Proyecto Fin de Máster

Máster en Ingeniería Electrónica, Robótica y Automática

Uso de redes neuronales para identificación de señales de tráfico

Autor: José Enrique Maese Álvarez

Tutor: Antonio Javier Gallego Len

Dpto. de Ingeniería de Sistemas y Automatización
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Proyecto Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Uso de redes neuronales para identificación de señales de tráfico

Autor:
José Enrique Maese Álvarez

Tutor:
Antonio Javier Gallego Len
Profesor ayudante doctor

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Proyecto Fin de Máster: Uso de redes neuronales para identificación de señales de tráfico

Autor: José Enrique Maese Álvarez

Tutor: Antonio Javier Gallego Len

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia y amigos.

Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a mi familia, por ser un apoyo inquebrantable durante todo el año.

A los profesores que me han acompañado en este máster, gracias por la oportunidad de aprendizaje que habéis brindado.

Finalmente, a mis compañeros e increíbles amigos con los que compartido este año. Sin vosotros esta experiencia no habría sido la misma y, desde luego, menos divertida.

José Enrique Maese Álvarez

Sevilla, 2024

Resumen

En este proyecto de fin de máster nos centramos en la aplicación de la visión artificial y las redes neuronales para el reconocimiento de señales de tráfico, con un enfoque particular en el modelo YOLOv8. La visión artificial ha revolucionado el campo de la inteligencia artificial, permitiendo a las máquinas interpretar y comprender el entorno visual de manera similar a los humanos. El reconocimiento de matrículas y señales de tráfico, debido a su componente de seguridad vial, es una de las aplicaciones más complejas de esta tecnología, contribuyendo significativamente a la automatización de vehículos.

En el proyecto examinamos los fundamentos teóricos de las redes neuronales convolucionales y su evolución, destacando cómo modelos avanzados como YOLOv8 han mejorado la precisión y eficiencia del reconocimiento de objetos en tiempo real. Se discutirán las técnicas de entrenamiento de modelos utilizando grandes bases de datos, así como la implementación práctica de estos sistemas en entornos reales.

A través de una metodología rigurosa que incluye la recolección y procesamiento de datos, el entrenamiento del modelo en plataformas de computación en la nube y la evaluación de resultados en condiciones diversas, se demuestra la robustez y aplicabilidad de YOLOv8 en el reconocimiento de señales de tráfico. En el estudio también abordamos las limitaciones actuales y se propondrán áreas de mejora futura.

En conclusión, este trabajo evidencia el potencial de las redes neuronales y la visión artificial para transformar la seguridad y eficiencia en la conducción autónoma, subrayando la importancia de la continua investigación y desarrollo en este campo para el avance de la inteligencia artificial en consonancia con la industria automotriz.

Abstract

In this master's thesis project, we focus on the application of computer vision and neural networks for traffic sign recognition, with a particular focus on the YOLOv8 model. Computer vision has revolutionized the field of artificial intelligence, allowing machines to interpret and understand the visual environment in a similar way to humans. The recognition of number plates and road signs, due to its road safety component, is one of the most complex applications of this technology, contributing significantly to vehicle automation.

In the project we examine the theoretical foundations of convolutional neural networks and their evolution, highlighting how advanced models such as YOLOv8 have improved the accuracy and efficiency of real-time object recognition. Model training techniques using large databases will be discussed, as well as the practical implementation of these systems in real environments.

Through a rigorous methodology that includes data collection and processing, model training on cloud computing platforms, and evaluation of results under varying conditions, the robustness and applicability of YOLOv8 in traffic sign recognition is demonstrated. The study also addresses current limitations and will propose areas for future improvement.

In conclusion, this work highlights the potential of neural networks and computer vision to transform safety and efficiency in autonomous driving, underlining the importance of continued research and development in this field for the advancement of artificial intelligence in line with the automotive industry.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

El mundo de la automoción ha evolucionado rápidamente desde la creación del primer vehículo a motor a finales del siglo XIX. Los numerosos avances conseguidos han ido mejorando las prestaciones de los vehículos hasta alcanzar los estándares de la actualidad. Junto a estas mejoras de velocidad o potencia se han ido desarrollando nuevas tecnologías en el ámbito de la seguridad vial [1]. Desde la inclusión del cinturón de seguridad hasta el sistema antibloqueo de ruedas (ABS), pasando por la optimización de los materiales y diseño de la carrocería. En este proyecto, trataremos uno de los temas de más repercusión en la última década: el uso de la visión artificial para mejorar la seguridad vial.

El acercamiento de la industria automovilística a la inteligencia artificial está encontrando solución a diversas tareas que afectan a la conducción [2]. Podemos encontrar aplicaciones capaces de mantener el coche en el carril, aparcar automáticamente e incluso un principio de conducción autónoma en vehículos particulares. En este proyecto nos centraremos en este último punto, concretamente en la detección y clasificación de señales de tráfico.

Para realizar esta tarea se han tenido en cuenta diversas consideraciones. En primer lugar, las imágenes en movimiento serán captadas mediante una cámara situada en la parte frontal del vehículo. Para realizar un reconocimiento de imágenes óptimo se requieren unas condiciones ambientales regulares, sin embargo, esto no es posible al completo en la tarea que vamos a desarrollar por lo que deberemos conseguir un modelo neuronal lo suficientemente robusto.

El tratamiento de la imagen se realizará en dos partes para maximizar la eficiencia de los resultados. Comenzaremos localizando las señales de tráfico mediante un modelo neuronal de una sola pasada que aporta buenos resultados con poco tiempo de cálculo. Este mismo modelo realizará una primera clasificación de las señales según su clase (advertencia, peligro, obligatoriedad…), sin concretar el tipo específico de la señal. A continuación, se segmentará la señal previamente localizada y, tras un preprocesado, la clasificaremos mediante un modelo específico para su clase. Con este acercamiento en dos pasos limitamos la capacidad de clasificación de cada modelo reduciendo los errores finales.

1.1 Objetivos

1.1.1 Objetivos generales

El objetivo principal del proyecto es crear y estudiar los resultados del reconocimiento y clasificación de señales de tráfico en un entorno real mediante redes neuronales. Para ello se utilizarán dos tipos de modelo, un modelo YOLO ampliamente utilizado, aunque de reciente creación y un modelo de clasificación. Además de estudiar los resultados obtenidos, este proyecto tiene como finalidad familiarizarse con los entornos de trabajo de redes neuronales, así como de entrenamiento en la nube. También es conveniente mencionar la experiencia adquirida en la búsqueda, creación y gestión de bases de datos de gran tamaño utilizando varios lenguajes de programación.

1.1.2 Objetivos específicos

Para realizar el proyecto debemos dividirlo en varias fases y objetivos más concretos. Debemos crear y gestionar bases de datos, crear algoritmos de entrenamiento de los modelos neuronales a utilizar y procesar videos reales para ver los resultados:

- Crear un modelo capaz de localizar y segmentar las señales de tráfico.
- Crear un modelo capaz de clasificar imágenes recortadas de señales en un amplio rango de clases posibles.
- Gestionar las bases de datos necesarias para entrenar ambos modelos neuronales.
- Unificar ambos procesos en un único programa capaz de analizar un video de forma completa.

1.2 Metodología

La metodología seguida en este proyecto será la siguiente:

- Conseguir y preprocesar una base de datos de imágenes tomadas desde el frontal del vehículo. Dada la estructura del modelo YOLO [3] debemos conseguir una gran cantidad de imágenes por lo que descartamos la creación propia de las mismas.
- Entrenar un modelo YOLO capaz de localizar las señales y clasificarlas en las clases especificadas con alta eficiencia. Este punto es clave ya que los errores cometidos en esta clasificación se mantendrán en el resultado final.
- Conseguir otra base de datos, en este caso de imágenes recortadas de señales de tráfico.
- Entrenar un segundo modelo neuronal para la clasificación de las imágenes segmentadas.
- Aplicar ambos modelos de forma consecutiva a cada fotograma del video de entrada para analizar los resultados finales obtenidos.

2 ESTADO DEL ARTE

El nacimiento del primer automóvil se remonta a 1769, con el desarrollo de un vehículo a vapor a manos del inventor francés Nicolas-Joseph Cugnot. Otras fuentes hablan de una invención más temprana, a finales del siglo XVIII, por un miembro de las misiones jesuitas en China, Ferdinand Verbiest. En 1885, Karl Friedrich Benz diseñó el considerado como primer vehículo propulsado por un motor de combustión interna, el modelo Benz Patent-Motorwagen . Sin embargo, fue en e1 verano de 1908 cuando Henry Ford comenzó la producción en serie gracias a las nuevas cadenas de montaje. Este hecho supuso el inicio de una nueva era en la movilidad y la industria automovilística se convirtió rápidamente en un pilar fundamental de la economía global y un símbolo del progreso tecnológico.

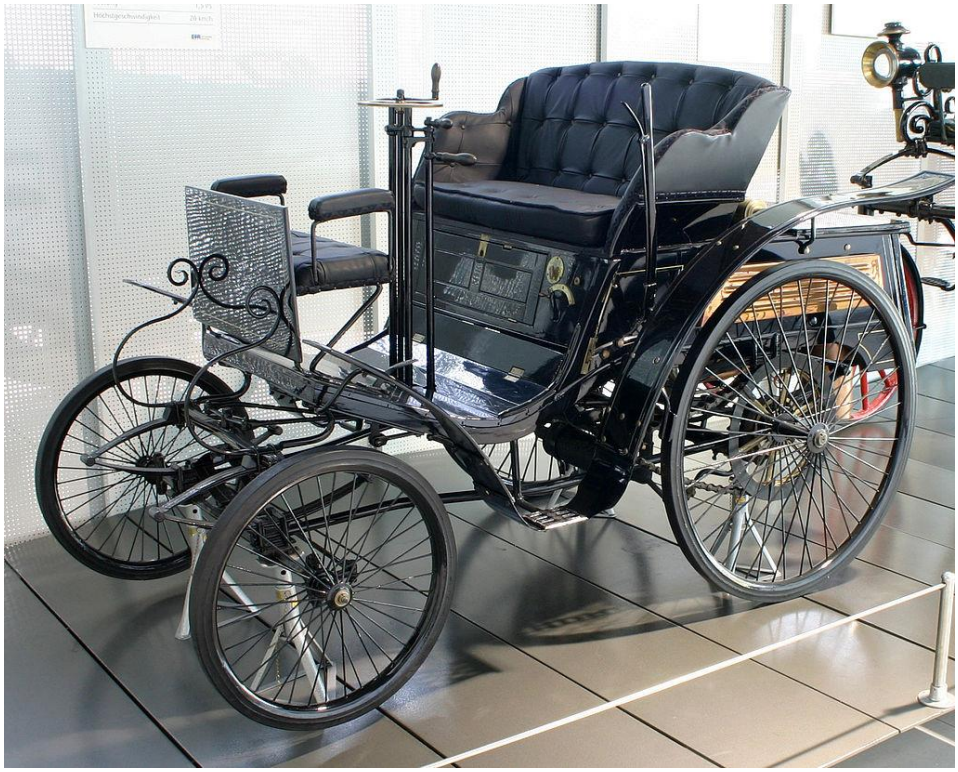

Figura 2-1. Modelo Benz "Velo", 1894.

Debido a la importancia social y económica de la industria automotriz, desde sus primeros días los fabricantes e ingenieros han buscado de forma constante cómo mejorar la eficiencia y funcionalidad de los vehículos a través de la automatización. A medida que la tecnología avanza, se han ido introduciendo nuevos sistemas para simplificar las tareas de la conducción y mejorar la seguridad de los ocupantes de un vehículo, así como de los transeúntes. En 1939, General Motors introdujo un importante cambio creando el primer sistema de transmisión automático. Desde entonces, los avances hacia la conducción

autónoma han sido imparables: control de crucero adaptativo, sistema antibloqueo de frenos (ABS) o incluso faros con encendido automático según las condiciones meteorológicas son solo algunos ejemplos [4].

2.1 Vehículos autónomos

2.1.1 Historia

El Desarrollo de los vehículos completamente autónomos, si bien ha cobrado mayor relevancia en los últimos 20 años, tiene sus raíces en proyectos que se remontan décadas atrás. Un hito significativo sucedió en 1925, cuando el ingeniero eléctrico Francis Houdina presentó el primer prototipo de un vehículo autónomo en Nueva York. Aunque este automóvil era controlado a distancia, su exhibición pública marcó un importante punto de partida. A pesar de algunos percances durante su recorrido inicial, el vehículo, conocido como Chandler, despertó el interés de la gente y se construyó entre 1926 y 1930, sentando así las bases para futuras investigaciones en este campo.

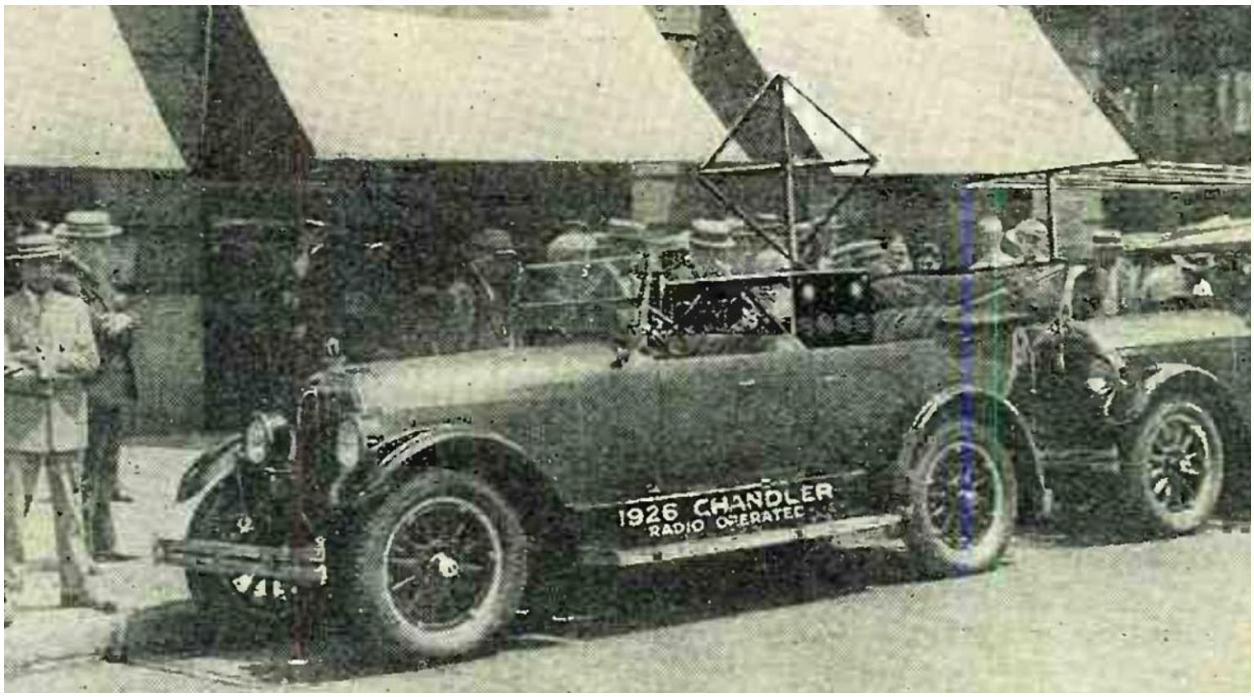

Figura 2-2. Vehículo con control a distancia del ingeniero Francis Houdina.

