



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo fin de Máster

Máster Universitario en Ingeniería Informática

Análisis de datos para la detección de objetos en vehículos autónomo

**Realizado por:
Rafael Manzano Fuentes**

**Dirigido por:
María del Mar Martínez Ballesteros
Manuel Carranza García**

**Departamento
Lenguajes y Sistemas Informáticos**

**Convocatoria <Segunda convocatoria extendida>
Curso <2023/2024>**

Sevilla, Octubre 2024

Agradecimientos

... agradecimientos ...

Resumen

... resumen en español ...

Abstract

... resumen en inglés ...

Índice general

Índice general **IV**

Índice de figuras **VII**

Índice de cuadros **XI**

1 Introducción **1**

1.1 Alcance **2**

1.2 Objetivos **3**

 1.2.1 Objetivos personales **3**

 1.2.2 Objetivos profesionales **4**

 1.2.3 Objetivos académicos **4**

2 Planificación y metodología de trabajo **5**

2.1 Planificación temporal **5**

2.2 Metodología de trabajo **6**

2.3 Estimación de costes **7**

3 Estado del arte **9**

3.1	Introducción	9
3.2	Tecnologías	9
3.2.1	Evolución Coche Autónomo	10
3.2.2	Sensorización	10
3.2.3	Inteligencia Artificial y sus aplicaciones	13
3.2.4	Redes Neuronales Convolucionales	14
3.2.5	Modelos de visión artificial y detección de objetos	18
3.3	Análisis de tecnologías	22
3.3.1	Modelos basados en dos etapas: R-CNN y variantes	23
3.3.2	RRPN: Radar Region Proposal Network	31
3.3.3	Modelos de detección en una etapa	34
4	Desarrollo de la solución	42
4.1	Introducción	42
4.2	Justificación	42
4.3	Arquitectura/Diseño/Aspectos relevantes de la implementación	43
4.3.1	Datasets utilizados	43
4.3.2	Modelos utilizados	45
4.4	Proceso de experimentación	48
5	Resultados	51
5.1	Resultados de YOLOv8	51
5.1.1	Resultados de YOLOv8 para el dataset NuScenes	51

5.1.2 Resultados para Waymo Open Dataset	54
5.2 Resultados de FasterR-CNN	56
6 Planificación final, Conclusiones y Trabajo Futuro	58
6.1 Planificación final, desviación y motivos	58
6.2 Conclusiones, resumen del proyecto y sus resultados	59
6.3 Objetivos cumplidos y trabajo futuro	59
6.4 Reflexión final	60
7 Bibliografía	62
Referencias	62
8 ANEXOS	67

Índice de figuras

1.1 Funcionamiento coche autónomo.	2
1.2 Ejemplo detección de peatones.	3
2.1 Diagrama de Gantt conteniendo la planificación temporal del Trabajo de Fin de Máster	6
2.2 Metodología DSRM aplicada en el proyecto	7
3.1 Sensores presentes en los coches de Waymo.	11
3.2 Radar en coche autónomo.	12
3.3 LiDAR en coche autónomo.	13
3.4 Valor de activación de la función sigmoide.	15
3.5 Estructura general de una red neuronal.	16
3.6 Estructura red neuronal convolucional.	16
3.7 Funcionamiento capa convolucional	17
3.8 Funcionamiento MaxPooling.	17
3.9 Tipos de tareas de visión artificial.	18
3.10 Desafío detección de pájaros.	19
3.11 Explicación Transfer Learning en redes convolucionales.	20

3.12 Diferentes ejemplos de <i>data augmentation</i> geométrico.	21
3.13 Ejemplos data augmentation fotométrico.	21
3.14 Ejemplo de técnicas de borrado de regiones.	22
3.15 Ejemplo de técnicas de reemplazo e intercambio de regiones.	22
3.16 Ejemplo técnicas de combinación de regiones.	22
3.17 Arquitectura de modelos de localización de objetos en una etapa.	23
3.18 Arquitectura de modelos de localización de objetos en dos etapas.	24
3.19 Evolución de los modelos de detección.	25
3.20 Ejemplo de Non-maxima suppression aplicada con IOU.	25
3.21 Esquemática de la arquitectura de R-CNN.	25
3.22 Arquitectura Fast-RCNN.	26
3.23 Arquitectura de Faster R-CNN.	28
3.24 Arquitectura del RPN.	29
3.25 Resultados de comparación OverFeat vs Faster R-CNN.	31
3.26 Imagen que representa los anchor boxes en RRPN.	31
3.27 Intersection over Union de forma esquematizada.	33
3.28 Comparación entre las bounding boxes reales, SS y RRPN.	34
3.29 Funcionamiento base de YOLO.	35
3.30 Proceso de funcionamiento de YOLO.	36
3.31 Red convolucional original de YOLO.	36
3.32 Comparación de modelos de detección en tiempo real.	38
3.33 Estructura de Darknet-19.	39

3.34 Estructura de Darknet-53.	40
3.35 Estructura de YOLOv4.	41
4.1 Distribución clases subset Waymo.	44
4.2 Distribución de clases del dataset NuScenes mini.	46
4.3 Ejemplo Faster R-CNN.	46
4.4 Arquitectura VGG.	47
4.5 Archiectura of MobilenetV3.	48
4.6 Estructura YOLOv8.	49
5.1 Evolución métricas durante el proceso de entrenamiento de NuScenes en la versión Preentrenada sobre ImageNet	52
5.2 Evolución métricas durante el proceso de entrenamiento de NuScenes en la versión desde cero	52
5.3 Curva Precision-Confidence para YOLOv8 Preentrenado usando NuScenes .	53
5.4 Curva Precision-Recall para YOLOv8 Preentrenado usando NuScenes . .	53
5.5 Curva Precision-Confidence para YOLOv8 desde cero usando NuScenes . .	53
5.6 Curva Precision-Recall para YOLOv8 desde cero Preentrenado usando NuScenes	53
5.7 Curva Precision-Confidence para YOLOv8 Preentrenado usando Waymo . .	54
5.8 Curva Precision-Recall para YOLOv8 Preentrenado usando Waymo . . .	54
5.9 Evolución métricas durante el proceso de entrenamiento de Waymo en la versión Preentrenada sobre ImageNet	54
5.10 Evolución métricas durante el proceso de entrenamiento de Waymo en la versión desde cero	55
5.11 Curva Precision-Confidence para YOLOv8 desde cero usando Waymo . .	55

5.12 Curva Precision-Recall para YOLOv8 desde cero usando Waymo	55
5.13 Enter Caption	57

Índice de cuadros

5.1	Tabla de métricas por clase para YOLOv8 sobre el dataset NuScenes, Pre-entrenado	51
5.2	Tabla de métricas por clase para YOLOv8 sobre el dataset NuScenes, desde cero	52
5.3	Tabla de métricas por clase para YOLOv8 sobre el dataset Waymo, Pre-entrenado	55
5.4	Tabla de métricas por clase para YOLOv8 sobre el dataset Waymo, desde cero	56
6.1	Tabla comparativa de horas estimadas vs horas finales	58

CAPÍTULO 1

Introducción

En la última década, los avances tecnológicos han revolucionado la industria automotriz de una manera sin precedentes. Uno de los desarrollos más significativos en esta transformación ha sido la llegada de los vehículos autónomos o vehículos sin conductor. Estos vehículos representan un cambio radical en la forma en que se concibe el transporte personal y la movilidad en general (www.xataka.com, 2016).

Los coches autónomos están diseñados para funcionar de manera independiente, sin intervención humana directa, mediante la utilización de una variedad de sensores, cámaras, radares, y sistemas de procesamiento de datos avanzados. Esto plantea no solo oportunidades significativas en términos de comodidad y eficiencia, sino también desafíos cruciales relacionados con la seguridad, la regulación y la aceptación pública.

Los objetivos principales de la implantación de este tipo de vehículos son: reducir los accidentes de tráfico, aliviar la congestión del tráfico, aumentar la accesibilidad para personas con movilidad reducida y revolucionar la logística y el transporte de mercancías. Sin embargo, para que esta visión se haga realidad, es necesario abordar una serie de cuestiones complejas como es el efecto de los datos usados para el desarrollo de dicha tecnología.

Estos datos incluyen información sobre la posición del vehículo, velocidad, comportamiento del mismo, condiciones del tráfico, interacciones con otros usuarios de la carretera y mucho más. El análisis de estos datos no solo es esencial para mejorar la seguridad y la eficiencia de los coches autónomos, sino que también desempeña un papel vital en la investigación y desarrollo continuo de esta tecnología.

Este Trabajo de Fin de Máster se centra en explorar algunos ejemplos utilizados en el desarrollo de dicha tecnología. A lo largo del mismo, se examinarán los desafíos y las oportunidades que surgen al trabajar con estos datos, así como las técnicas y herramientas que se utilizan para su procesamiento y análisis. También se considerarán las implicaciones éticas y legales relacionadas con la recopilación y el uso de estos datos, y cómo esto afecta



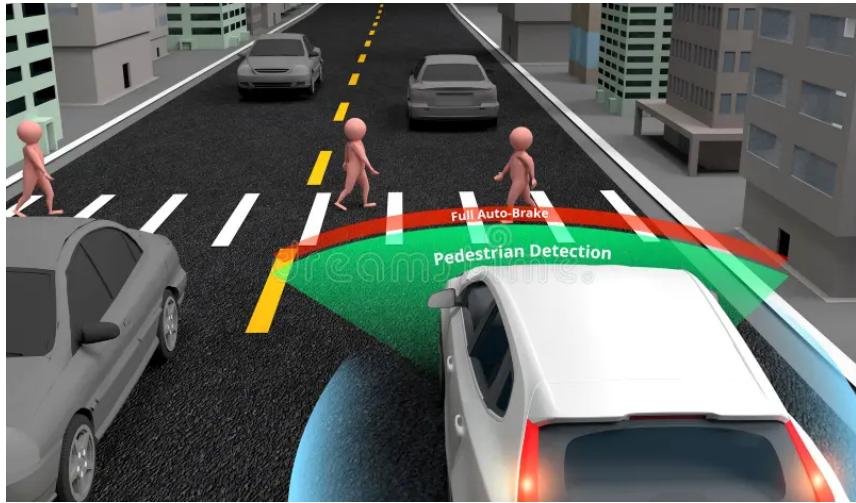
Figura 1.1: Funcionamiento coche autónomo.

a la adopción y aceptación de los coches autónomos en la sociedad.

En resumen, este trabajo tiene como objetivo arrojar luz sobre la importancia crítica del análisis de datos en el contexto de los coches autónomos, destacando su impacto en la seguridad, la eficiencia y el futuro de la movilidad. A medida que se avanza hacia una era donde los vehículos autónomos están cada vez más presentes en nuestras vidas, comprender cómo se gestionan y utilizan los datos se convierte en un aspecto fundamental de esta revolución tecnológica.

1.1– Alcance

Este trabajo se basa en la evaluación del estado del arte de la detección de objetos aplicada a los coches autónomos. La detección de objetos es un campo que tiene multitud de aplicaciones como detección de objetos mediante imágenes satelitales, detección de texto en imágenes, supervisión, detección de caras, pero, en la que se centra este trabajo es en la conducción autónoma. Aunque es cierto que no sólo es aplicable a la conducción autónoma, sino como asistencia a la conducción. Por ejemplo, en la Unión Europea, la normativa es muy restrictiva en cuanto al uso de coches autónomos, pero es mucho más laxa con los vehículos con conducción asistida, siendo de hecho obligatorio un mínimo de asistencia a la conducción para que un coche pueda ser comercializado. Esto se debe al dilema ético que surge en la toma de decisiones, principalmente cuando alguna vida (ya sea la del conductor o el peatón) esté en juego. Es por ello, que es necesario evaluar cómo de precisos son este tipo de modelos y en qué casos existe una mayor dificultad en la detección de peatones o ciclistas, siendo mucho más importante el hecho de un falso positivo (se detecta un peatón pero no existe), que un falso negativo (no se detecta peatón, pero existe), ya que cuando no se detecta el agente externo como el peatón correctamente, el vehículo se puede convertir en un peligro potencial para todo el entorno. Este proceso consiste en el uso de varios tipos de sensores para la correcta detección, aunque este trabajo se centrará en la detección mediante el uso de imágenes en 2D.



Fuente: https://www.shutterstock.com/shutterstock/photos/1244887726/display_1500/stock-photo-pedestrian-detection-technology-autonomous-self-driving-car-with-lidar-radar-and-wireless-signal-1244887726.jpg

Figura 1.2: Ejemplo detección de peatones.

1.2– Objetivos

En este trabajo, se parte de la hipótesis de que los coches autónomos y con conducción asistida presentan un dilema ético y tecnológico, debido a la dificultad en la toma de decisiones que existen en todo el proceso.

De acuerdo con ello, el objetivo principal de este trabajo es el de evaluar la capacidad de detección de vehículos, personas y ciclistas mediante las imágenes que captan los coches autónomos a través de las diferentes cámaras de las que disponen. Para realizar esta tarea, hay una gran variedad de modelos y tecnologías que se pueden aplicar, y por lo tanto, la principal hipótesis de investigación es conocer el estado del arte de la detección de objetos, para posteriormente aplicarlos a conjuntos de datos específicos de coches autónomos, para conseguir dicho fin serían necesarios los subobjetivos que se detallan en los siguientes apartados.

1.2.1. Objetivos personales

En este caso habría dos objetivos principales a perseguir a nivel personal:

1. **Mejora de la capacidad de estudio de estudios científicos:** En determinados casos puede ser complicado comprender determinados estudios por la complejidad del marco teórico detrás, pero asentando los conocimientos teóricos base del mismo será mucho más sencillo el estudio de los mismos.
2. **Mayor comprensión de un posible negocio en auge:** El modelo de negocio liderado por Waymo basado en el servicio de taxis autónomos puede provocar un antes y un después en la forma de moverse en las ciudades del ser humano, por lo

cual es muy interesante la comprensión de la técnica que hace posible este modelo de negocio y la complejidad de su aplicación. Además del dilema ético que presenta que máquinas tomen decisiones de la que dependen las vidas humanas.

1.2.2. Objetivos profesionales

Estos objetivos están más relacionados a nivel profesional y de utilidad en el entorno laboral, ya sea por propio interés o por oportunidades que se puedan dar en un futuro:

1. **Coordinación esfuerzo laboral-académico:** Debido a la necesidad de la coordinación en la realización de este Trabajo de Fin de Máster con un trabajo a jornada completa, ha sido necesaria una gran gestión del tiempo para aprovechar al máximo el tiempo invertido en la realización del mismo.
2. **Possibles oportunidades laborales:** Es un tema que presenta provocar gran interés para trabajar en el mismo ya que combina dos mundos muy interesantes como son el de la computación avanzada y la automoción.

1.2.3. Objetivos académicos

Estos objetivos son aquellos completamente enfocados en el estudio de la literatura y la utilización de recursos disponibles de forma abierta:

1. **Uso de fuentes de datos abiertas:** Se hará uso de conjuntos de datos abiertos a cualquier persona y para uso no lucrativo como el académico. El uso de este tipo de datos pasa por hacer un estudio de qué ofrecen y cómo se pueden utilizar, junto con qué tipo de preprocesamiento necesitan.
2. **Desarrollo de modelos de detección de objetos:** Se desarrollarán modelos de detección de objetos basado en Redes Neuronales Convolucionales, para poder así reconocer cómo funcionan en la realidad los coches autónomos y cómo comprueban los objetos que existen a su alrededor.

CAPÍTULO 2

Planificación y metodología de trabajo

En este capítulo de planificación, se describirán las etapas y estrategias esenciales para llevar a cabo el Trabajo de Fin de Máster. Se presentará un cronograma detallado de actividades y la asignación de recursos necesarios, lo que permitirá una gestión efectiva del tiempo y una organización adecuada del proceso, ya que la planificación es un aspecto crucial para asegurar el cumplimiento de los plazos establecidos y la obtención de resultados de calidad.

2.1– Planificación temporal

Para planificar la realización de este Trabajo de Fin de Máster a lo largo del tiempo se han planteado las siguientes tareas:

1. **Estudio de posibles temas y propuestas:** Debido al gran abanico de posibilidades y propuestas, es necesario dedicar un tiempo de análisis para ver el tipo de propuesta que se va a realizar y la temática del mismo. (10 horas)
2. **Estudio de aplicaciones en el mundo real de la propuesta:** Es necesario comprobar si la solución propuesta se aplica en algún aspecto del mundo real o si podría tenerla en un futuro. (25 horas)
3. **Estudio del estado del arte de la temática:** Se realizará una exploración concienzuda de los diferentes estudios científicos y aplicaciones actuales de la temática seleccionada. (50 horas)
4. **Creación del entorno de trabajo:** Será necesario crear el entorno de desarrollo en el que se usará lenguaje Python (Van Rossum y Drake, 2009) y la instalación de todos los paquetes propios necesarios para la implementación como *Ultralytics* (Jocher, Chaurasia, y Qiu, 2023) y PyTorch (Paszke y cols., 2019). (30 horas)

5. **Desarrollo del código:** Se desarrollará el código necesario para implementar los detectores de objetos así como la extracción de las métricas de los mismos. (100 horas)
6. **Redacción de la memoria:** Este proceso será simultáneo al resto, puesto que la parte de la memoria correspondiente a cada una de las tareas se hará durante la realización de la misma. (100 horas)
7. **Extracción de conclusiones y mejoras:** Esta última tarea se centrará en extraer las conclusiones de los resultados obtenidos durante la experimentación, así como estudiar futuras mejoras en la implementación. (30 horas)

En la Figura 2.1 se muestra un diagrama de Gantt (Maylor, 2001), en el cual se muestra el desarrollo de la planificación temporal del Trabajo de Fin de Máster.

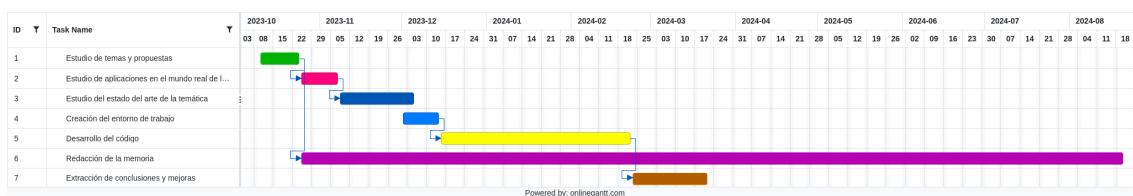
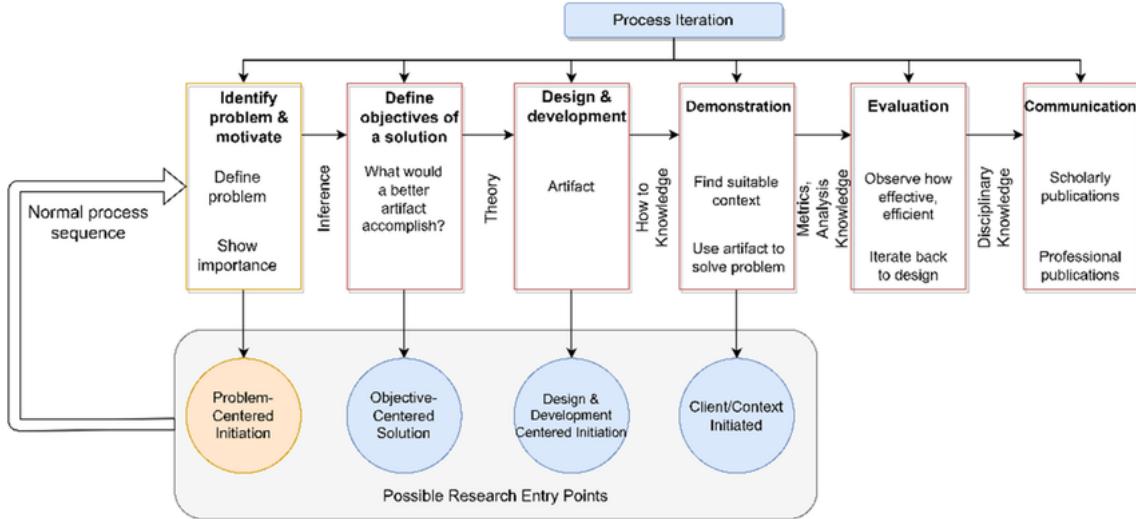


Figura 2.1: Diagrama de Gantt conteniendo la planificación temporal del Trabajo de Fin de Máster

2.2– Metodología de trabajo

Este Trabajo de Fin de Máster se orienta en el estudio de bibliografía científica y su aplicación al entorno del vehículo autónomo. Por ello, encaja perfectamente con proyectos de investigación focalizados en tecnología y aplicación al mundo real. Dado el tipo de proyecto una metodología que encaja a la perfección es DSRM o ***Design Science Research Methodology***, la cual se basa en los siguientes puntos (Hevner y Chatterjee, 2010), y cuyo proceso se refleja en la Figura 2.2:

- **Enfoque en la creación de artefactos:** DSRM se centra en el desarrollo de artefactos innovadores como modelos o sistemas.
- **Proceso de ciclo de investigación:** El cual consiste en: identificación del problema y motivación, definición de objetivos, diseño y desarrollo, evaluación y comunicación.
- **Iteraciones y refinamiento:** Esta metodología se centra en iterar en varias ocasiones, basándose en la evaluación y retroalimentación sobre anteriores iteraciones.
- **Contribución al conocimiento:** Gracias a la investigación y creación de artefactos se añadirán nuevos conocimientos al campo de investigación.



Fuente: (Sumarsono y cols., 2022)

Figura 2.2: Metodología DSRM aplicada en el proyecto

2.3– Estimación de costes

Uno de los costes adicionales de este TFM es la suscripción a la plataforma Overleaf para tener una mejor gestión de la redacción de la memoria, así como un tiempo de compilación más corto así como sincronización en la nube, la cual gracias a la licencia de estudiante ha sido de únicamente 7.99€.

Por otro lado, al utilizar redes neuronales, un framework compatible con procesamiento en GPU (PyTorch) y disponer de una tarjeta gráfica Nvidia RTX 2060, se ha hecho uso de la tecnología CUDA de Nvidia, la cual es extremadamente eficaz en el entorno de redes neuronales.

Al necesitar de un equipamiento muy potente en términos computacionales del mismo se ha hecho uso de un equipo con las siguientes características:

- **CPU:** Procesador AMD Ryzen 9 5900X
- **GPU:** Nvidia RTX 2060 6GB VRAM
- **Memoria RAM:** 4 módulos de 8GB cada uno a una frecuencia de 3200MHz CL16, con un total de 32GB.
- **Disco SSD:** Disco en estado sólido NVME de 256 dedicado exclusivamente al desarrollo del Trabajo.
- **Fuente de Alimentación:** Fuente de alimentación de 650W con certificación 80 Plus Gold.
- **Refrigeración CPU:** Refrigeración líquida de 240mm dedicada únicamente a la CPU.

Sumando el coste aproximado de la suscripción a la plataforma Overleaf, junto con el coste aproximado del equipo y el consumo de electricidad del equipo durante el proceso de entrenamiento de las redes neuronales suman un coste estimado de 1500€.

CAPÍTULO 3

Estado del arte

3.1– Introducción

El auge de esta tecnología ha comenzado en el último lustro, pero la investigación para su implementación lleva desarrollándose durante muchos años. Han comenzado a surgir diferentes fabricantes de automóviles que han convertido en los últimos años, que han convertido la conducción autónoma en el pilar fundamental de la empresa y en su imagen de marca. El mayor ejemplo de esta casuística es Tesla, que convirtió su sistema de conducción automática Autopilot (*Tesla Autopilot*, s.f.) en uno de los referentes dentro de la industria.

