Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

SaferAuto

Sistema de asistencia a la conducción.
Driving assistance system.

Ángel Luis Igareta Herráiz

La Laguna, 10 de junio de 2019

D. **Jesús Manuel Jorge Santiso**, con N.I.F. 42.097.398-S profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"*SaferAuto: Sistema de asistencia a la conducción.*"

ha sido realizada bajo su dirección por D. **Ángel Luis Igareta Herráiz**, con N.I.F. 79.062.100-Z.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2019

Agradecimientos

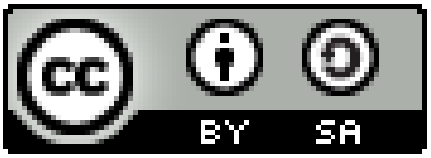
Agradecer a mi familia y amigos por el soporte indispensable y constante que me han dado durante la carrera incluso en los momentos en los que he podido estar mas estresado y reticente.

Agradecer a mis compañeros del Grado: Cristian, Carlos, Daute, Alberto y Juan Pablo por el increíble grupo que hemos formado éstos 4 años, resolviendo dudas y ayudándonos entre nosotros. Gracias porque sin ustedes no hubiera sacado la carrera de la misma manera.

Agradecer a todos los profesores de la carrera que tienen pasión por su trabajo, que son capaces de escuchar al alumnado, que transmiten todo el conocimiento que saben e intentan seguir aprendiendo de lo que aún no saben. Gracias porque son ustedes los que realmente hacen que el alumnado salga preparado y tenga pasión por la Ingeniería Informática.

Agradecer a las personas que he conocido al irme de Erasmus o alrededor de Europa a través de la asociación AEGEE. Gracias por haberme hecho aprender a disfrutar la vida y convertirme la persona positiva y extrovertida que soy hoy.

Licencia



Resumen

A lo largo de la historia, el uso de la Inteligencia Artificial y el Aprendizaje Automático ha ido creciendo exponencialmente, ayudando a resolver problemas relevantes como la detección de *spam*, la recomendación de productos o diagnósticos médicos, por ejemplo.

Sin embargo, existen problemas de suma importancia que todavía no han sido resueltos, como el de los accidentes de tráfico. Según la Dirección General de Tráfico [7], durante el año 2017 en España se produjeron 102.233 accidentes de tráfico con víctimas, de los que un 29 % fueron producidos por la distracción al volante.

Para contribuir a la reducción en los accidentes de tráfico causados por distracción y ayudar a mejorar la seguridad vial, se ha desarrollado SaferAuto, un sistema de asistencia a la conducción capaz de detectar señales de tráfico verticales y semáforos en tiempo real.

Este informe presenta los dataset europeos mas completos para el reconocimiento de señales de tráfico verticales y semáforos. A su vez, se realiza un estudio de las tecnologías del estado del arte que existen en la actualidad para la detección y clasificación de objetos en tiempo real, decidiendo utilizar YOLO en su versión 3 con la arquitectura *yolov3-tiny*.

Finalmente se ha desarrollado un sistema capaz de detectar 10 tipos de elementos viales (prohibición, peligro, obligación, información, ceda el paso, stop, no entrar, semáforo verde, semáforo ámbar y semáforo rojo) y clasificar las señales de límite de velocidad. La precisión media obtenida es del **88 %**, llegando al 94 % de mAP en clases como la de peligro o semáforo rojo. Además, se ha conseguido alcanzar una tasa de análisis de **220 imágenes por segundo.**

Palabras clave: inteligencia artificial, conducción autónoma, detección señales de tráfico

Abstract

During the last few years, the use of Artificial Intelligence and Machine Learning has been growing exponentially, helping to solve relevant problems such as the detection of spam, products recommendations or some medical diagnostics, for example.

However, there are major problems that have not yet been solved, such as traffic accidents. According to the *Dirección General de Tráfico* [7], during 2017 in Spain there were 102,233 traffic accidents with victims, of which 29 % were caused by distraction.

To contribute to the reduction in traffic accidents caused by distraction and help improve road safety, SaferAuto, a driving assistance system capable of detecting vertical traffic signs and traffic lights in real time, has been developed.

This report presents the most complete European datasets for the recognition of vertical traffic signs and traffic lights. At the same time, a study of the state of the art technologies for the detection and classification of objects in real time is made, deciding to use YOLO in its version 3 with its architecture *yolov3-tiny*.

Finally, a system has been developed capable of detecting 10 types of road elements (prohibition, danger, obligation, information, yield, stop, no entry, green traffic light, amber traffic light and red traffic light) and classifying speed limit signs. The average accuracy obtained is **88 %**, reaching 94 % of mAP in classes such as danger and red traffic light. Besides, we have achieved a detection rate of **220 images per second.**

Keywords: artificial intelligence, self-driving, traffic sign detection, real-time

Índice general

Capítulo 1

Introducción

1.1. Motivación

El término Big Data ha ganado gran popularidad en los últimos años, especialmente por su utilización en una amplia variedad de campos. Este concepto hace referencia a una gran cantidad de datos estructurados y no estructurados que tienen el potencial de ser analizados para obtener información. Cada día mas de 2.5 exabytes (2.500.000.000 gigabytes) de datos son almacenados. Esto indica que la generación de un 90 % de los datos recolectados en la historia de la humanidad se ha producido en tan solo los dos últimos 2 años [2]. Toda ésta información puede provenir de multitud de lugares como de sensores para recoger información meteorológica, publicaciones en redes sociales o las señales GPS móviles, por ejemplo.