En las décadas siguientes, el impulso por desarrollar vehículos autónomos continuó, destacándose el trabajo del ingeniero alemán Ernst Dickmanns en los años 80. Dickmanns es generalmente reconocido como el pionero del vehículo autónomo contemporáneo, al convertir una furgoneta Mercedes-Benz en un vehículo autónomo guiado por una computadora integrada. Su logro más notable fue en 1987, cuando el vehículo demostró su capacidad para recorrer calles sin tráfico a una velocidad considerable de 63 kilómetros por hora. Este avance marcó un hito en la demostración práctica de la viabilidad de la conducción autónoma y allanó el camino para investigaciones posteriores.

Figura 2-3. Interior de la furgoneta de Ernst Dickmanns.

La década de los 90 presenció avances significativos en el campo de los vehículos autónomos, con Dickmanns llevando a cabo proyectos que desafiaron los límites en la automoción de la época. Desde recorridos en entornos urbanos con tráfico hasta viajes de larga distancia entre ciudades europeas, los logros de Dickmanns subrayaron el potencial de la tecnología autónoma en transformar la movilidad. Financiado por la Comisión Europea a través del Proyecto Eureka, Dickmanns recibió un apoyo crucial que permitió la realización de investigaciones de vanguardia y sentó las bases para la evolución continua de los vehículos autónomos en las décadas siguientes.

2.1.2 Clasificación

En la actualidad, la conducción autónoma se organiza en seis niveles distintos, caracterizados por su grado de automatización y control por parte del conductor, según los estándares establecidos por la Sociedad de Ingenieros de Automoción (SAE, por sus siglas en inglés). Los niveles del 0 al 2 representan vehículos que incorporan características de asistencia al conductor, mientras que los niveles del 3 al 5 presentan funciones reales de automatización.

- Nivel 0. Los vehículos son los convencionales que encontramos en la mayoría de las carreteras, donde el conductor es responsable de todas las tareas y maniobras. Aunque estos vehículos pueden contar con sensores para detectar obstáculos cercanos, la conducción recae completamente en el conductor.
- Nivel 1. Se introducen sistemas de asistencia avanzados que controlan la dirección, la velocidad y el frenado. Estos incluyen funciones como el control de distancia, la asistencia de estacionamiento y el sistema automático de frenado de emergencia, aunque el conductor sigue siendo responsable de la mayoría de las funciones y debe mantener las manos en el volante.
- Nivel 2. Los vehículos pueden tomar el control de algunos sistemas de forma autónoma, pero el conductor debe permanecer alerta y listo para intervenir si es necesario. Esta etapa puede permitir una conducción "manos libres" en la que el conductor deja de controlar temporalmente el volante y el acelerador, mientras el vehículo gestiona ciertas funciones.
- Nivel 3. Este nivel marca un avance significativo, donde los vehículos autónomos son capaces de

analizar su entorno y tomar decisiones de manera inteligente utilizando sensores para registrar información y una visión computarizada para procesarla. Es a partir de este punto donde comienzan a entran en juego las aplicaciones realizadas en este proyecto respecto a la visión artificial [5].

- Nivel 4. Los vehículos pueden conducir sin intervención humana en situaciones específicas. Utilizan algoritmos avanzados de inteligencia artificial y pueden controlar todas las funciones críticas de conducción. En este punto, ya no se designa a un conductor, sino que los ocupantes se convierten en pasajeros.
- Nivel 5. Representa la cúspide de la conducción autónoma, donde los vehículos comparten información con su entorno a través de telecomunicaciones y el Internet de las Cosas (IoT). Este nivel de automatización plena solo se considera posible mediante la comunicación con tecnología G5, permitiendo una integración total con el entorno circundante para una conducción totalmente autónoma.

2.2 Legislación

Aunque los avances en inteligencia y visión artificial han llevado a la conducción autónoma hasta cotas inimaginables hace unos años, sigue enfrentándose a retos en el ámbito legislativo para poder aplicarse de forma generalizada. Por ello, la legislación europea sobre la conducción autónoma ha cambiado significativamente, estableciendo un marco regulatorio que protege tanto a los usuarios como a los fabricantes.

La Unión Europea ha legalizado la conducción autónoma de nivel 3, permitiendo que los coches tomen el control con ciertas limitaciones de velocidad y tipo de vías. La normativa europea se centra en asegurar que los vehículos autónomos puedan integrarse de manera segura y eficiente en las carreteras europeas, destacando la necesidad de que estos sistemas sean transparentes, seguros y sujetos a supervisión humana en todo momento.

Sin embargo, la regulación específica depende de cada país miembro, lo que introduce variaciones en la aplicación de estas normas. Una de las modificaciones más recientes en la legislación europea aborda la responsabilidad en caso de accidentes con coches autónomos. Tradicionalmente, el fabricante del vehículo era el responsable de cualquier mal funcionamiento, pero la nueva directiva permite que los proveedores de componentes demuestren que sus sistemas funcionaron correctamente en el momento del accidente. Esto invierte la carga de la prueba, favoreciendo a los fabricantes de vehículos autónomos, quienes ahora pueden demandar a los proveedores en caso de fallos.

3 METODOLOGÍA

En este capítulo se analizará el procedimiento seguido durante el proyecto. Se comentarán las herramientas utilizadas para tratar los datos, además de los lenguajes de programación empleados. Estudiaremos las decisiones tomadas para definir el problema y evitar los posibles problemas que nos encontraremos en el proceso. Finalmente, definiremos brevemente los modelos neuronales utilizados para alcanzar los objetivos propuestos.

3.1 Herramientas

Durante el desarrollo del proyecto se han empleado varias herramientas clave. En primer lugar, se ha escogido Python como lenguaje de programación principal debido a su versatilidad y amplia gama de bibliotecas especializadas en aprendizaje automático y visión artificial.

Se ha utilizado Google Colab para la programación y el entrenamiento en la nube, lo que permitió aprovechar la potencia de cálculo disponible en servidores remotos y GPUs especializadas. Además, se ha recurrido a diversas plataformas web de gestión de bases de datos, como Roboflow, Kaggle y GitHub, para acceder a grandes conjuntos de datos de calidad. El uso de estas herramientas ha permitido optimizar el tiempo de trabajo disminuyendo especialmente el tiempo de entrenamiento de la red YOLO, como se verá en secciones posteriores. Para el análisis de resultados, se empleó Weights & Biases (wandb), una herramienta web que facilita la visualización y comparación de métricas de rendimiento de modelos de aprendizaje automático. Por último, se utilizó Spider y Visual Studio como entornos de desarrollo locales para la escritura y depuración final del código.

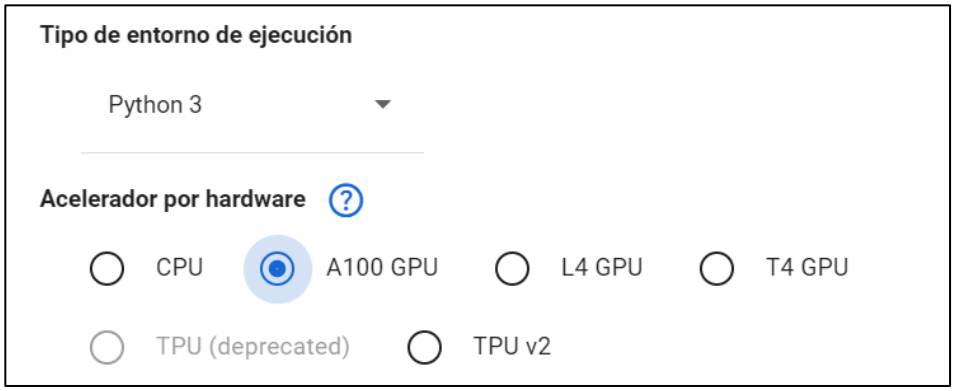

Figura 3-1. El entorno de Google Colab nos permite acceder a GPUs especializadas de alta capacidad.

El hardware del que se ha dispuesto para realizar este proyecto es un portátil ASUS ZenBook con una CPU Intel CORE i7 -1165G7 y una GPU Intel IRIS Xe Graphics. Este dispositivo nos ha permitido desarrollar gran parte del proyecto en condiciones óptimas, aunque los procedimientos con mayor carga computacional, como

los entrenamientos de los modelos neuronales, se ha realizado mediante las herramientas en la nube ya mencionadas.

3.2 Procedimiento

Antes de comenzar a desarrollar el proyecto se ha realizado un estudio previo para analizar los posibles caminos que podríamos seguir para resolver el problema. Los métodos clásicos de visión artificial se utilizan a modo de preprocesado, pero se descartaron al no ser adecuados para resolver el problema de forma efectiva. Posteriormente, se evaluaron distintos modelos de redes neuronales convencionales como ResNet-50 [6] o GoogleNet [7], sin embargo, no solucionan la dificultad inicial de detección de objetos.

Por este motivo y tras diversas consideraciones, se optó por utilizar YOLO (You Only Look Once) [8], un modelo de detección de objetos que permite una localización precisa de las señales de tráfico en una sola pasada. Con el uso únicamente de este modelo no conseguimos los resultados esperados, por lo que se ha incluido un modelo de clasificación en serie para mejorar el proceso.

El enfoque escogido consiste en dividir el problema en dos fases: una primera fase de localización y clasificación de objetos, una segunda de clasificación en subclases más específicas y una fase final para unificar ambos modelos. Este diseño consigue abordar los desafíos específicos asociados con la identificación precisa de objetos.

Figura 3-2. Clases de salida de la red YOLO. De arriba abajo: *other*, *danger*, *mandatory* y *prohibitory*.

3.3 División del problema

3.3.1 Fase 1: Localización de las señales y clasificación inicial

En esta fase inicial, se implementó un modelo de detección de objetos, específicamente YOLOv8, reconocido por su eficiencia en la detección y localización de múltiples objetos en una imagen en una sola pasada. El objetivo principal de esta etapa fue localizar las señales de tráfico presentes en las imágenes. Sin embargo, dada la diversidad y la cantidad de señales de tráfico en entornos viales, se decidió dividir las señales detectadas en cuatro clases distintas según su tipología y función, como se muestra en la Figura 3-2. Clases de salida de la red YOLO. De arriba abajo: *other*, *danger*, *mandatory* y *prohibitory*.Figura 3-2. Esta división inicial es fundamental para disminuir la complejidad del problema ya que el modelo YOLOv8 necesitará clasificar los elementos detectados en 4 clases frente a las decenas de clases del problema inicial.

Este enfoque de clasificación preliminar facilita una segmentación más precisa de las señales en grupos específicos, lo que resulta fundamental para el proceso de clasificación posterior. Además, permite reducir la complejidad del problema al tratar cada clase de señal de manera individual, mejorando así la eficiencia y la precisión del modelo.

3.3.2 Fase 2: Clasificación en subclases específicas

En la segunda fase del proceso, aplicamos un modelo de clasificación de imágenes distinto a cada clase de salida del modelo YOLO para asignar una categoría específica a cada señal de tráfico previamente segmentada. Para esta tarea, se implementó ResNet-50, una red neuronal convolucional profunda con eficacia demostrada en tareas de clasificación de imágenes debido a su arquitectura robusta y capacidad para aprender características complejas de las imágenes.

El modelo, como se mostrará más adelante, se entrenará con base de datos de señales específicas de la clase correspondiente. Par reducir el tiempo y los recursos de ejecución del proyecto se va a realizar el entrenamiento para una única clase del modelo de forma que puedan generalizarse los resultados.

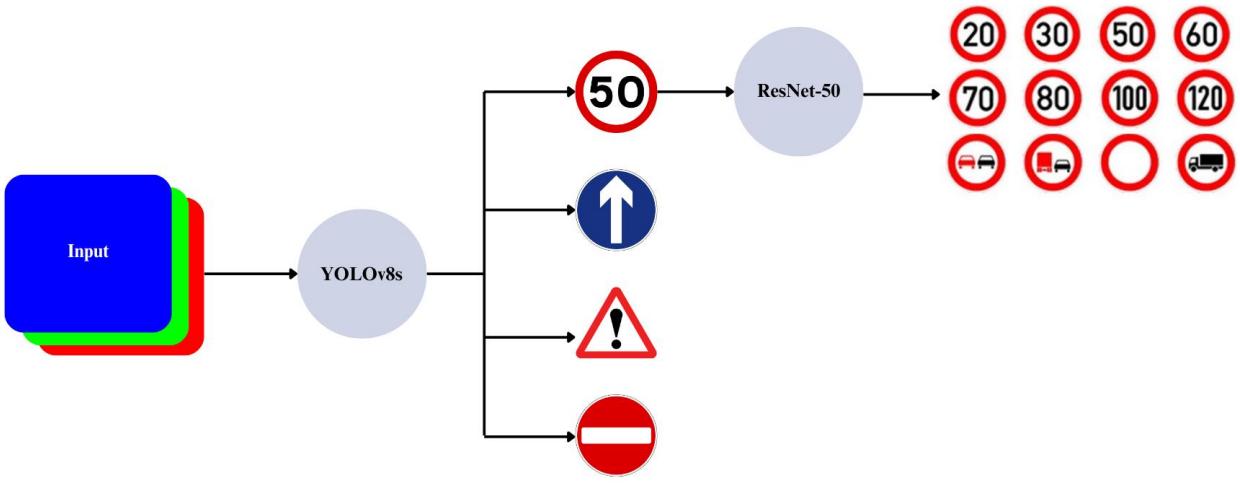

Figura 3-3. Diagrama de flujo de la clasificación de objetos en una imagen.

El uso de ResNet-50 en esta etapa permitió una clasificación precisa y detallada de las señales de tráfico, considerando características específicas de cada clase identificada en la fase anterior. Gracias a esta división del proceso en dos fases distintas, se logró una mejora significativa en la precisión y la eficiencia del sistema de identificación de señales de tráfico en imágenes, superando las limitaciones inherentes a un enfoque de detección y clasificación integrados en una única red neuronal.

3.3.3 Fase 3: Unificación de modelos

Una vez obtenidos ambos modelos neuronales debemos realizar un pequeño post procesado para unificarlos. Debemos tener en cuenta que la entrada del modelo YOLO es una imagen de un plano completo de la vista frontal desde un vehículo mientras que la entrada de la red ResNet-50 es un plano cercano de una señal de tráfico. Tras conseguir el código que ejecute ambos modelos de forma consecutiva se va a desarrollar otro capaz de recibir video en lugar de imágenes estáticas para comprobar su funcionamiento en tiempo real.

Finalmente, se comprobará el funcionamiento del sistema con videos grabados en entornos reales. Así se demostrará su funcionamiento frente a entornos no controlados de luminosidad o con interferencias de otros vehículos.

4 MODELO YOLOV8

En este capítulo vamos a describir en detalle el uso del modelo YOLOv8 para la identificación de señales de tráfico. Aunque en la fecha de creación de este documento existen versiones más avanzadas del modelo, se ha escogido la versión 8 por ser el más estable del momento. Este modelo, como se verá en esta sección está optimizado para ofrecer mejorar significativas en precisión y velocidad. A continuación, exploraremos los fundamentos teóricos de YOLOv8 [8] y su arquitectura, comparándolo con versiones anteriores y su posible aplicación en tiempo real [10].

4.1 Fundamentos teóricos

YOLO (You Only Look Once) es una de las arquitecturas de redes neuronales convolucionales más avanzadas para la detección de objetos en tiempo real. Su diseño permite realizar detecciones en una sola pasada a través de la imagen, lo que contrasta con otros modelos que requieren múltiples etapas de procesamiento. Esta característica hace que YOLO sea extremadamente rápido y eficiente, ideal para aplicaciones que requieren baja latencia, como los sistemas de conducción autónoma y vigilancia en tiempo real.