Otro de los factores que han influido en el crecimiento exponencial de la conducción autónoma son los grandes avances que se han llevado a cabo a nivel de hardware y sensores, esto es debido principalmente al auge de la Inteligencia Artificial y las Redes Neuronales (Muhammad, Ullah, Lloret, Ser, y de Albuquerque, 2021).

3.2– Tecnologías

Para que exista la posibilidad de crear coches autónomos de producción en masa, es necesario el desarrollo de múltiples tecnologías que han dado pasos agigantados a lo largo de los años, que de forma independiente no servirían para el desarrollo de este producto, pero que usadas de forma conjunta, sí que son de gran utilidad para crear coches autónomos o semi-autónomos.

3.2.1. Evolución Coche Autónomo

Desde comienzos del siglo XX, se han realizado diferentes pruebas de concepto para comprobar la viabilidad de la propuesta de la conducción autónoma. Las primeras pruebas de este tipo se realizaron en la década de 1930 con un distribuidor de coches en Milwaukee llamado Achen Motors (*Driver-less Car to be Demonstrated About City Streets next Saturday - Controlled Entirely by Radio*, 1932), el cual aún no era completamente autónomo, sino que estaba controlado por radio. A pesar de no ser totalmente autónomo, fue uno de los primeros acercamientos a esta tecnología.

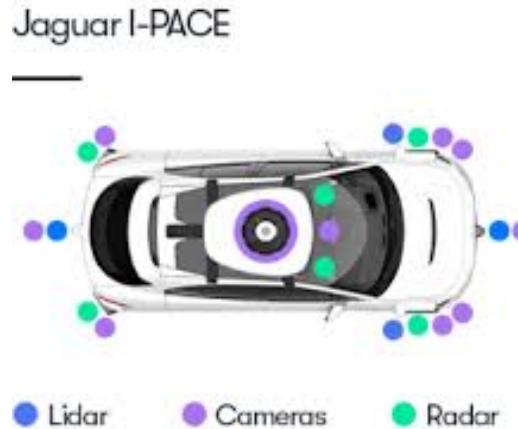
A lo largo de dicho siglo hubo muchos más proyectos en los que se involucraron muchas marcas muy conocidas como pueden ser Mercedes-Benz, Toyota u Honda. A pesar, de ello no fue hasta 1989 cuando la *Carnegie Mellon University*, se involucró en un proyecto promovido por la *DARPA (Defense Advanced Research Projects Agency)*, denominado *ALV* o *Autonomous Land driven Vehicle* el cual consistía en encontrar una solución funcional para la conducción 100 % autónoma en vehículos terrestres. En este proyecto, se implementó por primera vez el enfoque basado en redes neuronales, el cual es el enfoque usado en la actualidad, estableciendo así un estándar que se mantiene hasta nuestros días (Pomerleau, 1989).

A partir del año 2000, prácticamente todas las gigantes de la industria estaban en proceso de testeo de su tecnología de conducción autónoma. Uno de los grandes éxitos fue por parte de Audi el cual consiguió completar el recorrido de la famosa prueba *Pike's Peak* a velocidad de competición utilizando un Audi TTS que funcionaba de forma completamente autónoma (*Audi Autonomous Run in Pike's Peak*, 2010).

El último gran hito en la historia fue la comercialización de los primeros coches con tecnología *Full Self Driving*, lo que quiere decir que el coche sería capaz de hacer la funcionalidad completa sin intervención del conductor. Este logro lo consiguió Tesla con su *Autopilot*, a comienzos del 2015, la empresa distribuyó mediante una actualización de software en sus *Model S* lo cual llevó a un gran revuelo debido a los problemas legales en los que podría incurrir este tipo de funcionalidad, lo que llevó a su prohibición en muchos estados de EEUU.

3.2.2. Sensorización

Los sensores han sido de gran utilidad en muchos campos desde que fueron ideados: sensores de temperatura, de humedad, radares, cámaras, etc... A pesar de tener una gran utilidad en diversos campos, son un pilar fundamental para poder desarrollar un sistema de conducción autónoma; ya que es la única manera de que la inteligencia artificial encargada del manejo del vehículo consiga datos sobre el entorno que le rodea. En este caso existen dos tipos de sensores, sensores activos, los cuales son aquellos que buscan objetos en los alrededores utilizando ondas o cualquier método que interactúe con los objetos; y sensores pasivos, que simplemente obtienen información del entorno sin interactuar con el resto de objetos.



Fuente: <https://support.google.com/waymo/answer/9190838?hl=es-es>

Figura 3.1: Sensores presentes en los coches de Waymo.

Cámaras

Las cámaras son un básico en cuanto al equipamiento de los vehículos autónomos ya que son de gran utilidad para detectar y reconocer objetos utilizando algoritmos de inteligencia artificial como por ejemplo el algoritmo YOLO (Rashed y cols., 2020). Normalmente, se crea una vista en 360 grados del entorno de forma que así se tiene en cuenta toda la información posible y no se abandona ningún flanco del mismo. Este tipo de sensores tienen problemas como por ejemplo en situaciones de baja visibilidad (en los casos de cámaras de luz visible), además de que se necesita un hardware específico para el procesamiento de vídeo. Sin embargo, como solución a la falta de visibilidad también se utilizan cámaras de tipo infrarrojo que permite aumentar las capacidades de detección en dichas situaciones.

Radar

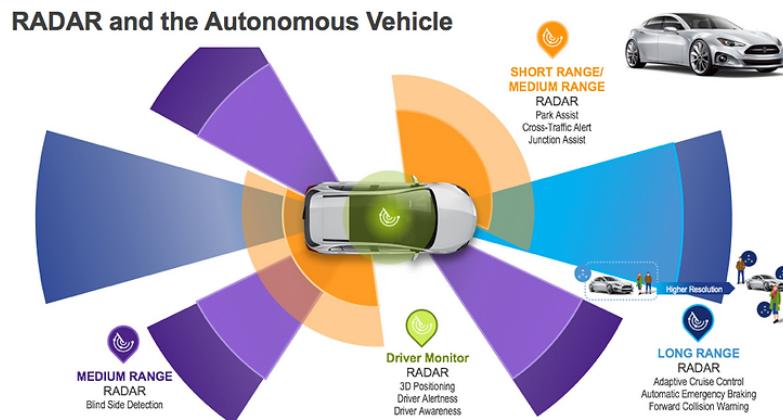
Los radares son otro tipo de sensores que son muy comunes en la gran mayoría de coches modernos debido a su utilidad para los sistemas de ayudas a la conducción como puede ser por ejemplo el control de crucero adaptativo. El principio de funcionamiento de este sistema es muy simple, éste emite ondas de radio hacia los objetos del entorno y recoge el rebote producido por los diferentes objetos. El tiempo que tardan las ondas en retornar es usado para calcular la distancia, ángulo y velocidad de los diferentes obtáculos (Software, s.f.).

Estos radares pueden operar a diferentes frecuencias en el intervalo [24 GHz-79 GHz] de forma que hay tres tipos de radares dependiendo de la frecuencia utilizada:

- Short-Range Radars (SRR): Son aquellos que permiten la monitorización, el mantenimiento de carril y el aparcamiento en vehículos autónomos.
- Medium-Range Radars (MRR): Son aquellos que permiten la detección de obtáculos en un rango de 100-150 metros con un ángulo respecto del sensor entre 30 y 160 grados.

- Long-Range Radars (LRR): Son aquellos que permiten el control de la distancia y frenado.

Este tipo de sensores opera con precisión milimétrica y asegura el poder detectar correctamente la posición y el movimiento que puedan tener todos los obstáculos en el entorno. En comparación con otro tipo de sensores, esta tecnología es muy confiable en condiciones de baja visibilidad como pueden ser la niebla, la lluvia o la nieve.



Fuente: (Behera y cols., 2022)

Figura 3.2: Radar en coche autónomo.

LiDAR

La tecnología *LiDAR* o *Light Detection and Ranging* es una tecnología de teledetección que aprovecha la luz láser para medir distancias y construir mapas del entorno. Funciona emitiendo pulsos láser y determinando el tiempo que tardan en reflejarse en los objetos y volver al sensor. Estos datos se utilizan para crear modelos tridimensionales del entorno. La tecnología LiDAR encuentra aplicación en múltiples sectores, como los vehículos autónomos, la geografía, la arqueología y la meteorología.

La principal diferencia entre el *LiDAR* y el Radar es el tipo de pulso usado, ya que en el caso del *LiDAR* se trata de pulsos de luz mientras que en el caso del Radar se trata de pulsos de radio.

Sin embargo, uno de los principales problemas de esta tecnología es que puede llegar a confundir objetos cercanos con lejanos y viceversa. Además tiene un funcionamiento mucho peor en condiciones complicadas como la lluvia, la niebla o la nieve respecto del Radar; además de tener un coste mucho mayor respecto de la misma.

Es por lo que esta tecnología cada vez está perdiendo más terreno respecto del radar, hasta el punto de que, en el momento que pierda la exclusividad de crear modelos tridimensionales del entorno posiblemente se descarte su uso completamente (Software, s.f.).



Fuente: (Buddi y cols., 2023)
Figura 3.3: LiDAR en coche autónomo.

3.2.3. Inteligencia Artificial y sus aplicaciones

La Inteligencia Artificial o IA, al igual que la humana, es un concepto complejo de definir. Aún no existe una definición formal y universalmente aceptada.

La Comisión Europea la define como sistemas de software (y posiblemente también de hardware) diseñados por humanos que, ante un objetivo complejo, actúan en la dimensión física o digital:

Percibiendo su entorno, a través de la obtención e interpretación de datos estructurados o no estructurados. Razonando sobre el conocimiento, procesando la información derivada de estos datos y decidiendo las mejores acciones para lograr el objetivo dado. Los sistemas de IA pueden usar reglas simbólicas o aprender un modelo numérico. También pueden adaptar su comportamiento al analizar cómo el medio ambiente se ve afectado por sus acciones previas. La historia de la IA se remonta a 1950, año en la que el matemático Alan Turing planteaba el dilema ético con la pregunta, *¿Pueden las máquinas pensar?*, en él, presenta el ejemplo *Juego de la Imitación* como un enfoque para determinar si las máquinas pueden ser consideradas inteligentes. El artículo plantea cuestionamientos importantes sobre la inteligencia artificial y sigue siendo una obra fundamental en el campo de la ciencia de la computación y la filosofía de la inteligencia artificial (Turing, 1950). Este campo a pesar de ser ampliamente seguido y estudiado de forma teórica, no pudo llevarse a la práctica hasta que la potencia de hardware y la cantidad de datos almacenados no alcanzó unas cotas nunca vistas a costes razonables. De esa forma los gigantes como *Google* o *Facebook* comenzaron a apostar muy fuerte en este campo hasta el punto de crear sus propios frameworks de inteligencia artificial (*Tensorflow* y *Caffe* respectivamente). Esta tecnología se ha aplicado con infinidad de finalidades tales como:

- **Procesamiento del Lenguaje Natural:** Aquellas inteligencias artificiales que son capaces de comprender el lenguaje natural e interactuar con el mismo imitando

el comportamiento de una interacción entre dos humanos, este caso es uno de los más mediáticos en la actualidad gracias al producto desarrollado por la compañía **OpenAI**: ChatGPT.

- **Forecasting:** El análisis de series temporales, siempre ha sido un campo de estudio muy importante y la inteligencia artificial ha mostrado ser una gran solución a esta problemática gracias a las redes neuronales recurrentes que han permitido la predicción de posibles hechos que ocurrirán en el futuro gracias a una serie de datos históricos almacenados.
- **Visión por ordenador:** El caso que ocupa en este trabajo, hay diferentes formas de aplicar la inteligencia artificial en el campo de la visión por ordenador, como puede ser la clasificación de imágenes que consiste en asignar una clase a una imagen o como el caso que se aplica en los coches autónomos la detección de objetos, proceso que trata de dividir una imagen en diferentes regiones, las cuales representan un objeto identificada con una determinada confianza.

3.2.4. Redes Neuronales Convolucionales

En primer lugar, antes de presentar las redes neuronales convolucionales es necesario comprender qué son y en qué consisten las redes neuronales. Una red neuronal artificial (Wu y Feng, 2018) establece un modelo computacional con nodos interconectados que simulan una red neuronal humana. En cada nodo se aplica una función de activación a los datos de entrada obteniendo una versión modificada de los mismos, mientras que cada enlace entre nodos representa un peso que pondera el efecto de dicho nodo. Cuando el dato de entrada de una neurona provenga de otra neurona, éste está sujeto a una operación matemática que consiste en la suma del producto de los pesos por el valor de la función de activación de la neurona de la que procede dicho dato, siguiendo la Ecuación 3.1

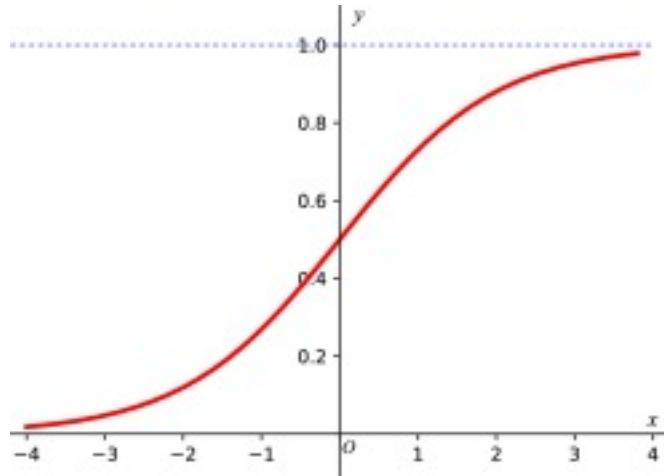
$$f(x) = k \left(\sum_{i=0}^n w_i * g(x) \right) \quad (3.1)$$

siendo i cada una de las neuronas de entrada del nodo, n el número de neuronas de entrada total y $g(x)$ el valor de salida de la neurona anterior, y k la función de activación a aplicar, una de las funciones más utilizadas en este caso es la función sigmoide (Chen y cols., 2024) la cual se expresa con la ecuación 3.2.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3.2)$$

cuyo valor de salida es un valor continuo siempre en el rango [0 - 1], siendo que cuanto mayor es el valor de entrada más cercano a 1 será el valor de salida y cuanto menor sea el valor de entrada, más cercano a 0 será el valor de salida, este funcionamiento se resume con la Figura 3.4.

Las redes neuronales son capaces de resolver infinidad de problemas, tanto de regresión como de clasificación y constan de infinidad de unidades de neuronas que generalmente se



Fuente: (Chen y cols., 2024)

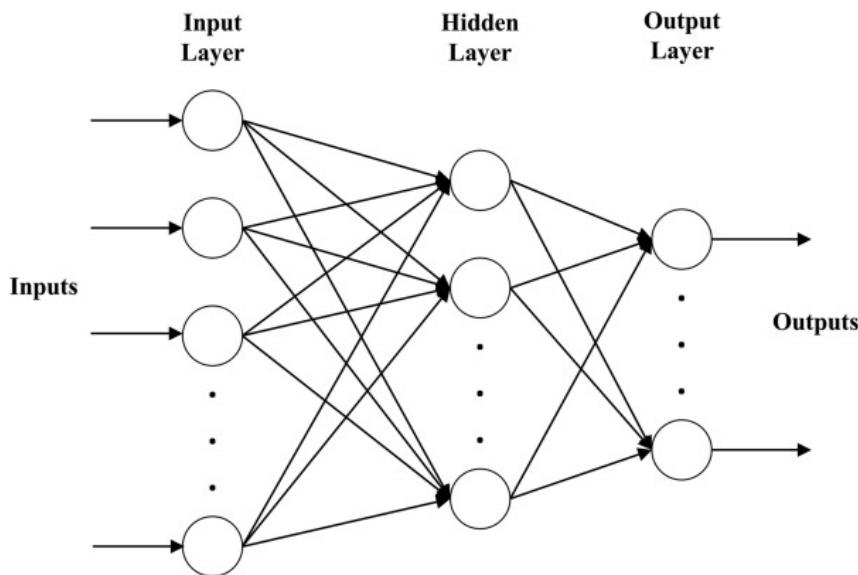
Figura 3.4: Valor de activación de la función sigmoide.

organizan en capas, siendo que cada capa puede constar de una o más neuronas. La estructura general de las redes neuronales consta de tres tipos de capas, aunque su estructura podría modificarse en casos más específicos, y que se puede resumir en la Figura 3.5:

- **Capa de entrada:** La capa de entrada es la primera capa de la red, y por lo general tiene tantas neuronas como número de entradas tiene el modelo. Las neuronas que existen en esta capa no suele aplicar ningún tipo de transformación a los datos, salvo algún tipo de escalado o normalización; y solo sirven como transición para la primera capa oculta.
- **Capas ocultas:** Las capas ocultas, son todas aquellas capas que existan entre la de entrada y la de salida. El tamaño de cada una de estas capas es completamente independiente del número de datos de entrada o de salida, pero teniendo en cuenta que la primera de ellas está conectada con la capa de entrada. En estas capas se aplica la transformación explicada anteriormente a los datos de manera que se puede obtener un deducción de conocimiento.
- **Capa de salida:** La capa de salida es la última capa de la red neuronal, y consta de tantas unidades como salidas tenga la red neuronal. En este paso, generalmente sí se aplica un función de activación al igual que en las capas ocultas, que en muchos casos puede ser la misma función usada anteriormente, aunque existe una función muy importante que se utiliza en tareas de clasificación que es la función Softmax, que se define con la ecuación 3.3 (J. Ren y Wang, 2023), la cual se encarga de transformar un conjunto de números en probabilidades.

$$\frac{e^{z_k}}{\sum_{k=1}^K e^{z_k}} \quad (3.3)$$

Para que una red neuronal sea capaz de resolver tareas es necesario entrenarlas, el entrenamiento de una red neuronal que consiste en el ajuste de los pesos de los enlaces entre neuronas. Esto se lleva generalmente se hace mediante un algoritmo denominado



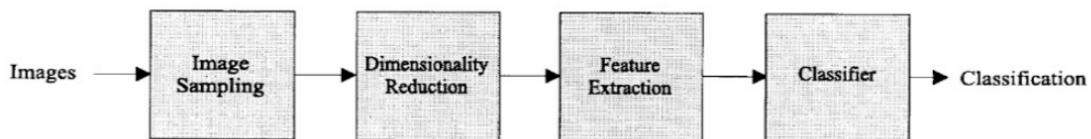
Fuente: (Venkateswarlu y Karri, 2022)

Figura 3.5: Estructura general de una red neuronal.

retropropagación del error, el cual consiste en la propagación de datos etiquetados hasta obtener un valor de salida y recalcular los pesos de los enlaces realizando el paso contrario retropropagando la salida hacia atrás aplicando derivadas para el cálculo.

A pesar de que la red neuronal fue un modelo computacional que surgió en la década de 1950, debido a la incapacidad de poder lidiar con ellos se abandonaron y no fue hasta la década de 1980 (1989), cuando gracias a la aplicación de la técnica de **retropropagación del error**, el informático Yann LeCun (LeCun y cols., 1989), crea el primer prototipo de red neuronal convolucional para el reconocimiento de códigos postales escritos a mano.

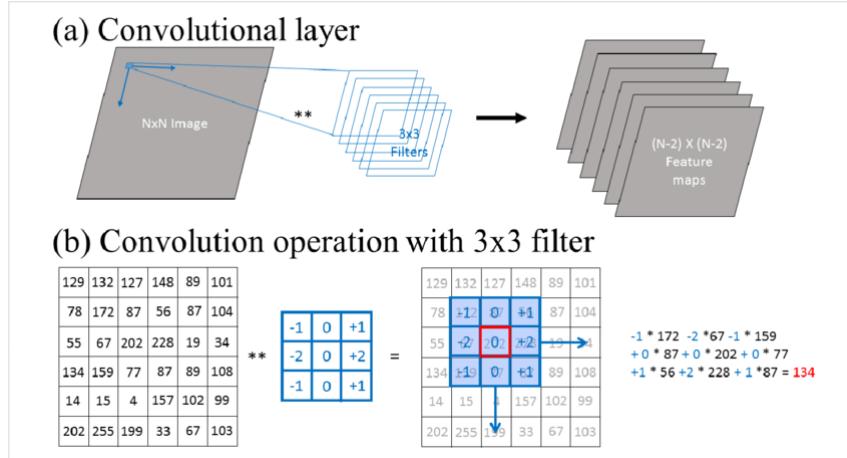
Una red neuronal convolucional (Indolia, Goswami, Mishra, y Asopa, 2018) es un tipo de red neuronal artificial con una estructura y tipos de capas muy concretos que surgen principalmente para tareas relacionadas con visión artificial, debido a su capacidad de aprender características abstractas. Este tipo de modelos tienen un número de parámetros mucho menor que las redes neuronales tradicionales lo que se traduce en una mayor capacidad de generalizar y de evitar el sobreentrenamiento. Este tipo de modelos constan de cuatro componentes principales: capa(s) convolucionales, capa(s) de *pooling*, función de activación y capa *fully-connected*, las cuales realizan las tareas respectivamente siguiendo la estructura que muestra la Figura 3.6.



Fuente: (Indolia y cols., 2018)

Figura 3.6: Estructura red neuronal convolucional.

- **Capas convolucionales:** Las capas convolucionales son el núcleo del funcionamiento de las redes neuronales convolucionales, y su funcionalidad se puede resumir en la creación de un mapa de características de la imagen original de forma que representa características invisibles a simple vista de la imagen. Este proceso se realiza aplicando un filtro o *kernel*, que no es más que una matriz que se utiliza como ventana deslizante, multiplicando a cada uno de los píxeles de la imagen y pudiendo sumar o no pequeño *bias*. También se aplica una función de activación como la usada en las redes neuronales estándar, como por ejemplo la sigmoide o la tangente hiperbólica.



Fuente: (Azuri y cols., 2021)

Figura 3.7: Funcionamiento capa convolucional

- **Capas de *pooling*:** La capa de pooling se centra en reducir la dimensionalidad de las características, debido a que una vez detectada, la localización exacta de la misma no tiene excesiva importancia. Este paso se aplica dividiendo el valor de entrada en un número de ventanas de un tamaño determinada, aplicando una operación concreta a todos los valores de la ventana, utilizando el resultado de aplicar dicha operación a cada ventana (como por ejemplo máximo o media). Por ejemplo, para una entrada de tamaño 4x4, el resultado final sería una matriz de tamaño 2x2. En la Figura 3.8 se muestra el funcionamiento y resultado de una operación de *max pooling* sobre una entrada de tamaño 4x4.

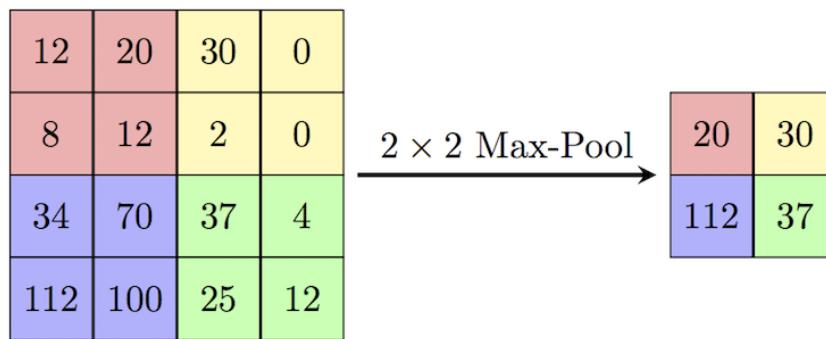


Figura 3.8: Funcionamiento MaxPooling.

