Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de las Telecomunicaciones

Detección y Clasificación Eficiente de Señales de Tráfico mediante Redes Convolucionales

Autor: Álvaro Muñoz Espinosa

Tutores: José Ramiro Martínez de Dios, Raúl Tapia López

Dpto. Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de las Telecomunicaciones

Detección y Clasificación Eficiente de Señales de Tráfico mediante Redes Convolucionales

Autor:
Álvaro Muñoz Espinosa

Tutor:
José Ramiro Martínez de Dios
Catedrático
Raúl Tapia López
Ingenierio Técnico Industrial

Dpto. Ingeniería Eléctrica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado: Detección y Clasificación Eficiente de Señales de Tráfico mediante Redes Convolucionales

Autor: Álvaro Muñoz Espinosa
Tutores: José Ramiro Martínez de Dios, Raúl Tapia López

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Me gustaría, en primer lugar, agradecer a mis padres por la gran educación que me han dado desde pequeño y por todo el apoyo incondicional que siempre me han brindado. Sin su guía y su ejemplo, nada de esto habría sido posible. A mi hermano Millán, por ser siempre esa referencia a la que mirar y seguir, no solo por ser el mayor, sino por la gran persona que es.

Quiero también expresar mi gratitud a todos mis amigos, quienes han creído en mí y me han ayudado a mantenerme enfocado en este camino, especialmente a mi mejor amiga y compañera de vida, Andrea, quien siempre ha confiado en mis capacidades y se ha preocupado constantemente por que no perdiera el rumbo y continuara estudiando.

Asimismo, me gustaría reconocerme a mí mismo, por la madurez que he adquirido a lo largo de estos años y por el esfuerzo que he puesto en corregir mis errores. Por todas esas horas de trabajo tras una barra, que me permitieron costearme la carrera, y por el profesional en el que me estoy convirtiendo. Hoy puedo decir con orgullo que todo ese sacrificio ha valido la pena.

No puedo dejar de agradecer a mis compañeros de trabajo Rubén y Rafa, por su ayuda y orientación durante la realización de este TFG. También a mi compañero de carrera Enrique, de quien he aprendido muchísimo y quien me ha contagiado esa motivación constante por mejorar.

Finalmente, quiero dar las gracias a mi tutor Ramiro, por su valiosa tutoría durante este proyecto, y a Raúl Tapia, por su generosa ayuda y el apoyo desinteresado que me ha ofrecido.

Resumen

En el ámbito de la seguridad vial y la asistencia a la conducción, el reconocimiento de señales de tráfico mediante técnicas de visión artificial se ha convertido en un área de gran interés. Este Trabajo Fin de Grado tiene como objetivo desarrollar un sistema eficiente para la detección y clasificación rápida de señales de tráfico, utilizando redes neuronales convolucionales profundas.

El proyecto se compone de dos etapas principales: la detección de señales, realizada mediante la arquitectura YOLOv8, y la clasificación de las mismas, para lo cual se han diseñado y entrenado redes personalizadas. A lo largo del trabajo se ha llevado a cabo un exhaustivo estudio del estado del arte en redes neuronales y arquitecturas profundas, así como un análisis detallado de los tipos de capas más relevantes y las métricas que se usan para evaluar el rendimiento de los modelos.

El sistema se ha entrenado y validado utilizando conjuntos de datos reconocidos, como BDD100K para la detección y GTSDB para la clasificación. Adicionalmente, se ha desarrollado un pipeline automatizado que permite integrar ambas etapas y procesar secuencias de video en tiempo real, mostrando la viabilidad del sistema para su aplicación en entornos reales.

Por último, se han comparado los resultados obtenidos por las redes personalizadas con los modelos de clasificación de YOLOv8 demostrando que las redes desarrolladas alcanzan un rendimiento competitivo en términos de precisión y una mejora significativa en la velocidad de inferencia, siendo hasta diez veces más rápidas.

Abstract

In the field of road safety and driver assistance, traffic sign recognition using computer vision techniques has become an area of great interest. This Final Degree Project aims to develop an efficient system for the fast detection and classification of traffic signs using deep convolutional neural networks.

The project consists of two main stages: traffic sign detection, performed using the YOLOv8 architecture, and classification, for which custom-designed and trained networks have been implemented. Throughout the project, an exhaustive study of the state of the art in neural networks and deep architectures was conducted, along with a detailed analysis of the most relevant types of layers and the metrics used to evaluate model performance.

The system was trained and validated using well-known datasets, such as BDD100K for detection and GTSDB for classification. Additionally, an automated pipeline was developed to integrate both stages and process video sequences in real time, demonstrating the feasibility of the system for real-world applications.

Finally, the results obtained by the custom networks were compared with the YOLOv8 classification models, showing that the developed networks achieve competitive performance in terms of accuracy and a significant improvement in inference speed, being up to ten times faster.

Índice Abreviado

Índice

1 Introducción

En un contexto donde la inteligencia artificial (IA) se ha convertido en un eje central del desarrollo tecnológico, sus aplicaciones en la industria automovilística están marcando un antes y un después en términos de seguridad vial y asistencia al conductor. Sistemas como el frenado automático de emergencia, el mantenimiento de carril y el reconocimiento de voz son solo algunas de las implementaciones que, gracias al aprendizaje profundo y las redes neuronales, están acercando a los vehículos hacia la conducción autónoma.

El reconocimiento de señales de tráfico constituye un pilar fundamental en este avance tecnológico. Permite incrementar la seguridad al alertar al conductor sobre la presencia de señales que, por diversas circunstancias, podría no percibir. Sin embargo, este problema presenta retos significativos debido a factores como variaciones en la iluminación, condiciones climáticas adversas y la gran diversidad de señales existentes.

Este Trabajo de Fin de Grado (TFG) aborda la tarea de reconocimiento de señales de tráfico mediante el uso de modelos avanzados de redes neuronales convolucionales, con un enfoque específico en la detección y clasificación rápida y eficiente de señales. El proyecto se centra en el desarrollo de un sistema modular compuesto por dos etapas principales: la detección y la clasificación. La detección se encarga de localizar las señales de tráfico presentes en una imagen, mientras que la clasificación tiene como objetivo determinar el tipo exacto de señal detectada. Este enfoque modular permite una mayor flexibilidad y escalabilidad del sistema, así como la posibilidad de optimizar y evaluar de manera independiente cada una de las etapas.

Para la detección se emplean modelos de la arquitectura YOLOv8. Por otro lado, se desarrollan redes de clasificación personalizadas, diseñadas y entrenadas específicamente para esta tarea. Además, se implementa un pipeline automatizado que enlaza ambas etapas, permitiendo el procesamiento continuo de secuencias de video y facilitando su aplicación en sistemas avanzados de asistencia al conductor *(ADAS)*.

Finalmente, el trabajo incluye un análisis exhaustivo de los errores de las redes, con el objetivo de identificar los principales desafíos del sistema y proponer posibles líneas de mejora. Este TFG combina el uso de arquitecturas de vanguardia en visión artificial con el desarrollo de soluciones personalizadas, planteando un sistema modular y escalable para el reconocimiento de señales de tráfico que podría servir como base para futuras investigaciones en el área de la seguridad vial y la conducción autónoma.

1.1 Objetivos

El principal objetivo de este Trabajo de Fin de Grado es desarrollar un sistema de reconocimiento de señales de tráfico utilizando modelos avanzados de redes neuronales convolucionales, con un

enfoque específico en la arquitectura YOLO (You Only Look Once). Este objetivo general se desglosa en los siguientes puntos específicos:

1. **Implementación de un sistema de detección basado en YOLOv8**:
   - Entrenar y ajustar modelos YOLOv8 para la tarea de detección de señales de tráfico utilizando el dataset BDD100K.
   - Evaluar el rendimiento de los modelos en términos de métricas como la precisión promedio (*AP*) y la precisión media promedio (*mAP*).
2. **Desarrollo de redes neuronales para la clasificación de señales de tráfico**:
   - Diseñar y entrenar redes neuronales convolucionales personalizadas utilizando el dataset GTSDB.
   - Realizar un *benchmarking* detallado para identificar las mejores arquitecturas diseñadas, evaluándolas en métricas como la precisión, *recall* y *F1-Score*.
   - Comparar el desempeño de las redes desarrolladas con los modelos de clasificación de YOLOv8, considerando tanto la precisión como la velocidad de inferencia.
3. **Automatización del pipeline de detección y clasificación**:
   - Desarrollar un pipeline automatizado que enlace las redes de detección y clasificación, permitiendo el procesamiento continuo de secuencias de video.
4. **Contribución a la investigación y aplicaciones prácticas**:
   - Proponer mejoras y ajustes en los modelos que puedan ser útiles para futuras investigaciones.

1.2 Estructura del documento

Este documento está estructurado en cinco capítulos principales, además de los apéndices, bibliografía e índices. A continuación, se presenta un resumen de su contenido:

- **Capítulo 2: Estado del arte.** Se presenta un análisis detallado de las arquitecturas de redes neuronales convolucionales, con especial énfasis en la evolución de la familia YOLO. También se examinan los principales datasets utilizados en el reconocimiento de señales de tráfico, destacando sus características y aplicaciones.
- **Capítulo 3: Descripción del problema**: En este capítulo se aborda el problema del reconocimiento de señales de tráfico, diferenciando las tareas de detección y clasificación. Además, se introducen las principales métricas empleadas para evaluar los modelos desarrollados.
- **Capítulo 4: Experimentos**: Este capítulo detalla el proceso experimental llevado a cabo, desde la preparación de los datos hasta el entrenamiento y evaluación de los modelos de detección y clasificación. También se discuten los resultados obtenidos y su impacto en el rendimiento de los modelos.
- **Capítulo 5: Conclusiones y desarrollos futuros**: Se exponen las conclusiones derivadas del trabajo realizado, junto con propuestas de mejora y líneas de investigación futuras.

2 Estado del arte

En un mundo cada vez más orientado hacía la movilidad y la seguridad vial, las aplicaciones de la inteligencia artificial en el ámbito automovilístico están a la orden del día. De hecho, algunas de ellas se plantean obligatorias por la Unión Europea para los coches de nueva matriculación, como es el sistema de mantenimiento de carril [6] . La ayuda de los múltiples sistemas de reconocimiento de objetos mediante inteligencia artificial y la actuación pasiva o activa sobre la conducción prevén una mejora crucial en la seguridad de los pasajeros. El objeto de este trabajo de fin de grado es conseguir detectar con la mayor exactitud posible diferentes elementos en carretera, llegando a clasificar múltiples señales de tráfico. Para ello, ha sido preciso la inmersión en el mundo de la inteligencia artificial.

Este capítulo se organiza en cinco secciones. En primer lugar, se introduce el concepto de redes neuronales y el aprendizaje profundo, repasando sus fundamentos y primeros avances. Posteriormente, se analiza en detalle la arquitectura de las Redes Neuronales Convolucionales, describiendo modelos clave como el Neocognitron, LeNet-5, AlexNet y ResNet, fundamentales para la evolución del campo. A continuación, se aborda el reconocimiento de señales de tráfico, presentando su progreso y aplicación en entornos reales.

El núcleo del capítulo se centra en el estudio de la arquitectura YOLO, explicando su origen y evolución a través de sus distintas versiones, desde YOLOv1 hasta YOLOv8, la cual se emplea en este trabajo. Se detallan las mejoras introducidas en cada iteración y su impacto en la detección en tiempo real. Finalmente, se revisan los principales datasets utilizados en el reconocimiento de señales de tráfico, esenciales para el entrenamiento y evaluación de estos modelos.

2.1 Introducción a redes neuronales y arquitecturas *deep learning*

En 1957, Frank Rossenblatt describió por primera vez el Perceptrón [20], un clasificador binario el cual genera una predicción basándose en un algoritmo combinado con el peso de las entradas. Esta es considerada como la primera red neuronal. En la Figura 2.1 podemos ver el modelo del Perceptrón.

Dos años después, Arthul Samuel popularizó el termino *Machine Learning*. El *Machine Learning* se centra en el uso de datos y algoritmos para imitar como las personas aprenden, mejorando gradualmente su acierto[10].

En la década de 1960, el descubrimiento y uso de las multicapas abrieron un nuevo camino en la investigación de las redes neuronales. Se descubrió que proporcionar y utilizar dos o más capas en el perceptrón ofrecía significativamente más poder de procesamiento que un perceptrón con una sola capa. Después de que el perceptrón abriera la puerta a las capas en las redes, se crearon otras versiones de redes neuronales y la variedad de redes neuronales continúa expandiéndose. El uso de múltiples capas llevó a las redes neuronales *feedforward* y a la retropropagación.

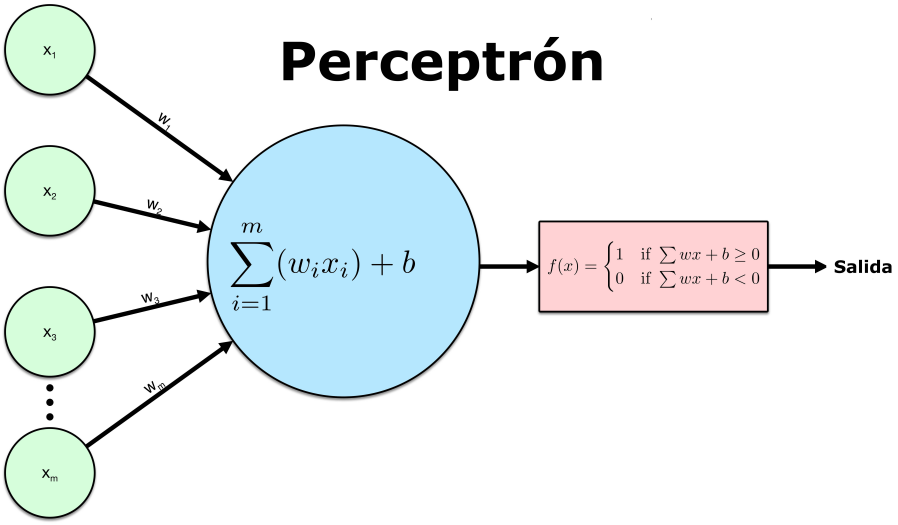

Figura 2.1 Modelo Perceptrón. Figura tomada del blog [1].

La retropropagación, desarrollada en la década de 1970, permite que una red ajuste sus capas ocultas de neuronas/nodos para adaptarse a nuevas situaciones. Describe la propagación hacia atrás de errores, como un error que se procesa en la salida y luego se distribuye hacia atrás a través de las capas de la red con fines de aprendizaje. La retropropagación se utiliza actualmente para entrenar redes neuronales profundas.

Estos avances fueron cruciales en la actualidad de la inteligencia artificial, además nos ayudan a entender mejor de donde partimos y la motivación del estudio de la IA. Tras esta pequeña introducción, nos centraremos en profundizar en el estudio de las redes neuronales convolucionales, ya que son las más utilizadas para el reconocimiento de imágenes que son el objeto de este TFG.

2.2 Arquitectura de redes neuronales convolucionales

En esta sección estudiaremos las redes convolucionales más relevantes. Ya sea por su gran rendimiento o por los hitos importantes que se consiguieron durante su desarrollo. Terminaremos centrándonos en la arquitectura de redes YOLO.

2.2.1 Neocognitron

Neocognitron fue la primera arquitectura de red convolucional y quizás el precursor más temprano de las CNNs. Los conceptos de extracción de características, capas de agrupación y el uso de convolución en una red neuronal fueron introducidos por ella. La estructura de la red estuvo inspirada en el sistema nervioso visual de los vertebrados. En toda la red, con sus capas alternas de S-cells (células simples o células hipercomplejas de orden inferior) y C-cells (células complejas o células hipercomplejas de orden superior), se repetía el proceso de extracción de características por las S-cells y la tolerancia al desplazamiento posicional por las C-cells.

Durante este proceso, las características locales extraídas en etapas inferiores se integran gradualmente en características más globales. Se utilizó para el reconocimiento de caracteres escritos a mano y otras tareas de reconocimiento de patrones, y allanó el camino para las redes neuronales convolucionales.

2.2.2 LeNet-5(1989-1998)

El nombre de las redes neuronales convolucionales realmente se originó con el diseño de LeNet, creado por Yann LeCun y su equipo. Fue desarrollado en gran parte entre 1989 y 1998 para la tarea

de reconocimiento de dígitos escritos a mano[12].

LeNet-5 se utilizó a gran escala para clasificar automáticamente los dígitos escritos a mano en cheques bancarios en los Estados Unidos, también cabe destacar que el reconocimiento de caracteres se había realizado principalmente mediante la ingeniería de características manual, seguida de un modelo de aprendizaje automático para aprender a clasificar las características diseñadas a mano. LeNet hizo redundante la ingeniería manual de características, porque la red aprende automáticamente la mejor representación interna a partir de imágenes sin procesar.

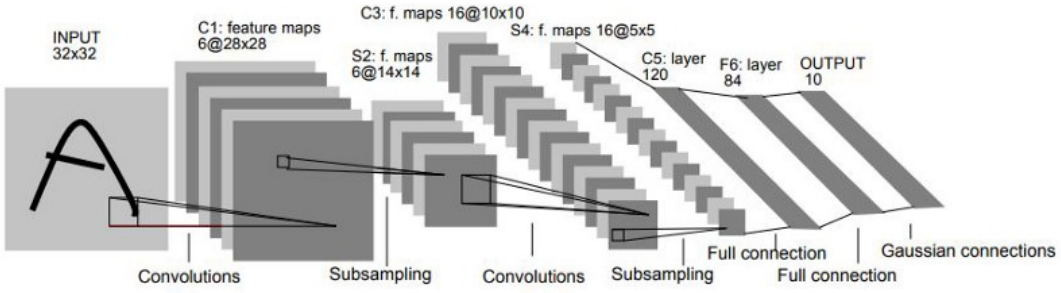

Figura 2.2 Arquitectura LeNet-5 obtenida del artículo [12].

En la Figura 2.2 se observa la arquitectura de la red LeNet-5 obtenida del artículo [12] que consta de 7 capas. La organización de estas comienza con una capa convolucional compuesta de 6 filtros que se crea a partir de una imagen de entrada 32x32 pixeles y genera 6 mapas de características de 28x28. Seguidamente nos encontramos con una capa de sampleo *(pooling)*, que nos permite reducir el tamaño de la red ahorrando coste computacional. Tras estas capas, repetimos patrón con otra capa convolucional y otro *pooling*. Por último, tenemos 3 capas completamente conectadas.

2.2.3 AlexNet(2012)

AlexNet fue la primera red neuronal convolucional que ganó el desafio "ImageNet large scale visual recognition challenge". Este reto comenzó en 2010 y llevo a un esfuerzo significativo de los investigadores a evaluar sus modelos de aprendizaje automático para la clasificación de imágenes.

AlexNet tiene un total de 8 capas (5 capas convolucionales y 3 capas completamente conectadas), entrenadas en el conjunto de datos ImageNet. Se introdujo por primera vez una capa de normalización llamada capa de normalización de respuesta. Normalizaba todos los valores en una ubicación específica en todos los canales de una capa dada. Además, también introdujo la unidad lineal rectificada (ReLU) como una función de activación.

2.2.4 ResNet(2015)

ResNet, término que procede de redes residuales, es una arquitectura de red neuronal profunda que ha sido fundamental en el campo del aprendizaje profundo. Fue introducida por Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun en el artículo "Deep Residual Learning for Image Recognition" en 2015 [9].

La idea principal detrás de ResNet es el concepto de *residual learning*. En lugar de aprender directamente la función deseada, ResNet aprende las diferencias entre la entrada y la salida.

La estructura de ResNet se basa en bloques residuales. Estos bloques tienen dos rutas principales: una ruta directa que pasa la entrada original a través de la red sin cambios (identidad) y una ruta residual que aplica algunas transformaciones a la entrada para hacer la predicción. La salida de estas rutas se suma y luego se pasa a través de una función de activación para obtener la salida final del bloque.

Esta estructura de bloque residual permite que las redes sean mucho más profundas sin sufrir problemas degradantes del rendimiento, como la desaparición o explotación del gradiente, que es un desafío común en el entrenamiento de redes muy profundas.

2.3 Reconocimiento de señales de tráfico

En esta sección vamos a indagar sobre el reconocimiento de señales de tráfico a través de computadoras, los primeros estudios sobre el tema y la evolución del estado del arte.

El primer artículo que encontramos en el IEEE sobre reconocimiento de señales de tráfico data del año 1989 y se titula "Traffic Sign Recognition in Color Image Sequences" [19] este describe el desarrollo de un sistema para el reconocimiento de señales de tráfico en secuencias de imágenes en color. Este sistema fue desarrollado como parte del proyecto europeo Prometheus, que busca mejorar la seguridad y la movilidad vehicular. El sistema utiliza una cámara montada en un vehículo para capturar imágenes, las cuales son procesadas en varias etapas. En la primera etapa, se utiliza una red neuronal de alto orden para la segmentación de color, identificando regiones que pueden contener señales de tráfico. A partir de estas regiones, se generan hipótesis sobre el tipo de señal presente, que luego son verificadas mediante una descripción simbólica y conocimiento a priori de las señales de tráfico y escenas exteriores. El conocimiento se organiza en una red basada en marcos que guía el proceso de análisis. Las conclusiones del estudio resaltan la eficacia del sistema para detectar y generar hipótesis correctas sobre las señales de tráfico en condiciones de prueba, aunque no aportan métricas ni formas evaluables de medir el desempeño del sistema.

En 1997 se público el artículo "Road traffic sign detection and classification"[5], en el los investigadores españoles de la Universidad Carlos III nos presentan un sistema de guía de vehículos basado en visión para la detección y clasificación de señales de tráfico. Este sistema consta de dos partes principales: la detección y la clasificación. La detección utiliza un umbral de color y un análisis de forma para identificar las señales, aprovechando los colores y formas distintivas de las señales en comparación con el entorno natural. La clasificación se realiza mediante una red neuronal, se probaron tres redes neuronales de tipo perceptron multicapa. Los resultados muestran que el sistema es efectivo en tiempo real, el algoritmo de detección opera en 220 milisegundos para imágenes de 256x256 píxeles. La clasificación toma aproximadamente 1.2 segundos. Este artículo supuso un gran avance en el reconocimiento de señales y tuvo gran acogida siendo citado en multitud de ocasiones por otros investigadores.

En el año 2011, se produjo una competencia de clasificación de señales de tráfico recogida en el artículo "The German Traffic Sign Recognition Benchmark: A Multi-Class Classification Competition" [21]. Esta competencia fue crucial para los sistemas avanzados de asistencia al conductor y representó un desafío significativo en el reconocimiento de patrones y visión por computadora. Se uso un conjunto de datos completo y realista que contiene más de 50000 imágenes de señales de tráfico, reflejando variaciones significativas en la apariencia visual. Este conjunto de datos abarca 43 clases con frecuencias de clase no balanceadas. El *dataset* es el usado para la red de clasificación creada en este proyecto, en el siguiente apartado se discutirá profundamente sobre el mismo.