4.1.1 Arquitectura

Las partes principales de la arquitectura son:

- **Backbone**: se conoce como *backbone* al bucle principal de la red, capaz de extraer características con gran detalle de la imagen original. En este caso está basado en CSPDarknet53, lo que significa una mejora sobre Darknet19 utilizado en YOLOv3. Es capaz de obtener patrones simples en las capas iniciales (como bordes o texturas) o elementos más complejos y abstractos en las capas profundas.
- **Neck**: funciona como nexo entre el *backbone* y la cabeza del modelo. Su función consiste en recoger mapas de características de distintas partes del *backbone*. Esto es especialmente útil para detectar objetos de distintos tamaños gracias a la fusión de características de diferentes escalas. Además, ayuda a integrar información del contexto de la escena para realizar la predicción.
- **Head*:*** se trata de la parte final de la red y está encargada de generar las predicciones de salida. Aporta los cuadros delimitadores y puntuaciones de confianza.

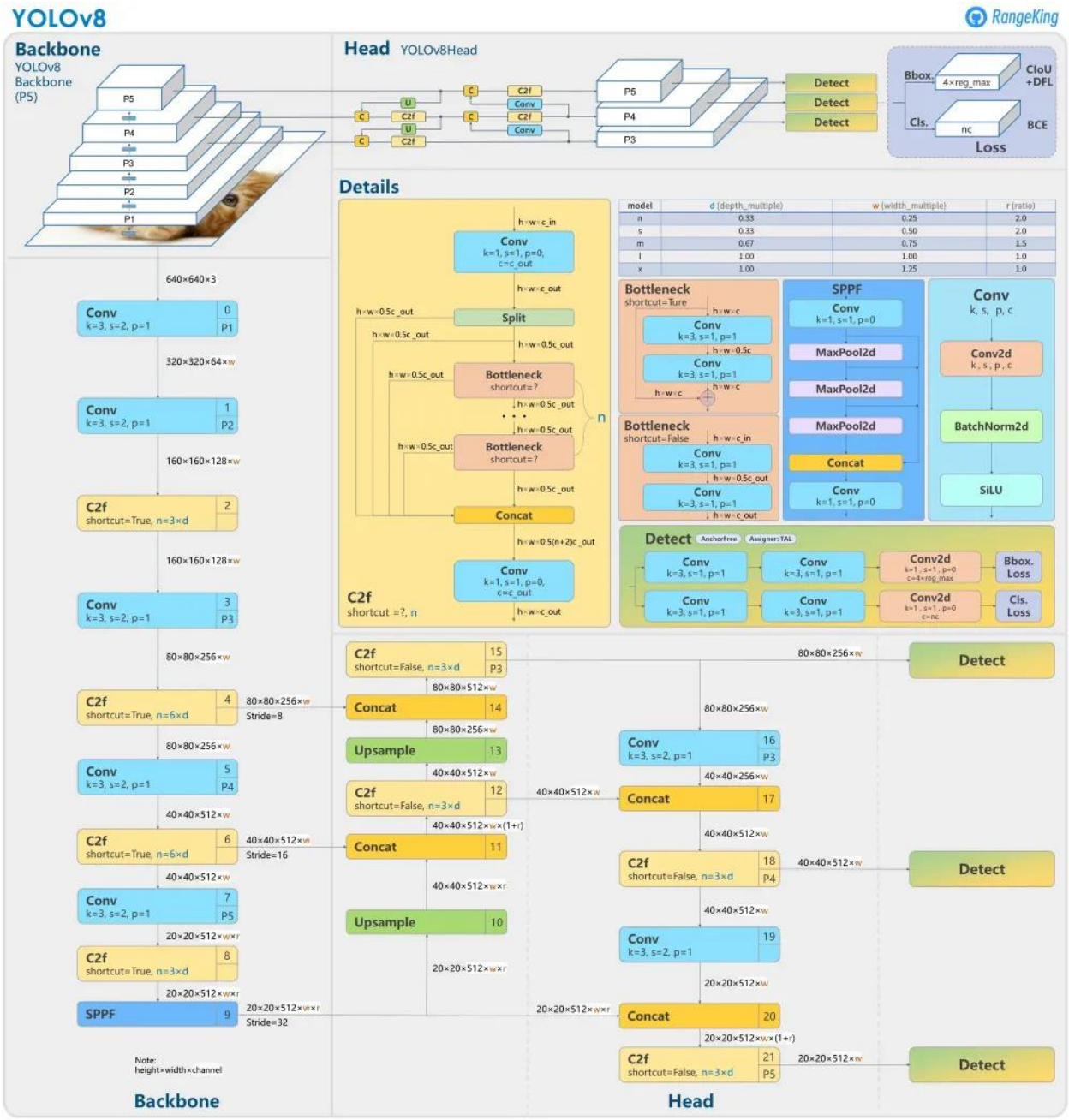

Figura 4-1. Arquitectura de la red YOLOv8 [11].

Como se ha mencionado anteriormente, la salida de la red YOLOv8 no es solo una predicción de la clase a la que pertenece la imagen. Su versatilidad radica especialmente en la capacidad de localizar objetos variables en tamaño dentro de una misma imagen y aportar un cuadro delimitador del mismo junto con la clase predicha. En la Figura 4-2 se muestra un fotograma de un video realizado para comprobar el funcionamiento del modelo realizado en este proyecto. En la imagen podemos ver los cuadros delimitadores junto con el grado de confianza en la predicción. Además, se muestra el ID de seguimiento del objeto en cuestión, como se explicará en la sección 6.

Figura 4-2. Ejemplo de la salida de la red YOLOv8.

4.1.2 Versiones

YOLOv8 presenta mejoras significativas respecto a sus versiones anteriores como YOLOv3, sin embargo, podemos encontrar distintas versiones de este modelo [11]. Las versiones varían tanto en escala como en rendimiento en tiempo real. Para este proyecto se ha escogido el modelo YOLOv8s por conseguir un equilibrio entre resultados y rendimiento aceptables. Aunque teóricamente puede llegar a generar la salida en tiempo real, para ello es necesario un hardware más potente que el utilizado.

Modelo	Tamaño (píxeles)	$Map^{VAL}_{50\text{-}95}$	Velocidad (ms/img)	Parámetros (M)
YOLOv8n	640	18.4	1.21	10.5
YOLOv8s	640	27.7	1.40	28.7
YOLOv8m	640	33.6	2.26	80.6
YOLOv8l	640	34.9	2.43	167.4
YOLOv8X	640	36.3	3.56	260.6

Tabla 4-1. Versiones y características del modelo YOLOv8 con la base de datos COCO [13] usando una GPU A100 TensorRT FP16.

4.2 Base de datos

Escoger una buena base de datos es una fase primordial para obtener un modelo neuronal óptimo. Si escogemos imágenes con condiciones uniformes en las que se puedan localizar fácilmente los objetos obtendremos un resultado muy cercano al 100% en el entrenamiento. Sin embargo, este modelo tendrá un funcionamiento muy pobre con imágenes externas a la base de datos o en condiciones reales [14].

Para este proyecto se ha escogido una base de datos de imágenes genéricas tomadas desde la parte delantera de un vehículo en condiciones variables de luminosidad y distancia. La base de datos, perteneciente a un conjunto de bases de datos de uso libre de la web Roboflow, consta de un total de 740 imágenes con un tamaño de 1360x800. Además, se ha utilizado este mismo entorno para gestionar las imágenes y detectar ciertas características.

Figura 4-3. Ejemplo de imagen segmentada tomada de la base de datos.

En total contamos con 1211 anotaciones de señales de tráficos (1.6 anotaciones por cada imagen) con un tamaño promedio de 1.09 Mpx. Las anotaciones están divididas en cuatro clases distintas: o*ther*, *prohibitory*, *danger* y *mandatory*. Como se ha explicado en la sección 3, se ha decidido segmentar tan solo en 4 clases distintas para simplificar el proceso de la red YOLO. Esto ayuda considerablemente a mejorar el resultado obtenido ya que a cada grupo pertenecen señales claramente diferenciadas en cuanto tamaño y forma.

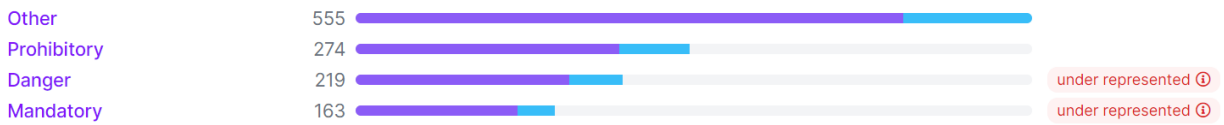

Figura 4-4. Balance de imágenes por clase. En morado: set de entrenamiento; en azul: set de validación.

En la imagen 4-4, obtenida del entorno de Roboflow, podemos comprobar que ciertas clases están mejor representadas que otras. Esto provocará que el modelo sea más preciso en cuanto a detectar las señales de este

grupo. Por este motivo se han escogido las señales de esta clase para entrenar la red ResNet-50 que comentaremos en la sección siguiente. Esta estadística también demuestra la complejidad que realizar un único modelo para detectar la señal exacta, ya que al aumentar el número de clases necesariamente habrá ciertos grupos con escasa representación.

Otra estadística clave aportada por Roboflow es un mapa de calor de la posición de todas las señales en la base de datos. Con esta gráfica podeos comprobar que las imágenes se ajustan a las tomadas desde la parte frontal de un vehículo ya que están más localizadas en las zonas intermedias de la imagen a ambos lados. Además, hay más representación en la parte derecha por ser el carril más utilizado durante la conducción. Por otra parte, al ejecutar un problema más complejo podría servirnos para hacer un preprocesado de la imagen recortando las zonas donde no se encuentran apenas ejemplos.

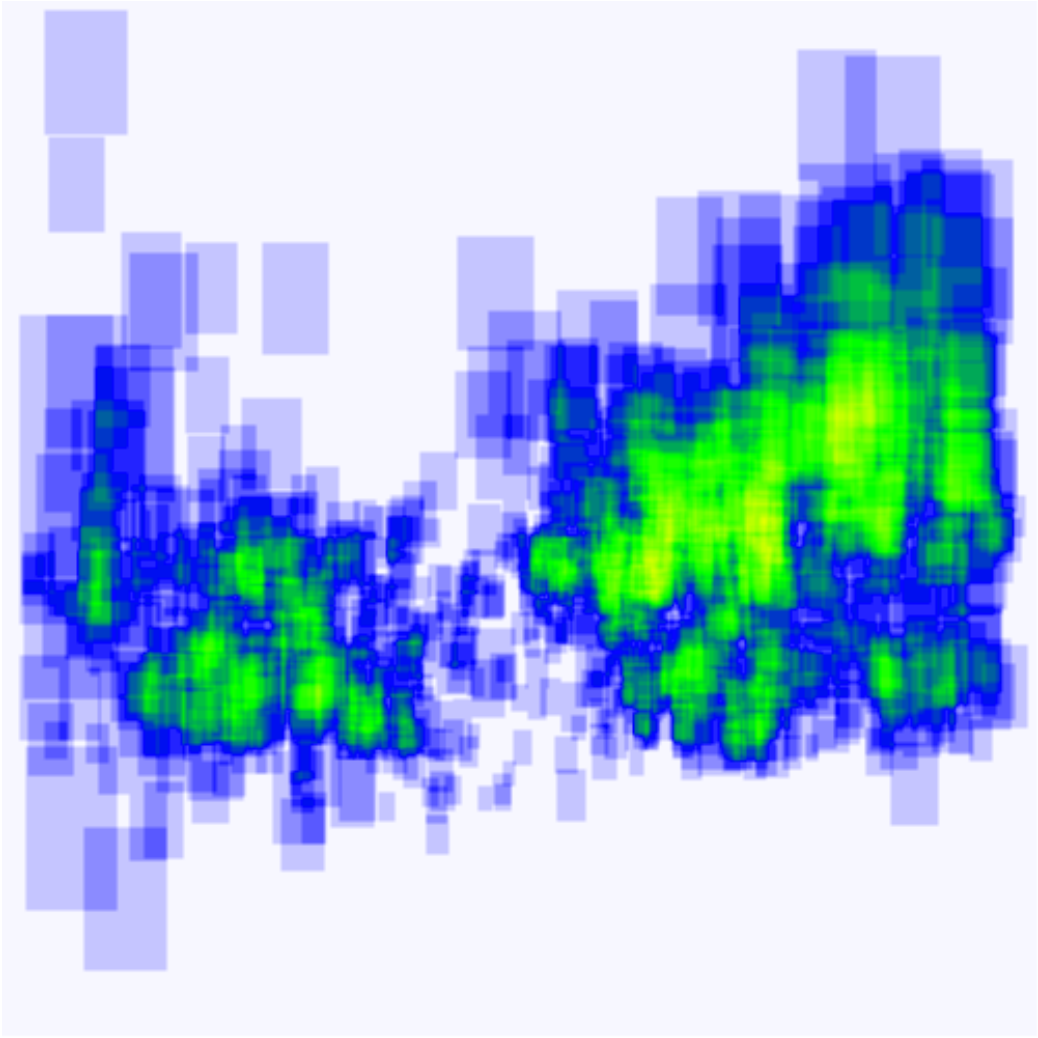

Figura 4-5. Mapa de calor de la posición de las señales en la base de datos.

4.3 Entrenamiento

En una primera aproximación al problema del entrenamiento de la red neuronal YOLO se intentó desarrollar en un entorno local mediante un Docker con la finalidad de no depender de herramientas externas. Sin embargo, debido a las limitaciones del hardware disponible (la duración del entrenamiento era del orden de días), se cambió el enfoque para realizar un entrenamiento en la nube. Con esta modificación el tiempo de entrenamiento se redujo a unas pocas horas.

El entorno en la nube escogido es Google Colab. Gracias a su versión premium tenemos acceso a más de 100 horas de uso de distintas GPUs. Estas GPUs, especializadas en tareas de inteligencia artificial y entrenamiento

de modelos neuronales, nos permiten reducir el tiempo de entrenamiento considerablemente. En concreto, se ha usado la GPU A100 [13] que nos permite un procesado 20 veces más rápido que la opción por defecto, Tesla T4.

```
                 from  n    params  module                                       arguments
  0                -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11           [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14           [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17          [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20           [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22      [15, 18, 21]  1   2117596  ultralytics.nn.modules.head.Detect           [4, [128, 256, 512]]
Model summary: 225 layers, 11137148 parameters, 11137132 gradients, 28.7 GFLOPs
```

Figura 4-6. Capas de la estructura YOLOv8s.

4.4 Resultados

Visualizar y entender los resultados aportados en el entrenamiento es un paso crítico a la hora de escoger el modelo que aporte mejores resultados a nuestro problema. Por ello, se ha implementado en Google Colab Weight & Biases, una herramienta de la web wand.ai que nos permite obtener métricas y estadísticas de cada entrenamiento. Esta plataforma nos permite realizar un seguimiento en tiempo real de la generación del modelo o ver resultados en imágenes de la base de datos. Además, nos muestra gráficas de consumo de energía, uso de la CPU, GPU y memoria RAM del sistema, aunque estas últimas no se van a desarrollaren este proyecto por no estar utilizando el sistema local.

Clase	Instancias	Precisión	Recall	mAP50	mAP50-95
Danger	46	0.991	0.957	0.966	0.840
Mandatory	30	0.954	0.833	0.901	0.783
Other	**111**	**0.991**	**0.982**	**0.994**	**0.870**
Prohibitory	61	0.993	0.915	0.973	0.851
Total	248	0.984	0.922	0.959	0.836

Tabla 4-2. Estadísticas del set de validación obtenidas de Gogle Colab separadas por clase.