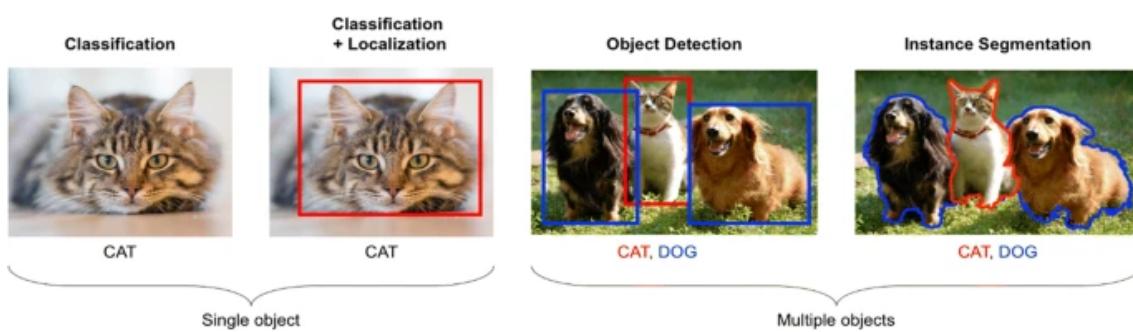
- **Capas *fully-connected* y función de activación:** Las capas fully connected son las que se encargan de extraer conocimiento de las características extraídas mediante las capas anteriores. En ellas se aplica el producto entre el dato de entrada, y el vector de peso de cada una de las neuronas, para posteriormente aplicar una función

de activación como por ejemplo la función sigmoide, la tangente hiperbólica, o, en el caso de tratarse de un problema de clasificación; la función *Softmax* en caso de ser la última capa de la red.

3.2.5. Modelos de visión artificial y detección de objetos

El entorno exterior del coche es muy importante ya que es algo que escapa al control del propio coche. Es por ello que son muy importantes la sensorización y cámaras del vehículo, ya que estas permiten llevar a cabo diferentes tipos de tareas. Tomando como base las imágenes recogidas por las cámaras se pueden llevar a cabo las siguientes tareas (Diwan, Anirudh, y Tembhere, 2023):

- **Clasificación de imágenes:** Es la tarea de clasificar una imagen o el objeto que aparece en una imagen dentro de un conjunto de categorías predefinidas. Este tipo de tareas pueden llevarse a cabo mediante técnicas de *machine learning* tradicional como SVM o árboles de decisión. Pero las variantes basadas en *Deep Learning*, concretamente aquellas basadas en redes convolucionales son las que obtienen mejores resultados.
- **Localización de objetos:** La localización de objetos es la tarea de determinar la posición de un objeto o múltiples objetos dentro de una imagen, gracias a una *bounding box*, que se trata de una caja rectangular que determina los bordes del objeto que se está localizando/clasificando.
- **Segmentación de imágenes:** Esta técnica es aún más específica que la localización de objetos, ya que se trata de dividir una imagen en varios segmentos, de manera que trata de detectar, todos los objetos clasificables, pero además se intenta distinguir de forma precisa los contornos de cada uno de ellos.



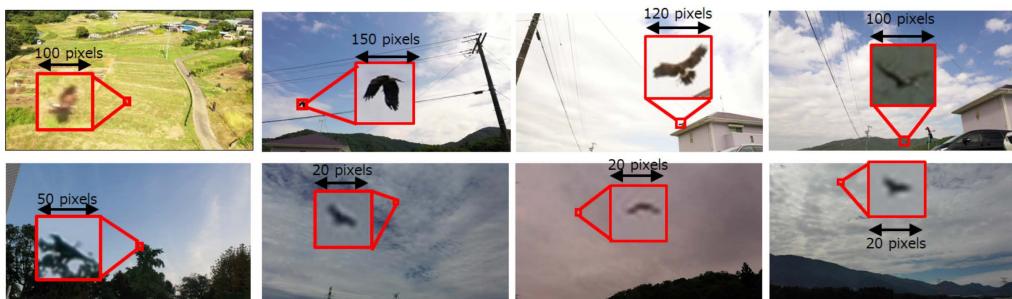
Fuente: (Diwan y cols., 2023)

Figura 3.9: Tipos de tareas de visión artificial.

El proceso de detección de objetos tiene muchas posibles aplicaciones, pero las más interesantes para el contexto del problema de la conducción autónoma es la detección de peatones, ciclistas y señales de tráfico; ya que por un lado, es necesario tener claro dónde están los elementos vulnerables del entorno exterior del vehículo, mientras que la detección e interpretación de las señales de tráfico es necesaria para que el vehículo respete en todo momento las normas de circulación de la vía.

La detección de objetos, a pesar de tener un gran espectro de diferentes aplicaciones y ser un proceso con las bases bien asentadas, sigue presentando una serie de desafíos:

- **Dependencia de la escala:** La mayoría de detectores de objetos se entrena para una entrada con una resolución concreta, lo que provoca que el rendimiento sea peor cuando se utilizan *inputs* con diferentes resoluciones, incluso si se aplica un reescalado.
- **Desbalanceo entre clases:** Este problema no es propio de este tipo de problemas, pero sí que es muy acusado, sobre todo en aquellos casos en la que la cantidad de objetos presente es pequeña.
- **Detección de objetos pequeños:** Si los objetos que se desean detectar/localizar son muy pequeños en relación al tamaño total de la imagen. Este es un problema que ocurre en la mayoría de algoritmos de detección de objetos. (Ver Figura 3.10 en sección)
- **Necesidad de grandes datasets y potencia de cómputo:** Los algoritmos de detección de objetos (sobre todo, aquellos basados en *deep learning*), son muy dependientes de los recursos computacionales disponibles, ya que pueden tardar mucho en entrenarse. Además, al necesitar un etiquetado manual de todos los objetos (tanto en clase como en localización), se convierte en una tarea que también tiene un alto grado de necesidad de recursos humanos.
- **Imprecisión en la localización de objetos:** Al tratarse de áreas rectangulares, en muchos casos, se introducen píxeles pertenecientes al fondo de la imagen, lo que puede provocar un rendimiento más pobre de los esperado a la hora de la predicción.

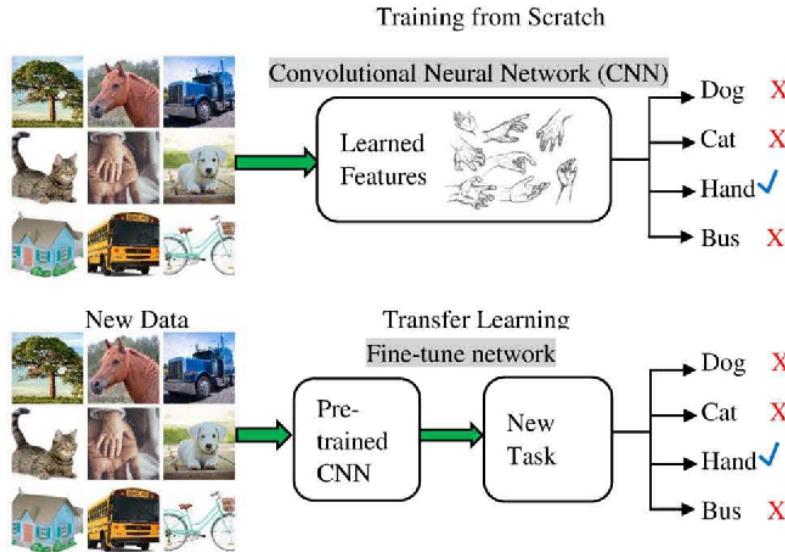


Fuente: (Fujii y cols., 2021)
Figura 3.10: Desafío detección de pájaros.

Transfer Learning y Data Augmentation

Las tareas de visión artificial, por lo general, son bastante complejas y muy dependientes de los datos de entrenamiento de los modelos. Es por ello que debido a que los datos pueden ser muy diversos, es complejo que este tipo de modelos generalicen correctamente, siendo que además este tipo de modelos pueden ser costosos de entrenar correctamente. Tanto es así, que puede resultar difícil encontrar una cantidad de datos suficiente, tanto en cantidad como calidad para obtener, partiendo desde cero en un modelo de detección de objetos. Por ello se hace uso de una técnica llamada *transfer learning*, que consiste en

partir de un modelo entrenado o parcialmente entrenado para una tarea o sobre un conjunto de datos concreto. Esto permite que al no partir de un conjunto de pesos aleatorios, el modelo tienda a converger mucho más rápido y por lo tanto obtener un rendimiento mejor en un número mucho menor de iteraciones. También sirve incluso en aquellos casos en los que se desea realizar una tarea diferente que para la que fue pensado el modelo original, como por ejemplo, una red convolucional que se dedica a clasificar entre coches y motos, se puede reutilizar para distinguir entre perros, gatos y pájaros únicamente reentrenando la parte clasificadora del modelo, como muestra la figura 3.11. Existen casos



Fuente: (Deep y Zheng, 2019)

Figura 3.11: Explicación Transfer Learning en redes convolucionales.

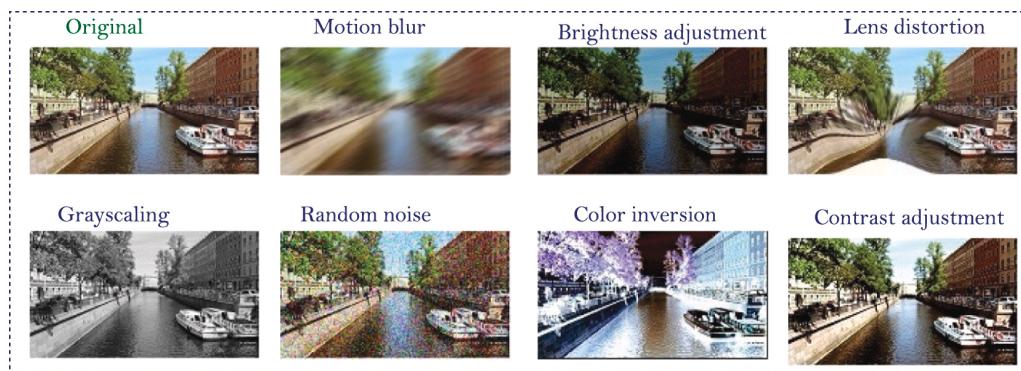
en los que incluso aplicando *transfer learning* no se consigue converger hacia una solución lo suficientemente buena, especialmente debido a la baja capacidad de generalizar de los modelos basados en imágenes. Por ello, existe otra técnica que se puede aplicar o no simultáneamente a *transfer learning* y se trata del *data augmentation*. Esta técnica, se basa en realizar modificaciones a datos ya existentes de forma que se pueden “crear” nuevos datos que pueden ser utilizados para entrenar el modelo, esta técnica es muy utilizada en los modelos de visión artificial debido a su incapacidad de generalizar fácilmente y debido a la gran cantidad de formas diferentes mediante la que aplicarla. Dependiendo de la operación aplicada para su obtención, las técnicas de *data augmentation* usadas sobre imágenes, se pueden clasificar de la siguiente forma (Mumuni y Mumuni, 2022):

- **Transformaciones geométricas:** Este tipo de transformaciones son las más básicas que se pueden aplicar a todo tipo de imágenes, y se basa en realizar cambios que alteran la estructura original de la imagen, pero sin alterar sus valores, como por ejemplo girar, recortar, modificar la perspectiva o reflejar la imagen. Existe un tipo especial de transformaciones geométricas más complejas que se utilizan principalmente en aplicaciones médicas que son mucho más complejas, denominadas **no-afines**.
- **Transformaciones fotométricas:** Son la contraparte de las transformaciones geométricas, ya que se basan en alterar el valor de los píxeles, pero sin alterar la posición de los mismos. Algunos ejemplos de este tipo de transformaciones, como por ejemplo la adición de *motion blur*, la inversión de color o el ajuste de brillo.



Fuente: (Engilberge y cols., 2022)

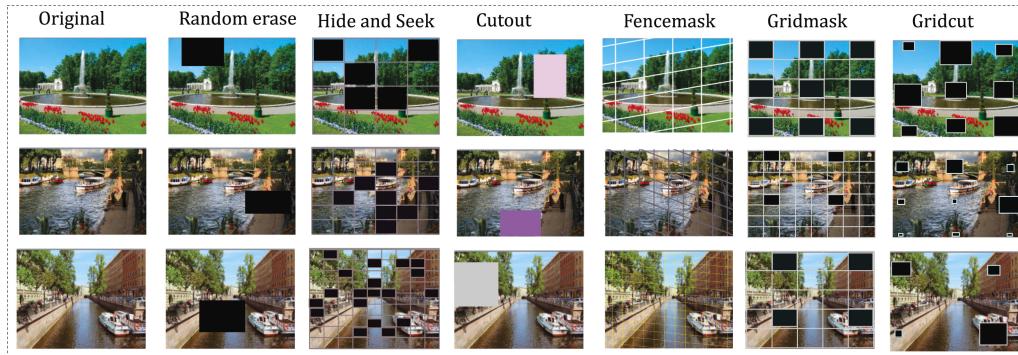
Figura 3.12: Diferentes ejemplos de *data augmentation* geométrico.



Fuente: (Mumuni y Mumuni, 2022)

Figura 3.13: Ejemplos *data augmentation* fotométrico.

- **Transformaciones a nivel de región:** Este tipo de transformaciones, en lugar de aplicar una transformación de forma uniforme en toda la imagen, se aplica en zonas o regiones y es un método que es muy útil para problemas de detección de objetos, cuando existe un número de objetos cercanos muy grande, provocando multitud de ROIs superpuestas. Hay diferentes formas de aplicar esta técnica: borrado de regiones, reemplazo de regiones y transformación aleatoria en diferentes regiones, incluso llegando a poder combinar diferentes regiones de diferentes imágenes para formar una nueva imagen basadas en parches del resto.



Fuente: (Mumuni y Mumuni, 2022)

Figura 3.14: Ejemplo de técnicas de borrado de regiones.



Fuente: (Mumuni y Mumuni, 2022)

Figura 3.15: Ejemplo de técnicas de reemplazo e intercambio de regiones.



Fuente: (Mumuni y Mumuni, 2022)

Figura 3.16: Ejemplo técnicas de combinación de regiones.

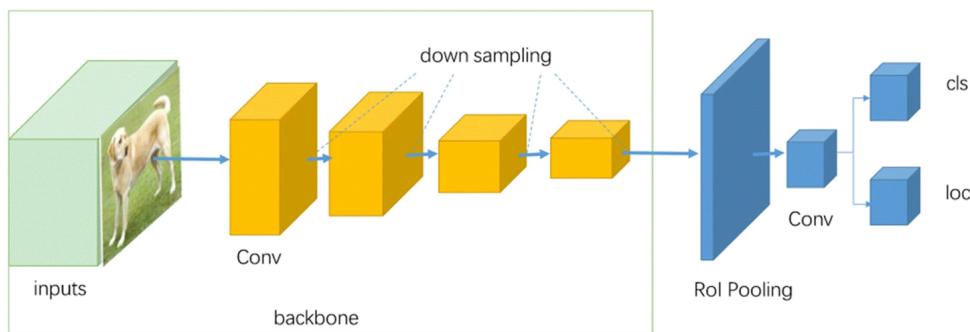
3.3– Análisis de tecnologías

El problema de la detección de objetos, es un tipo de problema que generalmente se plantea en dos etapas. La primera de ellas genera lo que denominan Regiones de Interés (RoI). Esta etapa se basa en la utilización de un tipo de red neuronal denominada *Region Proposal Network* (RPN), que son redes neuronales convolucionales (por lo general, completamente conectadas), cuyo propósito es simplemente generar todos los *bounding boxes*

que deben aparecer en la imagen, sin clasificarlos de ninguna manera, mientras que la segunda etapa está dedicada únicamente a la clasificación de cada una de las *bounding boxes* propuestas. Esta es la estructura que usan los modelos más populares de detección de objetos basados en redes neuronales convolucionales como pueden ser *R-CNN* (Girshick, Donahue, Darrell, y Malik, 2014), *Fast R-CNN* (Girshick, 2015) o *Faster R-CNN* (S. Ren, He, Girshick, y Sun, 2017).

Por otro lado también existen otros modelos de detección de objetos que se basan en una única etapa en el cual la localización se formula como un problema de regresión. Una de las mayores ventajas de este tipo de modelos, es la baja complejidad junto con una implementación sencilla por lo que es una de las opciones preferidas cuando se necesita aplicar este tipo de modelos en algún tipo de producto final. Algunos modelos que se basan en este tipo de arquitectura son *YOLO* (*You Only Look Once*) (Redmon, Divvala, Girshick, y Farhadi, 2016) o *SSD* (*Single Shot Detector*) (W. Liu y cols., 2016).

En las Figuras 3.17 y 3.18 se muestra de forma esquemática la estructura básica de las arquitecturas de cada una de los tipos de detectores, en las que se puede observar fácilmente las diferencias entre ambos tipos de paradigmas de detectores de objetos:



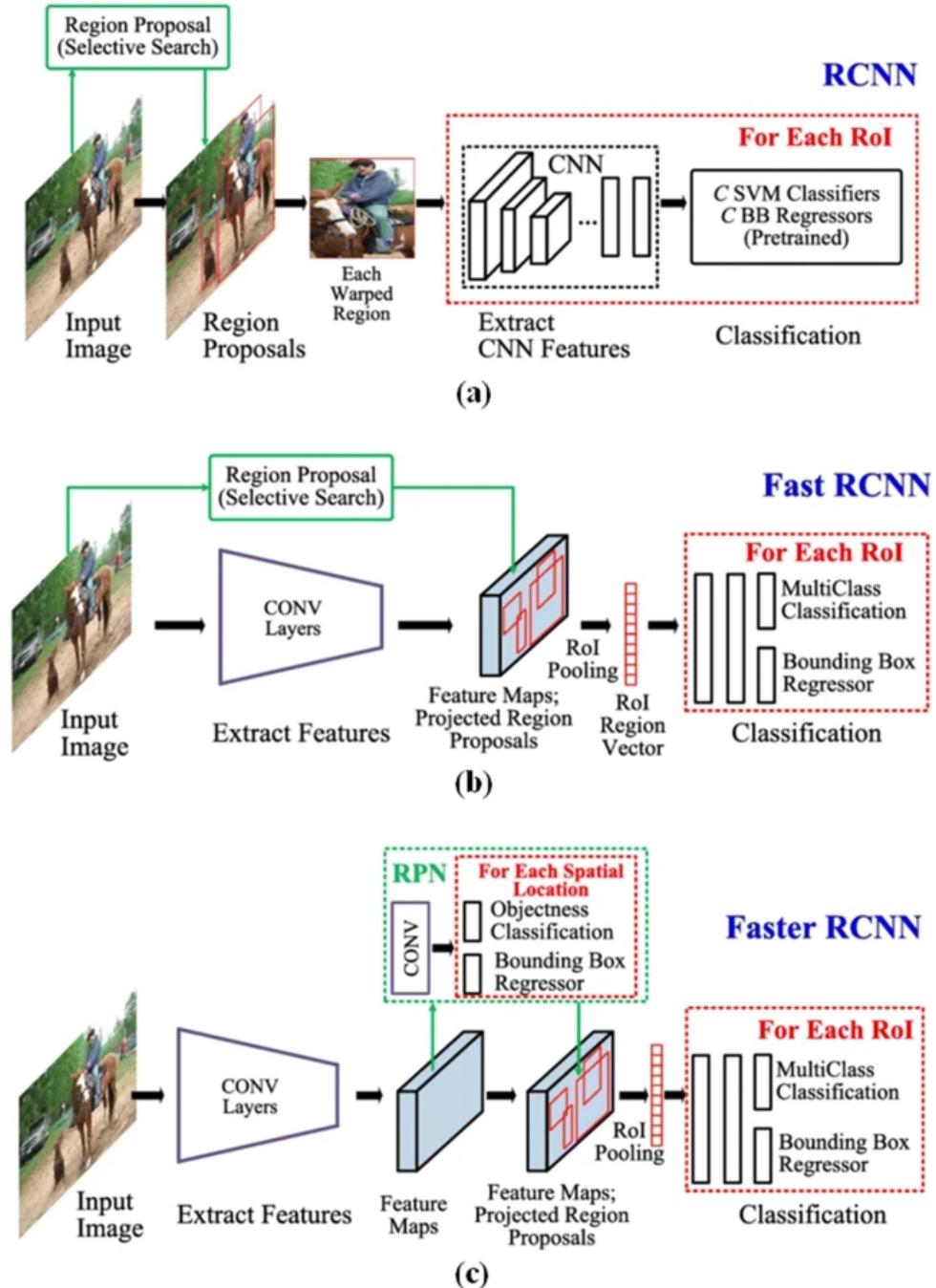
Fuente: (Diwan y cols., 2023)

Figura 3.17: Arquitectura de modelos de localización de objetos en una etapa.

Además, en la Figura 3.19, se puede ver que a partir de un único enfoque de los modelos tradicionales como ver tres ramificaciones, aunque este trabajo estará centrada en la segunda: en primer lugar existe la división relativa al uso de *anchors* o puntos de interés, y en segundo lugar dentro de aquellos basados en el uso de *anchors* aquellos en los que se usan una o dos etapas.

3.3.1. Modelos basados en dos etapas: R-CNN y variantes

Como se ha introducido anteriormente, los modelos de detección de objetos en dos etapas han sido los más utilizados para esta tarea durante mucho tiempo y para casos en los que no es necesaria la detección en tiempo real se sigue usando ampliamente en la actualidad. Antes de surgir este tipo de modelos, la mayoría de tareas de detección de objetos y clasificación de imágenes se basaban en SIFT (Scale-Invariant Feature Transform) (Lowe, 2004) y HOG (Histogram of oriented gradients) (Dalal y Triggs, 2005). Más tarde, con el auge del deep learning y las redes convolucionales, surge un nuevo enfoque para la detección de objetos que combina el uso de regiones con redes neuronales convolucionales, naciendo de esta forma **R-CNN** (Region Convolutional Neural Network) (Girshick y cols.,

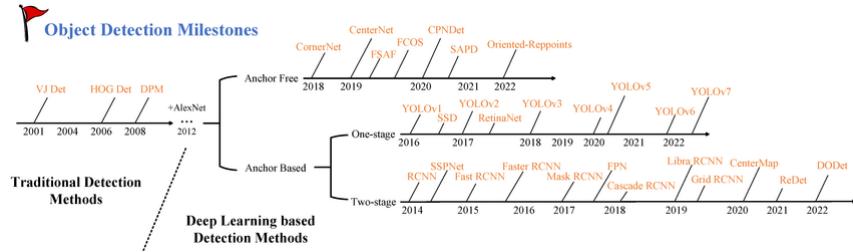


Fuente: (Diwan y cols., 2023)

Figura 3.18: Arquitectura de modelos de localización de objetos en dos etapas.

2014). Esta arquitectura se divide en tres módulos principalmente:

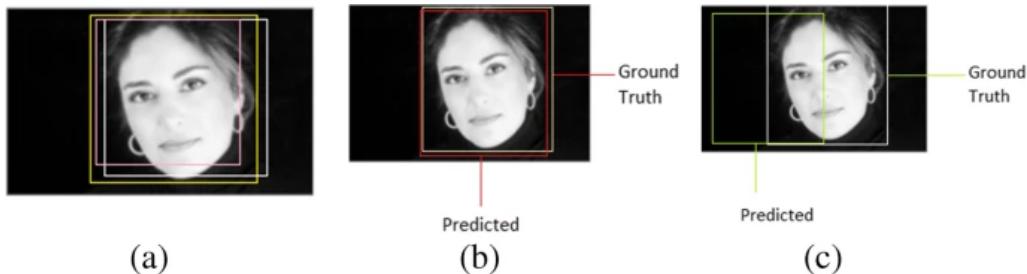
1. Propuesta de regiones que se generan mediante un algoritmo de búsqueda selectiva, aunque realmente, este módulo es completamente independiente ya que se puede aplicar cualquier algoritmo de propuesta de regiones.
2. Cada propuesta de región pasa por una red Convolucional, que originalmente estaba propuesta con cinco capas convolucionales, seguidas de dos capas *fully-connected*, generando un vector de características de tamaño 4096.



