónoma.

El vertiginoso crecimiento de la capacidad de procesamiento ha provocado el surgimiento de nuevas ramas de investigación tecnológica, entre ellas la de la Inteligencia Artificial (IA). En particular, dentro de la IA podemos destacar la aparición de las redes neuronales profundas como técnica para abordar problemas relacionados con la percepción visual, como puede ser la clasificación de objetos o la detección de objetos.

Uno de los campos donde la IA que está teniendo una mayor influencia es la creación de sistemas avanzados de asistencia para la conducción (ADAS). Estos sistemas de conducción se apoyan en una amplia variedad de cámaras y sensores que proporcionan toda la información necesaria para tomar decisiones con precisión y seguridad.

Un sistema ADAS está compuesto por distintos módulos. Uno de ellos se encarga de la detección en tiempo real de objetos. En este proyecto nos centramos en la detección en tiempo real de vehículos, ciclistas y peatones, aunque podrían incluirse líneas de carretera y señales de tráfico, entre otros objetos.

Para conseguir dicho objetivo dividiremos en dos partes diferenciadas el trabajo:

- Reentrenamiento de una red generalista para aprender a identificar vehículos, peatones y ciclistas.
- Estudio del rendimiento de la inferencia realizada por la red reentrenada en términos de precisión, velocidad de inferencia y consumo sobre un conjunto de dispositivos hardware.

KEYWORDS:

Computer vision, autonomous driving, convolutional neuronal network, object detection, deep learning, machine learning, Myriad, Tensorflow, Openvino, Darknet.

ABSTRACT

The main aim of this research and development project is the analysis and implementation of deep learning's architectures for real-time object detection of pedestrians, cyclists and cars on the scope of autonomous driving.

The breakneck growth of the computing capacity has caused the emergence of new technological research fields such as Artificial Intelligence (AI). Especifically, we can highlight the uncovering of deep neural networks as a technique to approach challenges related to visual perception, for example object classification or object detection.

One of the areas where the artificial intelligence is having a great influence is on the creation of advanced driver-assistance systems (ADAS). These systems take advantage of cutting-edge cameras and sensors that feed the systems with all the necessary information to take decisions with accuracy and security.

An advanced driver-assistance system is made of several modules. One of them perform object detection in real-time. The main focus of this project is real-time detection of pedestrians, cyclists and cars.

To achieve this objective, we will divide the work into two parts:

- Fine-tuning of a generalist network to learn how to identify pedestrians, cyclists and cars.
- Perfomance analysis in terms of efficiency, speed and power consumption during the inference performed by the fine-tuned network over a set of hardware devices.

1. Introducción y motivación

El desarrollo de sistemas avanzados de asistencia al conductor (ADAS) es una realidad a día de hoy y no un sueño futurista. Actualmente es una tecnología cuyo desarrollo se ha divido en fases con el objetivo de ir cumpliendo objetivos progresivamente en pro de lograr un sistema de conducción *completamente* autónomo.

La *Sociedad de Ingenieros de Automoción* [53] establece una diferenciación en cuanto al nivel de autonomía en la conducción en la que se distinguen seis categorías o niveles:

- Nivel 0: Ninguna automatización, el conductor debe realizar todas las tareas relativas a la conducción.
- Nivel 1: Asistencia en la conducción, se cuenta con algún sistema automatizado, como por ejemplo para controlar el movimiento lateral o longitudinal, pero no ambos a la vez, y no se dispone de sistemas para responder ante objetos. Debiendo realizar el resto de las tareas el conductor.
- Nivel 2: Automatización parcial, aparecen sistemas para tanto el control lateral como longitudinal, eximiendo al conductor de controlar el movimiento, pero no del resto de tareas como la detección de objetos y respuesta ante ellos.
- Nivel 3: Automatización condicional, se controla automáticamente tanto el movimiento como se monitoriza el entorno, pero se requiere al conductor disponible en segundos por si ocurriera algún fallo, saltando una alerta.
- Nivel 4: Automatización elevada, se tiene un sistema autónomo, en casi cualquier situación, controlando el movimiento, detectando objetos y actuando ante fallos, teniendo que tomar el control el conductor en rara ocasión.
- Nivel 5: Automatización completa, se cuenta con sistemas autónomos ante cualquier situación, y el conductor nunca está obligado a tomar control manualmente si no lo desea.

Actualmente, nos encontramos a medio camino entre el tercer y el cuarto nivel. Se requiere de una tecnología mas avanzada conforme aumentan los niveles, siendo tecnologías imprescindibles la sensorización, el procesado de las señales de estos y la toma de decisiones basándose en la información de los sensores procesada. Una parte de este trabajo se enmarcará en aportar soluciones dentro de una de estas tareas, la detección de objetos con técnicas de Inteligencia Artificial.

Otro aspecto a tener en cuenta para el despliegue de la conducción autónoma es la energía que deben consumir todos los sistemas de detección, procesamiento y toma de decisiones en función de la información recabada. Por ello, otra parte de este proyecto se dedicará a analizar la eficiencia energética de los sistemas utilizados.

Podemos resumir la motivación de este trabajo en dos cuestiones abiertas:

- ¿Cómo procesar toda la información recabada por el vehículo de forma que permita tomar decisiones en tiempo real?
- ¿Cómo crear un Sistema de Control Autónomo óptimo en términos de eficiencia energética?

Ambas cuestiones son de vital importancia para el desarrollo de la tecnología necesaria para el coche autónomo. Para responder a la primera cuestión nos apoyaremos en el área de la visión por computador, donde los mayores éxitos en cuanto a rendimiento han llegado con el crecimiento exponencial de las redes neuronales profundas, en concreto, con la evolución de las redes neuronales convolucionales. Estas redes han demostrado bajas tasas de error y una menor necesidad de recursos computacionales que sus predecesoras, las redes neuronales completamente conectadas (*fully-connected*).

Por otro lado, es un reto mayúsculo el crear un sistema de conducción autónoma eficiente en consumo. Este desafío podemos dividirlo en dos subproblemas:

- Estudiar el software que sea capaz de procesar las imágenes en tiempo real y cumpla las restricciones de energía necesarias.
- Estudiar el hardware más adecuado para dicho software en búsqueda de la eficiencia energética.

El procesamiento de imágenes ha sido estudiado ampliamente, y se ha demostrado que lo más óptimo es hacer uso de varios núcleos en lugar de unos pocos, como suele ocurrir en CPUs estándar. En este trabajo trataremos el uso de GPU y Myriad, este último consiste en hardware específico para ofrecer recursos orientados a implementar redes neuronales, desarrollado por Intel.

Por último está el desafío de orquestar todo este software y hardware. Para el desarrollo de este proyecto se han utilizado los datasets KITTI, Tshingua-Daimler y Berkely Deepdrive. Estos conjuntos son presentados mas detalladamente en la sección 6. La detección de objetos se ha realizado usando tres tipos de redes: SSD, Faster y YOLO, que se presentarán en la sección 5.

El reentrenamiento de las redes se ha realizado usando el framework Tensorflow, del cual ampliaremos más información en las sección 9. La evaluación de las redes se ha realizado aprovechando el conjunto de herramientas desarrolladas por Intel, en el framework OpenVINO, el cual permite la ejecución en dispositivos específicos de inferencia de redes neuronales profundas, como Myriad. En la sección 9 se expondrán los dispositivos utilizados.

2. Historia

Tras el nacimiento de los coches motorizados llevo poco tiempo a los inventores de la época comenzar a pensar en los vehículos autónomos. En 1925, el inventor Francis Houdina [20] demostró como un coche controlado mediante ondas de radio conducía por las calles de

Manhattan sin nadie al volante. Según el *New York Times* [40], el vehículo controlado por radio podía encender el motor, cambiar de marchas y hacer sonar el claxon, "como si una mano fantasmal estuviera al volante". Ese coche se muestra en la siguiente imagen:

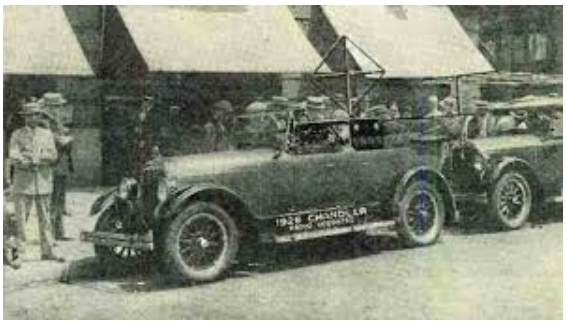

Figura 2.1: Vehículo controlado por radio sin conductor en 1925. Fuente: [20]

El concepto del vehículo autónomo se vio representado en la convención New York World's Fair en 1939, en la sección Futurama [41]. General Motors creó esta sección como una exhibición de como esperaban que fuera el futuro en América en 20 años. Los ingenieros incluyeron un sistema automatizado de autovías en los cuales los coches autónomos llevarían a la gente de un lugar a otro. Desgraciadamente, ha costado mas de 60 años hasta que los primeros vehículos robotizados empezaran a transitar nuestras carreteras.

Jhon McCarthy, uno de los padres fundadores de la inteligencia artificial, describió algo similar al concepto moderno de vehículo autónomo en un ensayo titulado "*Computer-Controller Cars*" [38] en 1968. McCarthy hace referencia a un vehículo capaz de navegar en la vía pública, usando una cámara de televisión que captará la misma información del entorno de la que dispone un ser humano. Además, los usuarios serán capaces de introducir un destino a través de un teclado al igual que realizar paradas para descanso. Observamos que McCarthy realizó una gran aproximación de como serían estos coches, siendo sus predicciones muy cercanas a la realidad del día de hoy, donde los vehículos autónomos usan potentes cámaras y radares, y se comunican con el usuario a través de pantallas táctiles. El vehículo propuesto por McCarthy no fue construido, pero el ensayo de McCarthy inspiró a muchos investigadores a trabajar en esta dirección.

En la década de 1990, el investigador Dean Pomerleau de la universidad Carnegie Mellon escribe una tesis doctoral [44], describiendo como las redes neuronales podrían permitir a los vehículos autónomos procesar imágenes de la carretera y producir una salida de ordenes capaz de controlar el vehículo en tiempo real. Pomerleau no fue el único investigador en trabajar en coches autónomos, pero su uso de las redes neuronales ofrecía un camino mucho mas eficiente que los intentos hasta la fecha. En 1995, Pomerleau y su compañero de investigación Todd Jochem prueban su sistema de conducción autónoma Navlab [61] en la carretera. La camioneta, en la cual tenía que controlar velocidad y frenado , viajó

2797 millas de costa a costa, desde Pittsburgh, Pensilvania a San Diego, California en un viaje que la pareja nombró "*No Hands Across America*" [42].

En 2002, DARPA(Defense Advanced Research Projects Agency) anuncia *Great Challenge* [17] , ofreciendo a las instituciones investigadoras un premio de un millon de dólares si eran capaces de construir un vehículo autónomo capaz de conducir las 142 millas a través del desierto de Mojave. La fecha del desafió concluyó en 2004, ninguno de los 15 competidores fueron capaces de lograr completar el trayecto. El mejor consiguió realizar menos de 8 millas [18], algo que le costó horas, antes de terminar envuelto en llamas. El fracaso fue tremendo, tanto que se llegó a pensar que se necesitarían décadas para conseguir el objetivo que construir coches totalmente autónomos.

Google comienza a desarrollar en secreto un proyecto de vehículos de conducción autónoma, llamado Waymo [63] en 2009. Al poco tiempo, Google [56] anunció que sus vehículos autónomos habían recolectado 300000 millas sin accidentes. Hasta la fecha, Google cuenta con mas de una treintena de prototipos, que han rodado unas 2000000 de millas en cuatro ciudades distintas. A día de hoy, otras compañías como Uber o Tesla han adelantado a Google en la comercialización del vehículo autónomo, hecho que se debe principalmente a la insistencia de la cúpula directiva de Google de solo comercializar vehículos completamente autónomos, y no uno *parcialmente* autónomo como hacen sus competidores.

Sin embargo, no podemos hablar de coches autónomos sin mencionar a Elon Musk. Elon Musk fundó Tesla Motors con el dinero de la venta de Paypal. Su intención era que Tesla fuera un fabricante de vehículos eléctricos y eficientes, basados en energías limpias [54]. Tras la compra de la compañía, Elon reunió a la prensa y comunicó que su intención era desarrollar vehículos completamente autónomos asequibles en un plazo de 3 a 5 años. Aunque le llevó mucho mas tiempo del que inicialmente prometió, en 2014 Tesla incorporó en sus coches el hardware necesario para dar la capacidad al vehículo de conducción autónoma. Estos sensores y cámaras daban una cobertura de 360 grados al vehículo, capaz de reconocer todo su entorno. Toda esta tecnología daba una capacidad casi militar de conducción autónoma, aunque el sistema de conducción que incorporaba no era **totalmente** autónomo aún. No obstante, a día de hoy se trata del sistema de conducción mas testeado, fiable y seguro en el mercado. Se espera que en los próximos años llegue la evolución del sistema de conducción autónoma, de tal manera que logre ser completamente autónomo.

3. Contribución de cada estudiante

En esta sección se detallarán las contribuciones de los dos alumnos que han realizado el proyecto de fin de grado con el objetivo de cumplir con la normativa del TFG.

La totalidad del trabajo se ha realizado en conjunto, ya sea la investigación del software a utilizar, el estudio de los detectores de objetos, la realización de la memoria o el aprendizaje de utilización de las herramientas utilizadas. Además ha habido una constante comunicación acerca de como realizar el trabajo, que nuevas ideas podemos probar o feedback sobre lo realizado hasta el momento.

3.1. Contribución de Alejandro

En la siguiente lista presento un resumen en líneas generales de mi contribución al trabajo:

- Desarrollo de la introducción y motivación del proyecto. Para ello se han comprendido otros artículos de investigación y noticias de este tema en particular.
- Análisis e investigación de los posibles conjuntos de datos a utilizar. Algunos de los conjuntos de datos investigados son Waymo, Nuscenes, Berkeley Deepdrive Dataset, Udacity, Tshingua-Daimler y KITTI.
- Descripción y presentación de los conjunto de datos utilziados en el conjunto de datos: KITTI, Tshingua-Daimler, Berkeley Deepdrive Dataset.
- Investigación del contexto histórico de la rama de a conducción autónoma, así como principales hitos de investigación.
- Análisis y comprensión del trabajo realizado con el objetivo de presentar los objetivos cumplidos y conclusiones del trabajo.
- Investigación del estado del arte de los detectores de objetos, en particular, lectura y comprensión de los artículos de publicación de las redes: SSD Mobile Lite, SSD Inception V2 y Faster R-CNN.
- Investigación y presentación del estado del arte en el ámbito de la vision por computador, deep learning, redes convolucionales y transfer learning.
- Estudio e investigación de la documentación de los frameworks usados a lo largo del proyecto, especialmente centrándome en Tensorflow y Openvino.
- Búsqueda, edición, adaptación y programación de convertidores de formato entre conjuntos de datos para adaptarlos a los frameworks usados en el proyecto.
- Entrenamiento en los conjuntos de datos de las redes SSD Inception V2, SSD Mobile Lite y Faster R-CNN.
- Recolección de resultados tras el reentrenamiento de las redes SSD Inception V2, SSD Mobile Lite y Faster R-CNN.

- Presentación y análisis de los resultados obtenidos en el punto anterior mediante el diseño de gráficas y tablas adecuadas.
- Inferencia sobre el conjunto de datos KITTI para test creado de las redes SSD Inception V2, SSD Mobile Lite y Faster R-CNN.
- Recolección de resultados tras la inferencia de las redes SSD Inception V2, SSD Mobile Lite y Faster R-CNN.
- Presentación y análisis de los resultados obtenidos en el punto anterior mediante el diseño de gráficas y tablas adecuadas.
- Experimentación de las diferentes técnicas de data augmentation, modificación de anchors por defecto, ajuste de hiperparámetros de los modelos con el objetivo de mejorar la precisión.
- Creación del repositorio de Github donde se presentan las libretas de comandos para el entrenamiento e inferencia de los modelos. Además están presentes los archivos de configuración de cada una de las redes utilizadas, especificando los hiperparámetros escogidos y los conjuntos de datos.

3.2. Contribución de Rafael

A continuación presento un resumen de las principales tareas y contribuciones realizadas por mí durante el trabajo:

- Investigación del estado del arte en el ámbito de la visión por computador y en específico la detección de objetos, comparando las diferentes arquitecturas y modelos propuestos en los últimos años.
- Estudio de utilización de los frameworks Darknet, Tensorflow y OpenVINO, en cuanto a los diferentes modelos preentrenados que ofrece cada uno, la metodología o flujo de trabajo general para el reentrenamiento y fine-tuning de modelos y ejecución de inferencia y diferentes herramientas ofrecidos por cada uno.
- Investigación teórica de los detectores de objetos SSD Mobile Lite, YOLOv4, YOLOv3 y YOLOv3 Tiny.
- Investigación de las diferentes técnicas de data augmentation, otras técnicas de preprocesamiento y ajuste de hiperparámetros de las distintas redes.
- Estudio de cuáles de las anteriores técnicas pueden ser implementadas en los distintos frameworks y cómo.
- Investigación del proceso de conversión de la representación de los modelos entre los formatos de los distintos frameworks.
- Análisis de los diferentes conjuntos de datos disponibles para la conducción autónoma, entre ellos KITTI, Tshingua-Daimler, Beerkeley Deedrive Dataset, Waymo,

NuScenes, Oxford Radar Robotcar, CityPersons y Caltech.

- Investigación, desarrollo y modificación de scripts para la conversión entre los formatos de las anotaciones o cajas delimitadoras de los distintos conjuntos de datos utilizados.
- Elaboración de la descripción de los frameworks utilizados.
- Desarrollo del resumen, introducción, objetivos y conclusiones del trabajo.
- Obtención de resultados de las redes SSD Mobile Lite, SSD Inception V2, Faster R-CNN, YOLOv3 y YOLOv3 Tiny.
- Elaboración de diferentes tablas descriptivas con los resultados obtenidos por los modelos.
- Elaboración de gráficas comparativas entre los resultados obtenidos por los distintos modelos al realizar la inferencia en los diferentes dispositivos hardware utilizados atendiendo a distintas métricas como la precisión, la velocidad de inferencia o el ratio velocidad-consumo.
- Contribución al repositorio de github actualizando los scripts, archivos de configuración y cuadernos desarrollados por mí.

4. Estado del arte

Este proyecto abarca principalmente dos campos de la inteligencia artificial, que son la visión por computador y el deep learning. Esta sección tiene el propósito de hacer una introducción teórica a ambas ramas de la inteligencia artificial. En primer lugar, se realiza una introducción al campo de la visión por computador. Progresivamente se presentan los problemas más destacados de este campo. En segundo lugar, se recorre históricamente los hechos que han llevado al nacimiento del deep learning. Luego, se exponen los factores que mas han influido en el ascenso meteórico de popularidad que ha recibido este campo. A continuación, se introduce el concepto de redes neuronales convolucionales. La conclusión de la sección se lleva a cabo con la introducción del concepto de transfer learning, muy conveniente en este proyecto.

4.1. Introducción a la Visión por Computador

Uno de las ramas de investigación más fascinantes de la IA es la Visión por Computador. Este campo se centra en replicar partes del complejo sistema de visión humano, de forma que permita a los ordenadores identificar y procesar objetos en imágenes y vídeos de forma similar a como los humanos los hacemos. Sin embargo, hasta hace poco la capacidad de la visión por computador era muy limitada.

Gracias a los avances en la inteligencia artificial junto a las innovaciones en deep learning y redes neuronales, el campo ha crecido enormemente en los últimos años, siendo capaz

de sobrepasar a los humanos en ciertas tareas de detección y etiquetación de objetos. A grandes rasgos, podemos distinguir dos factores han provocado este crecimiento.

En primer lugar, destaca el sustancial incremento de la cantidad de datos que generados actualmente, estos permiten entrenar y mejor la visión por computador. Se estima que mas de 1800 millones de imágenes son compartidas de forma online cada día [26].

En segundo lugar, el significativo aumento de la capacidad de cómputo que permite analizar los datos. Es fácil ver que este campo se ha visto altamente beneficiado por el surgimiento de algoritmos mas eficientes y el diseño de hardware mas robusto y potente.

4.1.1. Problemas dentro de la Visión por Computador

La visión por computador nos permite abordar una serie de tareas muy diversas. Estos retos son complicados principalmente debido a nuestro desconocimiento de como funciona la visión humana en primer lugar, sin embargo, como cualquier problema relacionado con el cerebro aun queda un largo camino. Principalmente predomina el estudio sobre los siguientes problemas:

- Clasificación de objetos: Dentro de un conjunto de categorías conocidas a priori, ¿A qué categoría de objetos pertenece la siguiente fotografía?
- Identificación de objetos: ¿Qué tipo de objeto es el que se observa en la fotografía?
- Verificación de objetos: Dado una categoría de objetos, ¿Es el objeto de la imagen?
- Detección de objetos: Dados una serie de objetos a reconocer a priori, ¿Dónde se encuentran los objetos en la imagen?
- Segmentación de objetos: ¿Qué píxeles pertenecen al objeto de la imagen?
- Reconocimiento de objetos: ¿Qué objetos hay en la imagen y donde se sitúan?

Con la ayuda de la siguiente imagen podemos entender mas fácilmente en que consisten los distintos problemas que afronta la visión por computador.

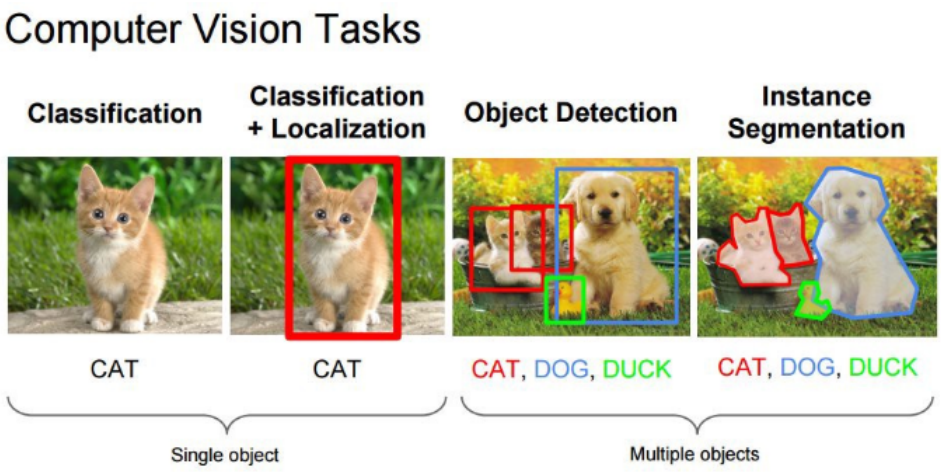

Figura 4.1: Desafíos en la Visión por Computador. Fuente: [11]

En particular, nosotros nos vamos a centrar en el reconocimiento de objetos, es decir, identificar que objeto hay en la imagen y dónde se encuentra. Al estar centrados en la conducción autónoma, los objetos a reconocer y situar serán: peatones, ciclistas y vehículos.

4.2. Deep Learning

Deep Learning [23] es una rama dentro del aprendizaje automático cuya meta es aprender características de alto nivel a partir de un conjunto de datos usando arquitecturas jerárquicas. Este enfoque permite a los modelos computacionales basados en múltiples capas, aprender y extraer información imitando como el cerebro percibe y asimila la información, permitiendo capturar patrones subyacentes en conjuntos de datos a gran escala.

El surgimiento del interés por esta tecnología se ha producido debido a que ha superado ampliamente el rendimiento de las técnicas punteras previas en muchas áreas de interés de la inteligencia artificial, como pueden ser procesamiento del lenguaje natural [58], visión por computador [15, 1], transfer learning [14, 28] y muchas más.

La ambición de crear un modelo que simulara el cerebro humano provocó el inicio del desarrollo de las redes neuronales. En 1943, McCulloch and Pitts [39] intentaron comprender como el cerebro podía producir patrones de alta complejidad usando células básicas interconectadas, llamadas neuronas. El modelo de una neurona de McCulloh y Pitts, llamado el modelo MCP, fue una contribución vital al desarrollo de las redes

neuronales artificiales. En la siguiente tabla [6] se muestran cuales han sido las mayores contribuciones a este campo que han contribuido a llegar a la era actual del deep learning:

Milestone/contribution	Contributor, year
MCP model, regarded as the ancestor of the Artificial Neural Network	McCulloch & Pitts, 1943
Hebbian learning rule	Hebb, 1949
First perceptron	Rosenblatt, 1958
Backpropagation	Werbos, 1974
Neocognitron, regarded as the ancestor of the Convolutional Neural Network	Fukushima, 1980
Boltzmann Machine	Ackley, Hinton & Sejnowski, 1985
Restricted Boltzmann Machine (initially known as Harmonium)	Smolensky, 1986
Recurrent Neural Network	Jordan, 1986
Autoencoders	Rumelhart, Hinton & Williams, 1986 Ballard, 1987
LeNet, starting the era of Convolutional Neural Networks	LeCun, 1990
LSTM	Hochreiter & Schmidhuber, 1997
Deep Belief Network, ushering the "age of deep learning"	Hinton, 2006
Deep Boltzmann Machine	Salakhutdinov & Hinton, 2009
AlexNet, starting the age of CNN used for ImageNet classification	Krizhevsky, Sutskever, & Hinton, 2012

Figura 4.2: Hitos más importantes en la historia de las redes neuronales y el aprendizaje automático, que han conducido a la era del deep learning. Fuente: [24]

El nacimiento de la era del deep learning se produjo con la disruptiva contribución de Hilton [21] en el año 2006, cuando introdujo la Deep Belief Network. Esta red fue la principal causante del desarrollo exponencial de la última década en cuanto a arquitecturas profundas y algoritmos de deep learning.

Sin embargo, las redes neuronales existen desde ha mediados del siglo XX. Una pregunta natural que surge es por qué las redes neuronales profundas han comenzado a tener un interés masivo recientemente. Hay muchos factores que contribuyen al ascenso meteórico de la popularidad de las redes neuronales profundas, algunos de ellos son:

- Aparición de datasets de entrenamiento masivos con etiquetas de alta calidad.
- Avances en las capacidades de computación paralela e implementaciones multi-core y multi-thread.
- Aumento de la capacidad de procesamiento, especialmente debido al uso de GPUs, así como, la disminución del coste del hardware de computación.
- Surgimiento de numerosas plataformas de software como Tensorflow [36], PyTorch [2], Caffe [65], Chainer [52], etc, que permiten una integración sencilla y a alto nivel de diversas arquitecturas en un framework de computación de GPU, sin la necesidad de entrar en detalles de bajo nivel.
- Introducción de nuevas técnicas de regularización que ayudan a evitar el overfitting y mejoran el rendimiento conforme se escala . Algunas de estas técnicas son: dropout, data augmentation, early stopping etc.
- Avances en la optimización de los algoritmos de aprendizaje automático que son capaces de producir soluciones muy cercanas a las óptimas.