En el artículo se detallan las etapas de la competencia y presenta los resultados obtenidos en el primer conjunto de pruebas. Los participantes emplearon diversos métodos de aprendizaje automático, como redes neuronales convolucionales y máquinas de soporte vectorial. El mejor resultado se obtuvo al usar una red neuronal convolucional y una perceptrón multicapa(MLP), logrando un acierto del 98,98 % [4].

Existen distintos estudios actuales sobre la detección y clasificación de señales de tráfico, tras un análisis de las publicaciones, desde 2011 no encontramos en los estudios un cambio de paradigma muy grande en cuanto a clasificación de señales de tráfico, gran parte de estos, en el problema de clasificación siguen usando el *dataset* GTSRB, además la mayoría se centran en la implementación

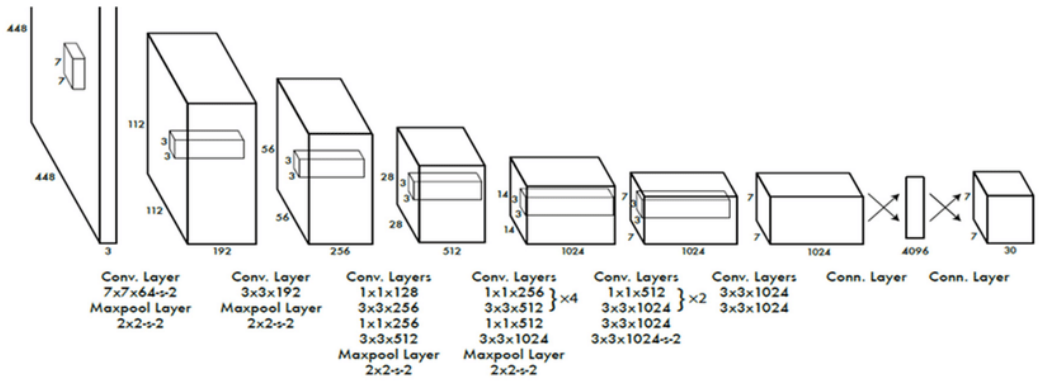

Figura 2.3 Arquitectura YOLOv1 obtenida del artículo [16].

y prueba en dispositivos *edges*. Muchos de los estudios utilizan la redes YOLO para este propósito. En la siguiente sección vamos a introducir que es YOLO y cuales son las distintas versiones que existen, comparando su rendimiento y arquitectura.

2.4 YOLO

Joseph Redmon en 2016 presentó su artículo "You Only Look Once: Unified, Real-Time Object Detection"[16]. En este, introdujo un enfoque revolucionario de detección de objetos de extremo a extremo que permitía un procesamiento en tiempo real. Esto fue un avance significativo para la comunidad.

A diferencia de los métodos anteriores de reconocimiento de imagenes, YOLO es una arquitectura eficiente de detección de objetos que solo requiere un paso por la red. Eliminando la necesidad de múltiples ejecuciones o un proceso de dos pasos.

2.4.1 YOLOv1

El algoritmo YOLO toma una imagen como entrada y luego utiliza una red neuronal convolucional (CNN) profunda y simple para detectar objetos en la imagen. En la Figura 2.3 tenemos un esquema de las capas del modelo anotado en la propia imagen el contenido de dichas capas, obtenida la figura del artículo citado anteriormente [16]

Las primeras 20 capas convolucionales del modelo se entrenan previamente utilizando ImageNet, agregando temporalmente una capa de pooling promedio y una capa completamente conectada. Luego, este modelo preentrenado se convierte para realizar la detección, ya que investigaciones previas han demostrado que agregar capas convolucionales y conectadas a una red preentrenada mejora el rendimiento. La última capa completamente conectada de YOLO predice tanto las probabilidades de clase como las coordenadas de la caja delimitadora.

YOLO divide una imagen de entrada en una cuadrícula de tamaño $S \times S$. Si el centro de un objeto cae en una celda de la cuadrícula, esa celda de la cuadrícula es responsable de detectar ese objeto. Cada celda de la cuadrícula predice B cajas delimitadoras y puntuaciones de confianza para esas cajas. Estas puntuaciones de confianza reflejan qué tan seguro está el modelo de que la caja contiene un objeto y cuán precisa cree que es la caja predicha.

YOLO predice múltiples cajas delimitadoras por celda de la cuadrícula. Durante el entrenamiento, solo queremos que un predictor de caja delimitadora sea responsable de cada objeto. YOLO asigna un predictor como responsable de predecir un objeto en función de cuál de las predicciones tiene el actual Índice de Superposición Unión más alto con la verdad fundamental. Esto lleva a una especialización entre los predictores de las cajas delimitadoras. Cada predictor se vuelve mejor para

prever ciertos tamaños, relaciones de aspecto o clases de objetos, mejorando el puntaje global de recall.

Una técnica clave utilizada en los modelos de YOLO es la supresión de no máximos (NMS). NMS es un paso de post-procesamiento que se utiliza para mejorar la precisión y eficiencia de la detección de objetos. En la detección de objetos, es común que se generen múltiples cajas delimitadoras para un solo objeto en una imagen. Estas cajas delimitadoras pueden superponerse o estar ubicadas en diferentes posiciones, pero todas representan el mismo objeto. NMS se utiliza para identificar y eliminar cajas delimitadoras redundantes o incorrectas y para generar una sola caja delimitadora para cada objeto en la imagen.

A pesar de que YOLOv1 era un detector de objetos rápido, tenía algunas limitaciones. YOLO tenía un error de localización más significativo que los métodos más avanzados como Fast R-CNN. Estas limitaciones se podían atribuir a dos causas principalmente: YOLO solo podía detectar un máximo de dos objetos de la misma clase en una celda de la cuadrícula. YOLO tenía dificultades para predecir objetos con relaciones de aspecto que no estaban presentes en los datos de entrenamiento.

2.4.2 YOLOv2

En Julio de 2017 Redmon presentó un artículo llamado "YOLO9000:Better,Faster,Stronger"[17] que introducía un sistema de detección de objetos capaz de identificar más de 9000 categorías. Este modelo consiguió una sorprendente precisión media en el *dataset* "PASCAL VOC 2007" con una puntuación de 78,6 % mejorando a su antecesor en mas que 15 puntos.

El sistema había mejorado en varios aspectos. El uso de la normalización por lotes en las capas convolucionales para mejorar la convergencia y reducir el sobreajuste. La arquitectura se modificó para incluir capas completamente convolucionales, incluyendo una columna vertebral llamada DarkNet, que contenía 19 capas convolucionales y 5 capas de *max-pooling*. También introdujeron un método para entrenar conjuntamente la clasificación y la detección.

2.4.3 YOLOv3

En abril del 2018, cuando se lanzó YOLOv3 con la publicación de artículo llamado "YOLOv3:An Incremental Improvement"[18], la última versión liderada Joseph Redmon, el estandar para la comparativa de modelos de detección de objetos cambió de "PASCAL VOC" a "Microsoft COCO". A partir de este momento, todos las versiones de YOLO serán evaluadas con el conjuntos de datos "Microsoft COCO". Este modelo incluyó cambios significativos y una arquitectura enorme para estar a la par con el estado del arte mientras se mantenía el rendimiento en tiempo real.

2.4.4 YOLOv4

Alexey Bochkowskiy publicó el artículo "YOLOv4: Optimal Speed and Accuracy of Object Detection"[3]. Yolov4 mantiene la misma filosofía de sus antecesores: tiempo real, código abierto y procesamiento en un solo paso con el framework DarkNet. En esta versión adoptada por la comunidad como la oficial, se intentó encontrar el equilibrio óptimo experimentando con muchos cambios categorizados como *bag-of-freebies* y *bag-of-especials*.

Bag of freebies son métodos que solo cambian la estrategia de entrenamiento y aumenta el costo del entrenamiento, pero no aumentan el tiempo de inferencia. Siendo el más común la ampliación de datos. Por otro lado, los *bag-of-especials* son métodos que si aumentan el tiempo de inferencia pero ayudan a mejorar la precisión significativamente. Ejemplos de estos métodos son aquellos para ampliar el campo receptivo, combinar característicsa y post-procesamientos, entre otros.

2.4.5 YOLOv5

En junio de 2020, la empresa Ultralytics lanzó YOLOv5. Esta versión a diferencia de las anteriores no se lanzó como un artículo científico. YOLOv5 se basa en muchas de las mejoras que introdujo la

versión antecesora, pero la principal diferencia es que se desarrollo utilizando PyTorch en lugar de DarkNet. PyTorch es una librería de aprendizaje automático de código abierto basada en la librería Torch, utilizada para aplicación como la visión artificial y el procesamiento de lenguaje natural. La empresa Ultralytics mantiene activo YOLOv5 como un proyecto de código abierto con más de 250 colaboradores.

2.4.6 YOLOv6

El modelo YOLOv6 fue públicado en un artículo llamado "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications"[13] en septiembre de 2022. Similar a YOLOv5, esta versión proporciona varios modelos de diferente tamaño para aplicaciones industriales. En algunos de ellos mejorando el rendimiento a su versión antecesora. YOLOv6 fue creada por otros autores.

2.4.7 YOLOv7

En julio de 2022, los mismo autores de YOLOv4 publicaron un articulo donde presentaban "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors"[23].

Este detector de objetos superó a todos los demás en velocidad y precisión para objetos que varían desde 5 fotogramas por segundos(FPS) hasta 160 FPS.

La arquitectura de YOLOv7 cuenta con tres características principales: "E-ELAN" para un aprendizaje eficiente, escalado del modelo para diferentes tamaños y el enfoque *bag-of-freebies* para precisión y eficiencia.

Extended Efficient Layer Aggregation Network es el bloque computacional principal de YOLOv7, esta arquitectura permitía mejorar su capacidad de aprendizaje, lo que facilitaba la mejora continua de la capacidad de aprendizaje de la red sin destruir la ruta de gradiente original.

El enfoque *bag-of-freebies* incluye la reparametrización, que mejora el rendimiento del modelo.

El último modelo de YOLOv7 superó a YOLOv4 al lograr una reducción del 75 % en los parámetros y una reducción del 36 % en la computación, al mismo tiempo que mejoró la precisión promedio en un 1,5 %. YOLOv7-tiny, la versión más ligera de YOLOv7, también redujo los parámetros y la computación en un 39 % y un 49 %, respectivamente, sin comprometer demasiado el mAP.

2.4.8 YOLOv8

En enero de 2023 Ultralytics lanzo la versión YOLOv8. Un mes más tarde comencé con la tarea de este TFG y probamos esta arquitectura. Es por ello, que precisa una descripción más detallada y una explicación mayor del funcionamiento ya que es el utilizado en parte del desarrollo de este trabajo. Cabe mencionar que al igual que las versiones anteriores hicieron con sus antecesoras, YOLOv8 mejora el rendimiento de las versiones anteriores.

En la Figura 2.4 obtenida de la fuente [22] se observa un esquema de las distintas capas que conforman la arquitectura YOLOv8 incluso las diferencias entre sus modelos n, s, m, l y x. Dichas capas serán detalladas con mayor precisión en la Sección 3.4.1.

YOLOv8 utiliza un modelo sin anclajes, lo que significa que predice directamente el centro de un objeto en lugar del desplazamiento desde una caja de anclaje conocida.

En versiones anteriores de YOLO, las cajas de anclaje eran una parte complicada, ya que podían representar la distribución de las cajas del conjunto de datos de referencia, pero no necesariamente la distribución del conjunto de datos personalizado. La detección sin anclajes reduce el número de predicciones de cajas, lo que acelera el proceso de supresión no máxima. En lugar de depender de anclajes predefinidos, el modelo sin anclajes permite una mayor flexibilidad en la detección de objetos y puede ser especialmente útil cuando se trabaja con conjuntos de datos personalizados o en escenarios donde las formas y tamaños de los objetos varían considerablemente.

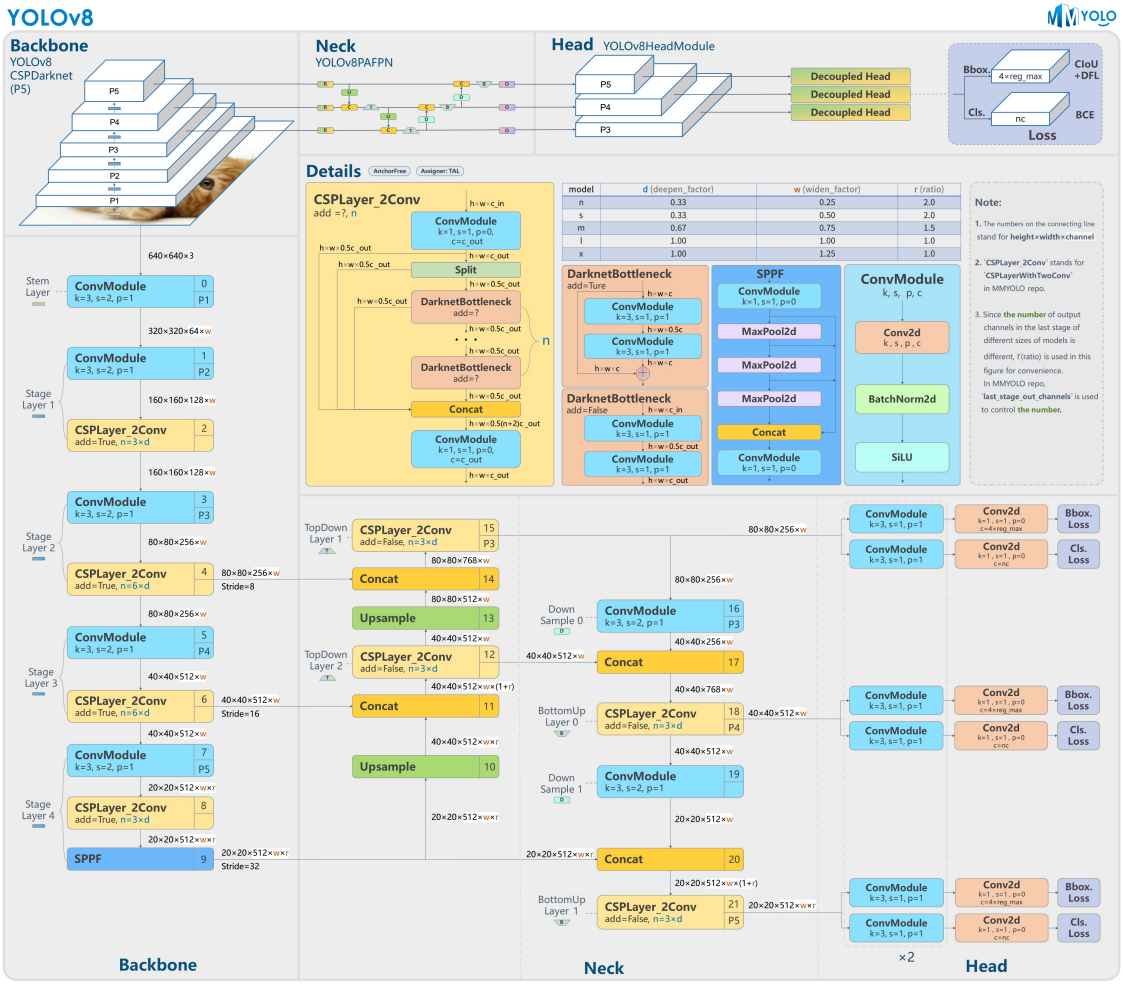

Figura 2.4 Arquitectura de YOLOv8. Figura obtenida de la web [22].

Por otra parte, la investigación en aprendizaje profundo tiende a centrarse en la arquitectura del modelo, pero la rutina de entrenamiento de YOLOv8 es una parte esencial de su éxito. YOLOv8 aumenta las imágenes durante el entrenamiento en línea. En cada época, el modelo ve una variación ligeramente diferente de las imágenes que se le han proporcionado. A uno de estos aumentos se le denomina aumento de mosaico. Este implica unir cuatro imágenes juntas, obligando al modelo a aprender sobre objetos en nuevas ubicaciones, en oclusiones parciales y contra diferentes píxeles circundantes.

Otra funcionalidad importante es la incorporación de forma nativa algoritmos de tracking. Los algoritmos de tracking ayudan a mejorar la precisión del modelo porque no solo tienen en cuenta la imagen instantánea, si no que también consideran las anteriores. Como opciones de implementación de tracking en YOLOv8 nos encontramos con BoT-SORT y ByteTrack.

2.4.9 Futuro de los modelos YOLO y tendencia actual del reconocimiento de imágenes

El 27 de septiembre asistí telemáticamente a un evento que organizó la empresa Ultralytics que introdujo como "la única conferencia en el mundo que se enfoca en el desarrollo y progreso de la inteligencia artificial en visión de código abierto". En ella hubo 18 exponentes y una larga jornada de charlas sobre las aplicaciones actuales y futuras de YOLOv8.

El evento se centro en la implementación de las redes YOLOv8 en dispositivos periféricos o *edge*. Los dispositivos *edge* se refieren a dispositivos informáticos que se encuentran en el borde o cerca del lugar donde se generan los datos. Estos dispositivos pueden incluir sensores, cámaras, drones,

teléfonos inteligentes, *gateways IoT* y otros dispositivos IoT. La computación en el borde implica realizar el procesamiento y análisis de datos directamente en estos dispositivos, en lugar de enviar todos los datos a servidores centralizados en la nube para su procesamiento. Esto es particularmente útil para aplicaciones que requieren respuestas rápidas y análisis en tiempo real.

Diferentes empresas del mundo presentaron sus aportaciones, varias mostraron sus desarrollos en la cuantificación de los modelos. La cuantificación es el proceso mediante el cual los valores numéricos en un modelo de aprendizaje automático se representan con menos bits de precisión. Esto aporta a las redes menor coste computacional facilitando la predicción en dispositivos periféricos, además ayudan a la generalización de la red.

Con este mismo propósito se presentaron los distintos modelos de la empresa Nvidia para la computación periférica, las serie Jetson y sus distintos productos.

Por otra parte, la empresa Ultralytics presentó una herramienta web Ultralytics Hub mediante la que personas sin conocimiento de programación podrán crear sus propios modelos entrandolos y llevandolos a producción con esta herramienta y sin necesidad de programar, acercando la tecnología a muchos más usuarios y empresas.

Como resumen de la jornada, vemos como la tendencia del reconocimiento de imágenes nos lleva a la computación descentralizada ya que mejoraría los tiempos de respuesta, pero todavía nos queda un largo camino porque excluyendo algunas aplicaciones donde los tiempos sean realmente cruciales y el desembolso económico no sea un problema, como pueden ser aplicaciones médicas o militares, no creo que sean soluciones económicas rentables para la mayor parte de productos comerciales en empresas.

2.5 Datasets de tráfico

Como último apartado en esta sección del estado del arte, vamos a dar un repaso a los *datasets* que encontramos de señales de tráfico y *self-driving*.

El conjunto de datos German Traffic Sign Recognition Benchmark (GTSRB) se compila a partir de aproximadamente 10 horas de vídeo grabado en carreteras de Alemania durante 2010, utilizando una cámara "Prosilica GC 1380CH". Las grabaciones, realizadas a 25 fps y con una resolución de 1360×1024 píxeles, se procesaron para extraer imágenes de señales de tráfico. Se definieron instancias de señales de tráfico y se crearon pistas para representar secuencias de imágenes de una sola señal física. De un total de 133.000 imágenes etiquetadas, el conjunto de datos final incluye aproximadamente 50.000 imágenes de 1.700 instancias, distribuidas en 43 clases.

Figura 2.5 Clases del conjunto de datos GTSRB.

Para garantizar la diversidad y evitar el desequilibrio en el conjunto de datos, se establecieron criterios de inclusión que descartan pistas con menos de 30 imágenes y limitan el número de

imágenes por pista a 30. Las imágenes reflejan variaciones en la apariencia de las señales debido a cambios en resolución, desenfoque por movimiento, e iluminación. Este conjunto de datos no presenta homogeneidad en cuanto a número de imágenes por clase y lo podemos observar en la Figura 2.6 como algunas clases tienen mayor presencia respecto a otras. En dicha figura se representa el porcentaje de frecuencia de aparición de las distintas clases respecto del total de imágenes del conjunto de datos.

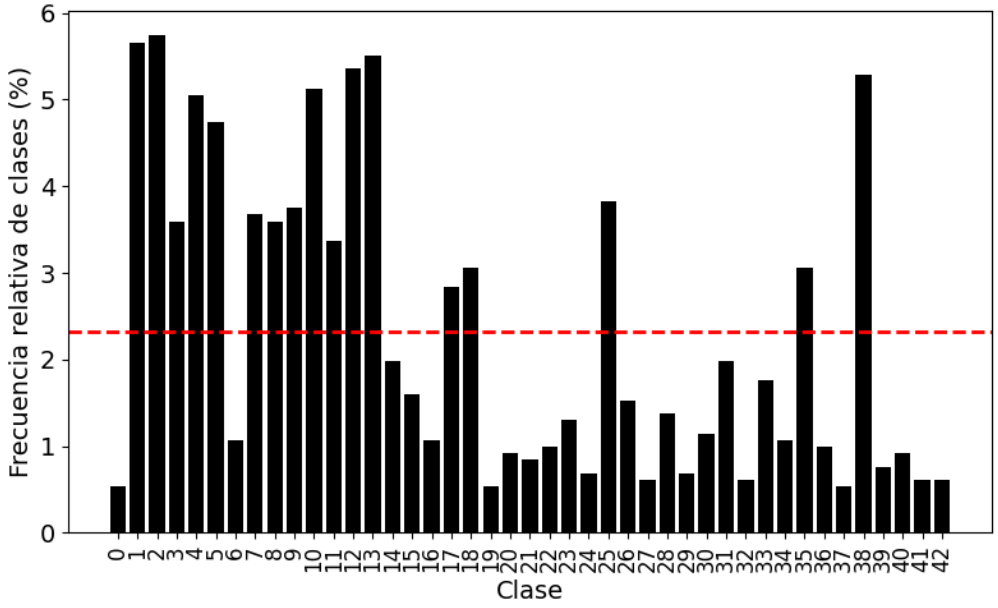

Figura 2.6 Frecuencia relativa de instancias por clase del conjunto de datos GTSRB.

