Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Desarrollo de un modelo de IA para la detección y el reconocimiento de señales de tráfico en imágenes

---

Alumno: Gonzalo Bobillo Rincón

Tutores: Fernando Díaz Gómez
Vincenzo Pagano

Desarrollo de un modelo de IA para la detección y el reconocimiento de señales de tráfico en imágenes

Gonzalo Bobillo Rincón

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Quiero agradecer a todas las personas que han hecho posible la realización de este trabajo. A mis padres por su apoyo incondicional en todo momento, a Fernando Díaz por estar pendiente hasta el último segundo, a Vincenzo Pagano por darme la oportunidad de llevar a cabo este proyecto en AVL, a Quique Martínez por su disposición y amabilidad a la hora de ayudarme con cualquier problema, a mis compañeros de piso Miguel y David que han hecho más fácil la travesía de este último año en Segovia, a Marina por su cariño, paciencia y por siempre estar ahí y a mi perro Gauss, que me ha aguantado incluso en los momentos donde todo se veía más oscuro.

Resumen

Recientemente, los avances en inteligencia artificial y visión por computador han impulsado el desarrollo de tecnologías innovadoras en varios sectores, entre ellos, la industria automotriz. Los vehículos autónomos y ADAS han aprovechado estas técnicas para interpretar el entorno con mayor precisión y rapidez. Una de las áreas clave para la seguridad en carretera y la efectividad de estos sistemas es la identificación de señales de tráfico, esenciales en la toma de decisiones en la conducción.

Este trabajo se enfoca en la creación de un modelo de *deep learning*, específicamente YOLOv7, para la detección y clasificación de señales de tráfico en imágenes. El desarrollo del proyecto incluye la recopilación y el preprocesamiento de conjuntos de datos públicos, la implementación, entrenamiento y evaluación del modelo y el análisis de varios hiperparámetros críticos. Los resultados sugieren un gran potencial para mejorar la seguridad y fiabilidad de los sistemas autónomos, proponiendo futuras investigaciones para aumentar la precisión y adaptabilidad del modelo a entornos más complejos, impulsando aún más la integración de la IA en la industria automotriz.

Palabras claves: Inteligencia artificial, *deep learning*, detección de objetos, señales de tráfico, ADAS, YOLO.

Abstract

Recently, advances in artificial intelligence and computer vision have driven the development of innovative technologies in various sectors, including the automotive industry. Autonomous vehicles and ADAS have leveraged these techniques to interpret the environment with greater precision and speed. One of the key areas for road safety and the effectiveness of these systems is the identification of traffic signs, which are essential for driving decision-making.

This work focuses on the creation of a deep learning model, specifically YOLOv7, for the detection and classification of traffic signs in images. The project development includes the collection and preprocessing of public datasets, implementation, training, and evaluation of the model, as well as the analysis of several critical hyperparameters. The results suggest a significant potential to improve the safety and reliability of autonomous systems, proposing future research to increase the model's accuracy and adaptability to more complex environments, further advancing the integration of AI in the automotive industry.

Keywords: Artificial intelligence, deep learning, object detection, traffic signs, ADAS, YOLO.

Parte I

Descripción del Proyecto

Capítulo 1

Introducción

En los últimos años, el avance de la inteligencia artificial (IA) y la visión por computador ha permitido el desarrollo de tecnologías revolucionarias en diversos campos, especialmente en la industria automotriz [33]. Los vehículos autónomos y los sistemas avanzados de asistencia al conductor (ADAS, *Advanced Driver Assistance Systems*) se han beneficiado de estas nuevas técnicas, que son capaces de interpretar el entorno de manera precisa y rápida. Una de las áreas críticas para la seguridad vial y la eficacia de estos sistemas es la capacidad de detectar y reconocer correctamente las señales de tráfico, que proporcionan información esencial para la toma de decisiones en la conducción.

Sin embargo, la variabilidad en las condiciones ambientales (lluvia, niebla, luminosidad cambiante), obstrucciones parciales (por ejemplo, debido a la vegetación), el desgaste de las señales y su diversidad en diferentes regiones plantean desafíos considerables para los sistemas de reconocimiento. Superar estos obstáculos es crucial para mejorar la eficacia y seguridad de los sistemas autónomos.

Este trabajo se centra en el desarrollo de un modelo de IA capaz de detectar y reconocer señales de tráfico a partir de imágenes, con el objetivo de contribuir al avance de los sistemas de conducción autónoma y ADAS.

La idea para este proyecto surge a partir de la realización de las prácticas obligatorias del Grado, en la empresa AVL, donde me proponen este trabajo y lo acepto. La visión por ordenador es un campo que está avanzando muy rápidamente durante estos últimos años y consideré que era una gran oportunidad para adentrarme en él y aprender todo lo posible sobre las técnicas de detección y reconocimiento de objetos, en este caso aplicadas a las señales de tráfico.

1.1. Planteamiento del problema

En el contexto de los vehículos autónomos y los sistemas avanzados de asistencia al conductor, la identificación precisa de señales de tráfico es crucial para la conducción. La información necesaria para esta tarea se obtiene mediante una cámara del vehículo, que captura un vídeo continuo de la carretera. El desafío consiste en procesar este vídeo para extraer y reconocer las señales de tráfico presentes. Este problema puede ser simplificado

al descomponer el vídeo en fotogramas individuales, lo que permite tratar cada fotograma como una imagen estática. La tarea central, por tanto, se reduce a identificar las posibles señales de tráfico contenidas en cada una de estas imágenes.

Existen muchos modelos que solucionan el problema de detectar y clasificar objetos en imágenes. En este trabajo desarrollaré un modelo que obtenga una alta precisión, en el menor tiempo de procesamiento posible, dentro de los recursos computacionales a los que tengo acceso.

En concreto, he desarrollado este proyecto centrándome en la identificación de señales de límite de velocidad, como objetivo para la empresa a la que va dirigido dada su importancia crítica para la regulación del comportamiento del vehículo en la carretera.

1.2. Objetivos del trabajo

Objetivo general: desarrollar un modelo de IA capaz de detectar y reconocer señales de tráfico a partir de imágenes con alta precisión y eficiencia.

Objetivos específicos:

1. Investigación del estado del arte
2. Adquisición y preprocesado de los datos
3. Entrenamiento del modelo
4. Análisis y comunicación de resultados

Restricciones:

- **Duración del trabajo**. Las prácticas tienen una duración de 495 horas, dentro de las cuales hay otras actividades a desarrollar aparte de este proyecto, como formación específica de la empresa o reuniones. Por tanto, el alcance quedará limitado por este factor.
- **Recursos computacionales**. El entrenamiento de modelos de aprendizaje profundo usualmente utilizados en la detección automática de objetos es una tarea computacionalmente muy costosa. No tengo acceso a una GPU (*Graphics Processing Unit*) de alto rendimiento, por tanto no se podrán usar los modelos que requieran una gran cantidad de recursos.

1.3. Estructura de la memoria

La memoria está dividida en varias partes.

1. Descripción del proyecto.

- Introducción. Se presenta el contexto y la relevancia del proyecto, así como una visión general del mismo, junto a los objetivos y restricciones.
- Metodología. Se detalla el enfoque y los métodos empleados para llevar a cabo el proyecto.
- Planificación. Se describen las etapas del proyecto, la organización temporal y el presupuesto económico.

2. Marco teórico.

   - Inteligencia Artificial y Aprendizaje Profundo. Se explican los conceptos principales sobre los que se trabaja.
   - Clasificación de imágenes y detección de objetos. Se describen las métricas de evaluación y se analizan algunos de los modelos más usados en estos campos.

3. Desarrollo de la propuesta y resultados.

   - Modelo elegido: YOLOV7. Se justifica la elección del modelo y se explica en profundidad.
   - Dominio de aplicación. Se describe el contexto específico en el que se aplica el modelo: la detección y reconocimiento de señales de tráfico.
   - Conjuntos de datos. Se detallan las fuentes de los datos y el preprocesado necesario para construir los *datasets* con los que se entrena el modelo.
   - Entrenamientos. Se explican los diferentes entrenamientos realizados y los ajustes efectuados.
   - Resultados y conclusiones. Se analizan los resultados obtenidos a partir de los experimentos y se resumen las conclusiones.
   - Trabajo futuro. Se sugieren futuras líneas de investigación.

4. Apéndices: contenido adicional y manual. Se incluyen detalles adicionales como la estructura del material adjunto y un manual de uso del modelo final.

   En último lugar se encuentra la bibliografía consultada.

Capítulo 2

Metodología

2.1. Herramientas utilizadas

2.1.1. Microsoft Teams

Microsoft Teams [38] ha sido la herramienta elegida para la comunicación con mi tutor en la empresa. Ofrece varios servicios, como videoconferencias, calendario o mensajería instantánea, incluyendo el envío de imágenes u otro tipo de archivos.

2.1.2. Visual Studio Code

Visual Studio Code [39] es un editor de código desarrollado por Microsoft. Goza de una gran aceptación en la comunidad de programadores debido a su enorme variedad de extensiones, que permiten añadir funcionalidades extra. He elegido utilizar este programa a lo largo del proyecto por la familiaridad que he desarrollado gracias a su uso en otras asignaturas del Grado. Otro factor ha sido la integración con la herramienta GitHub Copilot [15], asistente de programación que facilita la creación automática de código o la búsqueda de errores.

2.1.3. GitHub

Github [14] es una plataforma de desarrollo colaborativo donde se gestionan y comparten proyectos de software. Yo la he utilizado para acceder al repositorio oficial del modelo YOLOv7 [59], que he descargado para poder manipularlo de forma local. Aparte del código fuente, se encuentran ejemplos y guías para poder sacar el máximo partido al modelo. Una sección muy importante dentro del repositorio es la de *Issues*, donde los usuarios comparten sus problemas y preguntas sobre el código y obtienen respuestas de otros usuarios o del mismo creador.

2.1.4. Anaconda

Anaconda [3] es una distribución libre de Python y R, que está especialmente diseñada para la ciencia de datos. Permite manejar paquetes y gestionar entornos virtuales de forma sencilla. Lo he usado para crear un entorno de Python 3.9 con todos los paquetes necesarios para llevar a cabo el proyecto, y de esta forma aislarlo del resto de paquetes previamente instalados en mi ordenador, para no tener problemas de compatibilidad de versiones.

2.1.5. Librerías

En este proyecto se han hecho uso de diversas librerías de python con distintos objetivos: facilitar la lectura de archivos json con la librería `json`, librerías propias de vision computacional como `torchvision`, manejar los archivos del sistema con la librería `os`, creación de gráficos con `seaborn` y `mathplotlib`, y otras más, que se explicarán más profundamente en la parte III.

2.1.6. LaTeX

En la redacción de esta memoria he usado LaTeX[28], un sistema de composición de textos de alta calidad, ampliamente utilizado en la creación de literatura de carácter científico. Proporciona más flexibilidad de cara al usuario que otras herramientas como Microsoft Word y permite escribir fácilmente todo tipo de expresiones matemáticas. A través de la plataforma Overleaf [44] (editor de texto en línea especializado en LaTeX) se ha escrito toda la memoria, permitiendo acceder a los archivos que la componen desde cualquier dispositivo, ya que se guardan en una nube. Además, esta plataforma proporciona una gran ayuda pudiendo visualizar los comandos que se escriben en tiempo real. Por ello se ha elegido esta alternativa frente a otros programas de edición de LaTeX.

2.1.7. Mendeley

Mendeley [37] es un software de gestión de referencias y red social académica que permite a los investigadores organizar y compartir documentos o colaborar en línea. Principalmente lo he usado porque es un programa facilita la gestión de bibliografías y citas y simplifica el proceso de creación de referencias, en este caso en formato `bib`. Su plataforma intuitiva y sus herramientas de organización lo convierten en una herramienta muy útil en el sector de la investigación.

2.1.8. ChatGPT

ChatGPT es un modelo de lenguaje desarrollado por OpenAI [43], basado en la arquitectura GPT (*Generative Pre-trained Transformer*). Utiliza técnicas de aprendizaje profundo para comprender y generar texto de manera coherente y contextual. ChatGPT puede responder preguntas, redactar textos, traducir idiomas y asistir en la generación de contenido, lo que lo convierte en una herramienta valiosa para tareas de procesamiento

de lenguaje natural. En el contexto de este proyecto, ha sido utilizado como asistente de programación (al igual que GitHub Copilot), aplicándose también como herramienta de apoyo en la corrección sintáctica del texto generado. Ha servido, asímismo, para producir tablas en código LaTeXo aplicar formateo de texto.

2.2. Definición de siglas y abreviaturas

Siglas	Significado
IA	Inteligencia Artificial
ADAS	*Advanced Driver Assistance Systems* (Sistemas Avanzados de Asistencia al Conductor)
ML	*Machine Learning*
NLP	*Natural Language Processing*
CNN	*Convolutional Neural Network* (Red Neuronal Convolucional)
IoU	*Intersection over Union* (Intersección sobre Unión)
mAP	*mean Average Precision*
NMS	*Non Maximun Suppression*
CPU	*Central Processing Unit* (Unidad Central de Procesamiento)
GPU	*Graphics Processing Unit* (Unidad de Procesamiento Gráfico)
ROI	*Region of Interest* (Región de Interés)
RGB	*Red*, *Green*, *Blue* (Rojo, Verde, Azul - formato de imagen)
JSON	JavaScript Object Notation
FPS	*Frames Per Second* (Imágenes por Segundo)
YOLO	*You Only Look Once*
PERT	*Program Evaluation and Review Technique*
ECTS	*European Credit Transfer System* (Sistema Europeo de Transferencia de Créditos)
RRHH	Recursos Humanos
TFG	Trabajo Fin de Grado

Tabla 2.1: Siglas usadas a lo largo de la memoria

Capítulo 3

Planificación

En este capítulo se aboradarán las cuestiones relativas a la planificación del trabajo.

3.1. Estimación temporal

Un Trabajo Fin de Grado consta de 12 créditos ECTS. Cada crédito ECTS equivale a 25 horas de trabajo según la universidad de Valladolid [56]. Por tanto, se deberían dedicar 300 horas a un TFG. Sin embargo, dado que mi proyecto forma parte de la realización de las prácticas de empresa curriculares, el número de horas dedicadas a este TFG será mayor.
El periodo de prácticas comienza el 8 de enero y termina el 31 de mayo, dedicando 5 horas diarias de lunes a viernes. Teniendo en cuentas las vacaciones de Semana Santa y festivos, suman un total de 495 horas laborables. Suponiendo que actividades de formación de la empresa y reuniones sobre temas ajenos a este proyecto ocupen un 10 % de ese tiempo, se estiman 445 horas exclusivas sobre el tema de trabajo.
Se usará la técnica PERT (*Program Evaluation and Review Technique*) [9] para estimar el tiempo que llevará finalizar el proyecto. La duración de las tareas se considera una variable aleatoria con distribución Beta, y aproximaremos el tiempo esperado (TE) a partir de los valores T_o, T_m, T_p .

- T_o es el tiempo optimista, suponiendo que no hay ningún problema y sale al primer intento.
- T_m el tiempo más probable, bajo condiciones normales.
- T_p es el tiempo pesimista, donde ocurren contratiempos que retrasarían el tiempo de finalización de la tarea.
- TE es el tiempo esperado, la esperanza media de la variable aleatoria, que se calcula de la siguiente manera:

$$TE = \frac{T_o + 4T_m + T_p}{6}$$

Con desviación típica,

$$\sigma = \frac{T_o - T_p}{6}$$

Tarea	T_o	T_m	T_p	TE
Investigación del estado del arte				
Investigar los conceptos básicos	20	30	40	30,0
Revisar la literatura	30	45	55	44,2
Analizar los modelos	15	20	25	20,0
Elegir el modelo	5	10	15	10,0
Seleccionar las fuentes relevantes	5	10	20	10,8
Seleccionar las tecnologías y frameworks	5	10	15	10,0
Adquisición y preprocesamiento de los datos				
Recolectar los datos	20	25	40	26,7
Anotar los datos	0	20	35	19,2
Preprocesar los datos	20	25	40	26,7
Primer entrenamiento				
Preparar la configuración del entorno	10	15	20	15,0
Entrenar el modelo por primera vez	20	30	35	29,2
Evaluar el modelo obtenido	3	5	7	5,0
Ciclo de ajustes y entrenamiento				
Analizar los errores y ajustar el modelo	3	5	10	5,5
Reentrenar con los nuevos ajustes	15	20	25	20,0
Evaluar el modelo obtenido	3	5	7	5,0
Análisis de resultados				
Comparar los entrenamientos	15	30	40	29,2
Documentar el proceso	20	25	30	25,0
Redactar la memoria	30	40	50	40,0
			Tiempo esperado total:	432,8

Tabla 3.1: Tiempo necesario para realizar las tareas, en horas

En base a los objetivos específicos descritos en 1.2, he dividido cada objetivo en tareas necesarias para alcanzarlo. En la Tabla 3.1 se muestran los tiempos estimados que corresponden a cada tarea, en horas.

1. Investigación del estado del arte. En esta primera etapa del trabajo me centro en la parte más teórica. Investigo sobre los conceptos básicos, como las cajas delimitadoras, las redes convolucionales o las métricas que miden el rendimiento de los modelos, hasta llegar a estudiar los modelos de detección y reconocimiento de objetos más recientes. También selecciono las tecnologías que voy a usar y las fuentes más relevantes que tendré en cuenta en los siguientes pasos. La mayor carga temporal reside en la revisión de la literatura, incluyendo libros y artículos del estado del arte de la identificación de señales de tráfico.

2. Adquisición y preprocesado de los datos. Busco conjuntos de datos públicos y los analizo para saber si puedo utilizarlos en el entrenamiento del modelo. También está abierta la posibilidad de encontrar imágenes de señales de tráfico y anotarlas yo mismo usando software específico. Una vez fijo mi conjunto de datos, debo preprocesarlo y dejarlo limpio para el entrenamiento.

3. Entrenamiento del modelo. Lo he dividido en dos principales partes: un primer entrenamiento del modelo que suele ser más lento debido a la configuración del entorno y la inexperiencia, y una segunda parte de ciclos de ajuste. A partir de la primera evaluación, se encuentran los posibles errores y se intentan subsanar para crear un modelo más robusto, modificando los hiperparámetros o aumentando los datos. He considerado realizar 3 ciclos de este tipo.

4. Análisis y comunicación de resultados. Se da por finalizado el entrenamiento del modelo y se analizan y comparan los distintos entrenamientos entre sí, así como los factores que han originado las distintas variaciones en los resultados. En este objetivo también se incluyen las tareas de documentar el proceso y redactar esta memoria. Estas tareas se elaborarían durante todo el tiempo de proyecto, imprescindibles para comunicar los resultados y explicar correctamente el proceso seguido.