Fuente: (Wang y cols., 2023)

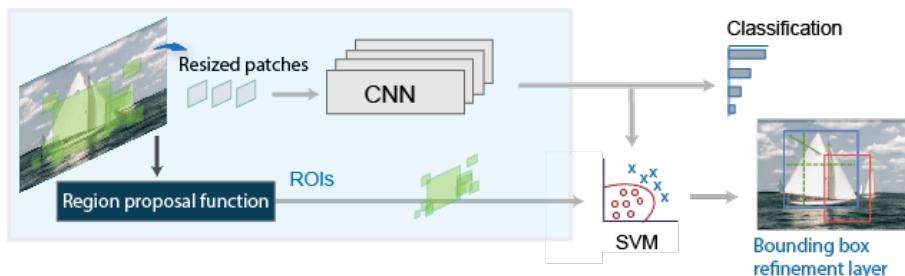
Figura 3.19: Evolución de los modelos de detección.

- Como último paso, el vector de características se usa como entrada para una serie de modelos lineales pre-entrenados para cada una de las clases a detectar, para posteriormente aplicar *Non-Maxima suppression* (Hosang, Benenson, y Schiele, 2017), que es un algoritmo cuya finalidad es eliminar todas las detecciones de objetos que pudieran estar duplicadas. Este algoritmo hace uso del concepto de Intersección sobre la Unión (IOU) (Rahman y Wang, 2016), la cual se define como la proporción del área superpuesta entre las *bounding boxes* propuestas y las verdaderas, el valor de esta función oscila entre 0 y 1, de forma que se pueden descartar aquellas detecciones superpuestas con otras obteniendo el mayor área posible de intersección entre las mismas.



Fuente: (Diwan y cols., 2023)

Figura 3.20: Ejemplo de Non-maxima suppression aplicada con IOU.



Fuente: (Shima y cols., 2023)

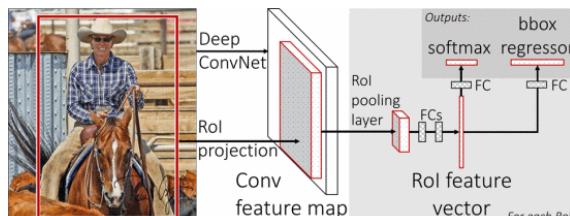
Figura 3.21: Esquemática de la arquitectura de R-CNN.

Este tipo de arquitectura ha evolucionado a lo largo de los años proponiendo diferentes actualizaciones a la arquitectura original, pero manteniendo una propuesta similar a la original de R-CNN. Una de las primeras actualizaciones propuestas y que obtiene importantes mejoras se denomina *Fast R-CNN*.

Fast R-CNN

Fast R-CNN (Girshick, 2015) es propuesto por el mismo creador del *R-CNN* original. Esta evolución aporta grandes mejoras tanto en la precisión en la detección de objetos como en la velocidad de ejecución del modelo. En este caso a diferencia de *R-CNN*, la imagen pasa por una red neuronal Convolucional para crear un vector de características, mientras que las ROI se calculan basándose en la imagen original (más tarde con la especificación de *Faster R-CNN* se demuestra la importancia de este detalle). A pesar de que, *R-CNN* tenía un gran rendimiento en detección de objetos, también mostraba varios aspectos negativos muy importantes para determinadas aplicaciones:

- El entrenamiento del modelo consistía en realizar *fine-tuning* de la red neuronal convolucional, además de entrenar los modelos de clasificación que usan los vectores de características generados (SVM en este caso).
- El entrenamiento es muy caro tanto temporalmente, como el espacio necesario para almacenar las imágenes y las ROI, ya que con redes neuronales muy profundas como por ejemplo *VGG-16*, el proceso tomaba 2 días y medio de entramiento en GPU para únicamente 5000 imágenes, además de necesitar cientos y cientos de gigabytes para almacenar toda la información.
- En tiempo de inferencia, las características se extraen de cada ROI, provocando que al usar *VGG-16* como red convolucional, la inferencia tardaba unos 47 segundos con procesamiento en GPU.



Fuente: (Girshick, 2015)
Figura 3.22: Arquitectura Fast-RCNN.

En la Figura 3.22, se puede ver como la red convolucional procesa la imagen completa y aplica un max pooling para crear una matriz de características. Entonces, para cada una de las ROI, se extrae un vector de características de longitud fija de la matriz ya generada a través de una capa de pooling. Finalmente, cada vector de características se utiliza como entrada para una serie de capas *fully connected* que termina con dos capas: una capa *softmax* que clasifica sobre el total de clases, más una clase *background* genérica, y otra capa que genera cuatro valores numéricos mostrando las coordenadas de la *bounding box* para una de las clases, teniendo esta la forma (r, c, h, w) , siendo r, c las coordenadas de esquina superior izquierda y (h, w) la altura y anchura del ROI respectivamente.

La red neuronal encargada de extraer la matriz de características de la imagen generalmente procede de un modelo preentrenado, al que se le deben aplicar tres transformaciones. En primer lugar, la última capa de max pooling de la misma se debe configurar con unos hiperparámetros $H \times W$ para que sean compatibles con la primera capa *fully connected*. En segundo lugar, la última capa *fully connected* y *softmax* se debe reemplazar con las

dos capas de salida mencionadas anteriormente, y por último, la red debe modificarse para tomar dos inputs, una lista de imágenes y una lista de ROIs provenientes de esas imágenes.

A continuación, es importante plantear un enfoque diferente al entrenamiento, ya que para que el modelo sea realmente mucho más rápido que R-CNN, el método de entrenamiento se plantea de manera que las ROIs pertenecientes a la misma imagen siempre pertenecen al mismo mini-batch. De forma que se crean batches de N imágenes y R/N ROIs de cada imagen, siendo que los mejores resultados se obtuvieron con 2 imágenes y 128 ROIs de cada una.

Para poder entrenar este modelo, también se debe plantear una función de pérdida específica ya que tenemos dos salidas, siendo una de ellas un formato muy específico, que se plantea como una función de pérdida multitarea para poder entrenar la clasificación y la regresión referente a la posición del *bounding box*.

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v), \quad (3.4)$$

Siendo que $L_{cls}(p, u) = -\log p_U$ es la función de pérdida para la clase real u . Mientras que L_{loc} , se define por la tupla de *bounding box* real para la clase u , $v = (v_x, v_y, v_w, v_h)$ y una tupla $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$ también para una clase u . Por otro lado $\lambda[u \geq 1]$, sirve para que el valor de esta parte del loss sea 0 en los casos en los que se detecte background, $u = 0$ por convención, mientras que en caso contrario valdrá 1.

$$L_{loc}(t^u, v) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i^u - v_i), \quad (3.5)$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0,5x^2 & \text{if } |x| < 1 \\ |x| - 0,5 & \text{otherwise,} \end{cases}$$

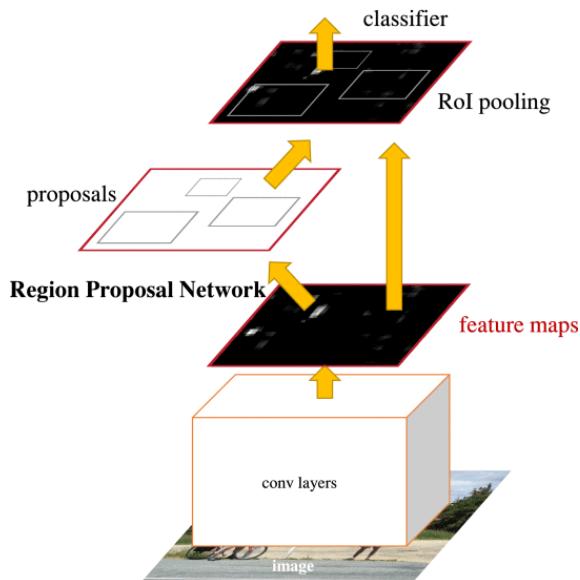
El hiperparámetro λ de la Ecuación 3.4 controla el equilibrio entre las dos pérdidas de tareas. Normalizamos los objetivos de regresión vi para que tengan media cero y varianza unitaria. En todos los experimentos se utiliza $\lambda = 1$.

Como propuesta para obtener una detección más rápida se aplica *SVD* truncado (Hansen, 1987), en la cual se aplica una matriz de dimensión uxv y denominada W . En el caso de querer comprimir la red, la capa *fully connected* que corresponde a W , se reemplaza con dos capas *fully connected* con colinearidad entre ellas, de manera que el método hace que la detección sea más rápida con un número de ROIs alto.

La conclusión final, es que la mayor mejora del modelo es la relacionada con el tiempo de cómputo y el tiempo de detección, el cual mejora aún más aplicando una *sparse proposal network*, en la cual se descartan muchos de los ROIs propuestos que no superen un valor de confianza establecido.

Faster R-CNN

La última iteración de este tipo de modelo se denomina Faster R-CNN (S. Ren y cols., 2017), ya que, a pesar de todos los avances propuestos en la subsección 3.3.1, y conseguir funcionar prácticamente en tiempo real, la propuesta de ROIs actuaba como cuello de botella a la hora de realizar detecciones. Por lo general, los algoritmos de propuesta de ROIs funcionaban prácticamente todos únicamente con procesamiento en CPU, a diferencia de las redes neuronales convolucionales que sí podían aprovecharse de su procesamiento en GPU. Por lo tanto, se propone un cambio algorítmico, consistente en aprovechar la red neuronal convolucional que extrae la matriz de características de la imagen, para también obtener la propuesta de ROIs, lo que provocaría que en tiempo de inferencia se compartan las convoluciones acelerando el mismo, llegando al punto de que sea casi descartable (10 ms por imagen). Es de esta manera que nacen las RPNs (*Region Proposal Networks*). Esta RPN, agrega nuevas capas convolucionales sobre la red convolucional original, de forma que puede extraer los límites de los objetos, como se muestra en la Figura 3.23.



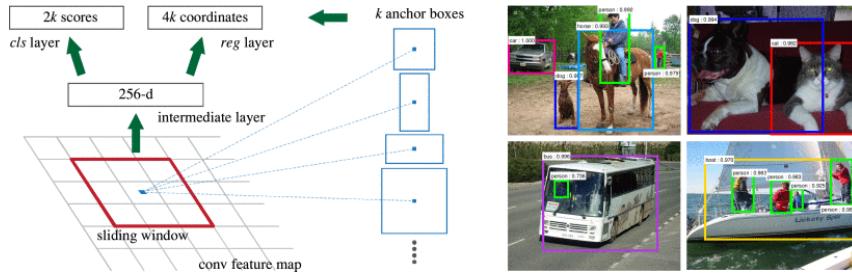
Fuente: (S. Ren y cols., 2017)

Figura 3.23: Arquitectura de Faster R-CNN.

Para generar las ROIs, se aplica una pequeña red sobre la matriz de características creada por la red convolucional, siendo que esta red toma una ventana deslizante $n \times n$ de la matriz de características, siendo que cada ventana se proyecta en un espacio de características con menor dimensionalidad (dependiendo de la red convolucional usada). Este vector de características se utiliza como entrada a dos capas *fully-connected*, una de ellas aplicando regresión y la otra clasificación. Este funcionamiento se puede observar detalladamente en la Figura 3.24.

En cada ventana, se pueden predecir múltiples ROIs, donde el máximo a detectar se denomina k . Por lo que la red neuronal tiene diferentes salidas: la parte regresora, tiene un total de $4k$ salidas que corresponden con las coordenadas de las k regiones propuestas, mientras que el clasificador produce un total de $2k$ que estiman la posibilidad de que la propuesta sea o no un objeto a detectar realmente.

Cada región propuesta se denomina *anchor*, y cada anchor está centrada en dicha



Fuente: (Brito y cols., 2018)
Figura 3.24: Arquitectura del RPN.

ventana. En este experimento se plantean tres escalas y ratios de aspecto diferentes ($k = 9$). Lo que quiere decir que para una matriz de características WXH , habrá un total de WHk anchors. Gracias a esta propuesta multiescalar, se puede usar el vector de características generado por las convoluciones, calculado en una única escala de imagen para luego ampliar a otras escalas.

En este caso la función objetivo a minimizar sería la siguiente:

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) \quad (3.6)$$

$$+ \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*). \quad (3.7)$$

Donde i es el índice del anchor en el minibatch y p_i es la probabilidad de que el anchor sea realmente un objeto. p_i^* tiene valor 0 cuando el anchor no es objeto y 1 en el caso contrario. Mientras que t_i es el vector que incluye las coordenadas reales de la bounding box asociada a un anchor evaluado como objeto real y L_{cls} la entropía cruzada entre las dos clases. Para la regresión, se utiliza como función de pérdida la misma utilizada en 3.3.1, y donde la inclusión de p_i^* implica que el término solo tendrá un valor diferente a 0 cuando se trate realmente de un objeto a detectar.

Los dos términos de la ecuación están normalizados por N_{cls} y N_{reg} respectivamente, y siendo que el término de la regresión está balanceado por un parámetro λ . En este caso N_{cls} se asigna al tamaño del mini-batch (256 en este caso). Mientras que N_{reg} se asigna al número de posibles localizaciones de anchor. Por defecto el valor de λ se establece en 10 aunque durante la experimentación se demostró que cambiar el valor del mismo apenas afecta al resultado final. Para las coordenadas de la regresión para proponer el bounding box se utiliza la siguiente ecuación:

$$t_x = (x - x_a)/w_a, \quad t_y = (y - y_a)/h_a, \quad (3.8)$$

$$t_w = \log(w/w_a), \quad t_h = \log(h/h_a), \quad (3.9)$$

$$t_x^* = (x^* - x_a)/w_a, \quad t_y^* = (y^* - y_a)/h_a, \quad (3.10)$$

$$t_w^* = \log(w^*/w_a), \quad t_h^* = \log(h^*/h_a), \quad (3.11)$$

Donde x, y, w, h denotan las coordenadas (x, y) del centro del *bounding box* junto con su anchura y altura (w, h) . Por otro lado x, x_a, x^* son los valores para la *bounding box* predicha, la del *anchor* y la real (se aplica la misma lógica para y, w, h).

Una vez definidos la forma de entrenar cada una de las redes convolucionales es el momento de entrenarlas, de manera que las redes no se entrenen de forma independiente y que se compartan las capas convolucionales. Por lo tanto se proponen diferentes maneras de llevar esto a cabo:

1. **Entrenamiento Conjunto Aproximado:** Ambas redes (RPN y Fast-RCNN) se unen en una sola red, de forma que las regiones propuestas por RPN se toman como propuestas pre-computadas cuando se entrena la parte correspondiente a Fast RCNN, mientras que ambos valores de pérdida, son combinados en aquellas capas convolucionales comunes. Un aspecto tener en cuenta relativo a esta metodología, es que se ignora la derivada aplicada a las coordenadas del *bounding box*, de ahí su denominación aproximada. Este enfoque, provoca una muy pequeña pérdida de rendimiento, pero reduce el tiempo de entrenamiento entre un 25-50 %.
2. **Entrenamiento Conjunto no-aproximado:** Este caso es muy similar al anterior, con la diferencia de que no se ignora la derivada, y por lo tanto se necesita una capa de *maxpooling* diferenciable, lo cual hace la solución mucho más compleja, y al no existir apenas diferencia con la solución aproximada se descartó completamente.
3. **Entrenamiento Alternado en cuatro pasos:** Este fue el enfoque que se aplicó en (S. Ren y cols., 2017), el cual como primer paso se entrena la red RPN, en el cual se crean mini-batches de *anchors* a partir de una única imagen, tratando de tener un ratio 1:1 entre positivos y negativos, y en el caso de existir menos *anchors* positivos que el tamaño de mini-batch definido, se aplica un padding con *anchors* negativos aunque se rompa el ratio 1:1. A continuación, se inicializan las nuevas capas utilizando una distribución aleatoria (Gaussiana es la recomendación del autor), mientras que las capas convolucionales comunes se deben inicializar con los pesos de un modelo entrenado para optimizar el resultado. Como segundo paso, se entrena la red de detección usando los ROI propuestos en el paso 1. En el tercer paso, se usa la red del detector para inicializar la RPN, pero congelando las capas convolucionales comunes. Finalmente, se aplica *fine-tuning* sobre las capas independientes de forma que las capas convolucionales son compartidas. Debido a que muchos de los ROIs propuestos se sobreponen, se aplica *non-maxima suppression*, basándose en la probabilidad de clase, de forma que se establece un umbral que en el caso de no superarlo para ninguna de las clases, se descarta automáticamente.

Este modelo de detección de objetos fue sido comparado con el modelo OverFeat (Sermanet y cols., 2014), el cual es un modelo de una única etapa el cual se propone con anterioridad a la existencia de YOLO. Esta comparación se aplicó sobre el dataset MS COCO (Lin y cols., 2015), y utilizando diferentes escalas y ratios de aspecto, siendo que los resultados arrojados por esta comparación le dan una ventaja de 5 puntos de mAP, como se puede observar en la Figura 3.25.

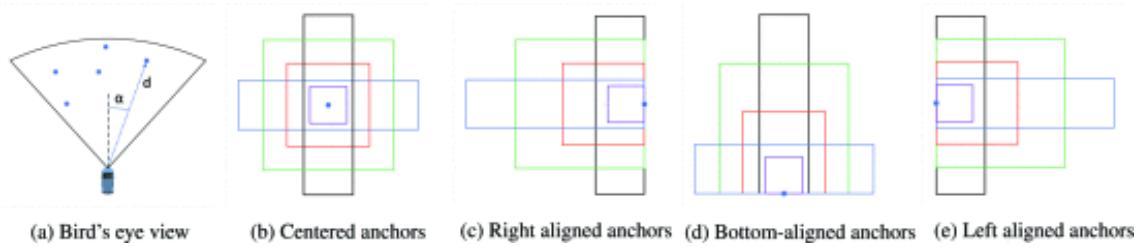
	proposals		detector	mAP (%)
Two-Stage	RPN + ZF, unshared	300	Fast R-CNN + ZF, 1 scale	58.7
One-Stage	dense, 3 scales, 3 aspect ratios	20000	Fast R-CNN + ZF, 1 scale	53.8
One-Stage	dense, 3 scales, 3 aspect ratios	20000	Fast R-CNN + ZF, 5 scales	53.9

Fuente: (S. Ren y cols., 2017)

Figura 3.25: Resultados de comparación OverFeat vs Faster R-CNN.

3.3.2. RRPN: Radar Region Proposal Network

Los RPN son un elemento propio de los modelos de detección en dos etapas, y son los que consumen gran parte de los recursos y tiempo de cómputo necesarios para llevar a cabo la detección de objetos, es por ello, que los modelos en dos etapas, suelen ser más precisos pero mucho más lentos que los modelos de una única etapa, esta es la razón de la siguiente propuesta (Nabati y Qi, 2019), en la cual se aplica el uso de la información proporcionada por el radar para proponer las RoI, utilizando un modelo denominado RRPN (*Radar Region Proposal Network*) que se basa en el uso de la información del radar para proponer la localización de las *anchor boxes*, que son aquellas localizaciones preselecciones para posteriormente ser seleccionadas como *bounding box*, a continuación se transforman y escalan basándose en la distancia del objeto al vehículo.



Fuente: (Nabati y Qi, 2019)

Figura 3.26: Imagen que representa los anchor boxes en RRPN.

Esta propuesta tiene una gran ventaja que es que se puede utilizar en cualquier modelo de dos etapas sustituyendo a la primera de ellas. La mejora se centra en el descarte de toda la información que no sea de utilidad para la toma de decisiones que se realizan durante la conducción autónoma, ya que generalmente, toda la información de la imagen es de igual importancia, mientras que durante la conducción autónoma, la atención se debe centrar en todos aquellos objetos que se encuentren en la carretera. Este objetivo se lleva a cabo utilizando un método de atención concentrándose en la partes importantes, para ello, este modelo se divide en tres etapas:

Transformación de perspectiva

En el primer paso, se mapean las detecciones del radar desde coordenadas basadas en el vehículo a coordenadas basadas en la vista de la cámara. En este caso la detecciones del radar se reportan en forma de distancia y ángulo respecto del sistema de coordenadas del vehículo. Con este mapeo lo que se consigue es que las coordenadas de la imagen y las coordenadas del radar se encuentren en el mismo sistema de referencia. Por lo tanto esta transformación de un punto tridimensional $P = [X; Y; Z; 1]$ y ese mismo punto en una

imagen bidimensional $p = [x; y; 1]$ se puede expresar de la siguiente manera, donde, en el caso de aplicaciones de conducción autónoma la matriz H se obtiene de los parámetros de calibración de la cámara.

$$p = HP, \quad H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \end{bmatrix} \quad (3.12)$$

Generación de anchors

Tras hacer el mapeado entre el radar y la imagen, se puede extraer la localización aproximada de cada objeto en la imagen. Cada uno de estas localizaciones se llama **Punto de interés** o **POI**. El siguiente paso es simplemente introducir una *bounding box* centrada en cada POI. Aunque este enfoque provoca dos problemas: el primero de ellos es que las detecciones del radar no siempre se mapean al centro de los objetos que aparecen en la imagen, y el segundo es que el radar no provee de información acerca del tamaño de los objetos detectados por usar un tamaño fijo de *bounding box* provocaría imprecisiones en muchos de los casos.

Como solución a los problemas planteados, se crean varias *bounding boxes* con diferentes tamaños y relaciones de aspecto, siendo 4 tamaños diferentes y tres relaciones de aspecto diferentes. Adicionalmente, para solucionar el problema de que cada POI no coincide con el centro real del objeto detectado, también se plantean diferentes *bounding boxes* pero en lugar de sólo cambiar el tamaño y relación de aspecto, también se cambia la posición respecto al POI, siendo que el centro de cada una de ellas no tiene por qué coincidir con el POI, pero sí que debe contenerlo. Un ejemplo de la aplicación de estas soluciones se puede ver en la Figura 3.26.

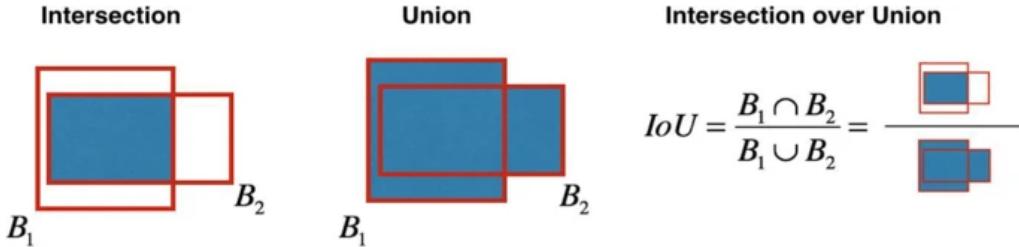
Compensación de distancia

Generalmente, el tamaño de los objetos tiene una relación inversa con la distancia a la cámara. Los objetos detectados mediante el radar, sí que contienen la información de la distancia por lo tanto se utilizará en este paso para escalar todos los *anchors* generados en el paso anterior. Se usa la Ecuación 3.13 para determinar el factor de escalado que se debe usar en cada uno de ellos.

$$S_i = \alpha \frac{1}{d_i} + \beta \quad (3.13)$$

Donde d_i es la distancia al i-ésimo objeto, y α y β son dos parámetros que se utilizan para ajustar el factor de escalado. Cuyos valores se obtienen maximizando la función de **Intersección Sobre la Unión**. Este proceso se representa a través de la Ecuación 3.14.

$$\operatorname{argmax}_{\alpha, \beta} \sum_{i=1}^N \sum_{j=1}^{M_i} \max_{1 < k < A_i} IOU_{jk}^i(\alpha, \beta) \quad (3.14)$$



Fuente: (Diwan y cols., 2023)

Figura 3.27: Intersection over Union de forma esquematizada.