4.2.1. Redes Convolucionales

Los avances en visión por computador con la llegada del deep learning han sido desarrollados y perfeccionados con el tiempo gracias a un algoritmo en particular: las Redes Neuronales Convolucionales [67].

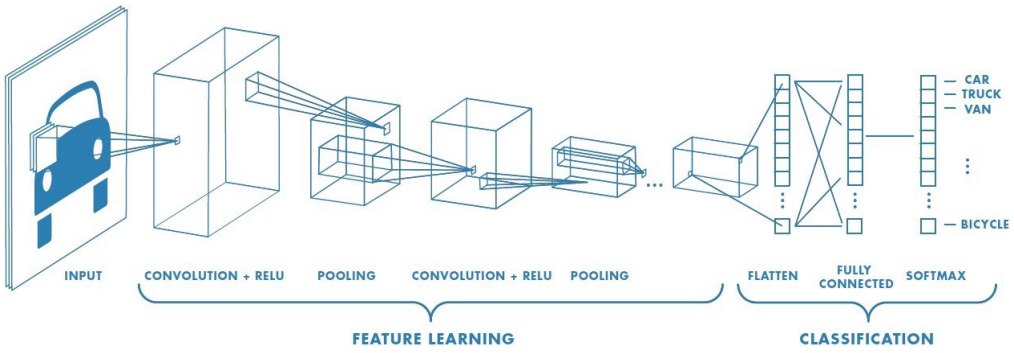

Figura 4.3: Arquitectura General de Redes Neuronales Convolucionales. Fuente: [12]

Una Red Neuronal Convolucional(CNN) es un algoritmo de Deep Learning que toma una imagen de entrada, asigna importancia (conjunto de pesos aprendidos) a los aspectos/-características de la imagen y es capaz de diferenciar unos de otros. Mientras que antes de la llegada de estas redes se aplicaban filtros diseñados a mano para detectar los aspectos relevantes de una imagen, con suficiente entrenamiento, las CNN han conseguido aprender esas características reseñables de las imágenes.

La arquitectura de una CNN intenta replicar los patrones de conectividad de las neuronas del cerebro humano y está inspirada en la organización del córtex visual [34]. Esta arquitectura permite realizar un balance entre una alta capacidad de aprendizaje de las características relevantes de la imagen y la escalabilidad de la red sobre datasets masivos [68].

Con el fin de cumplir con esas expectativas, la CNN realiza una reducción de las imágenes a una forma mucho mas sencilla de procesar, sin perder las características críticas necesarias para realizar una buena predicción. Para ello se basa en la sucesiva aplicación de dos capas: la capa de Convolución y la capa de Pooling.

La capa de convolución se basa en la operación de *convolución* sobre la imagen. El objetivo de esta operación es extraer las características de alto nivel de la imagen de entrada. El término convolución en matemáticas hace referencia a la combinación de dos funciones para producir una tercera función. En el caso de una CNN, la convolución se realiza sobre una imagen de entrada usando un filtro(también llamado kernel) que produce un mapa de características. Más específicamente, la convolución es ejecutada deslizando el filtro sobre la imagen de entrada. En cada localización, la multiplicación matricial se realiza y la suma del resultado se guarda en el mapa de características. La siguiente imagen refleja esta operación:

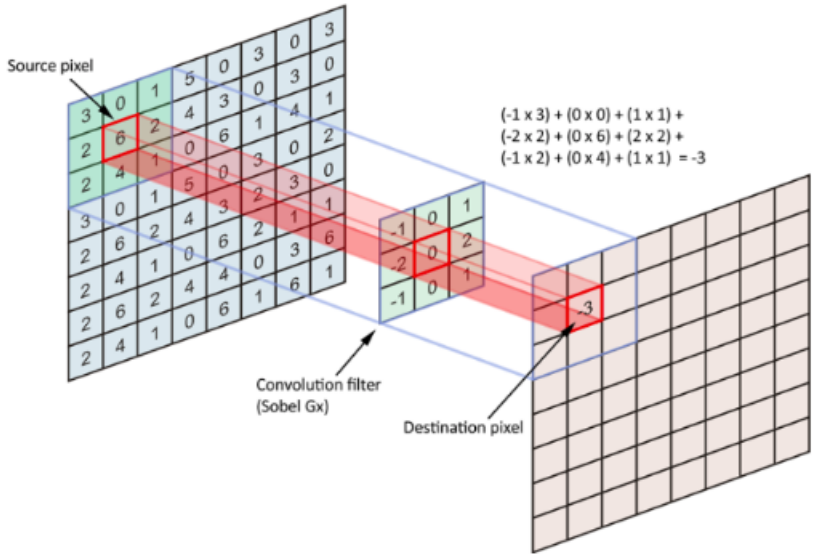

Figura 4.4: El filtro se desliza sobre la entrada y la operación tiene un resultado que se almacena en la nueva capa. Fuente [13]

Una CNN necesita no estar limitada en el número de capas convolucionales, así que, se suelen aplicar sucesivamente. Tradicionalmente, la primera capa convolucional es responsable de capturar características de bajo nivel como los bordes, el color, la orientación del gradiente, etc. Conforme se añaden capas, la arquitectura se adapta a las características de alto nivel, proporcionando a la red una comprensión total de las imágenes del dataset.

La capa Pooling se encarga de reducir el tamaño espacial de las características convolucionadas. El objetivo es reducir el coste computacional requerido para procesar los datos reduciendo la dimensionalidad de estos. Además, permite extraer las características dominantes que son invariantes frente a rotaciones y cambios de posición, manteniendo la efectividad del modelo durante el entrenamiento.

Se distinguen dos tipos de capas pooling: Max Pooling y Average Pooling. Max Pooling devuelve el máximo valor de la porción de imagen analizada, mientras que por otro lado, Average Pooling devuelve la media de todos los valores de la porción de imagen analizada. En general, Max Pooling ofrece un mejor rendimiento. La siguiente imagen ilustra la operación max pooling descrita:

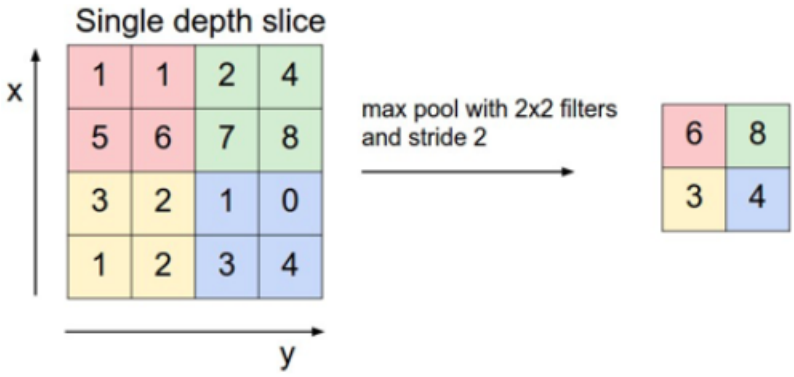

Figura 4.5: Max pooling toma los valores más grandes. Fuente [13]

La capa Convolucional y la capa Pooling, forman juntas la capa i-ésima de una CNN. Dependiendo de la complejidad de la imagen, el número de capas puede aumentar para capturar incluso mas detalles a bajo nivel, pero con un incremento del coste computacional.

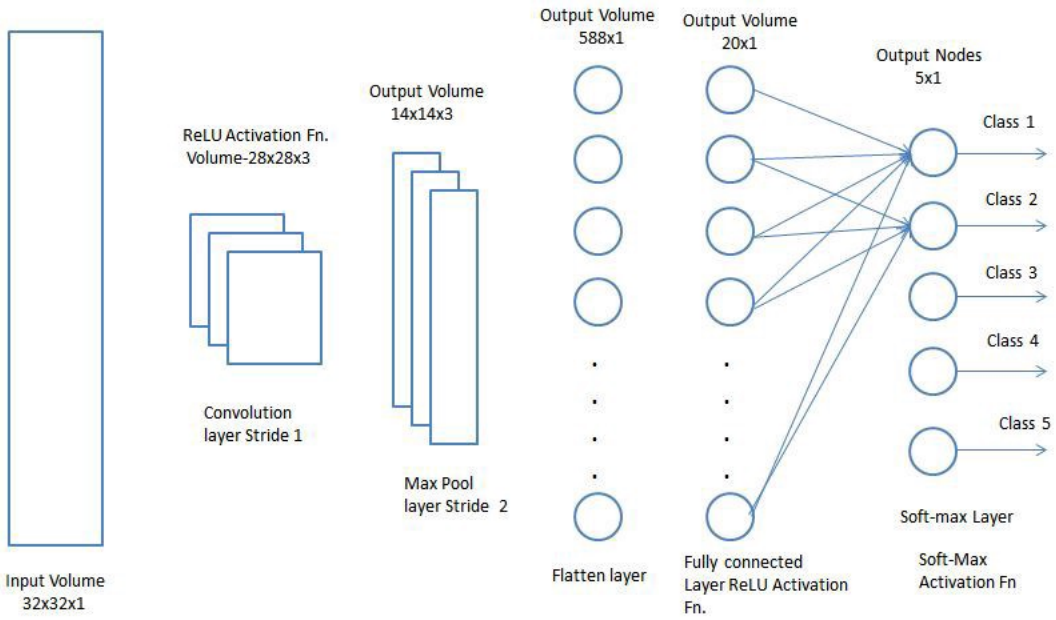

Figura 4.6: Capa Fully-Connected y Red Neuronal Feed-Forward. Fuente [12]

Tras la sucesiva aplicación de estas capas, el modelo es capaz de reconocer las principales características. La salida de este proceso suele ser la entrada de una capa Fully-Connected. Esta capa permite aprender las relaciones no-lineales entre las características capturadas por las capas anteriores.

Finalmente, la salida la capa Fully-Connected permite alimentar a una red neuronal Feed-Forward. Esta red neuronal, mediante los sucesivos entrenamientos aprenderá a distinguir entre las características dominantes y las características a bajo nivel de las imágenes, permitiéndole clasificar las imágenes en la categoría correspondiente.

Finalmente mencionar que aunque en este trabajo se ha querido realizar un breve introducción de las CNN, existen numerosas arquitecturas de CNN disponibles, cada una con sus características especificas y sus ventajas para determinados problemas. Algunas de las más conocidas son:

- LeNet [66]
- AlexNet [1]
- GoogLeNet [9]
- ResNet [30]

4.2.2. Transfer Learning

Transfer Learning es una técnica que permite abordar dos de los principales problemas del deep learning:

- Alta inversión en tiempo de reentrenamiento de redes neuronales debido a conjuntos de datos con cantidades de datos masivas.
- Enorme gasto de recursos computacionales para el reentrenamiento de redes.

Este método basado en la utilización del conocimiento aprendido por un modelo tras ser entrenado sobre un dataset con el fin de reducir el tiempo de reentrenamiento [51]. La idea es reentrenar el modelo sobre un nuevo dataset partiendo del conocimiento adquirido por el modelo sobre el dataset anterior, con el objetivo de acelerar el proceso de reentrenamiento. A menudo, la transferencia del aprendizaje se produce de un modelo mas complejo hacia un modelo mas sencillo. En nuestro caso, las redes están entrenadas sobre el dataset generalista COCO [33]. Este conjunto de datos esta enfocado en la detección y localización de 90 tipos de objetos, y usamos transfer learning, para acelerar el reentrenamiento sobre los conjuntos de datos de conducción autónoma usados en este proyecto.

En la siguiente imagen ilustramos como se produce este proceso de transferencia de conocimiento desde un modelo mas complejo hacia un modelo mas sencillo:

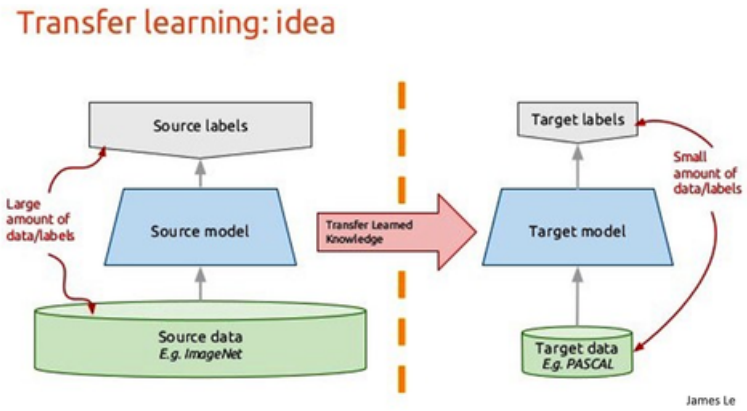

Figura 4.7: Transfer Learning. Fuente [62]

En el contexto de las CNN, transfer learning se implementa tomando los parámetros de la capa de captación de características para inicializar la capa de captación de características sobre el nuevo dataset. En concreto, se suelen coger los parámetros de las capas que se encargan de captar las características de bajo nivel como los bordes, texturas, esquinas y formas; pues estás son características presentes en todos los conjuntos de datos.

En el campo de la detección de objetos, se requiere aun mas procesamiento por parte de las capas tras la extracción de características, pues se sigue de un clasificador, ya que la detección de objetos debe predecir no solo la clase del objeto, sino su localización también. Esto implica que el reentrenaminento de estas redes sea muy costoso en tiempo y capacidad de cómputo. Gracias a este método, evitamos necesitar de semanas para reentrenar las redes.

5. Arquitecturas de Detección de Objetos

Dentro de las arquitecturas *deep learning* se distinguen dos líneas de investigación a día de hoy, siendo la primera mas antigua que la segunda. La primera se conoce como **arquitectura de dos fases**, generando primero regiones candidatas de la localización del objeto en la imagen y posteriormente clasificando los objetos asignándole una categoría, por ejemplo en este caso: coche, ciclista o peatón. Este tipo de detectores también se les conoce por detectores de objetos **basados en regiones**.

Por otro lado, el segundo tipo de detectores se conocen por tener una **arquitectura de una sola fase**. Estos realizan una abstracción mayor, pues ven el problema de detección de objetos como un problema de regresión(estimación de las cajas delimitadoras)

y clasificación. De esta forma, en una sola fase, son capaces de inferir la localización y clasificación del objeto.

5.1. Arquitecturas de dos fases

Las arquitecturas de dos fases son también conocidas como las arquitecturas basadas en regiones. El nacimiento de estas arquitecturas se produjo en 2012, cuando AlexNet [1] ganó el desafío ILSVRC, trayendo consigo el dominio de las CNN al campo de la detección de objetos. Un enfoque por fuerza bruta de detección de objetos es usar una ventaja deslizante de izquierda a derecha, de arriba a abajo para identificar objetos usando clasificación. Un ejemplo sería la imagen inferior:

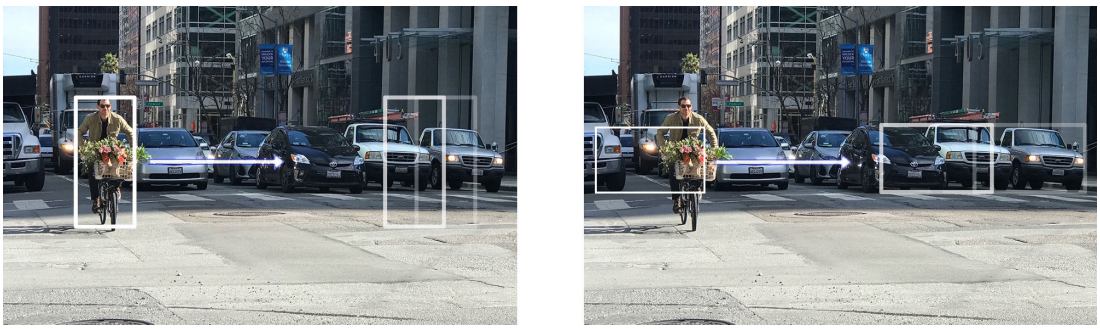

Figura 5.1: Ventana deslizante de izquierda a derecha sobre la imagen. Fuente: [49]

Como vemos en la figura superior, si queremos detectar diferentes objetos debemos usar ventanas de tamaños y formas variadas. En cada paso de deslizamiento de la ventana se recortan parches de la imagen según estas ventanas. Previamente a *alimentar* la red convolucional clasificadora, el parche es *redimensionado* a la entrada de la red, que a menudo suele ser fija. Este tipo de transformaciones no impactan la precisión de la red, ya que están entrenados para manejar estas imágenes. Esta red se encarga de extraer un número determinado de características del parche. Luego se aplica SVM(*support vector machine*) sobre las características extraídas para identificar a que clase pertenece el objeto encuadrado en el parche. Además, usamos un método de regresión lineal sobre las características para estimar los límites de la cajas delimitadoras. En la imagen inferior podemos ver como sería esta arquitectura:

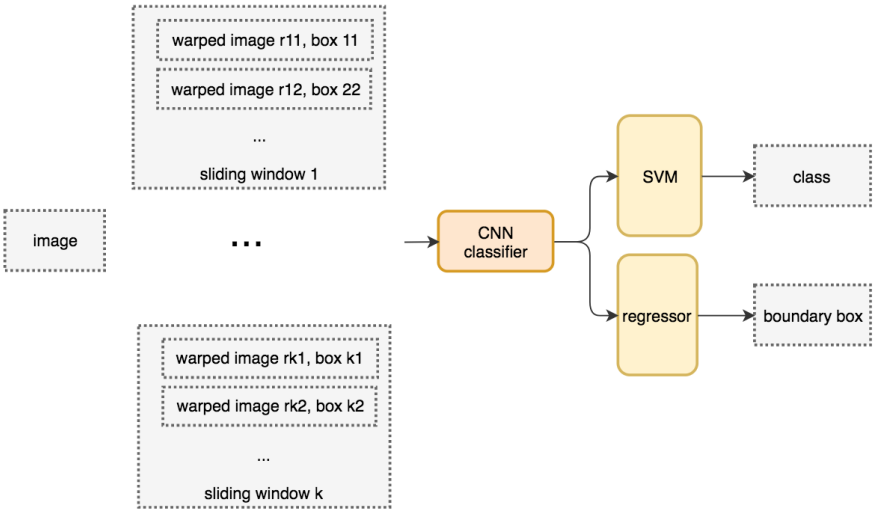

Figura 5.2: Arquitectura basada en un enfoque de fuerza bruta. Fuente: [49]

Sin embargo el enfoque por fuerza bruta es demasiado costoso en tiempo de inferencia y entrenamiento, así como intensivo en recursos computacionales. Una mejora obvia es reducir el número de ventajas deslizantes. De esta forma, se crea un método que se encargue de *proponer* **regiones de interés** o **ROIs** a partir de una imagen. Un ejemplo de este método es la **búsqueda selectiva**.

Al comienzo cada píxel es un grupo en si mismo. Posteriormente, se calcula la textura de cada grupo y se unen aquellos que sean mas similares. Con el objetivo de evitar que los grupos grandes se *coman* a los pequeños, tendrán prioridad la unión de grupos pequeños frente a los grandes. Se continua este proceso de unión hasta que se cumpla alguna condición de convergencia. En la imagen inferior, podemos ver como estos grupos se van formando en la primera fila, mientras que los rectángulos azules de la segunda fila muestra todas las posibles regiones de interés generadas en cada paso de la búsqueda selectiva.

Figura 5.3: Método de proposición de regiones: búsqueda selectiva. Fuente: [31]

En 2013 Ross Girshick [50] propone la primera arquitectura que hace uso de métodos de proposición de regiones de interés, **R-CNN**. Esta arquitectura propone cerca de 2000 ROIs por imagen. Luego cada región se utiliza como entrada de una red neuronal convolucional, de forma que cada región es procesada de manera individual. Este proceso es seguido de capas totalmente conexas que permiten realizar la clasificación del objeto y la estimación de la caja delimitadora. En la imagen inferior podemos ver esta arquitectura:

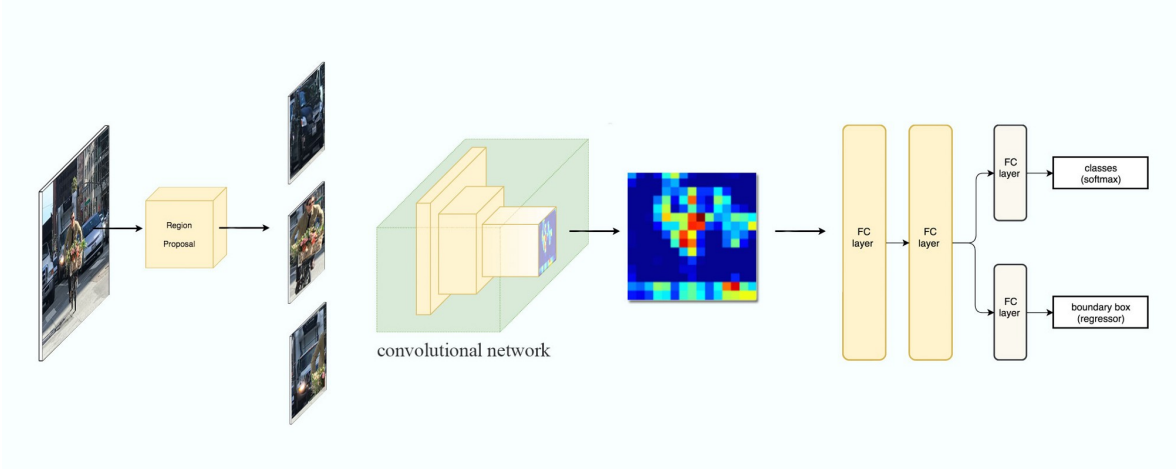

Figura 5.4: Arquitectura del detector R-CNN. Fuente: [49]

La ventaja de R-CNN frente al método de fuerza bruta usando ventanas deslizantes es notables, principalmente se debe a que es capaz de identificar ROIs de más calidad, aumentando la precisión en la detección de objetos. Además el tiempo de ejecución es mucho menor que el método de fuerza bruta. Aunque se ha mejorado sustancialmente respecto al método de fuerza bruta, R-CNN es aún lento en entrenamiento e inferencia, mayormente debido a que propone demasiadas regiones de interés para ser preciso, y muchas de estas regiones se sobreponen unas con otras. Además, si tenemos 2000 ROIs, cada una es procesada de forma independiente por la CNN, realizando 2000 extracciones de características sobre las mismas zonas.

En lugar de extraer características de cada parche de la imagen partiendo desde cero, podemos usar la CNN para extraer características de toda la imagen en primer lugar. Paralelamente utilizamos un método externo de proposición de regiones, como búsqueda selectiva, para crear ROIs que mas adelante se combinan con su correspondientes características extraídas formando los parches. Estos parches son la entrada de capas totalmente conexas que se encargan de la clasificación y la localización del objeto. En la imagen inferior presentamos esta arquitectura:

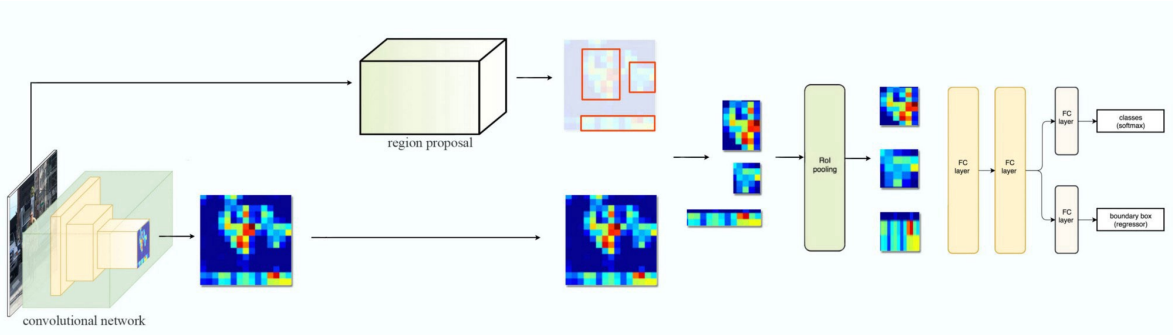

Figura 5.5: Arquitectura del detector Fast R-CNN. Fuente: [49]

Este detector de objetos es conocido por Fast R-CNN, que fue introducido por el mismo investigador que introdujo R-CNN, Ross Girshick [22] en 2015. Este detector redujo significativamente el tiempo de procesamiento para la detección de objetos con respecto a R-CNN. No obstante, Fast R-CNN sigue siendo lento, especialmente debido al método externo de proposición de regiones de interés, el cual es ejecutado en CPU, requiriendo la mayor parte del tiempo de procesamiento del detector.

Investigando como acelerar Fast R-CNN, resolvieron el problema de la generación de ROIs, Shaoquig Ren y su equipo propusieron **Faster R-CNN**[55]. El diseño de Faster R-CNN es similar al de Fast R-CNN, salvo por que reemplaza el método de proposición de ROIs por una red neuronal profunda llamada **Region Proposal Network** o **RPN**. Esta nueva red propone regiones a partir de las características extraídas por la CNN, siendo mucho mas eficiente en la generación de ROIs. En la siguiente imagen podemos ver la arquitectura de esta red:

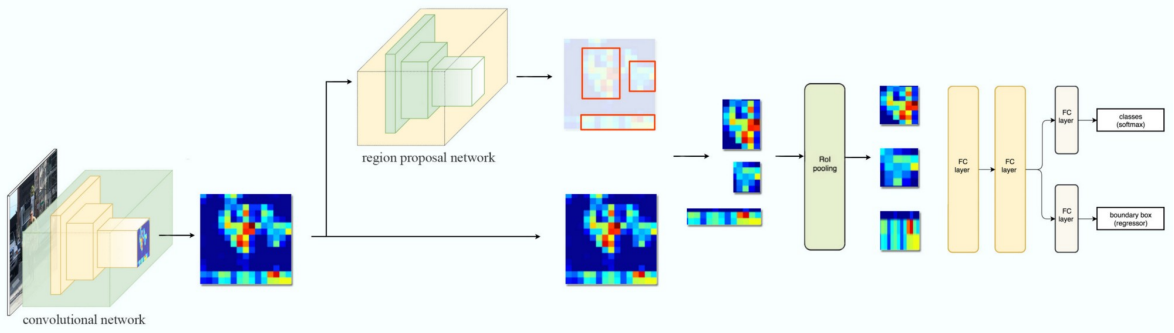

Figura 5.6: Arquitectura del detector Faster R-CNN. Fuente: [49]

Observamos como la arquitectura de Faster R-CNN es similar a la de Fast R-CNN salvo que reemplazando el método de proposición de regiones por RPN. Este reemplazo tiene como consecuencia que Faster R-CNN sea mas veloz que su predecesora. Este detector es el que hemos utilizado en este proyecto como detector de objetos. No obstante, a día de hoy existen arquitecturas que superan a Faster R-CNN.