Como se puede observar, se obtienen 432,8 horas de tiempo esperado, que da un margen de unas 12 horas respecto al tiempo reservado al proyecto para hacer frente a imprevistos.

3.2. Diagrama de Gantt

El correspondiente diagrama de Gantt permite observar de forma gráfica la distribución temporal de las tareas. El proyecto empieza el 8 de enero de 2023 y finaliza el 31 de mayo. El eje X representa las semanas, es decir, cada rectángulo coloreado corresponde a una semana. La duración de las tareas es aproximada, para facilitar su lectura en el

diagrama y no tener que partirlo en varios diagramas de Gantt más pequeños y subdivididos. Los colores simbolizan los distintos objetivos y las tareas que los componen.

En la etapa de investigación se aprecia como la tarea de seleccionar fuentes relevantes se extiende toda la duración posible, ya que es algo que se va realizando mientras se hacen el resto de tareas.

En los ciclos de entrenamiento, se han señalizado las 3 repeticiones de cada tarea.

Las tareas de documentación del proceso y redacción de la memoria, en la parte baja del diagrama, aparecen durante una gran parte de la duración del proyecto, ya que se deben realizar de forma complementaria al resto de tareas. Las últimas dos semanas se reservan a la escritura formal de esta memoria, una vez que se han finalizado todas las tareas anteriores.

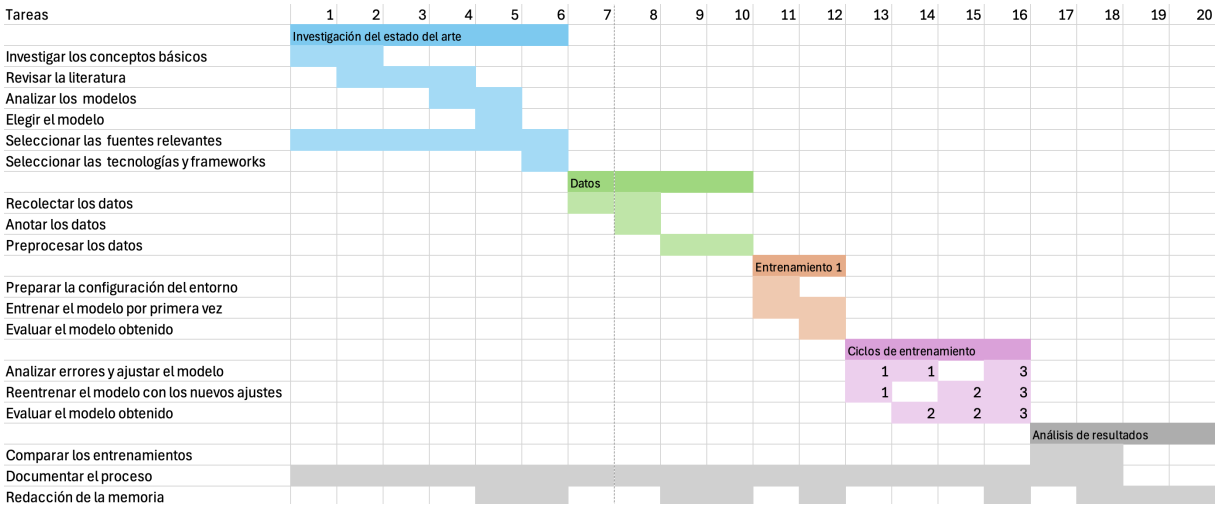


Figura 3.1: Diagrama de Gantt

3.3. Presupuesto económico

Para la estimación del presupuesto se tendrán en cuenta las herramientas utilizadas (hardware y software, con los factores de impacto que correspondan al duración del proyecto), junto con los recursos humanos necesarios, según la planificación de tareas, y el tipo de rol correspondiente a cada tarea. Para ello conviene tener una tabla actualizada del coste por hora de cada tarea del proyecto.

3.3.1. Hardware y software

Hardware

El trabajo ha sido realizado en un ordenador portátil, modelo HP Pavilion Laptop 14-ce2xxx. Posee una CPU Intel Core i7-8565U y 8 GB de memoria RAM. Su precio es 649 €.
Se estima una vida útil de 5 años y una duración del proyecto de 5 meses, por tanto, el coste del hardware utilizado ha sido de $649\ € \cdot \frac{5\ \text{meses}}{12 \cdot 5\ \text{meses}} = 54,08\ €$

Software

Todo el software utilizado ha sido gratuito, no ha tenido impacto en el presupuesto.

3.3.2. Recursos humanos

Las tareas realizadas en este proyecto caen bajo dos perfiles de trabajo: investigador y Data Scientist. La primera etapa es pura investigación, mientras que el resto son tareas propias de un Data Scientist: recopilar, analizar, limpiar y preparar los datos, elegir un modelo y entrenarlo, evaluar los resultados.
Por ello, sumando las tareas correspondientes, son 125 horas de investigador y 307,8 horas de Data Scientist. La web Glassdor [16] recoge de forma anónima los sueldos de cualquier puesto de empleo y muestra públicamente los datos. Para el puesto de investigador científico en España, muestra un horquilla de 25.000 a 45.000€ calculada a partir de 65 muestras. Escogeré la estimación más baja porque representaría a un investigador sin experiencia previa, 25.000€.
Para el puesto de Data Scientist, a partir de una muestra de 2133 sueldos recogidos, se estima un sueldo medio entre 30.000 y 45.000€. También tomaré la estimación baja, 30.000€.

Suponiendo que se trabajan de media 21 días al mes (descontando festivos y vacaciones) y 8 horas diarias, los sueldos por horas son de 12,40 €/h para el investigador y de 14,88 €/h para el Data Scientist. Finalmente,

$$12,40€/\text{h} \cdot 125\text{h} + 14,88€/\text{h} \cdot 307,8\text{h} = 6130,06€$$

Por tanto, el coste de los recursos humanos en este proyecto es de 6130,06 €.

3.3.3. Presupuesto total

Sumando los costes de los distintos elementos, se obtiene un presupuesto total de 6184,14 €.

	Hardware	Software	RRHH	Total
Estimación	54,08 €	0 €	6130,06 €	*6184,14 €*

Tabla 3.2: Presupuesto total

3.4. Balance

En esta sección se mostrarán y analizarán las diferencias entre la estimación inicial y la realidad, una vez acabado el proyecto.

3.4.1. Balance temporal

En la Tabla 3.3 se compara el tiempo estimado con el tiempo real de cada tarea. Se puede observar como se ha seguido la estimación en la mayoría de tareas, menos algunas excepciones:

- Selección de las tecnologías y frameworks. Gracias a las tareas revisar la literatura y analizar los modelos leí muchos artículos donde se detallaba cómo se implementaba cada modelo. Gracias a eso me fue fácil elegir las tecnologías a usar.
- Adquisición y preprocesamiento de los datos. Las tres tareas de este objetivo difieren con su planificación. La recolecta de datos me llevo más tiempo debido a las múltiples bases de datos existentes y la poca información disponible en internet, que me obligó a descargar varios datasets y analizarlos manualmente. La anotación de datos no fue apenas necesaria, ya que los *datasets* elegidos estaban bien anotados y solo tuve que anotar una pequeña parte. El preprocesamiento de los datos excedió mi previsión debido a las diferencias entre formatos de las distintas fuentes de imágenes y sus respectivas anotaciones. Tuve que realizar varios *scripts* en Python para convertir todo al formato que usa el modelo YOLO.
- Entrenamientos. El primer entrenamiento duró más tiempo del que creí, por problemas con el código fuente que solucioné gracias a las preguntas y ejemplos de otros usuarios. El resto de entrenamientos fueron cada vez más rápidos gracias a la experiencia que iba ganando.
- Redacción de la memoria. Esta tarea se extendió más de lo que había planificado por mi poca experiencia en documentos de este tamaño, lo que me obligó a terminarla fuera del plazo establecido.

El proyecto debería haber terminado el 31 de mayo, sin embargo, debido a la sobrecarga temporal y el periodo de exámenes finales de junio, ha sido terminado en julio, con 30,2 horas de trabajo más que las estimadas.

Tabla 3.3: Comparación de la duración estimada de cada tarea frente al tiempo real, en horas

Tarea	Tiempo real	TE
Investigación del estado del arte		
Investigar los conceptos básicos	32	30,0
Revisar la literatura	45	44,2
Analizar los modelos	17	20,0
Elegir el modelo	10	10,0
Seleccionar las fuentes relevantes	10	10,8
Seleccionar las tecnologías y frameworks	5	10,0
Adquisición y preprocesamiento de los datos		
Recolectar y analizar los datos	35	26,7
Anotar los datos	10	19,2
Preprocesar los datos	50	26,7
Primer entrenamiento		
Preparar la configuración del entorno	25	15,0
Entrenar el modelo por primera vez	40	29,2
Evaluar el modelo obtenido	6	5,0
Ciclo de ajustes y entrenamiento 1		
Analizar los errores y ajustar el modelo	5	5,5
Reentrenar con los nuevos ajustes	20	20,0
Evaluar el modelo obtenido	3	5,0
Ciclo de ajustes y entrenamiento 2		
Analizar los errores y ajustar el modelo	5	5,5
Reentrenar con los nuevos ajustes	17	20,0
Evaluar el modelo obtenido	3	5,0
Ciclo de ajustes y entrenamiento 3		
Analizar los errores y ajustar el modelo	3	5,5
Reentrenar con los nuevos ajustes	15	20,0
Evaluar el modelo obtenido	2	5,0
Análisis de resultados		
Comparar los entrenamientos	30	29,2
Documentar el proceso	25	25,0
Redactar la memoria	50	40,0
Tiempo total	**463**	**432,8**

3.4.2. Balance económico

Existen discrepencias entre la estimación y el coste real porque estos dependen exclusivamente de la duración del proyecto, que ha aumentado en 30 horas comparado a la estimación inicial y un mes más que la fecha límite.

En cuanto al hardware, el coste aumenta a 64,90 €.

En cuanto a los recursos humanos, rehaciendo los cálculos con 119 horas de investigador y 344 de científico de datos,

$$12,40€/\text{h} \cdot 119\text{h} + 14,88€/\text{h} \cdot 344\text{h} = 6594,32€$$

	Hardware	**Software**	**RRHH**	**Total**
Estimación	54,08 €	0 €	6130,06 €	*6184,14* €
Realidad	64,90 €	0 €	6594,32 €	*6659,22* €
Diferencia	10,82 €	0 €	464,26 €	**475,08** €

Tabla 3.4: Comparación del presupuesto frente al coste real

Debido a estas diferencias, el coste total final supera en 475,08€ a la estimación inicial.

Parte II

Marco teórico

Capítulo 4

Inteligencia Artificial y Aprendizaje Profundo

La inteligencia artificial se ha convertido en una de las tecnologías más impactantes del siglo XXI, afectando campos que van desde la medicina hasta la economía. El aprendizaje automático es una subdisciplina de la IA que ha demostrado ser particularmente potente, lo que ha permitido avances significativos en el reconocimiento de la voz, la visión por computadora y la generación de texto, entre otros. A lo largo de este capítulo, veremos cómo funcionan estos conceptos desde sus cimientos y, en particular, prestaremos especial atención a las redes neuronales convolucionales (CNN, *Convolutional Neural Network*), en las que se basan los sistemas de detección y reconocimiento de objetos, como el modelo descrito en la parte III de la memoria.

4.1. Inteligencia Artificial

La inteligencia artificial no es concepto que haya surgido recientemente, la idea de que existan máquinas que pudieran pensar y resolver problemas de forma similar a la que lo haría un humano se remonta siglos atrás. Sin embargo, fue en 1956 cuando se acuñó el término inteligencia artificial, en la conferencia de Dartmouth organizada por John McCarthy, Marvin Minsky, Nathaniel Rochester y Claude Shannon[35].

4.1.1. Tipos de inteligencia artificial

La **IA débil**, también conocida como IA estrecha, se refiere a sistemas diseñados para realizar tareas específicas. Estos sistemas no tienen conciencia ni comprensión general, pero pueden superar a los humanos en sus tareas específicas. Es el tipo de IA al que tenemos acceso en la actualidad. Algunos ejemplos son:

- **Vehículos autónomos:** Sistemas de conducción autónoma desarrollados por empresas como Tesla y Waymo. Estos vehículos utilizan una combinación de sensores,

algoritmos de aprendizaje profundo y datos de mapas para navegar y operar sin intervención humana.

- **Modelos del lenguaje:** Son algoritmos diseñados para generar y manipular texto en lenguaje natural, simulando una conversación. ChatGPT, desarrollado por Open AI [43], es uno de los ejemplos más avanzados y representativos de esta categoría de IA débil.

- **Sistemas de recomendación:** Algoritmos que sugieren productos o contenido en plataformas como Amazon y Netflix.

- **Sistemas especializados en juegos**. En ajedrez, la máquina Deep Blue de IBM, que ganó al campeón del mundo Garry Kasparov en 1997, es un ejemplo icónico. Actualmente, existen sistemas mucho más avanzados como Stockfish, utilizados por ajedrecistas profesionales para entrenar y mejorar sus habilidades.

La **IA fuerte**, o IA general, es un concepto más avanzado y aún meramente teórico. Se refiere a sistemas que poseen la capacidad de realizar cualquier tarea intelectual que un ser humano podría hacer. Estos sistemas tendrían una comprensión general del mundo y podrían aplicar su inteligencia a una amplia gama de problemas. Aunque la IA fuerte es un objetivo de largo plazo, aún no se ha logrado y permanece en el ámbito de la ciencia ficción y la investigación teórica.

4.2. Ramas de la IA

La inteligencia artificial abarca diversas ramas. Entre las más destacadas se encuentran: aprendizaje automático (ML, *Machine Learning*), procesamiento del lenguaje natural (NLP, *Natural Language Processing*), visión por computador (*computer vision*), robótica, y muchas otras más.

En este TFG se desarrolla un modelo que combina el campo de la visión por computador con el aprendizaje automático.

4.2.1. Aplicaciones

La inteligencia artificial se puede aplicar en casi cualquier campo donde el ser humano realice tareas «inteligentes», susceptibles de ser reemplazadas por algoritmos especializados. En la mayoría de los casos, la IA no ha avanzado lo suficiente como para sustituir completamente a profesionales, pero constituye una gran ayuda en el desempleo del trabajo, complementando a la persona.

Ejemplos a continuación.

Medicina: IA se utiliza para el diagnóstico de enfermedades, la personalización de tratamientos y la administración de medicamentos. Los sistemas de IA pueden analizar grandes

volúmenes de datos médicos y detectar patrones que pueden no ser evidentes para los médicos humanos, como en la tarea de realizar diagnósticos a partir de imágenes médicas [60].

Finanzas: Los algoritmos de IA se emplean en la detección de fraudes [25], la gestión de carteras y la negociación algorítmica. Estos sistemas pueden analizar datos financieros en tiempo real y tomar decisiones de inversión más rápidamente que los humanos.

Atención al cliente: Los *chatbots* y asistentes virtuales utilizan IA para proporcionar soporte al cliente 24/7, responder preguntas frecuentes y resolver problemas comunes. Esto mejora la eficiencia y reduce los tiempos de espera para los clientes [4].

Agricultura: La IA se aplica en la agricultura de precisión para monitorear las condiciones del suelo, optimizar el riego [1] y predecir rendimientos de cultivos. Esto ayuda a los agricultores a maximizar la producción y minimizar el uso de recursos.

4.3. Deep Learning

El *machine learning* es una rama de la IA. El propósito del *machine learning* es desarrollar y aplicar algoritmos y modelos computacionales que permitan a los sistemas automatizar el aprendizaje a partir de datos, mejorando así su capacidad para realizar tareas específicas, como predicción, clasificación, optimización o descubrimiento de patrones, sin requerir una programación explícita para cada escenario. Dentro de esta rama, existen múltiples métodos: SVM, árbol de decisión, *random forest*, k-vecinos, regresión logística, etc. Nos centraremos en las redes neuronales, de las cuales derivan las redes neuronales profundas o *deep learning*.

4.4. Redes neuronales

Las redes neuronales surgen a partir de la idea de replicar la capacidad de aprendizaje del órgano cerebral. El cerebro está compuesto por neuronas interconectadas, que se comunican entre sí gracias a impulsos eleéctricos. A grandes rasgos, cuando una neurona recibe señales de sus neuronas vecinas, si la suma de las potencias de las señales supera cierto umbral, la neurona se activa y produce una nueva señal propia, que se propaga a las siguientes neuronas. Este proceso se repite en cada neurona. Simulando esta idea, se construye el concepto de neurona artificial en 1943, [36] y más tarde de red neuronal artificial, gracias al modelo de aprendizaje de Hebb [20]. La primera implementación efectiva de una red neuronal fue el Perceptrón de Frank Rosenblatt en 1961 [48].

Una red neuronal toma como entrada un vector de longitud n, $X = (X_1, X_2, ..., X_n)$ y devuelve como salida $f(X)$, siendo la función f no lineal. La función f es ajustada mediante un entrenamiento a partir de datos conocidos, pares de la forma (X, Y) donde Y es la respuesta asociada a X. El algoritmo de entrenamiento se le conoce como retropropagación o *backpropagation*, ideado en 1986 por Rumelhart[49], que se explica en el subapartado 4.4.3.

4.4.1. Elementos de una neurona

Las elementos fundamentales de una neurona son

- **Entrada**: es un vector X de valores numéricos que representa una serie de características. Por ejemplo, si el dato de entrada es una imagen en escala de grises, estos valores pueden ser el valor de intensidad de cada píxel.

- **Salida**: valor que devuelve la neurona.

- **Pesos**: son parámetros ajustables de la red mediante el entrenamiento. Determinan la importancia de cada entrada. Se expresan como w_i, asociados a la entrada X_i.

- **Sesgo**: también es un parámetros ajustables. Están asociados a neuronas y actúa como un regulador de la función de activación. Se expresa como w_0.

- **Función sumatoria**: es una suma ponderada usando los pesos y el sesgo.

$$s(X) = w_0 + \sum_i w_i X_i$$

- **Función de activación**: es una función no lineal g que transforma el resultado de la función sumatoria. Existen diferentes función de activación que modelan el comportamiento de la neurona. Algunas de las más utilizadas son la función sigmoidea ($\frac{1}{1+e^{-x}}$), ReLU ($\max(0, x)$), Heavyside ($1_{x>0}$) o tangente hiperbólica ($\frac{e^x - e^{-x}}{e^x + e^{-x}}$).

En los términos anteriores, la salida de una neurona quedaría definida como

$$g(s(X)) = g(w_0 + \sum_i w_i X_i) \tag{4.1}$$

Una red neuronal está compuesta de capas de neuronas artificiales conectadas de forma que la salida de una neurona de la capa anterior se convierte en la entrada de alguna neurona en la siguiente capa. El cálculo señalado en la ecuación 4.1 se repite para cada neurona de la red.