El Mapillary Traffic Sign Dataset es uno de los conjuntos de datos de señales de tráfico más grande y diverso disponible públicamente, diseñado para enseñar a las máquinas a detectar y reconocer señales de tráfico. Este conjunto incluye 100.000 imágenes de diversas partes del mundo, abarcando una amplia gama de variabilidad en términos de clima, hora del día, sensores de cámara y perspectivas. Contiene más de 320.000 señales de tráfico etiquetadas que corresponden a más de 300 clases diferentes. Más de 52.000 imágenes han sido verificadas y anotadas completamente por humanos, mientras que las restantes fueron anotadas parcialmente mediante tecnología de visión por ordenador.

El conjunto de datos de Mapillary se ha utilizado para establecer sólidas referencias en tareas de detección y clasificación, demostrando su eficacia para el aprendizaje por transferencia en otros conjuntos de datos de señales de tráfico a gran escala.

El conjunto de datos BDD100K, lanzado por el Berkeley AI Lab (BAIR) en mayo de 2018 incluye 100.000 videos en alta definición, cada uno con una duración de aproximadamente 40 segundos a una resolución de 720p y 30 fps. Las imágenes clave se muestrean en el décimo segundo de cada vídeo, resultando en 100.000 imágenes de tamaño 1.280x720 que están anotadas con 10 categorías entre ellas encontramos, semáforo, señal de tráfico, coche, persona, autobús, camión, ciclista, motorista, tren y motor.

El BDD100K es notable por su diversidad geográfica, ambiental y climática. Los datos están divididos en conjuntos de 70.000 para entrenamiento, 10.000 para validación y 20.000 para pruebas.

El Swedish Traffic Sign Dataset contiene más de 20.000 imágenes, de las cuales el 20 % están etiquetadas, y abarca información de 3488 señales de tráfico. Las secuencias fueron grabadas en carreteras y ciudades de más de 350 km de vías suecas. Cada etiqueta de señal incluye el tipo de señal (como cruce peatonal, carril designado a la derecha, prohibido parar o estacionar, carretera

prioritaria, ceder el paso, 50 km/h, o 30 km/h), el estado de visibilidad (ocultado, borroso o visible) y el estado de la carretera (si la señal está en la carretera que se recorre o en una vía lateral).

3 Descripción del problema

Como se mencionó en el capítulo anterior, las aplicaciones de la inteligencia artificial en el ámbito automovilístico están experimentando un notable crecimiento. Ejemplos de ello incluyen sistemas como el Frenado Automático de Emergencia, que proporciona frenado urbano e interurbano para prevenir atropellos y colisiones con otros vehículos. Otro ejemplo es el mantenimiento predictivo, que utiliza sensores y otros componentes para anticipar posibles fallos o averías. También, el reconocimiento de voz se ha implementado para mejorar la seguridad del conductor y evitar distracciones al interactuar con pantallas u otros dispositivos. Podemos seguir nombrando infinidad de aplicaciones de la inteligencia artificial en la automoción y como esta acerca también al vehículo a la conducción autónoma.

En este TFG nos vamos a centrar en la aplicación de la IA en el reconocimiento de señales de tráfico, este permite al piloto mejorar la seguridad vial propia y del resto de automóviles al mantenerlo siempre en alerta a cerca de las señalizaciones que hay en la carretera.

Existen diversas circunstancias que pueden llevar a un conductor a no percibir las señales de tráfico a tiempo. Por un lado, factores como la fatiga o la distracción juegan un papel crucial; un conductor cansado o distraído por el uso del teléfono móvil o conversaciones dentro del vehículo puede fácilmente no notar una señal. Además, las condiciones ambientales adversas, como la lluvia intensa, la niebla o la falta de iluminación adecuada, pueden dificultar la visibilidad de las señales, haciendo que pasen desapercibidas.

Las advertencias reconocidas por el sistema pueden presentarse de manera visual, utilizando la pantalla del automóvil o tecnologías como el *Head-up Display*, que proyecta en la luna del coche diversas informaciones, como las indicaciones de tráfico más recientes. También es posible que estas alertas sean transmitidas de forma sonora, asegurando que el conductor esté siempre consciente de las condiciones de la vía.

El capítulo se organiza en seis secciones, donde se analiza el problema del reconocimiento de señales de tráfico desde una perspectiva teórica y técnica, abordando los elementos fundamentales que intervienen en este proceso.

En primer lugar, se introduce la problemática del reconocimiento de señales de tráfico y su importancia en el ámbito de la seguridad vial y la asistencia al conductor. A continuación, se describen en detalle los principales tipos de capas utilizadas en las redes neuronales convolucionales, como las capas de convolución, agrupación, normalización y activación, destacando su papel en la extracción y procesamiento de características.

Posteriormente, se analizan las métricas empleadas para evaluar el rendimiento de los modelos de clasificación y detección, proporcionando un marco cuantitativo para comparar su eficacia. En la cuarta sección, se presentan las redes de clasificación utilizadas en este trabajo, incluyendo tanto las redes basadas en YOLOv8 como las redes desarrolladas específicamente para esta tarea. Del mismo modo, en la quinta sección se introduce la arquitectura de la red de detección, detallando su funcionamiento y las configuraciones empleadas en el proyecto.

Finalmente, el capítulo concluye con un apartado de conclusiones en el que se resumen los principales desafíos abordados y los aspectos clave que influyen en el., preparando el terreno para los experimentos y evaluaciones que se presentarán en los siguientes capítulos.

3.1 Reconocimiento de señales de tráfico

En esta sección vamos a indagar más en profundidad sobre el reconocimiento de señales de tráfico. Como vimos en el capítulo anterior, este problema se ha afrontado desde los primeros estudios publicados sobre el tema como dos problemas distintos pero complementarios.

El enfoque adoptado de dividir el reconocimiento de señales de tráfico en dos tareas, detección y clasificación, se fundamenta en la naturaleza del problema y en la necesidad de optimizar el rendimiento en cada etapa. La detección permite localizar de manera precisa las señales presentes en cada imagen, lo que es crucial en un entorno dinámico como el de la conducción. Una vez identificadas las señales, la clasificación se encarga de determinar su tipo, asegurando así que el sistema pueda interpretar correctamente la señalización. Este enfoque modular facilita el entrenamiento específico de cada etapa, permite evaluar y mejorar de forma independiente las redes diseñadas para cada tarea, y se alinea con las metodologías empleadas en estudios previos sobre reconocimiento de señales de tráfico.

En primer lugar, es fundamental abordar el problema de la detección, que consiste en identificar si un fotograma específico, capturado por la cámara del vehículo, contiene una señal de tráfico. Este proceso implica analizar cada imagen de manera sistemática para determinar la presencia de señales, lo cual puede ser complejo debido a factores como las variaciones en la iluminación, la presencia de sombras, o la posición y tamaño de las señales dentro de la imagen.

Una vez que se ha confirmado la presencia de una señal en un fotograma y se ha delimitado la región específica donde se encuentra, surge el segundo problema: la clasificación. Este paso implica determinar a qué tipo de señal corresponde la detección realizada.

Para el problema de detección, hemos utilizado la arquitectura de YOLOv8 que nos provee de distintos modelos de detección. Además, nos ayuda a abstraernos un poco de las entrañas de la arquitectura y de sus modelos, facilitándonos el uso de ellos ya sea por librería de Python o usando su interfaz de comandos (CLI). Desde ambas podremos, entrenar el modelo de detección, validar sus resultados obteniendo métricas sobre el rendimiento del mismo, predecir desde distintos formatos, ya sea en tiempo real, sobre un vídeo o una imagen, o extraer el modelo a distintos formatos para su implementación en dispositivos.

En el contexto del problema de clasificación, hemos desarrollado y entrenado varias redes neuronales convolucionales, ajustando diferentes parámetros para optimizar su rendimiento. A lo largo de este proceso, hemos llevado a cabo un benchmarking exhaustivo que nos ha permitido evaluar y comparar el desempeño de cada red. Este enfoque nos ha permitido identificar las virtudes de cada red, las cuales hemos puesto a prueba frente a los modelos de clasificación de YOLOv8. La comparación con YOLOv8 nos ha proporcionado una referencia robusta para evaluar la competitividad de nuestras redes, así como para determinar sus ventajas y limitaciones en la aplicación del reconocimiento de señales de tráfico.

Antes de profundizar en los detalles de la red de detección y las redes de clasificación, es fundamental entender las diferentes capas y funciones que comúnmente se emplean en las redes neuronales convolucionales. También explicaremos en detalles las métricas utilizadas para evaluar los distintos modelos que hemos entrenado.

3.2 Tipos de capas

En esta sección vamos a describir los diferentes tipos de capas que usualmente componen las redes neuronales convolucionales y cual es su función.

3.2.1 Capa de convolución

La capa de convolución es el componente fundamental de una red neuronal convolucional[14]. Esta capa soporta la mayor parte de la carga computacional de la red.

En la Figura 3.1 podemos observar un esquema en el que se muestra como a partir de una matriz de entrada y un *kernel* da lugar a la salida.

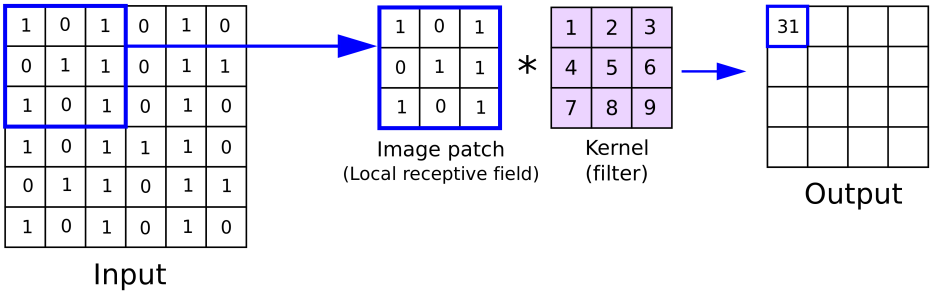

Figura 3.1 Convolución. Figura obtenida del artículo [11].

En esta capa, se realiza un producto escalar entre dos matrices: una matriz corresponde al conjunto de parámetros que se aprenden, conocidos como *kernel* o filtro, y la otra matriz es la porción restringida del campo receptivo. El filtro es dimensionalmente más pequeño que la imagen. Si la imagen está compuesta por tres canales RGB, la altura y el ancho del filtro serán espacialmente pequeños, pero su profundidad abarcará los tres canales.

Durante el proceso de propagación hacia adelante, el filtro se desliza a lo largo de la altura y el ancho de la imagen, generando la representación de la imagen en esa región receptiva. Este proceso produce una representación bidimensional de la imagen conocida como mapa de activación, que refleja la respuesta del filtro en cada posición espacial de la imagen.

El tamaño del desplazamiento del filtro se denomina *stride*. Un *stride* mayor significa que el filtro se moverá más posiciones en cada paso, lo que resulta en un mapa de activación más pequeño. Por otro lado, un *stride* más pequeño permite que el filtro capture más detalles, pero genera un mapa de activación más grande.

Para controlar cómo el filtro se aplica en los bordes de la imagen, se usa el término *padding*. El rellenado implica generar valores, normalmente ceros, alrededor de la imagen antes de aplicar la convolución. Hay dos enfoques comunes de *padding*:

- ***Valid Padding*:** no se agregan valores adicionales y el tamaño de la salida es más pequeño.
- ***Same Padding*:** se añaden suficientes ceros para que el mapa de activación tenga el mismo tamaño que la entrada original.

Para una entrada de tamaño W x W x D y D_{out} numero de *kernels* con un tamaño de F, con un stride de S y un rellenado de P, el tamaño del volumen de salida se puede determinar mediante la siguiente expresión:

$$\text{Tamaño de salida} = \left(\frac{W-F+2P}{S}+1\right) \times \left(\frac{W-F+2P}{S}+1\right) \times D_{out}$$

La capa de convolución en una red neuronal no solo es el componente fundamental, sino que también introduce varias funcionalidades clave que mejoran el rendimiento y la eficiencia de la red. Estas funcionalidades incluyen la interacción dispersa, el uso compartido de parámetros y la representación equivariante.

A diferencia de las capas tradicionales de redes neuronales, que utilizan la multiplicación de matrices donde cada unidad de salida interactúa con cada unidad de entrada, las redes neuronales convolucionales implementan una interacción dispersa lograda mediante el *kernel*.

Por ejemplo, mientras una imagen puede contener millones de píxeles, el *kernel* puede detectar información relevante procesando solo decenas o cientos de píxeles. Este enfoque reduce la cantidad de parámetros necesarios para almacenar, lo que disminuye los requisitos de memoria del modelo y mejora su eficiencia estadística.

En una CNN, si calcular una característica en un punto específico de la imagen es útil, también lo será en otros puntos de la misma. Para lograrlo, los pesos aplicados en una región de la imagen se comparten a lo largo de toda la imagen. Esto significa que, a diferencia de las redes tradicionales donde los pesos se utilizan una vez y luego no se reutilizan, en una red convolucional, los mismos pesos se aplican repetidamente en diferentes posiciones de la imagen. Esto no solo reduce la cantidad de parámetros, sino que también permite que la red detecte características similares en diferentes partes de la imagen.

Gracias al uso compartido de parámetros, las capas de una red neuronal convolucional adquieren una propiedad conocida como equivarianza ante la traslación. Esto significa que si la entrada se modifica mediante una traslación, la salida también se ajustará de manera correspondiente, manteniendo la coherencia en la representación de las características detectadas. En otras palabras, si un objeto en la imagen se desplaza, la red aún podrá reconocerlo correctamente en su nueva posición. Estas funcionalidades hacen que la capa de convolución sea especialmente eficaz para extraer características importantes de las imágenes y para optimizar el procesamiento en redes neuronales, especialmente en tareas relacionadas con la visión por computadora.

3.2.2 Capa de agrupación

La capa de *pooling* se encarga de reemplazar la salida de la red en ubicaciones específicas, obteniendo un resumen estadístico de las salidas cercanas. Esta capa es crucial para reducir el tamaño espacial de la representación, lo que disminuye la cantidad de cómputo y los pesos necesarios. La operación de *pooling* se aplica de manera individual a cada segmento de la representación.

Existen varias funciones de *pooling*, como el promedio de la ventana, el mínimo dentro de la ventana y un promedio ponderado basado en la distancia desde el píxel central. Sin embargo, el método más utilizado es el *max pooling*, que reporta el valor máximo de la salida dentro de la ventana.

Por ejemplo, si tenemos un mapa de activación de tamaño $W \times W \times D$, un *kernel* de *pooling* de tamaño espacial F, y un stride S, el tamaño del volumen de salida se puede determinar mediante la siguiente expresión:

$$\text{Tamaño de salida} = \left(\frac{W-F}{S}+1\right) \times \left(\frac{W-F}{S}+1\right) \times D$$

En todos los casos, el pooling ofrece cierta invariancia a la traslación, lo que significa que un objeto puede ser reconocido independientemente de dónde aparezca en el fotograma. Esta propiedad es fundamental para mejorar la capacidad de la red neuronal en la detección de objetos y características, independientemente de su posición dentro de la imagen.

3.2.3 Capa totalmente conectada

La capa totalmente conectada, también conocida como capa densa, es un componente fundamental en las redes neuronales en el que cada neurona de la capa anterior se conecta con todas las neuronas de la capa actual. Esta completa interconexión es lo que le otorga el nombre de "completamente conectada". La capas totalmente conectadas se suelen ubicar en las etapas finales de la arquitectura de una red neuronal y desempeñan un papel crucial en la generación de las predicciones finales del modelo.

3.2.4 Capa de normalización

La capa de normalización por lotes, conocida como *Batch Normalization* (BN)[15], es una técnica ampliamente utilizada en redes neuronales convolucionales para normalizar la entrada de cada neurona, de manera que tenga una media de cero y una varianza unitaria. Este proceso es fundamental porque contribuye a estabilizar el proceso de aprendizaje, evitando el problema del desplazamiento interno de covariables, que ocurre cuando la distribución de las entradas a una capa cambia durante el entrenamiento. La normalización que realiza BN se basa en la media y la desviación estándar del lote de datos de entrada y se aplica generalmente después de las capas convolucionales y totalmente conectadas.

Entre los efectos positivos que tiene la capa de normalización por lotes en las redes neuronales convolucionales, destaca su capacidad para acelerar el proceso de entrenamiento al reducir el desplazamiento interno de covariables, lo que permite utilizar tasas de aprendizaje más altas. Además, BN ayuda a regularizar el modelo, haciéndolo menos susceptible al sobreajuste mejorando así su capacidad de generalización. Esto se traduce en un mejor desempeño del modelo cuando se enfrenta a datos no vistos previamente.

La capa de normalización permite que el modelo sea menos sensible a los valores iniciales de los pesos, lo cual es beneficioso para la estabilidad del entrenamiento. En resumen, la normalización por lotes es una técnica poderosa y esencial en las redes neuronales convolucionales, ya que estabiliza el proceso de aprendizaje y mejora el rendimiento general del modelo al normalizar las entradas de cada neurona, reducir el desplazamiento interno de covariables y regularizar el modelo, lo que conduce a una mejor generalización en nuevos conjuntos de datos.

3.2.5 Funciones de activación

Su principal función es introducir no linealidad en el modelo, lo que permite a la red aprender y modelar relaciones complejas y no lineales en los datos. En una red neuronal, cada neurona toma una entrada, realiza una operación y luego aplica una función de activación a la salida de esa operación. Esta función de activación transforma el valor resultante en un nuevo valor, que luego se pasa a la siguiente capa de la red. Sin funciones de activación, la red sería simplemente una combinación lineal de las entradas, lo que limitaría severamente su capacidad para resolver problemas complejos, como la clasificación de imágenes o el reconocimiento de patrones.

En las CNN, las funciones de activación se aplican generalmente después de cada operación de convolución y/o después de cada operación de agrupamiento. Esto se hace para romper la linealidad inherente a las operaciones de convolución y pooling, permitiendo que la red capture y represente patrones más complejos.

Existen varios tipos de funciones de activación, a continuación vamos a listar algunas de ellas:

1. **Sigmoide:** La función sigmoide es $\sigma(\kappa) = \frac{1}{1+e^{-\kappa}}$. El rango de salida está entre 0 y 1. La principal ventaja es que suaviza la salida y es útil para tareas donde se necesita una interpretación probabilística.
2. **ReLu (Rectified Linear Unit):** Se define como $f(\kappa) = \max(0, \kappa)$. La función de activación ReLu es la más usada en las CNN debido a su simplicidad de cálculo y su eficiencia.

3. **LeakyReLu:** Similar a la ReLu pero en lugar de hacer que los valores negativos sean 0, les asigna un pequeño valor negativo. Se define como $f(\kappa) = \text{máx}(0.01\kappa, \kappa)$.

4. **Softmax:** Se utiliza comúnmente en la última capa de una red con tarea de clasificación. Convierte un vector de valores en un vector de probabilidades que suman 1. Su función se define como: $\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$

5. **GELU:** GELU combina características de linealidad y no linealidad de una manera suave y basada en la probabilidad. GELU aplica una activación que modela la probabilidad de una entrada aleatoria de seguir una distribución normal gaussiana. Esto significa que las entradas se multiplican por una probabilidad basada en la función de error gaussiana, permitiendo una transición suave entre valores negativos y positivos. En términos prácticos, GELU asigna pesos más pequeños a entradas negativas que están cerca de cero, en lugar de simplemente descartarlas como hace ReLU.

   La función GELU se define como:

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x \cdot \frac{1}{2}\left(1 + \frac{2}{\sqrt{\pi}} \cdot \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt\right)$$

   Donde $\phi(x)$ es al función de distribución acumulativa de la distribución normal estándar y erf es la función de error gaussiana.

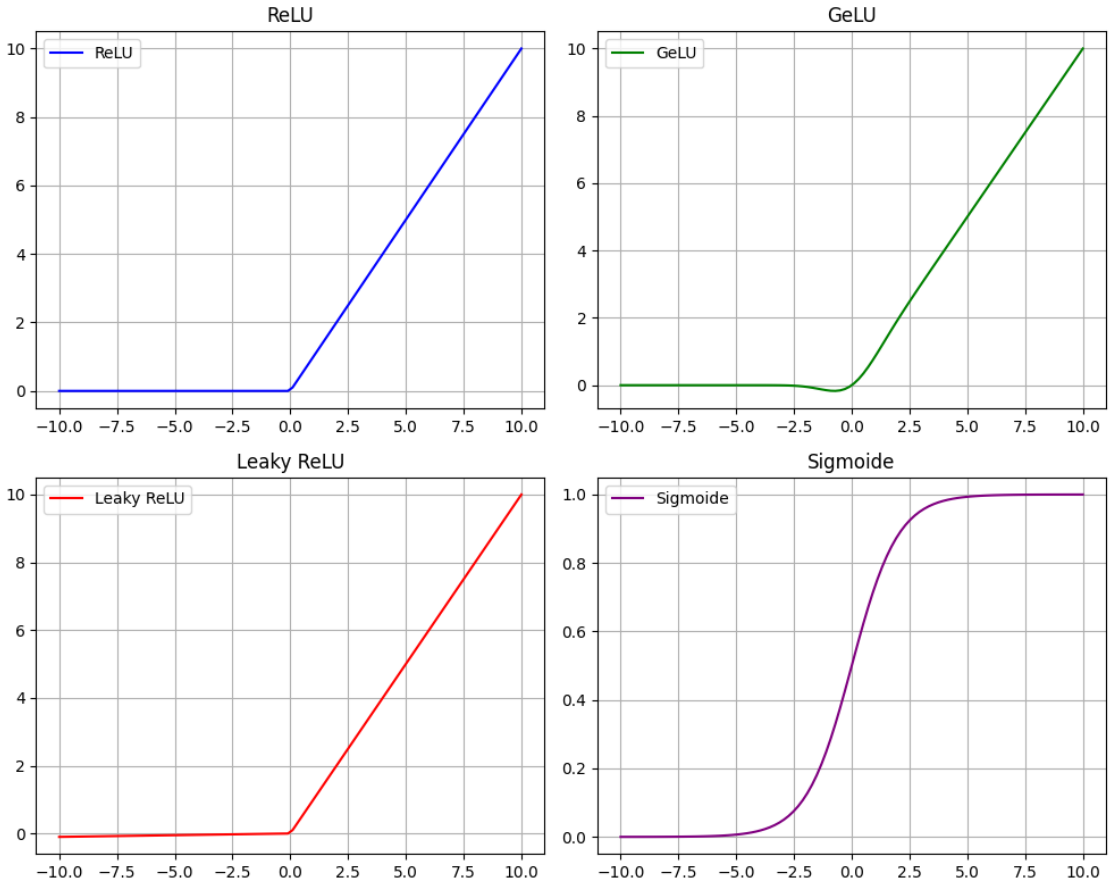

Figura 3.2 Funciones de activación.