En esta ecuación, N representa el número de imágenes, M_i es el número de *bounding boxes* reales que existen en la imagen i , A_i es el número de *anchors* generados sobre la imagen i , e IOU_{jk}^i es la *IoU* entre la **bounding box** real j -ésima en la imagen i y el *anchor* k -ésimo propuesto en dicha imagen.

Para comprobar el rendimiento de *RRPN* se usó el dataset *NuScenes* (Caesar y cols., 2020) el cual se explica más detalladamente en la Subsección 4.3.1. Se realizaron dos experimentos con dos muestras diferentes del dataset, uno de ellos tomando únicamente los radares y cámara frontales, con unas 23000 imágenes en total (NS-F). El segundo subset contiene el primer subset además de datos provenientes de la cámara trasera y los dos radares traseros, los cuales forman un dataset de 45000 imágenes en total (NS-FB). Por lo general las cámaras y radares frontales tienen una mayor precisión por lo que hace que el subset NS-F sea más preciso con aquellos objetos lejanos al vehículo. Finalmente se realiza una separación 0.85 - 0.15 en entrenamiento y test respectivamente.

Debido a que *RRPN* se limita a la propuesta de ROIs, es necesario un modelo en dos fases para completar el proceso de detección de objetos. En este caso, se ha seleccionado dos modelos basado en Fast R-CNN (detallado en Sección 3.3.1). ResNet-101 (R1-1) (He, Zhang, Ren, y Sun, 2015) y ResNeXt-101 (X-101) (Xie, Girshick, Dollár, Tu, y He, 2017). En la fase de entrenamiento, los modelos usados parten de un pre-entrenamiento sobre el dataset COCO y se realiza fine-tuning (ver Subsección 3.2.5), además de comparar los resultados utilizando el mismo dataset sobre el algoritmo *Selective Search* (SS). En la Figura 3.28 se puede observar el comportamiento de SS y *RRPN* respecto de las *bounding boxes reales*, en la primera fila se encuentran las propuestas reales, en la segunda las propuestas de SS y en la última las *bounding boxes* propuestas por *RRPN*.

Para medir el rendimiento, se han utilizado las mismas métricas que se utilizan para el dataset MSCOCO (Lin y cols., 2015), siendo estos el **AP** o *Average precision* y el **AR** o *Average Recall*.

Tras realizar la experimentación, los resultados de *RRPN* eran superiores o muy similares respecto de SS en la mayoría de clases probadas que fueron (Coche, Camión, Persona, Motocicleta, Bicicleta y Bus), aunque con una gran ventaja respecto de SS, que consiste en



Fuente: (Nabati y Qi, 2019)

Figura 3.28: Comparación entre las bounding boxes reales, SS y RRPN.

la rapidez, ya que mientras que SS tomaba alrededor de 2-7 segundos por imagen; RRPN era capaz de generar alrededor de 70-90 imágenes por segundo lo que se traduce en una eficiencia de mas de 100 veces superior, teniendo además una precisión igual o superior; demostrando también un mejor comportamiento en zonas de la imagen complejas como aquellas con mucho contraste o superposición entre diferentes objetos.

3.3.3. Modelos de detección en una etapa

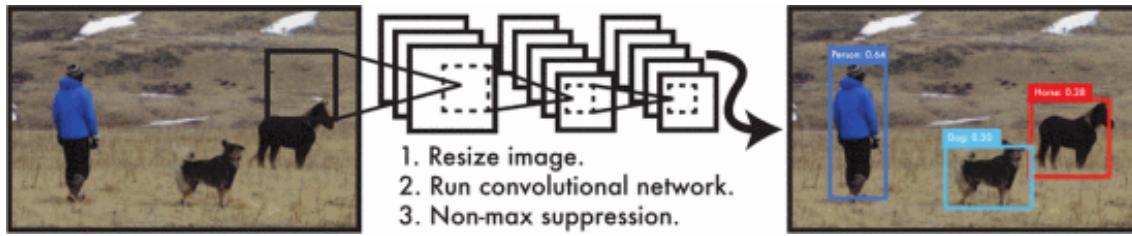
A diferencia de los modelos de detección en dos etapas, los métodos basados en una única etapa se basan en modelos de regresión en lugar de clasificación, y solo se utiliza un modelo detector que por lo general es también una red neuronal convolucional, a través de la cual se obtienen tanto las posiciones como las categorías de cada uno de los objetos presentes en la imagen. Alguna de las arquitecturas más importantes basadas en esta metodología son YOLO (Redmon y cols., 2016), SSD (W. Liu y cols., 2016) y DSSD (Fu, Liu, Ranga, Tyagi, y Berg, 2017).

YOLO (You Only Look Once)

El algoritmo YOLO (Redmon y cols., 2016) es el pionero del paradigma de detectores de objetos en una sola etapa, y surge como respuesta a una de las principales problemáticas que presentan los modelos en dos etapas, y se trata de la velocidad de procesamiento. Cuando surge YOLO, el algoritmo en dos etapas contemporáneo es R-CNN el cual necesitaba de un algoritmo extremadamente costoso en cuanto a tiempo y potencia de cómputo. Para ello, reformula el problema de la detección de objetos en un problema de regresión simple, tomando como base los pasos expuestos en la Figura 3.29.

Las principales ventajas de YOLO respecto de las arquitecturas en dos pasos serían:

1. YOLO es extremadamente rápido, siendo que es capaz de procesar vídeo en tiempo real con solo 25 milisegundos de latencia y pudiendo analizar más de 150 fps



Fuente: (Redmon y cols., 2016)

Figura 3.29: Funcionamiento base de YOLO.

(imágenes por segundo) en su versión *fast*.

2. YOLO realiza las detecciones tratando la imagen de forma global, y no generando regiones, arreglando uno de los principales problemas de Fast R-CNN (ver Subsección 3.3.1).
3. YOLO tiene una mayor capacidad de generalizar representaciones de objetos, por lo que es fácilmente generalizable en caso de ser usado como base para reconocer nuevos objetos y es menos sensible a cambios inesperados en alguno de los objetos detectables por el mismo.

Como se puede observar, esta arquitectura tiene muchas ventajas sobre el resto de detectores en dos pasos, sin embargo, sí que existe una desventaja principal, y es la precisión, ya que le resulta muy difícil detectar determinados objetos, especialmente si son de tamaño muy reducido. Además, también presenta dificultad en objetos pequeños que aparecen en grupo (una bandada de pájaros por ejemplo) o en casos en los cuales el ratio de aspecto de los objetos es muy diferente del que se ha usado en los datos de entrenamiento.

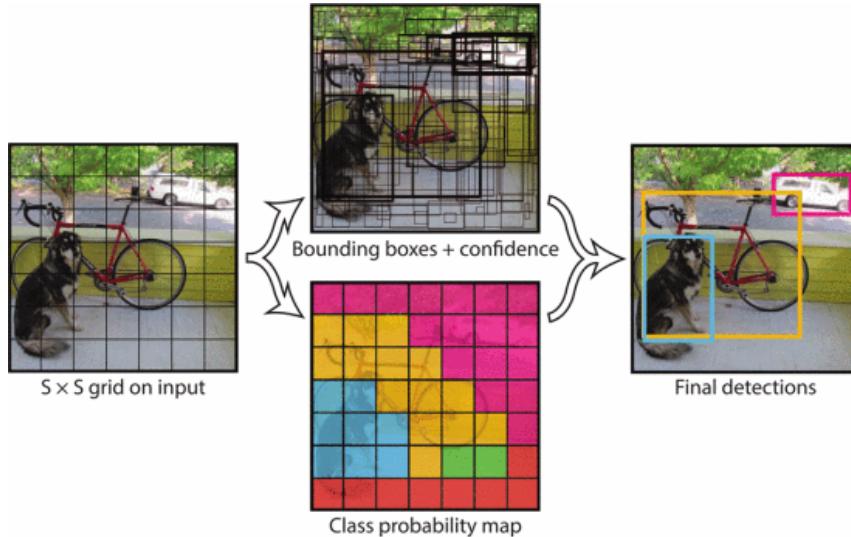
YOLO divide la imagen de entrada en una matriz de $S \times S$, siendo que si un objeto coincide con una celda de la misma, ésta es la responsable de la detección del objeto. Usando como base cada celda, se calcula un número determinado de *bounding boxes*, junto con una confianza para la misma. Siendo que esta confianza se define con la expresión $P(O) * IOU_{pred}^{truth}$, y por tanto ésta debe valer 0 si no existe ningún objeto presente en la misma.

Cada *bounding box* consta de 5 valores a predecir ($x, y, w, h, conf$). Siendo que (x, y) son las coordenadas del centro de la celda, (w, h) son la altura y la anchura y $conf$ la confianza que se representa con la expresión anterior, y depende del IOU (ver Subsección 3.3.2 entre la *bounding box* predicha y cualquiera real).

Cada celda predice la probabilidad condicional de que el objeto sea de una determinada clase, $P(Class_i|Object)$, y el valor de la misma depende del número de celda que contienen un objeto, de forma que se predice un conjunto de posibilidades que tiene tantos valores como número de clases posible. Luego, en tiempo de inferencia, se multiplica la probabilidad condicional de clase con la confianza del *bounding box* siendo la Ecuación 3.15.

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \quad (3.15)$$

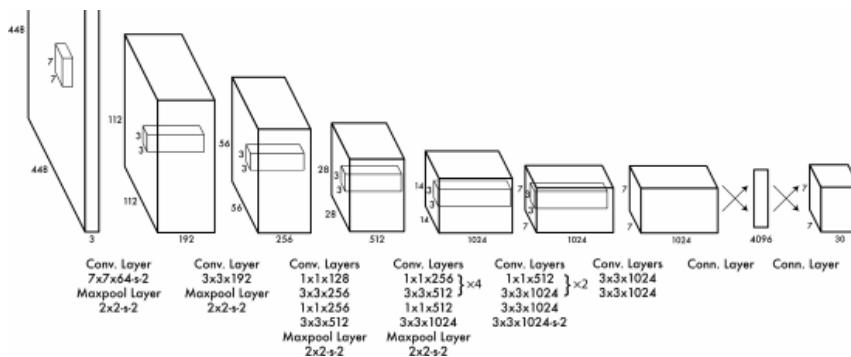
De esta manera, se obtiene la confianza por clase para cada *bounding box* y por tanto los objetos detectados en la imagen, junto a la clase a la que pertenecen.



Fuente: (Redmon y cols., 2016)

Figura 3.30: Proceso de funcionamiento de YOLO.

Para implementar este planteamiento, se utiliza una red neuronal convolucional, en la cual las capas convolucionales extraen las características de la imagen y las capas *fully-connected* extraen las probabilidades y coordenadas. La red neuronal original, estaba planteada usando como base GoogLeNet (Szegedy y cols., 2014), de forma que constaba de 24 capas convolucionales, seguido de 2 capas *fully-connected*. Usando capas de reducción 1x1 seguido de capas convolucionales 3x3, como se puede ver en la Figura 3.31.



Fuente: (Redmon y cols., 2016)

Figura 3.31: Red convolucional original de YOLO.

Como funciones de activación, se usa una función lineal para la capa final, mientras que el resto de capas usa una función *LeakyReLU*, que se formula mediante la siguiente Ecuación 3.16:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0,1x, & \text{otherwise} \end{cases} \quad (3.16)$$

La función a optimizar será la suma de cuadrados de los errores, debido a su facilidad para optimizar aunque no se alinee completamente con la meta de maximizar la precisión media. Un problema de esto, es que pondera de igual manera el error de clasificación y el de localización, además, esto provoca la pérdida de muchos gradientes debido a la falta de objetos en muchas de las celdas, y provocando que las celdas positivas y por consecuencia con gradiente positivo tengan excesivo peso en la optimización y provocando una rápida divergencia durante el entrenamiento. Para solucionar, estos problemas, se aplican dos parámetros de normalización λ_{coord} y λ_{noobj} , dando como función de pérdida la Ecuación 3.17:

$$\begin{aligned} & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \quad (3.17)$$

en la cual $\mathbb{1}_i^{obj}$ significa si el objeto aparece en la celda i y $\mathbb{1}_{ij}^{obj}$ significa que el *bounding-box* j -ésimo en la celda i es el que se hace cargo de dicha predicción.

Un punto a destacar, es que la función de pérdida sólo penaliza el error de clasificación si un objeto está presente, siendo que la probabilidad de clase se computa de forma diferentes, además solo penaliza la coordenada errónea de la *bounding box* si es la seleccionada para dicha celda (*bounding box* con mayor IOU para dicha celda).

A pesar de que no es necesario gracias a su diseño, algunos objetos muy cercanos o muy grandes pueden provocar la aparición de varias *bounding boxes* superpuestas, por lo que no es completamente necesario, pero si recomendable, aplicar *Non-maxima suppression* para refinar aún más el rendimiento total del modelo.

También se realizó una comparativa comparando, qué modelos de detección eran capaces de realizar detecciones sobre vídeo en tiempo real (considerando que tiempo real, es 30 imágenes por segundo o más). Siendo que los resultados arrojados, mostraron que solo

DPM (Deformable Parts Model) (Felzenszwalb, Girshick, McAllester, y Ramanan, 2010) y YOLO pudieron superar el umbral del tiempo real, pero siendo YOLO el único en superar 50 de precisión media, llegando hasta 63 en su mejor resultado y 52 en su versión *Fast*, obteniendo 45 y 155 imágenes por segundo respectivamente.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
<hr/>			
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Fuente: (Redmon y cols., 2016)

Figura 3.32: Comparación de modelos de detección en tiempo real.

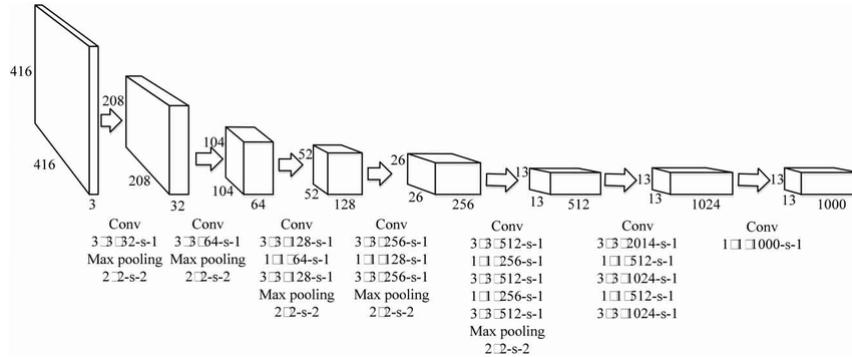
YOLO es uno de los detectores que ha pasado por un mayor número de iteraciones aplicando diferentes mejoras, siendo YOLOv2 y YOLOv3, las dos primeras, que son propuestas por el mismo autor del modelo YOLO original.

YOLOv2 y YOLOv3

YOLOv2 (Redmon y Farhadi, 2017) aplica varios cambios sobre la versión original de YOLO que mejoran tanto el rendimiento como la velocidad de procesamiento. Las mejoras se puede resumir como:

1. **Normalización en batch:** Se aplica *batch normalization* (Ioffe y Szegedy, 2015), como forma de regularización de manera que se pueden eliminar las capas el Dropout sin que el modelo tenga sobreajuste.
2. **Aumento de resolución:** Al cambiar la red convolucional utilizada, también se ha aumentado la resolución soportada de 224 a 448 para todo el proceso tanto de entrenamiento como de detección.
3. **Combinar capas convolucionales con anchor boxes:** Se elimina una capa de *pooling* para no reducir la dimensionalidad en exceso, además se utiliza un tamaño impar, de forma que siempre exista una celda central única.
4. **Clusters Dimensionales:** Una de las desventajas del YOLO original era que el tamaño del *anchor box* son predefinidos. La red neuronal puede parecer a ajustarlos de forma correcta, pero si se ajustan automáticamente, por lo tanto, se aplica un clustering *K-Means* con varios valores de *k*, de manera que los centroides de cada clúster dan lugar a *boxes* de diferentes ratios de aspecto, y aparece menor número de ellos.
5. **Nuevo modelo convolucional (Darknet-19):** Se aplica un modelo diferente llamado Darknet-19, que es bastante similar a la familia de modelos VGG. Este modelo

utiliza filtros 3x3, así como capas *max pooling* que duplican el número de características y convoluciones 1x1, además de capas *global average pooling* para hacer las predicciones finales.



Fuente: (Han y cols., 2021)

Figura 3.33: Estructura de Darknet-19.

YOLOv3 (Redmon y Farhadi, 2018), por otro lado es la última mejora que plantea el autor original de YOLO y se basa en mejorar las debilidades que presentaba YOLOv2. Una de las grandes novedades que presenta respecto a su antecesor es la *objectness score*, para cada una de las *bounding boxes* que se calcula mediante una **regresión logística**.

Como solución a otro de los problemas de YOLOv2, que eran las *bounding boxes* superpuestas, se realiza un cambio en la función a aplicar para la clasificación, ya que se sustituye la función Softmax por varios clasificadores logísticos independientes para cada una de las clases, de forma que una detección de objeto puede encajar en varias clases y no solo a una. (Por ejemplo, un hombre podría clasificarse como persona y como hombre al mismo tiempo).

Además de todo lo ya mencionado, también añade la característica Otro de los problemas que presentaba YOLO originalmente es la detección de objetos pequeños, y YOLOv3 plantea una solución, que se trata de incluir interconexiones entre las capas convolucionales para poder extraer información de forma más detallada. Esta novedad viene dada por el nuevo *backbone* usado que se trata de Darknet-53 que incluye 53 capas convolucionales, todo ello manteniendo una precisión muy buena y un tiempo de inferencia muy bueno respecto de la competencia, siendo de esta forma de las mejores opciones en cuanto a detectores para aplicaciones en tiempo real.

YOLOv4 y sucesores

La versión de YOLOv4 (Bochkovskiy, Wang, y Liao, 2020) es la primera de las mejoras (YOLOv3 será la última) que no está propuesta por el autor original de YOLO, sino que las mejoras son planteadas por la propia comunidad que surge a raíz del desarrollo de este tipo de detectores de objetos. En YOLOv4 se aplican 3 principales mejoras, además de un cambio de arquitectura en cuanto a la red convolucional:

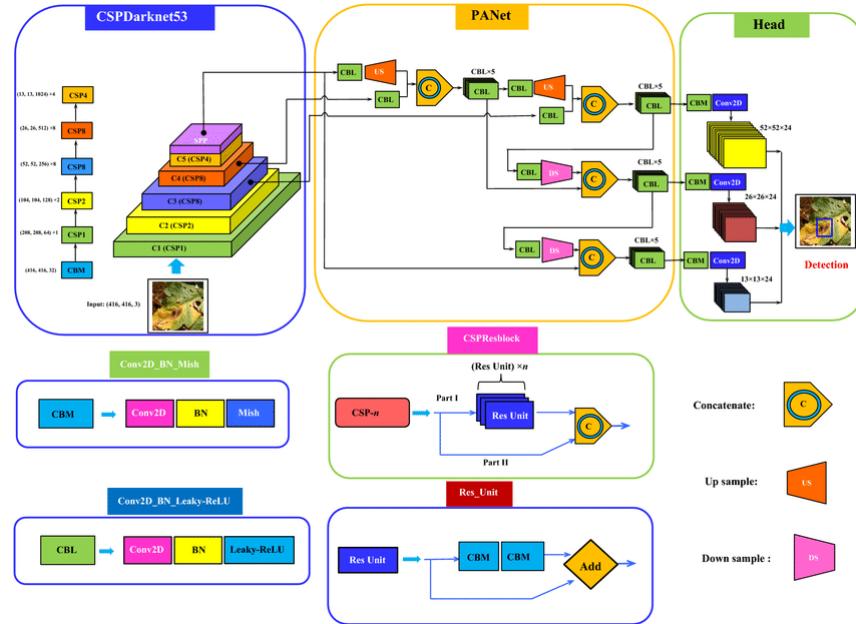
Type	Filters	Size	Output
1x	Convolutional	32	256×256
	Convolutional	64	128×128
	Convolutional	32	1×1
	Convolutional	64	3×3
2x	Residual		128×128
	Convolutional	128	64×64
	Convolutional	64	1×1
	Convolutional	128	3×3
8x	Residual		64×64
	Convolutional	256	32×32
	Convolutional	128	1×1
	Convolutional	256	3×3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

Fuente: (Redmon y Farhadi, 2018)

Figura 3.34: Estructura de Darknet-53.

- **Bag of freebies:** En este caso, se trata de una técnica que consiste en cambiar la técnica de entrenamiento, pero manteniendo la inferencia, en este caso se usaron tres: Data Augmentation, aplicando ciertas técnicas de modificación de imágenes, corregir la distribución semántica del dataset en caso de desbalanceo y el cambio de la función de pérdida.
- **Bag of specials:** Son métodos que solo empeoran ligeramente el tiempo de inferencia, pero produciendo un aumento muy significativo del rendimiento.
- **Self-adversarial Training (SAT):** En este caso, se aplica una técnica de manipulación de los datos de entrada que engañan al modelo para no detectar la presencia en un objeto que sí existe en la realidad, permitiendo al modelo obtener una mayor resiliencia ante posibles problemas en los datos de entrada en el mundo real

A partir de este punto hay propuestas interesantes como YOLOv5 (Jocher, 2020) y YOLOv8 (Jocher y cols., 2023) desarrollados por Ultralytics, pero la mayoría de los cambios se centran en variar la arquitectura de la red convolucional y realizar cambios en las funciones de pérdida para obtener el mejor balance posible entre precisión y tiempo de inferencia, porque el objetivo de YOLO sea cual sea la versión es su uso en aplicaciones en tiempo real.



Fuente: (Roy y cols., 2022)
Figura 3.35: Estructura de YOLOv4.

Tabla comparativa, Análisis crítico de las ventajas e inconvenientes de cada opción de cara al objetivo del TFM

CAPÍTULO 4

Desarrollo de la solución

4.1– Introducción

Para realizar una evaluación de la complejidad de la detección de objetos en el contexto de la conducción autónoma, y tras el estudio del estado del arte y las diferentes técnicas de detección de objetos que se pueden aplicar. Se ha diseñado un experimento para comprobar cuán bueno puede ser el rendimiento en los modelos detección de detección y cómo de usables en dicho contexto, ya que una de las principales necesidades que tienen los vehículos autónomos es la inmediatez en la obtención de datos, ya que las órdenes para el manejo del mismo también deben ser inmediatas, siendo incluso positivo un ligero compromiso de precisión por rapidez de detección.

4.2– Justificación

Como se ha especificado en la sección anterior (Sección 3.3), existen infinidad de modelos de detección de objetos siendo éstos de una o de dos etapas. Los modelos de detección en dos etapas sobresalen por su magnífico rendimiento en cuanto a precisión en la localización tanto en la clase de objeto a localizar, pero tienen estructuras mucho más complejas, mientras que los modelos en una etapa comprometen ligeramente la magnífica precisión en la localización que presentan los objetos en dos etapas, para obtener una velocidad de procesamiento extraordinaria, además de una complejidad mucho menor en la implementación, ya que el proceso de localización se realiza en conjunto con la clasificación del objeto. Adicionalmente, este tipo de modelos también presentan una gran dificultad a la hora de generalizar y realizar inferencia, debido a la gran diferencia que puede existir entre los casos del mundo real o debido a que la cantidad de datos de entrenamiento no es suficiente, esto también se puede solucionar mediante *data augmentation*. Por último aspecto a destacar, es que debido a la gran variedad de modelos entrenados que existe, se comprobará la utilidad de partir de un modelo ya preentrenado de forma que sólo habría

que recalcular pesos de determinadas capas durante la fase de entrenamiento.