4.4.2. Funcionamiento red neuronal

Para entender el funcionamiento de las redes neuronales, describiremos una red neuronal sencilla compuesta por 3 capas: una entrada, una capa oculta intermedia y una salida.

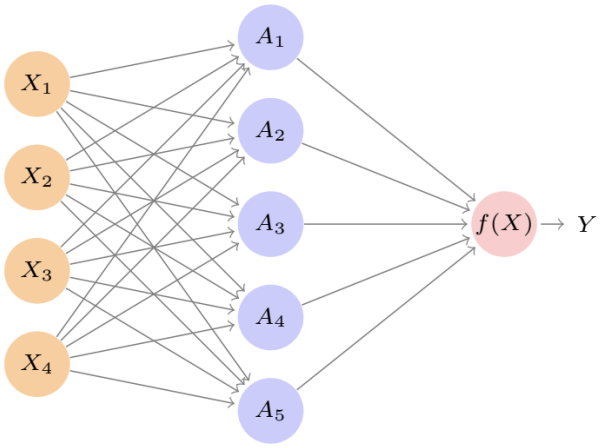


Figura 4.1: Red neuronal de una sola capa oculta. El vector de entrada X tiene 4 componentes, 5 neuronas en la capa oculta y la salida es unidimensional [24].

La salida de una red neuronal con una sola capa oculta de K neuronas tiene la forma

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{n} w_{kj} X_j).$$

g es la función de activación.
w_{ij} es el peso asociado a la neurona oculta i y la entrada X_j.
w_{i0} es el sesgo de la i-ésima neurona en la capa oculta.
β_i es el peso de la capa de salida asociado a la i-ésima neurona oculta .
β_0 es el sesgo de la capa de salida.

En la actualidad, las redes neuronales que se utilizan suelen tener más de una capa. Aunque teóricamente una sola capa con muchas neuronas serviría para aproximar la mayoría de funciones (Teorema de Aproximación Universal [21]), podemos conseguir el mismo resultado usando más capas y menor número de neuronas por capa [24].

4.4.3. Entrenamiento

El entrenamiento de una red neuronal se realiza mediante el algoritmo de **retropropagación**. Se necesitan datos ya etiquetados, donde cada dato de entrada tiene una etiqueta correspondiente. En primer lugar, el dato se introduce en la red neuronal, donde es procesado a través de sus capas para generar una salida. A continuación, la salida generada por la red se compara con la etiqueta real del dato. Esta comparación se realiza utilizando una función de error (o función de pérdida), que cuantifica la discrepancia entre la predicción de la red y el valor real.

Para reducir el error, se emplean algoritmos de optimización, como el descenso de gradiente o *Adam*. Estos algoritmos ajustan los parámetros (pesos y sesgos) de la red de manera iterativa para minimizar la función de error. El proceso de ajuste de parámetros

se repite para cada dato del conjunto de entrenamiento. Una vez que se completa una iteración completa del conjunto de datos, se denomina una época. Este ciclo de propagación hacia adelante, cálculo del error y ajuste de parámetros se repite durante múltiples épocas. El entrenamiento continúa hasta que se cumplen ciertos criterios de parada, como una reducción mínima en la función de error o un número máximo de épocas.

Si el entrenamiento se prolonga durante demasiadas épocas, existe el riesgo de sobreajuste o ***overfitting***, donde la red neuronal se ajusta excesivamente a los datos de entrenamiento y pierde la capacidad de generalizar a nuevos datos. Para prevenir el sobreajuste, se pueden utilizar técnicas como la validación cruzada, el uso de un conjunto de validación, regularización y otras estrategias de control del entrenamiento.

El entrenamiento de una red neuronal es un proceso computacionalmente muy costoso, pero existe la técnica de ***fine-tuning*** para solventar en ocasiones este problema. Es un proceso en el que una red neuronal preentrenada se entrena adicionalmente en un nuevo conjunto de datos, generalmente más pequeño, para adaptarlo a una tarea o dominio específico. Este enfoque aprovecha el conocimiento ya capturado por el modelo durante su entrenamiento inicial, permitiendo que se adapte rápidamente a nuevas tareas con relativamente menos datos y recursos computacionales.

4.5. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal artificial que buscan replicar el funcionamiento de las neuronas que se encargan de procesar la visión en el cerebro, en la corteza visual. La operación fundamental en la que se basan estas redes es la convolución de matrices.

4.5.1. Operación de convolución

La convolución es una operación entre dos matrices, una matriz de entrada y un filtro (también llamado *kernel*) que suele ser de tamaño menor. Se necesitan definir tres parámetros adicionales:

- El ***padding*** es la técnica de añadir ceros alrededor de la matriz de entrada. Esto se hace para controlar el tamaño de la salida y permitir que el filtro se aplique a los bordes de la matriz de entrada.
- El ***stride*** es el número de pasos que el filtro se desplaza sobre la matriz de entrada.
- La **profundidad** se refiere a la cantidad de filtros que se aplican a la matriz de entrada. Cada filtro produce una matriz de salida (mapa de características). Si usamos múltiples filtros, la salida tendrá una profundidad igual al número de filtros.

Para calcular el resultado, deslizamos el *kernel* sobre la matriz de entrada (la longitud del desplazamiento la marca el *stride*) y realizamos la multiplicación elemento a elemento seguida de la suma de esos productos. La salida de la convolución es una nueva matriz,

cuyo tamaño puede ser diferente a la matriz inicial, dependiendo de los valores de los parámetros *padding* y *stride*.

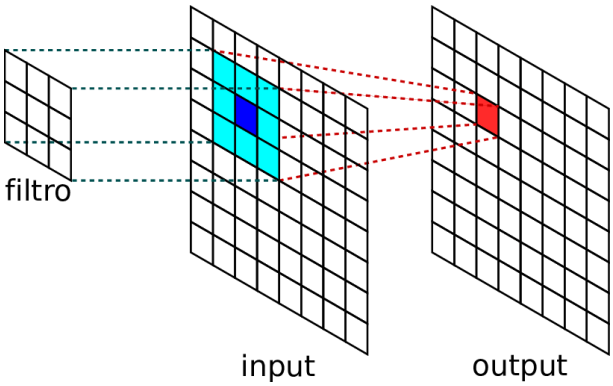


Figura 4.2: Ilustración de la aplicación de un filtro en la operación de convolución.

Veámoslo con un ejemplo. Usaremos un filtro de detección de bordes, *padding* $= 1$, *stride* $= 1$ y profundidad $= 1$. El resultado tendrá el mismo tamaño que la matriz inicial 5x5.

$$\text{Matriz de entrada} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Filtro} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\text{Resultado} = \begin{bmatrix} 4 & -2 & -1 & -2 & 4 \\ -2 & 2 & 1 & 2 & -2 \\ -1 & 1 & 0 & 1 & -1 \\ -2 & 2 & 1 & 2 & -2 \\ 4 & -2 & -1 & -2 & 4 \end{bmatrix}$$

4.5.2. Capas de una CNN

Una CNN se compone de varias capas. Los principales tipos son: capa de convolución, capa de *pooling*, capa *fully connected* y capa *softmax*.

La capa de convolución es la que se acaba de explicar.

La capa de *pooling*, también conocida como capa de reducción, tiene como objetivo reducir la dimensionalidad de la representación y hacer que la red sea más tolerante a las variaciones en la posición de los objetos dentro de la imagen. Se realiza utilizando operaciones como el máximo (*max-pooling*) o el promedio (*average-pooling*) sobre regiones pequeñas de la imagen, que quedan reducidas a ese valor.

La capa *fully connected* o densa es una capa tradicional de una red neuronal, donde todas las neuronas están conectadas entre sí.

La capa *softmax* es comúnmente utilizada como la última capa de una CNN en problemas de clasificación multiclase. Transforma las salidas de la capa anterior en probabilidades que suman uno. Cada valor de salida representa la probabilidad estimada de que la entrada pertenezca a una clase particular.

Las primeras capas de una CNN pueden detectar elementos simples como curvas y líneas, mientras que las capas más avanzadas son capaces de reconocer formas complejas como siluetas y rostros, lo que las hace altamente efectivas para resolver problemas de visión artificial, como la detección y clasificación de imágenes.

En la Figura 4.3 se muestra la arquitectura de la popular red de clasificación de imágenes AlexNet [27], compuesta por 5 capas de convolución, 3 de *max-pooling* y por último, 3 capas *fully conected*.

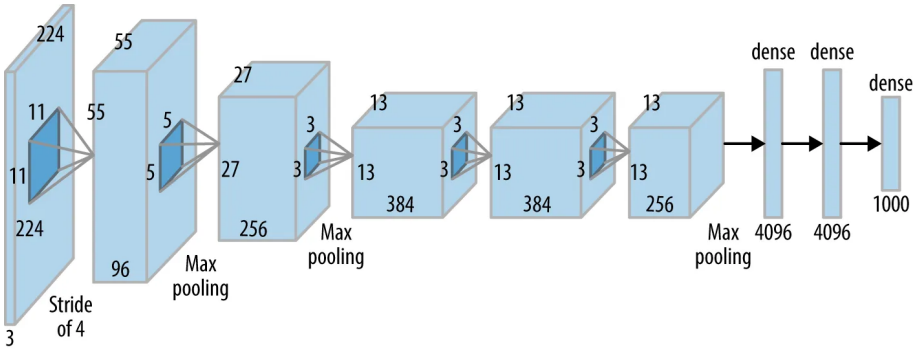


Figura 4.3: Diagrama de bloques que representa la arquitectura de la CNN AlexNet [27].

Capítulo 5

Clasificación de imágenes y detección de objetos

La clasificación de imágenes y la detección de objetos son dos de las áreas más fundamentales en el campo de la visión por ordenador. Estos procesos permiten a los sistemas automatizados identificar y localizar objetos dentro de imágenes y vídeos, habilitando aplicaciones en una amplia gama de industrias, desde la seguridad y la vigilancia hasta la conducción autónoma y el análisis médico.

En este capítulo, se explicarán las métricas más usuales y abordaremos los conceptos y métodos clave para la clasificación de imágenes y la detección de objetos. Exploraremos los modelos de clasificación, entre las que sobresalen diferentes arquitecturas de redes neuronales diseñadas para asignar etiquetas a las imágenes completas. También analizaremos los modelos de detección, que no solo clasifican los objetos en las imágenes, sino que también identifican sus ubicaciones precisas mediante el uso de cajas delimitadoras, permitiendo identificar múltiples objetos de diferentes clases dentro de una misma imagen.

5.1. Diferencias entre Clasificación y Detección

Mientras que la clasificación de imágenes se enfoca en etiquetar una imagen completa, la detección de objetos requiere identificar y localizar múltiples objetos dentro de una imagen. La clasificación de imágenes es adecuada para tareas donde solo es necesario saber qué está presente en una imagen, como la identificación de escenas o el reconocimiento facial. En contraste, la detección de objetos es esencial para aplicaciones donde es importante conocer la ubicación exacta de los objetos, como en la robótica y la navegación autónoma.

A pesar de ser campos distintos, la clasificación de imágenes puede considerarse una subdisciplina dentro de la detección de objetos. Esto se debe a que uno de los pasos fundamentales en la detección de objetos es la identificación y clasificación de los diferentes objetos presentes en una imagen. Por lo tanto, comprender y describir las técnicas y métodos de clasificación de imágenes es esencial para un enfoque integral en la detección de

objetos. Esta interrelación subraya la importancia de abordar la clasificación de imágenes en el presente trabajo, ya que forman parte del proceso integral de la detección de objetos.

5.2. Métricas de evaluación

En el contexto de la detección y reconocimiento de objetos en imágenes, es crucial evaluar la precisión y el rendimiento de los modelos, y para ello se usan diferentes métricas.

Definamos los siguientes conceptos:

- **TP** (*True Positive*). Un verdadero positivo ocurre cuando un modelo identifica correctamente la presencia de un objeto o clase que está realmente presente en la imagen.

- **FP** (*False Positive*). Un falso positivo ocurre cuando un modelo identifica incorrectamente la presencia de un objeto o clase que no está presente en la imagen. Por ejemplo, si un modelo de detección marca incorrectamente una región de una imagen como un gato cuando no hay ningún gato presente, eso se considera un falso positivo.

- **FN** (*False Negative*). Un falso negativo ocurre cuando un modelo no logra identificar la presencia de un objeto o clase que está realmente presente en la imagen. Por ejemplo, si un modelo de detección no detecta un perro en una imagen que contiene un perro, eso se considera un falso negativo.

- **TN** (*True Negative*). Aunque menos relevante en la detección de objetos, un verdadero negativo ocurre cuando un modelo correctamente identifica la ausencia de un objeto o clase que no está presente en la imagen.

En la tarea de clasificación, métricas como precisión, *recall* o F1 son usadas para caracterizar el rendimiento de los modelos. En la detección de objetos, se añaden el concepto de IoU y la métrica mAP. Estas métricas se definen a continuación, para lo cual necesitamos introducir el concepto de de matriz de confusión.

5.2.1. Matriz de confusión

La matriz de confusión es una diagrama que permite representar visualmente en una tabla el rendimiento del modelo, comparando las predicciones con las etiquetas reales.

Tabla 5.1: Matriz de Confusión

		Valores Predichos		
		Perro	Gato	Cerdo
Valores Reales	Perro	50	2	1
	Gato	4	45	3
	Cerdo	2	3	40

En la matriz de confusión de ejemplo 5.1, vemos que la diagonal principal de la matriz corresponde a los TP. El resto de valores fuera de la diagonal corresponden a los FP.

5.2.2. Precisión y Recall

La **precisión** es una métrica que mide la proporción de verdaderos positivos entre todos los resultados positivos (tanto verdaderos como falsos). En la tarea de detección se traduce en medir, dentro de los objetos reales que se pueden detectar en la imagen, cuántos se han detectado. Su fórmula es:

$$\text{Precisión} = \frac{TP}{TP + FP}$$

La métrica ***recall*** mide la proporción de verdaderos positivos entre todos los casos verdaderamente positivos (verdaderos positivos más falsos negativos). En la tarea de detección se traduce en medir, dentro de todas las predicciones hechas por el modelo, cuántas han sido acertadas. Su fórmula es:

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

5.2.3. F1

La métrica F1 combina las dos anteriores, de forma que se convierte en un indicador más fiable de la calidad de un modelo. Su fórmula es la media armónica de la precisión y el *recall*:

$$\text{F1} = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}$$

5.2.4. Intersection over Union

En primer lugar, definiremos el concepto de ***Intersection Over Union*** **(IoU)**. IoU mide la superposición entre la caja delimitadora predicha por el modelo y la caja delimitadora real del objeto en la imagen.

$$\text{IoU} = \frac{\text{Área de Intersección}}{\text{Área de Unión}}$$
$$0 \leq \text{IoU} \leq 1$$

Cuanto más se acerque a 1, la predicción es más certera, y cuando IoU= 1, la caja predicha coincide exactamente con la caja real. Ahora podemos marcar un umbral IoU que separe una predicción correcta de un falso positivo, luego IoU determina cuando se produce un TP, FP o FN. Se puede observar un ejemplo en la imagen 5.1.

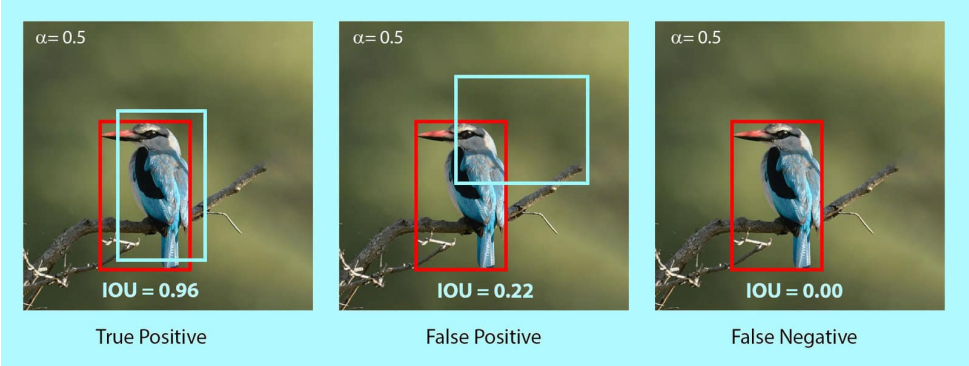


Figura 5.1: Ejemplo de TP, FP y FN bajo el umbral IoU = 0,5 [29].

5.2.5. Confianza

Un concepto importante es la confianza. Cuando los modelos realizan una predicción, en realidad asignan probabilidades a los todos los posibles resultados. Así que la confianza es la seguridad que tiene el modelo sobre el resultado que arroja, la probabilidad de que sea cierto. De esta forma nace el umbral de confianza, que limita los posibles resultados a aquellos que sobrepasen un determinado umbral, entre 0 y 1.

Si se escoge un umbral de confianza bajo, por ejemplo 0,01, se obtienen muchos positivos, ya que muchos resultados tendrán al menos un valor de confianza de 0,01. Sin embargo, de esos positivos, habrá muchos falsos positivos y verdaderos positivos, lo que ocasionará un *recall* alto y una precisión baja.
Por el otro lado, si se elige un umbral de confianza alto, con el mismo modelo habrá muchos menos positivos pero esos positivos tendrán una alta probabilidad de ser realmente verdaderos positivos, lo que ocasionaría un *recall* alto y una precisión alta.

Según esta explicación, existe una clara relación entre umbral de confianza, precisión y *recall*, que se puede expresar gráficamente en la curva Precisión-Recall o curva PR. Para cada valor del umbral de confianza c, se obtiene un par (p_c, r_c). Pintando estos puntos en una gráfica donde el eje X es la precisión y el eje Y es el *recall*, y luego interpolando los valores necesarios, se obtiene la curva PR, imprescindible en el cálculo de la métrica AP.

En la Figura 5.2 se muestra una conjunto de curvas PR de un entrenamiento de detección de señales de limitación de velocidad. La línea azul muestra la curva media de todas las clases. Se observa que las curvas PR son decrecientes, debido a la relación entre precisión y *recall* antes explicada.