En cuanto a las métricas obtenidas, debemos comprender qué representan y como debemos modificar los parámetros de entrenamiento para conseguir un modelo óptimo.

- **Intersección sobre Unión (IoU).** El IoU mide la precisión de la localización de los objetos, comparando la superposición entre las predicciones del modelo y las ubicaciones reales de los objetos. Un IoU bajo nos indica la existencia de problemas en la localización precisa de los objetos, lo que puede requerir ajustes en los métodos de predicción de cuadros delimitadores.

$$IoU = \frac{Área\ superpuesta}{Área\ de\ unión}$$

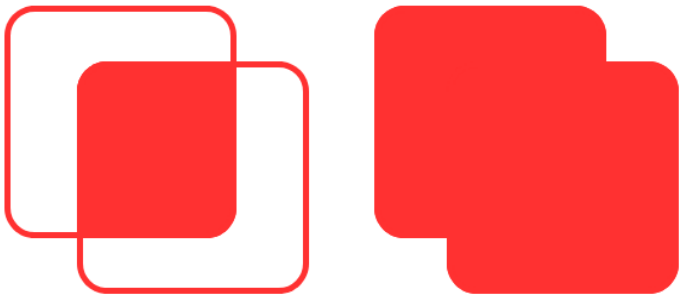

Figura 4-7. Izquierda: área superpuesta. Derecha: área de unión.

- **Precisión media (AP).** La precisión media por clase ayuda a identificar las clases con las que el modelo tiene más dificultades. La obtención de puntuaciones bajas en ciertas clases indica que requieren más datos específicos de esas clases o ajustes en los pesos durante el entrenamiento.
- **Precisión media promedio (mAP).** El mAP es una métrica que evalúa la precisión general del modelo. Se calcula promediando la precisión en varios puntos de recall. Un mAP bajo nos sugiere que el modelo necesita refinamientos generales en su capacidad de detección. El mAP50 está calculado con un umbral de intersección sobre unión (IoU) de 0.5, por lo que se considera una medida de la precisión considerando exclusivamente las detecciones más sencillas. El mAP50-95, por el contrario, ofrece una precisión medida con umbrales de 0.5 a 0.95 por lo que sus resultados aportan una visión más profunda del modelo.

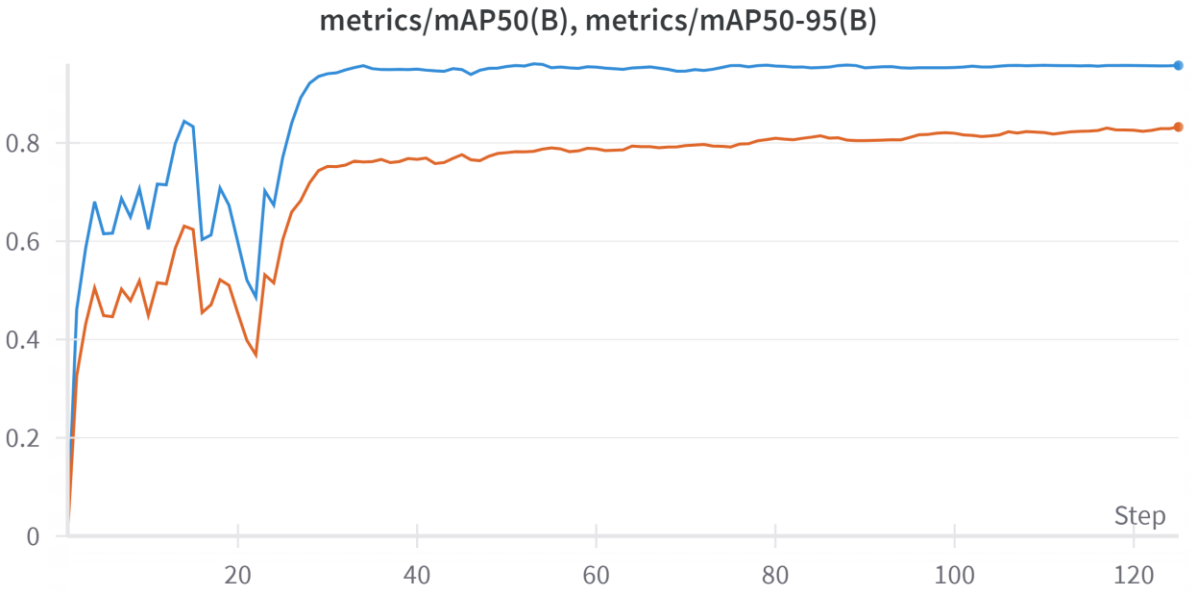

Figura 4-8. Azul: mAP50. En naranja: mAP50-95.

- **Precisión.** La precisión se enfoca en minimizar las falsas detecciones. Una precisión baja significa que el modelo está identificando demasiados objetos inexistentes. Podemos ajustar los umbrales de confianza para mejorar esta métrica.

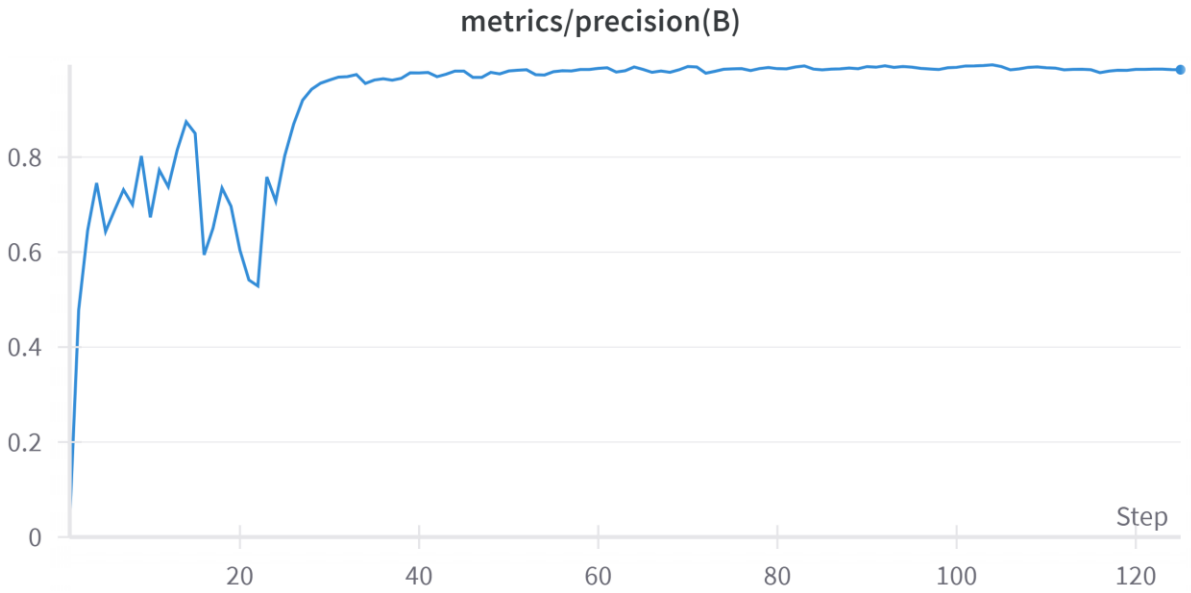

Figura 4-9. Resultado de la medida de la precisión.

- **Recall.** El recall mide la capacidad del modelo para detectar todas las instancias de un objeto. Un recall bajo sugiere que el modelo está pasando por alto objetos reales. Para mejorar esta métrica, puede ser necesario incrementar la calidad de la extracción de características o utilizar más datos de entrenamiento.

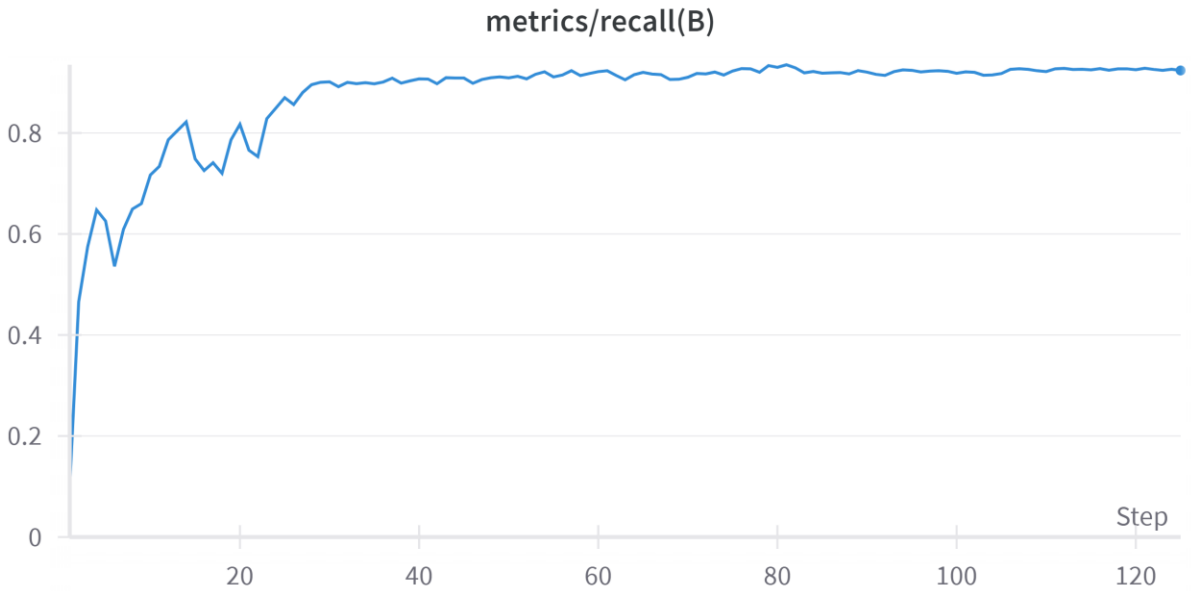

Figura 4-10. Resultado de la medida del recall.

Tras entrenar varios modelos y analizando los resultados obtenidos, podemos concluir que el modelo YOLOv8 entrenado presenta un rendimiento excelente tanto en precisión como en recall. Por otra parte, la mAP alta confirma la capacidad del modelo para detectar y localizar objetos con gran exactitud, lo que lo hace ideal para aplicaciones en entornos reales.

- **mAP50.** Podemos observar que esta métrica alcanza valores cercanos a 0.9, lo que indica un excelente desempeño en la detección de objetos con un umbral de IoU del 50%.
- **mAP50-95.** Esta métrica, que es más estricta, muestra valores cercanos a 0.8. Aunque es inferior a mAP50, sigue indicando un buen rendimiento general del modelo.
- **Precisión.** La precisión se estabiliza en torno a 0.9 después de algunas fluctuaciones iniciales, sugiriendo que el modelo es muy eficiente en minimizar falsas detecciones.
- **Recall.** La métrica de recall se estabiliza alrededor de 0.9, indicando que el modelo es capaz de detectar la mayoría de las instancias de los objetos en las imágenes.

Por último, otra gráfica necesaria para realizar un análisis exhaustivo de los resultados es la matriz de confusión. En esta matriz, cada fila representa las instancias de una clase verdadera, mientras que cada columna representa las instancias de una clase predicha. Esto permite visualizar no solo el desempeño del modelo en términos de aciertos (diagonal principal), sino también los errores cometidos (fuera de la diagonal principal), proporcionando una visión detallada de cómo el modelo clasifica cada categoría.

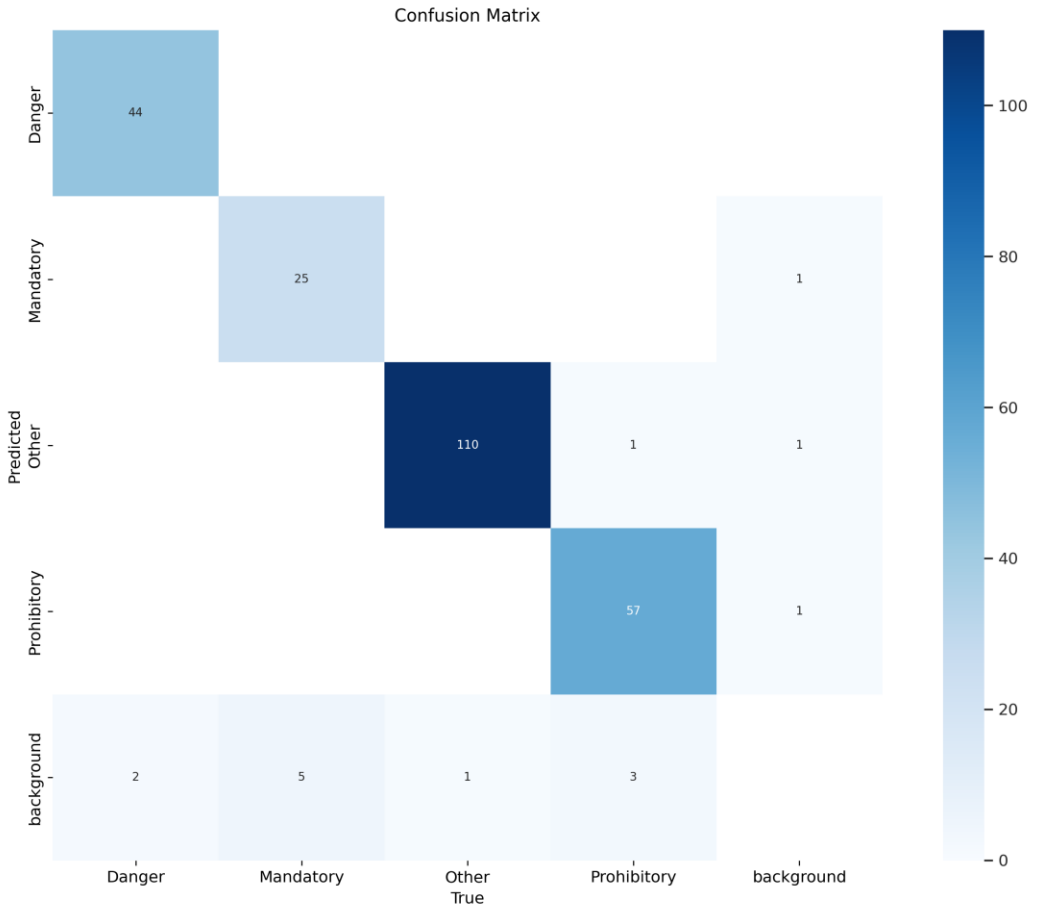

Figura 4-11. Matriz de confusión.

- El modelo tiene un buen rendimiento general en la clase *other*, con 110 aciertos y muy pocos errores. Este resultado es fundamental para el proyecto ya que as imágenes obtenidas de esta clase son las que vamos a utilizar para realizar el posterior proceso de clasificación.

- Las clases *danger* y *prohibitory* también muestran un rendimiento aceptable, con 44 y 57 aciertos respectivamente, aunque hay algunos errores menores. Aun así, podría mejorarse el resultado aumentando la base de datos.
- La clase *mandatory* muestra un rendimiento moderado, con 25 aciertos y algunos errores. Al igual que en el caso anterior, esto podría resolverse con una ampliación de la base de datos.
- La clase *background* presenta una mayor cantidad de errores, indicando que el modelo a veces confunde objetos con el fondo. Al estar tratando con imágenes generalistas de la vista frontal de un vehículo este resultado era esperable. Al realizar el seguimiento de objetos (tracking) se mejorará parcialmente en este aspecto ya que una vez detectado un objeto es más sencillo seguir su recorrido en la secuencia de imágenes.