4.3– Arquitectura/Diseño/Aspectos relevantes de la implementación

Este experimento se ha diseñado pensando en evaluar diferentes tipos de modelos, en combinación con diferentes tipos de datos y partiendo o no de modelos ya preentrenados.

4.3.1. Datasets utilizados

En este caso se ha hecho uso de una submuestra de dos datasets diferentes: *Waymo Open Dataset*(LLC, 2019) y *NuScenes*(Caesar y cols., 2020).

Waymo Open Dataset

Waymo Open Dataset, es un conjunto de datos recolectados por la empresa *Waymo*, la cual es la división de vehículos autónomos de Google, y presta servicio de taxis autónomos en varias ciudades de Estados Unidos como San Francisco. El conjunto de datos está obtenido a través de estos mismos vehículos en una gran variedad de condiciones y situaciones, y se ha utilizado para multitud de estudios científicos como por ejemplo (Sun y cols., 2020), el cual fue en parte creado por los autores de dicho estudio. Este conjunto de datos se compone de más de 2000 segmentos de 20 segundos cada uno, los cuales están grabados a una frecuencia de 10Hz, lo que significa que cada segmento consta de 200 frames aproximadamente, esto hace que el dataset completo conste de aproximadamente unas 400000 imágenes en diversas zonas geográficas y condiciones ambientales. Estos segmentos cuentan con datos basados en los siguientes sensores y características (*Waymo Open Dataset*, 2019):

- 1 LiDAR de rango medio
- 4 LiDAR de rango corto
- 5 cámaras, formando un ángulo de captura hacia delante de 180°
- Datos LiDAR e imágenes tomadas por las cámaras sincronizados en tiempo y espacio.
- Proyecciones LiDAR - Cámara
- Calibraciones de los sensores.

A su vez, para poder utilizar estos datos para tareas de detección, también dispone de anotaciones sobre las imágenes capturadas:

- Etiquetas para cuatro clases de objetos diferentes: Vehículos, Peatones, Ciclistas y Señales (Esta última no está disponible en las anotaciones de cámara, sino a través de la API de mapas).
- Más de 1200 segmentos de datos lidar etiquetados en alta calidad.
- Más de 12.6 millones de *bounding boxes* en tres dimensiones para datos LiDAR.
- Etiquetado de alta calidad en más de 1000 segmentos para los datos de cámara.
- Más de 11.8 millones de *bounding boxes* etiquetadas en dos dimensiones para datos de cámara.

Como se puede observar, la cantidad de datos presente en el dataset es inmensa, por lo que para poder abordar el problema con los recursos disponibles se ha extraído una submuestra que consta de 20 segmentos, los cuales constan de 19830 imágenes de las cuales 2975 son para validación y el 16855 para entrenamiento. En estas imágenes existe un total de 141780 detecciones de objetos de entrenamiento, junto con 29560 detecciones de objetos de validación, haciendo un total de 171340 detecciones, cada una de estas detecciones consta de las coordenadas del centro de la *bounding box* junto con su altura y anchura. En estas anotaciones están incluidos vehículos, peatones y ciclistas de la siguiente manera:

- **Vehículos:** 106144 en entrenamiento y 20289 en validación, haciendo un total de 126433.
- **Peatones:** 35242 en entrenamiento y 9017 en validación, haciendo un total de 44259.
- **Ciclistas:** 394 en entrenamiento y 254 en validación, haciendo un total de 648.

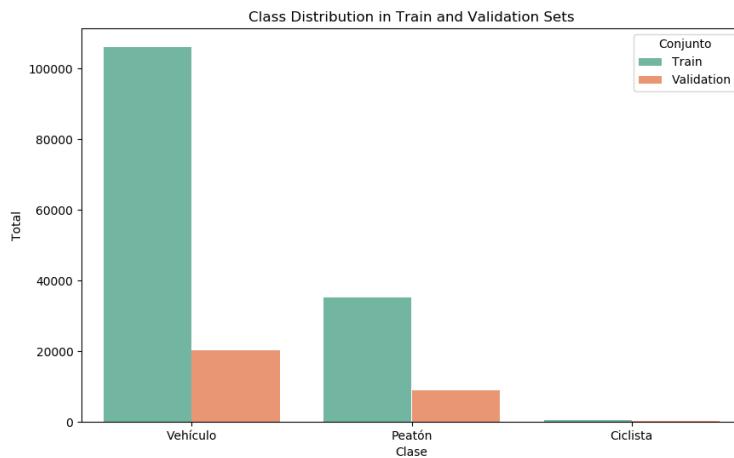


Figura 4.1: Distribución clases subset Waymo.

Como se puede observar existe un claro desbalanceo en el dataset, la cual deja la clase ciclista con una muestra mucho menor que el resto, lo cual puede afectar la bondad de las decisiones de la misma.

NuScenes

Por otro lado, se ha utilizado el dataset NuScenes (Caesar y cols., 2020), el cual es uno de los conjuntos de datos más utilizados en relación con la conducción autónoma. Este es un dataset públicamente accesible, que incluye datos de radar, cámaras, LiDAR y GPS; aunque en este caso se utilizarán los datos procedentes de cámaras. En su versión completa, consta de 1000 escenas, de las cuales 850 están etiquetadas y 150 no lo están, de forma que solo pueden ser utilizados como test. Cada escena dura 20 segundos y está etiquetada a 2Hz, de forma cada escena consta de 100 imágenes. Una peculiaridad de este dataset, es que los puntos sólo están etiquetados en *bounding boxes* en tres dimensiones por lo que es necesario realizar la proyección a 2D, siendo que los datos están recogidos siguiente equipamiento a nivel de sensores:

- Lidar de 32 haces de luz, el cual es capaz de generar entornos 3D con una gran resolución.
- 6 cámaras que cubren el entorno del vehículo en 360º.
- 6 radares que cubren el entorno del vehículo en 360º.

Para este experimento se ha hecho uso del subset mini de NuScenes, el cual se compone de 10 escenas las cuales al estar etiquetadas con una frecuencia de 2 Hz, significa que cada escena consta de 40 imágenes, siendo que el total de imágenes del conjunto de datos usado es de 334 imágenes de entrenamiento y 70 imágenes de validación, creando un dataset total de 404 imágenes, cuyas anotaciones hacen un total de 1411, divididas entre 1151 de entrenamiento y 260 de validación, y las cuales se componen de vehículos, peatones y ciclistas distribuidos de la siguiente manera:

- **Vehículos:** 551 en entrenamiento y 124 en validación, con un total de 675.
- **Peatones:** 590 en entrenamiento y 133 en validación con un total de 723.
- **Ciclistas:** 10 en entrenamiento y 3 en validación con un total de 13.

Como se puede observar existe un claro desbalanceo en el dataset, la cual deja la clase ciclista con una muestra mucho menor que el resto, lo cual puede afectar la bondad de las decisiones de la misma.

4.3.2. Modelos utilizados

Uno de los objetivos este Trabajo de Fin de Máster es comprobar los dos paradigmas diferentes que existen en los modelos de detección de objetos, los modelos de una etapa y los modelos de dos etapas. En este caso se han utilizado dos modelos ya detallados en el apartado de tecnologías utilizadas pero con algunos matices. Para el caso de los modelos en dos etapas, se ha hecho uso del modelo **Faster R-CNN**, el cual se ha detallado anteriormente en la Sección 3.3.1, y el detector **YOLO**, el cual se ha detallado anteriormente en la Sección 3.3.3.

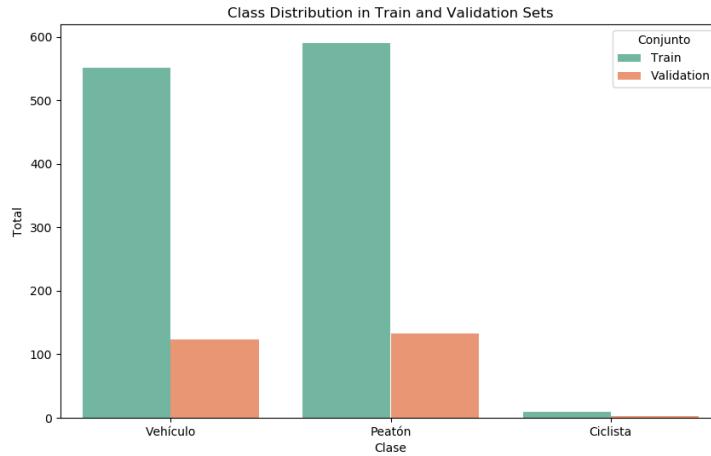
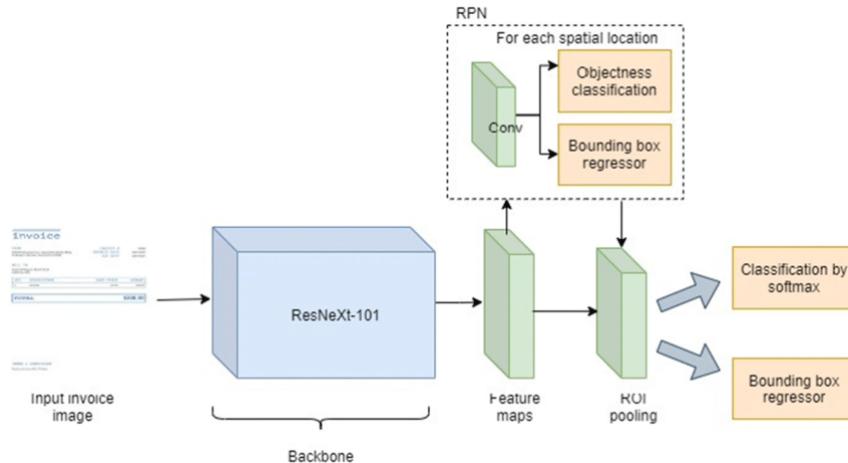


Figura 4.2: Distribución de clases del dataset NuScenes mini.

Faster R-CNN y MobileNetv3

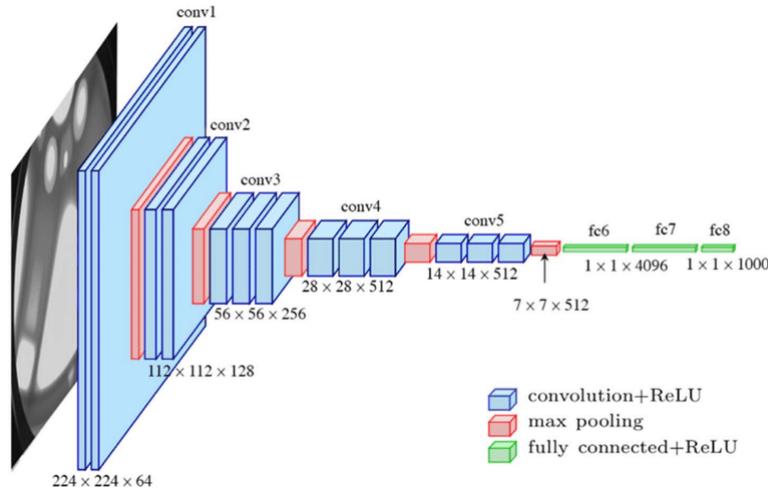
Esta iteración del modelo **R-CNN** (Girshick y cols., 2014) intenta dar solución a uno de los mayores problemas que presente este paradigma de detectores, que es el tiempo de computación. Esto lo hacía mediante un Red Neuronal independiente denominada **Region Proposal Network**, haciendo un modelo más complejo que por un lado extrae las regiones de interés de la imagen, y por otro aplica la clasificación por cada una de ellas. Debido a esta complejidad la velocidad tanto de entrenamiento como inferencia del mismo, es un algoritmo muy pesado que puede no ser adecuado para tareas en tiempo real como puede ser la conducción autónoma.



Fuente: (Chazhoor y Sarobin, 2022)

Figura 4.3: Ejemplo Faster R-CNN.

Originalmente, este detector contaba con un backbone convolucional basado en la arquitectura *VGG* (*Visual Geometry Group Network*), la cual fue introducida en (S. Liu y Deng, 2015) y consta de la siguiente arquitectura (ver Figura 4.4). En la cual se puede observar que existen 5 bloques convolucionales, aplicando un filtro mayor cada vez, con función de activación ReLU, seguidos de capas de *max pooling* al final de cada bloque, para finalmente aplicar tres capas *fully-connected* también con función de activación ReLU.



Fuente: (S. Liu y Deng, 2015)

Figura 4.4: Arquitectura VGG.

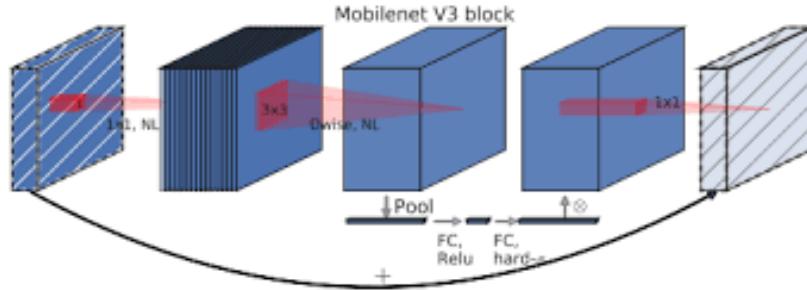
Esta arquitectura resultó seguir siendo no demasiado eficiente por lo que con el paso del tiempo se han usado nuevos backbones convolucionales como por ejemplo el que se ha utilizado en este experimento como es **MobileNetV3** (Howard y cols., 2017). En este caso se trata de una red convolucional completamente pensada para la detección de objetos en tiempo real en dispositivos livianos como por ejemplo smartphones, y surgió en el año 2017 en su primera versión (Howard y cols., 2017). Esta arquitectura también ha pasado por varias revisiones a lo largo de los años, siendo **MobileNetv3** la más reciente. Esta última iteración, **MobileNetV3**(Howard y cols., 2019), aplica varias mejoras a la versión original como es un proceso de diseño semi-automatizado denominado **NAS** (*Neural Architecture Search*, unas funciones de activación mejoradas como por ejemplo **Hard-sigmoid** (Ver Ecuación 4.1) o Hard-swish (ver Ecuación 4.2 la cual no es más que una modificación la función swish original que usaba la función sigmoide como producto de la propia variable, las cuales conservan la forma básica de la función Sigmoid, pero evitando el uso de la función exponencial, la cual es muy lenta en entornos computacionales.

$$f(x) = \max \left(0, \min \left(1, \frac{x+1}{2} \right) \right) \quad (4.1)$$

$$\text{HardSwish}(x) = x \cdot \frac{\text{ReLU6}(x+3)}{6} \quad (4.2)$$

$$\text{ReLU6}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x < 6 \\ 6 & \text{if } x \geq 6 \end{cases} \quad (4.3)$$

También se introduce como novedad, la introducción de bloques *Squeeze-and-excitation*, los cuales no son más que bloques compuestos por una capa de *GlobalAveragePooling*, seguido de dos capas fully connected, una de ellas con activación *H-Sigmoid* y otra de ellas con *H-swish*. Y por último se complementa con convoluciones algo más pequeñas que las originales (5x5 vs 3x3).



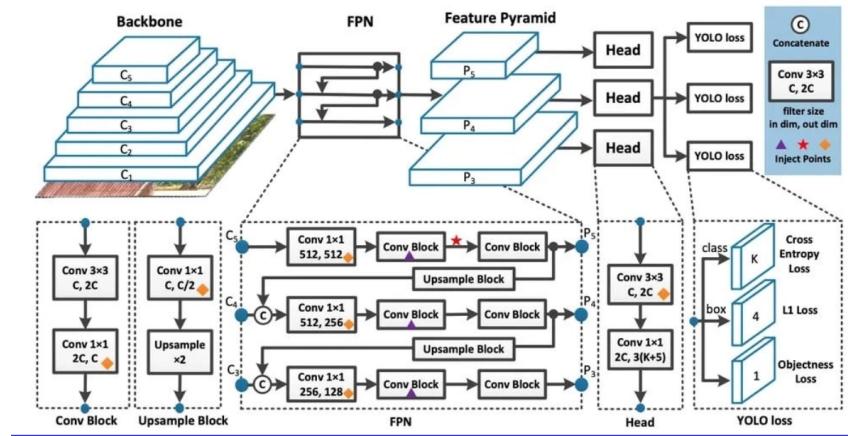
Fuente: (Al Musalhi y cols., 2024)
Figura 4.5: Archiecture of MobilenetV3.

YOLO (YOLOv8)

En este caso, como representante de los modelos basados en una etapa se ha seleccionado **YOLO** (*You Only Look Once*). Este detector es uno de los más usados y estudiados en la literatura y tiene un abanico de variantes y evoluciones. El modelo **YOLO** 3.3.3 original fue evolucionado por el mismo autor hasta su versión número 3, a partir de la cual las diferentes evoluciones y modificaciones han sido realizadas por la comunidad que heredó el desarrollo del mismo tras el retiro del autor. En este caso se ha seleccionado la versión 8, la cual es una de las versiones que más repercusión ha tenido, debido también al renombre de los creadores y su gran interfaz y facilidad de uso, se trata de la empresa Ultralytics (Jocher y cols., 2023), los cuales han creado tanto una API como una librería de Python en torno a este modelo en diferentes versiones. La versión 8, como principales mejoras incluye un nuevo modelo convolucional denominado **CSPDarknet53**, el cual sigue constando de 53 capas convolucionales con activación Leaky ReLU al igual que Darknet-53 (versión mejorada de (Han y cols., 2021)), pero incluye una conexión denominada *Cross-Stage Partial*, la cual provoca que la matriz de características se separe en determinados momentos, de forma que partes de la misma pasarán por varias capas convolucionales mientras que otras, pueden saltar varias de ellas. Además otra de las grandes novedades que presenta **YOLOv8**, es que la detección utiliza una versión modificada de la cabeza de **YOLO**, la cual es la parte del modelo que reporta la puntuación de objeto, la probabilidad de clase y las coordenadas de la *bounding box*, permite reportar estos datos en diferentes escalas gracias a la introducción de *Path Aggregation Network* (PANet). Por último, se ha incluido una nueva función de pérdida basada en la Intersección sobre la Unión 3.3.2, la cual ayuda a mejorar las predicciones sobre objetos superpuestos. En la Figura 4.6 se puede observar la estructura completa de este modelo.

4.4– Proceso de experimentación

La experimentación se basa en aplicar los dos modelos sobre los dos datasets explicados en la Sección 4.3.2, tanto en sus versiones preentrenadas como entrenando desde 0 y evaluar cómo de bueno es el rendimiento y cuál es el tiempo de inferencia para cada imagen, para así determinar si los modelos son viables o no para su aplicación en conducción autónoma. Las versiones preentrenadas de los modelos están preentrenadas sobre el dataset Imagenet (Deng y cols., 2009), el cual es un dataset de imágenes público que cuenta con más 1281167



Fuente: <https://yolov8.org/what-is-yolov8/>

Figura 4.6: Estructura YOLOv8.

imágenes de entrenamiento, 50000 de validación y más de 100000 de test, con objetos etiquetados en 1000 clases diferentes.

Antes de realizar el entrenamiento de los modelos, ha sido necesario realizar un preprocesamiento de los datos para poder extraer las anotaciones con las coordenadas de las *bounding boxes* y las imágenes. Cada modelo necesita de un formato diferente de datos para poder ser entrenado, pero para realizar un trabajo más eficiente y no generar dos datasets de entrada en formato completamente diferente, se hace un único formateo del dataset al formato YOLO, el cual consiste en las imágenes en crudo por un lado, y las anotaciones por otro, existiendo un fichero de anotaciones por cada imagen, siendo que cada fichero almacena una *bounding box* en cada línea con el formato:

class_id Coord X, Coord Y Anchura Altura. Siendo que las coordenadas X, Y corresponden al centro de la *bounding box*. Por otro lado, el formato necesario para FasterRCNN, en lugar de usar las coordenadas del centro, utiliza las coordenadas de las esquinas por lo que necesita las coordenadas **XMin, YMin, XMax, YMax** sin normalizar, pero esta conversión se utiliza mediante un DataLoader (<https://pytorch.org/docs/stable/data.html>) de Pytorch que lo hace al momento de cargar las imágenes junto con sus anotaciones para su posterior uso ya sea en entrenamiento o validación. Para el dataset NuScenes es necesario realizar la proyección de los *bounding boxes* a dos dimensiones, ya que las coordenadas vienen dadas en tres dimensiones. Cada experimento se ha llevado a cabo con la siguiente configuración, usando un optimizador mediante gradiente descendente (SGD) en ambos:

- **Experimentos con YOLOv8:** Se aplica un redimensionado a las imágenes a 480 y se usan batches de 32 imágenes cada uno, además el número de epochs de entrenamiento será de 100 para NuScenes y 15 para Waymo. El resto de valores será el que se usa por defecto, como 0.937 para el *momentum* y 0.01 para el *learning rate*.
- **Experimentos con Faster R-CNN:** Se reescalan las imágenes a 800 y se usan batches de 4 imágenes cada uno (limitado por VRAM), el número de epochs de entrenamiento será de 50 para NuScenes y 6 para Waymo Open Dataset. En este caso, se usará un *learning rate* de 0.01 y un *momentum* de 0.95.

Para poder medir el rendimiento obtenido en cada uno de los experimentos y comprobar la evolución del mismo a través de los *epochs* de entrenamiento se hará uso de las siguientes métricas:

- **Precision:** Esta métrica mide cuántas de las *bounding boxes* son correctas. Representa la proporción de ellas que ha sido correctamente predicha, sobre el total de datos, la cual se representa mediante la Ecuación 4.4:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (4.4)$$

Donde TP representa la cantidad de ellas que ha sido correctamente predicha y FP es la cantidad que han sido predichas de forma errónea. Esta medida se aplica por clase y se obtiene la media de todas las clases.

- **Recall:** El *recall* calcula cuántos de los objetos de la imagen fueron detectados correctamente por el modelo, sobre el total de objetos que aparecen en la imagen, la cual se representa mediante la Ecuación 4.5:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (4.5)$$

Donde TP representa la cantidad de ellas que ha sido correctamente predicha y FN es la cantidad de objetos que no han sido detectados correctamente. Esta medida se aplica por clase y se obtiene la media de todas las clases.

- **mAP50:** Es la principal medida a la hora de evaluar modelos de detección de objetos, la cual se basa en la media para todas las clases del *Average Precision*. El *Average Precision* mide el compromiso entre *Recall* y *Precision*. Se calcula obteniendo el Recall y la Precision usando un umbral de IoU (ver Subsección 3.3.2) de 0.5. Esto se representa mediante la Ecuación 4.6

$$AP = \sum_{n=1}^N \text{Precision}(n) \cdot (\text{Recall}(n) - \text{Recall}(R_{n-1})) \quad (4.6)$$

Donde n significa el umbral de IoU para el que se desea extraer la métrica, en el caso de AP50, sería $n = 0.5$ y $n - 1 = 0$.

- **mAP5095:** Aplica exactamente lo mismo que para el mAP50 pero en este caso se aplica la media para el AP50 entre todos los umbrales entre 50-95 en intervalos de 0.05. Esta métrica es algo más significativa que el mAP50 debido a que utiliza diferentes umbrales, a diferencia del AP50 que se rige únicamente por uno.

CAPÍTULO 5

Resultados

En este apartado se mostrarán los resultados obtenidos mediante la experimentación especificada en el apartado anterior, en ellos se muestra la evolución de los resultados a lo largo del entrenamiento como al finalizar el mismo, así como mostrar algunos ejemplos de imágenes que se ha utilizado para detección final, tras el entrenamiento de los modelos.