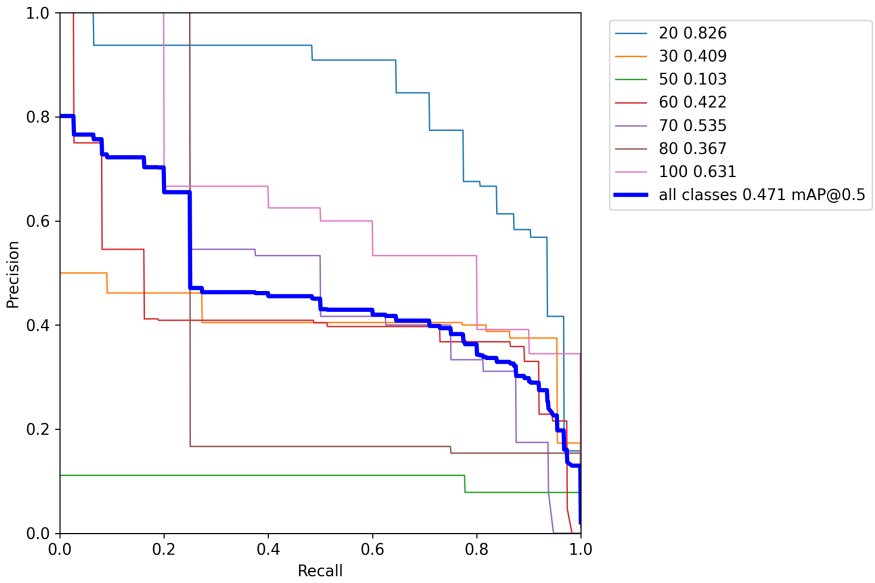


Figura 5.2: Curvas PR obtenidas de los datos de entrenamiento del modelo YOLOv7 en señales de limitación de velocidad.

5.2.6. AP y mAP

AP o *Average Precision* es una métrica calculada como el área bajo la curva PR.

$$\mathrm{AP} = \int_0^1 p(r)\, dr$$

Este valor indica un mejor rendimiento del modelo cuanto más se acerque a 1, ya que mostrará que la precisión y el *recall* están cerca del 100 %, sin importar el umbral de confianza.

Si se tienen varias clases de objetos, nos interesa obtener un valor que nos indique el rendimiento del conjunto total. La métrica **mAP** o *medium Average Precision* lo logra mediante una media aritmética de los AP de cada clase.

$$\mathrm{mAP} = \frac{1}{N} \sum_{i=1}^{N} \mathrm{AP}_i \ ,$$

donde N es el número de clases diferentes.

En la Figura 5.2, el valor mAP es igual a 0,471. Este valor indica que aún hay mucho margen de mejora, el modelo entrenado no genera buenas predicciones.
Existen varias formas de calcular mAP según los umbrales IoU. Escribir mAP@.5 representa un umbral IoU fijo de 0,5. Por otro lado mAP@.5:.95:.5 simboliza una forma diferente de calcular esta métrica. El umbral IoU varía entre 0,5 y 0,95 a un paso de 0,05. Para cada valor, se calcula el mAP y finalmente se hace la media.

5.3. Clasificación de imágenes

Con la llegada del aprendizaje profundo, las técnicas de clasificación de imágenes han experimentado un avance significativo. Las redes neuronales convolucionales han demostrado ser especialmente efectivas para esta tarea, debido a su capacidad para aprender características directamente de los datos de imagen. A partir del año 2012, gracias a la red neuronal AlexNet [27], que ganó el concurso de reconocimiento de imágenes ImageNet, se popularizó el uso de las redes neuronales convolucionales. Estas redes superaban fácilmente los resultados anteriores obtenidos por otros tipos de arquitecturas. Se han empleado las CNN en muchos campos demostrando ser la mejor alternativa en la actualidad, con aplicaciones prácticas en diversos campos como medicina [60] o reconocimiento de señales de tráfico [40].

Los modelos de clasificación de imágenes se pueden reutilizar como parte de las arquitecturas de detección de objetos. Se denomina *backbone* a la parte del modelo que extrae las características de la imagen, que después serán utilizadas por la siguiente fase del modelo. Con este rol son aplicadas las arquitecturas ResNet[19], GoogleNet[52], Inception V2, MobileNet [22] o DarkNet, entre otras [26].

5.4. Detección de objetos

Existen diversos modelos y enfoques para la detección de objetos, que se pueden clasificar principalmente en dos categorías: modelos de una fase y modelos de dos fases. Casi todos los modelos del estado del arte caen dentro del paradigma de las CNN, tal como se explica en [26].

5.4.1. Métodos de dos fases

Los modelos de dos fases abordan la tarea de detección de objetos en dos pasos consecutivos. Primero, generan propuestas de regiones candidatas que probablemente contengan objetos. Luego, en la segunda etapa, estas regiones propuestas son refinadas y clasificadas. Los modelos de dos fases tienden a ser más precisos, aunque a costa de un mayor tiempo de procesamiento. Estos modelos son preferidos en aplicaciones donde la precisión es más crítica que la velocidad.

El principal exponente de esta categoría es la familia de detectores RCNN, que nació en 2014 cuando se publicó la arquitectura RCNN (*Regions with CNN features*) [13]. Más tarde surgieron mejoras de este modelo, que aumentaron su velocidad y precisión: Fast-RCNN[12] y Faster-RCNN[47]. El modelo Mask-RCNN[18], aparte de detectar objetos, añade la posibilidad de segmentar las instancias, es decir, obtiene una máscara de píxeles que se adapta al objeto.

5.4.2. Métodos de una fase

Realizan el proceso en una sola fase, esto permite alcanzar una velocidad mucho más alta para su uso en aplicaciones en tiempo real. Los métodos de esta categoría omiten la primera fase de los RCNN que genera regiones candidatas y directamente aplican una CNN a toda la imagen, que devuelve los cuadros delimitadores y sus respectivas clases.

- YOLO (*You Only Look Once*). En 2016 surge la primera *versión* [46]. El método de YOLO consiste en dividir la imagen en una cuadrícula. Para cada celda de la cuadrícula, se dibujan cuadros delimitadores según unos cuadros ancla predefinidos (*anchors*). Se calcula una probabilidad de clase y valores de desplazamiento (*offset*) para cada uno de estos cuadros delimitadores. Entonces se seleccionan aquellos con una probabilidad que supere cierto valor umbral y todos los demás se descartan. Desde este primer modelo, se han ido desarrollando nuevas versiones de YOLO, con mejoras respecto al original. Actualmente (2024) existe hasta la versión YOLOv10, pero en el momento de comenzar este trabajo la última versión era YOLOv8.
  En la Tabla 5.2 se puede ver una comparación en un *dataset* general de detección de objetos, COCO2017[31]. En la tabla se observa que el mejor resultado es alcanzado por YOLOv7[59], seguido de Scaled-YOLOv4 y YOLOv5.

- SSD (*Single Shot Detector*) [32]. Este modelo usa una red convolucional para obtener varios mapas de características de la imagen. Cada mapa de características tiene diferentes escalas, como 38×38, luego 19×19, etc. Para cada mapa de características, se utiliza un filtro convolucional 3×3 en cada ubicación para evaluar un pequeño conjunto de *anchors*. Simultáneamente, se predice el desplazamiento del cuadro delimitador y las probabilidades de clase para cada cuadro.

- EfficientDet [54]. Modelo desarrollado por Google Research. Utiliza como *backbone* EfficientNet, conocido por su balance entre eficacia y rendimiento computacional. Se introduce una nueva estructura de pirámide de características bidireccional, que permite fusionar eficientemente características de diferentes resoluciones. Existen 7 tamaños distintos del modelo.

- RetinaNet [30]. Es un modelo desarrollado por Facebook AI Research. Introduce una nueva función de pérdida llamada *Focal Loss* que reduce el impacto de los ejemplos bien clasificados (fáciles) y pone más énfasis en los ejemplos difíciles. Esto mejora significativamente la detección de objetos pequeños y desbalanceados. Se puede combinar con diferentes *backbones*.

Tabla 5.2: Comparación de las versiones de YOLO. Proveniente del artículo [55]. La métrica reportada para YOLO y YOLOv2 fue en el *dataset* VOC2007, mientras que el resto se reportaron en COCO2017.

Versión	**Fecha**	**Anclaje**	**Framework**	**Backbone**	**AP (%)**
YOLO	2015	No	Darknet	Darknet24	63.4
YOLOv2	2016	Sí	Darknet	Darknet24	78.6
YOLOv3	2018	Sí	Darknet	Darknet53	33.0
YOLOv4	2020	Sí	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Sí	Pytorch	YOLOv5CSPDarknet	55.8
PP-YOLO	2020	Sí	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Sí	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Sí	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Sí	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	**56.8**
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9
YOLO-NAS	2023	No	Pytorch	NAS	52.2

Parte III

Desarrollo de la propuesta y resultados

Capítulo 6

Dominio de aplicación

En este capítulo se describe el contexto específico en el que se aplica el modelo de inteligencia artificial desarrollado para la detección y reconocimiento de señales de tráfico.

6.1. Contexto

El problema al que me enfrento es la identificación de señales de tráfico en imágenes. Esta es una tarea crucial en el desarrollo de ADAS y vehículos autónomos, ya que las señales de tráfico proporcionan información esencial sobre las reglas de la carretera, advertencias y restricciones que deben ser seguidas para garantizar la seguridad de los conductores, pasajeros y peatones. Un sistema capaz de identificar correctamente estas señales puede prevenir accidentes, minimizar el riesgo de errores humanos y mejorar la eficiencia del transporte.

Para lograr este objetivo, se debe construir un modelo que reconozca señales de tráfico en imágenes. No sirve con un clasificador de imágenes, ya que puede haber varias señales de tráfico en una misma imagen. Por tanto, el modelo que resuelve el problema es un detector de objetos, que localice y clasifique las señales de tráfico oportunas. Según hemos visto en la sección 5.4, existen varios modelos del estado del arte con arquitecturas basadas en CNN. Estos modelos se han aplicado en este contexto en múltiples ocasiones, como ponen de manifiesto los artículos [53], [5], [8] y muchos otros que se nombran en este capítulo y el siguiente. En el capítulo 7 se detallan los motivos de la elección de YOLOv7 entre el resto de opciones.

6.1.1. Aplicaciones comerciales actuales

En los vehículos modernos, los sistemas de detección de señales de tráfico ya se están implementando como parte de los ADAS. Estos sistemas utilizan cámaras y algoritmos de visión por computadora para identificar señales y alertar al conductor, o incluso para ajustar automáticamente la velocidad del vehículo. Ejemplos de estos sistemas incluyen el *Traffic Sign Recognition* de Tesla y el *Traffic Sign Assist* de Mercedes-Benz, entre

otros. Estos sistemas no solo mejoran la seguridad del conductor y los pasajeros, sino que también contribuyen a una conducción más eficiente y menos estresante.

6.2. Desafíos técnicos

El desarrollo de un modelo para la detección de señales de tráfico enfrenta varios desafíos:

- **Variedad de señales**: Existen muchas categorías de señales de tráfico, cada una con diferentes formas, colores y significados. Incluso existen diferentes representaciones de la misma señal en países distintos. El modelo debe ser capaz de distinguir entre estas diversas señales de manera precisa.
- **Condiciones ambientales**: Las imágenes de las señales de tráfico pueden variar significativamente debido a condiciones ambientales como la iluminación, las condiciones meteorológicas y la obstrucción parcial por otros objetos (árboles, vehículos, etc.). El modelo debe ser robusto ante estas variaciones.
- **Calidad de la imagen**: La resolución y calidad de las imágenes pueden afectar la capacidad del modelo para detectar y reconocer señales pequeñas o distantes.
- **Recursos computacionales**: Entrenar y desplegar modelos de aprendizaje profundo, como los utilizados para la detección de objetos, requiere una considerable capacidad computacional. En este proyecto, la disponibilidad de recursos es limitada, lo cual impone restricciones adicionales en la elección y entrenamiento del modelo.
- **Recolección de datos**: Para el entrenamiento de una red neuronal se necesitan grandes cantidades de datos. En este caso serían imágenes anotadas de señales de tráfico. Se pueden obtener tomando fotografías y anotando manualmente cada imagen, o recurriendo a conjuntos de datos públicos creados por otras personas.

6.3. Adaptación a los desafíos técnicos

El desafío técnico que más peso tiene en este proyecto son los recursos computaciones, ya que condiciona el tiempo de entrenamiento del modelo, y por tanto el resto de decisiones:

- **Variedad de señales**: Se restringe la variedad de señales exclusivamente a las señales de limitación de velocidad, ya que abarcar muchas clases de señales conllevaría más tiempo de computación en el entrenamiento, que con los recursos actuales es inviable.

- **Condiciones ambientales**: Se ha intentado conseguir imágenes de entrenamiento con diferente luminosidad y condiciones ambientales, sin embargo, es complicado obtener estas imágenes manualmente por la rareza de los sucesos y la poca variedad de condiciones en los conjuntos de datos públicos, así que el tiempo atmosférico más común en las imágenes de entrenamiento es soleado.
- **Calidad de la imagen**: A mayor calidad de imagen, mayor tiempo de entrenamiento. Esto es debido a que se tienen que analizar más cantidad de píxeles para encontrar los objetos a detectar. Por tanto, la resolución no podrá ser demasiado grande. Por el otro lado, tampoco conviene reducir excesivamente la resolución porque puede llegar a emborronar los objetos y convertirse en indetectables. La resolución idónea ha sido encontrada a través de los experimentos, balanceando tiempo de entrenamiento y rendimiento del modelo.
- **Recursos computacionales**: Solamente tengo acceso a un portátil, por lo que no se han usado los modelos más potentes del estado del arte, que elevarían el tiempo de entrenamiento a varias semanas.
- **Recolección de datos**: Se han usado bases de datos públicas, ya que tomar las imágenes manualmente requeriría una grandísima cantidad de tiempo. Se puede llegar a anotar manualmente algún *dataset* que identifique señales de tráfico pero que no distinga entre las diferentes clases de señales, aunque en este trabajo no ha hecho falta.

Capítulo 7

Modelo elegido: YOLOv7

En este capítulo se detallan los motivos por los que se ha elegido el modelo YOLOv7, frente al resto de alternativas mencionadas del estado del arte de la detección de objetos.

7.1. Criterios de elección

La elección del modelo de *deep learning* para este proyecto se fundamenta en varios criterios clave que se alinean con los objetivos y las limitaciones técnicas del trabajo.

1. **Precisión** del modelo. Esta precisión es habitualmente expresada con la métrica mAP, a partir de conjuntos de datos típicos en *benchmarking*, como COCO2017 [31]. Nos interesa un modelo que alcance la mayor precisión posible, teniendo en cuenta la eficiencia computacional.

2. **Eficiencia** computacional. Una de las principales limitaciones del proyecto es la disponibilidad de recursos computacionales. Se busca un modelo que posea un equilibrio óptimo entre precisión y eficiencia.

3. **Soporte** y comunidad. La disponibilidad de recursos, documentación y soporte comunitario es otro factor muy importante que facilita el desarrollo del proyecto.

4. **Código *Open Source***. Debe ser un modelo de código abierto. De esta manera, está permitido acceder, modificar y adaptar el código fuente del modelo según mis necesidades específicas.

7.2. Comparación de modelos

Todos los modelos nombrados en la sección 5.4 cumplen con el criterio de ser *Open Source*, por tanto nos fijaremos en el resto de factores clave.

7.2.1. Modelo de una o dos fases

En primer lugar, hay que elegir entre un modelo de dos fases o de una fase. Los modelos de dos fases están caracterizados por el paso previo de generación de regiones candidatas. Esta distinción provoca una notable diferencia en el tiempo de procesamiento de una imagen, como muestra [51] en los datos de la Tabla 7.1. Este artículo estudia el rendimiento de diferentes modelos de detección de objetos aplicado a un escenario sencillo de detección de señales de tráfico.

Faster R-CNN tiene un tiempo de inferencia 52 veces más lento que los modelos YOLO. El conjunto de datos en el que se probaron los modelos no era complicado y dos modelos consiguen un 100 % de mAP: Faster R-CNN y YOLOv8 nano. Sin embargo, cabe notar la diferencia de tamaño y de tiempo de computación, que nos indica que YOLO es más eficiente. Además, el modelo SSD obtiene los peores resultados.

Tabla 7.1: Comparación de precisión y tiempo de procesamiento en diferentes métodos de detección de objetos aplicadas a señales de tráfico. Datos provenientes de [51].

Método	*Backbone*	Tamaño	mAP@0.5	FPS	Tiempo de inferencia
Faster R-CNN	ResNet101	72.8 MB	**100 %**	19	**0.0528** seconds
SSD	Mobilenet-v2	9.2 MB	86 %	433	0.00023 seconds
YOLOv4	CSPDarknet53	162.2 MB	96 %	1083	0.0009 seconds
YOLOv4-Tiny	CSPDarknet53-Tiny	22.5 MB	97 %	1015	0.0009 seconds
YOLOv5 small	yolov5s	14.1 MB	99 %	1002	0.001 seconds
YOLOv8 nano	EfficientNet	6.2 MB	**100 %**	1100	**0.0009** seconds

7.2.2. Comparación de modelos de una fase

El trabajo de Zhu, Yan[65] enfrenta los modelos de una fase SSD y YOLO en la tarea de detección de señales de tráfico. Las resultados de los experimentos concluyen que YOLOv5 supera a SSD tanto en precisión como en velocidad. YOLOv5 logra un mAP de 0,98, mientras que SSD obtiene un mAP de 0.91. Además, YOLOv5 es significativamente más rápido, operando a 30 FPS, en comparación con los 3.49 FPS de SSD. Esto indica que YOLOv5 es más adecuado para el reconocimiento de señales de tráfico en tiempo real.

Rajendran, Ganapath [45] demostraron que YOLO supera a EfficientDet y RetinaNet en términos de precisión, como se muestra en la Tabla 7.2. **YOLOv4** obtiene los mejores resultados de precisión utilizando el conjunto de datos GTSDB [42] de señales de tráfico, también empleado en este proyecto, aunque en su trabajo solamente distinguen 3 superclases en vez de las 42 clases de señales.

En el artículo de Nusayer Ashik et al. [6], se eligen varias arquitecturas para un mismo conjunto de datos de señales de tráfico. Se puede observar en los resultados de la Tabla

Tabla 7.2: Comparación de modelos de detección de objetos en el *dataset* GTSDB. Datos provenientes de [45].