La utilización de capas totalmente conexas para la clasificación es muy costoso en tiempo. En su lugar, se comenzaron a investigar nuevas redes de clasificación basadas en capas convolucionales que son muy mas rápidas que las totalmente conexas. De esta forma se trabajó en la integración de nuevas redes como **Resnet** [30] y R-FCN: object detection via region-based fully convolutional networks [9].

Integrando Resnet en la arquitectura Faster R-CNN, se diseñó la red **R-FCN** [29], un detector totalmente convolucional que consigue superar ampliamente tanto en precisión como en rendimiento a Faster R-CNN. Como conclusión, hay nuevas arquitecturas que superan a las ya presentadas, pero la intención era dar una introducción a las arquitecturas de dos fases que se ha conseguido en esta sección.

5.2. Arquitecturas de una fase

A lo largo de la sección anterior se han presentado las arquitecturas de dos pasos, viendo su evolución desde su comienzo con R-CNN hasta el estado del arte actual. En particular, en este proyecto hemos usado Faster R-CNN como representante de esta arquitectura. Mientras que este tipo de redes son el estado del arte en cuanto a precisión en la localización y clasificación de objetos, tienen la desventaja de ser demasiado lentas para la detección en tiempo real de objetos. Debido a la necesidad de la capacidad de detección de objetos en tiempo real, surgieron las arquitecturas de un paso. Los expertos decidieron reducir el número de fases, logrando en una fase arquitecturas capaces de inferir directamente a partir de una imagen las coordenadas de las cajas delimitadoras y la verosimilitud por clase. En esta sección se presentarán las arquitecturas de un paso usadas para la detección de objetos en este proyecto.

5.2.1. Single Shot Detector o SSD

SSD fue diseñada en 2015[35] para la detección de objetos en tiempo real. Esta red aumentó la velocidad del proceso de inferencia eliminando la necesidad de la RPN(*region proposal network*). Además, para recuperar la bajada en la precisión debida a esta eliminación, SSD incluye pequeñas mejoras incluyendo cajas por defecto y características multi-escala. En otros problemas de detección como puede ser sobre el dataset COCO, estas mejoras permiten alcanzar la precisión ofrecida por Faster R-CNN usando imágenes de baja resolución, manteniendo al mismo tiempo la capacidad de detección en tiempo real de objetos, una característica crucial para nuestro proyecto. Este detector de objetos esta compuesto por dos partes:

1. Extracción de las características de la imagen.
2. Detección de objetos mediante la aplicación de filtros de convolución.

Con el objetivo de facilitar la compresión del proceso, comenzaremos mostrando como realiza SSD detecciones de objetos con un *solo predictor*. La siguiente imagen muestra la arquitectura de una SSD con un único predictor:

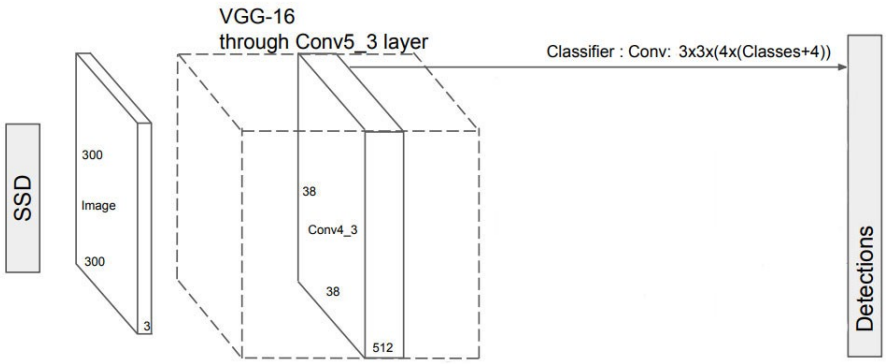

Figura 5.7: Arquitectura SSD con un solo predictor. Fuente: [35]

En la figura 5.7, se observa como inicialmente redimensiona la imagen a un tamaño de entrada 300×300 píxeles. Este redimensionamiento permite a la red una alta velocidad a la hora de realizar inferencia, con el coste de ser incapaz de detectar objetos pequeños. Tras el redimensionamiento se realiza la extracción de las características de la imagen usando una CNN, en concreto, VGG16. En realidad se trata de una adaptación de esa CNN, pues originalmente VGG16 se usa para la clasificación de imágenes de gran calidad. Esta adaptación se lleva a cabo mediante la sustitución de las capas totalmente conexas por capas convolucionales que se encargan de la extracción de características en diversas escalas.

SSD divide la imagen en $C \times C$ cuadriculas, donde cada celda que represente el centro de un objeto será la responsable de detectarlo. Con este objetivo, cada casilla predice B cajas delimitadoras y la probabilidad de que esta caja contenga un objeto dentro. La capa Conv4_3 es un ejemplo de una capa de detección de objetos que realiza este división en 38×38 celdas. Con el objetivo de mostrar un ejemplo, usaremos una simplificación de la capa Conv4_3, de forma que divida la imagen en 8×8, de esta forma tenemos:

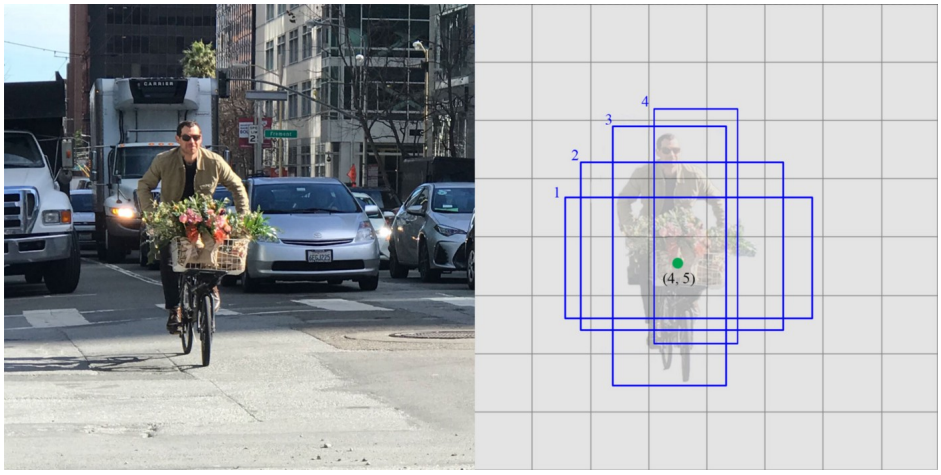

Figura 5.8: Conv4_3 realizando 4 predicciones por casilla. Fuente: [57]

Cada predicción es la composición de una caja delimitadora y 4 puntuaciones(una por clase, en nuestro caso ciclista, peatón y coche, más una clase extra para cuando no hay objetos). Después, se le asigna a la predicción de la clase con mayor puntuación, en este caso, la clase ciclista. Se observa que el coste computacional de esta red depende del número de casillas en las que se divida la imagen. Por ejemplo, Conv4_3 realiza un total de $38 \times 38 \times 4 = 5776$ predicciones, de las cuales la mayoría no contienen objetos, por ello reserva la clase cero para este caso particular. La siguiente imagen se muestra la fase de puntuación por clase:

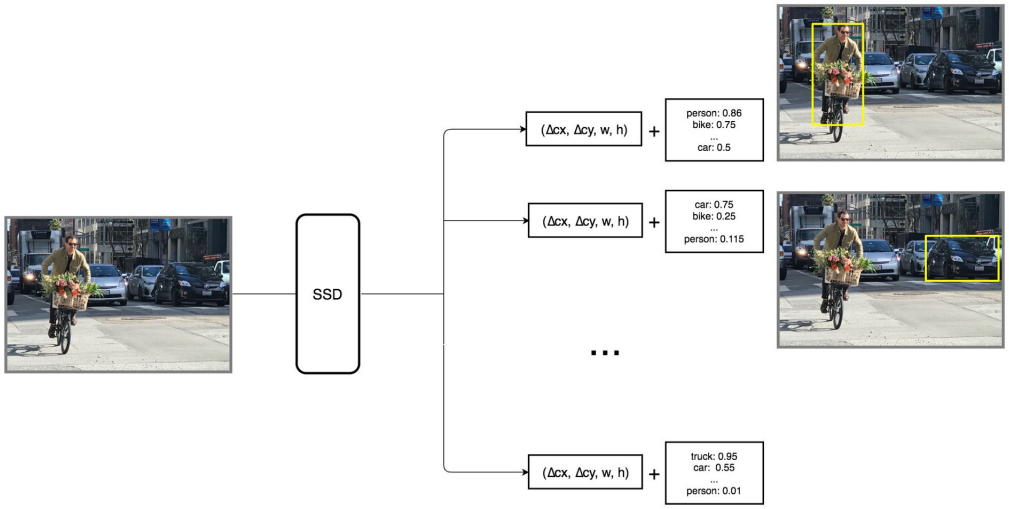

Figura 5.9: Cada predicción incluye una caja delimitadora y 4 puntuaciones para 4 clases(una para cuando no hay objeto). Fuente: [57]

El número de casillas en las que se dividen las imágenes es una de las grandes delimitaciones de la red por qué si son demasiado grandes son incapaces de detectar objetos pequeños, pues habrá mas de un objeto por casilla siendo incapaz de detectar la red estos objetos.

En realidad, SSD tiene múltiples detectores que permiten detectar mas de un objeto en una imagen de forma independiente. Como CNN reduce la dimensión espacial de forma gradual, la resolución de las capas de extracción de características también se reduce. Sin embargo, SSD aprovecha este hecho para detectar objetos de distintas escalas, siendo una de las claves la pirámide de redes convolucionales que constituye esta arquitectura. Esta estructura piramidal permite que cada capa convolucional se dedique a detectar objetos de distinto tamaño, siendo cada capa entrenada para ese tamaño específico. Por ejemplo, la capa 4×4 se usa para objetos de gran escala, mientras que la capa 8×8 para objetos mas pequeños:

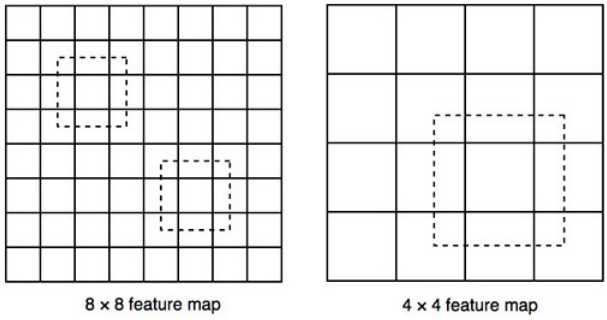

Figura 5.10: Capa 4×4 y 8×8 detectando objetos de distinto tamaño. Fuente: [57]

Finalmente, mostramos la estructura de la arquitectura multi-detector de SSD:

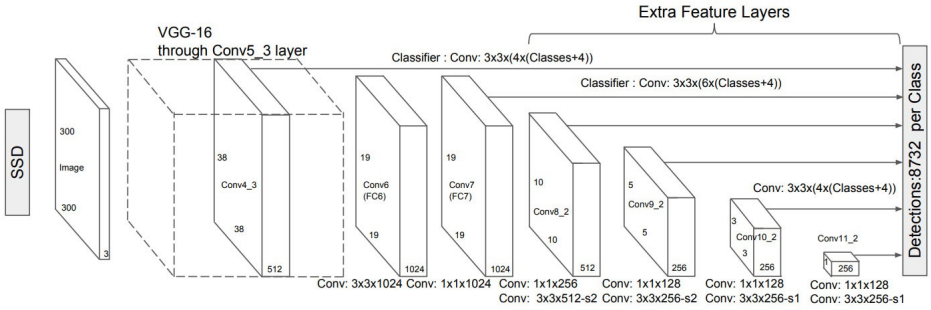

Figura 5.11: Arquitectura multi-detector SDD. Fuente [35]

Esta arquitectura contiene todas las capas comentadas en esta sección. Primeramente redimensiona la imagen a un tamaño 300×300. Posteriormente aparece la aplicación de filtros para la detección de objetos, por ejemplo usando la capa Conv4_3. Finalmente, las últimas capas de la derecha se encargan de la extracción de características.

Otro detector que se basa en un arquitectura SSD estudiado en este proyecto es SSD Mobile Lite [37]. Se trata de una red entrenada sobre el dataset COCO, que fue especialmente diseñado para entornos con restricciones de recursos, como puede ser el de la conducción autónoma. Este detector tiene la ventaja de necesitar hasta diez veces menos parámetros que YOLOV2, siendo capaz de mantener la precisión conseguida por las mejores redes SSD. Gracias a estas características, destaca por ser altamente eficiente, siendo de esta forma un perfecto candidato para la inferencia en tiempo real de detección de objetos.

5.2.2. YOLO

Los sistemas de detección actuales reutilizan clasificadores para realizar la detección. Para detectar un objeto, estos sistemas toman un clasificador para ese objeto y lo evalúan a lo largo de la imagen en varios lugares y escalas. Los sistemas como los modelos de partes deformables o deformable parts models (DPM) utilizan un enfoque de ventana corrediza en el que el clasificador se ejecuta en lugares uniformemente espaciados en toda la imagen.

Enfoques más recientes como el R-CNN utilizan métodos como propuesta de regiones para generar primero las potenciales cajas delimitadoras en una imagen y luego ejecutar un clasificador en cada una de estas cajas propuestas. Después de la clasificación, el post-procesamiento se utiliza para refinar las cajas delimitadoras, eliminar las detecciones duplicadas, y volver a clasificar las cajas en base a otros objetos de la imagen. Estos sistemas son lentos y difíciles de optimizar porque cada componente individual debe ser entrenado por separado.

YOLO [46] replantea la detección de objetos como un único problema de regresión, directamente de los píxeles de la imagen a las coordenadas de la caja delimitadora y las probabilidades de clase. En esta nueva forma de afrontar la detección propuesta por YOLO, una sola red convolucional predice simultáneamente múltiples cajas delimitadoras y probabilidades de clase para esas cajas. Podemos ver una ilustración de este funcionamiento en la figura 5.13. YOLO se entrena en imágenes completas y optimiza directamente el rendimiento de la detección. Este modelo unificado tiene varios beneficios sobre los métodos tradicionales de detección de objetos:

- YOLO es **extremadamente rápido**. Al enfocar la detección como un problema de regresión, no se necesitan sistemas complejos. Simplemente se ejecuta la red neuronal en una nueva imagen para predecir las detecciones. Además, YOLO alcanza más del doble de la precisión media de otros sistemas en tiempo real.
- YOLO **trabaja globalmente sobre la imagen**. A diferencia de las técnicas de ventana corredera y propuestas de regiones, YOLO ve toda la imagen durante el tiempo de entrenamiento y prueba, de manera que implícitamente codifica la información contextual sobre las clases, así como su apariencia.
- YOLO **aprende representaciones generalizables** de objetos. Cuando se entrena en imágenes naturales, YOLO supera los métodos de detección más avanzados como el DPM y R-CNN por un amplio margen. Dado que YOLO es altamente generalizable, es menos probable que se descomponga cuando se aplica a nuevos dominios o entradas inesperadas.

A pesar de estas ventajas, YOLO todavía se queda atrás de los sistemas de detección de última generación en la precisión. Aunque puede identificar rápidamente los objetos en las imágenes, tiene problemas para localizar con precisión algunos objetos, especialmente los pequeños.

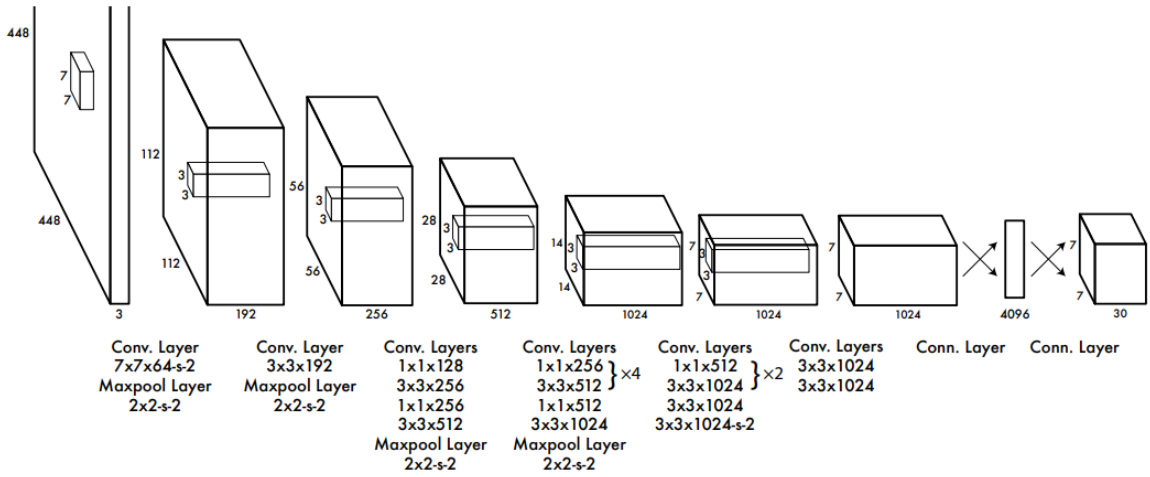

Figura 5.12: Estructura de la estructura de red convolucional de la primera versión de YOLO, con 24 capas convolucionales seguidas por dos capas totalmente conectadas. Fuente [46].

YOLO ha ido publicando varias versiones, desde la version inicial cuya estructura podemos ver en la figura 5.12 hasta la última de ellas YOLOv4[8] publicada el 23 de Abril de 2020, pasando por las versiones YOLOv3[48] y YOLOv3 tiny[25], la cual será de especial interés debido a su menor tamaño y por tanto mayor rapidez, una característica fundamental en la conducción autónoma para intentar conseguir procesamiento en tiempo real.

Funcionamiento global de estructuras YOLO

La filosofía de las estructuras YOLO consiste en unificar los diferentes componentes de la detección de objetos en una sola red. El funcionamiento global de la inferencia, que podemos ver de manera visual en la figura 5.13, es el siguiente:

1. Primeramente, la imagen es dividida en una cuadrícula de tamaño SxS. Si el centro de un objeto cae en una celda de la cuadrícula, esa celda es la responsable de detectar el objeto.

2. Cada celda detecta B cajas delimitadoras y nivel de confianza para dichas cajas, que reflejan cómo de seguro está el modelo de que esa caja contiene un objeto y cómo de seguro está de que efectivamente la caja se corresponde al objeto que ha detectado, es decir, $Confianza = P(Objeto) * IOU_{pred}^{truth}$. Si no hay objeto en esa celda, la confianza debería de ser 0, en otro caso debería ser el IOU entre la caja detectada y la read.

3. Cada una de las cajas delimitadoras está compuesta por 5 predicciones: x, y, w, h y la confianza. (x, y) representa el centro de la caja detectada y (w, h) la anchura

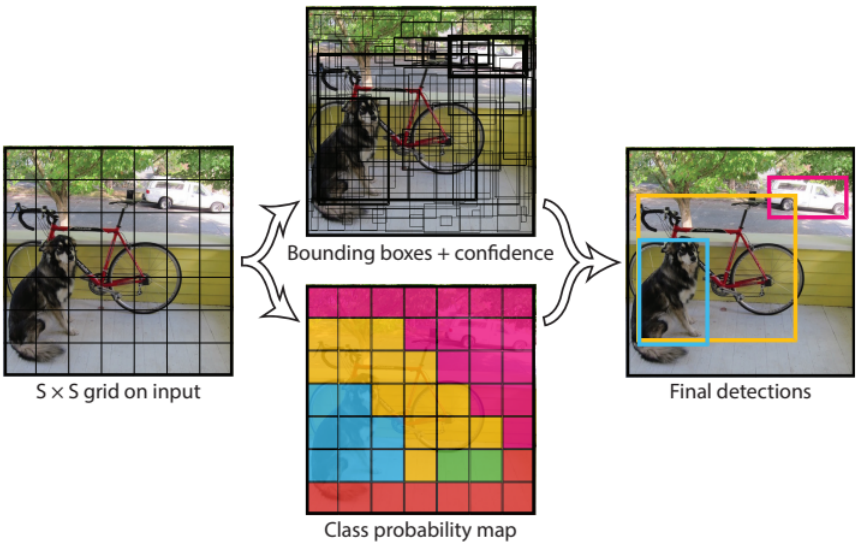

Figura 5.13: Ilustración del funcionamiento global de YOLO, donde primero se divide la imagen en cuadrículas más pequeñas, sobre éstas se calculan diferentes cajas delimitadoras o bounding boxes y las probabilidades de las distintas clases, y finalmente se obtienen las detecciones. Fuente [46].

y altura respectivamente.

4. Además, cada celda de la cuadrícula también predice C probabilidades condicionales $P(Clase_i|Objeto)$, una por cada clase, que representan las probabilidades de que en esa celda se encuentre el objeto de la clase i. Remarcar que se predice un vector de C probabilidades condicionales por cada celda, independientemente del número de cajas detectadas B.

5. En la inferencia, se multiplican las probabilidades condicionales de cada clase y la confianza de cada caja detectada, obteniendo así las probabilidades específicas de cada clase por cada caja detectada:

$$P(Clase_i|Objeto) * P(Objeto) * IOU_{pred}^{truth} = P(Clase_i) * IOU_{pred}^{truth}$$

Esta medida nos indica la probabilidad de que cada clase aparezca en las cajas detectadas y cómo de bien se ajusta dicha caja.

YOLOv3

Vamos a mencionar brevemente las mejoras que implementa YOLOv3 con respecto a la primera versión, alguna de estas mejoras implementadas en YOLOv2 y otras finalmente en YOLOv3:

- Normalización de los lotes en las capas convolucionales.
- Clasificador de alta resolución
- Última capa convolucional con anchor boxes o cajas de anclaje, también llamadas cajas a priori. En vez de predecir directamente las cajas, se predicen los offsets con respecto a unas cajas de anclaje previamente predefinidas. Esto es muy útil pues podemos definir dichas cajas de anclaje para que se adecuen a la forma de los objetos de nuestro dataset.
- Clusters dimensionales: Como hemos mencionado, en muchos problemas de detección los objetos tienen una forma determinada. Para calcular las K cajas a priori que mejor cubren esta gama de formas en un dataset, se utiliza el algoritmo K-Means.
- Predicción directa de la localización

Predicción de cajas delimitadoras

Para la predicción de las cajas delimitadoras utiliza la misma metodología que YOLO9000[47]. La red predice 4 coordenadas para cada caja delimitadora $t_x, t_y.t_w, t_h$, de forma que si la celda de la cuadrícula tiene un offset con respecto a la esquina superior izquierda de la imagen de (c_x, x_y) y la caja delimitadora a priori tiene anchura p_w, p_h, entonces las predicciones finales son:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

Es decir, la red predice offsets con respecto a cajas a priori o cajas de anclaje (anchor boxes). Podemos ver una visualización de estas ecuaciones en la figura 5.14 Esto es muy útil pues en muchos problemas de detección los objetos tienen una forma determinada. Por tanto, podemos definir dichas cajas de anclaje para que se adecuen a la forma de los objetos de nuestro dataset. Para calcular las K cajas a priori que mejor cubren esta gama de formas en un dataset, se utiliza el algoritmo K-Means.

Durante el entrenamiento se usa la suma de cuadrados como función de pérdida, de forma que el gradiente viene dado por $\hat{t}_* - t_*$ donde $\hat{t}_*$ es el offset predicho por la red y t_* es el offset real.

YOLOv3 predice una puntuación de objeto para cada caja delimitadora usando regresión logística. Si el cuadro delimitador a priori no es la mejor, pero se superpone a un objeto por encima de un umbral (en este caso 0.5), se ignora la predicción.

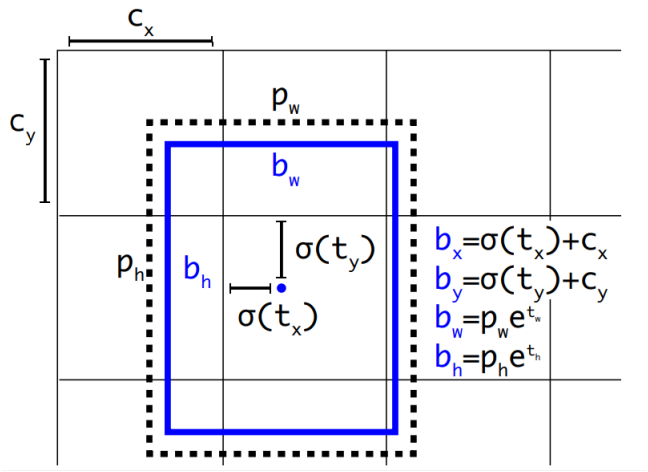

Figura 5.14: Cajas delimitadoras con dimensiones previas y predicciones. Se predice el ancho y la altura de la caja como offsets desde el centro de la caja. Fuente [48].

Predicción de clases

Cada casilla predice las clases que puede contener la caja delimitadora utilizando la clasificación multi-etiqueta, usando para ello clasificadores logísticos independientes. Durante el entrenamiento se usa como función de pérdida la entropía cruzada binaria.

Predicciones en diferentes escalas

YOLOv3 realiza predicciones en 3 escalas diferentes. El sistema extrae *features* o características de cada una de estas 3 escalas utilizando un concepto similar a las redes piramidales de características o feature pyramid networks[32]. En la estructura, después de las capas de extracción de características se añaden diferentes capas convolucionales, la última de ellas encargada de predecir un tensor 3d codificando las predicciones de cajas delimitadoras, probabilidad de contener un objeto y predicciones de clase. Así, el tensor final es de la forma $N \times N \times (N_{boxes} * (4 + 1 + N_{classes})$ donde N_{boxes} es el número de cajas predichas en cada escala, 4 se corresponde a los offsets de la caja delimitadora, 1 a la predicción del objeto o probabilidad de contener un objeto y $N_{classes}$ es el número de clases del dataset.