5.1– Resultados de YOLOv8

Esta sección mostrará los resultados obtenidos por el modelo *YOLOv8*, en ambos datasets y para las versiones con *backbone* preentrenado y partiendo desde cero.

5.1.1. Resultados de YOLOv8 para el dataset NuScenes

En este apartado se muestran los resultados obtenidos por el modelo YOLOv8 sobre el dataset NuScenes. Como se puede observar en la evolución sobre las épocas de

	vehículo	peatón	ciclista	media
precision	0.766	0.641	0.914	0.774
recall	0.806	0.631	0.667	0.701
AP50	0.846	0.703	0.670	0.740
AP5095	0.593	0.419	0.242	0.418

Cuadro 5.1: Tabla de métricas por clase para YOLOv8 sobre el dataset NuScenes,
Preentrenado

entrenamiento se puede observar en las Figuras 5.1 y 5.2, que representan el proceso de

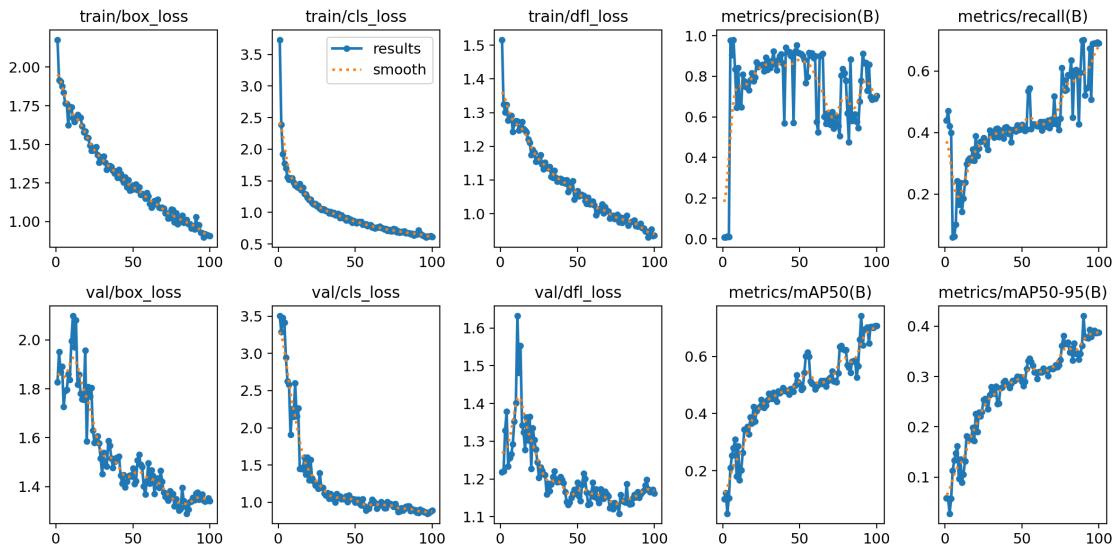


Figura 5.1: Evolución métricas durante el proceso de entrenamiento de NuScenes en la versión Preentrenada sobre ImageNet

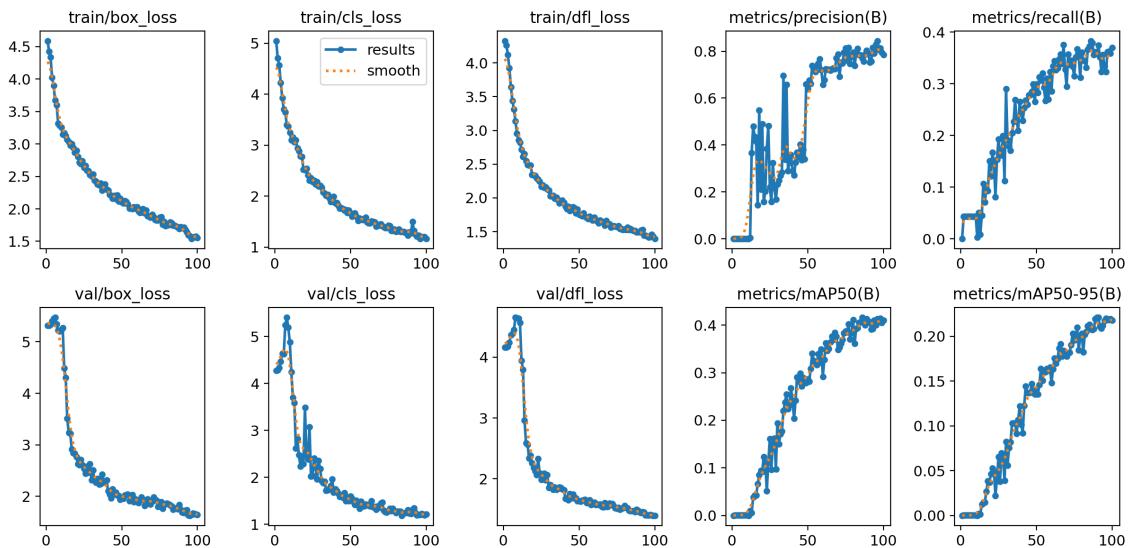


Figura 5.2: Evolución métricas durante el proceso de entrenamiento de NuScenes en la versión desde cero

	vehículo	peatón	ciclista	media
precision	0.641	0.755	1.00	0.7987
recall	0.630	0.441	0	0.357
AP50	0.671	0.560	0	0.4103
AP5095	0.400	0.263	0	0.221

Cuadro 5.2: Tabla de métricas por clase para YOLOv8 sobre el dataset NuScenes, desde cero

entrenamiento sobre NuScenes, todas las métricas siguen mejorando hasta la última época

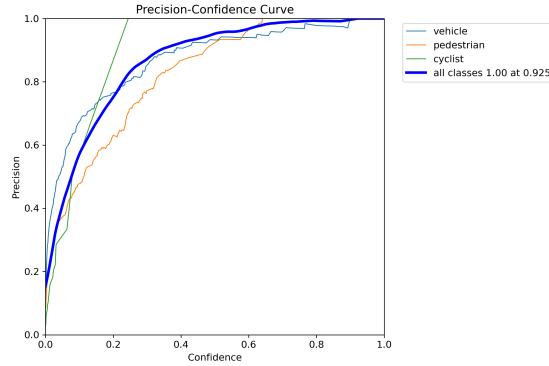


Figura 5.3: Curva Precision-Confidence para YOLOv8 Preentrenado usando NuScenes

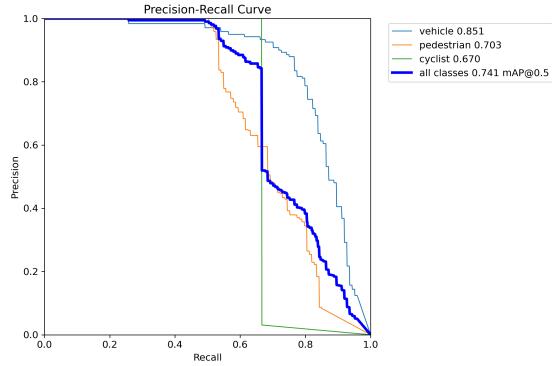


Figura 5.4: Curva Precision-Recall para YOLOv8 Preentrenado usando NuScenes

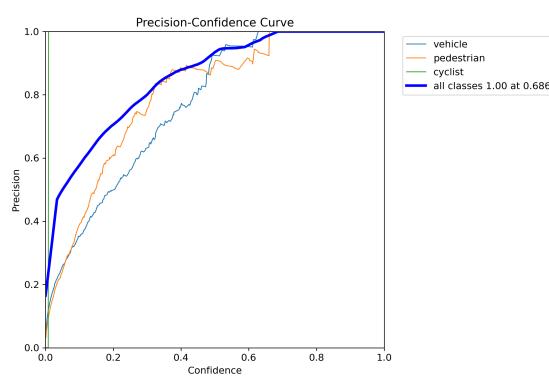


Figura 5.5: Curva Precision-Confidence para YOLOv8 desde cero usando NuScenes

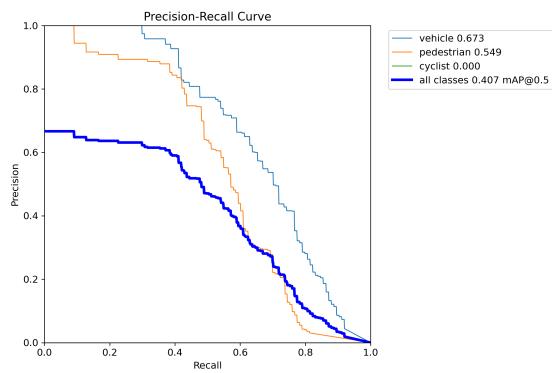


Figura 5.6: Curva Precision-Recall para YOLOv8 desde cero Preentrenado usando NuScenes

de entrenamiento, lo que significa que podría alargarse el proceso de entrenamiento para conseguir unas métricas mejores en la detección de objetos. Este hecho ocurre en ambos casos tanto para la versión preentrenada como para la versión desde cero, aunque en la versión desde cero ocurre de forma más acusada. En las Figuras 5.3 y 5.5 Precision-Confidence de las versiones pre-entrenadas y sin pre-entrenamiento, en las cuales se observa que en la gran mayoría de los casos los objetos se han detectado correctamente, incluso en casos con una confianza de la predicción más baja, sin embargo en el caso de la versión pre-entrenada el área bajo la curva es algo superior lo que quiere decir que en los casos de baja confianza funciona algo mejor esta versión.

Por otro lado, en las curvas Precision-Recall para ambas versiones (Figuras 5.4 y 5.6), se puede observar el compromiso entre clasificar correctamente las detecciones de objetos, y no detectar objetos que existen debido a una baja confianza. En este caso, se puede observar que la principal debilidad de ambos modelos es la gran cantidad de falsos negativos que existen cuando baja la precisión, lo que significa que hay muchos objetos que deberían ser detectados y no lo son debido a su baja confianza en la predicción.

Por último, en las Tabla 5.1 y 5.2, se puede observar que el modelo pre-entrenado clasifica correctamente un 77 % de los objetos son clasificados correctamente siendo el peatón el que obtiene un peor resultado, al contrario que el modelo sin pre-entrenamiento, en el cual es la clase con mayor *precision*. Revisando el AP50 y el AP5095 (junto con

mAP50 y mAP5095), las cuales son las métricas más importantes en este caso, el modelo pre-entrenado tiene un funcionamiento mucho mejor gracias las confianzas reportadas en las precisiones son mucho mayores, además de que detecta un mayor número de objetos que la versión sin pre-entrenamiento. También se puede observar que el desbalanceo de la clase ciclista en el dataset afecta al rendimiento de la misma, y es detectada en un menor número de casos y con una confianza de forma general mucho más baja.

5.1.2. Resultados para Waymo Open Dataset

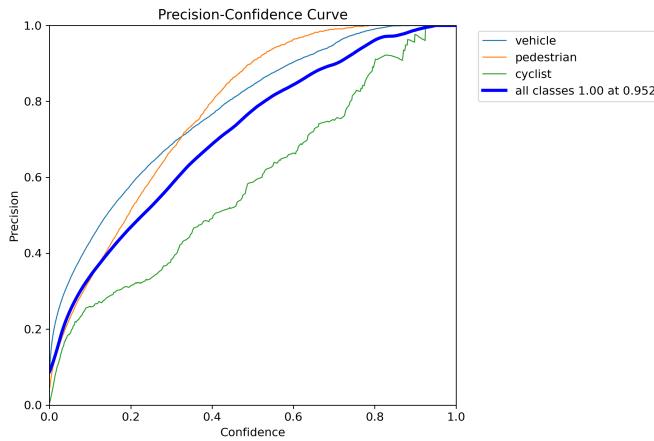


Figura 5.7: Curva Precision-Confidence para YOLOv8 Preentrenado usando Waymo

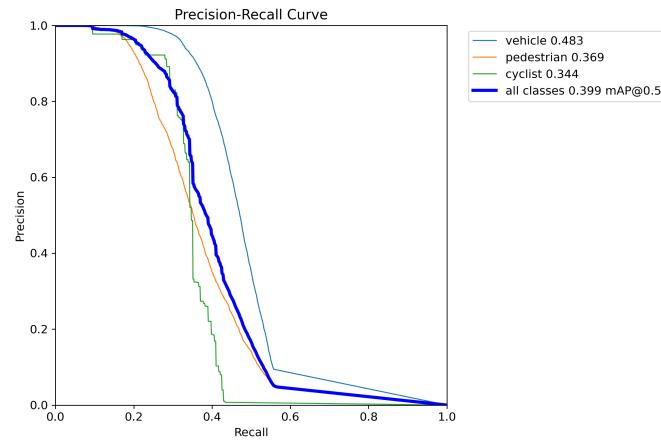


Figura 5.8: Curva Precision-Recall para YOLOv8 Preentrenado usando Waymo

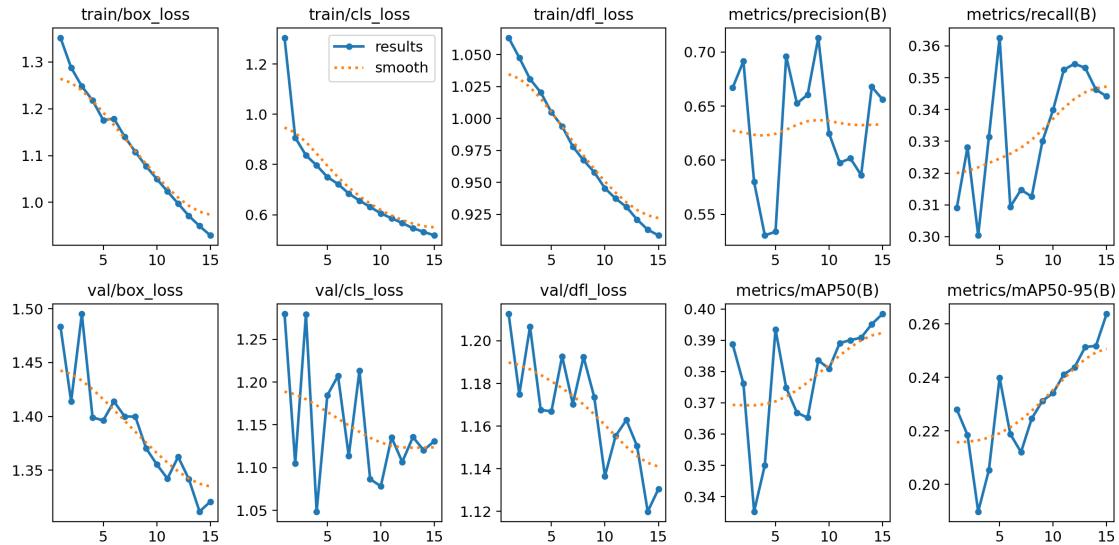


Figura 5.9: Evolución métricas durante el proceso de entrenamiento de Waymo en la versión Preentrenada sobre ImageNet

El análisis que se puede realizar sobre el dataset de Waymo arroja las mismas conclusiones que sobre el dataset NuScenes, pero obteniendo unos resultados peores de forma generalizada, esto se debe a que el dataset es mucho más grande y por lo tanto es más difícil generalizar, además de tener un proceso de entrenamiento mucho más costoso. En las Figuras 5.8 y 5.12 se puede observar como se parte de un Recall mucho más bajo, lo que quiere decir que hay un número relativamente grande de objetos que no han sido

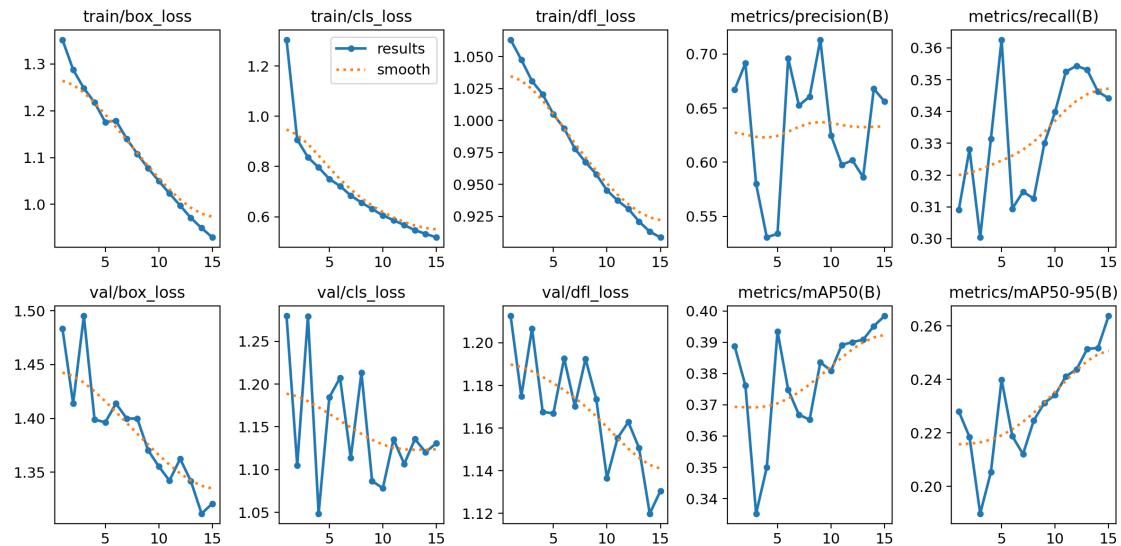


Figura 5.10: Evolución métricas durante el proceso de entrenamiento de Waymo en la versión desde cero

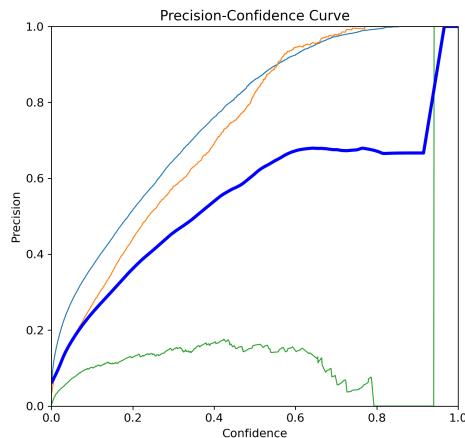


Figura 5.11: Curva Precision-Confidence para YOLOv8 desde cero usando Waymo

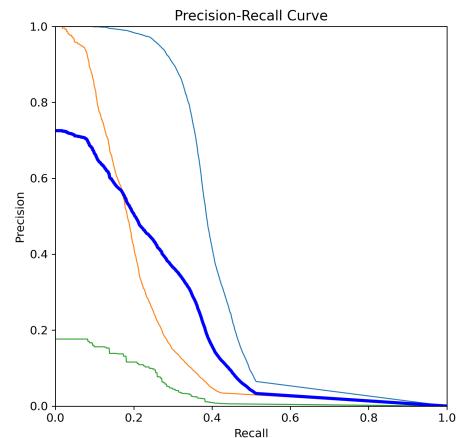


Figura 5.12: Curva Precision-Recall para YOLOv8 desde cero usando Waymo

	vehículo	peatón	ciclista	media
precision	0.742	0.755	0.476	0.658
recall	0.416	0.265	0.350	0.344
AP50	0.483	0.369	0.345	0.399
AP5095	0.334	0.191	0.267	0.264

Cuadro 5.3: Tabla de métricas por clase para YOLOv8 sobre el dataset Waymo, Preentrenado

detectados correctamente y se han considerado fondo de la imagen, provocando un recall muy bajo.

En cuanto a los resultados finales, reflejados en las Tablas 5.2 y 5.3, se puede observar

	vehículo	peatón	ciclista	media
precision	0.591	0.526	0.136	0.418
recall	0.374	0.178	0.146	0.233
AP50	0.407	0.207	0.046	0.220
AP5095	0.256	0.088	0.012	0.119

Cuadro 5.4: Tabla de métricas por clase para YOLOv8 sobre el dataset Waymo, desde cero

un comportamiento similar al dataset NuScenes, pero con unos peores resultados de forma general. Esto posiblemente se deba al menor número de épocas de entrenamiento utilizado debido a la magnitud del dataset, por lo que necesitaría un número mucho mayor de iteraciones de entrenamiento para poder obtener unos resultados mucho más positivos. En este caso, también se refleja que el desbalanceo del dataset provoca un resultado mucho peor para la clase **ciclista** que para **vehículo** y **peatón**, aunque gracias al mayor número de etiquetas de este tipo no se expresa de forma tan acusada como en el caso del dataset NuScenes.

5.2– Resultados de FasterR-CNN

Esta sección mostrará los resultados obtenidos por el modelo *FasterR-CNN*, en ambos datasets y para las versiones con *backbone* preentrenado sobre ImageNet y partiendo desde cero. En este caso se reportarán las métricas precision, recall y AP50.

Evolution of Precision, Recall, and AP50 by class in FasterR-CNN training for NuScenes, pretrained

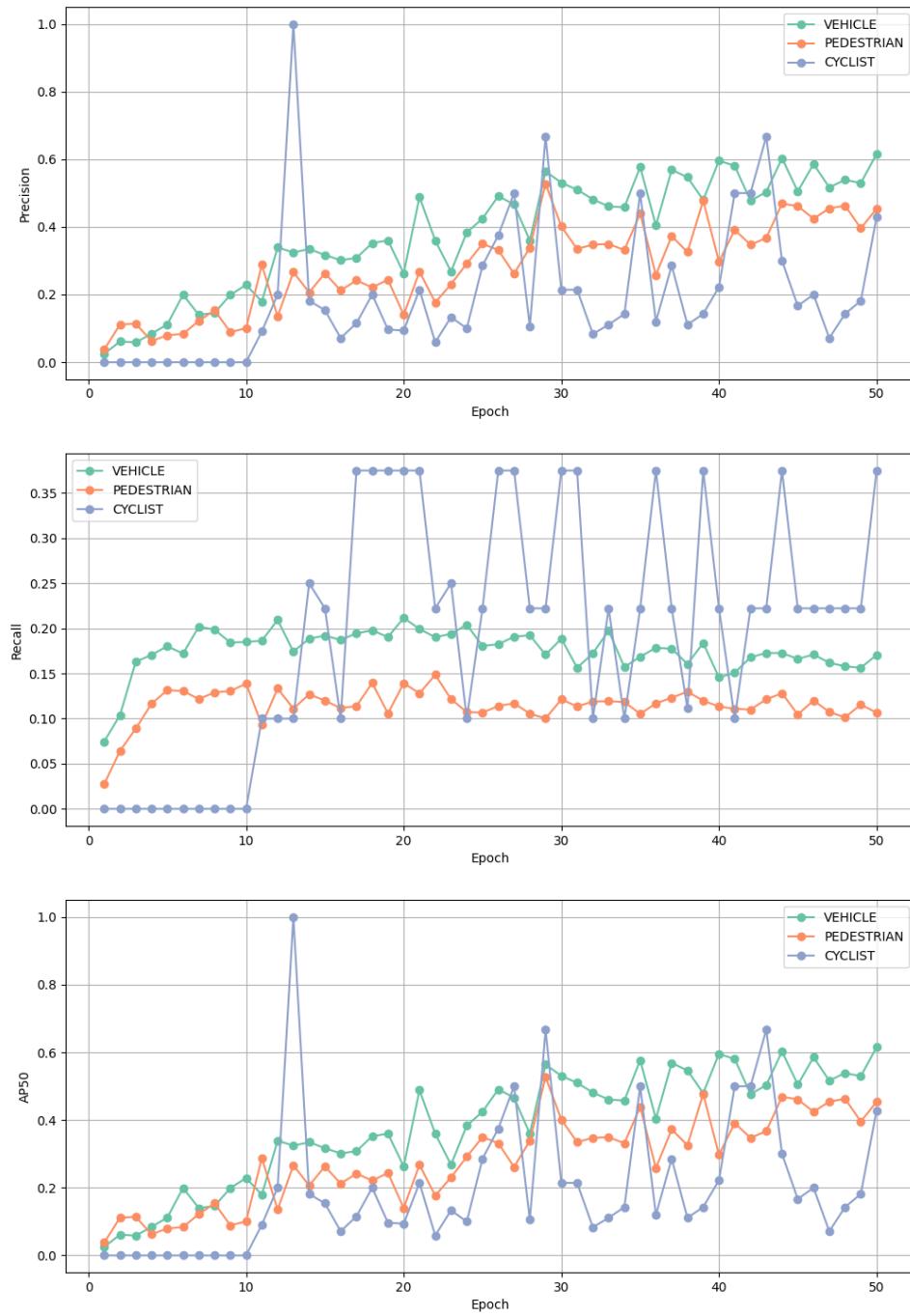


Figura 5.13: Enter Caption

CAPÍTULO 6

Planificación final, Conclusiones y Trabajo Futuro

Este capítulo se centrará en comparar el tiempo utilizado en cada tarea, con las estimaciones originales; así como las conclusiones que se pueden extraer de la tecnología evaluada siguiendo la hipótesis planteada, además de cómo se han desarrollado los objetivos planteados y cómo se podría aportar una continuidad al planteamiento de este Trabajo de Fin de Máster.