Modelo de Detección de Señales de Tráfico	# Parámetros (Millones)	mAP (%)	Tiempo de Inferencia Promedio (mseg)	Frames por Segundo (fps)
Mobilenet basado en Faster R-CNN	4.43	71.8	261.87	3.82
YOLOv3	61.58	92.2	101.31	9.87
RetinaNet	37.25	96.7	197	5.07
YOLOv4	64.00	**97.7**	108.85	9.18
EfficientDet-D0	3.88	24.96	25	40
EfficientDet-D1	6.63	41.33	49.2	20.33
EfficientDet-D2	8.09	60.05	72.3	13.83
EfficientDet-D3	12.02	79.68	129.5	7.72
EfficientDet-D4	20.71	85.69	234.10	4.27
EfficientDet-D5	33.65	94.85	512.80	1.95
EfficientDet-D6	51.88	96.16	770.40	1.30
EfficientDet-D0 (WBiFPN)	3.88	24.59	26.30	38.02
EfficientDet-D1 (WBiFPN)	6.63	46.25	52.20	19.16
EfficientDet-D2 (WBiFPN)	8.09	62.67	76.20	13.12
EfficientDet-D3 (WBiFPN)	12.02	76.77	136.35	7.33
EfficientDet-D4 (WBiFPN)	20.71	87.37	255.65	3.91
EfficientDet-D5 (WBiFPN)	33.65	95.3	523.45	1.91
EfficientDet-D6 (WBiFPN)	51.88	96.76	889.90	1.12

7.3 que el modelo **YOLOv7** obtiene la mayor puntuación en todas las métricas, incluso superando al modelo de dos fases Faster-RCNN.

Tabla 7.3: Comparación de resultados de los modelos usados en [6]

Modelo	mAP	Precisión	Recall	Puntuación F1
EfficientDet-D2	0.765	0.937	0.592	0.726
Faster-RCNN InceptionResNet-v2	0.751	0.765	0.634	0.724
CenterNet Hourglass-104	0.847	0.902	0.703	0.790
YOLOv5-X	0.822	0.847	0.749	0.795
YOLOv7	**0.889**	**0.939**	**0.809**	**0.869**

Por último, en el análisis de de Zhang et al. [63], volvemos a encontrar como vencedor al modelo YOLO, en este caso la versión YOLOv5, según vemos en la Tabla 7.4 domina en precisión y velocidad.

Tabla 7.4: Resultados de diferentes modelos en el *dataset* de señales de tráfico CCTSDB 2021 [63].

Método	P (%)	R (%)	mAP (%)	F1	FPS
Faster R-CNN	84.43	54.98	56.58	0.60	4.87
SSD	86.47	27.74	49.20	0.42	22.33
RetinaNet	86.70	52.88	57.78	0.65	8.88
YOLOv3	84.63	42.71	50.48	0.54	20.34
Libra R-CNN	83.72	60.04	61.35	0.70	8.81
YOLOv4	76.16	52.50	51.69	0.59	16.55
Dynamic R-CNN	86.98	58.33	60.01	0.69	9.03
Sparse R-CNN	94.12	52.58	59.65	0.67	8.45
YOLOv5	90.80	69.20	**76.30**	**0.78**	**123.46**

Por los resultados previamente mostrados, obtenidos de diferentes fuentes, concluyo que el mejor modelo para la tarea de detección de señales de tráfico sujeto a las restricciones de este proyecto es la serie YOLO. Dentro de las versiones disponibles en el momento de realizar el trabajo (hasta la versión 8) me decanto por la versión **YOLOv7**, apoyado en los resultados de la Tabla 5.2 donde se comparan todas las versiones de YOLO en un mismo *dataset*. La versión YOLOv8 también ha demostrado conseguir muy buenos resultados [51]. Sin embargo, en el momento de realizar este proyecto, su lanzamiento era reciente, lo que implicaba una menor disponibilidad de soporte y una comunidad menos desarrollada en comparación con la versión elegida.

7.3. YOLOv7

La arquitectura de YOLOv7 es muy compleja y no corresponde a este trabajo describirla en profundidad. Nos limitaremos a resaltar que es una versión mejorada de YOLOv4 [7], ya que esa versión constituyó la base sobre la que se construyó YOLOv7, añadiendo nuevos métodos y capas, además de optimizando los procesos ya existentes.

YOLOv7 tiene varios tamaños disponibles, en función del número de parámetros del modelo, mostrados en la Tabla 7.5. Están separados en dos grupos con una arquitectura ligeramente diferente. El primero compuesto por YOLOV7-tiny, YOLOv7 y YOLOv7-X y el segundo por YOLOv7-W6, YOLOv7-E6, YOLOv7-D6, YOLOv7-E6E. La principal diferencia radica en que el segundo grupo tiene capas añadidas para detectar detalles más pequeños, con la desventaja de ser más pesados computacionalmente.

La tabla presenta varias características de las variantes de YOLOv7:

- **Parámetros (M)** representa el número de parámetros en millones que tiene cada modelo. Los parámetros son los valores que el modelo ajusta durante el proceso de entrenamiento, los pesos y sesgos.
- **GFLOPs** significa *Giga Floating Point Operations per second*, es decir, miles de millones de operaciones de punto flotante por segundo. Es una medida del costo computacional del modelo. Un mayor número de GFLOPs implica que el modelo requiere más capacidad de procesamiento para realizar predicciones.
- **Tamaño de Entrada** se refiere a las dimensiones en píxeles de las imágenes de entrada cuadradas con las que se han calculado las GFLOPs. En los modelos del segundo grupo se han utilizado imágenes de 1280x1280 píxeles en vez 640x640 píxeles porque están diseñados para captar detalles de imágenes más grandes.

Tabla 7.5: Tamaños de los modelos YOLOv7. Datos de [59].

Modelo	Parámetros (M)	GFLOPs	Tamaño de Entrada
YOLOv7-tiny	6.2	13.8	640
YOLOv7	36.9	104.7	640
YOLOv7-X	71.3	189.9	640
YOLOv7-W6	70.4	360.0	1280
YOLOv7-E6	97.2	515.2	1280
YOLOv7-D6	154.7	806.8	1280
YOLOv7-E6E	151.7	843.2	1280

Debido a las restricciones de potencia computacional del proyecto, se utilizan los modelos YOLOv7-tiny y YOLOv7.

Capítulo 8

Conjuntos de datos

En este capítulo se describen los *datasets* que se han utilizado en el entrenamiento del modelo, las fuentes de donde se han obtenido y el preprocesado aplicado.

8.1. Conjuntos de datos originales

En primer lugar, se describirán las fuentes de los conjuntos de datos y después cómo se han manipulado para formar los *datasets* usados.

Existen varios *datasets* públicos de detección de señales de tráfico, entre los cuales he elegido los más convenientes para mi objetivo, después de realizar una comparación.
He excluido los siguientes:

- *Mapillary Traffic Sign Dataset* [10], 42.000 imágenes de resolución muy grande (3000 y 4000 píxeles) que no permitían realizar una reducción de resolución debido al pequeño tamaño de las señales contenidas.
- *Belgium Traffic Sign Dataset* (BTSD) [34], 8.000 imágenes donde todas las señales de límite de velocidad pertenecían a la misma clase, haría falta retocar todas las anotaciones manualmente.
- *The Laboratory for Intelligent and Safe Automobiles* (LISA) *Dataset* [41], una parte de las imágenes del conjunto estaban en blanco y negro.
- *The Mapping and Assessing the State of Traffic Infrastructure* (MASTIF) [50], 3000 imágenes donde todas las señales de límite de velocidad pertenecían a la misma clase.

Los datasets que sí elegí para este trabajo son los expuestos a continuación.

8.1.1. GTSDB

El conjunto de datos GTSDB (*German Traffic Sign Detection Benchmark*) [42], consta de 900 imágenes captadas en Alemania destinadas al *benchmarking* de modelos de

detección de objetos. Está dividido en 600 imágenes para el conjunto de entrenamiento y 300 imágenes para el conjunto de validación. Posee 43 clases de señales distintas, entre ellas las limitaciones de velocidad de 20, 30, 50, 60, 70, 80, 100, 120 km/h. La resolución de las imágenes es de 1360x800 píxeles y están en el formato de imagen `.ppm` (*Portable Pixmap Format*). Posee señales que sufren oclusión, daños y mala iluminación, por lo que constituye un *dataset* muy variado, especialmente útil para entrenar un modelo robusto.

Las anotaciones que acompañan a cada imagen se encuentran en un archivo `.txt` donde cada línea tiene el mismo formato: separados por punto y coma (`;`) se encuentra el nombre del archivo de la imagen, 4 coordenadas que definen el cuadro delimitador (columna izquierda, fila superior, columna derecha y fila inferior, respectivamente) y el ID de la clase a la que pertenece la señal. Las clases asociadas a cada ID se describen en un archivo `ReadMe.txt`.

Figura 8.1: Imagen de ejemplo del *dataset* GTSDB [42].

Por ejemplo la anotación de la imagen de la Figura 8.1 es :

`00012.ppm;979;364;1034;418;1`

El nombre de archivo es `00012.ppm`, el cuadro en el que se encuentra la señal está definido por las coordenadas en el eje X (`979`, `1034`) y en el eje Y (`364`, `418`). La señal tiene el ID `1`, por lo que corresponde a la clase del límite de velocidad 30.

8.1.2. DFG

El conjunto de datos DFG[53] consta de 6957 imágenes capturadas en diversas localizaciones de Eslovenia, dividido en 5254 para el conjunto de entrenamiento y 1703 para el conjunto de validación, de forma que se mantienen las proporciones de cada clase en cada conjunto. Las imágenes tienen una resolución de 1920x1080 píxeles. El conjunto cubre 200 clases de señales distintas, cada una de ellas con un mínimo de 20 instancias.

Entre las clases de señales se incluyen varias limitaciones de velocidad: 10, 30, 40, 50, 60 y 70 km/h. Las anotaciones que acompañan a las imágenes están estructuradas en un formato de diccionario `json`, donde cada anotación de una señal tiene su propio objeto con múltiples campos, como un ID de la señal, el área del cuadro delimitador, sus coordenadas, una máscara de segmentación de la señal, un ID de clase y un ID de imagen.

En la Figura 8.3 se encuentra la anotación de la imagen mostrada en la Figura 8.2, a modo de ejemplo. El ID de la imagen es el `556` y están anotadas 3 señales, dos en primer plano y la tercera al fondo.

Figura 8.2: Imagen de muestra del *dataset* DFG [53].

8.1.3. TT100K

El conjunto de datos de señales de tráfico chinas TT100K (Tsinghua-Tencent 100K) [66] consta de 10,000 imágenes que contienen señales y 90,000 imágenes sin señales. Las imágenes tienen una resolución de 2048x2048 píxeles. Este conjunto de datos incluye un total de 201 clases de señales distintas, aunque solo 45 clases cuentan con más de 100 instancias.

Entre las clases de señales se incluyen prácticamente todas las limitaciones de velocidad posibles: 5, 10, 15, 20, 25, 30, 35, 40, 50, 60, 65, 70, 80, 90, 100, 110, y 120 km/h. Las anotaciones que acompañan a cada imagen están estructuradas en formato `json`. En la Figura 8.5 se describe el formato de una anotación, asociada a la imagen 8.4. En el objeto se encuentra la ruta de la imagen, su ID y un array con las señales contenidas, caracterizadas por las coordenadas del cuadro delimitador y la clase de la señal. El significado de las clases está explicado en un archivo PDF perteneciente al *dataset*.

```
[
  {
    "id": 867,
    "area": 11978,
    "bbox": [1615, 235, 106, 113],
    "category_id": 52,
    "segmentation": [
     [1677, 236, 1688, 239, 1698, 244, 1706, 251, 1712, 260, 1717, 269,
        1720, 279, 1721, 290]
    ],
    "image_id": 556,
    "ignore": false,
    "iscrowd": 0
  },
  {
    "id": 868,
    "area": 3772,
    "bbox": [19, 114, 46, 82],
    "category_id": 60,
    "segmentation": [
    [34, 114, 34, 114, 42, 116, 49, 132, 60, 138, 61, 142, 63, 171,64,
       177, 62, 185, 57, 192, 50, 196, 42, 195, 35, 168, 21, 164, 20,
       159, 20, 155, 19, 140, 20, 133, 22, 126]
    ],
    "image_id": 556,
    "ignore": false,
    "iscrowd": 0
  },
  {
    "id": 869,
    "area": 24492,
    "bbox": [1592, 84, 157, 156],
    "category_id": 22,
    "segmentation": [
      [1592, 223, 1680, 84, 1749, 240, 1592, 223]
    ],
    "image_id": 556,
    "ignore": false,
    "iscrowd": 0
  }
]
```


Figura 8.3: Anotación JSON asociada a la imagen de muestra de la Figura 8.2 del *dataset* DFG [53].

Figura 8.4: Imagen de muestra del dataset TT100K [66].


```
{
  "path": "test/1012.jpg",
  "id": 1012,
  "objects": [
    {"bbox": {"xmin": 843.0, "ymin": 487.0, "xmax": 938.0, "ymax": 572.0},
        "category": "p26"},
    {"bbox": {"xmin": 529.0, "ymin": 847.0, "xmax": 621.0, "ymax": 941.0},
        "category": "p9"},
    {"bbox": {"xmin": 531.0, "ymin": 752.0, "xmax": 618.0, "ymax": 840.0},
        "category": "ph4.5"},
    {"bbox": {"xmin": 645.0, "ymin": 516.0, "xmax": 735.0, "ymax": 603.0},
        "category": "pl30"},
    {"bbox": {"xmin": 742.0, "ymin": 512.0, "xmax": 837.0, "ymax": 600.0},
        "category": "pm20"}
  ]
}
```


Figura 8.5: Anotación JSON del dataset TT100K [66] asociada a la imagen de ejemplo mostrada en la Figura 8.4.

8.2. Reducción y combinación de los conjuntos de datos

Los conjuntos de datos utilizados en el entrenamiento del modelo no son los originales, sino que han sido modificados y combinados para adaptarse al objetivo específico de este trabajo. En particular, la modificación común realizada en los tres conjuntos de datos previamente explicados es la siguiente: se han restringido únicamente a incluir las señales de tráfico que indican limitaciones de velocidad.

8.2.1. *Dataset* 1

El que llamaré a partir de ahora *Dataset* 1, es simplemente una filtración de clases del conjunto GTSDB, reduciéndolo a las señales de limitación de velocidad. Como se puede observar en la Figura 8.6, se ha eliminado la clase de velocidad 20 porque tenía muy pocas instancias.

Este conjunto de datos está dividido en entrenamiento y validación según el *dataset* original, preservando la proporción de clases. En concreto, 214 imágenes de entrenamiento y 102 de validación.

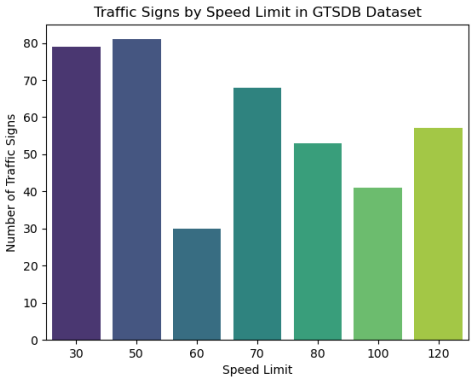


Figura 8.6: Distribución de clases en *Dataset* 1.

8.2.2. *Dataset* 2

El *Dataset* 2 es una amplicación del *Dataset* 1. Se añaden todas las señales de límite de velocidad del conjunto esloveno DFG, excepto las de límite 10, por las pocas instancias disponibles. Por tanto, como se observa en la Figura 8.7, han aumentado las instancias de las clases 30, 40, 50, 60 y 70.

El conjunto de entrenamiento tiene 487 imágenes y el conjunto de validación, 235 imágenes. Las nuevas imágenes se han añadido a estos conjuntos siguiendo la misma división que en el dataset DFG original.

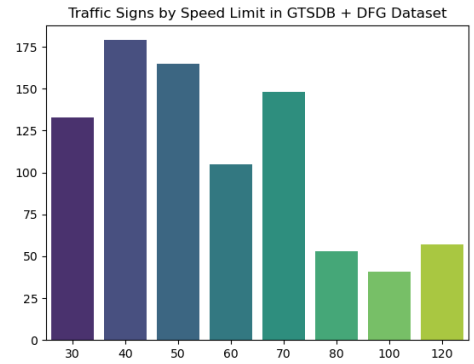


Figura 8.7: Distribución de clases en *Dataset* 2.

8.2.3. *Dataset* 3

El *Dataset* 3 amplía al *Dataset* 2. Su propósito es solucionar el problema de desbalance de las clases del conjunto, añadiendo imágenes de señales de las clases menos representadas: 60, 80, 100, 120. Se toman imágenes del conjunto de datos chino TT100K. Las imágenes incluidas en este conjunto de datos tienen una resolución más alta (2048x2048). Esto podría causar, al reducir la resolución durante el entrenamiento del modelo, que las señales presentes en las imágenes de 2048x2048 píxeles se vuelvan demasiado pequeñas, dificultando su detección por el modelo. Por esta razón, se ha tenido en cuenta el tamaño de las señales en las imágenes escogidas, filtrando por aquellas cuya área de la caja delimitadora supere cierto umbral (64x64). Con las imágenes añadidas, el conjunto de entrenamiento tiene 712 imágenes y el conjunto de validación, 367.

En la Figura 8.8, vemos como todas las clases superan las 120 instancias, lo que debería traducirse en un mejor rendimiento del modelo entrenado.

Además, para evaluar en el impacto del color en la tarea de detección de señales, se ha creado otro dataset con las mismas imágenes pero con el color cambiado a escala de grises. En la sección 8.3.3 se explica el preprocesado aplicado.

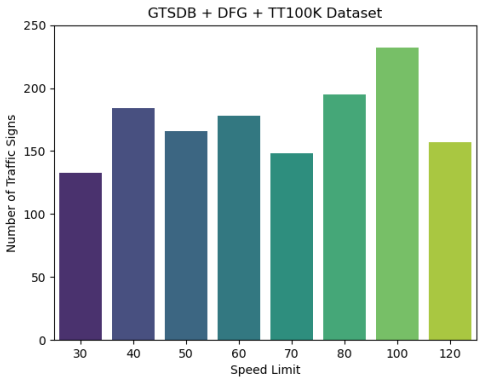
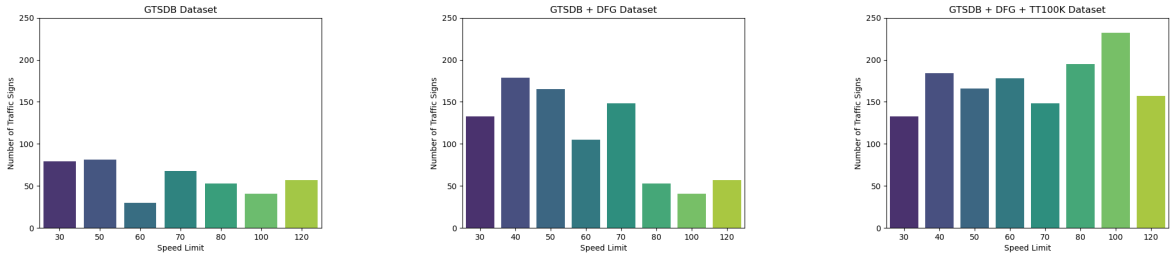


Figura 8.8: Distribución de clases en *Dataset* 3.