En la Figura 3.2 se muestra en la esquina izquierda la representación gráfica de la función de activación ReLU, en la esquina derecha la función de activación GeLu, en la parte inferior izquierda la función de activación LeakyReLu y en la parte inferior derecha la función de activación Sigmoide.

3.3 Métricas

En el ámbito de la IA, las métricas son criterios cuantitativos utilizados para evaluar el rendimiento de modelos y algoritmos de IA. Estas métricas permiten medir la precisión, eficiencia, robustez y generalización de un modelo, entre otras características. Son fundamentales para comprender cómo de bien o mal se desempeña un modelo en tareas específicas y para comparar diferentes modelos entre sí.

3.3.1 Métricas para problemas de clasificación

La matriz de confusión[8] es una tabla que compara las clasificaciones reales con las predicciones realizadas por el modelo, permitiendo visualizar claramente el número de aciertos y errores. Esta herramienta es especialmente útil para evaluar el rendimiento de un modelo, ya que un alto número de elementos en la diagonal que va de la esquina superior izquierda a la inferior derecha indica un buen desempeño. Además, la matriz de confusión permite identificar si el modelo tiende a confundir ciertas clases con otras similares, proporcionando una visión detallada de sus posibles debilidades.

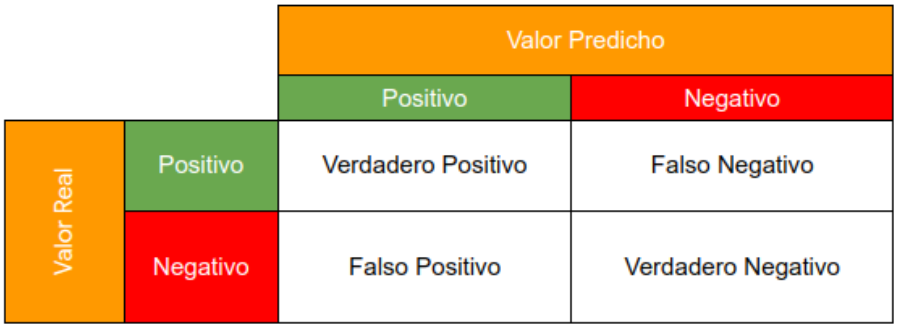

Figura 3.3 Matriz de confusión.

En la Figura 3.3 se muestra una matriz de confusión, donde se comparan las etiquetas reales de los datos contra las predicciones realizadas por un modelo de clasificación. A partir de esta matriz, podemos definir conceptos fundamentales utilizados en el campo de la inteligencia artificial:

- **Verdadero Positivo (*True Positive, TP*):** Casos donde el modelo predice correctamente una muestra positiva como positiva.
- **Falso Positivo (*False Positive, FP*):** Casos donde el modelo predice incorrectamente una muestra negativa como positiva.
- **Verdadero Negativo (*True Negative, TN*):** Casos donde el modelo predice correctamente una muestra negativa como negativa.
- **Falso Negativo (*False Negative, FN*):** Casos donde el modelo predice incorrectamente una muestra positiva como negativa.

La matriz de confusión no solo nos permite identificar los casos de verdadero positivo, falso positivo, verdadero negativo y falso negativo, sino que también es fundamental para calcular varias métricas de evaluación que nos ayudan a comprender mejor el rendimiento de un modelo de clasificación. Entre estas métricas, las más relevantes son la Exactitud (*Accuracy*), la Precisión (*Precision*), el Sensibilidad(*Recall*) y el *F1-Score*.

- ***Accuracy*:** La exactitud mide la proporción de predicciones correctas sobre el total de predicciones realizadas. Es una métrica global que indica qué tan bien el modelo clasifica tanto las muestras positivas como las negativas.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  Esta métrica es útil cuando las clases están equilibradas, pero puede ser engañosa en conjuntos de datos desequilibrados, donde una clase es mucho más frecuente que la otra.

- **Precisión:** La precisión expresa la proporción de casos en los que el modelo predice que un individuo es positivo y, efectivamente, lo es. En otras palabras, la precisión nos indica cuánta confianza podemos tener en el modelo cuando predice que un individuo es positivo.

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

- ***Recall*:** El *recall* mide la proporción de verdaderos positivos detectados entre todos los positivos reales. Es crucial en situaciones donde es importante minimizar los falsos negativos, como en diagnósticos médicos.

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

- ***F1-Score*:** El *F1-Score* es la media armónica de la precisión y el recall. Es una métrica equilibrada que es útil cuando se necesita un balance entre precisión y recall, especialmente en conjuntos de datos desequilibrados.

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

3.3.2 Métricas problemas de detección

La Intersección sobre Unión[7] es una métrica fundamental para tareas como la detección y segmentación de objetos. Desempeña un papel crucial en la evaluación de la calidad y precisión de estos modelos.

La intersección sobre unión se calcula tomando el área de intersección de dos cajas delimitadoras y dividiéndola por el área de la unión de estas cajas. La expresión de la intersección sobre unión se expresa de la siguiente manera:

$$\text{Intersección sobre unión} = \frac{\text{Área de Intersección}}{\text{Área de Unión}}$$

Esta expresión cuantifica esencialmente el grado de superposición entre la caja delimitadora predicha y la caja delimitadora de referencia. Proporciona un valor numérico que indica qué tan bien se alinea la predicción del modelo con la ubicación real del objeto. Un puntaje de intersección sobre unión más alto indica una mejor coincidencia entre las cajas predicha y real, lo que significa una mayor precisión en la localización.

En la Figura 3.4 se muestra en un gráfico visual la expresión de la Intersección sobre Unión.

La intersección sobre unión es una métrica esencial porque mide la capacidad del modelo para localizar objetos con precisión dentro de las imágenes. En cuanto a la detección de objetos, la intersección sobre unión evalúa qué tan bien el modelo identifica las posiciones de los objetos. En las tareas de segmentación, evalúa la capacidad del modelo para distinguir objetos de sus fondos.

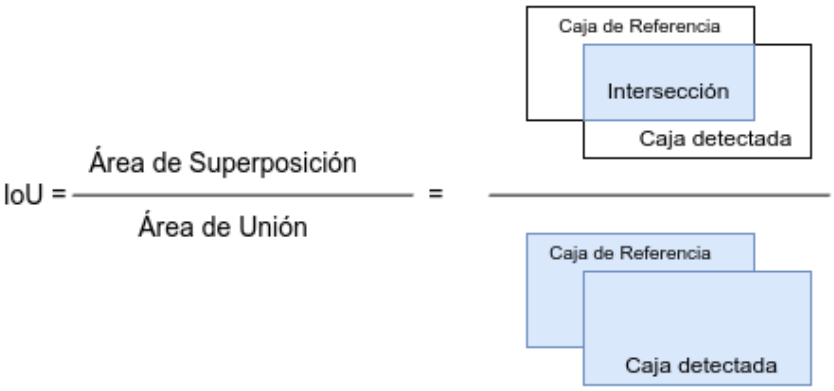

Figura 3.4 Intersección sobre Unión.

La Precisión Promedio y la Precisión Media Promedio[2] actúan como indicadores críticos para medir la capacidad de un modelo en identificar y localizar objetos con precisión dentro de las imágenes.

La precisión promedio analiza el compromiso entre precisión y sensibilidad al evaluar la precisión de un modelo de detección de objetos a lo largo de varios niveles de sensibilidad. La precisión indica la exactitud de las predicciones positivas del modelo, mientras que el sensibilidad cuantifica la capacidad del modelo para identificar todos los objetos relevantes. La precisión promedio logra un equilibrio armonioso entre falsos positivos y falsos negativos, encapsulando las complejidades del rendimiento del modelo. Lo hace calculando valores de precisión-sensibilidad en diferentes umbrales de confianza, formando una curva de precisión-sensibilidad, cuyo área bajo la curva representa la precisión promedio; cuanto mayor sea el área bajo la curva mejor rendimiento tendrá el modelo.

Si la curva la nombramos como $p(r)$, la ecuación que define la precisión promedio queda como:

$$\text{Precisión Promedio} = \int_0^1 p(r)dr$$

La precisión media promedio representa el promedio de las precision promedio calculadas para un valor dado de umbral de la intersección sobre la unión. Si el conjunto de datos contiene N clases, la precisión media promedio se obtiene promediando las precisiones promedio de cada clase individualmente. La ecuación que define este cálculo es:

$$\text{Precisión Media Promedio} = \frac{1}{N}\sum_{i=1}^{N} \text{Precisión Promedio}_i$$

donde $\text{Precisión Promedio}_i$ es la Precisión Promedio de la clase i.

3.4 Red de clasificación

Como se ha mencionado previamente, en el proceso de reconocimiento de señales de tráfico, una vez delimitada la señal, es necesario clasificarla correctamente indicando el tipo de señal correspondiente. Este proceso es realizado por las redes de clasificación.

Para abordar este problema de clasificación de imágenes, se ha empleado, por un lado, la arquitectura YOLOv8 con sus cinco modelos de clasificación. Por otro lado, se han desarrollado una serie de redes neuronales propias, que fueron comparadas entre sí a través de un proceso de *benchmarking*. Posteriormente, estas redes fueron contrastadas con YOLOv8 para evaluar su competitividad frente

al estado del arte, analizando las ventajas y desventajas de las redes desarrolladas en comparación con la arquitectura.

3.4.1 Redes de clasificación YOLOv8

YOLOv8 contiene 5 modelos de redes neuronales para la clasificación, todas ellas vienen preentranadas con el conjunto de datos ImageNet.

Tabla 3.1 Comparación de modelos YOLOv8 en términos de tamaño, precisión y velocidad.

Modelo	Tamaño (px)	Top1 (%)	Velocidad CPU (ms)	Nº Parámetros(M)
YOLOv8n-cls	224	69,0	12,9	2,7
YOLOv8s-cls	224	73,8	23,4	6,4
YOLOv8m-cls	224	76,8	85,4	17,0
YOLOv8l-cls	224	76,8	163,0	37,5
YOLOv8x-cls	224	79,0	232,0	57,4

En la Tabla 3.1 se muestran los cinco modelos de clasificación de YOLOv8 con algunas de sus características. La columna `Modelo` contiene los nombres de las redes de YOLOv8 de clasificación podemos diferenciarlas de las de detección porque utilizan el sufijo -cls. La columna `Tamaño(px)` especifica el tamaño de las imágenes de entrada, la columna `Top1` indica el *Accuracy* del modelo en el conjunto de datos ImageNet, la columna `Velocidad CPU` indica el tiempo que tarda el modelo exportado a formato ONNX en realizar una predicción en una instancia de *Amazon EC2 P4d* y la columna `Nº Parámetros(M)` indica el número de millones de parámetros que tiene el modelo.

En dicha tabla, se puede observar que la principal diferencia entre los modelos reside en el número de parámetros que poseen. En general, un mayor número de parámetros tiende a mejorar la precisión del modelo, aunque esto también implica un incremento en el tiempo de entrenamiento y una mayor latencia durante la ejecución. En este TFG, se entrenarán los cinco modelos utilizando el conjunto de datos GTSDB, y posteriormente se compararán con las redes neuronales convolucionales desarrolladas específicamente para este estudio.

Es importante destacar que ningún modelo es intrínsecamente mejor que otro; la elección del modelo adecuado dependerá siempre del caso de uso específico. Si la prioridad es la velocidad de ejecución, los modelos más ligeros serán la opción preferida. En cambio, si se busca maximizar la precisión, será más adecuado optar por los modelos con un mayor número de parámetros, a pesar de que esto conlleve un mayor coste computacional.

La estructura de los modelos de clasificación de YOLOv8 es la misma para los 5 casos, lo único que cambia es el número de parámetros de sus distintos bloques, esto lo podemos corroborar cuando ejecutamos el entrenamiento para el conjunto de datos deseado. YOLOv8 imprimirá por pantalla la Tabla 3.2 donde detalla la red a utilizar.

La primera columna `Desde` hace referencia a la capa de la que se obtiene los datos, la columna `Nº repeticiones` indica el número de veces que se repite el bloque, la columna `Parámetros` indica el número de parámetros que tiene el bloque, la columna `Módulo` indica el tipo de bloque que se está utilizando y la columna `Argumentos` indica los argumentos que se le pasan al bloque.

En esta tabla podemos ver la red está formada por 10 bloques y sus parámetros de entrada. Estos bloques son clases, que consultando el código fuente del directorio de github de YOLOv8 podemos ver que operaciones se realizan dentro de ellos. A continuación, vamos a explicar en detalle cada uno de estos bloques:

El bloque `Conv` es el bloque básico de convolución estándar, encargado de aplicar una serie de operaciones a los datos de entrada. Está compuesto por tres subcomponentes: la convolución en sí misma, la normalización por lotes y la función de activación.

Tabla 3.2 Estructura de la red con sus bloques, parámetros y argumentos.

Desde	Nºrepeticiones	Parámetros	Módulo	Argumentos
-1	1	464	ultralytics.nn.modules.conv.Conv	[3, 16, 3, 2]
-1	1	4.672	ultralytics.nn.modules.conv.Conv	[16, 32, 3, 2]
-1	1	7.360	ultralytics.nn.modules.block.C2f	[32, 32, 1, True]
-1	1	18.560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
-1	2	49.664	ultralytics.nn.modules.block.C2f	[64, 64, 2, True]
-1	1	73.984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
-1	2	197.632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
-1	1	295.424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
-1	1	460.288	ultralytics.nn.modules.block.C2f	[256, 256, 1, True]
-1	1	385.323	ultralytics.nn.modules.head.Classify	[256, 43]

- **Convolución:** La operación convolucional toma una entrada de $c1$ canales y la transforma a una salida de $c2$ canales utilizando un *kernel* de tamaño k y un stride s. Se aplica *padding* automático y dilatación, si es necesario.
- **Normalización por Lotes:** Después de la convolución, los datos pasan por una capa de normalización por lotes, que estabiliza y acelera el entrenamiento, reduciendo la variación interna de covariables.
- **Activación:** La activación predeterminada es SiLU, aunque es posible personalizarla según las necesidades. La activación no lineal introduce la capacidad de representar relaciones complejas entre las características.

El bloque `C2f` es una variante eficiente del bloque residual de tipo `CSP` *Cross-Stage Partial* diseñado para mejorar la eficiencia del modelo sin comprometer su capacidad de representación. Este bloque implementa dos convoluciones y n bloques internos que permiten un procesamiento más rápido.

- **Convoluciones Iniciales:** El bloque comienza con una primera convolución que reduce el número de canales de entrada a la mitad para optimizar el procesamiento. Se utilizan dos ramas de convolución para dividir los datos, lo que facilita la combinación posterior.
- **Módulos Residuales:** El bloque incluye una lista de bloques residuales *Bottleneck*, que se aplican secuencialmente a una parte de los datos. Estos módulos son responsables de realizar transformaciones adicionales, extrayendo características profundas.
- **Convolución Final:** Después de que los datos hayan sido procesados por los módulos internos, una última convolución los combina y los transforma al número de canales de salida deseado.

El uso de conexiones residuales dentro del bloque `C2f` permite que la red aprenda con mayor facilidad características profundas y complejas, manteniendo al mismo tiempo la eficiencia computacional. Además, al realizar el procesamiento en varias etapas y ramas, este bloque facilita la fusión de características a diferentes niveles de abstracción.

El bloque de clasificación (`Classify`) es la parte final de la arquitectura y se encarga de reducir la dimensionalidad de las características extraídas para generar las predicciones finales. Este módulo transforma los datos a partir de un tensor tridimensional hasta producir un vector con las probabilidades para cada clase.

- **Convolución:** El bloque comienza con una convolución que reduce las características extraídas a 1.280 canales, siguiendo el estándar del modelo *EfficientNet-B0*, lo que permite una mayor eficiencia computacional.

- **Pooling Adaptativo:** Se aplica la clase de pytorch *AdaptiveAvgPool2d* para reducir la resolución espacial de los datos a un solo píxel por canal, creando un vector de características que resuma toda la información relevante de la imagen.
- **Dropout:** Para prevenir el sobreajuste, se utiliza una capa de dropout, que introduce regularización durante el entrenamiento, apagando aleatoriamente conexiones en la red.
- **Clasificación lineal:** Finalmente, una capa totalmente conectada toma el vector resultante y lo convierte en una salida con tantas dimensiones como clases tenga el conjunto de datos. Durante la inferencia, esta salida se pasa por una función softmax para convertirla en probabilidades.

Este bloque es el encargado de proporcionar las predicciones finales, convirtiendo las características procesadas por la red en resultados concretos que indican la probabilidad de cada clase.

3.4.2 Redes de clasificación creadas

Como se ha mencionado en la introducción de esta sección, para la tarea de clasificación se ha diseñado una serie de redes neuronales, a las cuales se les ha aplicado un proceso de benchmarking para identificar las más eficientes que será detallado en el siguiente capítulo.

En la Figura 3.5 se muestra la estructura de la red de clasificación base empleada, partiendo de esta estructura se han generado distintas variaciones. La descripción de la red está en formato `JSON`. Dichas descripciones en formato `JSON` son las usadas por el código para crear la red.

```
{
    "layers" :[
        {"type": "conv", "in_channels": 3, "out_channels": 16, "kernel_size": 3,
             "stride": 1, "padding": "same"},
        {"type": "batchnorm", "out_channels": 16},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},

        {"type": "conv", "in_channels": 16, "out_channels": 32, "kernel_size":
            3, "stride": 1, "padding": "same"},
        {"type": "batchnorm", "out_channels": 32},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},

        {"type": "conv", "in_channels": 32, "out_channels": 64, "kernel_size":
            3, "stride": 1, "padding": "same"},
        {"type": "batchnorm", "out_channels": 64},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},

        {"type": "flatten"},
        {"type": "fc","in_features": 3136, "out_features": 1024},
        {"type": "fc","in_features": 1024, "out_features": 512},
        {"type": "fc","in_features": 512, "out_features": 43}
    ]
}
```

Figura 3.5 Estructura de la red de clasificación base definida en formato `JSON`.

En la Figura 3.6 se presenta también la estructura base usada para crear las redes de clasificación pero en este caso de forma gráfica. Como se observa en dicha figura, esta estructura está compuesta

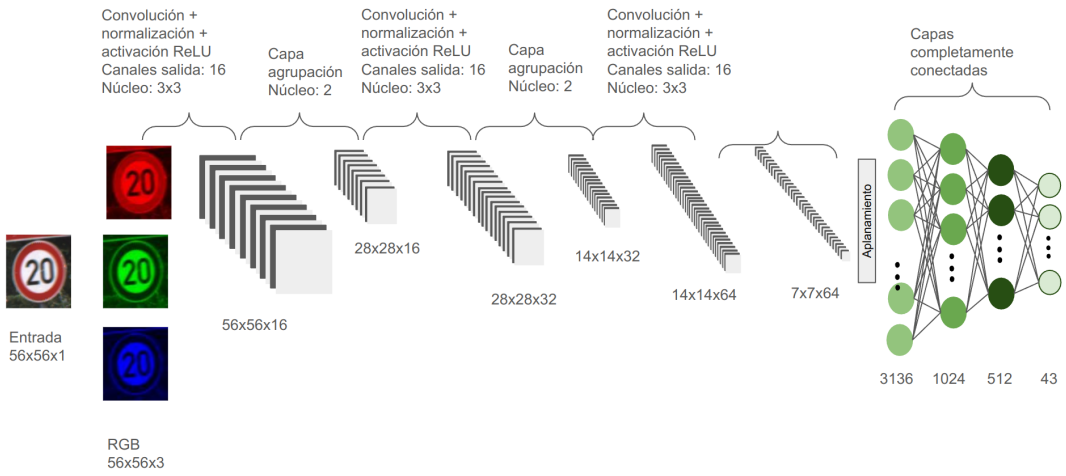

Figura 3.6 Estructura de la red de clasificación base.

por 3 bloques convolucionales. En cada bloque, el proceso comienza con una operación de convolución en la que, gracias al uso de *padding="same"*, se mantiene el tamaño de las dimensiones espaciales de la salida. A continuación, se aplica una capa de normalización por lotes con el fin de estabilizar las activaciones y acelerar el entrenamiento. Seguidamente, se introduce una función de activación ReLU, que permite introducir no linealidad en la red y, por lo tanto, habilitar la capacidad de la red para aprender relaciones más complejas. Finalmente, una capa de *pooling* reduce las dimensiones espaciales en un factor de dos, lo que no solo disminuye el costo computacional, sino que también ayuda a controlar el sobreajuste.

Tras los bloques convolucionales, se aplica una capa de aplanamiento que convierte la salida de la capa anterior en un vector unidimensional. Por último, se implementan tres capas completamente conectadas que van reduciendo gradualmente el número de neuronas hasta llegar a la capa de salida, que tiene 43 neuronas correspondientes al número de clases en el dataset GTSDB.

A partir de esta arquitectura base, se han generado cuatro variaciones adicionales de la red, con el objetivo de explorar cómo afectan diferentes modificaciones a su rendimiento:

1. Reducción a dos bloques convolucionales en lugar de tres.
2. Uso de un solo bloque convolucional.
3. En esta modificación se aplica *pooling* cada dos bloques de convolucional, normalización por lote y función de activación. Además, debido al mayor número de parámetros se agrega una capa más, *fully connected*, al final. La red queda descrita en el formato `JSON` de la Figura 3.7
4. Partiendo de la primera red descrita, eliminación de las capas de normalización por lotes para evaluar su impacto en el comportamiento de la red

En la Figura 3.7 se muestra la estructura de la red de clasificación en una de las variaciones propuestas. En este caso, se ha reducido el número de bloques convolucionales a dos, manteniendo el resto de la estructura intacta.

De este modo, contamos con un total de cinco redes distintas. Estas cuatro variaciones más la inicial serán replicadas tres veces, modificando en cada conjunto la función de activación utilizada. En la primera serie de redes se empleará la función ReLU, en la segunda se usará LeakyReLU, y en la tercera se aplicará GELU. Esto nos permite evaluar el impacto de diferentes funciones de activación en el rendimiento de cada una de las variantes de la red

```
{
    "layers" :[
        {"type": "conv", "in_channels": 3, "out_channels": 16, "kernel_size": 3,
             "stride": 1, "padding": "same"},
        {"type": "batchnorm", "out_channels": 16},
        {"type": "activation", "function": "ReLU"},
        {"type": "conv", "in_channels": 16, "out_channels": 32, "kernel_size":
            3, "stride": 1, "padding": "same"},
        {"type": "batchnorm", "out_channels": 32},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},
        {"type": "conv", "in_channels": 32, "out_channels": 64, "kernel_size":
            3, "stride": 1, "padding": "same"},
        {"type": "batchnorm", "out_channels": 64},
        {"type": "activation", "function": "ReLU"},
        {"type": "conv", "in_channels": 64, "out_channels": 128, "kernel_size":
             3, "stride": 1, "padding": "same"},
        {"type": "batchnorm", "out_channels": 128},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},
        {"type": "flatten"},
        {"type": "fc","in_features": 25088, "out_features": 8192},
        {"type": "fc","in_features": 8192, "out_features": 1024},
        {"type": "fc","in_features": 1024, "out_features": 512},
        {"type": "fc","in_features": 512, "out_features": 43}
    ]
}
```

Figura 3.7 Estructura de la red de clasificación con *pooling* cada dos bloques de convolucional.