6.1– Planificación final, desviación y motivos

Tras el desarrollo de este Trabajo de Fin de Máster, y en comparación con la planificación original, se ha sobrepasado el número de horas planificado de la siguiente forma:

Tarea	Horas Estimadas	Horas finales
Estudio de posibles temas y propuestas:	10	13
Estudio de aplicaciones en el mundo real de la propuesta	25	25
Estudio del estado del arte de la temática	50	70
Creación del entorno de trabajo	30	25
Desarrollo del código	100	130
Redacción de la memoria	100	150
Extracción de conclusiones y mejoras	30	20
Total	345	433

Cuadro 6.1: Tabla comparativa de horas estimadas vs horas finales

Como refleja la tabla, la planificación final refleja más horas que la planificación inicial, lo cual se debe principalmente a la repetición en la implementación de determinadas funcionalidad, ejecución de experimentos más lenta de la esperada y una ligera ineficiencia en la redacción de la memoria debido a la coordinación con otras tareas externas.

6.2– Conclusiones, resumen del proyecto y sus resultados

Tras el estudio de la literatura y el desarrollo de la experimentación del proyecto, las cuales han consistido principalmente en la evaluación de diferentes tipos de modelos detectores de objetos (una etapa y dos etapas) sobre dos conjuntos de datos públicos centrados en conducción autónoma, así como la evaluación de la importancia de usar modelos pre-entrenados sobre otros conjuntos de datos:

- Los **modelos en una etapa (YOLO)** en este caso, se ajustan mucho mejor a la tarea de la conducción autónoma, ya que el tiempo de inferencia es mucho menor (*real-time*), y en la conducción autónoma es necesaria la detección rápida ya que las instrucciones que deben darse para la conducción deben ser instantáneas en relación al entorno en el cual está en el mismo momento de la detección. Además, no existe una diferencia considerable en cuanto al rendimiento de la detección que haga pensar que invertir un mayor tiempo en la detección sea necesario (en **modelos de dos etapas**, FasterR-CNN).
- Los modelos **pre-entrenados** consiguen una precisión en la detección mucho mejor en un número mucho **menor de épocas** de entrenamiento gracias al conocimiento heredado, además, también ayuda a detectar mejor clases infrarrepresentadas en el dataset debido a que en el conjunto de datos del pre-entrenamiento posiblemente existían ejemplos de objetos similares.
- La mayor complejidad de los modelos en dos etapas como **FasterR-CNN** hace que el proceso de entrenamiento sea también mucho más lento y por lo tanto para conseguir el rendimiento superior de los mismos hay que invertir un tiempo mucho mayor.
- El único *backbone* que puede resultar útil en esta tarea para **modelos en dos etapas** que puede ser válido para tareas en **tiempo real** como la detección de objetos en la conducción autónoma es **MobileNet** en cualquier versión ya que el tiempo de inferencia se dispara exponencialmente usando redes como **ResNet**.
- La submuestra de datos utilizada, especialmente **NuScenes**, no es suficiente para obtener un modelo realmente útil para la detección de vehículos, personas y ciclistas, por lo que sería necesario el uso de un dataset más completo para obtener un rendimiento mucho mejor, ya que para que un coche sea capaz de conducir de forma autónoma necesita una capacidad de detección muy cercana al 100 % para que supere toda la legislación vigente.

6.3– Objetivos cumplidos y trabajo futuro

En este caso, se han cumplido todos los objetivos planteados en un principio, ya que se ha estudiado un gran número de *papers* de varias temáticas y de muchos autores diferentes de los cuales se ha podido extraer una comprensión del tema importante, además también se ha comprendido el modelo de negocio de la empresa Waymo y cuál será su *roadmap* en los próximos años así como el de sus competidoras.

Además se ha podido coordinar la realización de este trabajo con un trabajo a jornada completa, lo cual siempre requiere de un esfuerzo adicional. Por último, se han usado dos conjuntos de datos abiertos como son *Waymo Open Dataset* y *NuScenes* de forma correcta y aplicando todo el preprocesamiento necesario, todo ello desarrollando modelos basados en Redes Convolucionales para poder aplicar la detección de objetos, tanto usando modelos ya pre-entrenados como modelos desde cero.

En relación al trabajo futuro se plantean diferentes puntos a mejorar:

- **Utilización de datos adicionales a las imágenes:** Se plantea usar información adicional a las imágenes como pueden ser datos de *Radar*, aplicando el modelo explicado en la Subsección 3.3.2. Al usar este sistema, se puede aplicar como **RPN** en **FasterR-CNN** de forma que se pueden obtener los beneficios en la velocidad de procesamiento de **YOLO** en un sistema con mejor precisión de forma general que éste.
- **Ajuste de hiperparámetros:** En los experimentos de este Trabajo se ha hecho uso de una configuración estática que no ha variado a lo largo de los mismos, por lo tanto en futuras mejoras de este Trabajo se podría aplicar un ajuste de hiperparámetros para poder obtener un rendimiento mucho más perfeccionado con mejor generalización.
- **Aplicación de *data augmentation*:** En este caso tampoco se han aplicado técnicas de *data augmentation*, por lo cual se podrían obtener mejores resultados y una capacidad de generalización mucho mayor en aplicaciones del mundo real.
- **Modificación de los *backbones* convolucionales:** Para este Trabajo de Fin de Máster se ha utilizado redes convolucionales ya definidas, pero los resultados se podrían mejorar utilizando otras redes convolucionales diferentes o modificando las mismas añadiendo o eliminando capas para obtener una estructura que funcione mejor para este problema concreto.

6.4– Reflexión final

En este Trabajo de Fin de Máster, la comparación se han comparado los algoritmos **YOLO** y **Faster R-CNN** aplicados a tareas de conducción autónoma. Ambos algoritmos representan enfoques avanzados y complementarios en la detección de objetos, pero su rendimiento, precisión y eficiencia varían en función de los requisitos y condiciones específicas de una conducción autónoma segura y eficaz.

A lo largo de este estudio, se ha podido observar cómo **YOLO**, con su enfoque en la velocidad y capacidad de procesamiento en tiempo real, ofrece una solución adecuada para aplicaciones donde la inmediatez es fundamental, como en la detección de peatones que puedan invadir la carretera. Su diseño optimizado permite alcanzar resultados satisfactorios en situaciones dinámicas, aunque con un pequeño sacrificio en precisión en comparación con **Faster R-CNN**. Este último, en cambio, ha mostrado un rendimiento superior en tareas que requieren alta precisión, aunque a costa de una mayor demanda computacional y un menor rendimiento en tiempo real.

Estos resultados reflejan, en el fondo, el delicado equilibrio que enfrenta la conducción autónoma entre seguridad y eficiencia. En el futuro, la elección de un método u otro, dependerá principalmente de si se consigue mejorar la capacidad de procesamiento a un nivel que permita usar modelos más complejos como **FasterR-CNN**. Este proyecto ha reafirmado la importancia de desarrollar y probar diferentes modelos para entender no solo sus capacidades técnicas, sino también sus limitaciones en escenarios complejos y desafiantes, como los que plantea la conducción autónoma en el mundo real.

Para finalizar, esta investigación no solo demuestra los logros actuales de la visión artificial, sino también la necesidad de seguir avanzando en su integración con otras tecnologías de percepción y toma de decisiones. La comparación entre **YOLO** y **Faster R-CNN** no es un fin en sí mismo, sino un paso en la búsqueda constante de sistemas más robustos, seguros y eficientes que, en última instancia, permitan que los vehículos autónomos se conviertan en una realidad accesible y confiable.

CAPÍTULO 7

Bibliografía

Referencias

- Al Musalhi, N., Wahaibi, A., y Abbas, M. (2024, 05). Implementing real-time visitor counter using surveillance video and mobilenet-ssd object detection: The best practice. *Baghdad Science Journal*, 21, 1775. doi: 10.21123/bsj.2024.10540
- Audi autonomous run in pike's peak. (2010). https://web.archive.org/web/20120710202052/http://www.audiusa.com/us/brand/en/tools/news/pool/2010/07/new_look_reaffirmed.html.
- Azuri, I., Goldian, I., Regev-Rudzki, N., Fantner, G., y Cohen, S. (2021, 08). The role of convolutional neural networks in scanning probe microscopy: a review. *Beilstein Journal of Nanotechnology*, 12, 878-901. doi: 10.3762/bjnano.12.66
- Behera, P., Siddique, A., Delwar, T., Biswal, M., Choi, Y., y Ryu, J.-Y. (2022, 06). A novel 65 nm active-inductor-based vco with improved q-factor for 24 ghz automotive radar applications. *Sensors (Basel, Switzerland)*, 22. doi: 10.3390/s22134701
- Bochkovskiy, A., Wang, C.-Y., y Liao, H.-Y. M. (2020). *Yolov4: Optimal speed and accuracy of object detection*. Descargado de <https://arxiv.org/abs/2004.10934>
- Brito, P., Fontes, J., Miquelina, N., y Guevara Lopez, M. A. (2018, 11). Agatha: Benchmarking datasets for exploring criminal surveillance methods on open source data.. doi: 10.1109/ITCGI.2018.8602903
- Buddi, P., Moorthy, C. V. K. N. S. N., Venkateswarulu, N., y Myneni, D. (2023, 05). Exploration of issues, challenges and latest developments in autonomous cars. *Journal of Big Data*, 10. doi: 10.1186/s40537-023-00701-y
- Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., ... Beijbom, O. (2020). *nuscenes: A multimodal dataset for autonomous driving*. Descargado de <https://arxiv.org/abs/1903.11027>
- Chazhoor, A., y Sarobin, V. (2022, 08). Intelligent automation of invoice parsing using computer vision techniques. *Multimedia Tools and Applications*, 81. doi: 10.1007/s11042-022-12916-x
- Chen, Y., Li, L., Li, W., Guo, Q., Du, Z., y Xu, Z. (2024). Chapter 2 - fundamentals of neural networks. En Y. Chen, L. Li, W. Li, Q. Guo, Z. Du, y Z. Xu (Eds.),

- Ai computing systems* (p. 17-51). Morgan Kaufmann. Descargado de <https://www.sciencedirect.com/science/article/pii/B9780323953993000081> doi: <https://doi.org/10.1016/B978-0-32-395399-3.00008-1>
- Dalal, N., y Triggs, B. (2005). Histograms of oriented gradients for human detection. En *2005 ieee computer society conference on computer vision and pattern recognition (cvpr'05)* (Vol. 1, p. 886-893 vol. 1). doi: 10.1109/CVPR.2005.177
- Deep, S., y Zheng, X. (2019). Leveraging cnn and transfer learning for vision-based human activity recognition. *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, 1-4. Descargado de <https://api.semanticscholar.org/CorpusID:216588172>
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. En *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255). Descargado de <https://www.image-net.org/>
- Diwan, T., Anirudh, G., y Tembhurne, J. V. (2023, 01 de Mar). Object detection using yolo: challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, 82(6), 9243-9275. Descargado de <https://doi.org/10.1007/s11042-022-13644-y> doi: 10.1007/s11042-022-13644-y
- Driver-less car to be demonstrated about city streets next saturday - controlled entirely by radio.* (1932). <https://news.google.com/newspapers?id=PthNAAAAIBAJ&sjid=yYoDAAAIBAJ&hl=es&pg=6442%2C3879017>.
- Engilberge, M., Shi, H., Wang, Z., y Fua, P. (2022, 10). *Two-level data augmentation for calibrated multi-view detection.* doi: 10.48550/arXiv.2210.10756
- Felzenszwalb, P. F., Girshick, R. B., McAllester, D., y Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9), 1627-1645. doi: 10.1109/TPAMI.2009.167
- Fu, C.-Y., Liu, W., Ranga, A., Tyagi, A., y Berg, A. C. (2017). *Dssd : Deconvolutional single shot detector.* Descargado de <https://arxiv.org/abs/1701.06659>
- Fujii, S., Akita, K., y Ukita, N. (2021, 07). Distant bird detection for safe drone flight and its dataset. En (p. 1-5). doi: 10.23919/MVA51890.2021.9511386
- Girshick, R. (2015). Fast r-cnn. En *2015 ieee international conference on computer vision (iccv)* (p. 1440-1448). doi: 10.1109/ICCV.2015.169
- Girshick, R., Donahue, J., Darrell, T., y Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation.* Descargado de <https://arxiv.org/abs/1311.2524>
- Han, H., Hou, J., Bai, G., Li, B., Wang, T., Li, X., ... Gong, J. (2021, 05). A deep learning technique-based automatic monitoring method for experimental urban road inundation. *Journal of Hydroinformatics*, 23. doi: 10.2166/hydro.2021.156
- Hansen, P. C. (1987, 01 de Dec). The truncatedsvd as a method for regularization. *BIT Numerical Mathematics*, 27(4), 534-553. Descargado de <https://doi.org/10.1007/BF01937276> doi: 10.1007/BF01937276
- He, K., Zhang, X., Ren, S., y Sun, J. (2015). *Deep residual learning for image recognition.* Descargado de <https://arxiv.org/abs/1512.03385>
- Hevner, A., y Chatterjee, S. (2010). Design science research in information systems. En *Design research in information systems: Theory and practice* (pp. 9–22). Boston, MA: Springer US. Descargado de https://doi.org/10.1007/978-1-4419-5653-8_2 doi: 10.1007/978-1-4419-5653-8_2
- Hosang, J., Benenson, R., y Schiele, B. (2017). *Learning non-maximum suppression.* Descargado de <https://arxiv.org/abs/1705.02950>
- Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L.-C., Tan, M., ... Le, Q. (2019). Searching for mobilenetv3. En *2019 ieee/cvf international conference on computer*

- vision (iccv)* (p. 1314-1324). doi: 10.1109/ICCV.2019.00140
- Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017, 04). Mobilenets: Efficient convolutional neural networks for mobile vision applications.
doi: 10.48550/arXiv.1704.04861
- Indolia, S., Goswami, A. K., Mishra, S., y Asopa, P. (2018). Conceptual understanding of convolutional neural network- a deep learning approach. *Procedia Computer Science*, 132, 679-688. Descargado de <https://www.sciencedirect.com/science/article/pii/S1877050918308019> (International Conference on Computational Intelligence and Data Science) doi: <https://doi.org/10.1016/j.procs.2018.05.069>
- Ioffe, S., y Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. Descargado de <https://arxiv.org/abs/1502.03167>
- Jocher, G. (2020). *Ultralytics yolov5*. Descargado de <https://github.com/ultralytics/yolov5> doi: 10.5281/zenodo.3908559
- Jocher, G., Chaurasia, A., y Qiu, J. (2023). *Ultralytics yolov8*. Descargado de <https://github.com/ultralytics/ultralytics>
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., y Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 541-551. doi: 10.1162/neco.1989.1.4.541
- Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., ... Dollár, P. (2015). *Microsoft coco: Common objects in context*. Descargado de <https://arxiv.org/abs/1405.0312>
- Liu, S., y Deng, W. (2015). Very deep convolutional neural network based image classification using small training sample size. En *2015 3rd iapr asian conference on pattern recognition (acpr)* (p. 730-734). doi: 10.1109/ACPR.2015.7486599
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., y Berg, A. C. (2016). Ssd: Single shot multibox detector. En *Computer vision – eccv 2016* (p. 21–37). Springer International Publishing. Descargado de http://dx.doi.org/10.1007/978-3-319-46448-0_2 doi: 10.1007/978-3-319-46448-0_2
- LLC, W. R. (2019). *Waymo open dataset: An autonomous driving dataset*. <https://github.com/waymo-research/waymo-open-dataset>.
- Lowe, D. G. (2004, 01 de Nov). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2), 91-110. Descargado de <https://doi.org/10.1023/B:VISI.0000029664.99615.94> doi: 10.1023/B:VISI.0000029664.99615.94
- Maylor, H. (2001). Beyond the gantt chart:: Project management moving on. *European Management Journal*, 19(1), 92-100. Descargado de <https://www.sciencedirect.com/science/article/pii/S0263237300000748> doi: [https://doi.org/10.1016/S0263-2373\(00\)00074-8](https://doi.org/10.1016/S0263-2373(00)00074-8)
- Muhammad, K., Ullah, A., Lloret, J., Ser, J. D., y de Albuquerque, V. H. C. (2021). Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7), 4316-4336. doi: 10.1109/TITS.2020.3032227
- Mumuni, A., y Mumuni, F. (2022). Data augmentation: A comprehensive survey of modern approaches. *Array*, 16, 100258. Descargado de <https://www.sciencedirect.com/science/article/pii/S2590005622000911> doi: <https://doi.org/10.1016/j.array.2022.100258>
- Nabati, R., y Qi, H. (2019). Rrpn: Radar region proposal network for object detection in autonomous vehicles. En *2019 ieee international conference on image processing (icip)* (p. 3093-3097). doi: 10.1109/ICIP.2019.8803392

- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. En *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Descargado de <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- Pomerleau, D. (1989, December). Alvinn: An autonomous land vehicle in a neural network. En D. Touretzky (Ed.), *Proceedings of (neurips) neural information processing systems* (p. 305 - 313). Morgan Kaufmann.
- Rahman, M. A., y Wang, Y. (2016). Optimizing intersection-over-union in deep neural networks for image segmentation. En G. Bebis y cols. (Eds.), *Advances in visual computing* (pp. 234–244). Cham: Springer International Publishing.
- Rashed, H., Mohamed, E., Sistu, G., Kumar, V. R., Eising, C., El-Sallab, A., y Yogamani, S. (2020). Fisheyeyolo: Object detection on fisheye cameras for autonomous driving. En *Proceedings of the machine learning for autonomous driving neurips 2020 virtual workshop, virtual* (Vol. 11).
- Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2016). *You only look once: Unified, real-time object detection*. Descargado de <https://arxiv.org/abs/1506.02640>
- Redmon, J., y Farhadi, A. (2017). Yolo9000: Better, faster, stronger. En *2017 ieee conference on computer vision and pattern recognition (cvpr)* (p. 6517-6525). doi: 10.1109/CVPR.2017.690
- Redmon, J., y Farhadi, A. (2018). *Yolov3: An incremental improvement*. Descargado de <https://arxiv.org/abs/1804.02767>
- Ren, J., y Wang, H. (2023). Chapter 3 - calculus and optimization. En J. Ren y H. Wang (Eds.), *Mathematical methods in data science* (p. 51-89). Elsevier. Descargado de <https://www.sciencedirect.com/science/article/pii/B9780443186790000090> doi: <https://doi.org/10.1016/B978-0-44-318679-0.00009-0>
- Ren, S., He, K., Girshick, R., y Sun, J. (2017). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6), 1137-1149. doi: 10.1109/TPAMI.2016.2577031
- Roy, A. M., Bose, R., y Bhaduri, J. (2022, 03). A fast accurate fine-grain object detection model based on yolov4 deep neural network. *Neural Computing and Applications*, 34, 1-27. doi: 10.1007/s00521-021-06651-x
- Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., y LeCun, Y. (2014). *Overfeat: Integrated recognition, localization and detection using convolutional networks*. Descargado de <https://arxiv.org/abs/1312.6229>
- Shima, I. S., Gital, A. Y., Gamsha, A. M., Lawal, M. A., Patrick, K. O., y Chukwuka, O. C. (2023). A review on deep learning algorithms for real-time detection of multiple vehicle-based classes: Challenges and open opportunities. En G. Mathur, M. Bundele, A. Tripathi, y M. Paprzycki (Eds.), *Proceedings of 3rd international conference on artificial intelligence: Advances and applications* (pp. 363–377). Singapore: Springer Nature Singapore.
- Software, C. (s.f.). *The use of radar technology in autonomous vehicles*. <https://resources.system-analysis.cadence.com/blog/msa2022-the-use-of-radar-technology-in-autonomous-vehicles>.
- Sumarsono, S., Sakkina, I., Permanasari, A., y Pranggono, B. (2022, 07). Development of a mobile health infrastructure for non-communicable diseases using design science research method: a case study. *Journal of Ambient Intelligence and Humanized Computing*, 14, 1-12. doi: 10.1007/s12652-022-04322-w
- Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., ... Anguelov, D. (2020). Scalability in perception for autonomous driving: Waymo open dataset. En *2020 ieee/cvf conference on computer vision and pattern recognition*

- (cvpr) (p. 2443-2451). doi: 10.1109/CVPR42600.2020.00252
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2014). *Going deeper with convolutions*. Descargado de <https://arxiv.org/abs/1409.4842>
- Tesla autopilot*. (s.f.). https://www.tesla.com/es_ES/autopilot.
- Turing, A. M. (1950, 10). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236), 433-460. Descargado de <https://doi.org/10.1093/mind/LIX.236.433> doi: 10.1093/mind/LIX.236.433
- Van Rossum, G., y Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.
- Venkateswarlu, C., y Karri, R. R. (2022). Chapter 5 - data-driven modeling techniques for state estimation. En C. Venkateswarlu y R. R. Karri (Eds.), *Optimal state estimation for process monitoring, fault diagnosis and control* (p. 91-111). Elsevier. Descargado de <https://www.sciencedirect.com/science/article/pii/B9780323858786000105> doi: <https://doi.org/10.1016/B978-0-323-85878-6.00010-5>
- Wang, X., Wang, A., Yi, J., Song, Y., y Chehri, A. (2023, 06). Small object detection based on deep learning for remote sensing: A comprehensive review. *Remote Sensing*, 15, 3265. doi: 10.3390/rs15133265
- Waymo open dataset*. (2019). <https://paperswithcode.com/dataset/waymo-open-dataset>.
- Wu, Y.-c., y Feng, J.-w. (2018, 01 de Sep). Development and application of artificial neural network. *Wireless Personal Communications*, 102(2), 1645-1656. Descargado de <https://doi.org/10.1007/s11277-017-5224-x> doi: 10.1007/s11277-017-5224-x
- www.xataka.com. (2016). *Ya están aquí, los primeros coches autónomos de uber arrancan operaciones en pittsburgh*. Descargado de <https://www.xataka.com/automovil/ya-estan-aqui-los-primeros-coches-autonomos-de-uber-arrancan-operaciones-en-pittsburgh>
- Xie, S., Girshick, R., Dollár, P., Tu, Z., y He, K. (2017). *Aggregated residual transformations for deep neural networks*. Descargado de <https://arxiv.org/abs/1611.05431>

CAPÍTULO 8

ANEXOS

- a) Anexo I Todo el código utilizado está disponible en el siguiente repositorio de GitHub:
https://github.com/rmanzano-dev/Object_detection_TFM
- b) Anexo II