(a) *Dataset* 1 (b) *Dataset* 2 (c) *Dataset* 3

Figura 8.9: Crecimiento del número de señales por clase en cada *Dataset*.

8.3. Preprocesado

8.3.1. Formato de anotación YOLO

El modelo YOLO solamente admite datos de entrenamiento en un formato específico. Cada imagen debe tener asociado un archivo de anotaciones con el mismo nombre, simplemente cambiando la extensión a `.txt`. Cada fila del archivo `.txt` debe corresponder a una señal, con el siguiente formato:

`<clase> <centroX> <centroY> <ancho> <alto>`

Donde `<clase>` es el identificador de la clase del objeto..
`<centroX>` es la coordenada x normalizada del centro del cuadro delimitador.
`<centroY>` es coordenada y normalizada del centro del cuadro delimitador.
`<ancho>` es el ancho del cuadro delimitador, normalizado.
`<alto>` es el alto del cuadro delimitador, normalizado.

La normalización produce un valor entre 0 y 1. Los cuadros delimitadores de GTSDB, DFG y TT100K están definidos por los extremos en las coordenadas x e y. Esto obliga a realizar una transformación en las coordenadas para cumplir con el formato de anotación YOLO.
Las variables w y h corresponden a las dimensiones ancho y alto de la imagen.

$$x_{\text{centro}} = \frac{x_{\min} + x_{\max}}{2w}$$

$$y_{\text{centro}} = \frac{y_{\min} + y_{\max}}{2h}$$

$$\text{ancho} = \frac{x_{\max} - x_{\min}}{w}$$

$$\text{alto} = \frac{y_{\max} - y_{\min}}{\text{h}}$$

Una vez obtenidas estas variables, solo falta el número identificador de clase. En el formato YOLO, las clases deben estar numeradas a partir del 0, sin saltos.

Para lograr la conversión de la clase original al identificador en formato YOLO, se debe crear un diccionario con las clases que nos interesan, haciendo corresponder los IDs en YOLO con los IDs particulares utilizados en cada *dataset* de partida, es decir, `<YOLO_ID : DATASET_TARGET_ID>`. Por ejemplo, en el *Dataset* 1 (GTSDB), las clases *target* que queremos buscar son:

1 = speed limit 30,
2 = speed limit 50,
3 = speed limit 60,
4 = speed limit 70,
5 = speed limit 80,
7 = speed limit 100,
8 = speed limit 120.

Por tanto, el diccionario buscado sería `{ 0: 1, 1: 2, 2: 3, 3: 4, 4: 5, 5: 7, 6: 8 }`.

Finalmente, la anotación de la imagen mostrada en la Figura 8.1 realizando estas transformaciones, quedaría como:

```
0 0.740074 0.48875 0.040441 0.0675
```

Para aplicar este cambio de formato de anotación a los *datasets* enteros, me serví del lenguaje de programación Python, con el que automaticé este proceso.

8.3.2. Proceso de cambio de formato

Para cada *dataset* escribí un *script* distinto, pero en esencia seguí los mismos pasos en los *datasets* GTSDB, DFG y TT100K.

1. Identificar la estructura de las anotaciones del *dataset*.
2. Realizar un diccionario con el nombre que tienen las clases objetivo en este *dataset*.
3. Filtrar las imágenes que tienen al menos una señal de limitación de velocidad que nos interesa, es decir que pertenecen a las clases *target*: 30, 40, 50, 60, 70, 80, 100, 120.
4. Copiar la imagen a la nueva carpeta de imágenes.
5. Filtrar, dentro de la anotación asociada a la imagen, las señales pertenecientes a las clases objetivo.
6. Encontrar los datos pertenecientes a la clase y la ubicación de la caja delimitadora.

7. Realizar las conversiones de formato de la clase y las coordenadas de la caja delimitadora.

8. Escribir el nuevo archivo de anotaciones con el formato YOLO, en la carpeta nueva.

Hay que tener en cuenta que se debe realizar la separación del *dataset* en carpeta de entrenamiento y carpeta de validación. Así que las rutas de las nuevas imágenes y anotaciones deben cambiar en base a esto.

Para realizar esta tarea en Python, utilicé varias librerías.

- `os`: para navegar por las diferentes carpetas de anotación e imágenes, usé la función `os.path.join()` para unir nombres de carpeta con nombres de archivos, la función `os.listdir()` para listar todos los archivos de una carpeta o la función `os.mkdir()` para crear nuevas carpetas.

- `json`: para leer los archivos `.json` donde venían las anotaciones de los conjuntos de datos DFG y TT100K.

- `shutil`: para usar la función `shutil.copy()` con el objetivo de copiar imágenes de una carpeta a otra.

Como mención aparte, para realizar gráficas que ilustraran las distribuciones de clases de los *datasets*, usé las líbrerías `matplotlib` y `seaborn`.

Todos los scripts escritos en lenguaje Python con el fin de automatizar este proceso y generar gráficas, se distribuyen en el contenido adicional que acompaña a esta memoria.

8.3.3. *Dataset* 3 en escala de grises

Para convertir todas las imágenes del *Dataset* 3 a imágenes en escala de grises, se ha usado la librería `PIL`. Se han producido dos cambios, la conversión a escala de grises y un aumento del contraste. Un mayor contraste puede ayudar a diferenciar mejor los objetos del fondo, enfatizando los bordes y contornos, facilitando al modelo la tarea de identificar y localizar los objetos de interés.

Igual que en el proceso anterior, se ha usado la función `os.listdir()` para listar todos los archivos de las carpetas de entrenamiento y validación. Después, se han utilizado funciones de la librería `PIL` para convertir la imagen a escala de grises en un solo canal de color, aumentar el contraste y guardar la imagen en formato `.jpg`, para que así pueda usarla el modelo YOLOv7, que no está diseñado para un solo canal de color. El código de la conversión es el siguiente:

```
import os
from PIL import Image, ImageEnhance

def convert_images_to_grayscale(folder_path, folder_new):
    file_list = os.listdir(folder_path)
```

```
# Iterar sobre cada archivo de la carpeta
for file_name in file_list:
    # Checkear que el archivo corresponde a una imagen
    if file_name.endswith(('.jpg', '.jpeg', '.png', '.bmp')):
        # Abrir la imagen
        image_path = os.path.join(folder_path, file_name)
        image = Image.open(image_path)

        # Convertir a gris
        grayscale_image = image.convert('L')

        # Aumentar el contraste
        enhancer = ImageEnhance.Contrast(grayscale_image)
        enhanced_image = enhancer.enhance(1.5)

        # Guardar la imagen con la extension .jpg
        new_file_name = os.path.splitext(file_name)[0] + '.jpg'
        new_image_path = os.path.join(folder_new, new_file_name)
        enhanced_image.save(new_image_path)

        # Cerrar la imagen
        image.close()
```

Capítulo 9

Entrenamiento

En este capítulo se detalla el proceso de entrenamiento del modelo y los diferentes entrenamientos que se han llevado a cabo.

Para realizar el entrenamiento *fine-tuning* de YOLOv7 mediante las herramientas proporcionadas en el repositorio oficial [58], se debe hacer uso de varios archivos.

1. En primer lugar, hay que instalar las librerías que se encuentran en el archivo `requirements.txt`, necesarias para que funcionen los *scripts* del repositorio. En la consola de comandos, escribimos

   ```
   pip install -r requirements.txt
   ```

2. En segundo lugar, hay que crear un archivo en formato YAML indicando las rutas del *dataset* dividido en carpetas de entrenamiento y validación. Ambas deben estar divididas a su vez en una carpeta `labels` con las anotaciones y otra carpeta `images` con las imágenes. También se debe escribir en el archivo YAML el número total de clases y los nombres asociados a cada clase. Se muestra un ejemplo en el Listado 9.1.

Listing 9.1: archivo YAML de datos del *Dataset* 1.

```
train: ./GTSDB/soloVelocidad/train
val: ./GTSDB/soloVelocidad/val

# number of classes
nc: 8

# class names
names: [
  '20', '30', '50', '60', '70', '80', '100', '120'
  ]
```

3. En tercer lugar, hay que crear otro archivo en formato YAML donde se indican los hiperparámetros del modelo y las posibilidades de *data augmentation*, que se explican en la sección 9.1 y 9.2.

4. En cuarto lugar, modificar el archivo de configuración del modelo de tamaño correspondiente, que se encuentra en la ruta `cfg/training`. En este proyecto se han usado los tamaños *tiny* (YOLOv7-tiny) y *medium* (YOLOv7), así que se usarán los archivos `cfg/training/yolov7-tiny.yaml` y `cfg/training/yolov7.yaml`. La única modificación que hay que realizar es la variable del número de clases, en la primera línea. Siguiendo con el ejemplo del *Dataset* 1, la línea de este archivo YAML cambiada sería:

```
nc: 8  # number of classes
```

5. Por último, hay que ejecutar el archivo `train.py` mediante la consola de comandos del sistema, donde añadiremos como argumentos las rutas de los dos archivos anteriores, además de otras variables como la resolución de imagen o los pesos iniciales del modelo, explicados en la siguiente sección.

9.1. Hiperparámetros

Se denomina hiperparámetro a todo aquel parámetro que se utiliza para controlar el proceso de entrenamiento y que se define antes de comenzar el entrenamiento. Es importante diferenciarlos de los parámetros que se actualizan durante el entrenamiento, es decir, los pesos del modelo.

La modificación de hiperparámetros en YOLOv7 se produce desde 2 lugares diferentes: los argumentos que toma como entrada la ejecución del archivo `train.py` y las variables del archivo YAML de hiperparámetos.

Empecemos explicando los argumentos más importantes de `train.py`, el resto se pueden consultar en el código del archivo.

- `--weights`: Ruta del archivo de pesos iniciales del modelo, en formato `.pt`. Este archivo contiene los pesos preentrenados en el *dataset* COCO2017. Utilizaremos la versión *tiny* y *medium*, que deben ser descargados del directorio de GitHub, en la sección *Releases*.

- `--cfg`: Ruta del archivo de configuración del modelo (en formato `.yaml`). Este archivo define la arquitectura del modelo.

- `--data`: Ruta del archivo de configuración de datos (en formato `.yaml`). Este archivo contiene la información sobre el *dataset*, tal como se ha mostrado en el Listado 9.1.

- `--hyp`: Ruta del archivo de hiperparámetros (en formato `.yaml`). Este archivo especifica los hiperparámetros que se utilizarán durante el entrenamiento y se define el *data augmentation*.

- `--epochs`: Número de épocas para entrenar. Una época corresponde a un ciclo completo por el conjunto de datos de entrenamiento. Si son demasiadas, provoca *overfitting*.
- `--batch-size`: Tamaño del lote. Define cuántas muestras se procesan juntas antes de actualizar los pesos del modelo. Suele ser recomendable aumentar esta variable hasta alcanzar el límite de la capacidad del ordenador, marcado por la memoria RAM o GPU, dependiendo del hardware usado.
- `--img-size`: Tamaños de las imágenes de entrenamiento y prueba. Se especifican dos valores: uno para el entrenamiento y otro para la prueba, lo que permite utilizar resoluciones diferentes en cada fase.
- `--rect`: Habilita el entrenamiento rectangular. Mantiene la relación de aspecto original de las imágenes.
- `--multi-scale`: Varía el tamaño de la imagen en $\pm 50\,\%$.
- `--single-cls`: Entrena datos multiclase como una sola clase. Esto simplifica el problema de clasificación a un solo tipo de objeto.
- `--adam`: Usa el optimizador Adam en vez de SGD (*Stochastic Gradient Descent*) . Adam es un optimizador adaptativo que ajusta las tasas de aprendizaje individuales para cada parámetro, lo que puede mejorar la convergencia.
- `--freeze`: Congela capas específicas del modelo. Impide que ciertas capas del modelo se actualicen durante el entrenamiento.
- `--device`: Especifica el hardware en el que se realizará el entrenamiento, GPU o CPU.
- `--workers`: Determina cuántos subprocesos se utilizan para cargar y preprocesar los datos durante el entrenamiento.

Por otro lado, en el archivo YAML del argumento `--hyp` se modifican los siguientes hiperparámetros:

- `lr0`: Tasa de aprendizaje (*learning rate*) inicial, que determina la magnitud de los ajustes que el modelo realiza en los pesos con cada paso de optimización. Una tasa de aprendizaje demasiado alta puede hacer que el modelo no converja, mientras que una tasa demasiado baja puede hacer que el entrenamiento sea extremadamente lento.
- `lrf`: Factor que reduce la tasa de aprendizaje al final del ciclo de entrenamiento utilizando la estrategia OneCycleLR. Durante el entrenamiento, la tasa de aprendizaje se ajusta de forma dinámica: empieza aumentando, alcanza un valor máximo y luego disminuye hacia el final. Esto ayuda a que el modelo aprenda mejor y evita que se quede atrapado en mínimos locales. La tasa de aprendizaje final es `lr0*lrf`.

- `momentum`: Parámetro que ayuda a acelerar el entrenamiento en la dirección de los gradientes y suavizar las oscilaciones. Para SGD, este valor representa el momento, mientras que para Adam, representa el parámetro *beta1*, que controla la tasa de decaimiento exponencial de la media de los gradientes pasados. Es un valor entre 0 y 1.
- `weight_decay`: Término de regularización que penaliza los valores grandes de los pesos del modelo, ayudando a prevenir el *overfitting*.
- `warmup_epochs`: Número de épocas al inicio del entrenamiento en las cuales la tasa de aprendizaje se incrementa gradualmente desde un valor muy bajo hasta `lr0`. Esto ayuda a estabilizar el entrenamiento en las primeras etapas.
- `warmup_momentum`: Valor inicial del momento durante el período de calentamiento. Proporciona un arranque suave en el entrenamiento para estabilizar la convergencia.
- `warmup_bias_lr`: Tasa de aprendizaje inicial para los sesgos durante el calentamiento. Los sesgos pueden requerir una tasa de aprendizaje diferente a los pesos normales.
- `box`: Factor de ponderación para la función pérdida de las cajas delimitadoras. Ajusta la importancia de la precisión en la localización de los objetos durante el entrenamiento.
- `cls`: Factor de ponderación para la función pérdida de clasificación, ajustando la importancia de clasificar correctamente los objetos detectados.
- `cls_pw`: Peso positivo para la pérdida de clasificación *BCELoss* (*Binary Cross-Entropy Loss*). Ajusta la importancia relativa de las muestras positivas en la función de pérdida de clasificación.
- `obj`: Factor de ponderación para la función pérdida de objeto, ajustando la importancia de detectar correctamente los objetos presentes en la imagen.
- `obj_pw`: Peso positivo para la pérdida de objeto *BCELoss*. Ajusta la importancia relativa de las muestras positivas en la función de pérdida de objeto.
- `iou_t`: Umbral IoU para considerar una predicción como positiva durante el entrenamiento.
- `fl_gamma`: Parámetro *gamma* para la pérdida focal. La pérdida focal ajusta la contribución de las muestras difíciles y fáciles en la función de pérdida, mejorando el rendimiento en casos desbalanceados.
- `loss_ota`: Valor booleano indicador del uso de *ComputeLossOTA*. Esta es una técnica avanzada en el cálculo de la función perdida. Usar esta opción puede proporcionar una mejor optimización a costa de un tiempo de entrenamiento más largo.

La mayoría de los hiperparámetros de esta última lista no han sido modificados respecto a los valores por defecto proporcionados en el archivo `data/hyp.scratch.custom.yaml`, creado específicamente para *datasets* personalizados. Esto se debe a que dichos valores han sido establecidos basándose en un conocimiento experto y experiencia previa. Ajustarlo en base a mis *datasets* requerirían un estudio intensivo y ese nivel de optimización no es el objetivo de este TFG. Por tanto, he optado por conservar los valores propuestos para asegurar la efectividad del entrenamiento del modelo, aunqeu no sea óptimo. Únicamente se ha cambiado el valor de `lr0` a `0,001` por el comentario en el código `# initial learning rate (SGD=1E-2, Adam=1E-3)` y se ha usado el optimizador Adam.

En la sección 9.3 se detallarán el resto de hiperparámetros que se han modificado, es decir, los argumentos de `train.py`.

9.2. *Data augmentation*

El *data augmentation* o aumento de datos es una técnica con la que se incrementa la cantidad y diversidad de los datos sin necesidad de recopilar nuevas muestras. Esto se logra aplicando transformaciones a las imágenes existentes en el conjunto de datos, como rotaciones o traslaciones. Es una técnica comúnmente empleada en el campo del *machine learning*, ya que crea modelos más robustos.

En el archivo YAML de hiperparámetros asociado al argumento `--hyp`, se encuentran las operaciones de *data augmentation* que se pueden aplicar a las imágenes del *dataset* en el entrenamiento de YOLOv7:

- `hsv_h`, `hsv_s`, `hsv_v` : Fracción de aumento para el tono (*Hue*), saturación (*Saturation*) o valor (*Value*), respectivamente, en el espacio de color HSV. Realiza variaciones aleatorias en estos valores para modificar los colores de la imagen y reforzar el modelo.

- `degrees`: Grados de rotación de la imagen.

- `translate`: Fracción de traslación de la imagen. Desplaza las imágenes horizontal o verticalmente para mejorar la robustez del modelo ante desplazamientos de los objetos.

- `scale`: Ganancia de zoom de la imagen, en porcentaje. Ajusta el tamaño de las imágenes para simular objetos más grandes o más pequeños.

- `shear`: Aplica una transformación que distorsiona las imágenes de manera no uniforme, se mide en grados.

- `perspective`: Fracción de perspectiva de la imagen. Aplica una transformación de perspectiva para simular diferentes puntos de vista.

- `flipud`, `fliplr`: Probabilidad de voltear la imagen de arriba a abajo o de izquierda a derecha.
- `mosaic`: Probabilidad de usar el aumento de mosaico, que combina cuatro imágenes en una sola.
- `mixup`: Probabilidad de usar el método *mixup*, que combina dos imágenes y sus etiquetas para mejorar la generalización del modelo. Técnica de *data augmentation* creada en 2018 por Zhang [62].
- `copy_paste`: Probabilidad de recortar y pegar objetos de una imagen a otra.
- `paste_in`: Probabilidad de recortar y pegar regiones de una imagen a otra, sin necesidad de ser objetos.

En nuestros conjuntos de datos vamos a aplicar una ligera rotación de ±10 grados, una traslación del 10 % y un escalado de ±10 %, porque permiten seguir reconociendo las señales añadiendo variación. Se han desactivado la opciones de volteo de imagen y *shear* porque son efectos que no se producen en la realidad y dificultan la identificación de las señales. Para acelerar el entrenamiento, se han desactivado las opciones de copiar y pegar, *mixup* y mosaico. El resto de opciones de *data augmentation* se han mantenido en la configuración por defecto del archivo `data/hyp.scratch.custom.yaml`, pensado para *datasets* personalizados como el mío.