A se coge el feature map o mapa de características de las dos capas anteriores y se aumenta en 2×. También se coge un mapa de características de las capas anteriores de la red y se fusiona con las características aumentadas usando concatenación. Este método nos permite obtener información semántica más significativa.

Se realiza el mismo proceso una vez más para predecir cajas para la escala final. Así, las predicciones para la 3ª escala se benefician de todos los cálculos previos.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 5.15: Extractor de características Darknet-53. Fuente[48].

Extractor de características

Se utiliza un enfoque híbrido entre el extractor de características utilizado por YOLOv2 (Darknet-19) y capas residuales. Utiliza capas convolucionales sucesivas de 3×3 y 1×1, con algunas interconexiones o atajos. Tiene 53 capas convolucionales en total y recibe el nombre de **Darknet-53**. Podemos ver un resumen de su estructura en la figura 5.15.

En el caso de **YOLOv3 tiny** la estructura se reduce considerablemente a un total de 15 capas convolucionales, lo que aumenta considerablemente su velocidad en la inferencia pero sacrifica precisión, sobre todo en objetos muy pequeños.

Comparación con otras estructuras

Podemos ver una comparación en los resultados finales con otras estructuras populares para la detección de objetos en la figura 5.16

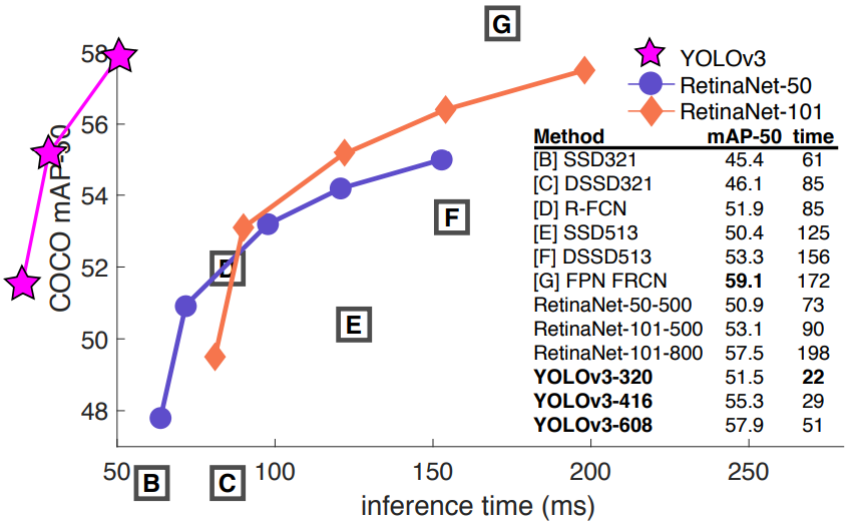

Figura 5.16: Comparación distintas estructuras de redes neuronales convolucionales donde podemos observar el tradeoff o balance entre velocidad y precisión para .5 IOU, destacando YOLOv3 por su gran velocidad y manteniendo una buena precisión. Fuente[48].

YOLOv4

Haremos un breve comentario sobre la última versión de YOLO, YOLOv4 [8], que presenta mejoras considerables con respecto a su predecesor, YOLOv3, tanto en velocidad de inferencia como en precisión de en torno a un 10-12 %, como podemos observar en la figura 5.16. Podemos además encontrar una descripción muy detallada junto a los parámetros que utiliza la red en este enlace.

Sin embargo, aunque es una propuesta muy interesante para la conducción autónoma por su gran velocidad en cuanto a FPS y manteniendo una muy buena precisión con respecto a otros detectores de última generación, no lo trataremos en este trabajo debido a que uno de los objetivos finales es la evaluación de estas estructuras o redes en dispositivos de bajo consumo, utilizando para ello el framework openVINO 9.2, el cual no tiene implementado aún soporte para esta estructura.

6. Presentación de conjuntos de datos utilizados

Los resultados de las redes de más conocidas de detección de objetos han sido decepcionantes sobre el dataset KITTI como se muestra en la sección 11.1. En ellos se observa un claro problema para la localización y detección de los objetos en las imágenes presentes en el dataset KITTI. El problema se ve especialmente acentuado en las clases peatón y ciclista. Uno de los principales desafíos que intentaremos solventar en este proyecto

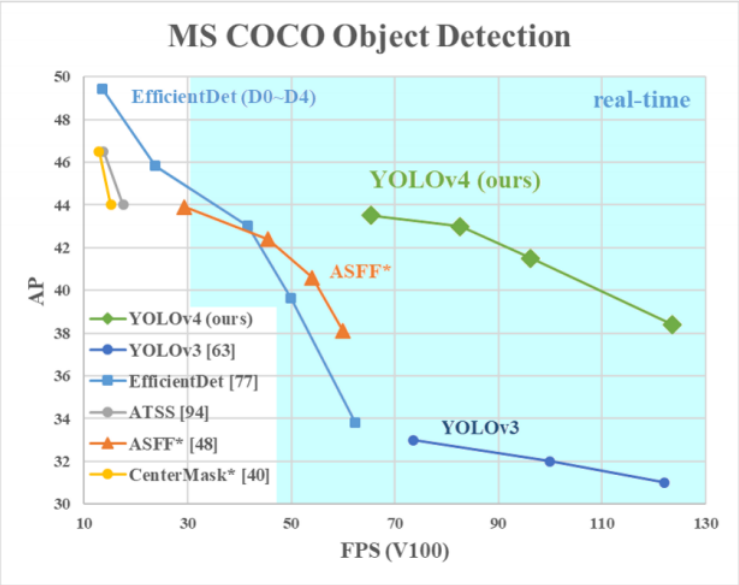

Figura 5.17: Comparación de la propuesta YOLOv4 y otros detectores de objetos de última generación. YOLOv4 mejora la precisión media (AP) y los FPS de YOLOv3 en un 10 % y un 12 %, respectivamente. Fuente[8].

es como mejorar la precisión de nuestras redes en las clases peatón y ciclista sin perder precisión a la hora de localizar un clasificar objetos de la clase coche.

Entre las soluciones presentadas, en esta sección detallaremos como hemos aumentado el número de objetos por clase, logrando así un mayor conjunto de entrenamiento. La idea es que este aumento de objetos de las tres clases permita aumentar la capacidad de detección y localización de las redes estudiadas. Partiendo del dataset KITTI, donde realizamos una separación aleatoria entre los conjuntos de entrenamiento y test, teniendo así un 80 % para entrenamiento y un 20 % para test.

Los datasets utilizados para aumentar las imágenes de las clases coche, peatón y ciclista deben ser similares a las imágenes recolectadas en el dataset KITTI. Como veremos en esta sección, estos conjuntos han sido creados basados en un entorno de conducción, es decir, grabadas desde un coche. Posteriormente las imágenes son filtradas de forma que tomemos aquellas imágenes grabadas durante el día y en buenas condiciones climatológicas. La razón de este filtro se debe a que el objetivo es mejorar la precisión sobre el conjunto de test del dataset KITTI, luego el conjunto de entrenamiento debe ser similar al conjunto de test. En particular, las imágenes del test cumplen estas condiciones. Somos conscientes que en un problema real de conducción autónoma un vehículo debería ser capaz de conducir de forma autónoma en cualquier situación climatológica adversa, sin embargo este proyecto se centra en una simplificación del problema real.

6.1. KITTI

El conjunto de datos KITTI [5] fue presentado en 2012. El dataset de detección de objetos consiste en 7481 imágenes a color de diferentes escenas, almacenadas como imágenes *png*. El tamaño de las imágenes es aproximadamente 375×1275, ya que el tamaño no es fijo y cambia ligeramente a lo largo del dataset. Las imágenes fueron tomadas por una cámara colocada sobre un coche mientras se conducía por zonas rurales y urbanas en buenas condiciones climatológicas y durante el día. El vehículo que grabó las imágenes es el siguiente:

Figura 6.1: Coche que realizó los vídeos en los que se basa el dataset KITTI. Fuente [5]

En cada imagen se etiquetan todos los objetos que pertenezcan a alguna de la siguiente categorías:

- Car
- Van
- Truck
- Pedestrian
- Person(sitting)
- Cyclist
- Tram
- Miscelanous

En la figura 6.2 vemos el número de instancias por clase de objeto:

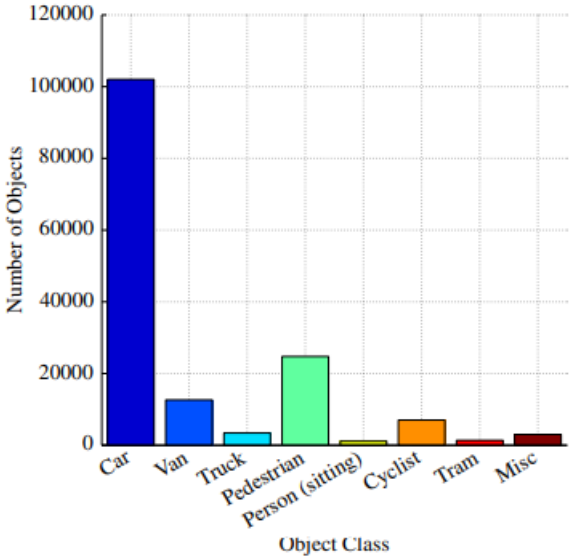

Figura 6.2: Número de instancias por clase en el dataset KITTI.Fuente [5]

Es fácilmente apreciable el desbalanceo de clases presente en el conjunto de datos KITTI. En nuestro caso, restringimos el problema a las clases coche, peatón y ciclistas. Surge de forma natural la necesidad de aumentar el conjunto de imágenes de entrenamiento con presencia de peatones y ciclistas, pues se observa un claro desequilibrio entre el número de objetos por clase. Además de la etiqueta de clase, también contienen información de las cajas delimitadoras en 2D, cuyas coordenadas son expresadas en píxeles. Toda la información de las etiquetas es almacenada en un archivo .txt que se crea para cada imagen. Un ejemplo de este dataset podemos verlo en la siguiente imagen:

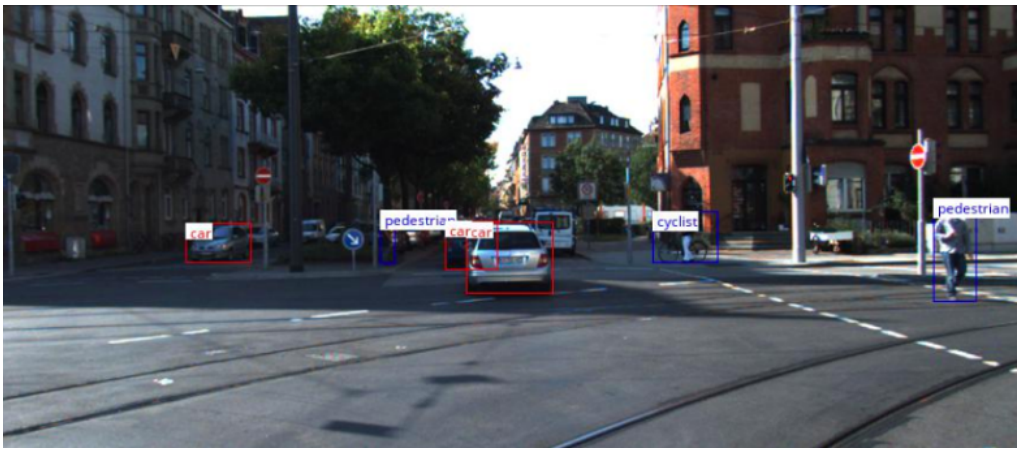

Figura 6.3: Ejemplo etiquetado del dataset KITTI.

En la figura 6.3 podemos observar las tres clases que vamos a estudiar en este dataset, es decir: coche, peatón y ciclista.

6.2. Tshingua-Daimler

El conjunto de datos Tshingua-Daimler [64] fue creado en 2016. El dataset de detección de objetos de entrenamiento cuenta con 9741 imágenes de con anotaciones para la clase ciclista. En concreto, solo se han etiquetado aquellos ciclistas que son totalmente visibles y cuyo área total sea mayor a 60 píxeles. Las imágenes son almacenadas como archivos .png. Por otro lado, contamos con un archivo de etiqueta por cada imagen que contiene tanto la clase como las cajas delimitadoras de cada objeto ciclista identificado en la imagen. Las etiquetas son almacenadas en formato .json. Un ejemplo de este dataset sería:

Figura 6.4: Ejemplo etiquetado del dataset Tshingua-Daimler.

6.3. Berkeley Deepdrive Dataset

El conjunto de datos Berkely Deepdrive Dataset [19] también conocido como BDD, fue puesto a disposición de la comunidad investigadora en 2018. Se trata de un dataset masivo grabado en distintas áreas geográficas de Estados Unidos. La figura 6.5 muestra la distribución geográfica de la grabación de imágenes:

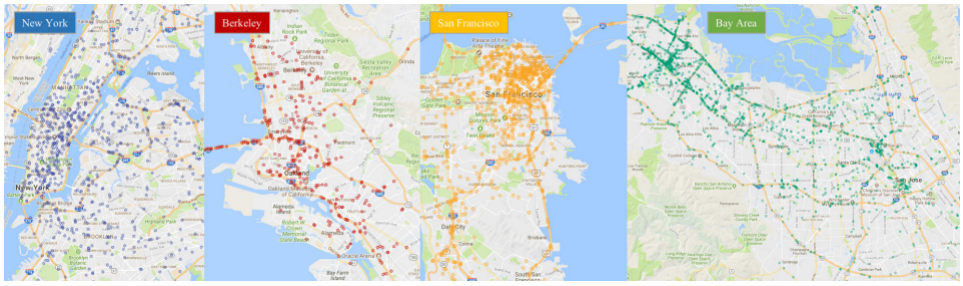

Figura 6.5: Distribución geográfica de las fuentes de datos. Cada punto representa el comienzo de la grabación de un vídeo. Fuente [19]

Este dataset consiste en 100000 vídeos. Cada vídeo tiene una duración de 40 segundos, esta grabado en 720p y 30 frames por segundo. Las imágenes son almacenadas en formato .jpg y las etiquetas en un archivo .json. En esta sección veremos como BDD es uno de los conjuntos de imágenes mas completos y variados que hay actualmente disponibles para la comunidad investigadora. Esta base de datos cubre diferentes condiciones climatológicas como pueden ser: soleado, nublado, nevado o lluvioso tanto como en diferentes momentos del día incluyendo de esta forma imágenes de día y de noche. Comparamos este conjunto con otros en la siguiente tabla:

	KITTI	Cityscapes	ApolloScape	Mapillary	BDD100K
# Sequences	22	~50	4	N/A	100,000
# Images	14,999	5000 (+2000)	143,906	25,000	120,000,000
Multiple Cities	No	Yes	No	Yes	Yes
Multiple Weathers	No	No	No	Yes	Yes
Multiple Times of Day	No	No	No	Yes	Yes
Multiple Scene types	Yes	No	No	Yes	Yes

Figura 6.6: Comparación con otros dataset de conducción. Fuente [19]

Observamos como el dataset BDD es enorme y muy diverso. Además, las etiquetas realizan diferencias entre la situación de las imágenes, distinguiendo de esta forma entre seis tipos de escenas:

- City Street
- Tunnel
- Highway

- Residential
- Parking
- Gas Station

Otra ventaja de este dataset que contiene muchas instancias por objeto clasificado, contando con una presencia muy alta de coches y peatones. En la figura 6.6 se muestra el número de instancias por clase:

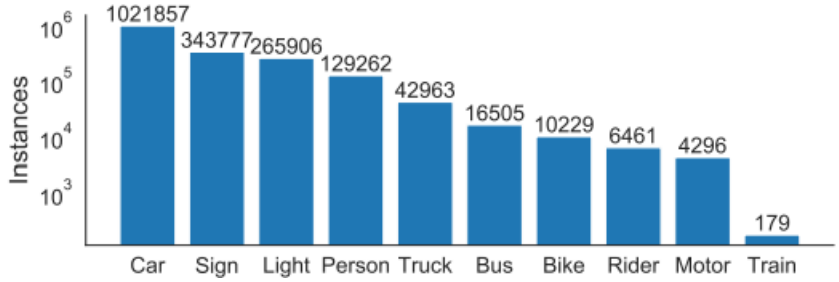

Figura 6.7: Número de instancias por objeto. Fuente [19]

En nuestro caso, hemos filtrado el dataset quedándonos únicamente con imágenes de día en buenas condiciones climatológicas, es decir, de días soleados e imágenes diurnas. Además, solo hemos tomado aquellas con presencia de peatones y coches, pues estas son las clases que nos interesan para nuestro trabajo. En la siguiente imagen mostramos un ejemplo del Berkeley Deepdrive Dataset:

Figura 6.8: Ejemplo etiquetado del dataset BDD.

En la figura 6.8 vemos un ejemplo de las dos clases cuya imágenes incorporaremos al dataset final para mejorar la capacidad de predicción de las redes analizadas.

6.4. Conjuntos de datos creados

El conjunto de entrenamiento está compuesto por los datasets presentados en las secciones anteriores. Aumentando el número de imágenes se pretende conseguir una mejor capacidad de localización y clasificación de objetos. El conjunto de datos de entrenamiento está compuesto por:

- El 80 % de las 7481 imágenes del KITTI. Por tanto, contendrá 5981 imágenes del KITTI.
- Las imágenes de ciclistas del Tshingua-Daimler.
- Las imágenes de peatones y vehículos del Berkeley Deepdrive dataset, en condiciones climatológicas buenas, es decir, despejado y de día.

En la siguiente tabla presentamos el número de coches, peatones y ciclistas presentes en el conjunto de entrenamiento:

Conjunto de datos	**Coche**	**%Total**	**Peatón**	**%Total**	**Ciclista**	**%Total**	**Total por conjunto**
KITTI	32184	87,10 %	3485	9,43 %	1281	3,47 %	36950
Tshingua-Daimler					16202	100 %	16202
BDD	139407	89,26 %	16777	10,74 %			156184
Total por clase	171591	81,97 %	20262	9,68 %	17483	8,35 %	209336

Por otro lado, hemos creado un conjunto de test para evaluar las capacidades de inferencia de las redes estudiadas. Este conjunto está compuesto por el 20 % de las imágenes del KITTI, que se corresponde con las 1500 imágenes restantes. En la siguiente tabla se muestra el número de cada tipo de objeto presente en este conjunto:

Conjunto de datos	**Coche**	**% Total**	**Peatón**	**% Total**	**Ciclista**	**% Total**	**Total por conjunto**
KITTI Test	7853	85,35 %	1002	10,89 %	346	3,76 %	9201

Observando ambas tablas vemos como el conjunto de entrenamiento tiene una mayor presencia de ciclistas y peatones, con la consecuencia de tener un menor porcentaje de coches presentes. Se ha realizado con la idea de conseguir aumentar la precisión en la detección de ciclistas y peatones sin perder capacidad de detección de coches.

7. Dispositivos hardware utilizados

En este apartado vamos a presentar los distintos dispositivos hardware que más tarde se utilizarán para realizar la inferencia de los distintos modelos y algunas de sus características más importantes, siendo de especial interés en el ámbito de la conducción autónoma

la potencia de diseño térmico (TDP) que 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.

7.1. CPU Intel® Xeon® E3-1225 v3

La CPU utilizada será el modelo Intel® Xeon® E3-1225 v3, un modelo desarrollado y fabricado por Intel de la gama Xeon, pensados para servidores y centros de procesamiento. Algunas de las características más relevantes son:

- Núcleos: 4
- Subprocesos: 4
- Frecuencia básica: 3.20 GHz
- Frecuencia turbo: 3.50 GHz
- **TDP: 90W**
- Gráficos de procesador:
  - Frecuencia de base: 350 MHz
  - Frecuencia dinámica máxima: 1.20 GHz

7.2. GPU GEFORCE GTX 980

Una de las GPUs utilizadas será la GTX 980, una GPU de alto rendimiento y a la que haremos referencia como *GPU AR* (GPU Alto Rendimiento) para diferenciarla de la GPU integrada de bajo consumo que presentaremos más adelante. Algunas de las especificaciones más relevantes son:

- Núcleos CUDA: 2048
- Frecuencia normal: 1126 MHz
- Frecuencia acelerada: 1216 MHz
- Memoria: 4 GB DDR5
- **TDP: 165W**

7.3. GPU UHD Intel® 620

La otra GPU utilizada será la gráfica UHD Intel® 620 integrada en la CPU Intel® Core™ i7-8550U y a la que haremos referencia simplemente como *GPU* para diferenciarla de la GPU de alto rendimiento. Algunas de las especificaciones más relevantes tanto de la CPU como de la GPU integrada son:

- Núcleos: 4
- Subprocesos: 8
- Frecuencia básica: 1.80 GHz
- Frecuencia turbo: 4.00 GHz
- **TDP: 15W**
- Gráficos de procesador:
  - Frecuencia de base: 300 MHz
  - Frecuencia dinámica máxima: 1.15 GHz

7.4. Intel® Neural Compute Stick 2

El último de los dispositivos que utilizaremos para realizar la inferencia de las distintas redes será Intel® Neural Compute Stick 2. Se trata de un dispositivo USB basado en la unidad de procesamiento de visión (VPU) Intel Movidius Myriad X (VPU) y que cuenta con soporte del conjunto de herramientas de OpenVINO. NCS2 acelera de forma asequible la velocidad de desarrollo de aplicaciones para inferencias basadas en redes neuronales profundas y simplifica la creación de prototipos. Haremos referencia a este dispositivo hardware como *Myriad* a partir de ahora. Algunas de las características más relevantes son:

- Frecuencia base del procesador: 700 MHz
- **TDP: 1W**

8. Medidas de precisión en la detección de objetos

La medida mas popular utilizada en el campo de investigación de los detectores de objetos para describir la precisión de un modelo es mAP. En concreto, significa *mean average precision* y es una métrica que requiere la explicación de otras cuatro medidas: precisión, recall, IOU y AP.

8.1. Precisión y Recall

Comenzamos describiendo las dos primeras métricas necesarias para definir mAP:

1. Precisión: mide como de precisas son las predicciones de los modelos mediante un porcentaje de predicciones correctas. Una definición más matemática sería:

$$Precision = \frac{TP}{TP + FP}$$

donde TP representa *true positive*, FP representa *false positive* y FN representa *false negative*.

2. Recall: mide la capacidad del modelo de detectar casos positivos, en este caso, objetos que pertenezcan a alguna de las clases del dataset. Una definición mas matemática sería:

$$Recall = \frac{TP}{TP + FN}$$

   por tanto **recall** es un porcentaje de los casos positivos correctamente acertados.

8.2. IOU

Además de los conceptos de precision y recall, necesitamos definir IOU para poder explicar mAP. *Intersection Over Union* o IOU mide el área de intersección entre dos cajas delimitadoras. Matemáticamente seria el cociente del área de intersección y el área de la unión de las cajas:

$$IOU = \frac{area\ de\ interseccion}{area\ total\ de\ la\ union}$$

Esta métrica nos permite establecer cuando una predicción realizada por un objeto es buena o no. Habitualmente se establece un umbral a partir del cual damos por buena una predicción, en este proyecto hemos tomado por buenas aquellas predicciones que superaran el 0,5 de este valor. La siguiente figura muestra la intersección entre una caja delimitadora predicha y la real etiquetada en la imagen, conocida por *ground truth*:

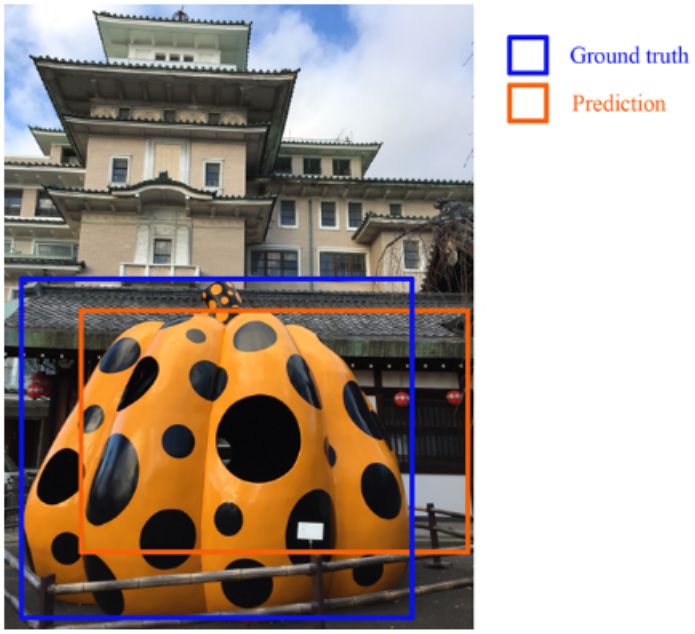

Figura 8.1: Intersection Over Union. Fuente [27]

8.3. AP y mAP

Por otro lado introducimos *average precision* o AP. Si representamos en un gráfico tomando como ejes y = precisión y eje x = recall, AP es el área bajo la curva, es decir, bajo el gráfico generado por ambas métricas. Un ejemplo lo vemos en la siguiente gráfica:

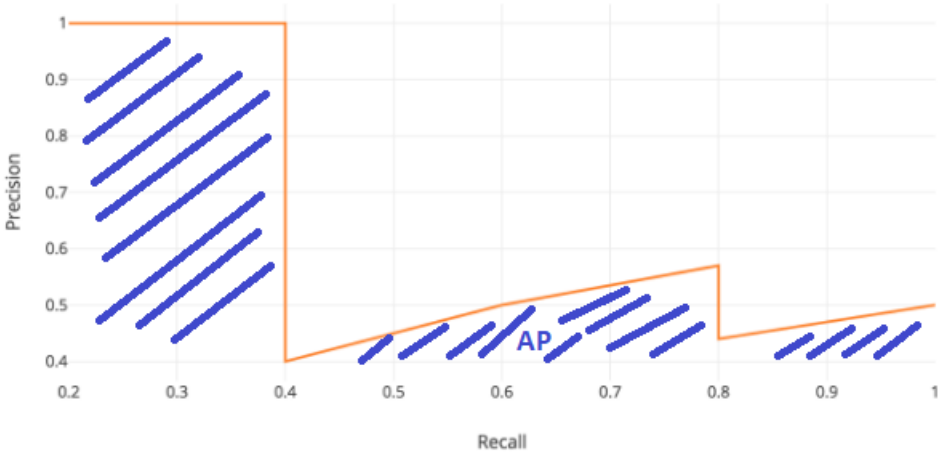

Figura 8.2: Area bajo la curva de la gráfica precición-recall. Fuente [27]

La definición matemática mas conocida es:

$$AP = \int_0^1 p(r)dr$$

donde $p(r)$ es la curva formada por el gráfico precisión-recall. Finalmente definimos mAP como la media de AP para un umbral de IOU dado. Un umbral clásico para IOU es [0.5, 0.05, 0.95], de esta forma mAP será la media de los valores AP para cada valor de IOU, que va desde 0.5 a 0.95 con un paso de 0.05.