3.5 Red de detección

En el proceso de detección de señales de tráfico, no solo es necesario identificar la presencia de una señal, sino también localizarla en la imagen mediante la delimitación precisa de su posición.

Para solucionar este problema de detección de objetos, se ha empleado la arquitectura YOLOv8 en dos de sus modelos de detección. Cada uno de estos modelos fue entrenado con el objetivo de identificar y localizar correctamente las señales de tráfico dentro de las imágenes del conjunto de datos. Adicionalmente, los modelos también detectarán otras nueve clases entre las que se encuentran: semáforos, coches, personas, autobuses, motocicletas, camiones, trenes…

En la Tabla 3.3 podremos ver la comparación de los distintos modelos de detección que nos ofrece la arquitectura YOLOv8. De todos ellos se han utilizado los modelos YOLOv8n y YOLOv8m. Entrenando con el conjunto de datos BDD100K mencionado en el estado del arte.

Tabla 3.3 Comparación de modelos de detección de YOLOv8 en términos de tamaño, precisión y velocidad.

Modelo	Tamaño(px)	mAPval 50-95	Velocidad CPU(ms)	Parámetros (M)
YOLOv8n	640	37,3	0,99	3,2
YOLOv8s	640	44,9	1,20	11,2
YOLOv8m	640	50,2	1,83	25,9
YOLOv8l	640	52,9	2,39	43,7
YOLOv8x	640	53,9	3,53	68,2

Al igual que ocurre con los modelos de clasificación, los modelos de detección también comparten estructura y la diferencia entre ellos es el número de parámetros de sus diferentes bloques. Esta estructura se detalla al ejecutar el entrenamiento.

En la Tabla 3.4 podremos ver que bloques se utilizan, estos referencian a clases del código fuente e indagando en él podemos ver que operaciones se realizan en cada uno de ellos.

Tabla 3.4 Estructura de la red con sus bloques y los parámetros del modelo YOLOv8n.

Desde	Nºrepeticiones	Parámetros	Módulo	Argumentos
-1	1	1.392	ultralytics.nn.modules.conv.Conv	[3, 48, 3, 2]
-1	1	41.664	ultralytics.nn.modules.conv.Conv	[48, 96, 3, 2]
-1	2	111.360	ultralytics.nn.modules.block.C2f	[96, 96, 2, True]
-1	1	166.272	ultralytics.nn.modules.conv.Conv	[96, 192, 3, 2]
-1	4	813.312	ultralytics.nn.modules.block.C2f	[192, 192, 4, True]
-1	1	664.320	ultralytics.nn.modules.conv.Conv	[192, 384, 3, 2]
-1	4	3.248.640	ultralytics.nn.modules.block.C2f	[384, 384, 4, True]
-1	1	1.991.808	ultralytics.nn.modules.conv.Conv	[384, 576, 3, 2]
-1	2	3.985.920	ultralytics.nn.modules.block.C2f	[576, 576, 2, True]
-1	1	831.168	ultralytics.nn.modules.block.SPPF	[576, 576, 5]
-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, nearest]
[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
-1	2	1.993.728	ultralytics.nn.modules.block.C2f	[960, 384, 2]
-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, nearest]
[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
-1	2	517.632	ultralytics.nn.modules.block.C2f	[576, 192, 2]
-1	1	332.160	ultralytics.nn.modules.conv.Conv	[192, 192, 3, 2]
[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
-1	2	1.846.272	ultralytics.nn.modules.block.C2f	[576, 384, 2]
-1	1	1.327.872	ultralytics.nn.modules.conv.Conv	[384, 384, 3, 2]
[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
-1	2	4.207.104	ultralytics.nn.modules.block.C2f	[960, 576, 2]
[15, 18, 21]	1	3.781.486	ultralytics.nn.modules.head.Detect	[10, [192, 384, 576]]

Como se observa en la tabla, la estructura de las redes de detección de YOLOv8 está compuesta por 22 bloques. Algunos de estos bloques son compartidos con la red de clasificación, por lo tanto, vamos a hacer hincapié en explicar solo los bloques no comentados en el apartado anterior.

El bloque `SPPF` (*Spatial Pyramid Pooling - Fast*) es una versión optimizada de la capa SPP tradicional. Su objetivo es capturar características a múltiples escalas a partir de un solo mapa de características. Esto mejora la capacidad de la red para detectar objetos de diferentes tamaños en la imagen.

El bloque `SPPF` se compone de los siguientes elementos clave:

- **Convolución inicial:** Se aplica una convolución 1x1 que reduce el número de canales de entrada a la mitad, lo cual disminuye la carga computacional de las operaciones siguientes.
- ***MaxPooling* piramidal:** Se utilizan tres operaciones de *MaxPooling* con un tamaño de *kernel* 5 por defecto , que agregan características espaciales a diferentes escalas. Cada operación de *pooling* genera un mapa de características de tamaño reducido, que se concatena con el original.
- **Convolución final:** Se aplica una convolución final que combina las características provenientes de las distintas escalas, devolviendo el mismo número de canales que en la entrada.

El bloque `Upsample` realiza un remuestreo ascendente de las características para aumentar la resolución espacial de los mapas de características. Este proceso es fundamental para la combinación de características de diferentes niveles de la red, facilitando que características de bajo nivel (de alta

resolución) y de alto nivel (de baja resolución) se fusionen para mejorar la precisión en la detección de objetos pequeños.

En YOLOv8, la interpolación se realiza generalmente usando el modo *nearest neighbor*, que es una técnica simple y eficiente.

El bloque `Concat` es responsable de la fusión de características de diferentes capas de la red. Esta operación combina mapas de características de diferentes resoluciones o niveles de la arquitectura, lo que permite que la red integre información de diferentes etapas del procesamiento.

Las características de las capas anteriores, que tienen mayor resolución pero contienen menos abstracción, se concatenan con características de capas más profundas, que contienen información más abstracta pero de menor resolución. Esta combinación de información mejora la precisión de las detecciones, ya que permite a la red aprovechar tanto las características espaciales finas como las representaciones más globales.

El bloque `Detect` es el encargado de realizar las predicciones finales de la red YOLOv8. Este bloque genera las coordenadas de los *bounding boxes* y las probabilidades de clasificación para los objetos detectados. La implementación del bloque `Detect` está diseñada para ser flexible y eficiente, permitiendo predecir objetos en múltiples escalas de resolución.

Algunos elementos clave del bloque `Detect` son:

- **Predicciones de las clases:** Se genera un mapa de predicciones de clases, en el que cada celda de salida contiene la probabilidad de pertenencia a una clase determinada.
- **Predicciones de las cajas:** Además de las clases, también se predicen las coordenadas de las cajas delimitadoras, utilizando representaciones normalizadas de las coordenadas de los objetos dentro de la imagen.
- **Escalas múltiples:** Este bloque aprovecha múltiples escalas de resolución para predecir objetos, lo que permite mejorar la detección de objetos pequeños y grandes en la imagen.

El bloque `Detect` es el componente de la red que decide si existe un objeto en una región específica de la imagen y a qué clase pertenece, proporcionando las coordenadas del objeto en la imagen de entrada.

3.6 Conclusiones

En este capítulo se ha presentado el contexto del problema de reconocimiento y clasificación de señales de tráfico, un área de creciente interés dentro de las aplicaciones de la inteligencia artificial en la automoción. Se ha descrito detalladamente el doble enfoque necesario para resolver este problema: por un lado, la detección de señales en imágenes capturadas en tiempo real, y por otro, la correcta clasificación de dichas señales.

Para la tarea de detección, se ha utilizado la arquitectura YOLOv8, aprovechando su balance entre precisión y velocidad para garantizar un rendimiento adecuado en aplicaciones de conducción asistida. Este modelo ha sido entrenado con el conjunto de datos BDD100K, adaptado para la detección de señales y otros objetos en carretera.

En cuanto a la clasificación de señales, se ha desarrollado una serie de redes neuronales convolucionales personalizadas, que han sido entrenadas y evaluadas usando el conjunto de datos GTSDB. A lo largo del capítulo se han detallado las variaciones arquitectónicas implementadas, como la modificación del número de capas convolucionales, la eliminación de las capas de normalización por lotes, y la introducción de diferentes funciones de activación. Estas modificaciones se aplicaron con el objetivo de evaluar cómo diferentes configuraciones de la red afectan al rendimiento y, como se describe en el siguiente capítulo, pasarán por un proceso de *benchmarking* para analizar el desempeño de dichas variaciones. Además, las configuraciones que muestren un rendimiento prometedor

serán comparadas con los modelos de clasificación proporcionados por YOLOv8, asegurando que las redes desarrolladas sean competitivas y adecuadas para la clasificación de señales de tráfico.

4 Experimentos

En este capítulo se detalla el proceso experimental llevado a cabo para entrenar y validar los modelos de detección y clasificación de señales de tráfico.

Para la fase de detección de señales de tráfico, se ha seleccionado el dataset BDD100K descrito en la Sección 2.5 debido a su gran tamaño, diversidad geográfica y calidad de anotaciones, permitiendo analizar el rendimiento de los modelos en escenarios variados y realistas. Además, se detalla el tratamiento de datos necesario, incluyendo la conversión de anotaciones al formato requerido por YOLOv8. Se explica el proceso de entrenamiento y validación de dos de los modelos de detección de YOLOv8, los modelos YOLOv8m y YOLOv8n, descritos previamente en la Sección 3.5. Adicionalmente, se presentan gráficos sobre el desempeño de la red y ejemplos de predicciones de estos modelos de detección.

En la fase de clasificación, se ha empleado el dataset GTSDB 2.5, que incluye 43 clases de señales de tráfico. Esta sección describe el tratamiento de los datos previos al entrenamiento, incluyendo la creación de las divisiones de entrenamiento, validación y prueba.

Como se introdujo en la Sección 3.4.2, se diseñaron 15 redes neuronales distintas para la tarea de clasificación. En este capítulo, se detallan los procesos de entrenamiento y validación de estas redes. Para evaluar y optimizar el rendimiento de cada una, se realizó un proceso de *benchmarking* exhaustivo mediante una estructura que permite cargar diversas combinaciones de hiperparámetros desde un archivo de formato `JSON`, incluyendo optimizadores, tasa de aprendizaje, épocas y tamaño de lote. Este procedimiento facilitó la identificación de la combinación óptima de parámetros para maximizar el rendimiento de cada red. Durante el *benchmarking*, se registraron métricas como el *accuracy*, la *precision* , el *recall* y el *F1-score*, permitiendo seleccionar los mejores modelos para su posterior comparación con los modelos de clasificación de YOLOv8.

Asimismo, se describe el proceso de entrenamiento de los modelos de clasificación de YOLOv8, seguido de una comparativa entre los resultados obtenidos por ambos enfoques.

Este capítulo contiene un análisis comparativo de los resultados experimentales y una discusión de las ventajas y limitaciones de los modelos desarrollados frente a los modelos de referencia de YOLOv8 en el contexto de aplicaciones de conducción autónoma.

Por último, el capítulo concluye con la automatización del enlace entre la red de detección y la red de clasificación, asegurando un flujo continuo de procesamiento en tiempo real y evaluando la viabilidad del sistema para su aplicación en entornos reales.

4.1 Instalación de librerías y paquetes necesarios

Para llevar a cabo el experimento, se han utilizado diversas librerías y paquetes de Python además de una serie de software que se detallarán a continuación.

4.1.1 Librerías de Python

Tendremos que instalar la librería de Ultralytics. Desde la cual podremos hacer uso de los modelos de YOLOv8 y de las diferentes herramientas que nos proporcionan. Para instalar la librería, se puede hacer mediante el gestor de paquetes pip. Para ello, se ejecuta el siguiente comando en la terminal:

```
pip install ultralytics
```

Al instalar la librería de ultralytics, se nos instalarán automáticamente las dependencias necesarias para su funcionamiento. Entre ellas nos encontramos con las librerías de CUDA para su uso a través de python y otras libreria tales como torch, opencv, numpy…

El entrenamiento por CPU es mucho más lento que por GPU en el caso del tratamiento de imágenes, por lo que se recomienda tener una GPU compatible con CUDA. Para instalar CUDA, se puede seguir la guía de instalación de la página oficial de NVIDIA. En el caso de este TFG que ha sido desarrollado en linux bajo la distribución de Ubuntu, CUDA se nos instalará junto a los drivers de la GPU.

4.2 Red detección

Tras evaluar exhaustivamente los set de datos de señales de tráfico disponibles, se ha optado por utilizar el *datasets* BDD100K para la red de detección. Este conjunto de datos se describió en el capítulo Estado del Arte en la sección 2.5. Como se comento, dicho conjunto de datos está compuesto por 100.000 imágenes de alta resolución y divididos en 70.000 imágenes de entrenamiento, 10.000 de validación y 20.000 de test. Cada imagen puede contener múltiples objetos, los objetos están anotados con un caja delimitadora que definen donde se encuentra dicho objeto en la imagen. Existen 10 clases de objetos en este conjunto de datos.

En la Figura 4.1 se muestra un gráfico de barras con el número de instancias de cada clase que aparece en los datos de entrenamiento. Se observa que el conjunto de datos de entrenamiento está desbalanceado en cuanto a instancias de cada clase de objeto, aunque cabe destacar que las señales de tráfico tienen una aceptable cantidad de instancias que son el objeto de este TFG.

4.2.1 Tratamiento de datos

Al descargar el conjunto de datos BDD100K, se obtiene un archivo comprimido que en su interior se subdivide en las 3 carpetas correspondientes a los conjuntos de entrenamiento, validación y test. Las anotaciones de los conjuntos de entrenamiento y validación se encuentran en dos únicos archivos con formato `JSON` que comparten la misma estructura. Cada archivo es una lista de objetos, cada objeto tiene un *name* asociado que es el nombre de la imagen y una lista de etiquetas que contienen la información de la caja delimitadora y la clase del objeto. Este formato inicial, debe ser convertido al formato utlizado por YOLOv8.

YOLOv8 usa un archivo con extensión yaml para describir el *dataset*. En este archivo se especifica la ruta a los conjuntos de entrenamiento, validación y test. También se especifica el número de clases y se proporciona un nombre a cada id de clase.

Por otra parte, cada imagen del conjunto de datos de entrenamiento y validación deben estar etiquetadas. Las etiquetas en YOLOv8 se guardan en un archivo `txt` por cada imagen, con nombre igual al nombre de la imagen. Estas se ubican en un directorio llamado *labels* dentro de la carpeta de entrenamiento y validación, a la misma altura que la carpeta *images* que contendrá las imágenes. La caja delimitadora en el *dataset* BDD100K se encuentra en formato (X_{min}, Y_{min}, X_{max}, Y_{max}), donde (X_{min}, Y_{min}) es la esquina inferior izquierda y (X_{max}, Y_{max}) es la esquina superior derecha. En YOLOv8, la caja delimitadora se encuentra en formato (x, y, w, h), donde (x, y) es el centro de la

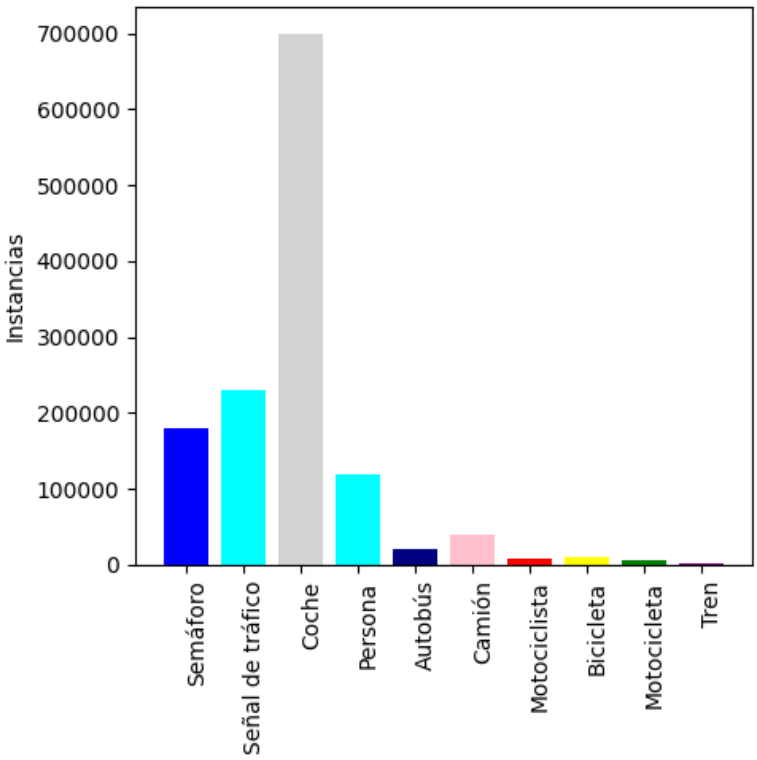

Figura 4.1 Instancias por categoría en *dataset* BDD100K.

caja y (w, h) son el ancho y alto de la caja. Para convertir de un formato a otro, se ha utilizado las siguientes ecuaciones:

$$x = \frac{X_{min} + X_{max}}{2}$$
$$y = \frac{Y_{min} + Y_{max}}{2}$$
$$w = X_{max} - X_{min}$$
$$h = Y_{max} - Y_{min}$$

Resumiendo, para cada imagen se crea un archivo `txt`, el archivo contendrá una línea por cada objeto etiquetado que haya en la imagen. En cada línea se especifica la clase del objeto y las coordenadas de la caja delimitadora en el formato (x, y, w, h).

4.2.2 Entrenamiento

Una vez tratados los datos, se pasará a la fase de entrenamiento. Para ejecutar el entrenamiento existen dos formas de hacerlo: A través de la terminal de comandos o a partir de la libreria Ultralytics para python. A continuación se muestra como sería el comando para entrenar un modelo de detección de YOLOv8 a través de la terminal:

```
yolo detect train data=$nombre_dataset.yaml$ model=$yolov8n.yaml$
```

Donde `nombre_dataset.yaml` es el archivo que contiene la información del *dataset* el cual contamos en la subsection anterior 4.2.1 que formato debe tener. Y `yolov8n.yaml` es el archivo que contiene el modelo de red que se va a utilizar. Los modelos de red disponibles en YOLOv8 son los descritos en la Tabla 3.3.

En la Figura 4.2 se muestra como podemos entrenar el modelo desde código python.

```
from ultralytics import YOLO

# Cargar un modelo
model = YOLO("yolo8n.yaml") # construir un nuevo modelo desde YAML

# Entrenar el modelo
results = model.train(data="coco8.yaml", epochs=100, imgsz=640)
```

Figura 4.2 Ejemplo de código de entrenamiento de YOLO en Python.

En la Tabla 4.1 se muestran algunos de los hiperparámetros que se pueden modificar en el entrenamiento, además se describe la utilidad de cada uno de ellos.

En el entrenamiento llevado a cabo se han utilizado los hiperparámetros por defecto, pero cabe comentar la gran versalitidad que nos proporciona YOLOv8 para modificar los hiperparámetros y ajustar el entrenamiento a nuestras necesidades. Aunque se podría haber usado *early stopping* con el argumento *patiente*, se ha preferido dejar el entrenamiento con un número fijo de épocas para poder comparar los resultados de los distintos modelos, ya que tras el entrenamiento YOLOv8 nos guardará los pesos de la red en la mejor ejecución y en la última ejecución.

Para ejecutar el entrenamiento en YOLOv8, es fundamental contar con un conjunto de datos de validación. En caso de no disponer de uno, YOLOv8 utiliza un conjunto predeterminado, lo cual podría afectar a la precisión del ajuste de la red.

En este estudio, se entrenaron dos modelos de YOLOv8: el modelo más ligero, YOLOv8n, y el modelo de tamaño mediano, YOLOv8m. La Tabla 3.3 muestra las características de los diferentes modelos de YOLOv8. Los dos únicos parámetros ajustados durante el entrenamiento fueron el número de épocas, fijado en 100 para ambos modelos, y el tamaño de lote, que se definió en función del porcentaje de memoria disponible. Gracias a este ajuste, YOLOv8 adapta automáticamente el tamaño de lote en función de la VRAM de la tarjeta gráfica, fijándose en ambos casos en un 60 % de la memoria de la GPU.

El entrenamiento de la red YOLOv8n, el modelo más liviano, tomó aproximadamente 10 horas en completarse, mientras que el entrenamiento del modelo mediano YOLOv8m requirió aproximadamente dos días. Debido a estos tiempos, no se entrenaron modelos de mayor complejidad, como YOLOv8x, cuya duración estimada se aproximaría a una semana. Además, es importante considerar que, a medida que aumenta el tamaño del modelo, también se incrementa el tiempo necesario para la inferencia. En aplicaciones de detección de señales de tráfico en tiempo real, es fundamental que el tiempo de inferencia sea lo más rápido posible. Asimismo, si se planease implementar el modelo en un dispositivo embebido, como una Raspberry Pi o cualquier dispositivo *edge*, los modelos de mayor tamaño podrían no ser viables.

La Figura 4.3 muestra la matriz de confusión normalizada de la red YOLOv8m evaluada en el conjunto de validación. En la matriz se observa como el modelo tiene un buen desempeño en la detección de las señales de tráfico. También se ve que debido al conjunto de datos usado y al desbalance entre números de instancias por clases, las clases que tienen mayor presencia son las que mejor resultado obtienen.

En cuanto al conjunto de datos de test nos encontramos con 10.000 imágenes sin etiquetar por lo tanto no se han podido evaluar los modelos en este conjunto de datos.

En la Figura 4.4 se presenta un ejemplo de las predicciones generadas por YOLOv8. La primera imagen del mosaico muestra la imagen original, una escena de tráfico común. A continuación, se observa la predicción generada por el modelo YOLOv8n, seguida por la imagen resultante

Tabla 4.1 Descripción de algunos argumentos para el entrenamiento de detección de YOLOv8.