9.3. Entrenamientos

Como se ha explicado al comienzo de este capítulo, el entrenamiento se produce mediante la ejecución del archivo `train.py`. Ya se ha detallado el significado de cada argumento, así que directamente se especificará el contenido de aquellos utilizados.

- `--weights`: Archivo `yolov7_training.pt` o `yolov7-tiny.pt` como pesos iniciales preentrenados, dependiendo si usamos la variante *medium* o *tiny*.
- `--cfg`: Archivo de configuración adaptado al número de clases del *dataset*.
- `--data`: Archivo YAML con los datos del *dataset*.
- `--hyp`: Archivo de hiperparámetros `data/hyp.scratch.custom.yaml` personalizado con los cambios explicados en las secciones 9.1 y 9.2.
- `--rect`: Habilito el entrenamiento rectangular, ya que las imágenes de los conjuntos de datos GTSDB y DFG no son cuadradas.
- `--img-size`: Utilizo la misma resolución tanto en entrenamiento como en validación. En cada *dataset* se han probado dos resoluciones, 640 y 1280 píxeles de ancho.

- `--batch-size`: Maximizando el rendimiento de la memoria RAM, establezco un tamaño de lote de 8 para los modelos entrenados a resolución de 640 píxeles y un tamaño de lote de 64 para los modelos entrenados a 1280 píxeles.
- `--epochs`: Dependiendo del *dataset* varía el número de épocas. 150 épocas para el *Dataset* 1, 175 épocas para el *Dataset* 2 y 200 épocas para el *Dataset* 3. Sin embargo, en cada época se evalúa el mAP del modelo y se guardan los pesos de la época donde se alcanza el máximo, previniendo el *overfitting*.
- `--adam`: Utilizo el optimizador Adam por su capacidad para ajustar dinámicamente las tasas de aprendizaje, mejorando la convergencia del modelo.
- `--device`: Indico que el entrenamiento debe realizarse con la CPU.
- `--workers`: Mi portátil tiene 8 núcleos, así que utilizo 8 subprocesos.
- `--name`: Nombre que tendrá la carpeta con los resultados del entrenamiento.

Todos los entrenamientos realizados se resumen en la Tabla 9.1. Como ejemplo, se muestra el comando del primer entrenamiento.

```
python train.py
--weights yolov7_training.pt
--cfg cfg\training\yolov7-gtsdb.yaml
--data data\gtsdb.yaml
--hyp data\hyp.scratch.custom.yaml
--rect
--img-size 640 640
--batch-size 64
--epochs 200
--adam
--device 0
--workers 8
--name entrenamiento1
```

Como se observa en la Tabla 9.1, se han realizado 8 entrenamientos distintos, 7 con la variante *tiny* y 1 con la variante *medium*. No se han podido hacer más pruebas con la variante *medium* por el gran coste temporal que suponía, ya que la duración del entrenamiento se extendía a una semana.

Se ha realizado un cambio en el código fuente de YOLOv7 en todos lo entrenamientos. Las imágenes del *dataset* GTSDB, presentes en los tres *Datasets*, están en formato `.ppm` y el modelo no las admitía. Gracias a un comentario de un usuario en la sección *Issues* del repositorio, adapté el archivo `utils/datasets` para que el formato de imagen `.ppm` fuera válido.

Dataset	Resolución	A color	Modelo
Dataset 1	640	✓	YOLOv7-tiny
	1280	✓	YOLOv7 tiny
Dataset 2	640	✓	YOLOv7-tiny
	1280	✓	YOLOv7 tiny
Dataset 3	640	✓	YOLOv7-tiny
	1280	✓	YOLOv7-tiny
	640	Escala de grises	YOLOv7-tiny
	1280	✓	YOLOv7

Tabla 9.1: Entrenamientos realizados.

Capítulo 10

Resultados y conclusiones

En este capítulo se muestra un resumen de los resultados más relevantes de los entrenamientos realizados y las conclusiones que se alcanzan.

La métrica que se ha usado para cuantificar el rendimiento de cada modelo es mAP@0.5, aplicado al conjunto de validación del *dataset*. Los *Datasets* 2 y 3 tienen las mismas clases de señales, por lo que se evalúan con el mismo conjunto de imágenes: el conjunto de validación del *Dataset* 2. En cambio, el *Dataset* 1 posee una clase menos, así que su conjunto de validación es un subconjunto del anterior.

El máximo mAP alcanzado en cada entrenamiento se ha producido antes de llegar a la época final, lo que nos indica que el número de épocas elegido ha sido correcto, un número mayor no conduciría a resultados mejores. Se guardan automáticamente en cada modelo los pesos asociados al máximo mAP, considerado el producto final de cada entrenamiento.

Para no extender innecesariamente este capítulo con resultados de menor relevancia, en el contenido adicional de este TFG se encuentran todas las gráficas y métricas obtenidas en cada entrenamiento.

10.1. Resolución de imagen

Dentro del modelo YOLOv7-tiny, se han utilizado en todos los *datasets* dos resoluciones diferentes, 640 y 1280 píxeles. En la Figura 10.1 se puede observar la diferencia de rendimiento del modelo. Las columnas azul oscuro representan la resolución de 640 px y las columnas azul claro la resolución de 1280 px.

De forma clara se puede concluir que la resolución de 1280 píxeles consigue mejores resultados que la resolución de 640 píxeles, en los tres *datasets*. Esto se puede explicar porque todas las imágenes originales tienen una resolución de más de 1280 píxeles de ancho. Por tanto, al reducir a más de la mitad su tamaño, se pierden muchos detalles que dificultan la detección y clasificación de las señales de tráfico más pequeñas. Sin embargo, el entrenamiento en la resolución de 1280 px fue hasta 4 veces más lento (en el *Dataset* 3, 13,5h frente a 3h) debido a que aumenta la cantidad de píxeles que procesar y se realiza mayor cantidad de cálculos. Se obtuvo un modelo más preciso a costa de un tiempo de

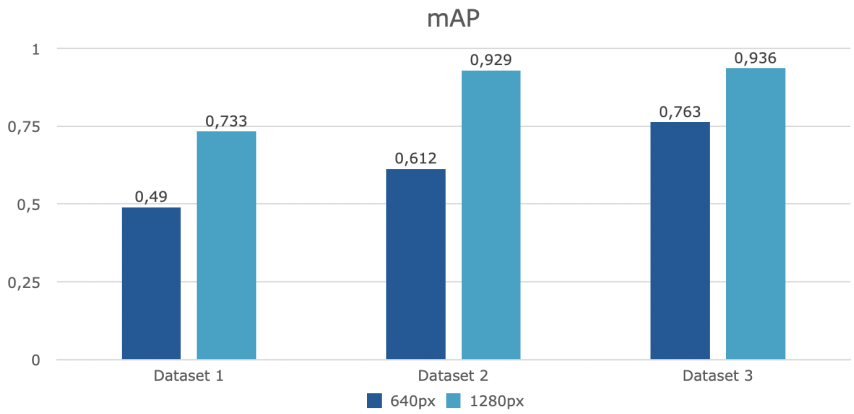


Figura 10.1: Métrica mAP@0.5 en cada *dataset* con el modelo YOLOv7-tiny, variando la resolución de imagen.

entrenamiento más prolongado, demostrando la importancia de este hiperparámetro en el proceso de detección de señales de velocidad.

10.2. Tamaño del conjunto de datos

Por la construcción de los datasets, se tiene que

$$Dataset\ 1 \subset Dataset\ 2 \subset Dataset\ 3$$

Observando la Figura 10.1 se deduce que el tamaño del conjunto de datos de entrenamiento afecta en gran medida a la puntuación final del modelo, ya que fijándonos en los resultados de los modelos entrenados en cualquiera de las dos resoluciones, se cumple que

$$\text{mAP}(Dataset\ 1) < \text{mAP}(Dataset\ 2) < \text{mAP}(Dataset\ 3)$$

Esta relación se justifica con el hecho de que un conjunto de datos más grande y balanceado proporciona una mayor diversidad de situaciones, que el modelo aprende y generaliza para poder aplicarlo en nuevas nuevas imágenes. También, se reduce el *overfitting* y hay una mayor representación de *outliers*, que en conjuntos más pequeños no se ven. Por otro lado, el balanceo de clases ayuda a que se aprendan las características de cada clase por igual.

Por tanto, se concluye que el *Dataset* 3 es el que ha producido mejores resultados, demostrando que el aumento del tamaño del *dataset* provoca un mejor rendimiento del modelo y balanceo de clases del conjunto de datos tiene un impacto positivo.

10.3. Escala de grises

Hay un único entrenamiento con imágenes en escala de grises. En la Tabla 9.1 vemos que existe otro entrenamiento igual, excepto en el color de la imagen, en la fila 5. En el entrenamiento a color, se obtuvo un mAP = 0,763, mientras que en el entrenamiento en

escala de grises se obtuvo un **mAP = 0,759**, ligeramente por debajo. Por tanto, no nos interesa el entrenamiento en escala de grises, ya que elimina información que facilita la detección de las señales.

Las señales de tráfico de limitación de velocidad destacan en el paisaje de carretera por el anillo exterior rojo, que ayuda a detectarlas. Al transformar la imagen a escala de grises, se pierde esa información y el modelo funciona levemente peor. En la Figura 10.2 se puede comparar la misma imagen del *Dataset* 3 en color y en escala de grises.

Pese a que un entrenamiento en escala de grises solamente requeriría un canal de color frente a los 3 canales convencionales RGB, disminuyendo la computación y el tiempo de entrenamiento, se produce una disminución no deseada en el rendimiento del modelo.

(a) Imagen en color

(b) Imagen en escala de grises

Figura 10.2: Comparación de una imagen del *Dataset* 3 en color frente a escala de grises

10.4. Tamaño del modelo

Para comparar el impacto del tamaño de la variante del modelo, se ha entrenado con las mismas condiciones (*Dataset* 3 con resolución de 1280 píxeles y en color) a la variante *tiny* y la variante *medium*.

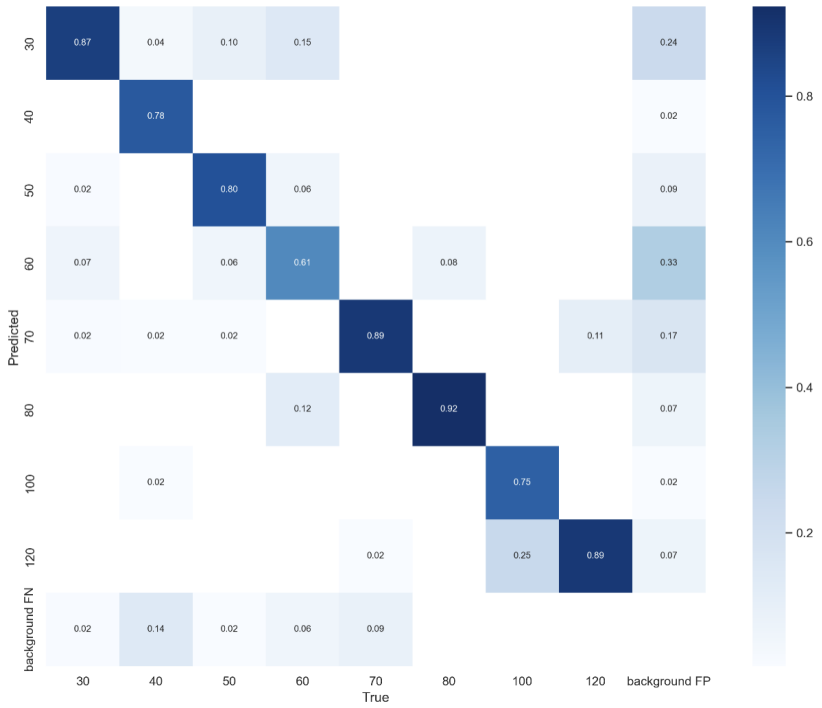
YOLOv7-tiny obtiene un mAP = 0,936, mientras que YOLOv7 obtiene un **mAP = 0,944**.

En la Figura 10.3 observamos las matrices de confusión de ambas variantes, obtenidas con un umbral de confianza de 0,01. Se puede apreciar que la diagonal del modelo *medium* está más marcada que la del *tiny*, indicando que el *recall* es superior. Además se producen menos confusiones entre las distintas clases (falsos positivos).

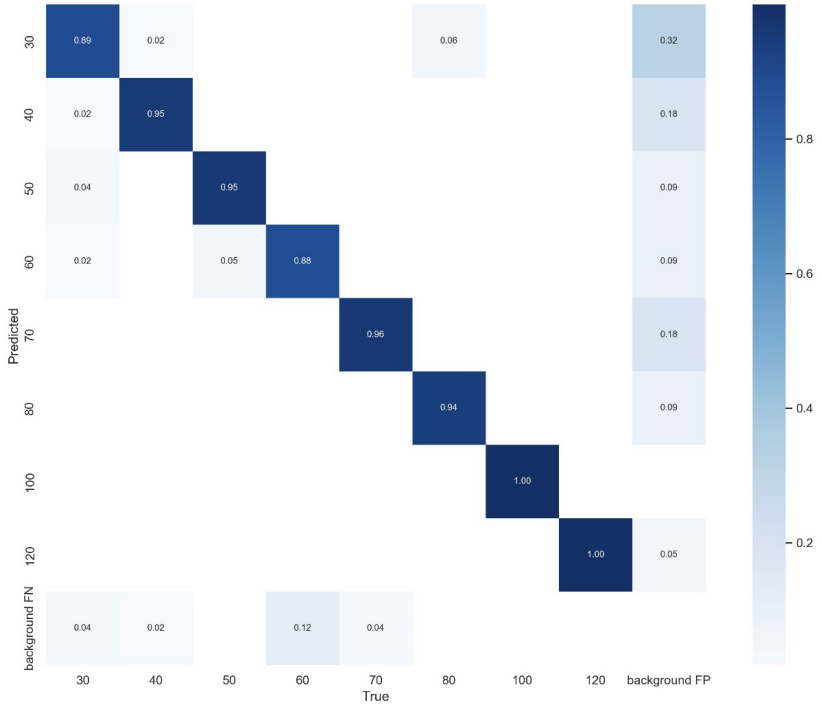
El modelo de mayor tamaño es más preciso, ya que ofrece una mayor capacidad para aprender y generalizar a partir de los datos de entrenamiento, lo que se traduce en un mejor rendimiento y una mayor precisión en la detección de señales.

10.5. Modelo final

Finalmente, se concluye que el modelo con mejores resultados ha sido el entrenado con la variante *medium*: YOLOv7. Cumple que todos los hiperparámetros que han sido



(a) Matriz de confusión del modelo YOLOv7-tiny



(b) Matriz de confusión del modelo YOLOv7

Figura 10.3: Comparación de las matrices de confusión de las variantes de YOLOv7 *tiny* y *medium* en el *Dataset* 3 a 1280 píxeles

vencedores en estas comparaciones finales, se encuentran en este modelo final. Se ha utilizado el *Dataset* 3, la resolución de 1280 píxeles, imágenes en color y la variante del modelo con mayor número de parámetros.

En la Figura 10.4 se muestra la gráfica de la métrica F1, que es la media armónica de la precisión y el *recall* según vimos en el capítulo 5.2. Esta medida varía según la confianza y en este modelo vemos que se mantiene bastante estable en el intervalo entre 0,2 y 0,9, además de obtener un valor por encima de 0,9. Esto nos indica que el modelo alcanza valores relativamente altos en precisión y *recall* simultáneamente. La métrica F1 alcanza su máximo en 0,763, así que ese valor de confianza es el que consigue que el modelo rinda de forma óptima.

Es el modelo más preciso generado en este trabajo para la tarea de detección y reconocimiento de señales de tráfico. Los pesos del modelo se encuentran en el contenido adicional de esta memoria y en el Anexo se incluye un manual para poder usar este modelo en imágenes o vídeos.

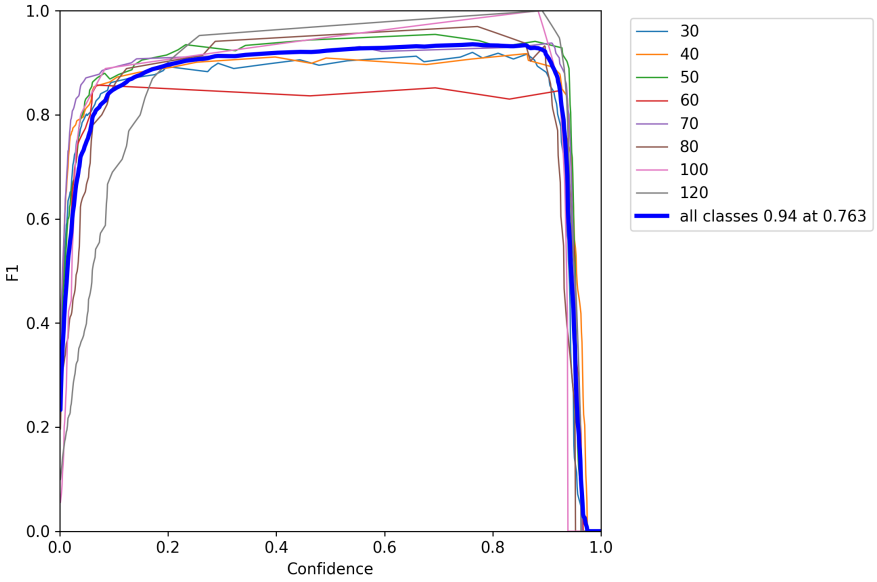


Figura 10.4: Gráfica de la métrica F1 en función de la confianza. Modelo final.

Capítulo 11

Trabajo futuro

En este capítulo se proponen varias direcciones para futuras investigaciones y desarrollos, que podrían mejorar y expandir los resultados obtenidos en este proyecto.