9. Frameworks

En este capítulo vamos a presentar brevemente los frameworks utilizados tanto para el entrenamiento de los modelos como para realizar la inferencia en distintos dispositivos

9.1. Tensorflow

TensorFlow es una plataforma o framework de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que les permite a los investigadores impulsar un aprendizaje automático innovador y, a los desarrolladores, compilar e implementar con facilidad aplicaciones con tecnología de aprendizaje automático. Fue originalmente desarrollado por el equipo de *Google Brain*, reemplazando a su predecesor de código cerrado, *DistBelief*.

Tensorflow pone a nuestra disposición una serie de recursos que facilitan la creación, entrenamiento y validación de modelos para el aprendizaje automático:

- Modelos y conjuntos de datos: Modelos previamente entrenados y conjuntos de datos que Google y la comunidad han desarrollado
- Herramientas: Ecosistema de herramientas para ayudar al usuario con TensorFlow
- Bibliotecas y extensiones: Bibliotecas y extensiones creadas en TensorFlow
- Programa de certificación de TensorFlow
- Aprende AA: Recursos educativos para aprender los aspectos básicos del AA con TensorFlow

9.1.1. Object Detection API

En nuestro caso, utilizaremos en concreto la API de detección de objetos [60] para el entrenamiento de los siguientes modelos o redes neuronales convolucionales proporcionadas

por Tensorflow previamente entrenados sobre el conjunto generalista COCO[10]. Estos modelos los podemos encontrar en el object detection model zoo [59] de tensorflow.

- SSD mobilenet inception V2 COCO
- SSDLite mobilenet inception V2 COCO
- Faster R-CNN inception V2 COCO

Dentro de esta API encontramos varios elementos que usaremos durante los entrenamientos:

- **model_main.py**: Se trata del script de python que se utiliza para el entrenamiento de las redes convolucionales para la detección de objetos. Implementa además la evaluación del modelo sobre un conjunto de validación cada cierto tiempo para ver la evolución del modelo y no solo de la función de pérdida. Entre sus parámetros podemos encontrar:
  - *pipeline_config_path*: Ruta al archivo de configuración, que pasamos a ilustrar brevemente más abajo.
  - *model_dir*: Ruta donde queremos que se guarde el modelo reentrenado junto a los últimos checkpoints generados.
  - *num_train_steps*: Número máximo de pasos a realizar en el entrenamiento.
  - *sample_1_of_n_eval_examples*: Tamaño relativo del dataset de validación tomado para cada evaluación
- **model zoo** [59]: Carpeta con los modelos preentrenados ofrecidos por el framework. Aquí podemos encontrar los modelos anteriormente mencionados.
- **samples**: En esta carpeta encontramos algunos ejemplos. Nos será de especial utilidad la subcarpeta **configs**, donde podemos encontrar plantillas de configuración para las distintas redes.
- **metrics**: Carpeta con distintas métricas a usar. En nuestro caso utilizaremos las métricas de COCO.
- **dataset_tools**: Carpeta donde podemos encontrar distintas herramientas. En nuestro caso utilizaremos el script *create_kitti_tf_record* para generar los archivos tensorflow records, que utilizará el framework tanto para el entrenamiento como las validaciones.

Cabe destacar que, como hemos mencionado anteriormente, nuestras imágenes deben ser transformadas previamente a **Tensorflow records** para poder ser procesadas por el framework. El formato TFRecord es un formato simple para almacenar una secuencia de registros binarios, de manera que sea más eficiente el almacenamiento y lectura en *batches* o lotes de los mismos.

Pasamos a describir ahora brevemente la **composición del archivo de configuración** a ser usado en el reentrenamiento mediante el script model_main.py. Este archivo tiene estructura de JSON y nos encontramos los siguientes campos:

- model: Nos permite especificar el modelo que vamos a reentrenar junto a los parámetros específicos del modelo y otro más generales como el número de clases, la generación de anchors, las funciones de pérdida y post-procesamiento.
- train_config: En este campo podemos especificar parámetros como el número de imágenes por lote, el optimizador a ser utilizado en el reentrenamiento (RMPS prop, ADAM, etc), el checkpoint del que partir si queremos realizar un fine tuning y no un reentrenamiento desde 0, data augmentation y dónde se encuentran nuestras imágenes en formato tf record para el reentrenamiento.
- eval_config: Nos permite especificar las métricas a utilizar (en nuestro caso COCO), el número de imágenes a utilizar en la evaluación, número de imágenes a visualizar en cada evaluación, incluir métricas por cada categoría y dónde se encuentran nuestras imágenes en formato tf record para las evaluaciones.

9.1.2. Tensorboard

Además, tensorflow pone a nuestra disposición la herramienta Tensorboard. TensorBoard proporciona la visualización y las herramientas necesarias para experimentar con el aprendizaje automático, como por ejemplo:

- Seguir y visualizar métricas, como la pérdida y la exactitud o accuracy.
- Visualizar el grafo del modelo (operaciones y capas)
- Mostrar imágenes

Podemos ver un ejemplo de su interfaz gráfica en la figura 9.1.

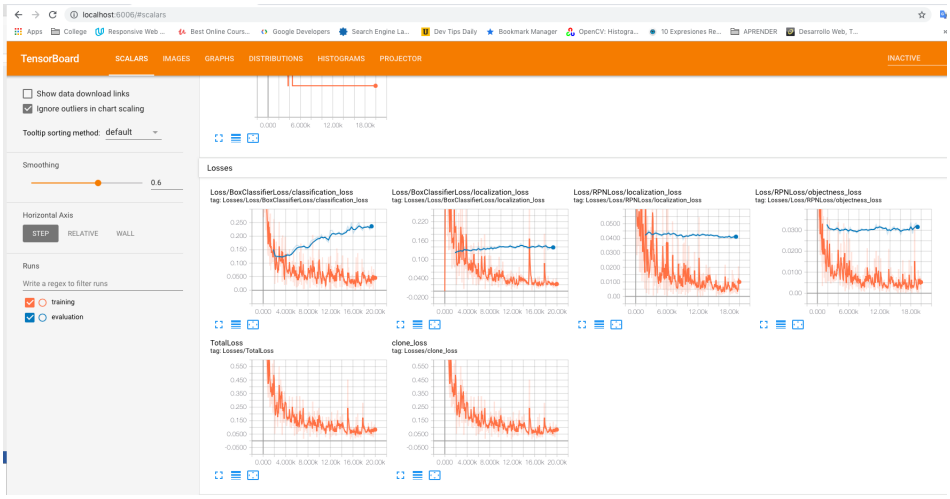

Figura 9.1: Interfaz gráfica de Tensorboard.

Usaremos esta herramienta para supervisar el aprendizaje de nuestras redes, ya que podremos ir viendo en cada iteración las funciones de pérdida, el accuracy y algunas imágenes mostrando las detecciones que realiza nuestra red.

9.2. Openvino toolkit

OpenVINO [43] (Open Visual Inference and Neural network Optimization) toolkit es un kit de herramientas gratuito que facilita la optimización de un modelo de Deep Learning o Aprendizaje Profundo procedente de diferentes frameworks y la implementación y ejecución mediante un motor de inferencia en el hardware de Intel. El kit de herramientas tiene dos versiones: OpenVINO toolkit, que es apoyado por la comunidad de código abierto y la distribución de Intel(R) de OpenVINO toolkit, que es apoyado por Intel.

El principal componente de este kit de herramientas es el Deep Learning Deployment Toolkit (DLDT), que nos ofrece dos funcionalidades principales:

1. Generar archivos IR (Representación Intermedia) haciendo uso del **optimizador de modelos** a partir de un modelo entrenado o uno público.
2. Ejecutar la inferencia haciendo uso del **motor de inferencia** en los dispositivos hardware especificados

9.2.1. Características principales

- Permite realizar inferencia sobre modelos de aprendizaje profundo basados en redes neuronales convolucionales

- Admite la ejecución heterogénea a través de una CPU Intel®, Gráficos Integrados Intel®, Intel® FPGA, Intel® Movidius™ Neural Compute Stick, Intel® Neural Compute Stick 2 e Intel® Vision Accelerator Design con VPU Intel® Movidius™.
- Acelera el tiempo de comercialización a través de una biblioteca fácil de usar con funcionalidades de visión por ordenador y núcleos preoptimizados
- Incluye llamadas optimizadas para los estándares de visión artificial, incluyendo OpenCV* y OpenCL™

9.3. Darknet

Darknet [45] es un framework o entorno para redes neuronales de código abierto escrito en C y CUDA. Es rápido, fácil de instalar y soporta computación de CPU y GPU. En nuestro caso utilizaremos la versión de AlexeyAB, que podemos encontrar en su repositorio de github[16].

Entre las funcionalidades que nos ofrece este entorno, destacan las redes neuronales YOLO (You Only Look Once, sólo miras una vez) que se caracterizan por aplicar una sola red neuronal a toda la imagen, la cual divide la imagen en regiones más pequeñas, luego predice las cajas delimitadoras y asigna probabilidades para cada una de estas regiones. Éste hecho hace que la inferencia en este tipo de estructuras sea más rápida que en otro tipo de estructuras de redes convencionales.

Proporciona distintas funcionalidades, entre las que encontramos:

- **darknet.exe detector**: Es el script principal e implementa las siguientes funcionalidades en función del segundo parámetro que introduzcamos:
  - train: Entrenamiento
  - valid: Validación
  - demo: Nos permite ejecutar la red sobre un archivo de video o en tiempo real utilizando una cámara.
- **cfg**: Carpeta con plantillas de configuraciones sobre distintas redes.
- **scripts**: Carpeta con diferentes scripts que nos seran de utilidad, como por ejemplo *gen_ anchors.py*, el cual utiliza K-Means para calcular de forma óptima las dimensiones de los anchors que utilizaran las distintas capas de la red.

9.4. Módulos

El kit de herramientas se encuentra encapsulado diferentes módulos o **componentes**. Los principales y de los que haremos uso serán los siguientes:

- **Deep Learning Deployment Toolkit**: Es el componente principal. Se subdivide en:
  - Optimizador de modelos: Permite importar modelos y prepararlos para una ejecución óptima mediante el motor de inferencia. El Optimizador de Modelos importa, convierte (a IR) y optimiza modelos de frameworks como Caffe, TensorFlow, MXNet, Kaldi y ONNX.
  - Motor de inferencia: Una API unificada para permitir la inferencia de alto rendimiento en distintos tipos de hardware.
  - Samples: Un conjunto de aplicaciones sencillas de consola que demuestran cómo utilizar el motor de inferencia.
  - Tools: Herramientas adicionales
- **Open Model Zoo**: Nos ofrece un conjunto de demos para utilizar el motor de inferencia, herramientas adicionales para descargar modelos y **medir la precisión** de los mismos y un conjunto de modelos preentrenados junto a documentación sobre los mismos.
- **Post-Training Optimization tool**: Herramienta que permite calibrar modelos y realizar inferencia usando enteros de 8 bits.
- **Deep Learning Workbench**: Entorno gráfico basado en la web que permite utilizar fácilmente varios componentes sofisticados del kit de herramientas OpenVINO™. Podemos ver un ejemplo de la interfaz gráfica en la figura 9.2.

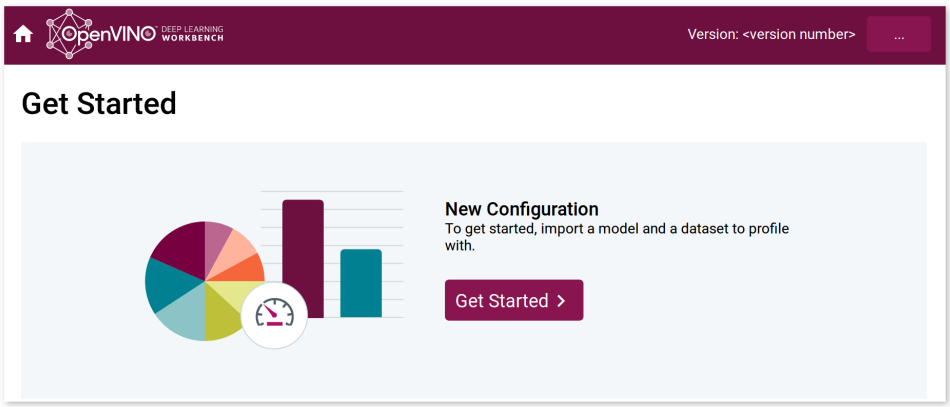

Figura 9.2: Interfaz gráfica de la herramienta Deep Learning Workbench.

10. Experimentos realizados

Hablar de la mejora de la precisión y reducción del error gracias a distintas técnicas como data augmentation, definir nuevos anchors, hard negative mining etc

En algun lado hay que presentar las medidas de precision estudiadas, IOU,MAP, etc

10.1. Anchor Boxes

En la etapa de predicción de múltiples objetos en una imagen, las redes neuronales como SSD o YOLO, realizan miles de predicciones y solo muestran aquellas que deciden que son un objeto.

El *estado del arte* de los sistemas de detección de objetos funciona de la siguiente manera:

1. Crea miles de *cajas por defecto* o **anchor boxes** para cada predictor de forma que represente la forma, tamaño y localización del objeto que se especializa.
2. Para cada anchor box, calcula que objeto de la imagen tiene la caja delimitadora con mayor IOU.
3. Si el IOU máximo es mayor del 50 %, entonces el anchor box que predice el objeto de la imagen que obtuvo mayor IOU.
4. En caso contrario, el anchor box decidirá que no hay objeto.

Este método funciona bien en la práctica, pero como vemos las cajas por defecto son totalmente vitales para que nuestro detector de objetos tenga una buena capacidad de predicción. Por ejemplo si observáramos los anchors boxes generados por nuestra red sería algo así:

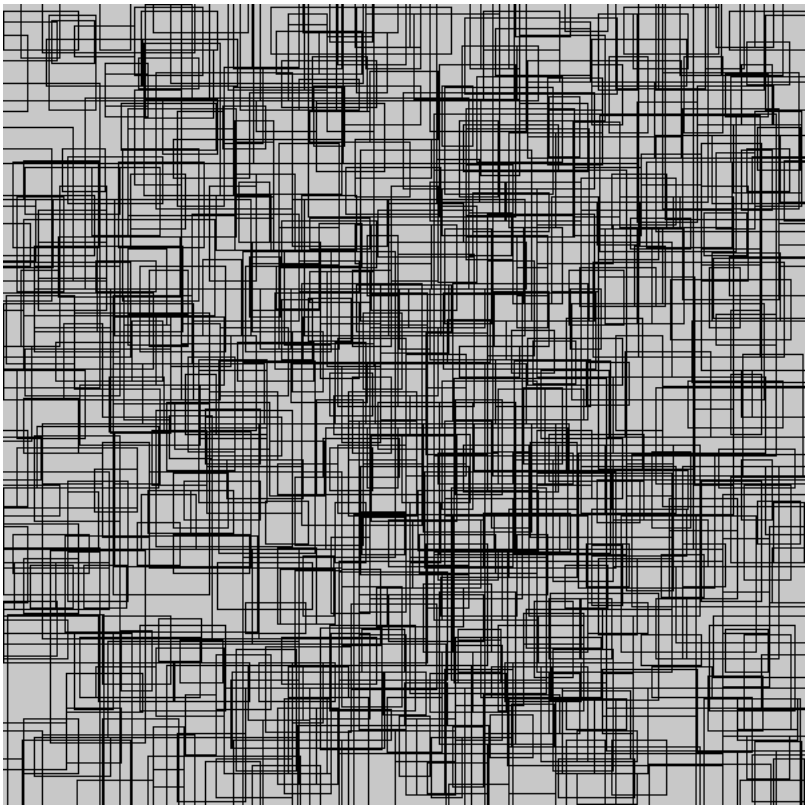

Figura 10.1: Anchor Boxes generados por un detector de objetos. Fuente [4]

Y en esta imagen solo estamos mostrando un bajísmo porcentaje de los anchor boxes existentes. Usar los anchor boxes por defecto puede crear predictores que están demasiado especializados y los objetos de la imagen serán incapaces de lograr un IOU mayor de 50 % con alguna de las cajas por defecto generadas. De forma que, la red neuronal no sabrá que esos objetos existen y nunca aprenderá de ellos. Por ello debemos de adaptar las cajas por defecto para tener una forma y tamaño ideal para nuestros objetos, logrando cajas por defecto mucho mas ajustadas. Un ejemplo de esas modificaciones sería:

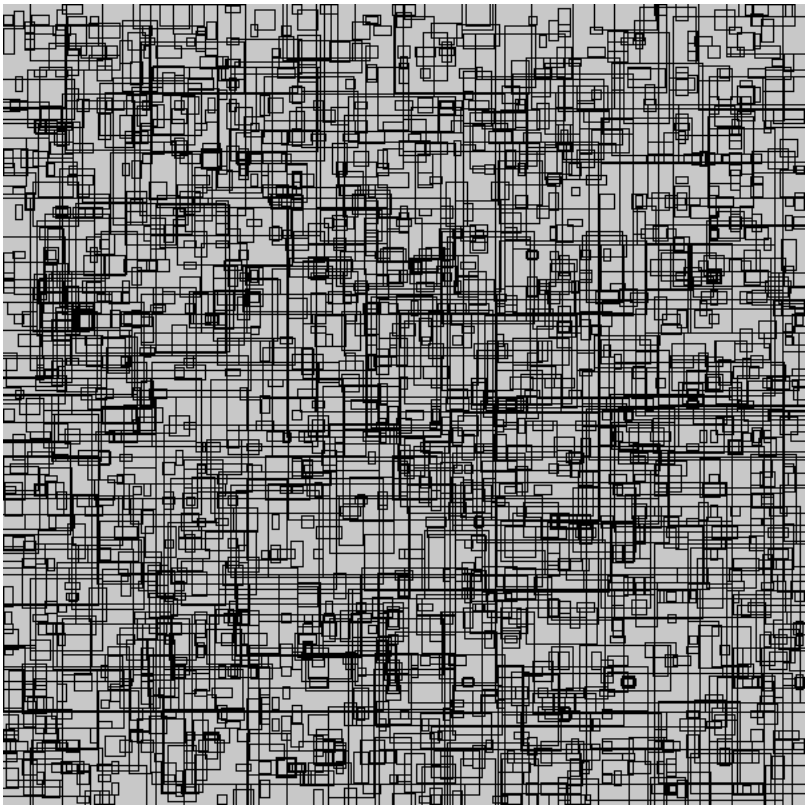

Figura 10.2: Anchor Boxes **adaptados** generados por un detector de objetos. Fuente [4]

Como conclusión para mejorar la configuración de las cajas por defecto estudiaremos:

- Menor tamaño de los objetos que queremos detectar.
- Mayor tamaño de los objetos que queremos detectar.
- Determinar la forma de las cajas por defecto, por ejemplo mas ancha y baja para detectar un coche o mas delgada y alta para detectar peatones.

10.2. Data augmentation

Image augmentation es un técnica que permite crear variaciones artificiales sobre imágenes del dataset para expandir el conjunto de imágenes existentes. De esta forma, creamos nuevas y diferentes imágenes a partir de las imágenes de partida creando un dataset mas diverso y rico en imágenes.

Esta técnica [7] se realiza aplicando diferentes transformaciones a las imágenes como darle la vuelta, hacer zoom, rotaciones entre otras. Por ejemplo, en la siguiente figura aplicamos distintas trasformaciones a la imagen de un león:

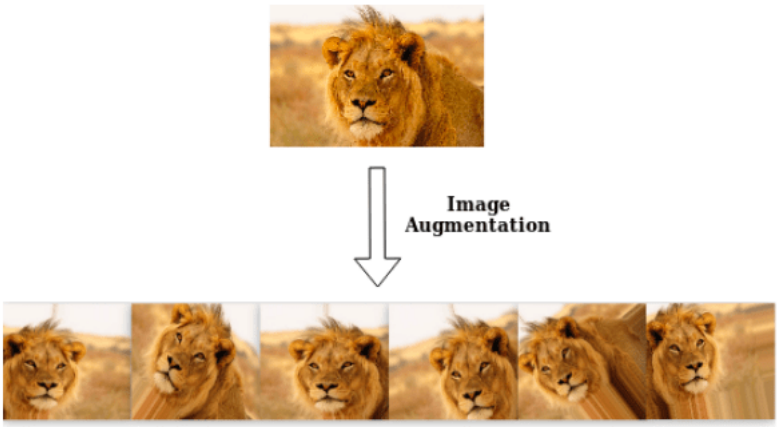

Figura 10.3: Ejemplo de Data Augmentation sobre la imagen de un león en la sabana. Fuente: [7]

Las redes neuronales convolucionales necesitan una enorme cantidad de imágenes para entrenar de forma efectiva el modelo. Esta técnica permite incrementar el rendimiento y robustez del modelo gracias a una mejor capacidad de generalización y una reducción del overfitting.

10.3. Hard Negative Mining

Tras la etapa de emparejamiento(*matching*), la mayor parte de las cajas de detección por defecto son negativas, especialmente si el número de cajas por defecto es elevado. Un caja delimitadora es negativa si no identifica objetos dentro de esta, ya sea porque la identificación de un objeto es parcial o porque simplemente no contenga objetos dentro de esta. Esto produce un gran desbalanceamiento entre el número de muestras negativas y positivas a la hora de realizar el entrenamiento, provocando un aumento en la dificultad del aprendizaje y una mala convergencia debido a un elevado número de falsos positivos.

Con el fin de abordar este problema de falsos positivos, se usará *hard negative mining* [3]. Cada red afronta este problema de forma distinta, ya sea acotando el ratio de muestras negativas y positivas o restringiendo el número máximo de muestras negativas. En el paper original del detector SSD [35], proponen que en lugar de usar todos los ejemplos negativos, ordenarlos por *highest confidence loss* para cada caja de detección por defecto. Después tomar los que se encuentran en una posición mas alta de forma que el ratio entre muestras negativas y positivas sea como mucho 3:1.

11. Resultados del Reentrenamiento

En este capítulo se presentan los resultados de la fase de entrenamiento. Se distinguen los resultados conseguidos tras el entrenamiento de las redes sobre el KITTI y sobre el conjunto de entrenamiento completo.

11.1. Resultados de la red SSD Inception V2

En esta subsección vamos a exponer los resultados de partida logrados por la red SSD Inception sobre los conjuntos de datos KITTI y el conjunto de datos completo, formado por KITTI, Tshingua-Daimler y Berkeley Deepdrive Dataset. Además se realizará una comparación del rendimiento por clase logrado por la red en fase de reentrenamiento.

11.1.1. Resultados SSD Inception V2 sobre KITTI

Los resultados de la red sobre el conjunto de datos KITTI son los siguientes:

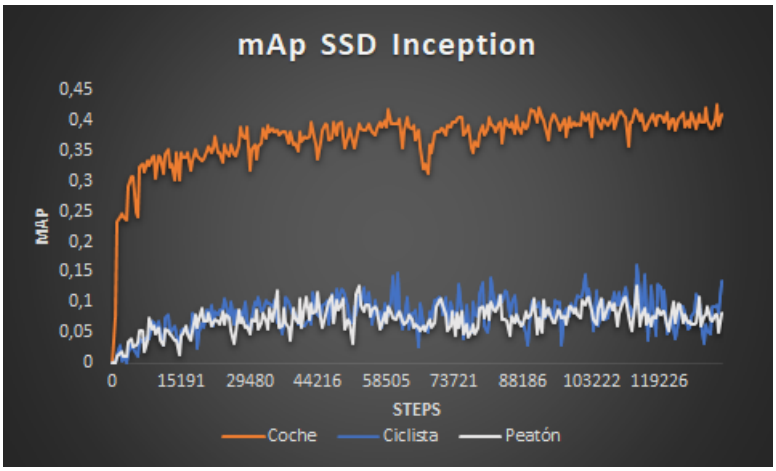

Figura 11.1: mAP de las clases coche, ciclista y peatón

En la figura 11.1, vemos que la precisión de la red sobre el conjunto de datos es muy pobre. En particular, detecta mas coches que ciclistas y peatones. Para la clase coche el mAP aumenta progresivamente a lo largo de las 130000 iteraciones hasta establecerse entorno a 0.4, es decir, solo es capaz de identificar en esta fase aproximadamente un 40 % de los coches.

Por otro lado, el valor de mAp logrado para ciclistas y peatones es muy bajo, siendo en ambos casos por debajo del 0.1 . Estos valores concluyen que la red es incapaz de detectar ciclistas o peatones.

La principal causa de la pobre capacidad de detección de esta red es su arquitectura, ya que SSD realiza un redimensionamiento de las imágenes a 300×300, con el objetivo de reducir el tiempo de inferencia. Este hecho la hace muy interesante para la detección en tiempo real de objetos, característica que demanda el desafío de la conducción autónoma. Sin embargo, las imágenes del conjunto KITTI son panorámicas, de un tamaño 375×1275, luego este redimensionamiento provoca que la superficie final en píxeles que ocupa un peatón o un ciclista en la imagen redimensionada sea mínima, siendo incapaz de extraer las características que permite al modelo aprender a identificar las clases peatón y ciclista.

En general, podemos concluir que el rendimiento es demasiado pobre, aunque es mejor en el caso de la clase coche. La creación del conjunto de datos aumentado tiene el objetivo de mejorar la capacidad de detección de la red en todas las clases.