Argumento	Valor por Defecto	Descripción
`model`	`None`	Especifica el archivo del modelo para entrenamiento. Acepta una ruta a un modelo preentrenado (.pt) o a un archivo de configuración (.yaml).
`data`	`None`	Ruta al archivo de configuración del *dataset*. Este archivo contiene parámetros específicos del *dataset*.
`epochs`	`100`	Número total de épocas de entrenamiento. Cada época representa un pase completo sobre el dataset.
`time`	`None`	Tiempo máximo de entrenamiento en horas. Si se establece, anula el argumento de `epochs`.
`patience`	`100`	Número de épocas que espera sin mejoras en las métricas de validación antes de detener el entrenamiento.
`batch`	`16`	Tamaño del lote, con tres modos: número entero, auto (utilización del 60 % de la memoria de la GPU), o una fracción especificada (por ejemplo, `batch=0.70`).
`imgsz`	`640`	Tamaño de imagen objetivo para el entrenamiento. Todas las imágenes se redimensionan a esta dimensión.
`save`	`True`	Habilita el guardado de puntos de control del entrenamiento y los pesos finales del modelo.
`save_period`	`-1`	Frecuencia de guardado de los puntos de control del modelo en épocas.
`cache`	`False`	Habilita la caché de las imágenes del *dataset* en memoria (True/ram) o en disco (disk).
`device`	`None`	Especifica el dispositivo de cómputo: una GPU, varias GPUs, CPU, o MPS (Apple Silicon).
`workers`	`8`	Número de hilos de trabajo para cargar datos (por RANK si es multi-GPU).
`project`	`None`	Nombre del directorio del proyecto donde se guardan los resultados del entrenamiento.
`name`	`None`	Nombre del experimento de entrenamiento. Crea un subdirectorio con este nombre.
`exist_ok`	`False`	Si es True, permite sobrescribir un directorio existente.
`pretrained`	`True`	Determina si se debe iniciar el entrenamiento con un modelo preentrenado.
`optimizer`	`áuto'`	Selección del optimizador para el entrenamiento (SGD, Adam, RMSProp, etc.).
`verbose`	`False`	Habilita salida detallada durante el entrenamiento.
`seed`	`0`	Establece la semilla aleatoria para asegurar la reproducibilidad.
`deterministic`	`True`	Fuerza el uso de algoritmos deterministas para garantizar la reproducibilidad.

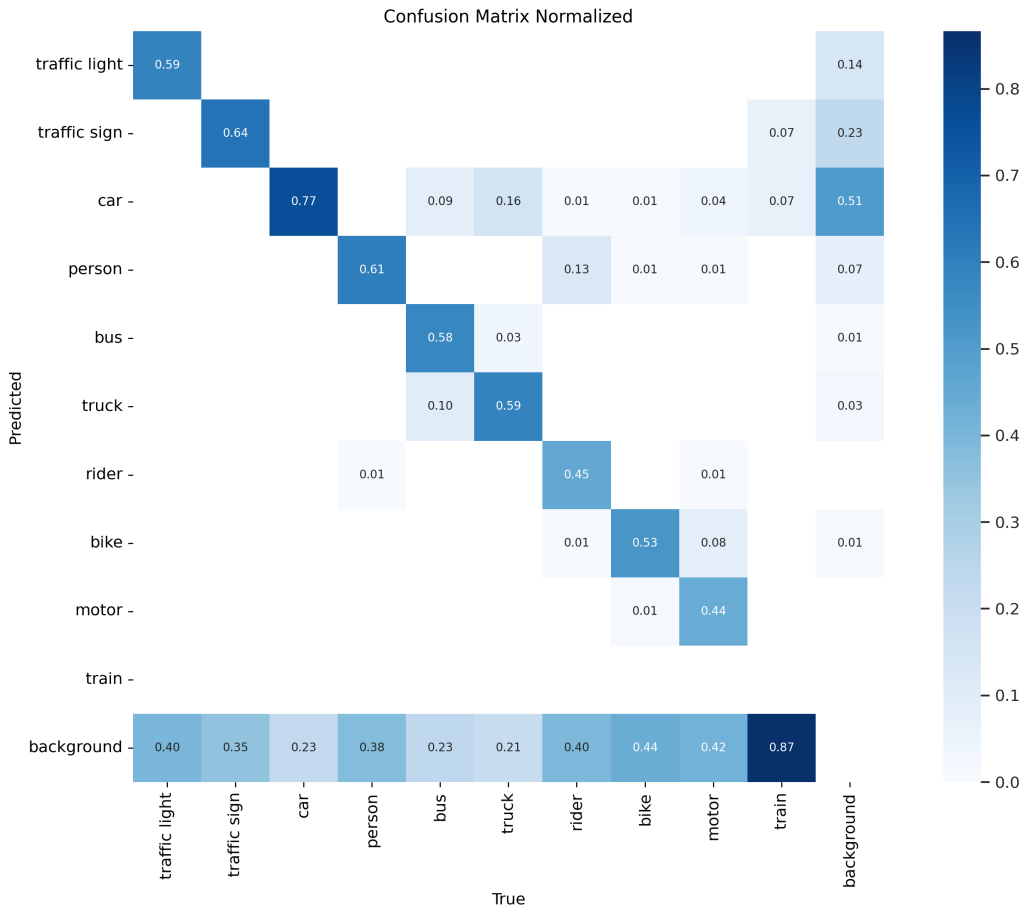

Figura 4.3 Matriz de confusión normalizada de YOLOv8m en el conjunto de validación.

del modelo YOLOv8m. En estas predicciones se aprecia una mejora en la precisión del modelo YOLOv8m en comparación con YOLOv8n.

Además de mostrar mayores umbrales de confianza, el modelo YOLOv8m logra identificar un vehículo en la esquina superior derecha de la imagen, parcialmente oculto por un motociclista, lo cual no es detectado por el modelo YOLOv8n. En ambos casos, se observa un buen rendimiento de los modelos de detección al delimitar e identificar correctamente el tipo de objeto y su posición en la imagen.

La Figura 4.5 muestra otro ejemplo en una situación de tráfico cotidiano. Al igual que en la imagen anterior, la primera imagen corresponde a la original, la segunda a la predicción del modelo YOLOv8n y la tercera a la predicción del modelo YOLOv8m. Nuevamente, se observa una mejora en la detección realizada por YOLOv8m en comparación con YOLOv8n. El modelo YOLOv8m no solo detecta un mayor número de objetos, sino que también asigna un grado de confianza superior a cada objeto identificado.

En la Figura 4.6 se muestra la detección de una señal de tráfico en un carretera secundaria.

La Figura 4.7 muestra la correcta actuación de las redes de detección ante una situación nocturna. Al igual que en el anterior mosaico, en la parte superior nos encontramos ante la imagen original, seguidamente tenemos la detección realizada por YOLOv8n y por último la detección por YOLOv8m. Ambas se comportan correctamente en esta situación de poca luminosidad y diferencian una gran cantidad de vehículos además de la señal de prohibido.

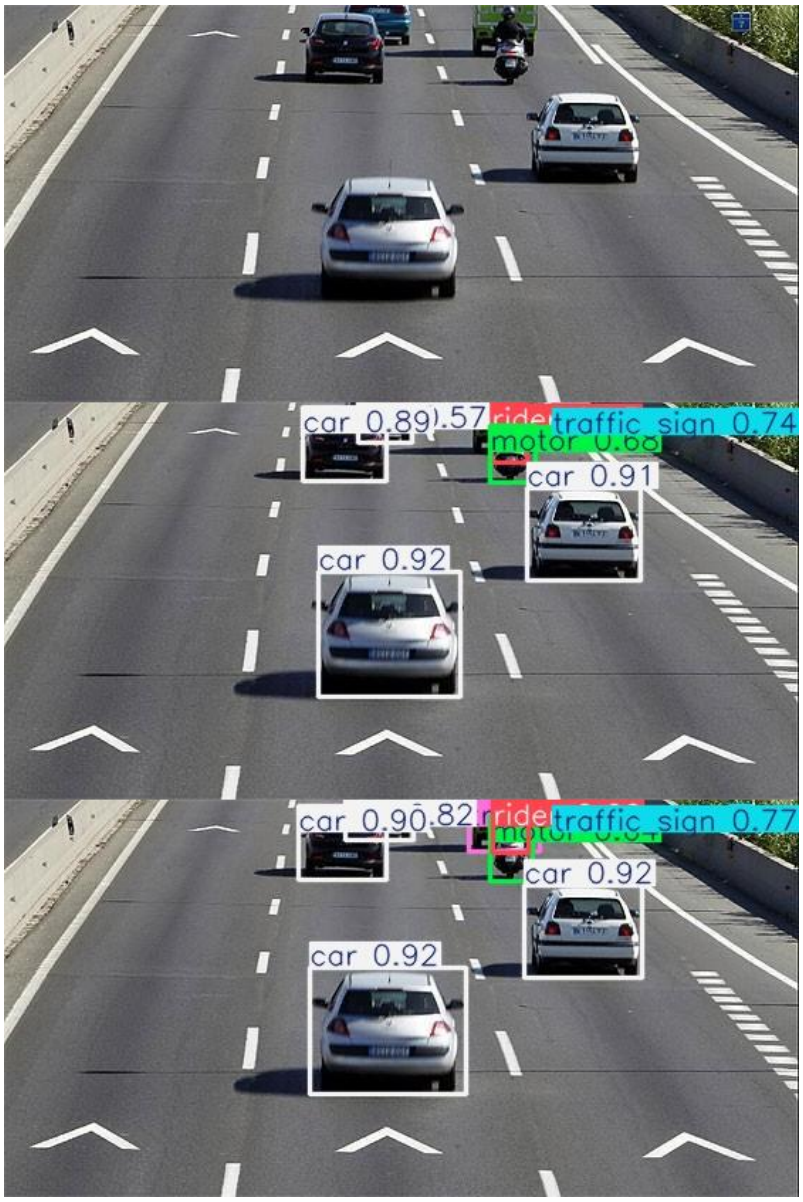

Figura 4.4 Predición de los modelos de detección de una imagen de tráfico.

4.3 Red clasificación

En cuanto a la red de clasificación, el conjunto de datos utilizado para entrenar es el GTSDB. Este conjunto contiene un total de 43 clases de distintas señales, en la Figura 2.5 podemos observar cuales son esas clases. El conjunto de datos descargado se compone de un total de 51.839 imágenes de señales de tráfico. Estas imágenes originalmente viene dividas en dos carpetas, una para las imágenes de entrenamiento y otra para las imágenes de test. La notación de las imágenes se encuentra en un `csv` adjunto con el conjunto de datos.

4.3.1 Tratamiento de datos

Al descargar el conjunto de datos GTSDB, se obtienen dos carpetas principales: una para las imágenes de entrenamiento y otra para las imágenes de test. Las etiquetas de las imágenes se

Figura 4.5 Predición de los modelos de detección de YOLOv8 ante una situación de tráfico común.

proporcionan en un archivo `csv` que contiene la relación entre el nombre de la imagen y la clase a la que pertenece.

Para facilitar el proceso de entrenamiento, el conjunto de datos de entrenamiento fue dividido en cuatro particiones diferentes, con el objetivo de realizar una validación cruzada en cuatro fases. En cada partición, el 75 % de las imágenes se utilizaron para entrenamiento y el 25 % restante para validación. Este método permite garantizar que el modelo se entrene y valide de manera equilibrada, evitando un posible sobreajuste a una única división del *dataset*.

La estrategia seguida para dividir el *dataset* en las cuatro particiones fue la siguiente:

Figura 4.6 Predición del modelo de detección YOLOv8m.

1. El 25 % inicial de las imágenes se utilizaron como conjunto de validación y el 75 % restante como conjunto de entrenamiento.
2. Las imágenes comprendidas entre el 25 % y el 50 % del *dataset* se usaron como conjunto de validación, mientras que el resto fueron utilizadas para el entrenamiento.
3. Entre el 50 % y el 75 % de las imágenes se emplearon como validación.
4. Finalmente, las imágenes del 75 % al 100 % del *dataset* se destinaron a la validación.

4.3.2 Entrenamiento y validación de las redes de clasificación creadas

El proceso de entrenamiento de las redes de clasificación se desarrolló utilizando una arquitectura modular, donde los parámetros clave del entrenamiento, como el optimizador, la tasa de aprendizaje, las épocas y el tamaño de lote, se leyeron de un archivo con formato `JSON` externo para facilitar el *benchmarking*. A continuación, se describe el flujo del entrenamiento y los principales aspectos de la implementación.

Estructura del archivo de configuración

Para gestionar de manera eficiente los distintos hiperparámetros a evaluar, se creó un archivo con formato `JSON` que contenía las posibles combinaciones de optimizadores, tasas de aprendizaje, tamaños de batch y particiones del *dataset*. El siguiente archivo muestra un ejemplo de la estructura utilizada:

Este archivo permitió probar múltiples combinaciones de hiperparámetros, facilitando el *benchmarking* y la optimización del modelo.

Para la selección de las tasas de aprendizaje y los tamaños de lote, se realizaron pruebas en el modelo base de red con diversos valores, con el objetivo de identificar un rango adecuado para obtener buenos resultados. En primer lugar, se estableció el tamaño de lote y se llevaron a cabo cuatro pruebas variando los valores de la tasa de aprendizaje.

En la Figura 4.9 se muestra el *accuracy* y las pérdidas obtenidas para las tasas de aprendizaje 0,1, 0,01, 0,001 y 0,5. Como se puede apreciar las tasas de 0,1 y 0,01 son las que mejores resultados obtienen. Con 0,5 de tasa de aprendizaje el entrenamiento se vuelve inestable y no converge. Por otro lado, con 0,001 la red tarda mucho en converger y no llega a obtener tan buenos resultados.

En cuanto al tamaño de lote, se realizaron pruebas con los valores 2, 256 y 2.048. En la Figura 4.10 se muestra el *accuracy* y las pérdidas obtenidas para cada uno de los tamaños de lote. El valor de 2.048 nos da un peor desempeño, mientras que los valores de 2 y 256 obtienen resultados similares.

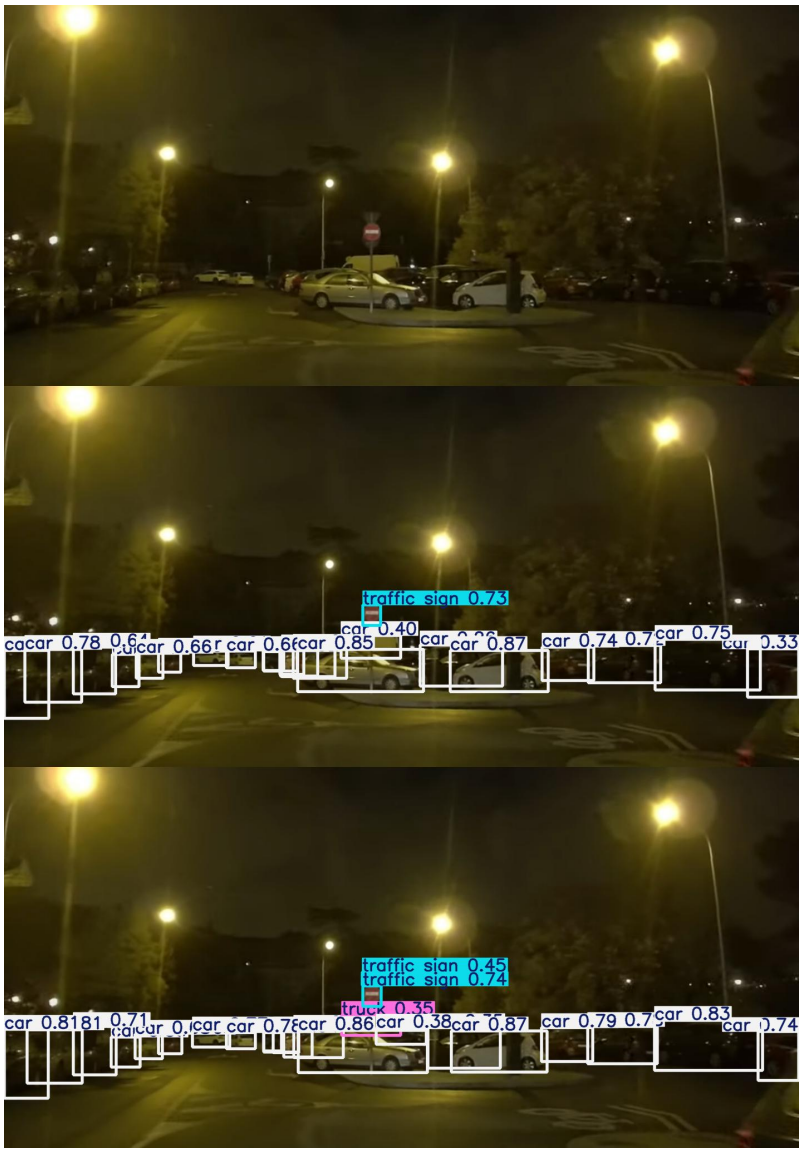

Figura 4.7 Predición de los modelos de detección de YOLOv8 ante una situación de tráfico común nocturna.

```
{
    "optimizers": ["sgd", "adam"],
    "epochs": [5, 10, 15, 20, 25, 30, 35, 40],
    "learning_rate": [0.05, 0.02, 0.015, 0.010, 0.005],
    "train_path": ["split1/Train", "split2/Train",
       "split3/Train", "split4/Train"],
    "batch_size": [64, 128, 256, 1024]
}
```

Figura 4.8 Archivo en formato `JSON` de configuración de hiperparámetros para el entrenamiento.

Por lo tanto decidimos escoger valores intermedios para el tamaño de lote. Además, un tamaño de lote muy pequeño hace que el entrenamiento se vuelva muy lento, mientras que a mayor tamaño de

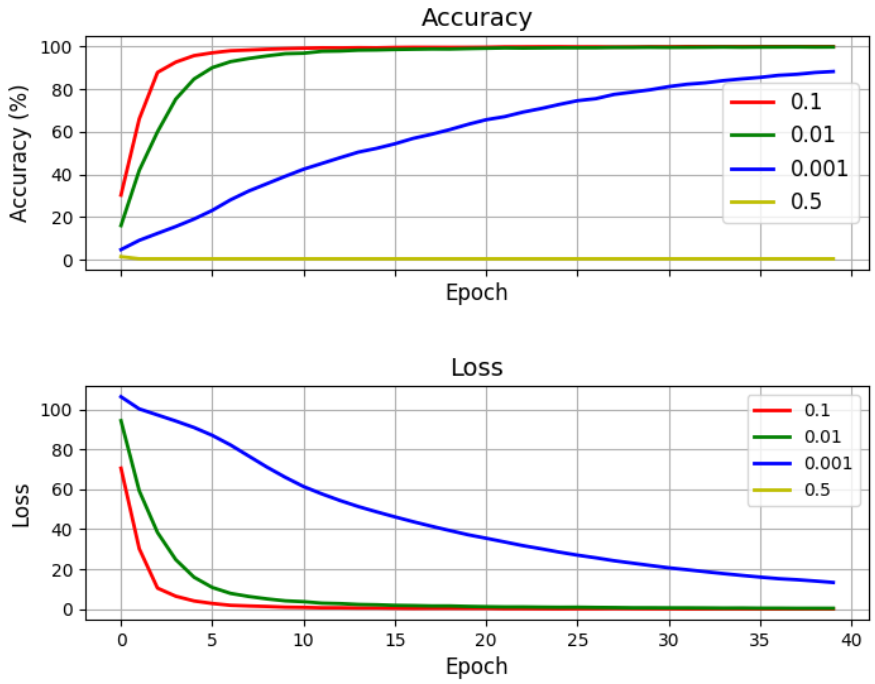

Figura 4.9 Evaluación de la tasa de aprendizaje en el modelo base de red.

lote, mayor será la memoria necesaria.

Implementación del entrenamiento

El código desarrollado para entrenar la red de clasificación tenía como objetivo recorrer todas las combinaciones de los hiperparámetros mencionados y registrar los resultados para su posterior análisis. Este script de python recibía por parámetros dos cadenas de texto con las rutas a los archivos con formato `JSON` de configuración. Por una parte, tenía el archivo con la descripción del estructura de red a usar 3.5, por otra parte recibía el archivo de configuración descrito en la Figura 4.8. El proceso de entrenamiento consistía en los siguientes pasos:

- **Lectura de los archivos de configuración**: Se leía ambos archivos de configuración.
- **Transformación de los datos**: Se aplicaron transformaciones a las imágenes del *dataset* para ajustarlas al tamaño de entrada de la red y normalizarlas adecuadamente.
- **Ciclo de entrenamiento**: Para cada combinación de parámetros:
  - Se cargaban los datos de entrenamiento y validación desde las carpetas correspondientes, usando el tamaño de lote correspondiente.
  - Se inicializaba el modelo utilizando las capas especificadas.
  - Se seleccionaba el optimizador correspondiente (`Adam` o `SGD`) y se ajustaba la tasa de aprendizaje.
  - Se entrenaban los datos durante el número de épocas especificado (el mayor de la lista de épocas en la Figura 4.8)
  - Por cada época descrita en el archivo de configuración, se evaluaba el rendimiento del modelo sobre el conjunto de validación.

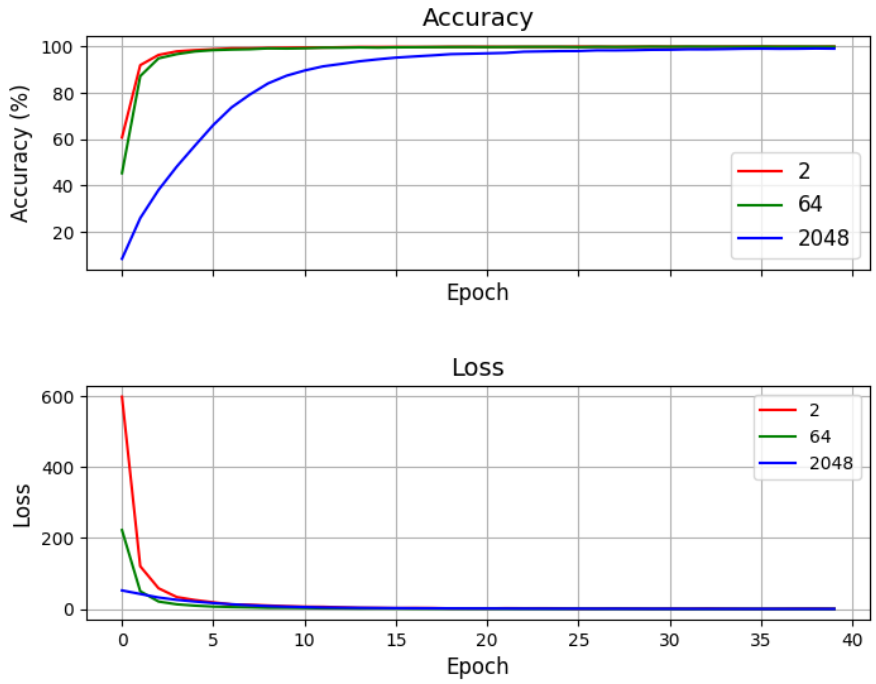

Figura 4.10 Evaluación del tamaño de lote en el modelo base de red.