- **Aumentar la cantidad de señales por clase**. Según se ha visto en este trabajo, el aumento del número de instancias de cada clase ha provocado un aumento en las capacidades de predicción del modelo. Por lo tanto, continuando en esa línea deberían mejorar los resultados obtenidos.
- **Incorporar más límites de velocidad y de distintas regiones.** En España, existen señales de limitación de velocidad que no se han cubierto en este modelo, como 10, 20, 90 y 110 km/h. Pero, por ejemplo, en Estados Unidos existen señales distintas a las europeas, que son blancas y rectangulares. Ampliar el rango de límites de velocidad y formas reconocidas por el modelo garantizaría una cobertura más completa y precisa de las señales de tráfico en distintas regiones, habilitando su uso en otros países.
- **Añadir imágenes con más variedad de condiciones meteorológicas**. La mayoría de imágenes de los *datasets* utilizados correspondían a días soleados. Para aumentar la robustez del modelo, debería entrenarse con imágenes de distintas condiciones meteorológicas, como niebla, lluvia, nieve, etc., además de condiciones de baja luminosidad para adaptarse a entornos nocturnos.
- **Ampliar las clases de señales:** Entrenar el modelo para reconocer una variedad más amplia de señales de tráfico, más allá de las limitaciones de velocidad. En los *datasets* explorados en este trabajo hay una gran variedad de señales que se podrían utilizar para crear nuevos modelos, con la condición de tener acceso a más recursos computacionales para acelerar el entrenamiento.
- **Explorar otros modelos de detección de objetos:** Probar diferentes arquitecturas de modelos de detección de objetos de dos fases, como Faster RCNN o Mask RCNN, que han obtenido buenos resultados de precisión en estas tareas, aunque en general sean modelos más lentos. Otra vía de investigación son las versiones más

recientes de YOLO (v9 y v10), que podrían ofrecer mejoras en precisión y velocidad de inferencia, permitiendo mejorar este sistema. Incluso se podría formar un ensemble entre varios modelos.

- **Probar modelos con mayor número de parámetros:** Con la ayuda de recursos computacionales más potentes de los que he dispuesto a lo largo de este trabajo, se podría experimentar con modelos más grandes y complejos que capturen características más detalladas y complejas, mejorando el rendimiento general del sistema. Por ejemplo, las variantes más potentes de YOLOv7, como YOLOv7-E6E deberían obtener mejores resultados si se calibran los hiperparámetros correctamente, aunque conlleve la necesidad de una mayor potencia computacional.

- **Incorporar un modelo de reconocimiento numérico:** Se podría integrar un submodelo de reconocimiento numérico dentro del modelo de detección de objetos. Cuando se detecte una señal de límite de velocidad, este submodelo se activaría para identificar y clasificar las cifras de la señal. Los modelos de reconocimiento numérico son relativamente básicos y existen métodos altamente eficientes que podrían mejorar los resultados obtenidos en este trabajo. Esta integración permitiría una clasificación más precisa y robusta de las señales de limitación de velocidad.

- **Integrar el modelo en un sistema operativo en tiempo real:** Este sistema podría ser implementado en vehículos para aplicar directamente el modelo durante la conducción, permitiendo la identificación y respuesta a las señales de tráfico en tiempo real. Esta integración facilitaría la evaluación práctica del modelo y su rendimiento en condiciones reales de conducción, además de aportar valiosos datos para futuras mejoras y optimizaciones.

Estas propuestas buscan mejorar la precisión, robustez y aplicabilidad del modelo de detección y reconocimiento de señales de tráfico, haciendo que el sistema sea más eficaz y adaptable a una variedad de condiciones y requerimientos del mundo real.

Parte IV

Apéndices

Apéndice A

Manual

Para utilizar el modelo YOLOv7 (tamaños *medium* o *tiny*) entrenado con señales de límite de velocidad y disponible en el contenido adicional de esta memoria, se necesita utilizar el repositorio de GitHub oficial de YOLOv7 [58].

Se necesita tener instalado Python y pip. Hay que clonar el repositorio. Una vez descargado localmente en tu ordenador, hay que instalar los requerimientos, es decir las librerías que utiliza el programa. Se encuentran listadas en el archivo `requirements.txt`. En la consola de comandos, escribimos

```
pip install -r requirements.txt
```

Una vez instaladas todas las librerías se puede hacer uso del archivo `detect.py`. El programa aplica YOLOv7 a la fuente de imágenes que se desee. Este archivo debe ser ejecutado desde la consola de comandos, acompañado de una serie de argumentos.

- `-weights`: Este argumento especifica la ruta del archivo de pesos del modelo que se utilizará. En este caso, los archivos de pesos proporcionados en el contenido adicional.
- `-source`: Define la fuente de entrada para la inferencia. Puede ser un archivo, una carpeta o `0` para usar la cámara web.
- `-img-size`: Este parámetro establece la resolución de las imágenes que se usarán para la inferencia, en píxeles.
- `-conf-thres`: Umbral de confianza para la detección de objetos.
- `-iou-thres`: Umbral de IoU para la supresión de no máximos (NMS, *Non Maximun Supression*). Si varias detecciones superan este umbral, prevalecerá aquella asociada a una mayor confianza serán descartadas. El valor predeterminado es `0.45`.
- `-device`: Especifica el dispositivo en el que se ejecutará la inferencia. Puede ser una GPU (`0`) o CPU (`cpu`).

- `-view-img`: Si se incluye este argumento, la imagen con los resultados de la detección se mostrará por pantalla.
- `-save-txt`: Si se incluye este argumento, los resultados de la detección se guardarán en archivo de texto (`*.txt`).
- `-save-conf`: Si se incluye este argumento, las confianzas de las detecciones se guardarán en archivo de texto.
- `-nosave`: Si se incluye este argumento, no se guardarán imágenes o videos de los resultados.
- `-agnostic-nms`: Si se incluye este argumento, se aplicará NMS sin tener en cuenta las clases.

Para que el modelo funcione lo mejor posible, se recomienda utilizar la resolución de 1280 px, ya que ha sido entrenado con esa resolución de imágenes. También. añadir la opción `-agnostic-nms`, un umbral IoU de 0,5 y un umbral de confianza alto, para minimizar los falsos positivos, por ejemplo 0,7.

Un comando de ejemplo, suponiendo que nos encontramos en la ruta del repositorio clonado, sería:

```
python detect.py
--weights best.pt --conf-thres 0.7 --agnostic-nms
--iou-thres 0.5 --img-size 1280 --view-img
--source imagenDePrueba.jpg
```

Figura A.1: Prueba del modelo YOLOv7 con una imagen de la web http://autoescuelashnosmeloy.blogspot.com/2013/09/senales-de-velocidadcomo-se-cumplen-y.html

Como se observa en la Figura A.1, el modelo YOLOv7 es capaz de detectar la señal correctamente, incluso en unas condiciones de luminosidad que no son óptimas.

Apéndice B

Contenido adicional

Se incluye como contenido adicional asociado a esta memoria y disponible en el repositorio habilitado por la Escuela a tal fin:

- Los pesos del modelo YOLOv7 final que ha obtenido mejores resultados, en formato `.pt`. Para hacer uso de este modelo, se debe seguir el Manual (A).
- Los pesos del segundo mejor modelo obtenido, con la variante YOLOv7-tiny.
- Los *scripts* con los que se han realizado la exploración y preparación de los *datasets*. No se incluyen los propios *datasets* por la cantidad de memoria que ocupan. Se pueden descargar los conjuntos originales consultando la bibliografía [42] [53] [66].
- Productos de los entrenamientos realizados:
  - Matriz de confusión.
  - Curva F1 frente a confianza.
  - Curva precisión frente a confianza.
  - Curva *recall* frente a confianza.
  - Curva P-R.
  - Imágenes de validación con las predicciones del modelo.

Bibliografía

[1] O. Adeyemi et al. «Advanced Monitoring and Management Systems for Improving Sustainability in Precision Irrigation». En: *Sustainability* 9 (2017), pág. 353. DOI: 10.3390/SU9030353.

[2] Hamed Habibi Aghdam y Elnaz Jahani Heravi. *Guide to Convolutional Neural Networks*. Springer International Publishing, 2017. ISBN: 978-3-319-57549-0. DOI: 10.1007/978-3-319-57550-6.

[3] Anaconda, Inc. *Anaconda Distribution*. Accedido: 04-04-2024. 2024. URL: https://www.anaconda.com/products/distribution.

[4] Ivan Martins De Andrade y Cleonir Tumelero. «Increasing customer service efficiency through artificial intelligence chatbot». En: *Revista de Gestão* (2022). DOI: 10.1108/rege-07-2021-0120.

[5] Álvaro Arcos-García, Juan A Alvarez-Garcia y Luis M Soria-Morillo. «Evaluation of deep neural networks for traffic sign detection systems». En: *Neurocomputing* 316 (2018), págs. 332-344.

[6] Ahmed Nusayer Ashik et al. «Recognizing Bangladeshi Traffic Signs in the Wild». En: IEEE, dic. de 2022, págs. 1004-1009. ISBN: 979-8-3503-4602-2. DOI: 10.1109/ICCIT57492.2022.10055612.

[7] Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV]. URL: https://arxiv.org/abs/2004.10934.

[8] Junzhou Chen et al. «A real-time and high-precision method for small traffic-signs recognition». En: *Neural Computing and Applications* 34.3 (2022), págs. 2233-2245.

[9] Charles E. Clark. «Letter to the Editor—The PERT Model for the Distribution of an Activity Time». En: *Operations Research* 10.3 (1962), págs. 405-406. DOI: 10.1287/opre.10.3.405. eprint: https://doi.org/10.1287/opre.10.3.405. URL: https://doi.org/10.1287/opre.10.3.405.

[10] Christian Ertler et al. «The mapillary traffic sign dataset for detection and classification on a global scale». En: *European Conference on Computer Vision*. Springer. 2020, págs. 68-84.

[11] Sergio Escalera et al. *Traffic-Sign Recognition Systems*. Springer London, 2011. ISBN: 978-1-4471-2244-9. DOI: 10.1007/978-1-4471-2245-6.

[12] Ross Girshick. «Fast r-cnn». En: *Proceedings of the IEEE international conference on computer vision*. 2015, págs. 1440-1448.

[13] Ross Girshick et al. «Rich feature hierarchies for accurate object detection and semantic segmentation». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, págs. 580-587.

[14] GitHub, Inc. *GitHub*. Accedido: 03-10-2024. 2024. URL: https://github.com.

[15] GitHub, Inc. *GitHub Copilot*. Accedido: 03-10-2024. 2024. URL: https://github.com/features/copilot.

[16] Glassdor. *Sueldo Data Scientist en España 2024 | Glassdor*. Accedido: 21-06-2024. 2024. URL: https://www.glassdoor.es/Sueldos/data-scientist-sueldo-SRCH_KO0,14.htm.

[17] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[18] Kaiming He, Georgia Gkioxari y Piotr Dollár. «Girshick Ross. Mask R-CNN». En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 2961-2969.

[19] Kaiming He et al. «Deep residual learning for image recognition». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 770-778.

[20] Donald O Hebb. «The first stage of perception: growth of the assembly». En: *The Organization of Behavior* 4.60 (1949), págs. 78-60.

[21] Kurt Hornik, Maxwell Stinchcombe y Halbert White. «Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks». En: *Neural Networks* 3.5 (1990), págs. 551-560. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(90)90005-6. URL: https://www.sciencedirect.com/science/article/pii/0893608090900056.

[22] Andrew G Howard et al. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». En: *arXiv preprint arXiv:1704.04861* (2017).

[23] Gareth James et al. *An Introduction to Statistical Learning*. Springer US, 2021. ISBN: 978-1-0716-1417-4. DOI: 10.1007/978-1-0716-1418-1.

[24] Gareth James et al. «Deep Learning». En: Springer, 2021, págs. 403-460. DOI: 10.1007/978-1-0716-1418-1_10.

[25] He Juné. «Financial Abnormal Data Monitoring and Analysis Algorithm Based on Data Mining and Neural Network». En: *2022 Second International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE)* (2022), págs. 1-4. DOI: 10.1109/ICATIECE56365.2022.10047500.

[26] Jaskirat Kaur y Williamjeet Singh. «A systematic review of object detection from images using deep learning». En: *Multimedia Tools and Applications* 83 (4 ene. de 2024), págs. 12253-12338. ISSN: 1380-7501. DOI: 10.1007/s11042-023-15981-y.

[27] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». En: *Advances in neural information processing systems* 25 (2012).

[28] Leslie Lamport. *LaTeX: A Document Preparation System*. Accedido: 04-04-2024. 1994. URL: https://www.latex-project.org/.

[29] LearnOpenCV. *Mean Average Precision (mAP) for Object Detection: Evaluation Metric*. Accedido: 11-06-2024. URL: https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/#How-to-calculate-Average-Precision-(AP)-manually.

[30] Tsung-Yi Lin et al. «Focal loss for dense object detection». En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 2980-2988.

[31] Tsung-Yi Lin et al. «Microsoft coco: Common objects in context». En: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2017, págs. 740-755.

[32] Wei Liu et al. «Ssd: Single shot multibox detector». En: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer. 2016, págs. 21-37.

[33] Yifang Ma et al. «Artificial intelligence applications in the development of autonomous vehicles: a survey». En: *IEEE/CAA Journal of Automatica Sinica* 7 (2 mar. de 2020), págs. 315-329. ISSN: 2329-9266. DOI: 10.1109/JAS.2020.1003021.

[34] Markus Mathias et al. «Traffic sign recognition—How far are we from the solution?» En: *The 2013 international joint conference on Neural networks (IJCNN)*. IEEE. 2013, págs. 1-8.

[35] John McCarthy et al. «A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955». En: *AI Magazine* 27.4 (2006), pág. 12. DOI: 10.1609/aimag.v27i4.1904. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1904.

[36] Warren S. McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *The Bulletin of Mathematical Biophysics* 5 (4 dic. de 1943), págs. 115-133. ISSN: 0007-4985. DOI: 10.1007/BF02478259.

[37] Mendeley Ltd. *Mendeley - Reference Management Software*. Accedido: 30-06-2024. 2024. URL: https://www.mendeley.com/.

[38] Microsoft Corporation. *Microsoft Teams*. Accedido: 02-02-2024. 2024. URL: https://www.microsoft.com/es-es/microsoft-teams/group-chat-software.

[39] Microsoft Corporation. *Visual Studio Code*. Accedido: 03-10-2024. 2024. URL: https://code.visualstudio.com/.

[40] Jayant Mishra y Sachin Goyal. «An effective automatic traffic sign classification and recognition deep convolutional networks». En: *Multimedia Tools and Applications* 81 (13 mayo de 2022), págs. 18915-18934. ISSN: 1380-7501. DOI: 10.1007/s11042-022-12531-w.

[41] Andreas Mogelmose, Mohan Manubhai Trivedi y Thomas B Moeslund. «Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey». En: *IEEE transactions on intelligent transportation systems* 13.4 (2012), págs. 1484-1497.

[42] Institut für Neuroinformatik. *German Traffic Sign Detection Benchmark*. Accedido: 23-04-2024. 2013. URL: https://benchmark.ini.rub.de/gtsdb_dataset.html.

[43] OpenAI. *ChatGPT: Language Model*. https://www.openai.com/chatgpt. Accessed: 2024-06-26. 2023.

[44] Overleaf. *Overleaf: Online LaTeX Editor*. Accedido: 03-10-2024. 2024. URL: https://www.overleaf.com.

[45] Shehan P. Rajendran y Sreelatha Ganapathy. «Comparative Analysis of YOLOv4 and EfficientDet based models for Traffic Sign Detection in Autonomous Vehicles». En: *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 2023, págs. 1-8. DOI: 10.1109/ICCCNT56998.2023.10308271.

[46] Joseph Redmon et al. «You only look once: Unified, real-time object detection». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788.

[47] Shaoqing Ren et al. «Faster r-cnn: Towards real-time object detection with region proposal networks». En: *Advances in neural information processing systems* 28 (2015).

[48] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Inf. téc. Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[49] David E Rumelhart, Geoffrey E Hinton y Ronald J Williams. «Learning representations by back-propagating errors». En: *nature* 323.6088 (1986), págs. 533-536.

[50] Siniša Šegvić et al. «Exploiting temporal and spatial constraints in traffic sign detection from a moving vehicle». En: *Machine vision and applications* 25 (2014), págs. 649-665.

[51] Emel Soylu y Tuncay Soylu. «A performance comparison of YOLOv8 models for traffic sign detection in the Robotaxi-full scale autonomous vehicle competition». En: *Multimedia Tools and Applications* (ago. de 2023). ISSN: 1380-7501. DOI: 10.1007/s11042-023-16451-1.

[52] Christian Szegedy et al. «Going deeper with convolutions». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, págs. 1-9.

[53] Domen Tabernik y Danijel Skočaj. «Deep learning for large-scale traffic-sign detection and recognition». En: *IEEE transactions on intelligent transportation systems* 21.4 (2019), págs. 1427-1440.

[54] Mingxing Tan, Ruoming Pang y Quoc V Le. «Efficientdet: Scalable and efficient object detection». En: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, págs. 10781-10790.

[55] Juan Terven, Diana-Margarita Córdova-Esparza y Julio-Alejandro Romero-González. «A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas». En: *Machine Learning and Knowledge Extraction* 5.4 (2023), págs. 1680-1716.

[56] Universidad de Valladolid. *¿Qué es el crédito E.C.T.S ?* Accedido: 24-06-2024. 2024. URL: https://www.uva.es/export/sites/uva/2.estudios/2.09.preguntasfrecuentes/detalle/97a8aaa3-c8f8-11eb-8692-00505682371a/.

[57] Ao Wang et al. «Yolov10: Real-time end-to-end object detection». En: *arXiv preprint arXiv:2405.14458* (2024).

[58] Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao. *YOLOv7 GitHub repository*. Accedido: 07-07-2024. 2023. URL: https://github.com/WongKinYiu/yolov7.

[59] Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao. «YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors». En: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, págs. 7464-7475. DOI: 10.1109/CVPR52729.2023.00721.

[60] Samir S. Yadav y S. Jadhav. «Deep convolutional neural network based medical image classification for disease diagnosis». En: *Journal of Big Data* 6 (2019). DOI: 10.1186/s40537-019-0276-2.

[61] Jiana Yao et al. «Research on detection and classification of traffic signs with data augmentation». En: *Multimedia Tools and Applications* 82 (25 oct. de 2023), págs. 38875-38899. ISSN: 1380-7501. DOI: 10.1007/s11042-023-14895-z.

[62] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization*. 2018. arXiv: 1710.09412 [cs.LG]. URL: https://arxiv.org/abs/1710.09412.

[63] Jianming Zhang et al. «CCTSDB 2021: a more comprehensive traffic sign detection benchmark». En: *Human-centric Computing and Information Sciences* 12 (2022).

[64] Wei Qi Qin Zhongbing y Yan. «Traffic-Sign Recognition Using Deep Learning». En: ed. por Wei Qi, Ho Harvey Nguyen Minh y Yan. Springer International Publishing, 2021, págs. 13-25. ISBN: 978-3-030-72073-5.

[65] Yanzhao Zhu y Wei Qi Yan. «Traffic sign recognition based on deep learning». En: *Multimedia Tools and Applications* 81 (13 mayo de 2022), págs. 17779-17791. ISSN: 1380-7501. DOI: 10.1007/s11042-022-12163-0.

[66] Z. Zhu et al. «Traffic-Sign Detection and Classification in the Wild». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://cg.cs.tsinghua.edu.cn/traffic-sign/. 2016, págs. 2110-2118.