11.1.2. Resultados SSD Inception V2 sobre el dataset completo

Los resultados de la red sobre el conjunto de datos completo son los siguientes:

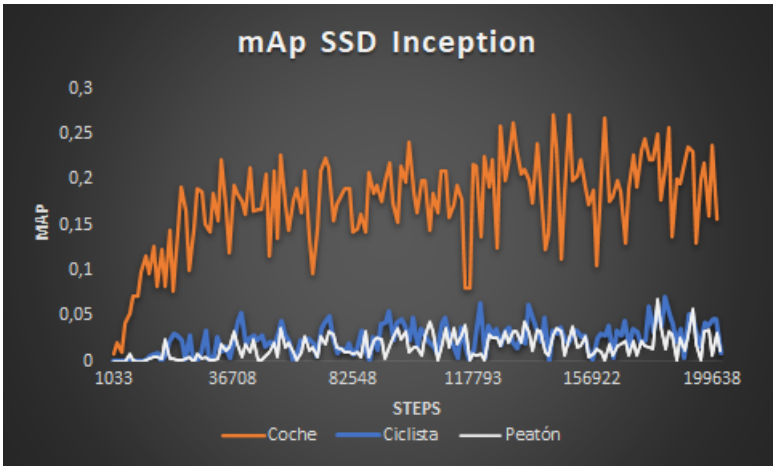

Figura 11.2: mAP de las clases coche, ciclista y peatón

En la figura 11.2 observamos una clara desmejoría respecto a la figura 11.1, donde los resultados ya eran bajos. En particular observamos como el valor mAp es un valor con gran volatilidad en la detección de coches, moviéndose en el intervalo $[0{,}15 - 0{,}25]$ en este caso. Esta gran variabilidad refleja que o bien la red está realizando overfitting, y por tanto si la imagen del offset escogida no es similar a la del conjunto de entrenamiento entonces no es capaz de detectar correctamente vehículos, o bien la red realiza underfitting y su capacidad de detección depende de cómo de similar sea la imagen del offset a las pocas imágenes que haya aprendido.

Se ha disminuido notablemente la capacidad de detección de coches, ciclistas y peato-

nes mediante el aumento de imágenes en el dataset de entrenamiento. Podemos observar como tras 20000 épocas la red alcanza una valor mAp de 0,2 en la detección de coches, mientras que es ligeramente inferior el valor 0,05 en ciclistas y peatones.

De esta forma podemos concluir que incrementar el conjunto de entrenamiento afecta notablemente a la capacidad de inferencia de la red, siendo por tanto una mala forma de aumentar la precisión. La disminución se debe a que aunque los conjuntos de datos tienen características comunes como haber sido grabados desde un vehículo o tratarse de imágenes diurnas, las diferencias son mayores, en concreto destaca:

- El tamaño de las imágenes es muy distinto, teniendo por un lado imágenes panorámicas en KITTI de tamaño 375×1275, a imágenes cuadradas de los conjuntos Berkeley Deepdrive Dataset y Tshingua-Daimler.
- El tamaño de los objetos del Berkely Deepdrive Dataset difiere mucho con los del KITTI, principalmente se debe a que BDD etiqueta todos los posibles objetos, incluso objetos al fondo complicados de identificar para un humano mientras que KITTI se centra en los objetos mas cercanos a la cámara. Todo esto provoca una disminución en la precisión del modelo a la hora de identificar los objetos al fondo, pues es prácticamente imposible tras el redimensionamiento de la imagen que sea capaz de lograrlo.

En la sección 12 veremos los resultados de la red entrenada en ambos conjuntos de datos a la hora de realizar inferencia sobre imágenes no vistas previamente del conjunto KITTI.

11.1.3. Comparación de resultados

En esta sección compararemos clase a clase los resultados obtenidos hasta el momento. Comenzaremos comparando la clase coche:

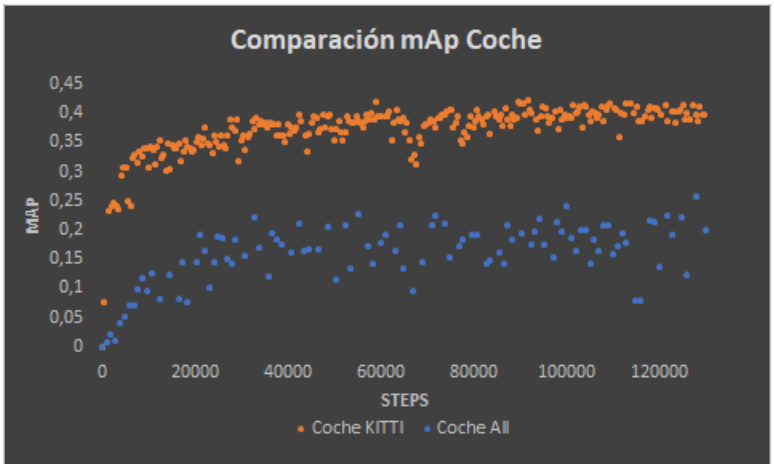

Figura 11.3: Comparación del mAP sobre la clase coche logrado por la red SSD Inception entrenada sobre KITTI y el dataset completo.

En la figura 11.3, vemos como la desmejoría ha sido del 50 % respecto de los resultados logrados con KITTI en esta clase. Además se ha aumentado la varianza de las estimaciones, añadiendo inestabilidad al modelo. Observamos la comparación en la clase ciclista:

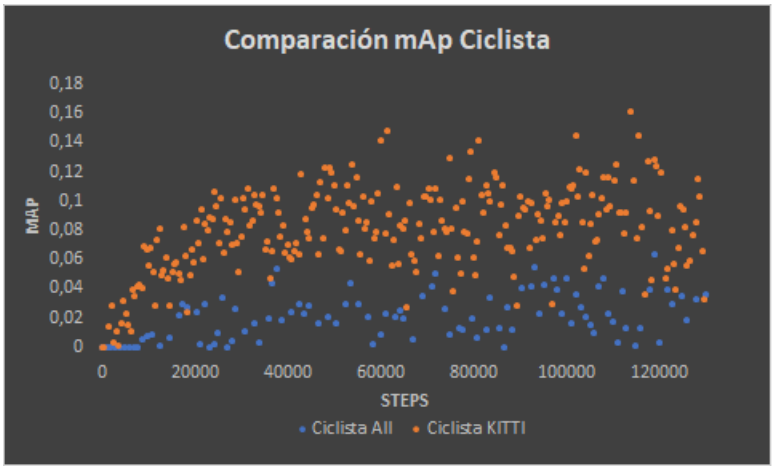

Figura 11.4: Comparación del mAP sobre la clase ciclista logrado por la red SSD Inception entrenada sobre KITTI y el dataset completo.

En la figura 11.4, se ve una clara disminución de la precisión con respecto a los resultados obtenidos con KITTI en la clase ciclista. No obstante, hay que mencionar que estos resultados se han logrado a costa de una disminución en la volatilidad de la capacidad de predicción de los ciclistas, siendo de esta forma un modelo mas estable. Este comportamiento posiblemente se deba a que el conjunto de datos Tshingua-Daimler cuenta con un conjunto de ciclistas que una vez redimensionados son distintos a los ciclistas del

conjunto KITTI. Como se ha comentado anteriormente, este hecho se debe a las diferencias existentes entre los conjuntos de datos de entrenamiento. Finalmente comparamos los resultados obtenidos sobre la clase peatón:

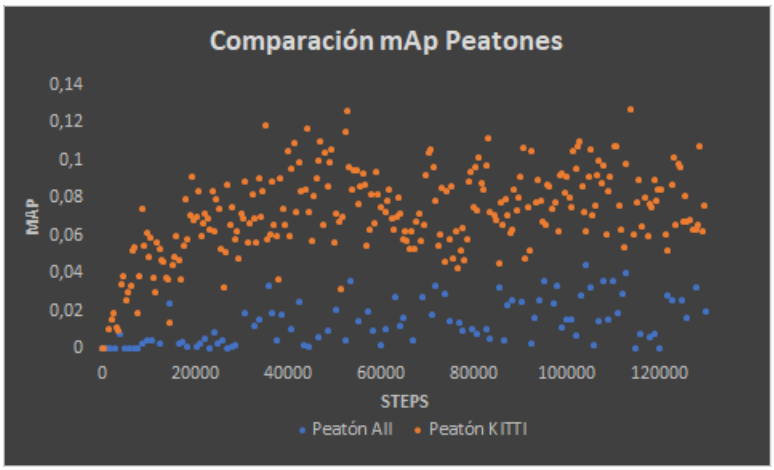

Figura 11.5: Comparación del mAP sobre la clase peatón logrado por la red SSD Inception entrenada sobre KITTI y el dataset completo.

En la figura 11.5, se muestra una disminución sustancial de la capacidad de detección de peatones de la red, siendo una disminución cercana al 80 % en la capacidad de detección de peatones. La varianza de los resultados obtenidos sobre el dataset completo ha disminuido ligeramente, pero un modelo mas estable no compensa la poca capacidad de inferencia de peatones.

Las razones del rendimiento de la red se han detallado anteriormente, señalando especialmente al redimensionamiento de esta red, a las diferencias de tamaño de imagen de los datasets y la diferencia en la localización de los objetos en la imagen como grandes problemas que influyen enormemente en la capacidad de detección de la red.

11.2. Resultados de la red Faster R-CNN

En esta subsección vamos a exponer los resultados de partida logrados por la red Faster R-CNN sobre los conjuntos de datos KITTI y el conjunto de datos completo, formado por KITTI, Tshingua-Daimler y Berkeley Deepdrive Dataset. Además se realizará una comparación del rendimiento por clase logrado por la red en fase de reentrenamiento.

11.2.1. Resultados Faster R-CNN sobre KITTI

Los resultados de la red sobre el conjunto de datos KITTI son los siguientes:

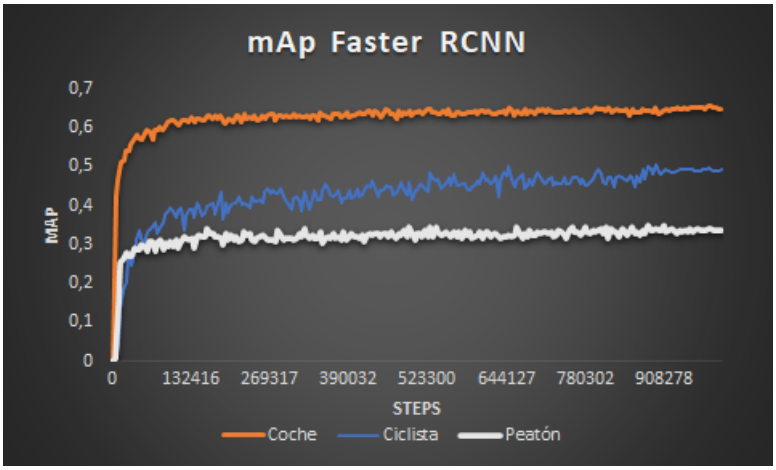

Figura 11.6: mAP de las clases coche, ciclista y peatón

En la figura 11.6, se observa como esta red tiene una mayor capacidad de detección que SSD a simple vista. Esta red por otro lado es mucho mas lenta, algo que probaremos mas adelante, siendo inviable para la inferencia en tiempo real sobre un vehículo. La precisión lograda en la clase coche es superior al 0,6 , mientras que la detección de ciclistas prácticamente llega al 0,5 tras 1000000 de épocas. Peatón es la clase que mas cuesta localizar y clasificar a esta red, con una precisión superior al 0,3. Finalmente, observamos que este modelo es mucho mas estable en sus detecciones que SSD al tener una menor variabilidad en sus inferencias y tiene una capacidad de predicción en fase de entrenamiento mucho mayor a la de SSD, logrando una precisión aproximada del 60 % en coches, del 50 % en ciclistas y del 30 % en peatones.

11.2.2. Resultados Faster R-CNN sobre el dataset completo

Los resultados de la red sobre el conjunto de datos completo son los siguientes:

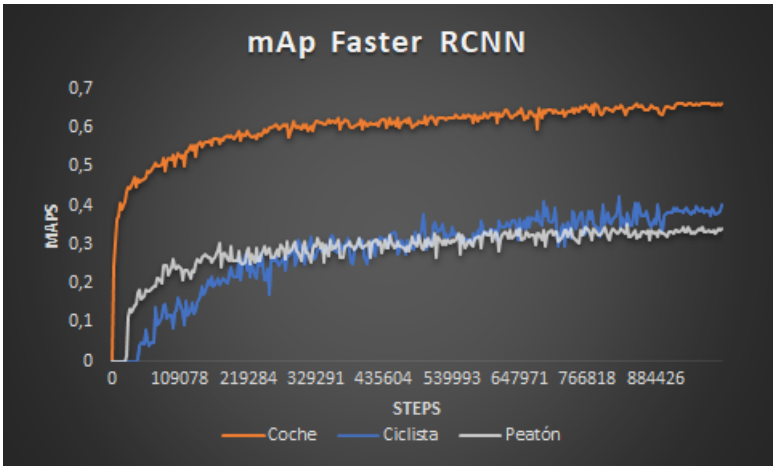

Figura 11.7: mAP de las clases coche, ciclista y peatón

En la figura 11.7 se observan unos resultados peores a los presentados sobre KITTI. Se reduce la capacidad de predicción en todas las clases y se observa una ligero aumento en la varianza de la capacidad de detección de objetos. Este comportamiento se debe a las diferencias en la localización de los objetos y el tamaño de las imágenes comentadas anteriormente entre lo conjuntos de datos KITTI, Berkeley Deepdrive Dataset y Tshingua-Daimler.

11.2.3. Comparación de resultados

En esta sección compararemos clase a clase los resultados obtenidos por esta red hasta el momento. Comenzaremos comparando la clase coche:

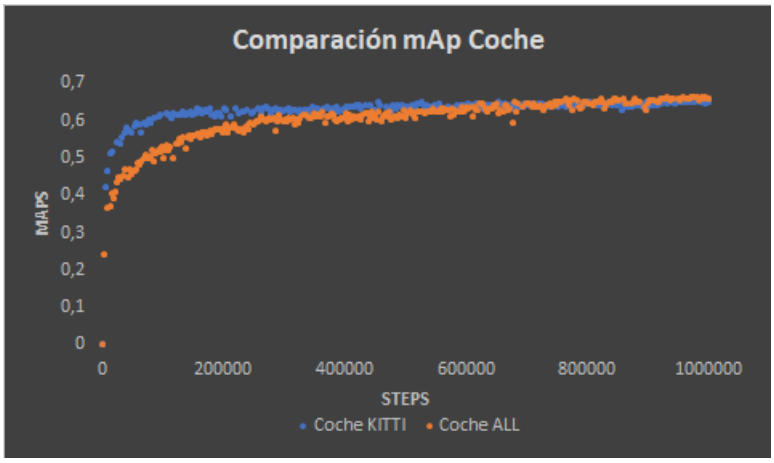

Figura 11.8: Comparación del mAP sobre la clase coche conseguido por la red Faster R-CNN entrenada sobre KITTI y el dataset completo.

El número de épocas entrenadas es de 1000000, dónde vemos que al comienzo la precisión es mayor sobre KITTI, pero al final termina siendo ligeramente superior sobre el conjunto de datos completo. En particular, se ha conseguido el objetivo de mantener o superar la precisión de la red sobre KITTI en esta clase. Ahora vemos los resultados sobre la clase ciclista:

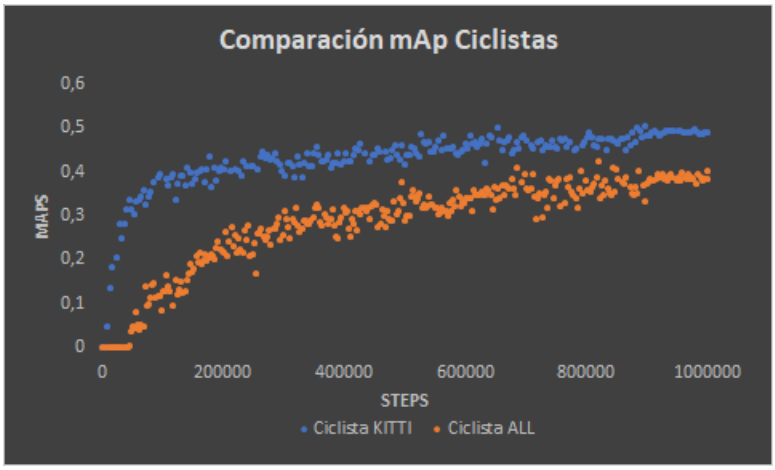

Figura 11.9: Comparación del mAP sobre la clase ciclista conseguido por la red Faster R-CNN entrenada sobre KITTI y el dataset completo.

En esta clase vemos que se ha producido una descenso considerable en la precisión, siendo esta diferencia mayor al comienzo del reentrenamiento y menor conforme este avanza. El objetivo de mejorar la precisión sobre esta clase no se ha logrado. Una de las posibles razones es que esta red realiza un redimensionamiento menor de las imágenes, a un tamaño 600×1024, teniendo como consecuencia una reducción del tamaño de los ciclistas. Además, los anchor boxes por defecto de la red no han sido tuneados, luego puede que no estén ajustados una forma adecuada para la clase ciclista. Otro posible problema es que si los ciclistas se encuentran al final de la imagen, y son pequeños, pueden no ser propuestos como regiones de interés, teniendo como consecuencia que la red no aprenda las características necesarias para la detección precisa de ciclistas.

Finalmente vemos los resultados sobre la clase peatón:

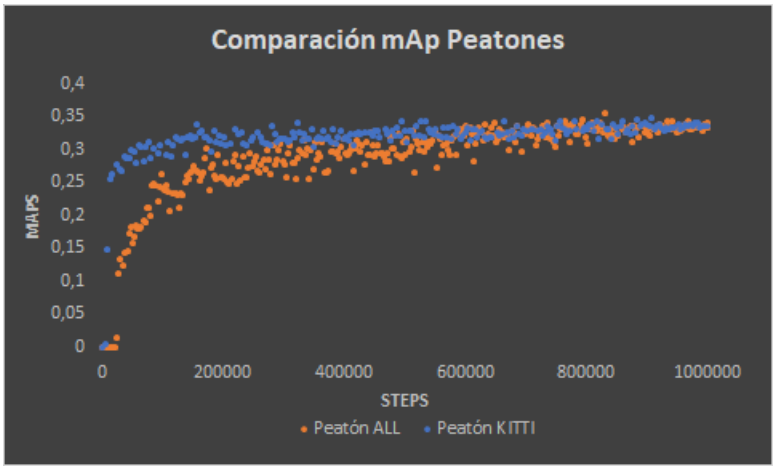

Figura 11.10: Comparación del mAP sobre la clase peatón conseguido por la red Faster R-CNN entrenada sobre KITTI y el dataset completo.

Al comienzo del reentrenamiento se observa una gran diferencia entre la precisión en la detección de la clase peatón sobre ambos datasets, sin embargo, al final podemos ver como esa diferencia se ha reducido completamente. El objetivo de mejorar la capacidad predictiva sobre la clase peatón no se ha logrado, teniendo el mismo problema que la clase ciclista. Las razones son similares a las expuestas en la clase ciclista, es decir, se trata de objetos difíciles de detectar debido a la posible ausencia de anchor boxes adecuados o bien que su tamaño en la imagen no es significativo a ser señalado como región de interés, luego la red no aprende de los casos en el fondo de la imagen.

11.3. Resultados de la red SSD Mobile Lite

En esta subsección vamos a exponer los resultados de partida logrados por la red SSD Mobile Lite sobre los conjuntos de datos KITTI y el conjunto de datos completo, formado por KITTI, Tshingua-Daimler y Berkeley Deepdrive Dataset. Además se realizará una comparación del rendimiento por clase logrado por la red en fase de reentrenamiento.

11.3.1. Resultados SSD Mobile Lite sobre KITTI

Los resultados de la red sobre KITTI son los siguientes:

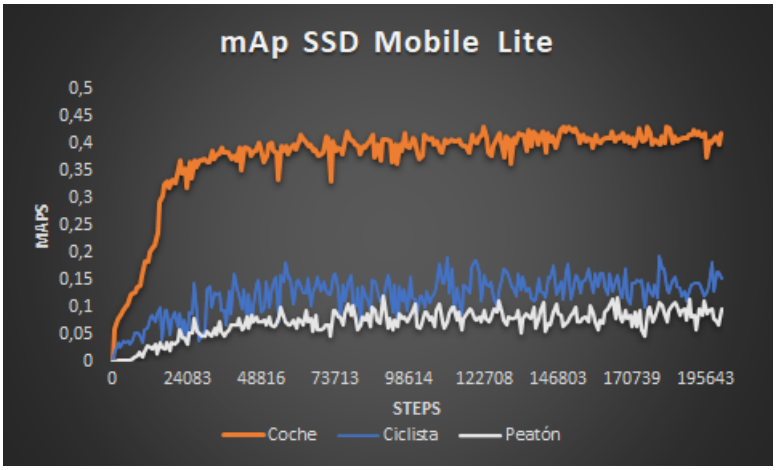

Figura 11.11: mAP de las clases coche, ciclista y peatón

Observamos los resultados sobre el reentrenamiento de 200000 épocas realizado. A primera vista, vemos unos resultados inferiores a los logrados con la red Faster R-CNN, aunque estos van en la línea de los logrados son SSD Inception, e incluso son mejores que los de esta red. En la clase coche tiene una precisión del 0,4, mientras que en la clase ciclista es de 0,15. La clase peatón es la que mas problemas tiene para localizar y clasificar, con un mAp cercano al 0,1. En resumen, es capaz de detectar aproximadamente el 40 % de los coches, el 15 % de ciclistas y el 10 % de peatones. Otra característica de esta red, es que muestra una mayor estabilidad, al mostrar resultados menos volátiles en las distintas evaluaciones realizadas durante el entrenamiento.

11.3.2. Resultados SSD Mobile Lite sobre el dataset completo

Los resultados de la red sobre el conjunto de datos completo son los siguientes:

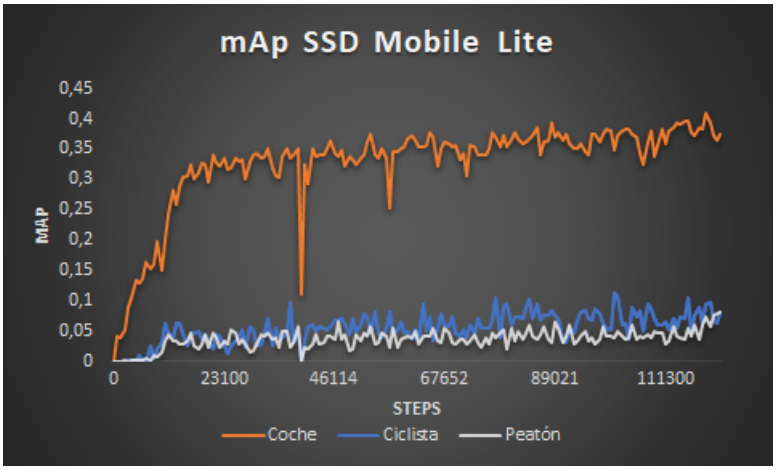

Figura 11.12: mAP de las clases coche, ciclista y peatón

A priori los resultados son peores que los conseguidos en KITTI, algo que veremos mas detalladamente en la siguiente sección. Al menos se mantiene la estabilidad de las predicciones, aunque la precisión ha bajado en todas las clases detectadas. En la clase coche muestra una precisión por debajo de 0,4 tras 120000 épocas, mientras que por otro lado las clases peatón y ciclista tienen un mAp por debajo del 0,1. En la siguiente sección veremos claramente que el dataset completo disminuye la precisión de la red, comportamiento similar al visto en las redes SSD Inception y Faster R-CNN.

11.3.3. Comparación de resultados

En esta sección compararemos clase a clase los resultados obtenidos por esta red hasta el momento. Comenzaremos comparando la clase coche:

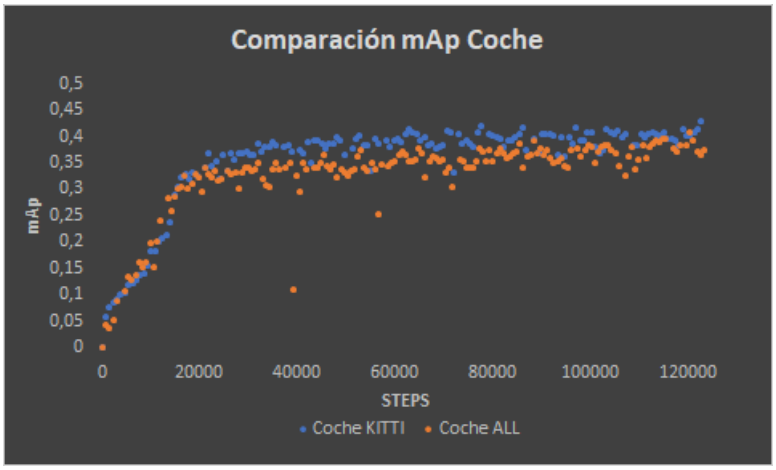

Figura 11.13: Comparación del mAP sobre la clase coche conseguido por la red SSD Mobile Lite entrenada sobre KITTI y el dataset completo.

La figura 11.13 muestra la menor capacidad de precisión sobre la clase coche en el dataset completo una vez se compara con el KITTI. Esta diferencia no es superior a dos puntos por décima. Por otro lado vemos los resultados en la clase ciclista:

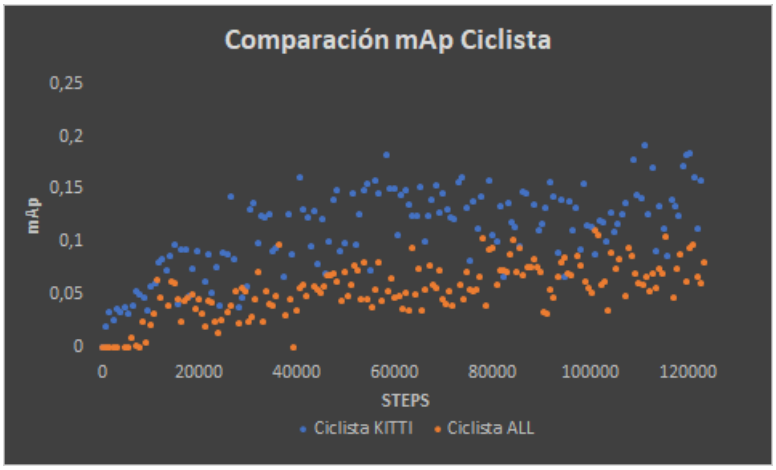

Figura 11.14: Comparación del mAP sobre la clase ciclista conseguido por la red SSD Mobile Lite entrenada sobre KITTI y el dataset completo.