- **Evaluación del rendimiento**: Por cada época descrita en el archivo de configuración se hacía una evaluación del rendimiento con el conjunto de datos de validación. En cada evalución del rendimiento, se calculaban las métricas incluyendo precisión, *recall* y *F1-score* (en sus versiones micro, macro y ponderadas), utilizando las predicciones obtenidas sobre el conjunto de validación.
- **Almacenamiento de resultados**: Se guardaron los resultados en archivos `csv` para facilitar la evaluación posterior. Se guardaron las métricas calculadas mencionadas y además también se guardaban los hiperparámetros utilizados en el entrenamiento. Por último, también se almacenaba el tiempo de entrenamiento.
- **Optimización del tiempo de entrenamiento**: Para reducir el tiempo de entrenamiento, se utilizó la GPU. También se implementó una parada anticipada,ssi una combinación de hiperparámetros no convergía pasadas varias épocas, se pasaba a la siguiente combinación de hiperparámetros. Se establecio como umbral más del 10 % de *accuracy* en la época 10.

Resultados obtenidos

Tras el entrenamiento se obtuvieron de 15 archivos `csv` con la información de cada entrenamiento realizado. En estos teniamos un máximo de 1280 entradas, ya que se realizaron 4 particiones del *dataset* y se probaron 320 combinaciones de hiperparámetros. Para la validación cruzada se promediaron las métricas obtenidas en las 4 particiones, quedando ahora un máximo de 320 entradas.

Para cada modelo se obtuvó la mejor combinación de hiperparámetros usando el *accuracy* como métrica de evaluación. En la Tabla 4.2 podemos ver por cada modelo de red (descrito en el capítulo anterior en la subsección 3.4.2) la combinación de hiperparámetros que obtiene el mejor *accuracy*. Además también vemos el tiempo de entrenamiento del modelo con dichos hiperparámetros en segundos.

Tabla 4.2 Combinación de hiperparámetros que obtiene el mejor *accuracy* para cada modelo de red.

Modelo	*Accuracy*	Época	Tiempo (s)	Tamaño de Lote	Tasa	Optimizador
1	98,865	15	222,23	64	0,015	sgd
2	97,331	5	74,36	64	0,01	sgd
3	96,478	35	496,45	1024	0,005	sgd
4	99,159	20	727,74	64	0,01	sgd
5	99,942	15	186,33	1024	0,005	sgd
6	98,771	5	52,14	64	0,02	sgd
7	97,239	25	267,17	64	0,005	sgd
8	96,015	25	373,02	64	0,005	sgd
9	98,960	25	871,42	64	0,01	sgd
10	97,784	20	222,20	64	0,02	sgd
11	98,491	10	113,67	64	0,01	sgd
12	97,669	20	253,56	1024	0,05	adam
13	95,930	25	274,34	1024	0,005	sgd
14	99,016	15	495,22	64	0,01	sgd
15	97,430	20	229,82	64	0,015	sgd

De las mejores combinaciones de cada modelo nos quedaremos con los 5 mejores modelos para compararlos con los modelos de YOLOv8 de clasificación y así podremos evaluar el rendimiento de las redes creadas respecto al estado del arte. Como podemos observar en la Tabla 4.2 los modelos 5, 4, 14, 9 y 1 son los que mejores resultados han obtenido, así que serán los seleccionados con sus respectivas configuraciones de hiperparámetros para la comparación.

El modelo 1 se corresponde con el descrito en la Figura 3.5. El modelo 4, 9 y 15 se corresponden con la Figura 3.7, la diferencia entre ellos es el tipo de capa de activación, en el modelo 4 se utilizó la ReLU, en el modelo 9 la LeakyReLU y en el modelo 15 la GeLU. Por último, el modelo 5 se corresponde con la Figura 4.11. Todos los modelos fueron descritos ya previamente en la sección 3.4.2

Una vez seleccionados los mejores modelos y sus configuraciones de hiperparámetros, procederemos a reentrenarlos, utilizando en esta ocasión el conjunto completo de datos de entrenamiento y validación. Esto implica que se emplearán todos los datos disponibles para el entrenamiento, excluyendo únicamente el conjunto de datos de prueba, que se mantendrá sin exponer a los modelos.

Finalmente, los modelos entrenados se evaluarán con el conjunto de datos de prueba. Los resultados obtenidos serán comparados con los modelos de clasificación de YOLOv8. En esta etapa de evaluación se calcularán nuevamente las métricas de precisión, *recall* y *F1-score*, en sus versiones micro, macro y ponderadas.

En la Tabla 4.3 se muestran las métricas obtenidas al evaluar los modelos seleccionados con el conjunto de datos de prueba. En la tabla se ve como para 4 de los 5 modelos creados el *accuracy* es superior a los 96 puntos. Sin necesidad de comparar con los resultados de las redes de clasificación de YOLOv8 se observa un gran desempeño de las redes en la clasificación de las señales de tráfico.

Tabla 4.3 Métricas de redes creadas con el conjunto de datos de prueba.

Modelo	*Accuracy*	*Precision*	*Recall*	*F1-score*	Tiempo (s)
5	86,286	0,870	0,862	0,862	539,35
4	97,672	0,9777	0,976	0,976	1136,57
14	97,553	0,976	0,975	0,975	1330,73
9	97,941	0,979	0,979	0,979	1869,52
1	96,967	0,971	0,969	0,969	265,78

```
{
    "layers" :[
        {"type": "conv", "in_channels": 3, "out_channels": 16, "
            kernel_size": 3, "stride": 1, "padding": "same"},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},
        {"type": "conv", "in_channels": 16, "out_channels": 32, "
            kernel_size": 3, "stride": 1, "padding": "same"},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},
        {"type": "conv", "in_channels": 32, "out_channels": 64, "
            kernel_size": 3, "stride": 1, "padding": "same"},
        {"type": "activation", "function": "ReLU"},
        {"type": "pool", "kernel_size": 2},
        {"type": "flatten"},
        {"type": "fc","in_features": 3136, "out_features": 1024},
        {"type": "fc","in_features": 1024, "out_features": 512},
        {"type": "fc","in_features": 512, "out_features": 43}
    ]
}
```

Figura 4.11 Estructura de la red de clasificación 5..

En la Figura 4.12 se muestra la matriz de confusión del modelo 9. En la matriz de confusión se puede observar como el modelo ha clasificado correctamente la mayoría de las señales de tráfico, obteniendo un *accuracy* del 97,941 %.

En la Figura 4.13 se muestra un mosaico de señales de tráfico del conjunto de datos de pruebas. Estas señales han sido clasificadas por la red creada 1. En la esquina superior izquierda de cada imagen podemos ver a que clase pertenece dicha señal.

4.3.3 Entrenamiento y validación de las redes de clasificación de YOLOv8

Los datos de entrenamiento y validación en este caso no necesitan una estructura distina a la usada en nuestra redes creadas manualmente. Es decir, las imágenes de entrenamiento estarán en la carpeta *train* y habrá otra carpeta llamada *val* con las imágenes de validación. Ambas estarán compuestas por un directorio por cada clase y las imágenes estarán en su interior.

Para el entrenamiento tendremos que ejecutar el siguiente comando en la terminal:

```
yolo task=classify mode=train model=$red_yolov8$ data=$path$
```

Donde *red_yolov8* es el nombre del modelo de red que se va a utilizar 3.1 y *path* es la ruta donde se encuentran las carpetas train y val con las imágenes de entrenamiento y validación respectivamente.

Al igual que en el entrenamiento de la red de detección tenemos en este caso las mismas posibilidades de configuración para el entrenamiento de clasificación, se muestran en la Tabla 4.1.

Para el entrenamiento, lo único que se fijó fue el número de épocas a 50, el resto de hiperparámetros se dejaron por defecto. YOLOv8 encuentra el optimizador, la tasa de aprendizaje y el tamaño de lote más adecuados para el entrenamiento automáticamente. Se entrenaron los 5 modelos de YOLOv8 directamente con todos los datos de validación y entrenamiento como un único conjunto de entrenamiento y se evaluaron las mismas métricas que en las redes creadas con el conjunto de datos de prueba.

En la Tabla 4.4 se muestran las métricas obtenidas al evaluar los modelos de YOLOv8 con el conjunto de datos de prueba. También mostramos el tiempo en segundos que se tardó en entrenar dichos modelos. En la tabla se aprecian como todos los modelos han obtenido una obtenido unas

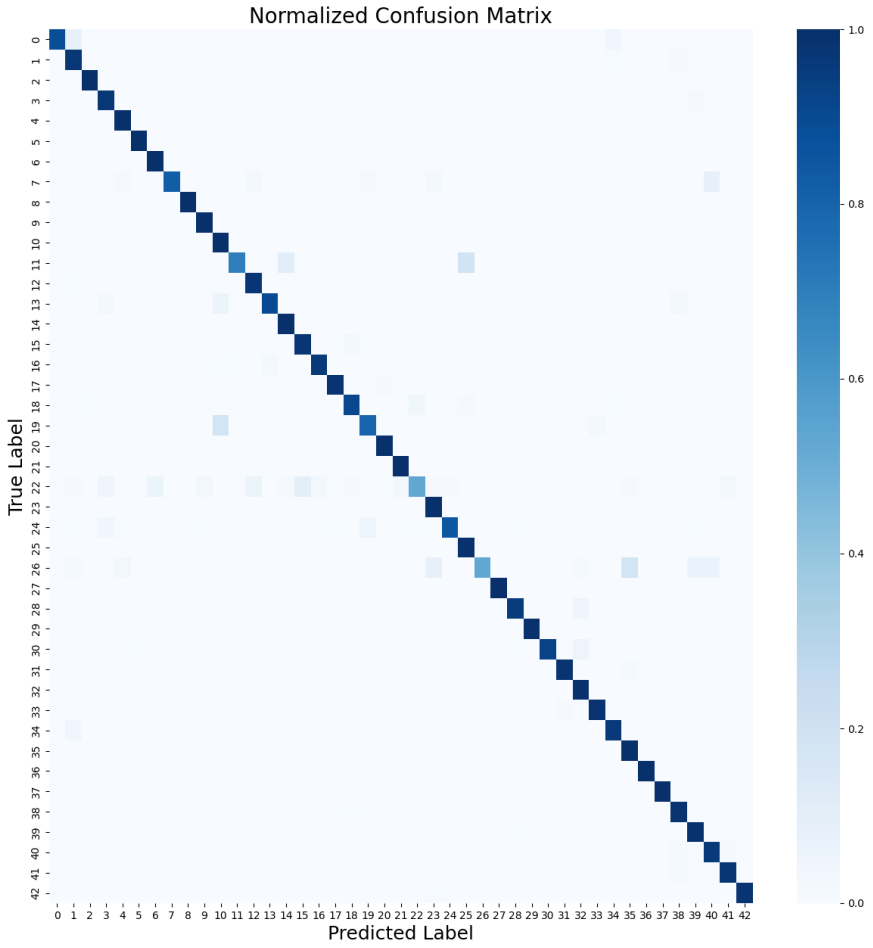

Figura 4.12 Matriz de confusión del modelo 9.

Tabla 4.4 Métricas de las redes de YOLOv8 para clasificación evaluadas con el conjunto de prueba.

Modelo	*Accuracy*	*Precision*	*Recall*	*F1-score*	Tiempo (s)
yolov8n-cls	97,957	0,979	0,979	0,978	3.006,34
yolov8s-cls	97,893	0,979	0,978	0,977	3.350,23
yolov8m-cls	97,806	0,978	0,978	0,976	3.640,73
yolov8l-cls	97,608	0,977	0,976	0,975	4.052,52
yolov8x-cls	97,743	0,979	0,977	0,977	4.265,78

métricas muy parejas. De hecho el modelo más ligero el YOLOv8n-cls es el que mejor *accuracy* ha obtenido, además sumado a que ha sido el modelo que menos tiempo ha tardado en entrenar y es el modelo más rápido en predecir será el mejor modelo a usar para este caso de uso.

La Figura 4.14 muestra la matriz de confusión del modelo YOLOv8n-cls.

YOLOv8 tras el entrenamiento genera un directorio que contiene los pesos de la entrenada en la mejor época y en la última(evaluada con el conjunto de validación). También, se genera un

Figura 4.13 Mosaico de 20 señales correctamente clasificadas por el modelo creado 1.

archivo con extensión *yaml* que contiene la configuración de la red. Adicionalmente se generan gráficos con mosaicos de muestras de lotes de imagénes tanto de entrenamiento como de validación. Finalmente, el sistema devuelve la matriz de confusión, la matriz de confusión normalizada y un archivo *result.csv* en el que se almacenan varias métricas, tales como la pérdida, la precisión (*accuracy*) y otras métricas relevantes. A partir de este archivo, se genera una gráfica en formato *.png* que representa visualmente las métricas registradas.

En la Figura 4.15 se muestra un mosaico de muestras de lotes de imágenes de entrenamiento. En este mosaico se pueden observar las imágenes que se han utilizado para entrenar el modelo. Estas imágenes distan de las originales del conjunto de entrenamiento ya que como se explico en el capítulo Estado del Arte en la sección 2.4.8, la arquitectura YOLOv8 preprocesa las imágenes de entrenamiento aumentandolas y tratandola con diferentes filtros. Además también podemos apreciar oclusiones en las imágenes, buscando con esto mejoras en la generalización del modelo y evitando el sobreajuste.

En la Figura 4.16 podemos ver gráficas de como evoluciona el *accuracy*, las perdidás del conjunto de datos de validación y las pérdidas del conjunto de datos de entrenamiento. En la métrica *accuracy_top5* se contea como positivo si la etiqueta coincide con una de las 5 predicciones con más peso.

4.3.4 Comparativa de las redes de clasificación de YOLOv8 y las creadas en este TFG

En esta sección se presenta una comparativa entre las redes de clasificación de YOLOv8, específicamente las cinco redes descritas previamente en este capítulo, y las redes diseñadas por el autor. El objetivo principal es analizar el rendimiento de ambos enfoques en términos de *accuracy* y velocidad de inferencia. Para ello, se examinan los resultados obtenidos sobre el conjunto de datos de prueba y se medirán los tiempos en predecir todas las imágenes del conjunto.

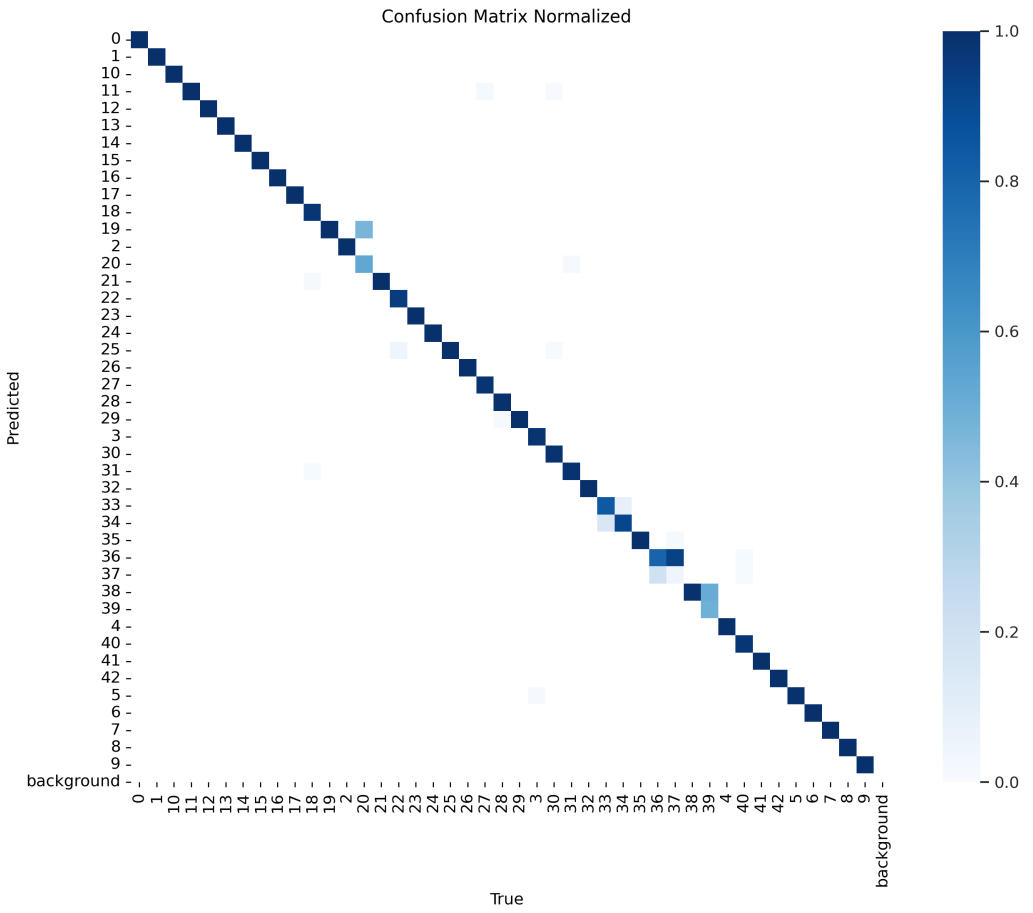

Figura 4.14 Matriz de confusión del modelo YOLOv8n-cls.

Por un lado, YOLOv8 constituye el estado del arte en arquitecturas de clasificación, siendo modelos preentrenados inicialmente con conjuntos de datos masivos como ImageNet, pero que, en este trabajo, también han sido entrenados específicamente con el *dataset* GTSDB para adaptarlos a la tarea de reconocimiento de señales de tráfico. Estas redes destacan por su alta precisión y capacidad de generalización.

Por otro lado, las redes desarrolladas en este TFG están diseñadas con un enfoque específico hacia el reconocimiento de señales de tráfico, utilizando exclusivamente el *dataset* GTSDB. Estas redes persiguen no solo un desempeño competitivo en términos de precisión, sino también una mayor simplicidad y menor coste computacional.

Para garantizar una comparativa justa entre ambos enfoques, todas las pruebas se han realizado bajo las mismas condiciones en el mismo equipo. Se ha asegurado que ni la CPU ni la GPU tuvieran otras cargas de trabajo activas durante las mediciones de tiempo. Para ello, se detuvieron todos los procesos innecesarios, y las pruebas se realizaron desde una terminal ligera, evitando el uso de entornos de desarrollo integrados (IDEs) que puedan consumir recursos significativos. Cada modelo fue ejecutado individualmente, registrando el tiempo de inicio y finalización de la inferencia. Dado que el conjunto de datos de prueba consta de 12.630 imágenes, se utilizó el tiempo promedio como una medida estadística representativa y fiable para la evaluación de la velocidad de predicción.

La máquina utilizada para realizar las pruebas es una `Asus Tuf Gaming A15`, modelo `FA507NV_-FA507NV`. El portátil incluye un procesador `AMD Ryzen 7 7735HS` con 16 GB de RAM con tecnología DDR5 que funciona a una frecuencia 4800 MHz. En cuanto a la GPU, el portátil cuenta con

Figura 4.15 Mosaico de muestra de lotes de imágenes de entrenamiento de YOLOv8.

una tarjeta gráfica `NVIDIA GeForce RTX 4060`, la cual dispone de 8 GB de VRAM con tecnología GDDR6.

En la Tabla 4.5 podremos observar una tabla comparativa de los distintos modelos ordenados de mayor a menor *accuracy*. En dicha tabla se observan las distintas métricas como el *accuracy*, la precisión, el *recall* y la *F1-score*, además del tiempo que será el promedio de tiempo que tarda el modelo en dar una predición en ms.

Tabla 4.5 Comparativa redes YOLOv8 con las redes creadas.

Modelo	***Accuracy***	**Tiempo (ms)**	**Precisión**	***Recall***	***F1***
yolov8n-cls	97,957	2,732	0,979	0,979	0,978
Creada 9	97,941	0,408	0,979	0,979	0,979
yolov8s-cls	97,893	2,785	0,979	0,978	0,977
yolov8m-cls	97,806	3,509	0,978	0,978	0,976
yolov8x-cls	97,743	5,656	0,979	0,977	0,977
Creada 4	97,672	0,415	0,9777	0,976	0,976
yolov8l-cls	97,608	4,433	0,977	0,976	0,975
Creada 14	97,553	0,399	0,976	0,975	0,975
Creada 1	96,967	0,305	0,971	0,969	0,969
Creada 5	86,286	0,299	0,870	0,862	0,862

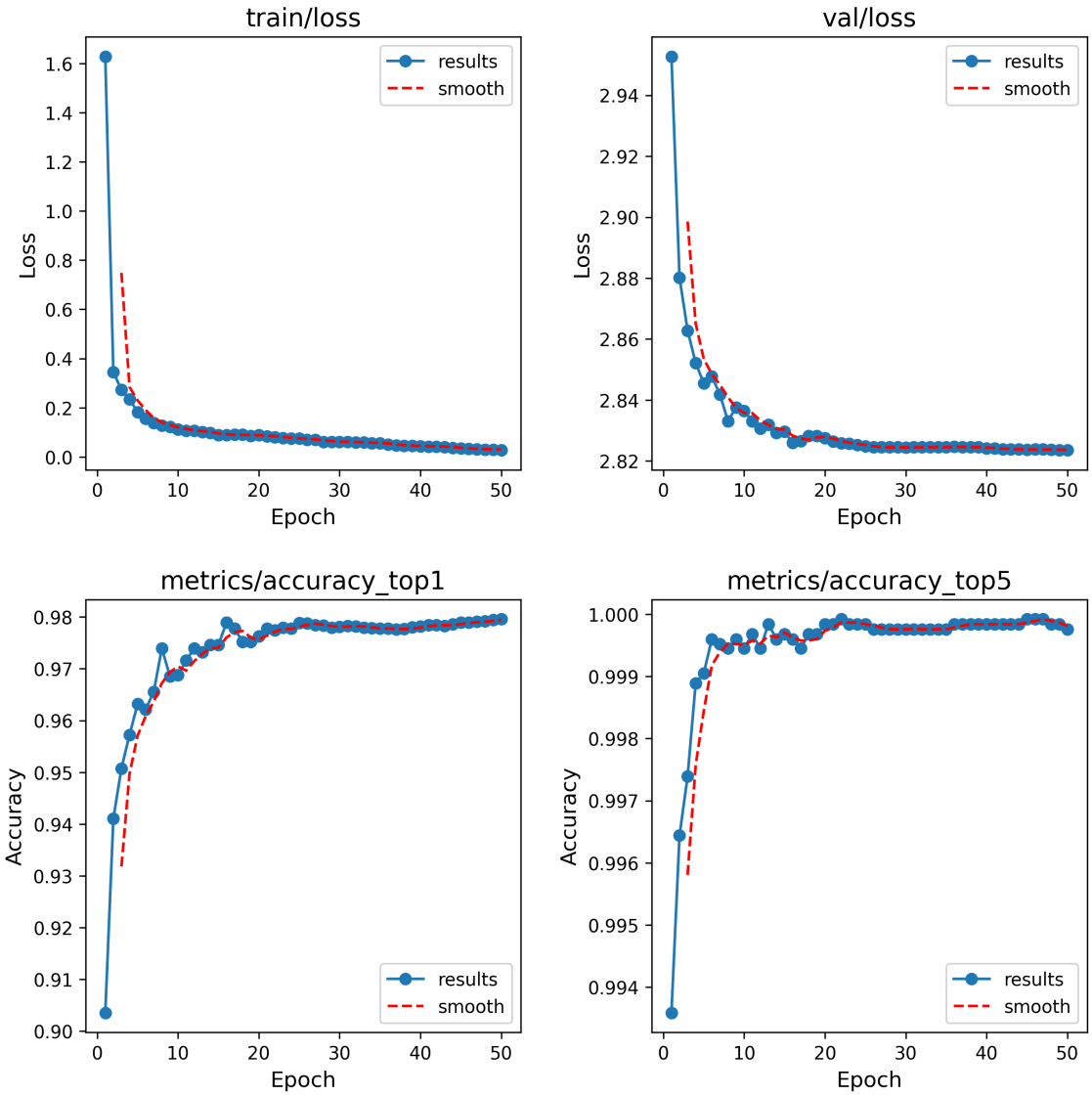

Figura 4.16 Resultados entrenamiento del modelo YOLOv8n.