Finalmente, podemos comprobar los resultados obtenidos en distintas imágenes del set de validación.

Figura 4-12. Resultado del entrenamiento probado sobre un batch del set de validación.

5 MODELO RESNET-50

En este capítulo describiremos el modelo utilizado en la clasificación de las señales de tráfico de una misma categoría, ResNet-50. Se desarrollarán los motivos por los que se ha seleccionado esta arquitectura. Se explicará en detalle cómo se ha entrenado el modelo y la base de datos utilizada para ello.

5.1 Fundamentos teóricos

Las redes neuronales residuals (ResNet) son un tipo de arquitectura de red neuronal convolucional profunda creada en 2015 por Kaiming He, Xiangyu Zhang, Shaoqing Ren, y Jian Sun, ingenieros de la división asiática de Microsoft. Tras mejorar los resultados frente a bases de datos muy conocidas como ImageNet detection, ImageNet localization [16], COCO detection, o COCO segmentation se ha convertido en una de las arquitecturas más utilizadas por su buena generalización en tareas de reconocimiento visual.

La arquitectura de las redes residuales aborda un problema que encontramos habitualmente en redes convolucionales profundas: el desvanecimiento del gradiente en redes muy profundas, lo que provoca un déficit en los resultados del entrenamiento. La clave de ResNet-50 radica en su uso de bloques residuales, que permiten la formación de redes muy profundas. Estos bloques incluyen conexiones de acceso directo que saltan una o más capas, ayudando a preservar el flujo de gradiente a través de la red. Este enfoque facilita el entrenamiento de redes con muchas más capas que las arquitecturas anteriores.

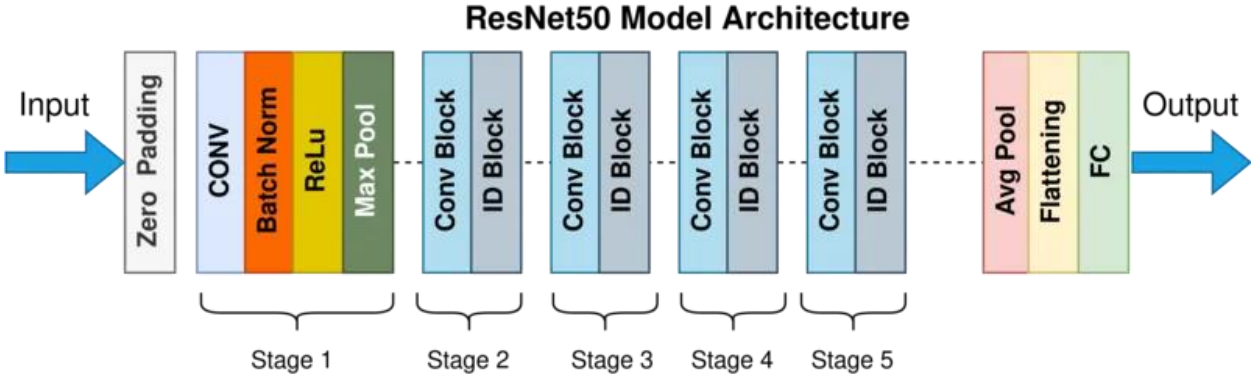

Figura 5-1. Arquitectura del modelo neuronal ResNet-50 .

La arquitectura de ResNet-50 se compone de una serie de bloques residuales, cada uno de los cuales contiene varias capas convolucionales. La red comienza con una capa convolucional inicial seguida de una capa de agrupamiento (pooling), y luego pasa por una serie de bloques residuales organizados en cuatro etapas

principales, cada una con un número creciente de filtros.

Cada bloque residual en ResNet-50 incluye dos o tres capas convolucionales y una conexión de acceso directo que suma la entrada del bloque a su salida. Además, ResNet-50 utiliza una combinación de capas de convolución, normalización por lotes (batch normalization), y funciones de activación ReLU, lo que ayuda a mejorar la estabilidad y el rendimiento del modelo.

La red finaliza con una capa completamente conectada (fully connected) seguida de una capa de softmax para la clasificación de las imágenes en múltiples categorías. ResNet-50 es ampliamente utilizada no solo por su alta precisión, sino también por su capacidad para ser adaptada y extendida a diferentes aplicaciones de visión por computadora.

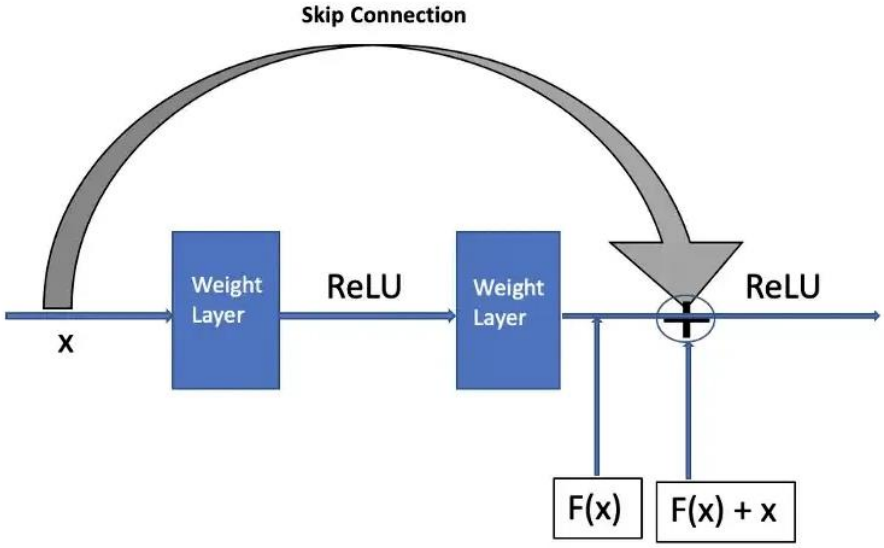

Figura 5-2. Diagrama de conexiones de capa residual.

5.2 Base de datos

El entrenamiento de una red profunda como ResNet-50 requiere de una base de datos amplia y convenientemente clasificada para conseguir un modelo óptimo. Para este proyecto se ha escogido una base de datos con señales alemanas, German Traffic Sign Detection Benchmark (GTSDB) [16]. Este modelo cuenta con más de 50.000 imágenes divididas en 43 clases. Para nuestro modelo se tendrán en cuenta cerca de 13.000 imágenes pertenecientes a 13 clases. Estas imágenes se han dividido utilizando un 70% en el set de entrenamiento, un 20% en el de validación y un 10% para el set de test.

- Límite de velocidad: 20, 30, 40, 50, 60, 70, 80, 100, 120 km/h.
- Prohibición de adelantamiento.
- Prohibición de adelantamiento dirigida a camiones.
- Acceso prohibido a camiones.
- Circulación prohibida.

Figura 5-3. Ejemplos de cada clase de la base de datos.

Un requisito impuesto al comienzo de este proyecto era crear los modelos capaces de detectar y clasificar señales de carreteras españolas para poder comprobar los resultados mediante experimentos de conducción reales. Aunque esta base de datos no cuenta con señales españolas, gracias a la normativa europea gran parte de las señales son similares. Como se menciona en la sección 3 se van a generalizar los resultados obtenidos para una sola clase de los resultados de YOLOv8, por lo que se ha escogido la clase *others* que consta casi en exclusiva de señales de tráfico de velocidad idénticas a las nacionales.

Figura 5-4. Ejemplo de señales de la base de datos en condiciones de resolución y luz variables.

Las imágenes de la base de datos presentan diversas condiciones de luz, desenfoque de movimiento u ocultamiento parcial de la señal. Estas características proporcionan una mejor generalización del modelo y nos permite poder aplicarlo a situaciones en condiciones reales como se verá en la sección 7.

5.3 Entrenamiento

El entrenamiento del modelo ResNet-50 se ha desarrollado en el entorno de Spyder. Un entrenamiento en la nube similar al realizado con la red YOLOv8 habría reducido notablemente los tiempos de entrenamiento. Debido a las limitaciones de espacio de almacenamiento en la nube y otras particularidades del entorno de Google Colab se ha escogido la ejecución local.

Para el entrenamiento se ha utilizado la librería de Tensorflow de Keras implementándose en Python una función personalizada. Como se ha mencionado, la arquitectura de la red ResNet-50 está preestablecida y estudiada, aunque para el caso que nos concierne se ha modificado la capa de salida para adecuarla a las 13 clases que debemos clasificar.

Se ha estudiado realizar un entrenamiento desde cero, aunque tras comparar los resultados se ha decidido iniciar los pesos de la red según el entrenamiento con la base de datos ImageNet [18]. Con esto queremos conseguir alcanzar mejores resultados con un número no demasiado elevado de entrenamientos sin provocar sobreentrenamiento *(*overfitting) [19]. El overfitting es un problema recurrente en el entrenamiento de redes convolucionales y especialmente preocupante en nuestro caso ya que las clasificaciones finales se realizarán en condiciones reales. Sin embargo, al tener una base de datos lo suficientemente amplia no se ha producido sobre overfitting.

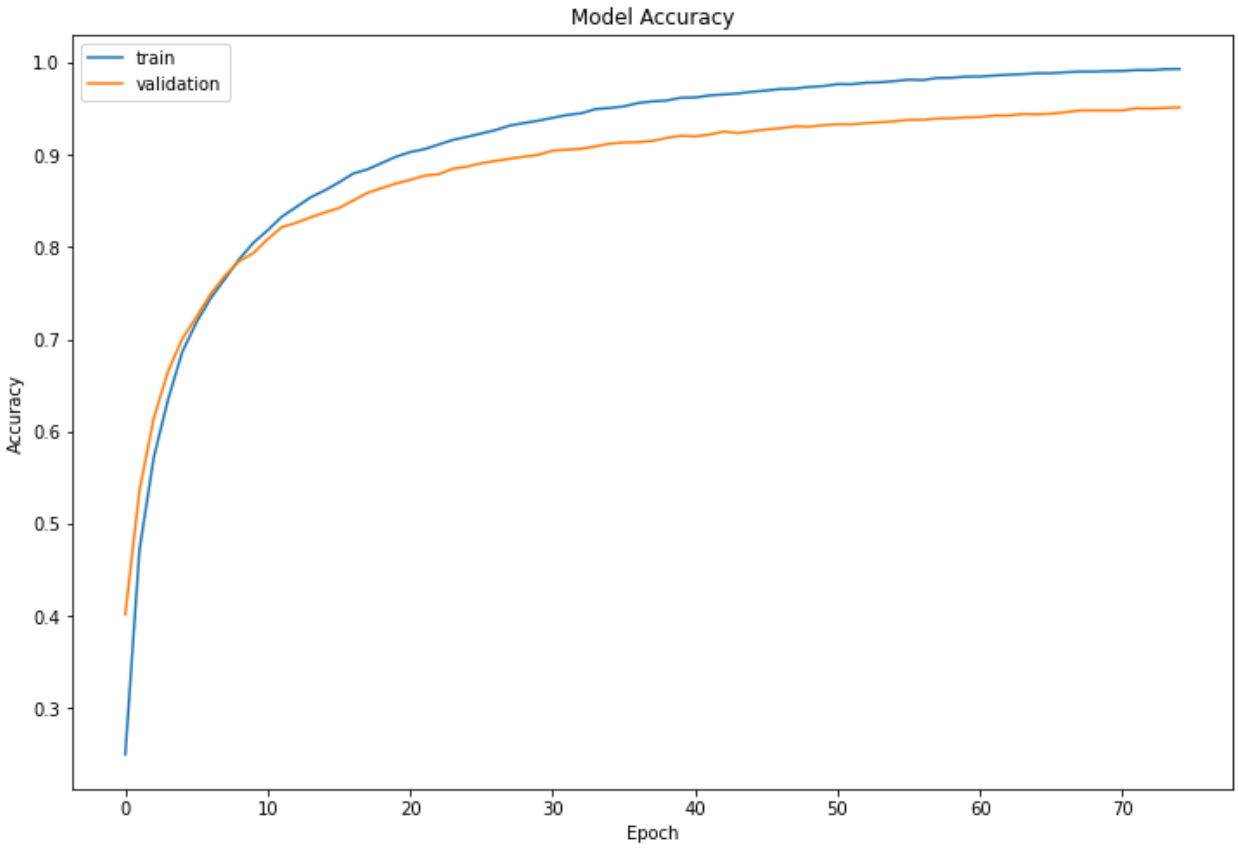

Figura 5-5. Curva de precisión del entrenamiento.

La figura 5-5 muestra la precisión del modelo durante el entrenamiento y la validación a lo largo de las 75 épocas. Podemos apreciar como la precisión del entrenamiento (línea azul) aumenta de manera constante y alcanza un valor cercano a la unidad, lo que nos indica que el modelo ha conseguido aprender de manera correcta las muestras correspondientes al conjunto de entrenamiento. La precisión de validación (línea naranja) también muestra una mejora constante, aunque se estabiliza alrededor de 0.92-0.94.

El hecho de que la precisión de validación sea un poco menor que la del entrenamiento sugiere que hay un ligero sobreajuste, es decir, el modelo se adapta muy bien a los datos de entrenamiento, pero presenta una

ligera pérdida de precisión con los datos de validación. Este hecho es común en redes profundas y puede mitigarse con técnicas adicionales como la regularización en las condiciones de las imágenes o la ampliación del conjunto de datos de validación.

5.4 Resultados

En la figura 5-6 podemos ver el resultado de aplicar el modelo creado sobre un lote de imágenes aleatorias de la misma base de datos correspondientes al set de test. Las imágenes de este set, al igual que las del resto de la base de datos, presentan serias dificultades de iluminación resolución y ocultamiento parcial de las señales por sombras o ramas de árboles. Sin embargo, al compartir estas dificultades con el resto de la base de datos se obtienen buenos resultados.