La figura 11.14 representa la diferencia existente entre la capacidad de predicción de ciclistas por la red una vez reentrenada sobre distintos conjuntos, siendo mucho menor en el caso del conjunto de datos completo. Finalmente vemos los resultados sobre la clase peatón:

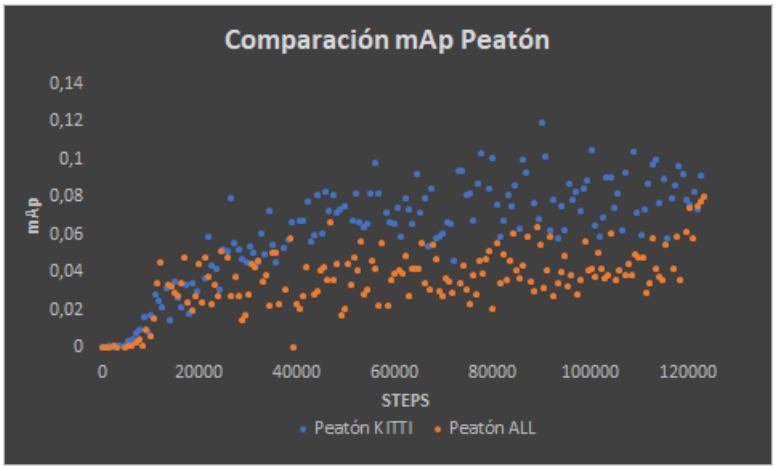

Figura 11.15: Comparación del mAP sobre la clase peatón conseguido por la red SSD Mobile Lite entrenada sobre KITTI y el dataset completo.

Los resultados ilustrados en la figura 11.15 van en la línea de lo comentado previamente, es decir, que la capacidad de precisión de la red ha disminuido al aumentar el conjunto de entrenamiento con mas imágenes.

Además de obvservar un rendimiento bajo en todas las clases, especialmente acentuado en las clases peatón y ciclista. Este comportamiento se debe principalmente a varias razones:

- La arquitectura de la red, de forma que el redimensionamiento que realiza penaliza especialmente a los objetos pequeños, como pueden ser los ciclista y peatones en una imagen.
- Las diferencias de tamaño de imagen de los conjuntos KITTI, Berkeley Deepdrive Dataset y Tshingua-Daimler.

En el apartado de conclusiones ampliaremos las causas y motivos de este comportamiento.

11.4. Resultados de la red YOLOv3

En esta subsección vamos a exponer los resultados de partida logrados por la red YOLOv3 sobre los conjuntos de datos KITTI y el conjunto de datos completo, formado por KITTI, Tshingua-Daimler y Berkeley Deepdrive Dataset. Además se realizará una comparación del rendimiento por clase logrado por la red.

11.4.1. Resultados YOLOv3 sobre KITTI

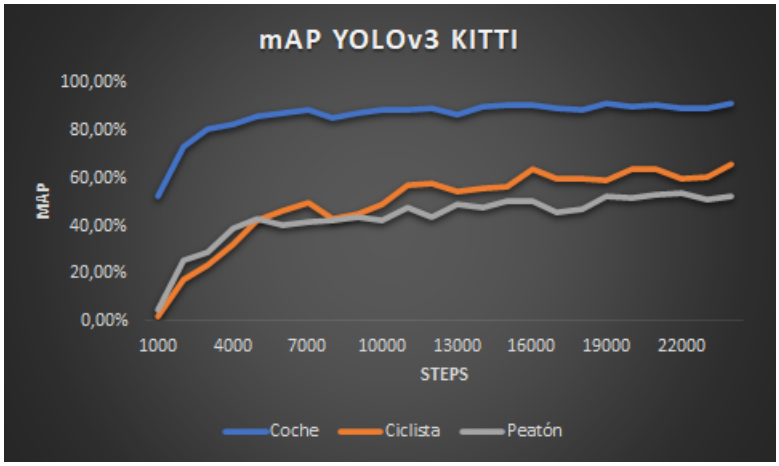

Figura 11.16: MAP durante el reentrenamiento de la red YOLOv3 sobre KITTI.

Como podemos ver, el aprendizaje inicial es grande gracias a factores como que el extractor de características está previamente entrenado. El mAP sobre el conjunto de validación sigue aumentando y tiende a estabilizarse sobre todo en la clase coche en torno a las 20000 iteraciones, que con un tamaño de 64 imágenes por lote, equivale a 1280000 imágenes procesadas.

11.4.2. Resultados YOLOv3 sobre el dataset completo

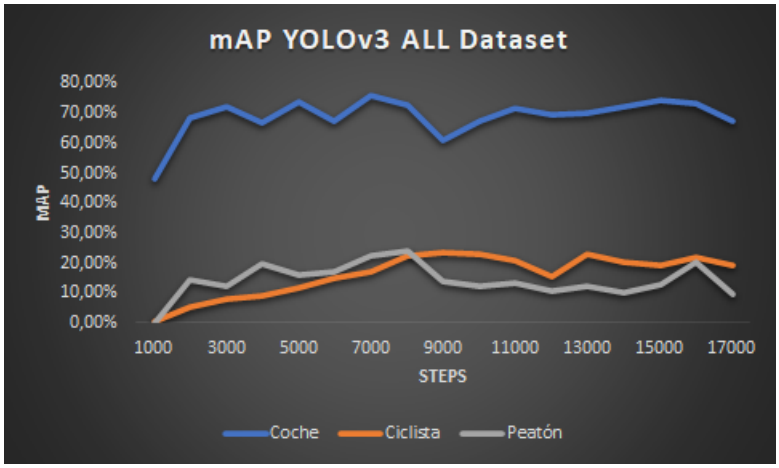

Figura 11.17: MAP durante el reentrenamiento de la red YOLOv3 sobre el dataset completo.

Al reentrenar la red sobre el dataset completo que mezcla 3 tipos diferentes de imágenes y cajas delimitadoras, vemos como el aprendizaje es más caótico llegando incluso a perder precisión sobre la clase peatón.

11.5. Resultados de la red YOLOv3 tiny

En esta subsección vamos a exponer los resultados de partida logrados por la red YOLOv3 tiny sobre los conjuntos de datos KITTI y el conjunto de datos completo, formado por KITTI, Tshingua-Daimler y Berkeley Deepdrive Dataset. Además se realizará una comparación del rendimiento por clase logrado por la red.

11.5.1. Resultados YOLOv3 tiny sobre KITTI

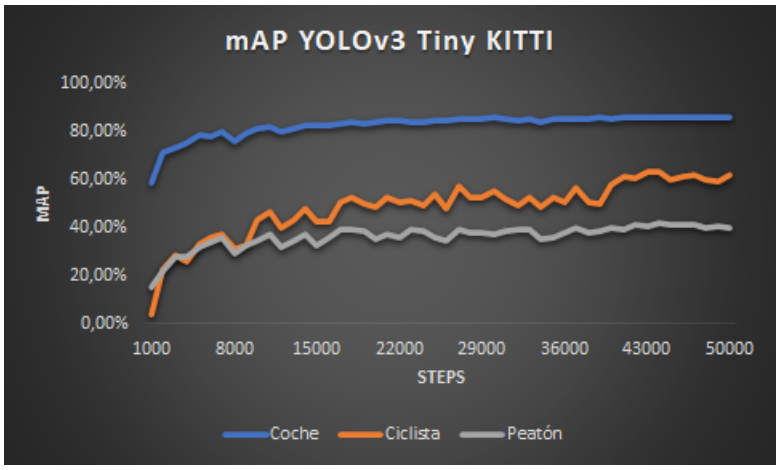

Figura 11.18: MAP durante el reentrenamiento de la red YOLOv3 Tiny sobre KITTI.

Vemos como al igual que con YOLOv3, el aprendizaje inicial es elevado pero debido a las limitaciones de la estructura más sencilla que presenta la versión tiny a cambio de mejorar en velocidad, el aprendizaje se estabiliza antes en las clases Coche y peatón y le cuesta más aprender. Sin embargo, el mAP de la clase ciclista sigue aumentando.

11.5.2. Resultados YOLOv3 tiny sobre el dataset completo

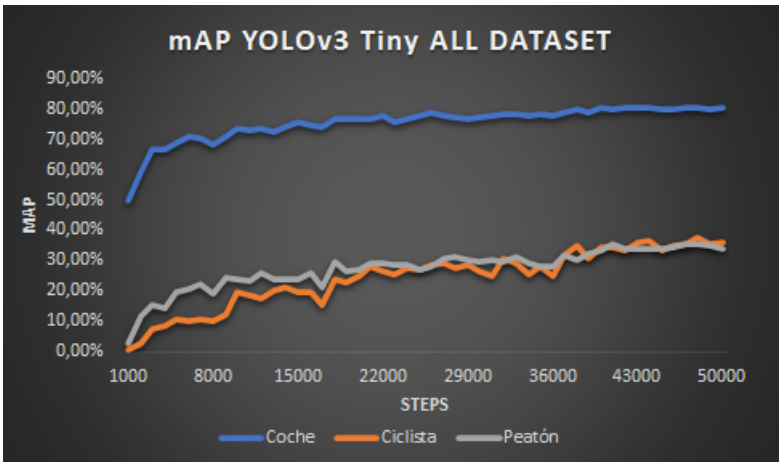

Figura 11.19: MAP durante el reentrenamiento de la red YOLOv3 Tiny sobre el dataset completo.

Presenta el mismo comporamiento que YOLOv3 al aumentar el dataset. Llama la atención como en este caso incluso perjudica y baja la precisión sobre todas las clases, siendo la diferencia bastante notable en la clase ciclista. Aun así, cabe destacar que el número de pasos en el reentrenamiento es el mismo, teniendo el dataset completo más imágenes que solo KITTI, así que habría que entrenarlo más para poder comparar realmente.

12. Resultado de Inferencia

En esta sección vamos a reflejar mediante tablas los resultados de la inferencia de los distintos modelos sobre el framework Openvino en el caso de SSD Inception, SSD MobileLite y Faster R-CNN, y sobre el framework Darknet en el caso de YOLO. La estructura será la siguiente:

- Presentación de tabla por modelo, conjunto de datos y dispositivo usado para la inferencia.
- Gráfica de comparación de consumo, velocidad de inferencia y precisión entre modelos distintos.

La inferencia se realiza sobre un subconjunto aleatorio del dataset KITTI de 1500 imágenes que no se han utilizado durante el entrenamiento de los modelos.

Respecto a los dispositivos utilizados para la inferencia, distinguimos entre los usados para inferir las redes SSD Inception V2, Faster R-CNN y SSD Mobile Lite: GPU, CPU y Myriad. Mientras que por otro lado se ha utilizado una GPU de alto rendimiento para la inferencia de las redes YOLOv3 y YOLOv3 Tiny.

Además es de remarcar que tanto la CPU trabaja internamente con precisión FP32 aunque se ejecute con la opción de precisión FP16, como Myriad que siempre opera con precisión FP16 aunque se especifique FP32. Debido a estar razones, los resultados con las opciones FP32 y FP16 sobre ambos dispositivos son iguales, por ello no se han presentado en las siguientes tablas, ya que era información redundante.

12.1. Inferencia de SSD Inception V2 en Openvino

Set	**Device**	**Float Point**	**(W)**	**Time (s)**	**FPS**	**FPS/W**	**MAP Person**	**MAP Cyclist**	**MAP Car**	**MAP media**
KITTI	GPU	FP32	15	83,67	17,93	1,20	33,04 %	32,15 %	73,54 %	46,24 %
KITTI	GPU	FP16	15	59,87	25,05	1,67	32,55 %	33,05 %	73,63 %	46,41 %
KITTI	CPU	FP32	95	47,19	31,79	0,33	33,04 %	32,15 %	73,54 %	46,24 %
KITTI	MYRIAD	FP16	1	60,46	24,81	24,81	33,25 %	32,18 %	73,57 %	46,33 %
FINAL	GPU	FP32	15	84,73	17,70	1,18	4,77 %	9,15 %	40,80 %	18,24 %
FINAL	GPU	FP16	15	54,84	27,35	1,82	4,77 %	9,15 %	40,83 %	18,25 %
FINAL	CPU	FP32	95	47,02	31,90	0,34	4,77 %	9,15 %	40,80 %	18,24 %
FINAL	MYRIAD	FP16	1	60,75	24,69	24,69	4,55 %	9,14 %	40,84 %	18,17 %

Tabla 12.1: Inferencia del modelo SSD Inception V2 usando el framework Openvino

En la tabla 12.1 podemos ver la comparación por precisión en cada clase, velocidad de inferencia y consumo de la red SSD Inception V2 sobre ambos conjuntos de datos y en cada uno de los dispositivos analizados.

Centrándonos en el mAP logrado por cada clase, vemos como el rendimiento sobre el conjunto de datos final es muy pobre, corroborando lo visto en la fase de fine-tuning. Esta disminución de la precisión es aproximadamente del 85 % en la clase peatón, del 72 % en la clase ciclista y del 54 % en la clase coche, conllevando una desmejoría sobre el mAP medio del 60 %.

Si por otro lado nos fijamos en la velocidad de inferencia, es decir, en los *frames por segundo*(FPS), vemos como únicamente GPU con FP32 no es capaz de realizar inferencia en tiempo real que se estima en 20 FPS. Observamos como CPU es el más rápido en realizar la inferencia con casi 32 FPS en ambos datasets, seguido de la GPU con precisión FP16 que logra un velocidad entorno a 25 FPS sobre el KITTI y 27 FPS sobre el conjunto final. Finalmente destaca Myriad siendo capaz de lograr una velocidad nada desdeñable entorno a 24 FPS, muy cerca de lo ofrecido por la GPU.

Finalmente si analizamos la métrica FPS/W, que relaciona velocidad de inferencia del modelo con consumo del dispositivo destaca claramente Myriad como dispositivo mas eficiente en esto términos. Su bajo consumo de 1 W, junto a ser un dispositivo diseñado por Intel para la inferencia de redes, le hacen una alternativa muy interesante en termino

de eficiencia. Observamos el dispositivo menos eficiente es la CPU, ya que tiene un consumo muy elevado de 95W, mientras que la GPU es mucho mas eficiente que la CPU en términos de consumo, ofreciendo mejores resultados con precisión FP36, aunque esta muy por detrás de Myriad en este aspecto.

12.2. Inferencia de SSD Mobile Lite en Openvino

Set	**Device**	**Float Point**	**(W)**	**Time (s)**	**FPS**	**FPS/W**	**MAP Person**	**MAP Cyclist**	**MAP Car**	**MAP media**
KITTI	GPU	FP32	15	29,61	50,65	3,38	36,79 %	37,53 %	70,72 %	48,35 %
KITTI	GPU	FP16	15	23,96	62,60	4,17	36,52 %	38,24 %	70,54 %	48,43 %
KITTI	CPU	FP32	95	22,13	67,78	0,71	36,79 %	37,53 %	70,72 %	48,35 %
KITTI	Myriad	FP16	1	44,87	33,43	33,43	36,54 %	38,06 %	70,66 %	48,45 %
FINAL	GPU	FP32	15	29,57	50,73	3,38	26,47 %	27,52 %	67,91 %	40,63 %
FINAL	GPU	FP16	15	23,99	62,53	4,17	26,59 %	27,70 %	67,93 %	40,74 %
FINAL	CPU	FP32	95	22,01	68,16	0,72	26,47 %	27,52 %	67,91 %	40,63 %
FINAL	Myriad	FP16	1	44,84	33,45	33,45	26,70 %	27,50 %	68,34 %	40,75 %

Tabla 12.2: Inferencia del modelo SSD Mobile Lite usando el framework Openvino

En la tabla 12.2 podemos ver la comparación por precisión en cada clase, velocidad de inferencia y consumo de la red SSD Mobile Lite sobre ambos conjuntos de datos y en cada uno de los dispositivos analizados.

Ateniéndonos a la precisión lograda por clase observamos como esta red ofrecía resultados especialmente buenos sobre el KITTI, y el hecho de aumentar el conjunto de imágenes por clase ha resultado en un empeoramiento por clase de la precisión. Esta desmejora ha sido entorno al 27 % en las clases peatón y ciclista, mientras que ha sido entorno al 4 % en la clase coche. Este empeoramiento se debe a que esta red no se ve beneficiada de la variabilidad de las cajas delimitadoras provocada por el aumento de imágenes, resultando en una peor capacidad de predicción.

Si por otro lado nos enfocamos en la velocidad de la red según el dispositivo, observamos que esta red es mucho mas veloz que la red SSD Inception V2. Especialmente destacable los casi 68 FPS logrados con CPU, aunque la GPU tampoco se queda atrás logrando 62 FPS. Aunque es cierto que Myriad ofrece menos de la mitad FPS que la CPU, alcanzando los 34 FPS.

Finalmente si analizamos la métrica FPS/W, el comportamiento es muy similar a la red SSD Inception V2, destacando principalmente Myriad como un dispositivo muy eficiente

en términos de velocidad/consumo, seguida por la GPU y en último lugar la CPU.

12.3. Inferencia de Faster R-CNN en Openvino

Set	**Device**	**Float Point**	**(W)**	**Time (s)**	**FPS**	**FPS/W**	**MAP Person**	**MAP Cyclist**	**MAP Car**	**MAP media**
KITTI	GPU	FP32	15	539,42	2,78	0,19	46,09 %	40,76 %	50,32 %	45,72 %
KITTI	GPU	FP16	15	357,22	4,20	0,28	44,41 %	40,13 %	49,62 %	44,72 %
KITTI	CPU	FP32	95	430,18	3,49	0,04	46,09 %	40,76 %	50,32 %	45,72 %
KITTI	Myriad	FP16	1	967,93	1,55	1,55	39,00 %	30,36 %	18,04 %	30,14 %
FINAL	GPU	FP32	15	533,83	2,81	0,19	15,99 %	28,80 %	24,81 %	23,20 %
FINAL	GPU	FP16	15	275,22	5,45	0,36	16,30 %	23,26 %	24,74 %	21,43 %
FINAL	CPU	FP32	95	442,70	3,39	0,04	15,99 %	28,80 %	24,81 %	23,20 %
FINAL	Myriad	FP16	1	966,56	1,55	1,55	15,40 %	22,89 %	17,36 %	18,55 %

Tabla 12.3: Inferencia del modelo Faster R-CNN usando el framework Openvino

En la tabla 12.3 podemos ver la comparación por precisión en cada clase, velocidad de inferencia y consumo de la red Faster R-CNN sobre ambos conjuntos de datos y en cada uno de los dispositivos analizados.

A primera vista vemos que esta red ha sufrido una bajada en su capacidad de precisión de manera muy notables al aumentar el conjunto KITTI. Este comportamiento se debe a que le cueste mucho extraer las características mas importantes de las imágenes al ser un conjunto de datos con cajas delimitadoras de formas muy variadas, pues el conjunto final esta compuesto por tres datasets distintos. Si bien los resultados no están mal para el conjunto de datos KITTI, la disminución es del 66 % en la clase peatón, del 25 % en la clase ciclista y del 50 % en la clase coche, siendo una disminución media entorno al 50 %. Además se observan diferencias en la precisión entre dispositivos, especialmente en el caso de Myriad que puede que indique un problema de compatibilidad con Faster R-CNN.

Si por otro lado nos fijamos en la velocidad, vemos que esta es la red que ofrece una menor velocidad, no llegando a superar los 20 FPS requeridos para hacer inferencia en tiempo real en un vehículo. Esto se debe a la arquitectura de la red, siendo esta en dos fases y por tanto mucho mas lenta que las anteriormente presentadas.

Finalmente si nos centramos en el valor de la métrica FPS/W, vemos que el hecho de que la velocidad de inferencia en términos de FPS sea tan baja en esta red provoca que esta métrica que relaciona velocidad con consumo energético tenga valores muy

bajos. Definitivamente esta red no es adecuada para este problema. Si nos atenemos a los dispositivos, la linea es similar a los modelos anteriormente presentados, destacando Myriad, seguido de GPU y con CPU a la cola en cuando a eficiencia.

12.4. Inferencia YOLOv3 en Darknet

Set	Device	(W)	Time (s)	FPS	Px/(s*W)	MAP Person	MAP Bicycle	MAP Car	MAP media
KITTI	GPU AR	165	49,00	30,61	21,40	55,72 %	66,23 %	91,00 %	70,98 %
FINAL	GPU AR	165	50,00	30,00	20,98	22,10 %	20,16 %	69,75 %	37,34 %

Tabla 12.4: Inferencia del modelo YOLOv3 ejecutado en la GPU de alto rendimiento usando el framework Darknet

Como podemos observar, la red YOLOv3 gana bastante en cuanto a precisión con respecto a las arquitecturas anteriores, especialmente en clases como peatón y ciclista. Esto se debe a varios factores, como pueden ser la utilización de anchors de diferentes escalas, las predicciones en 3 escalas diferentes que realiza la propia red, utilizando un concepto similar a las redes piramidales de características para el extractor de características y el tamaño de redimensionamiento de la red, que en este caso es 416x416. Sin embargo, aún deja que desear en cuanto a tiempo de ejecución, teniendo en cuenta que la ejecución de esta red ha sido realizada en una GPU de alto rendimiento con aceleradores como CUDA. Es recalcable como al entrenar sobre el dataset completo, se pierde precisión, al ser las imágenes de los datasets tan distintas y ser menor el entrenamiento total.

12.5. Inferencia YOLOv3 Tiny en Darknet

Set	Device	(W)	Time (s)	FPS	Px/(s*W)	MAP Person	MAP Bicycle	MAP Car	MAP media
KITTI	GPU AR	165	11,00	136,36	95,35	41,46 %	64,30 %	86,74 %	64,17 %
FINAL	GPU AR	165	11,00	136,36	95,35	35,71 %	35,35 %	80,01 %	50,36 %

Tabla 12.5: Inferencia del modelo YOLOv3 Tiny ejecutado en la GPU de alto rendimiento usando el framework Darknet

Como podemos observar, el tiempo de ejecución es bastante rápido en comparación con el resto de arquitecturas, aunque recalcamos que la GPU en la que ha sido ejecutada ha sido una GPU de alto rendimiento, así que no podemos compararla en este sentido con el resto de redes, solo con su versión más pesada YOLOv3, con la que mejora en casi 5 veces la velocidad de inferencia y sin sacrificar demasiada precisión. En cuanto a precisión, vemos cómo sigue superando al resto de redes gracias a los mismos factores que YOLOv3, aunque el extractor de características es más simple en este caso. Es recalcable

como al entrenar sobre el dataset completo, se pierde precisión, al ser las imágenes de los datasets tan distintas y ser menor el entrenamiento total.

13. Comparación

13.1. MAP

En la siguiente figura podemos observar una comparación del mAP obtenido en la inferencia final sobre la partición test de KITTI de las distintas redes, reentrenadas sobre KITTI en azul y sobre el dataset final en naranja:

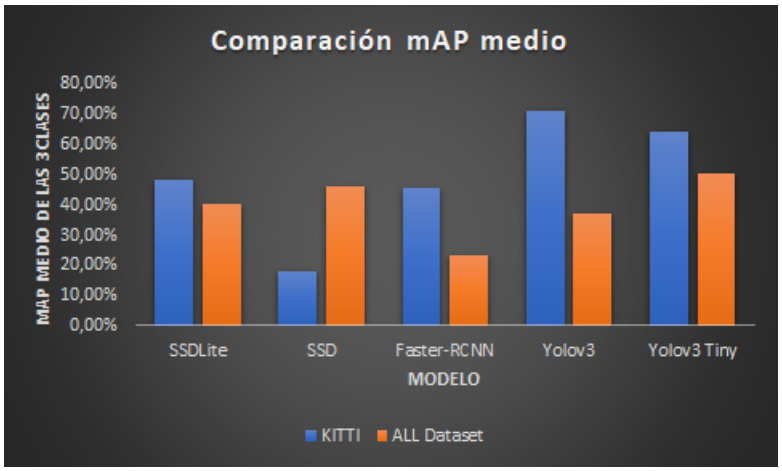

Figura 13.1: Comparación del mAP obtenido por las distintas redes.

Podemos observar cómo en todas las redes se pierde precisión al reentrenar sobre el dataset completo o aumentado. Esto no necesariamente indica que el modelo sea peor, simplemente puede deberse a varios factores como:

- La evaluación únicamente sobre un subconjunto del dataset KITTI.
- La variedad en los tamaños y formas de las cajas delimitadoras entre los distintos datasets, al tratarse el dataset KITTI de imágenes panorámicas.
- La existencia de cajas delimitadoras con un área muy pequeña en el dataset aumentado que, al trabajar con redes que redimensionan y disminuyen mucho el tamaño de la imagen, puede no solo no aportar nada al entrenamiento si no dificultarlo.
- Al aumentar el tamaño del dataset, se necesite más tiempo de entrenamiento.

13.2. Velocidad de inferencia

En este apartado vamos a comparar las distintas redes en cuanto a velocidad de inferencia se refiere. Para ello, usaremos la métrica **FPS** o *Frames Por Segundo*, que nos indica el número de imágenes por segundo que es capaz de procesar dicha red. Destacar que uno de los objetivos es superar el umbral de procesamiento en tiempo real de 20FPS.

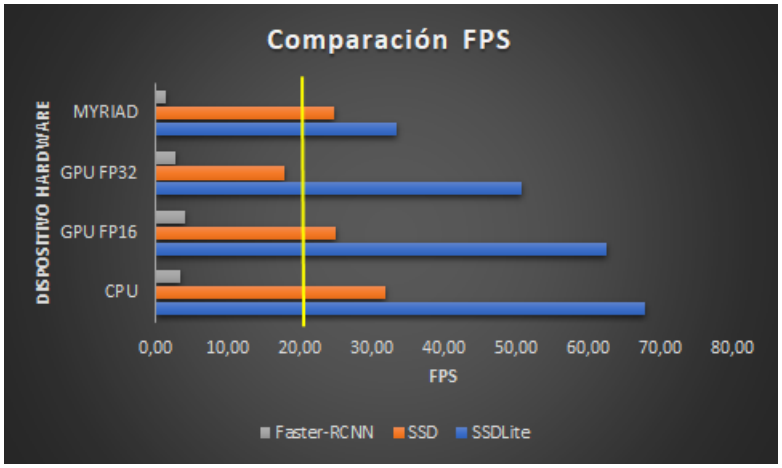

Figura 13.2: FPS alcanzado por las redes SSD, SDDLite y Faster-R-CNN ejecutadas sobre diferentes dispositivos hardware.

Podemos ver como entre los distintos dispositivos, Myriad es el más lento y la CPU el más rápido. También podemos observar una diferencia interesante en cuanto a la GPU con un modelo cuantizado con una precisión FP32, es decir, que trabajo con variables flotantes de 32 bits con respecto al mismo modelo cuantizado con FP16, mejorando notablemente la velocidad de la inferencia y sacrificando muy poca precisión en los resultados finales. Ésto es un concepto importante a tener en cuenta bajo el contexto de la conducción, donde la pérdida de un poco de precisión puede suponer alcanzar una velocidad de procesamiento aceptable. Por último, vemos como redes con más precisión como Faster-R-CNN se quedan muy lejos del umbral de procesamiento en tiempo real de 20FPS, mientras que redes más ligeras como SSD o SSDLite lo superan en casi todos los dispositivos.