De la tabla presentada se pueden extraer varias observaciones clave sobre el desempeño de las redes comparadas. En términos de *accuracy*, las primeras ocho redes, que incluyen las cinco de YOLOv8 y tres de las diseñadas en este TFG, muestran diferencias mínimas, inferiores a 0,4 puntos porcentuales. Los valores oscilan entre el 97,957 % alcanzado por YOLOv8n-cls, que es el modelo con mayor precisión, y el 97,553 % de la red Creada 14. Estos resultados evidencian que las redes diseñadas en este trabajo son capaces de competir en precisión con modelos de última generación como YOLOv8.

En cuanto a la velocidad de inferencia, las redes desarrolladas en este TFG destacan de manera notable frente a las de YOLOv8. Por ejemplo, la red más rápida entre las diseñadas, la Creada 5, requiere solo 0,299 ms por imagen, mientras que la red más rápida de YOLOv8, YOLOv8n-cls, necesita 2,732 ms, lo que representa una mejora de más de un orden de magnitud en términos de rapidez. Las redes Creada 4, Creada 9 y Creada 14, que están entre las más precisas, también mantienen tiempos de inferencia cercanos a 0,4 ms, logrando un excelente balance entre precisión y rapidez.

Estos resultados resaltan un compromiso interesante entre precisión y velocidad. Aunque YOLOv8

proporciona modelos altamente precisos con arquitecturas optimizadas para diversas tareas, las redes diseñadas en este TFG demuestran que es posible reducir significativamente los tiempos de inferencia sin comprometer gravemente la precisión. Esto las convierte en una alternativa atractiva para aplicaciones donde la velocidad es crítica, como en dispositivos embebidos o sistemas con recursos computacionales limitados.

El análisis de los tiempos de ejecución resulta coherente al observar la descripción estructural de los modelos en el capítulo 3 en la sección Red de Clasificación 3.4. Las redes con más capas o con capas más costosas computacionalmente, como las de YOLOv8, tienden a requerir mayor tiempo de inferencia. Por ejemplo, el modelo más complejo, YOLOv8x-cls, es el más costoso en términos de tiempo, mientras que el más ligero, YOLOv8n-cls, resulta el más eficiente. En el caso de las redes diseñadas en este TFG, se observa que las redes Creada 4, Creada 9 y Creada 14, que comparten la misma estructura pero se diferencian únicamente en la función de activación utilizada, presentan tiempos de ejecución prácticamente idénticos

En resumen, la comparativa pone de manifiesto que las redes desarrolladas en este TFG no solo alcanzan un rendimiento competitivo frente al estado del arte en precisión, sino que también destacan por su eficiencia computacional. Esta ventaja puede ser determinante para su implementación en entornos donde el tiempo de respuesta sea un factor clave.

4.3.5 Análisis de los errores de las redes

En este apartado se estudian las imágenes en las que las redes han fallado, identificando patrones específicos en los errores cometidos. Para ello, se han analizado tanto las 5 redes de clasificación de YOLOv8 como las 4 mejores redes creadas por el autor (se excluye la red creada 5 debido a su evidente mal ajuste, que podría introducir ruido en la interpretación de los fallos de las otras redes).

Para realizar este análisis, se guardaron los resultados de predicción del conjunto de prueba en un archivo `csv`. Cada fila del archivo contiene el nombre de la imagen, la etiqueta real y la etiqueta predicha por el modelo. Este formato permite identificar fácilmente las imágenes en las que las predicciones difieren de las etiquetas reales.

Tras procesar los datos, se obtuvo que 11.727 imágenes (el 92,85 % del conjunto de prueba) fueron correctamente clasificadas por todas las redes en todas las iteraciones.

Sin embargo, 2,575 errores fueron registrados en total, con 241 errores acumulados en solo 34 imágenes (0,26 % del conjunto de prueba). Estas imágenes problemáticas concentran un porcentaje desproporcionadamente alto de los fallos, lo que indica que presentan características que dificultan su clasificación para los modelos.

En la Figura 4.17 se observa un mosaico con dichas imágenes, en la esquina superior izquierda de cada imagen que compone el mosaico se indica la clase real a la que pertenece. Se puede observar cómo algunas de estas 34 señales son irreconocibles al ojo humano debido a las condiciones de las propias imágenes, como baja resolución, presencia de ruido o iluminación deficiente. Otras imágenes con múltiples errores corresponden a señales direccionales, las cuales han sido confundidas con sus contrarias.

Para examinar más a fondo estos errores y determinar si ciertas clases tienden a confundirse entre sí, se generó una matriz de confusión exclusivamente a partir de los errores registrados, la cual se muestra en la Figura 4.18. El análisis de esta matriz reveló un patrón recurrente: las señales direccionales, como las de giro a la izquierda o a la derecha, se confunden frecuentemente entre sí, vease que la clase 20 se ha confundido 222 con la clase 19, que corresponden con un 8,6 % del total de los errores. También se observan como la clase 33 ha sido confundida con la clase 34 en 193 ocasiones. Y el caso con más relevaciancia es el de la clase 36 con la 37, la clase 36 ha sido confundido 188 veces con la 37 y la 36 ha sido confundida 282 veces con la 37. En la Figura 2.5 se mostró un mosaico con las diferentes clases que contiene el *dataset* y una muestra de cada una ordenadas de esquina superior izquierda a esquina inferior derecha. Este tipo de error es comprensible, ya que estas señales comparten muchas características visuales y su principal

diferencia radica en la orientación de la flecha. Este hallazgo sugiere que los modelos podrían beneficiarse de un mayor número de ejemplos de estas clases durante el entrenamiento o del uso de técnicas de preprocesamiento específicas.

Además, actualmente las imágenes se introducen en los modelos sin filtrado previo y con los tres canales de color, lo que permite que los modelos utilicen tanto el contorno como el color para realizar las predicciones. Si bien este enfoque podría ser útil para distinguir entre señales de diferentes categorías, como señales de prohibición (rojas) y señales de obligación (azules), también hace que las redes no se centren exclusivamente en los contornos de las señales. Un enfoque alternativo podría ser preprocesar las imágenes resaltando únicamente los contornos, lo que potencialmente ayudaría a mejorar la clasificación de las señales direccionales. Sin embargo, este cambio podría afectar negativamente el rendimiento en otros tipos de señales, donde el color es un atributo diferenciador clave.

El análisis de los errores cometidos por las redes revela que la gran mayoría de fallos se concentran en dos tipos de imágenes: aquellas irreconocibles incluso para el ojo humano, debido a factores como baja resolución, ruido o malas condiciones de iluminación, y las señales direccionales, que tienden a confundirse frecuentemente con sus contrarias debido a la similitud visual y la orientación opuesta de sus flechas.

Figura 4.17 Mosaico de imágenes que concentran el mayor número de errores de clasificación.

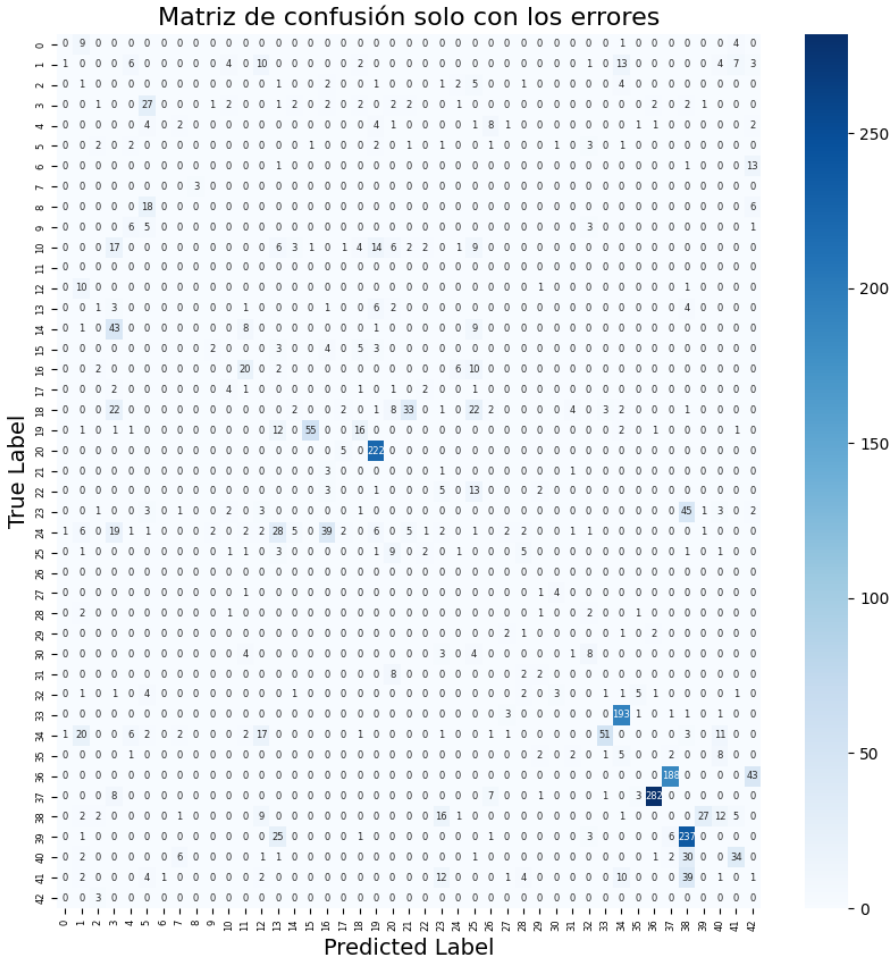

Figura 4.18 Matriz de confusión solo con los errores.

4.4 Automatización del enlace entre redes de detección y clasificación

En el sistema desarrollado, se ha implementado un flujo completamente automatizado que enlaza la red de detección de señales de tráfico con la red de clasificación. Este enfoque permite procesar tanto secuencias de video capturadas desde un móvil como mediante una cámara conectada al portátil, ofreciendo la capacidad de identificar y clasificar señales de tráfico en tiempo real.

El flujo de trabajo se inicia con la red de detección entrenada de YOLOv8, la cual procesa cada fotograma del video y localiza las señales presentes, devolviendo las coordenadas de las regiones donde se encuentran. A partir de estas coordenadas, se recortan las regiones de interés correspondientes a las señales detectadas. Estas regiones recortadas se pasan de manera automática como entrada a la red de clasificación, que determina el tipo exacto de señal.

Este proceso automatizado se detalla a continuación:

- **Captura del video**: El sistema permite usar tanto un archivo de video como un stream de una cámara conectada.

- **Detección de señales**: La red YOLOv8 se encarga de localizar las señales de tráfico en cada fotograma, devolviendo las coordenadas de las cajas delimitadoras.
- **Recorte de la región de interés**: Usando las coordenadas de las cajas delimitadoras, se recortan las regiones correspondientes a cada señal detectada.
- **Preprocesamiento**: Las imágenes recortadas se ajustan al tamaño y formato requerido por la red de clasificación.
- **Clasificación de la señal**: La región recortada se pasa a la red de clasificación, que identifica el tipo exacto de señal de tráfico.
- **Anotación del fotograma**: Se dibujan las cajas delimitadoras y se añade el tipo de señal clasificada sobre cada fotograma.
- **Visualización**: Los fotogramas anotados se muestran en tiempo real a través de una ventana.

Este enfoque automatizado garantiza una comunicación fluida entre ambas redes, eliminando la necesidad de intervención manual y permitiendo que el sistema funcione en tiempo real. Además, ofrece flexibilidad para su uso en diferentes dispositivos de captura, como móviles o cámaras portátiles.

5 Conclusiones y desarrollos futuros

Este Trabajo de Fin de Grado ha abordado la compleja tarea de reconocimiento de señales de tráfico mediante un enfoque modular compuesto por dos etapas principales: detección y clasificación. Para cada una de estas etapas, se han desarrollado e implementado soluciones basadas en modelos de última generación y redes neuronales diseñadas específicamente para este trabajo.

En la etapa de detección, se ha empleado la arquitectura YOLOv8, destacada por su alta precisión y velocidad, lo que la convierte en una opción ideal para aplicaciones en tiempo real. Los modelos entrenados han mostrado un buen desempeño sobre el conjunto de datos BDD100K, logrando identificar y delimitar correctamente múltiples clases de objetos presentes en las imágenes. El entrenamiento y validación de estos modelos permitió no solo asegurar una alta precisión en la detección, sino también evaluar la capacidad del sistema para operar en entornos reales y variados. Los resultados obtenidos en la detección son más que satisfactorios como se observa en la Figura 4.14.

La etapa de clasificación ha supuesto un reto significativo debido a la gran diversidad de señales de tráfico existentes y las condiciones cambiantes de las imágenes. Para esta tarea, se han desarrollado redes neuronales personalizadas, las cuales han sido entrenadas y optimizadas utilizando el conjunto de datos GTSDB. A través de un exhaustivo proceso de *benchmarking*, se identificaron las mejores configuraciones de redes y se llevó a cabo una comparación detallada con los modelos de clasificación de YOLOv8. Los resultados obtenidos muestran que las redes diseñadas específicamente en este trabajo alcanzan una precisión comparable a las de YOLOv8, con diferencias mínimas en torno al 0,4 % de *accuracy* . Sin embargo, una de las principales ventajas de las redes creadas radica en su velocidad de inferencia, siendo hasta diez veces más rápidas que las redes de YOLOv8, lo que representa un avance significativo en términos de eficiencia computacional 4.5

El análisis de los errores cometidos por las redes ha permitido identificar patrones relevantes. La mayoría de los fallos se concentran en imágenes irreconocibles incluso para el ojo humano, así como en señales direccionales que tienden a confundirse con sus contrarias debido a la similitud visual y la orientación opuesta de las flechas. Este hallazgo pone de manifiesto los principales puntos débiles de los modelos y sugiere posibles líneas de mejora, como el uso de técnicas de preprocesamiento avanzadas o el incremento de ejemplos de señales problemáticas durante el entrenamiento.

Adicionalmente, se ha desarrollado un *pipeline* automatizado que integra ambas etapas de detección y clasificación, permitiendo el procesamiento continuo de secuencias de video y demostrando la viabilidad de implementar este sistema en aplicaciones reales, como sistemas avanzados de asistencia al conductor (*ADAS*).

En conclusión, este TFG ha logrado desarrollar un sistema eficiente, modular y competitivo para el reconocimiento de señales de tráfico. La comparación con modelos de referencia y el análisis detallado de los resultados obtenidos respaldan la validez del enfoque adoptado, sentando una base sólida para futuros trabajos en este campo.

5.1 Futuras líneas de investigación

El abanico de posibilidades de ampliación de este TFG es inmenso, y las líneas futuras de investigación y desarrollo pueden abarcar desde mejoras en el rendimiento de los modelos hasta su integración en sistemas reales. A continuación, se detallan algunas de las principales áreas de trabajo que podrían explorarse:

En primer lugar, sería posible extender el proceso de entrenamiento llevado a cabo para las redes de clasificación utilizando un conjunto de datos más completo que incluya todas las señales de tráfico existentes en España. Actualmente, el conjunto de datos GTSDB cubre una gran variedad de señales, pero está limitado a un entorno específico. Un dataset más amplio y diverso permitiría entrenar un sistema capaz de reconocer cualquier tipo de señal en diferentes escenarios reales, como zonas urbanas, rurales o carreteras secundarias, aumentando la robustez y la capacidad de generalización del modelo. Esta ampliación requeriría no solo un proceso de etiquetado adecuado, sino también volver a hacer el proceso de *benchmarking* buscando la mejor configuración de hiperparámetros para este nuevo conjunto de datos.

Se podrían incorporar técnicas de *data augmentation* para potenciar la robustez del sistema de reconocimiento de señales de tráfico. En futuras iteraciones se estudiaría la aplicación de transformaciones geométricas, como rotaciones, traslaciones, escalados y volteos, junto con modificaciones fotométricas, que incluyan variaciones en brillo, contraste, saturación y la introducción controlada de ruido. Estas técnicas permitirán generar un conjunto de datos más diverso y representativo de las condiciones reales de captura, favoreciendo el aprendizaje de características invariantes ante alteraciones en la imagen. Además también se podría llegar a mitigar con dichas técnicas la poca descompensación del dataset en cuanto a número de imágenes. Aunque esto último no ha resultado problemático para llegar a buenos resultados en este TFG.

Otra línea de trabajo interesante sería la cuantificación de las redes desarrolladas. La cuantificación consiste en reducir la precisión de los pesos y activaciones de la red, pasando de 32 bits de punto flotante a representaciones de menor precisión, como 16 o incluso 8 bits. Este proceso reduce significativamente el coste computacional y el consumo de memoria, lo que hace que las redes sean más eficientes y adecuadas para su ejecución en dispositivos periféricos o *edge*. Una vez cuantificadas, las redes podrían integrarse en hardware especializado, como las plataformas NVIDIA Jetson o dispositivos FPGA, y evaluarse su comportamiento en términos de precisión y velocidad de inferencia en condiciones reales. Esta línea no solo permitiría llevar el sistema a entornos con restricciones de hardware, sino también explorar soluciones escalables y económicas para el mercado automovilístico.

En cuanto a la integración del sistema, se podría desarrollar una aplicación modular integrada en los vehículos, capaz de alertar al conductor sobre las señales detectadas en tiempo real. Esta aplicación podría conectarse a las pantallas de los vehículos o a sistemas de realidad aumentada, mostrando información relevante, como señales de prohibición, límites de velocidad o advertencias importantes. El desarrollo de una interfaz intuitiva y amigable para el conductor sería clave para asegurar una correcta interacción con el sistema, minimizando las distracciones y maximizando la utilidad de la información proporcionada.

Finalmente, se podrían explorar métodos de retroalimentación continua para el sistema. Esto implicaría desarrollar un mecanismo mediante el cual el sistema pueda aprender de los errores detectados durante su uso en campo, ajustando sus parámetros y mejorando su rendimiento sin necesidad de un reentrenamiento completo. Este enfoque de aprendizaje en línea permitiría que el sistema se adapte continuamente a nuevos entornos y situaciones, aumentando su eficacia a largo plazo.

En resumen, las líneas futuras de este TFG abarcan desde mejoras en los modelos y su entrenamiento hasta el desarrollo de aplicaciones prácticas e integraciones reales. Estas propuestas no solo

permitirían llevar el sistema a un nivel superior de madurez tecnológica, sino que también abrirían la puerta a nuevas investigaciones en el campo de la visión artificial aplicada a la seguridad vial.

Índice de Figuras

Índice de Tablas

Bibliografía

[1] Jose Mariano Alvarez, *El perceptrón como neurona artificial*, June 2018.

[2] Aqeel Anwar, *What is average precision in object detection & localization algorithms and how to calculate it?*, 2022.

[3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Tuan Mark Liao, *Yolov4: Optimal speed and accuracy of object detection*, (2020).

[4] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber, *A committee of neural networks for traffic sign classification*, The 2011 International Joint Conference on Neural Networks (2011).

[5] A. de la Escalera, L.E. Moreno, M.A. Salichs, and J.M. Armingol, *Road traffic sign detection and classification*, IEEE Transactions on Industrial Electronics (1997).

[6] Comisión Europea, *Reglamento de ejecución (ue) 2022/1426 de la comisión*, 2022.

[7] Ivan Gargate and Michael Gargate, *Metrics matter: A deep dive into object detection evaluation*, 2023.

[8] Margherita Grandini, Enrico Bagli, and Giorgio Visani, *Metrics for multi-class classification: an overview*, arXiv (2020).

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

[10] IBM, *What is machine learning?*

[11] Dr. Shalini Lamba, Pranav Upreti, and Priyanshu Tiwari, *Neuro-ai fusion: Connecting neuroscience and artificial intelligence*, (2024).

[12] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner, *Gradient-based learning applied to document recognition*, (1998).

[13] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei, *Yolov6: A single-stage object detection framework for industrial applications*, (2022).

[14] Mayank Mishra, *Convolutional neural networks, explained*.

[15] Petru Potrimba, *What is a convolutional neural network?*

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, *You only look once: Unified, real-time object detection*, (2016).

[17] Joseph Redmon and Ali Farhadi, *Yolo9000: Better, faster, stronger*, (2016).

[18] ———, *Yolov3: An incremental improvement*, (2018).

[19] W. Ritter, *Traffic sign recognition in color image sequences*, (1992).

[20] Frank Rosenblatt, *Principles of neurodynamics*, 1962.

[21] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel, *The german traffic sign recognition benchmark: A multi-class classification competition*, The 2011 International Joint Conference on Neural Networks (2011).

[22] Ultralytics, *Ultralytics x paperspace: Mejora de las capacidades de detección de objetos mediante la asociación*, 2025.

[23] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao, *Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, (2022).