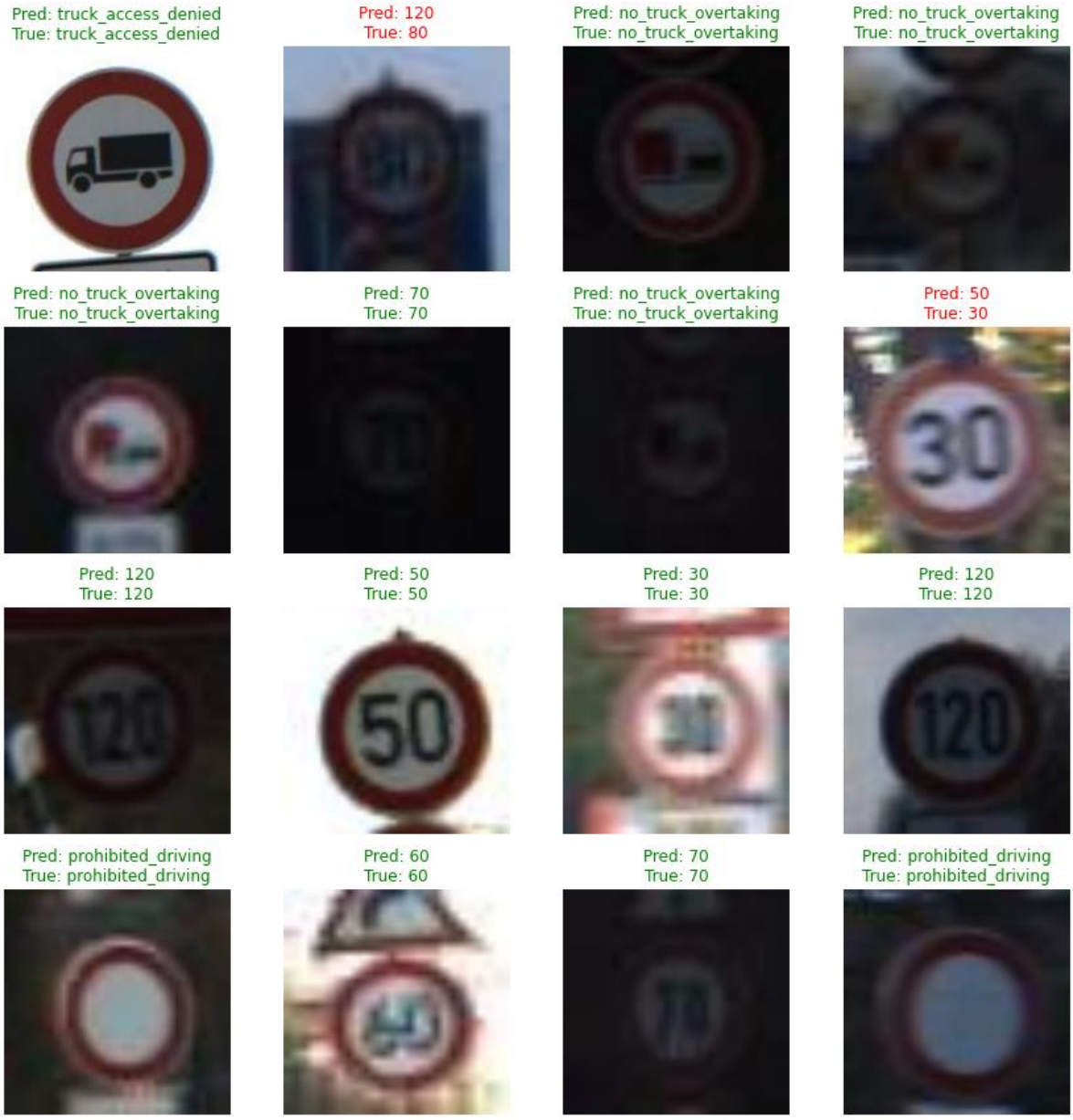

Figura 5-6.

Tras analizar los datos obtenidos en el set de test podemos deducir que ciertas clases presentan más tasa de

fallo debido a su menor representación (en cantidad y calidad) en la base de datos. Aunque el resultado general es realmente bueno, es esperable que la precisión de este modelo disminuya al probarse rente a imágenes reales tomadas desde un vehículo en movimiento.

6 SEGUIMIENTO DE OBJETOS

En los capítulos anteriores hemos conseguido detectar y, posteriormente, clasificar las señales de tráfico de una base de datos concreta. Para obtener los resultados hemos aplicado los modelos a las imágenes. Sin embargo, la utilidad real de esta aplicación pasa por la capacidad de analizar video, especialmente en tiempo real. Para conseguir el segundo punto debemos centrarnos más en la optimización del código y la actualización del hardware de trabajo, pero para analizar video no podemos limitarnos a analizar cada fotograma individual, debemos ir un paso más allá y hacer un seguimiento de los objetos detectados [20].

6.1 Problemas iniciales

Como se ha visto, gracias al modelo YOLOv8 somos capaces de detectar la zona de interés de la imagen donde aparecen ciertos objetos (Región of Interest, ROI). Ahora debemos asignarle un ID concreto a cada objeto detectado y que lo mantenga a lo largo del tiempo, es decir, reconocer el mismo objeto en distintos fotogramas. Para ello, debemos enfrentarnos a distintos varios problemas:

- Efectos visuales. Las condiciones ambientales como la luminosidad, el ruido o el desenfoque por movimiento dificultan el reconocimiento, por lo que la clase asignada a un elemento podría variar de un fotograma a otro de forma errónea.
- Oclusión. Un objeto detectado puede salir momentáneamente de la imagen o quedar oculto (de forma total o parcial) por otro elemento.
- Superposición de objetos. Uno o más objetos que deben ser detectados pueden quedar ocultos por otro de forma que solo se reconozca este último. En nuestro caso concreto esto no supone un problema porque en una experiencia de conducción real solo podríamos fijarnos en la señal que superpone a las demás, aunque se tendrá en cuenta para realizar el seguimiento.
- Transformación de un objeto. Se produce cuando las características del objeto reconocido cambian a lo largo del tiempo. En nuestro caso cambiará la inclinación horizontal de la señal a medida que nos acercamos a ella.

6.2 Algoritmos de seguimiento de objetos

Para solucionar estas cuestiones se han desarrollado diversos algoritmos de seguimiento de objetos para mantener la identificación en movimiento en secuencias de video. Inicialmente, el objetivo era identificar una clase de objeto y rastrear su localización. Inicialmente, el objetivo era identificar una clase de objeto y rastrear su localización.

Con el aumento del poder de cómputo, ahora podemos realizar seguimiento de objetos múltiples con identificación fotograma a fotograma, detectando diversas clases de objetos y manteniendo su identificación a lo largo del tiempo [21]. Algunos de los algoritmos más utilizados son SORT [22], DeepSort [23] o ByteTrack [24], entre otros.

- SORT. El algoritmo SORT (Simple Online Realtime Tracking) se basa en la posición y el tamaño de las cajas delimitadoras, prediciendo la trayectoria del objeto mediante una velocidad constante. DeepSort mejora considerablemente este enfoque añadiendo información de una red neuronal profunda que debe ser entrenada. Otras variaciones de este algoritmo, como StrongSort [25], realizan modificaciones en las funciones de coste para mejorar los resultados
- ByteTrack. Utiliza además la intersección sobre unión (IoU) en su algoritmo de detección. No solo tiene en cuenta el valor asociado a la zona detectada, también usa los valores de alta y baja confianza. El resultado mejora de forma considerable frente a situaciones complejas con ocultamiento de objetos en el seguimiento. Además, presenta una mayor eficiencia consiguiendo mayor número de fotograma por segundo que la competencia.

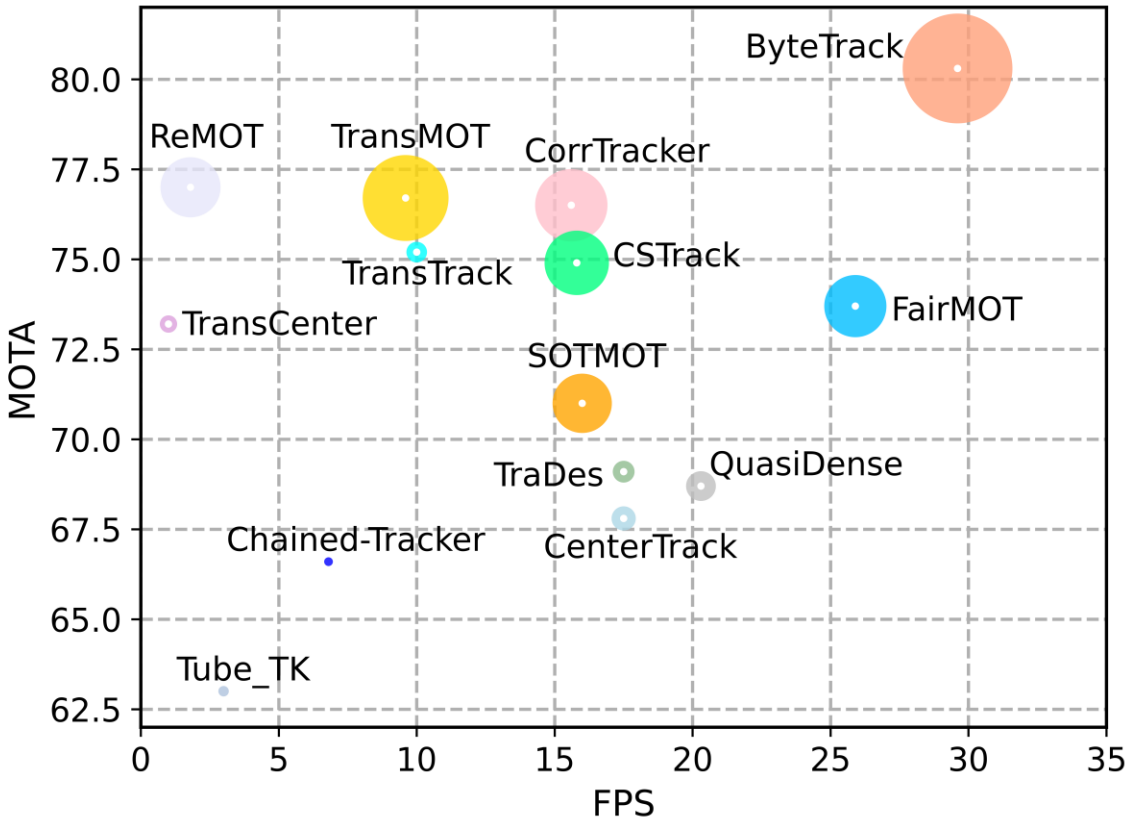

Figura 6-1. Resultados del MOT (Multi-Object Tracking) utilizando frente a la base de datos MOT17 del MOTChallenge.

- Bot-SORT. Este algoritmo, creado por Ultralytics, es una variación del algoritmo ByteTrack junto con ReID. Bot-SORT mejora la velocidad de cálculo y los resultados ligeramente. Es el modelo por defecto utilizado en la librería de YOLOv8, aunque se le han realizado modificaciones para personalizar los resultados a nuestra base de datos.

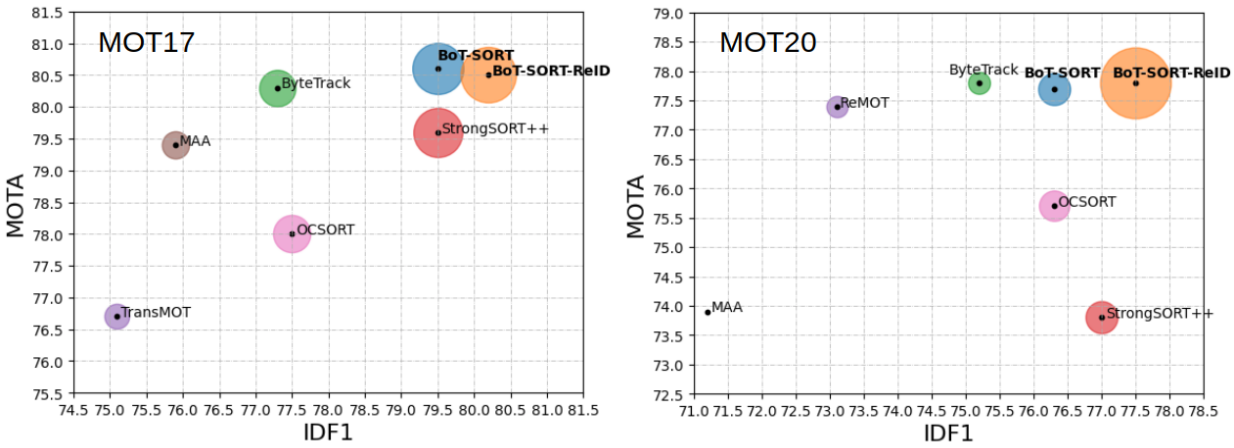

Figura 6-2. Resultados del MOT (Multi-Object Tracking) frente a la base de datos MOT17 y MOT20 del MOTChallenge [26].

6.3 Modificación del algoritmo

Para entender las mejoras implementadas en el algoritmo Bot-SORT, es esencial comprender el funcionamiento interno de ByteTrack. ByteTrack, el núcleo del cual deriva Bot-SORT, inicia su proceso con entradas que consisten en una secuencia de video, un detector de objetos (como YOLO) y un umbral de confianza. La salida esperada son las trayectorias de los objetos en el video, inicialmente vacías. En cada fotograma del video, el detector de objetos predice las cajas de detección y sus puntuaciones, que luego se dividen en dos grupos: detecciones de alta y baja puntuación, basadas en el umbral predefinido.

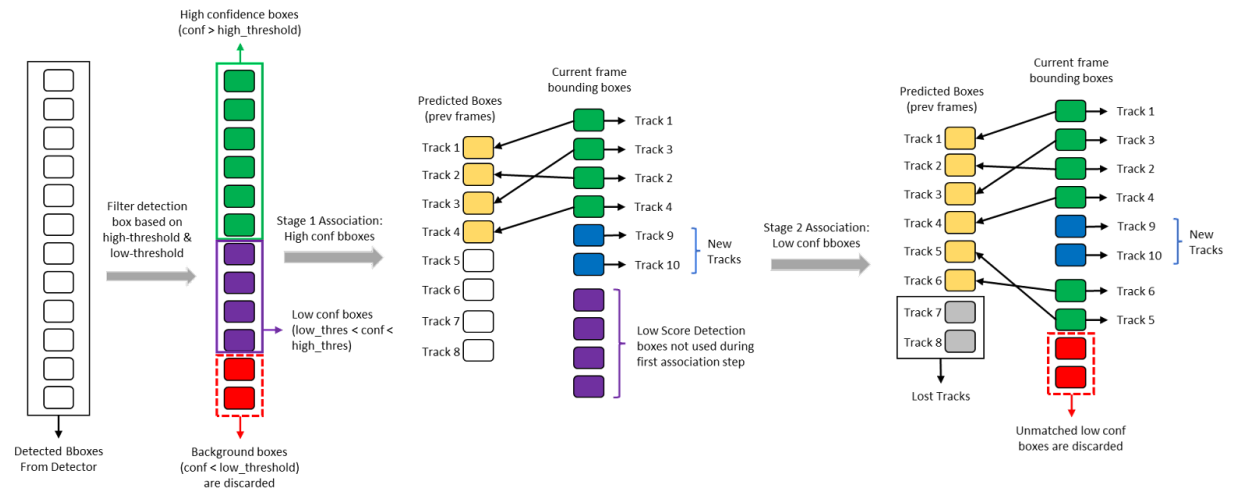

Figura 6-3. Diagrama del algoritmo ByteTrack.

Una vez clasificadas las detecciones, el filtro de Kalman predice las nuevas posiciones de las trayectorias en el fotograma actual. La primera asociación empareja las detecciones de alta puntuación con todas las trayectorias, incluidas aquellas que han sido perdidas, es decir, aquellas cuya presencia no se ha detectado en el nuevo fotograma del video. Las detecciones y trayectorias que no se emparejan se almacenan temporalmente en memoria.

En la segunda asociación, se intentan emparejar las detecciones de baja puntuación con las trayectorias restantes de la primera fase. Las trayectorias que aún no encuentran un emparejamiento permanecen almacenadas, mientras que las detecciones de baja puntuación no emparejadas se descartan. Las trayectorias que no se emparejan durante 30 fotograma consecutivos se eliminan definitivamente.

Finalmente, se crean nuevas trayectorias a partir de las detecciones de alta puntuación no emparejadas de la primera asociación. Las asociaciones pueden basarse en métodos de "ubicación" o "características". La innovación clave de ByteTrack radica en su capacidad para utilizar tanto detecciones de alta como de baja confianza para mantener las trayectorias a lo largo del tiempo.

El algoritmo Bot-SORT modifica ligeramente la manera en que se asocian las detecciones de baja y alta confianza, sin embargo, el núcleo central del algoritmo se mantiene estable. Para caracterizar el seguimiento de objetos se han modificados los límites establecidos para ciertas características de forma manual hasta dar con un resultado óptimo para nuestro sistema. Los límites modificados son:

- Valor mínimo para la primera asociación.
- Valor mínimo para la segunda asociación.
- Valor mínimo para iniciar un nuevo seguimiento si no se detecta ninguna asociación.
- Número de fotogramas máximo mantenemos un seguimiento sin emparejar.