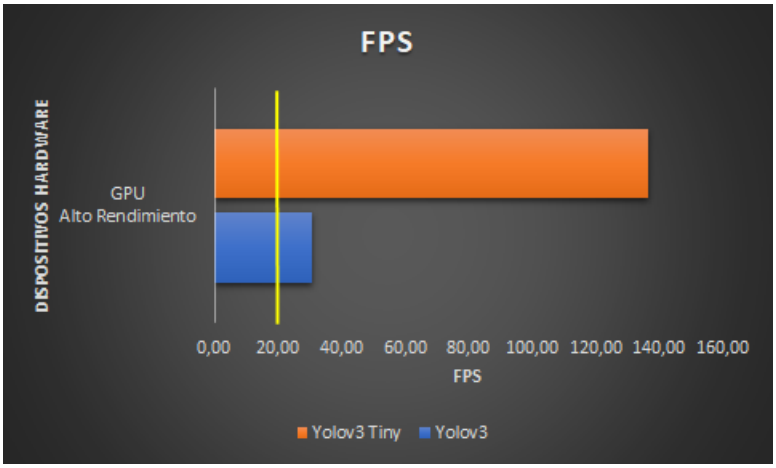

Figura 13.3: FPS alcanzado por ambas redes YOLO ejecutadas sobre una GPU de alto rendimiento.

Podemos ver la gran diferencia entre ambas versiones de YOLOv3, siendo la versión YOLOv3 Tiny casi 5 veces más rápida a cambio de sacrificar algo de precisión, como hemos visto en los resultados. De nuevo recalcar que la inferencia de estas dos redes se ha realizado en una GPU de alto rendimiento, así que los FPS no deberían compararse con el resto de redes.

13.3. Intercambio velocidad-consumo

Debido a que una de las variables críticas a considerar en la conducción autónoma es la autonomía del vehículo, vamos a comparar los FPS obtenidos por cada red teniendo en cuenta el consumo de los diferentes dispositivos hardware, para obtener así un ratio de esta velocidad-consumo.

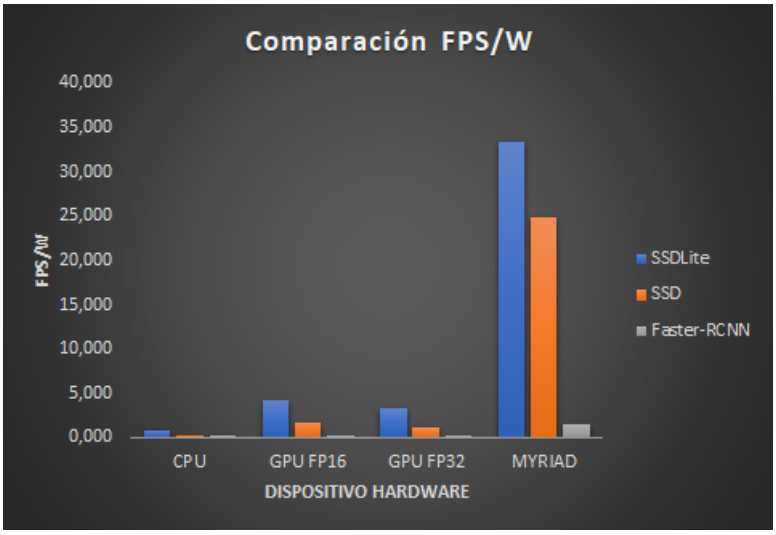

Figura 13.4: Comparación de los FPS por vatio de las distintas redes atendiendo a los consumos medios de los distintos dispositivos hardware.

Vemos, como era de esperar, cómo el dispositivo Myriad destaca por su gran rendimiento. Este dispositivo está diseñado para la ejecución de redes neuronales de manera eficiente ofreciendo un consumo muy bajo, de tan solo 1W, y podemos apreciar como cumple su cometido en la gráfica anterior. Este tipo de dispositivos es muy interesante para el punto de vista de la conducción, pudiendo ser la inferencia en paralelo una buena opción para lograr una velocidad de inferencia aceptable manteniendo un consumo bajo.

Vamos a comparar este ratio de FPS/W obtenido por las 5 redes intentando que la comparación sea lo más válida posible. Para ello vamos a los resultados obtenidos por las redes SSD, SSD Lite y Faster R-CNN al ejecutar la inferencia sobre la GPU integrada de bajo consumo y por las redes YOLOv3 y YOLOv3 Tiny sobre la GPU de alto rendimiento. Recalcar que no son el mismo dispositivo, pero en ambos casos se trata de una GPU por lo que la estructura y procesamiento son similares.

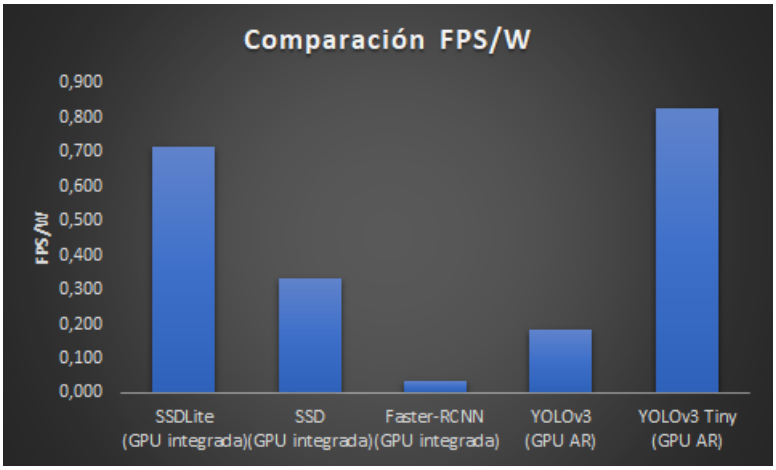

Figura 13.5: Comparación del ratio velocidad/potencia (FPS/W) al realizar inferencia con las distintas redes, en el caso de SSD, SSDLite y Faster R-CNN sobre una GPU integrada de bajo consumo(15W) y en el caso de YOLOv3 y YOLOv3 Tiny sobre una GPU de alto rendimiento (165W).

Esta gráfica nos muestra cómo la estructura de red neuronal YOLOv3 Tiny sigue siendo la que ofrece mejor ratio velocidad-consumo y, como hemos visto anteriormente, siendo también una de las que mejores resultados en cuanto a precisión se refiere.

14. Conclusiones

Este proyecto tenía principalmente dos objetivos que se descritos al comienzo de la memoria:

- Reentrenamiento de una red generalista para aprender a identificar vehículos, peatones y ciclistas.
- Estudio del rendimiento de la inferencia realizada por la red reentrenada en términos de precisión, velocidad de inferencia y consumo sobre un conjunto de dispositivos hardware.

Para responder a la primera pregunta nos centramos en el estudio de las distintas soluciones software que nos permiten procesar las imágenes en tiempo real, usando de esta forma frameworks conocidos en el campo de la detección de objetos como Darknet, Openvino y Tensorflow. Estos frameworks nos han permitido reentrenar redes neuronales de propósito generalista para aprender a identificar vehículos, peatones y ciclistas. En concreto se han trabajado con las siguientes redes:

1. SSD Inception V2
2. SSD Mobile Lite

3. Faster R-CNN
4. YOLOv3
5. YOLOv3 Tiny

El reentrenamiento se ha realizado sobre el conjunto de datos KITTI y otro conjunto de datos que consiste en la ampliación del conjunto de datos KITTI con algunas imágenes de los conjuntos Tshingua-Daimler y Berkeley Deepdrive Dataset. En la sección 11 se han analizado los resultados obtenidos durante los reentrenamientos de las distintas redes. A partir de ellos deducimos que al ampliar el conjunto de datos inicial con una mayor variedad de cajas delimitadoras dificulta el aprendizaje de la red, lo cual podemos ver reflejado en una función de pérdida más inestable y que decrece más lentamente.

En cuanto a los resultados finales realizados sobre el conjunto de validación, deducimos que no se consigue un mejor rendimiento aumentando el conjunto de entrenamiento, ya que todas las redes han visto reducido notablemente su precisión. En particular, si se quiere aumentar el conjunto de entrenamiento, este aumento debe ser con imágenes muy similares a las del conjunto de test.

En este caso, los conjuntos de datos tomados son imágenes tomadas desde un vehículo, todas de día y en condiciones favorables. Sin embargo, hay disparidades importantes entre los conjuntos de datos. KITTI es un conjunto de imágenes panorámicas, por tanto el ancho es 4 veces el alto de la imagen. Esta diferencia es crucial a la hora de redimensionar imágenes como se hace en las redes SSD Inception V2 y SSD Mobile Lite en las que se redimensiona a unas dimensiones fijas de 300x300, con la consecuencia de que los objetos con poca anchura inicial pueden llegar a ser imperceptibles.

Por otro lado, Berkeley Deepdrive Dataset cuenta con objetos con una superficie más reducida que en el KITTI. Esto se debe a que las imágenes de este conjunto etiquetan todos los objetos visibles en la imagen, incluso algunos tan en el fondo de la imagen que su superficie es menor a 20 píxeles. Por tanto, en las redes donde se produce un redimensionamiento, se reducen las dimensiones de las imágenes, provocando que estos objetos pasan a tener una superficie mínima. De esta manera es muy complicado que los detectores de objetos puedan identificarlos en la imagen. Además el tamaño de los anchors no ha sido modificado para adaptarlo a las peculiaridades del conjunto de datos, siendo una dificultad más añadida.

En cuanto al segundo objetivo, en la secciones 12 y 13 se han comparado la precisión, consumo y velocidad de inferencia de las redes sobre ambos conjuntos de datos y sobre tres dispositivos en el caso de las redes SSD Inception V2, SSD Mobile Lite y Faster R-CNN:

- CPU

- GPU integrada de bajo consumo
- Myriad

y en el caso de las redes YOLOv3 y YOLOv3 Tiny sobre una GPU de alto rendimiento.

Los resultados determinan que las mejores redes para lograr el objetivo de la conducción autónoma teniendo en cuenta la precisión son YOLOv3 y YOLOv3 Tiny. Ambas redes ofrecen una precisión por clase notablemente superior al resto de redes estudiadas. Otro aspecto vital en el contexto de la conducción autónoma es la velocidad de inferencia, en ese caso YOLOv3 Tiny es hasta cinco veces mas rápida que YOLOv3. No se han podido ejecutar éstas redes en los mismos dispositivos que el resto. Sin embargo, como podemos apreciar en la gráfica comparativa 13.5, evaluando el intercambio velocidad de procesamiento-consumo medido sobre dos arquitecturas similares, en este caso ambas GPUs, podemos ver cómo igualmente siguen siendo más prometedoras que el resto de redes.

Además de la velocidad de inferencia, la autonomía del vehículo es crucial en la conducción autónoma y, por tanto, el consumo de los distintos dispositivos evaluados. En la sección 13.3 se ha analizado el consumo de los dispositivos y los modelos ejecutados bajo los mismos. Para poder evaluar conjuntamente el rendimiento de un modelo y la eficiencia del dispositivo hardware sobre el que se ha ejecutado, se ha definido la métrica FPS/W que relaciona velocidad de inferencia con consumo. El resultado es que los modelos que tienen una arquitectura de una fase, es decir, las redes SSD y YOLO, son mas eficientes en términos de velocidad/consumo. Esta conclusión es natural teniendo en cuenta la baja velocidad de inferencia de Faster R-CNN en comparación con el resto.

Centrándonos en los dispositivos hardware, hay uno que resalta por encima del resto, y éste es Myriad. Éste dispositivo diseñado para realizar inferencia de redes neuronales de manera óptima ofrece un consumo mínimo, en concreto de 1W, y por tanto es el dispositivo mas eficiente en términos de velocidad/consumo. Es notable cómo a pesar de su bajo consumo, llega a realizar inferencia en tiempo real las redes SSD Inception V2 y SSD Mobile Lite. La GPU integrada con precisión FP16 ofrece unas velocidades de inferencia aceptables bajo un consumo de 15W, siendo por tanto la GPU de alto rendimiento y la CPU los dispositivos menos eficientes, por su consumo elevado, especialmente en el primer caso; y por no estar especializada en estas tareas en el caso de la CPU.

Como conclusión final, después del estudio y comparaciones realizadas, la arquitectura o modelo de detección de objetos de entre todas las analizadas en el estado del arte es, sin duda, YOLOv3 Tiny, ofreciendo una velocidad de inferencia muy notable y superando en precisión al resto, llegando a alcanzar sin problemas el procesamiento en tiempo real.

14.1. Futuros pasos

Este proyecto está basado en un campo de investigación que está en pleno florecimiento. Debido a esto, hay muchas posibles ideas a probar con el objetivo de continuar esta investigación.

Una opción es trabajar con conjuntos de datos mas modernos que contengan imágenes en 3D, además de información de sensores como LIDAR entre otros. Esta información puede mejorar enormemente la precisión lograda con imágenes en 2D, aunque probablemente tenga una velocidad de inferencia menor. Estudiar como equilibrar cuanta información incorporar al modelo de forma que no sea perjudicial en términos de velocidad de inferencia y consumo es una posible línea de investigación.

Otra opción es trabajar con redes mas modernas, como puede ser YOLOV4 [8] publicada en este año. Estas redes son el estado del arte en este campo, y por tanto conocer a fondo su funcionamiento e investigar como poder incorporarlas o crear versiones reducidas es otra posible línea de investigación.

Finalmente, otra posible línea de investigación sería la de usar un mayor número de dispositivos Myriad para realizar inferencia en paralelo, con la finalidad de mejorar la velocidad de inferencia con respecto a un solo Myriad al menor coste posible, es decir, con otro Myriad.

CLOSURE

This project had two main objectives which were set at the beginning of the report. The aim was to answer the following two questions:

- Fine-tuning of a generalist network in order to learn how to identify vehicles, pedestrians and cyclists.
- Perfomance analysis of the inference in terms of precision, inference speed and power consumption of the fine-tuned network over a set of hardware devices.

To address the first question we focused on the study of different software solutions that allowed us to process the images in real time, using well known frameworks in the field of object detection as Darknet, Openvino and Tensorflow. These frameworks have allowed us to fine-tune generalist neural networks to learn how to identify vehicles, pedestrians and cyclists. In particular, we have worked with the following networks:

1. SSD Inception V2
2. SSD Mobile Lite
3. Faster R-CNN
4. YOLOv3
5. YOLOv3 Tiny

The fine-tuning has been done on the KITTI dataset and another dataset consisting of the extension of the KITTI dataset with some images from the Tshingua-Daimler and Berkeley Deepdrive Dataset sets. In section 11 we have analysed the results obtained during the fine-tuning of the different networks, from which we deduce that extending the initial dataset with a higher variety of bounding boxes makes it difficult to learn the network, which can be seen in a more unstable and slowly decreasing loss function.

As for the final results over the validation set, we deduce that a better performance is not achieved by increasing the training set, since all the networks have seen their accuracy reduced significantly. In particular, if the training set is to be increased, this increase must be with images very similar to those of the test set.

In this case, the datasets consist of images taken from a vehicle, all during the day and under favourable conditions. However, there are significant disparities between the data sets. KITTI is a set of panoramic images, therefore the width is 4 times the height of the image. This difference is crucial when resizing images as is done in the SSD Inception V2 and SSD Mobile Lite networks where the image is resized to a fixed dimension of 300x300, with the consequence that objects with a small initial width can become imperceptible.

On the other hand, Berkeley Deepdrive Dataset has objects with a smaller surface area than in the KITTI. This is due to the fact that the labels in this set consist of all the objects visible in the image, even some so far back in the image that their surface area is less than 20 pixels. Therefore, in networks where resizing occurs, the dimensions of the images are reduced, causing these objects to have a minimum surface area. This makes it very difficult for object detectors to identify them in the image. Moreover, the size of the anchors has not been modified to adapt it to the peculiarities of the data set, being an additional difficulty.

As for the second objective, in the sections **??** and **??** we have compared the accuracy, consumption and inference speed of the networks on both datasets and on three devices in the case of the SSD Inception V2, SSD Mobile Lite and Faster R-CNN networks:

- CPU
- Integrated low-power GPU
- Myriad

and in the case of YOLOv3 and YOLOv3 Tiny networks on a high performance GPU.

The results determine that the best networks to achieve the goal of autonomous driving with regard to accuracy are YOLOv3 and YOLOv3 Tiny. Both networks offer significantly better accuracy per class than the other networks studied. Another vital aspect in the context of autonomous driving is the inference speed. In this case YOLOv3 Tiny is up to five times faster than YOLOv3. These networks have not been able to run on the same devices as the rest. However, as we can see in the comparative graph **??**, evaluating the exchange of processing-consumption speed measured on two similar architectures, in this case both GPUs, we can see how they continue to be more promising than the rest of the networks.

In addition to the inference speed, the vehicle autonomy is crucial in autonomous driving and, therefore, the consumption of the different devices evaluated. The consumption of the devices and the models executed under them have been analysed in the section çonsumption". In order to jointly evaluate the performance of a model and the efficiency of the hardware device on which it has been executed, the FPS/W metric has been defined, which relates inference speed with consumption. The result is that models that have a one phase architecture, i.e. SSD and YOLO networks, are more efficient in terms of speed/consumption. This conclusion is natural considering the low inference speed of Faster R-CNN compared to the rest.

Focusing on the hardware devices, there is one that stands out from the rest, and that is Myriad. This device designed to perform neural network inference in an optimal way offers a minimum consumption, specifically 1W, and therefore is the most efficient device

in terms of speed/consumption. It is remarkable how despite its low power consumption, comes to make real-time inference SSD Inception V2 and SSD Mobile Lite networks. The GPU integrated with FP16 precision offers acceptable inference speeds under a consumption of 15W, being therefore the high performance GPU and the CPU the least efficient devices, because of its high consumption, especially in the first case; and because it is not specialized in these tasks in the case of the CPU.

In conclusion, after the study and comparisons made, the object detection architecture or model among all those analyzed in the state of the art is, with no doubt, YOLOv3 Tiny, offering a very remarkable inference speed and surpassing in precision the rest, reaching without problems the real time processing.

14.2. Future steps

This project is based on a field of research that is in full bloom. Because of this, there are many possible ideas to be tested in order to continue this research.

One option is to work with more modern data sets containing 3D images, as well as sensor information such as LIDAR among others. This information can greatly improve the accuracy achieved with 2D images, although it will probably have a lower inference speed. Studying how to balance how much information to incorporate into the model so that it is not detrimental in terms of inference speed and consumption is a possible line of investigation.

Another option is to work with more modern networks, such as YOLOV4 [8] published this year. These networks are the state of the art in this field, and therefore to know in depth its operation and investigate how to incorporate them or create reduced versions is another possible line of research.

Finally, another possible line of research would be to use a greater number of Myriad devices to perform parallel inference, in order to improve the speed of inference with respect to a single Myriad at the lowest possible cost, that is, with another Myriad.

Referencias

[1] G. E. Hinton A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural networks, 2012.

[2] S. Chintala G. Chanan-E. Yang Z. DeVito Z. Lin A. Desmaison L. Antiga A. Paszke, S. Gross and A. Lerer. Automatic differentiation in pytorch, 2017.

[3] Ross Girshick Abhinav Shrivastava, Abhinav Gupta. Training region-based object detectors with online hard example mining, 2016.

[4] https://medium.com/@andersasac/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9.

[5] Raquel Urtasun Andreas Geiger, Philip Lenz. Are we ready for autonomous driving? the kitti vision benchmark suite, 2012.

[6] Anastasios Doulamis Athanasios Voulodimos, Nikolaos Doulamis and Eftychios Protopapadakis. Deep learning for computer vision: A brief review, 2018.

[7] Golnaz Ghiasi Tsung-Yi Lin Jonathon Shlens Quoc V. Le Barret Zoph, Ekin D. Cubuk. Learning data augmentation strategies for object detection, 2019.

[8] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[9] Yangqing Jia Pierre Sermanet-Scott E. Reed Dragomir Anguelov Dumitru Erhan Vincent Vanhoucke Christian Szegedy, Wei Liu and Andrew Rabinovich. Going deeper with convolutions, 2015.

[10] COCO dataset. `http://cocodataset.org/`. Accessed: 2020-05-30.

[11] https://medium.com/@SeoJaeDuk/archived-post-understanding-and-visualizing-convolutional-neural-networks-d6da3e2851cf.

[12] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[13] https://cs231n.github.io/convolutional-networks/.

[14] J. Schmidhuber D. C. Ciresan, U. Meier. Transfer learning for latin and chinese characters with deep neural networks, 2012.

[15] J. Schmidhuber D. Ciresan, U. Meier. Multi-column deep neural networks for image classification, 2012.

[16] Darknet by AlexeyAB. `https://github.com/AlexeyAB/darknet`. Accessed: 2020-05-30.

[17] https://en.wikipedia.org/wiki/DARPA_Grand_Challenge_(2004).

[18] https://www.darpa.mil/news-events/2014-03-13.

[19] Xin Wang Wenqi Xian-Yingying Chen Fangchen Liu Vashisht Madhavan Trevor Darrell Fisher Yu, Haofeng Chen. Bdd100k: A diverse driving dataset for heterogeneous multitask learning, Mayo 2018.

[20] https://www.discovermagazine.com/technology/the-driverless-car-era-began-more-than-90-years-ago.

[21] S. Osindero G. E. Hinton and Y.-W. Teh. A fast learning algorithm for deep belief nets, neural computation, 2006.

[22] Ross B. Girshick. Fast r-cnn, 2015.

[23] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S. Lew. Deep learning for visual understanding: A review, 2016.

[24] Bhiksha Raj Haohan Wang. On the origin of deep learning, 2017.

[25] He, Chang-Wei Huang, Liqing Wei, Lingling Li, and Guo Anfu. Tf-yolo: An improved incremental network for real-time object detection. *Applied Sciences*, 9:3225, 08 2019.

[26] https://www.firstpost.com/tech/news-analysis/now-upload-share-1-8-billion-photos-everyday-meeker-report-3652169.html.

[27] https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

[28] L. Xu J. S. J. Ren. On vectorization of deep convolutional neural networks for vision tasks, 2015.

[29] Kaiming He Jifeng Dai, Yi Li and Jian Sun. R-fcn: object detection via region-based fully convolutional networks, 2016.

[30] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. Deep residual learning for image recognition, 2015.

[31] Theo Gevers Arnold W. M. Smeulders Koen E. A. van de Sande, Jasper R. R. Uijlings. Segmentation as selective search for object recognition, 2011.

[32] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2016.

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[34] Grace W. Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future, 2020.

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed C.Y. Fu, and A.C. Berg. Single shot mutlibox detector. https://arxiv.org/pdf/1512.02325.pdf, 2016.

[36] P. Barham E. Brevdo Z. Chen C. Citro G. S. Corrado A. Davis J. Dean M. Devin S. Ghemawat I. Goodfellow A. Harp G. Irving M. Isard Y. Jia R. Jozefowicz L. Kaiser M. Kudlur J. Levenberg D. Mane R. Monga S. Moore D. Murray C. Olah M. Schuster J. Shlens B. Steiner I. Sutskever K. Talwar P. Tucker V. Vanhoucke V. Vasudevan F. Viegas O. Vinyals P. Warden M. Wattenberg M. Wicke Y. Yu M. Abadi, A. Agarwal and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015.

[37] Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen Mark Sandler, Andrew Howard. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[38] John McCarthy. Computer controlled cars, 1968.

[39] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity, bulletin of mathematical biology, 1943.

[40] Radio-driven auto runs down escort; wireless directs test car in wobbly course through heavy broadway traffic. escapes fire engine crash defect in steering mechanism blamed for erratic action – new demonstration planned. https://www.discovermagazine.com/technology/the-driverless-car-era-began-more-than-90-years-ago, Julio 1925.

[41] https://en.wikipedia.org/wiki/Futurama_(New_York_World%27s_Fair).

[42] No hands across america official press release. https://www.cs.cmu.edu/~tjochem/nhaa/official_press_release.html, 1995.

[43] OpenVINO toolkit. `https://docs.openvinotoolkit.org/`. Accessed: 2020-05-30.

[44] Dean A. Pomerleau. Neural network perception for mobile robot guidance, 1992.

[45] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[46] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[47] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[48] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[49] https://medium.com/@jonathan_hui/what-do-we-learn-from-region-based-object-detectors-faster-r-cnn-r-fcn-fpn-7e354377a7c9.

[50] Trevor Darrell Ross B. Girshick, Jeff Donahue and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[51] Q. Yang S. J. Pan. A survey on transfer learning, ieee transactions on knowledge and data engineering, 2010.

[52] S. Hido S. Tokui, K. Oono and J. Clayton. Chainer: a nextgeneration open source framework for deep learning, in proceedings of workshop on machine learning

systems (learningsys) in the twenty-ninth annual conference on neural information processing systems (nips), 2015.

[53] SAE. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. https://www.sae.org/misc/pdfs/automated_driving.pdf, 2018.

[54] https://www.tesla.com/blog/secret-tesla-motors-master-plan-just-between-you-and-me.

[55] Ross B. Girshick Shaoqing Ren, Kaiming He and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks, 2015.

[56] Prof. R. P. Bijwe Shweta N. Dethe, Varsha S. Shevatkar. Google driverless car, Mayo 2018.

[57] https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06.

[58] K. Chen T. Mikolov, I. Sutskever. Distributed representations of words and phrases and their compositionality, 2013.

[59] Tensorflow Object Detection model zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`. Accessed: 2020-05-30.

[60] Tensorflow Object Detection API. `https://github.com/tensorflow/models/tree/master/research/object_detection`. Accessed: 2020-05-30.

[61] Bala Kumar Todd Jochem, Dean Pomerleau and Jeremy Armstrong. A portable navigation platform, 1995.

[62] https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.

[63] waymo.com.

[64] Y. Yang H. Xiong M. Braun S. Pan K. Li X. Li, F. Flohr and D. M. Gavrila. A new benchmark for vision-based cyclist detection, June 2016.

[65] J. Donahue S. Karayev J. Long R. Girshick S. Guadarrama Y. Jia, E. Shelhamer and T. Darrell. Caffe: Convolutional architecture for fast feature embedding, 2014.

[66] J. Denker Y. LeCun, B. Boser. Handwritten digit recognition with a back-propagation network, advances in neural information processing systems 2, 1990.

[67] JS Denker D Henderson RE Howard W Hubbard Y LeCun, B Boser. Backpropagation applied to handwritten zip code recognition, 1989.

[68] Shou-tao Xu Xindong Wu Zhong-Qiu Zhao, Peng Zheng. Object detection with deep learning: A review, 2019.