7 RESULTADOS EN CONDICIONES REALES

En esta última sección analizaremos los resultados obtenidos en el sistema completo realizado durante el proyecto. Comprobaremos los resultados tanto del modelo YOLO como del modelo ResNet-50 para imágenes y video. Finalmente, analizaremos posibles caminos de mejora y las conclusiones del proyecto

7.1 Consideraciones iniciales

Para estudiar los resultados con precisión debemos tener en cuenta las dos etapas en las que hemos dividido el proyecto. En primer lugar, comprobaremos el correcto funcionamiento del sistema de detección y reconocimiento de YOLOv8 en condiciones reales. Es fundamental conseguir buenos resultados en este apartado ya que los objetos que no sean detectados ahora no acceden a la posterior clasificación de la red ResNet-50. Finalmente estudiaremos el resultado del sistema completo, analizando el resultado de la red ResNet-50. Por otra parte, se analizará el funcionamiento del seguimiento de objetos en tiempo real, aunque, para analizarlo correctamente es importante ver la secuencia de video completo.

Los resultados se extrapolarán de las grabaciones de video de la Tabla 7-1, donde se han utilizado dos tipos de cámara. A los videos se les ha aplicado

Video	YOLOv8	ResNet-50	Bot-SORT	Localización	Estabilizador
01	Sí	No	No	Autovía AP-4	No
02	Sí	Sí	Sí	Autovía AP-4	No
03	Sí	No	Sí	Autovía A-381	No
04	Sí	Sí	Sí	Autovía A-381	No

Tabla 7-1. Grabaciones realizadas en condiciones reales para el análisis de los modelos.

Como se ha mencionado en apartados anteriores, las condiciones climáticas no siempre ayudan al buen funcionamiento del sistema. Además, pueden darse situaciones de mayor complejidad como la conducción nocturna, con lluvia o con reflejos de luz. En la Figura 7-2 podemos apreciar como unas malas condiciones ambientales, en este caso reflejos de luz en la señal de tráfico, provocan un error en la clasificación. El modelo YOLOv8 detecta y clasifica la señal correctamente en la clase *others* pero el modelo ResNet-50 la clasifica como una señal de límite de velocidad a 30km/h siendo de 70 km/h. Podemos apreciar también como la confianza del resultado es de un valor muy bajo (0.147).

Figura 7-1. Ejemplo de fotograma del video de prueba 04 con condiciones climáticas óptimas.

Figura 7-2. Ejemplo de fotograma del video de prueba 04 con reflejos de luz.

7.2 Detecciones y clasificación con YOLOv8

Para analizar los resultados obtenidos con el modelo YOLOv8 hay que tener en cuenta que se trata de un modelo de clasificación y detección. Los errores en la clasificación son fácilmente detectables, pero los errores en la detección deben rastrearse de forma manual. Además, es necesario mencionar que un error en la detección se conservará durante todo el proceso de clasificación tanto ya que dicho objeto no será procesado por el modelo ResNet-50.

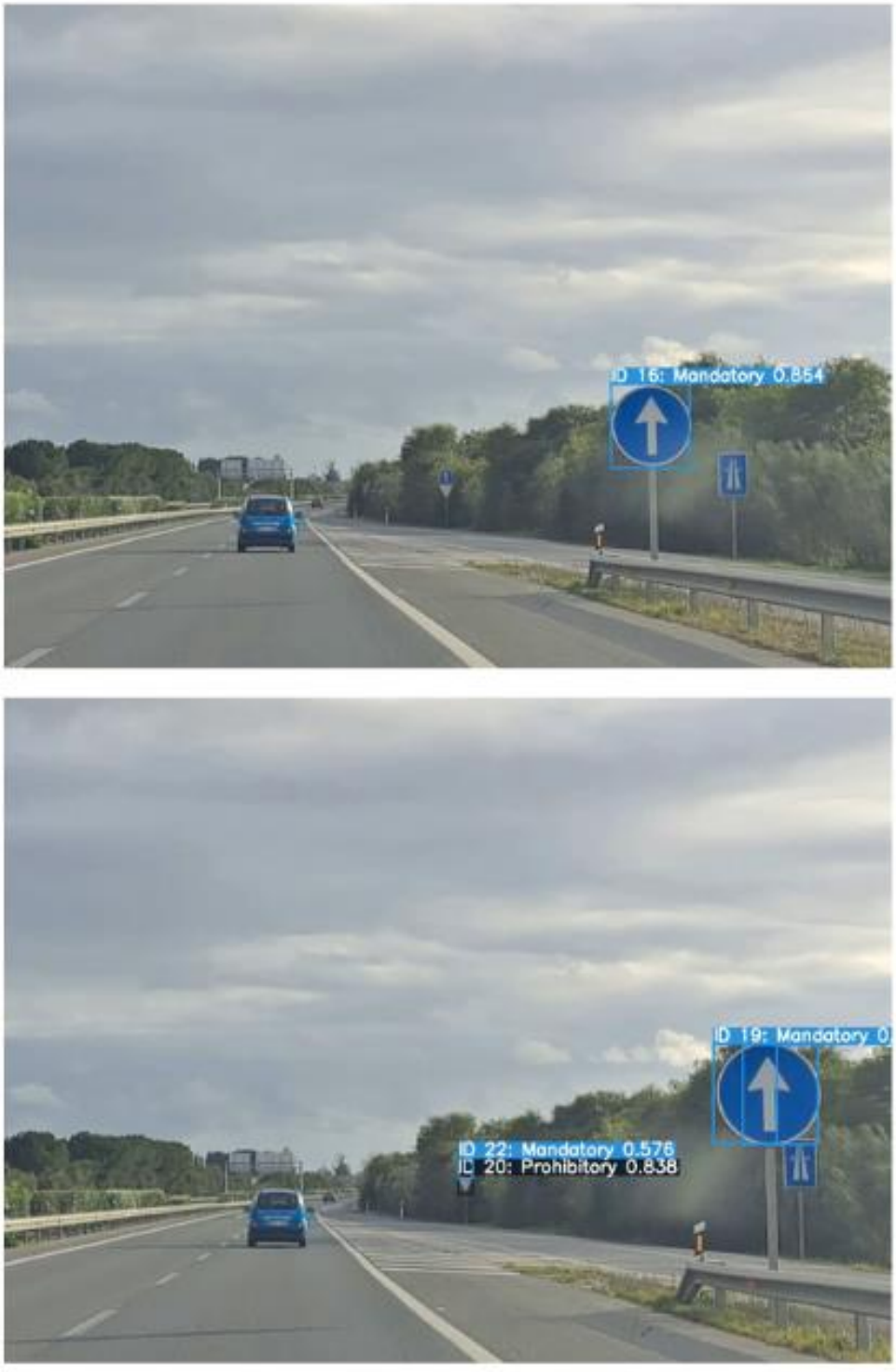

Figura 7-3. Ejemplo de fotograma del video de prueba 01. Arriba: YOLOv8 no detecta las señales más alejadas o localizadas en otro carril. Abajo: en el fotograma siguiente detecta correctamente las señales alejadas.

Tras visualizar los resultados del procesamiento de los videos podemos declarar que el modelo YOLOv8 presenta un resultado con un porcentaje de aciertos muy alto. Los fallos encontrados se localizan casi exclusivamente en señales de tráfico lejanas o situadas en otros carriles, aunque cuando se encuentran a menor distancia las acaba detectando. También se encontraron problemas a la hora de detectar señales muy cercanas al vehículo debido a la inclinación de estas respecto al observador. Sin embargo, tras añadir proceso de seguimiento de objetos, este problema se solucionó. Esto se debe a que el valor límite necesario para detectar un objeto ya detectado anteriormente es inferior al de una nueva detección.

El apartado de clasificación no presenta apenas fallos evidentes. Al estar clasificando solo 4 clases con características de color y forma claramente diferenciadas el resultado es óptimo.

Figura 7-4. Ejemplo de fotograma del video de prueba 01. Señales con detección y clasificaciones correctas (YOLOv8)

7.3 Clasificación con ResNet-50

Al analizar los errores cometidos por el modelo ResNet-50 debemos tener en cuenta ciertos criterios:

- Los fallos cometidos (por la no detección o clasificación errónea) en el apartado del modelo YOLOv8 arrastran a este paso, a una señal que no haya sido detectada no se le podrá aplicar el modelo ResNet-50.
- Debemos tener en cuenta que la clasificación de 4 clases con características físicas diferenciadas es más simple que la clasificación de 13 clases de señales parecidas en color y forma. En este caso, el modelo tendrá que clasificar señales basándose en los números lo que implica una mayor complejidad.

Teniendo estos criterios en cuenta, podemos decir que el modelo ResNet-50 presenta buenos resultados generales, aunque con ciertas deficiencias y clasificaciones cruzadas entre algunas clases. Los ejemplos más claros se aprecian entre las clases de límite de velocidad a 50 y 60 km/h y entre 40 y 70 km/h debido a la similitud entre los números. Este modelo, al igual que el modelo anterior, funciona peor en condiciones más extremas como con mucha luz o inclinación.

Figura 7-5. Ejemplo de fotograma del video de prueba 01. Señal con detección y clasificación correctas. (YOLOv8, ResNet-50)

Figura 7-6. Ejemplo de fotograma del video de prueba 01. Señal con detección y clasificación correctas (YOLOv8) pero clasificación secundaria errónea. (ResNet-50)

7.4 Seguimiento de objetos

El algoritmo de seguimiento de objetos escogido es uno de los más utilizados y depende en gran medida de la fiabilidad del modelo de detección. Como se ha visto en el apartado 7.2, el modelo YOLOv8 presenta resultados óptimos por lo que es esperable que el seguimiento de objetos se realice de manera correcta.

El único fallo que puede apreciarse es en la detección inicial de objetos. En ocasiones, detecta el mismo elemento en distintos fotogramas, pero debido a la baja confianza de la muestra no le realiza un seguimiento eficaz. Esto se podría solucionar bajando el límite necesario para iniciar el seguimiento, pero entonces provocamos falsas detecciones. Como este caso solo ocurre durante los primeros fotogramas en los que aparece el objeto y en condiciones de poca visibilidad, concluimos que no supone ningún problema de implementación real.

Como se ha podido ver en imágenes anteriores, sobre cada detección aparece un ID numérico que nos indica el número de seguimiento del objeto en cuestión. Para poder apreciar mejor este resultado es conveniente visualizar los videos en el repositorio de GitHub de este proyecto [27].

8 CONCLUSIONES

8.1 Conclusiones

El sistema de detección y reconocimiento de señales de tráfico en dos pasos ha logrado implementar y evaluar un sistema robusto para la identificación y clasificación de señales de tráfico mediante el uso de redes neuronales con variaciones de luminosidad, tráfico y otras perturbaciones de la imagen. A lo largo del desarrollo, se han abordado diversas fases que van desde la recopilación y preprocesamiento de datos hasta la implementación y evaluación de los modelos en entornos reales

En primer lugar, se ha demostrado que la combinación de YOLOv8 y ResNet-50 es eficaz para la localización y clasificación de señales de tráfico en imágenes capturadas desde la parte frontal de un vehículo. YOLOv8 ha proporcionado una detección rápida y precisa de las señales, mientras que ResNet-50 ha refinado esta detección clasificando las señales en subclases específicas con una alta precisión. Los resultados obtenidos, aunque presentan margen de mejora, se encuentran dentro de los límites impuestos al comienzo de este proyecto consiguiendo reconocer un total de 13 clases de señales de forma exacta.

Además, el uso de plataformas de computación en la nube, como Google Colab, ha permitido optimizar el tiempo de entrenamiento de los modelos, beneficiándose de la capacidad de procesamiento de GPUs avanzadas. Esto ha sido de vital importancia para manejar las grandes cantidades de datos y la complejidad computacional que requiere el entrenamiento de redes neuronales profundas avanzadas.

Se ha comprobado la robustez del sistema en condiciones diversas de iluminación y resolución, lo que sugiere su aplicabilidad en entornos reales de conducción. Las pruebas realizadas con videos en condiciones reales han mostrado que el sistema puede mantener un rendimiento alto, algo esencial para su implementación en sistemas de asistencia a la conducción y vehículos autónomos.

Finalmente, aunque se han obtenido resultados satisfactorios, durante el proyecto también han identificado áreas de mejora que se expondrán en el siguiente apartado.

8.2 Posibles mejoras

Entre las posibles mejoras encontramos:

- Mejora de la base de datos. En un proyecto con modelos neuronales siempre es conveniente comprobar cómo se puede mejorar los datos de entrenamiento y validación. Este proceso puede ser muy caro computacionalmente, ya que implicaría volver a entrenar los modelos con más datos, aunque puede mejorar ligeramente el resultado.
- Procesamiento de las salidas. Un aspecto menos estudiado de los modelos neuronales es la realización de un procesado de la salida. Generalmente, dividir el problema en partes más asumibles puede aportar mejores resultados. Un ejemplo claro es este proyecto: gracias a la división de la clasificación en dos secciones, YOLOv8 y ResNet-50, conseguimos mejores resultados. Asimismo, realizando un mejor procesado de la salida del modelo YOLO podríamos mejorar los resultados del modelo ResNet-50.
- Mejora del segundo modelo de clasificación. Durante el apartado anterior se ha comprobado como la mayoría de los errores estaban acumulados en la clasificación secundaria con el modelo ResNet-50. Para corregir estos fallos podríamos utilizar otro modelo más avanzado o incluso usar de nuevo un modelo YOLOv8 entrenado con una base de datos específica de señales con vista cercana. Al igual que con la primera posible mejora esto supone un coste computacional y de tiempo elevados.

- Por último, hay que mencionar que para asegurar que un sistema completo es implementable en un caso real es necesario probarlo frente a una mayor variedad de condiciones climatológicas. Es conveniente probarlo en condiciones de nieve (no se ha podido realizar en este proyecto por temas geográficos) o lluvia intensa entre otros.

REFERENCIAS

[1] R. Bishop, «Intelligent vehicle applications worldwide,» *IEEE Intelligent Systems and their Applications,* vol. 15, nº 1, pp. 78-81, 2000.

[2] D. Tabernik y D. Skočaj, «Deep Learning for Large-Scale Traffic-Sign Detection and Recognition,» *IEEE Transactions on Intelligent Transportation Systems,* vol. 21, nº 4, pp. 1427 - 1440, 2019.

[3] J. R. Farhadi, S. Divvala, R. Girshick y Ali, «You Only Look Once: Unified, Real-Time Object Detection,» arXiv:1506.02640v, 2016.

[4] X. Tang, Y. Li y X. Wei, «Environmental Perception for Intelligent Vehicles,» de *Lecture Notes in Intelligent Transportation and Infrastructure*, 2022, p. 61–106.

[5] D. Tokody, I. Mezei y G. Schuster, «An Overview of Autonomous Intelligent Vehicle Systems,» de *Vehicle and Automotive Engineering. Lecture Notes in Mechanical Engineering*, Springer, Cham, 2017.

[6] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» arXiv:1512.03385v1, 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, «Going Deeper with Convolutions,» arXiv:1409.4842v1, 2014.

[8] J. Redmon y A. Farhadi, «YOLOv3: An Incremental Improvement,» arXiv:1804.02767v1 , 2018.

[9] G. Jocher, M. Rizwan Munawar y A. Chaurasia, «Ultralytics,» 12 11 2023. [En línea]. Available: https://docs.ultralytics.com/#yolo-a-brief-history.

[10] J. Terven, E. Córdova, M. Diana, G. Romero y A. Julio, «A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS,» *Machine Learning and Knowledge Extraction,* vol. 5, p. 1680–1716, 2023.

[11] «Github/RangeKing,» 10 1 2023. [En línea]. Available: https://github.com/RangeKing.

[12] G. Jocher, Burhan, Laughing, A. Chaurasia y f. c. akyon, «Ultralytics,» 12 11 2023. [En línea]. Available: https://docs.ultralytics.com/es/models/yolov8/.

[13] T.-Y. L. Dollár, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick y Piotr, «Microsoft COCO: Common Objects in Context,» arXiv:1405.0312v3, 2015.

[14] T. A. G. Diwan y J. Tembhurne, «Object detection using YOLO: challenges, architectural successors, datasets and applications,» *Multimedia Tools and Applications,* vol. 82, p. 9243–9275, 2022.

[15] J. Choquette, W. Gandhi, O. Giroux, N. Stam y R. Krashinsky, «NVIDIA A100 Tensor Core GPU: Performance and Innovation,» *IEEE Micro,* vol. 41, nº 2, pp. 29 - 35, 2021.

[16] J. Dong, W. Dong, R. Socher, L.-J. Li, K. li y L. Fei-Fei, «Imagenet: A large-scale hierarchical image

database,» *IEEE conference on computer vision and patternrecognition,* pp. 248-255, 2009.

[17] J. Stallkamp, M. Schlipsing, J. Salmen y C. Igel, «The German Traffic Sign Recognition Benchmark: A multi-class classification competition,» de *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, USA, 2011.

[18] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing y R.-r. Feris, «Spottune: transfer learning through adaptive fine-tuning,» *Proceedings of theIEEE/CVF conference on computer vision and pattern recognition,* p. 4805–4814, 2019.

[19] S. S. Liu y Xiuwen, «Overfitting Mechanism and Avoidance in Deep Neural Networks,» arXiv:1901.06566v1, 2019.

[20] G. Jocher, I. Zhu y Burhan, «Ultralytics,» 12 11 2023. [En línea]. Available: https://docs.ultralytics.com/es/modes/track/.

[21] Q. Wang, F. Chen, W. Xu y M.-H. Yang, «An experimental comparison of online object-tracking algorithms,» *Wavelets and Sparsity XIV,* vol. 8138, p. 81381A, 2011.

[22] A. Bewley, Z. Ge, L. Ott, F. Ramos y B. Upcroft, «Simple online and realtime tracking,» *2016 IEEE International Conference on Image Processing (ICIP),* 2016.

[23] N. Wojke, A. Bewley y D. Paulus, «Simple Online and Realtime Tracking with a Deep Association Metric,» arXiv:1703.07402v1, 2017.

[24] Y. Z. Wang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu y Xinggang, «ByteTrack: Multi-Object Tracking by Associating Every Detection Box,» *arXiv:2110.06864v3,* 2022.

[25] Y. D. Meng, Z. Zhao, Y. Song, Y. Zhao, F. Su, T. Gong y Hongying, «StrongSORT: Make DeepSORT Great Again,» *https://arxiv.org/abs/2202.13514,* 2023.

[26] P. Dendorfer, A. Ošep, A. Milan, K. Schindler, D. Cremers, I. Reid, S. Roth y L. Leal-Taixé, «MOTChallenge: A Benchmark for Single-Camera Multiple Target Tracking,» *International Journal of Computer Vision,* vol. 129, p. 845–881, 2021.

[27] J. E. Maese, «Github/JoseMaese,» Github, 30 6 2024. [En línea]. Available: https://github.com/JoseMaese/Traffic-Sign-Recognition-and-Classification-Using-Neural-Networks.

GLOSARIO

YOLO: You Only Look Once

IA/AI: Inteligencia Artificial

DL: Deep Learning

ML: Machine Learning

CPU: Central Processing Unit

GPU: Graphics Processing Unit

RAM: Random Access Memory

ReLU: Rectified Linear Unit

API: Application Programming Interfaces

CNN: Convolutional Neural Network

BS: Batch Size

ResNet: Residual Network

FCNN: Full Convolutional Neural Network

CRNN: Convolutional Recurrent Neural Networks

ABS: Anti-lock Braking System

SAE: Society of Automotive Engineers

COCO: Microsoft Common Object in Context

IoU: Intersection over Union

AP: Average Precision

mAP: mean Average Precision

GTSBD: German Traffic Sign Benchmarks dataset

ROI: Region Of Interest:

SORT: Simple Online Realtime Tracking

ANEXO A

Código correspondiente a la función de la gestión de la base de datos.

```
'''
José E. Maese Álvarez.
TFM: Uso de redes neuronales para identificación de señales de tráfico.
Funciones de creacion de base de datos
'''

import os
import random
from shutil import copyfile

def split_data(input_folder, output_train, output_valid, output_test,
train_ratio=0.7, valid_ratio=0.1, test_ratio=0.2):
    # Verifica que las proporciones sumen 1.0
    assert train_ratio + valid_ratio + test_ratio == 1.0, "Las proporciones
deben sumar 1.0"

    # Obtén la lista de archivos en la carpeta de imágenes
    image_files = [f for f in os.listdir(os.path.join(input_folder, 'images'))
if f.endswith('.jpg')]

    # Baraja aleatoriamente la lista de archivos
    random.shuffle(image_files)

    # Calcula las divisiones de los conjuntos de datos
    total_files = len(image_files)
    train_split = int(train_ratio * total_files)
    valid_split = int(valid_ratio * total_files)

    # Divide los archivos en conjuntos de entrenamiento, validación y prueba
    train_set = image_files[:train_split]
    valid_set = image_files[train_split:train_split + valid_split]
    test_set = image_files[train_split + valid_split:]

    # Copia los archivos a las carpetas correspondientes
    copy_files(train_set, os.path.join(input_folder, 'images'), output_train)
    copy_files(valid_set, os.path.join(input_folder, 'images'), output_valid)
    copy_files(test_set, os.path.join(input_folder, 'images'), output_test)

def copy_files(file_list, source_folder, destination_folder):
    for file in file_list:
        source_path = os.path.join(source_folder, file)
        dest_path = os.path.join(destination_folder, 'images', file)
        copyfile(source_path, dest_path)

        # También copia el archivo de etiquetas correspondiente si existe
        label_file = os.path.splitext(file)[0] + '.txt'
        source_label_path = os.path.join(input_folder, 'labels', label_file)
        dest_label_path = os.path.join(destination_folder, 'labels', label_file)

        if os.path.exists(source_label_path):
            copyfile(source_label_path, dest_label_path)
```

```
# Rutas de entrada y salida
input_folder = r'C:\Users\josen\Archivos
TFM\crear_dataset\Datasets\Mapillary_YOLOv8_version\test'
output_train = r'C:\Users\josen\Archivos
TFM\crear_dataset\Datasets\Mapillary_DS\train'
output_valid = r'C:\Users\josen\Archivos
TFM\crear_dataset\Datasets\Mapillary_DS\valid'
output_test = r'C:\Users\josen\Archivos
TFM\crear_dataset\Datasets\Mapillary_DS\test'

# Crea las carpetas de salida si no existen
os.makedirs(output_train, exist_ok=True)
os.makedirs(output_valid, exist_ok=True)
os.makedirs(output_test, exist_ok=True)

# Llama a la función para dividir los datos
split_data(input_folder, output_train, output_valid, output_test)
```

ANEXO B

Código correspondiente a la función de entrenamiento de la red ResNet-50.

```
'''
José E. Maese Álvarez.
TFG: Uso de redes neuronales para identificación de matrículas.
Función para el entrenamiento de la red ResNet50
'''
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import funciones as fun
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.models import load_model
import tensorflow as tf
from tensorflow.keras.preprocessing import image_dataset_from_directory
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'


learn = 0.00001
epocas = 50


BATCH_SIZE = 64
IMG_SIZE = (180, 180)

directory = "Datasets/ClassificationTrafficSigns/"
train_dataset = image_dataset_from_directory(directory =
'Datasets/ClassificationTrafficSigns',
                                             shuffle=True,
                                             color_mode = 'rgb',
                                             batch_size=BATCH_SIZE,
                                             image_size=IMG_SIZE,
                                             validation_split=0.2,
                                             subset='training',
                                             seed = 42)

test_dataset = image_dataset_from_directory(directory =
'Datasets/ClassificationTrafficSigns',
                                            shuffle=True,
                                            batch_size=BATCH_SIZE,
                                            color_mode = 'rgb',
                                            image_size=IMG_SIZE,
                                            validation_split=0.2,
                                            subset='validation',
                                            seed = 42)
```

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
data_augmentation = fun.data_augmenter()


model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                                                  input_shape=(180,180,3),
                                                  pooling='avg',classes=13,
                                                  weights='imagenet')

pretrained_model.trainable = False
pretrained_model.summary()

model.add(pretrained_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(13, activation='softmax'))

opt = keras.optimizers.Adam(learning_rate=learn)
model.compile(optimizer=opt,
loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history = model.fit(train_dataset, validation_data=test_dataset, epochs=epocas)

model.save("ResNet_Model_local_v2.h5")

plt.figure(1)
df_loss_acc = pd.DataFrame(history.history)
df_loss= df_loss_acc[['loss','val_loss']]
df_loss.rename(columns={'loss':'train','val_loss':'validation'},inplace=True)
df_loss.plot(title='Model
loss',figsize=(12,8)).set(xlabel='Epoch',ylabel='Loss')
plt.close(1)

plt.figure(2)
df_acc= df_loss_acc[['accuracy','val_accuracy']]
df_acc.rename(columns={'accuracy':'train','val_accuracy':'validation'},inplace=T
rue)
df_acc.plot(title='Model
Accuracy',figsize=(12,8)).set(xlabel='Epoch',ylabel='Accuracy')
plt.close(2)

ResNet_pred = model.predict(test_dataset)

resultados_ResNet = df_loss_acc.to_numpy()
```

ANEXO C

Código correspondiente al sistema completo de detección de señales de tráfico a partir de video.

```
'''
José E. Maese Álvarez.
TFM: Uso de redes neuronales para identificación de señales de tráfico.
Código completo de detección e identificación de señales.
'''


import cv2
from ultralytics import YOLO
import os
import numpy as np
from tensorflow.keras.models import load_model

os.environ['KMP_DUPLICATE_LIB_OK'] = 'TRUE'

# Load the YOLOv8 model
model = YOLO("Modelos/modelo004.pt")
class_names = open('signal.names').read().strip().split('\n')

# Load ResNet50 model
modelo_resnet = load_model("Modelos/Clasificador/ResNet_Model_local.h5")


# Open the video file
cap = cv2.VideoCapture("Imagenes_prueba/video_prueba_2.mp4")

# Get video frame width and height
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Obtener la resolución original
original_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
original_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Definir la nueva resolución deseada (por ejemplo, la mitad de la resolución
original)
new_width = original_width // 1
new_height = original_height // 1

# Get video frame rate
fps = cap.get(cv2.CAP_PROP_FPS)

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('videos/output_video_2_con_clasificador_v1.mp4', fourcc,
fps, (new_width, new_height))

# Dibujamos los bounding boxes correspondientes
def dibuja_frame(frame, box, class_name, conf):
    color = (96, 174, 39)   # BGR
    conf = "{:.3f}".format(conf)
```

```
    if id is not None:
        if class_name == 0:
            clase = 'Danger'
            color = (18, 156, 243)

        if class_name == 1:
            clase = 'Mandatory'
            color = (219, 152, 52)

        if class_name == 2:
            # clase = 'Other'
            clase = analisis_velocidad(frame, box)
            color = (60, 76, 231)

        if class_name == 3:
            clase = 'Prohibitory'
            color = (42, 32, 23)

        text = f"ID {id}: {clase} {conf}"

        # Obtener dimensiones del texto
        (text_width, text_height), _ = cv2.getTextSize(text,
cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)
        # Dibujar fondo
        cv2.rectangle(frame, (box[0], box[1] - 4 - text_height), (box[0] +
text_width, box[1]), color, cv2.FILLED)
        # Escribir texto en blanco
        cv2.putText(frame, text, (box[0], box[1] - 4), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (255, 255, 255), 2)
    # Dibujar recuadro
    cv2.rectangle(frame, (box[0], box[1]), (box[2], box[3]), color, 2)




def analisis_velocidad(signal, box):
    tam = 180
    # Aumentar el tamaño de la región de interés (ROI) en un 10% en cada
dirección
    width_increase = int((box[2] - box[0]) * 0.1)
    height_increase = int((box[3] - box[1]) * 0.1)

    # Definir los nuevos límites del cuadro delimitador
    new_box = [max(0, box[0] - width_increase),
           max(0, box[1] - height_increase),
           min(signal.shape[1], box[2] + width_increase),
           min(signal.shape[0], box[3] + height_increase)]

    # Región de interés con el nuevo tamaño
    roi = signal[new_box[1]:new_box[3], new_box[0]:new_box[2]]

    # Region of interest
    # roi = signal[box[1]:box[3], box[0]:box[2]]

    # Preprocesa la imagen recortada para que sea compatible con la entrada de
la red ResNet
    roi_resized = cv2.resize(roi, (tam, tam))
```

```
    np_img = np.array(roi_resized)
    signal_preprocessed = np.reshape(np_img, [-1, tam, tam, 3])

    # Utiliza el modelo ResNet para predecir la clase de la señal de tráfico
    resultado = modelo_resnet.predict(signal_preprocessed)

    # Reducir la importancia de las últimas 4 clases
    resultado[0][-4:] *= 0.3

    class_index = resultado.argmax(axis=1)

    clases = {
       0: '100', 1: '120', 2: '20', 3: '30', 4: '40', 5: '50', 6: '60', 7: '70',
       8: '80', 9: 'no_overtaking', 10: 'no_truck_overtaking', 11:
'prohibited_driving',
       12: 'truck_access_denied'
   }

    # Devuelve la clase predicha
    return clases[class_index[0]]


# Loop through the video frames
while True:
    # Read a frame from the video
    ret, frame = cap.read()

    # If frame is read correctly, proceed
    if ret:

        frame = cv2.resize(frame, (new_width, new_height))
        results = model.track(frame, persist=True)

        # Check if there are any detections
        if results[0].boxes is not None:
            # Extract IDs if they exist
            ids = results[0].boxes.id.cpu().numpy().astype(int) if
results[0].boxes.id is not None else []
            class_index = results[0].boxes.cls.cpu().numpy().astype(int) if
results[0].boxes.cls is not None else []
            confianzas = results[0].boxes.conf.cpu().numpy().astype(float) if
results[0].boxes.conf is not None else []

            # Annotate frame with boxes and IDs
            for i, box in
enumerate(results[0].boxes.xyxy.cpu().numpy().astype(int)):
                # Asigna el ID correspondiente a la detección actual, si existen
IDs disponibles.
                # Si no hay IDs disponibles, id se establece en None.
                id = ids[i] if len(ids) > 0 else None
                # Obtenemos la clase predicha para esta detección y su confianza
                class_name = class_index[i] if len(class_index) > 0 else None
                conf = confianzas[i] if len(confianzas) > 0 else None

                dibuja_frame(frame, box, class_name, conf)
```

```
        out.write(frame)

        # Display the annotated frame
        cv2.imshow("YOLOv8 Tracking", frame)

        # Break the loop if 'q' is pressed
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break
    else:
        # Break the loop if the end of the video is reached or frame is not read
correctly
        break

# Release the video capture object and close the display window
cap.release()
out.release()
cv2.destroyAllWindows()
```