Debido a este aumento en la cantidad de información disponible, cada vez resulta mas difícil tratar con los datos, por lo que se requiere del uso de nuevas tecnologías para poder obtener información útil de los mismos. En consecuencia, el uso de la Inteligencia Artificial y el Aprendizaje Automático ha ido creciendo exponencialmente, con el fin de generar una inteligencia que aprenda de esos datos y que sea capaz de resolver retos como puede ser el del reconocimiento de dígitos escritos a mano. De hecho, el conjunto de datos abierto mas popular para ese problema se llama MNIST [18] y se considera el 'Hola Mundo' en la Visión por Ordenador. Este conjunto de datos se publicó en 1999 y contiene miles de imágenes con dígitos escritos a mano. Al ser uno de los problemas mas estudiados en la literatura, podemos encontrarnos incluso aplicaciones para nuestro móvil capaces de reconocer dígitos escritos a manos con precisión aceptable.

Sin embargo, existen problemas con mayor importancia que todavía no han sido resueltos, como el de los accidentes de tráfico. Según la Dirección General de Tráfico [7], durante el año 2017 en España se produjeron 102.233 accidentes de tráfico con víctimas, en los que 1.830 personas perdieron la vida y 139.162 resultaron heridas. El 29 % de los accidentes con víctimas se deben a la distracción, lo que indica una falta de atención a la carretera pudiendo ser por la utilización del teléfono móvil, falta de sueño u otros factores externos. Como consecuencia, un 46 % de los fallecidos en accidentes son los colectivos vulnerables: peatones, ciclistas y motoristas.

1.2. Objetivo General

Para contribuir a la reducción en los accidentes de tráfico causados por distracción y ayudar a mejorar la seguridad vial, este Trabajo de Fin de Grado consistirá en desarrollar un sistema de asistencia a la conducción de código abierto, capaz de detectar elementos de la carretera como señales de tráfico verticales o semáforos y notificar al conductor de manera inmediata para ofrecer un mayor tiempo de reacción y poder reducir el número de accidentes.

Cabe añadir que el sistema a desarrollar no pretende en ningún caso sustituir la responsabilidad del conductor, se trata de una sistema añadido para asistir durante la conducción. A pesar de que un requisito fundamental sea obtener la máxima precisión en la detección de señales, SaferAuto podría no detectar alguna señal o clasificar algunos elementos viales de manera errónea.

1.3. Objetivos Específicos

1. **Creación de un modelo de Aprendizaje Profundo (Deep Learning) para la detección de señales de tráfico verticales.** Además, se perseguirá clasificar las señales de límite de velocidad. Es decir, interpretar la velocidad máxima que indica la señal detectada.

2. **Adaptación del modelo creado anteriormente para hacer posible la detección de semáforos.** Al igual que en el caso anterior, se perseguirá clasificarlos según su superclase: rojo, ámbar o verde.

3. **Creación de una interfaz gráfica de usuario que visualice la entrada de vídeo junto a las detecciones de los elementos contemplados en el modelo.** Además de visualizar las notificaciones, se avisarán mediante mensajes de audio, simulando un *chatbot*.

1.4. Requisitos

En este apartado se presentarán una serie de requisitos necesarios para que el producto a desarrollar sea útil.

En primer lugar, se requerirá que la detección de los elementos de la carretera presentados anteriormente como los semáforos o las señales de tráfico verticales se realice en **tiempo real**. Esto quiere decir que el sistema sea capaz de analizar mas de 30 imágenes por segundo (fps). Para que un asistente a la conducción como SaferAuto sea realmente útil, debería poder avisar al conductor antes de que a este le haya dado tiempo a ver o diferenciar el elemento vial, por ello es tan importante que la tasa de análisis de imágenes por segundo sea lo mas alta posible.

En segundo lugar, es importante que **la detección de los elementos sea precisa y que el sistema sea capaz de detectar diferentes tamaños del mismo elemento**.

Se debe tener en cuenta que, a la hora de conducir, el tamaño de los elementos que se visualiza mas a la distancia varía significativamente cuando el vehículo se ha acercado a los mismos. Por ello, el modelo debe ser entrenado para que detecte éstos objetos en gran variedad de tamaños.

En tercer lugar, en cuanto a la precisión del sistema, se deberá fijar un umbral para que se reduzca al máximo la cantidad de falsos positivos, aquellos elementos que el sistema detecta de una clase cuando éstos no pertenecen a la misma. Este requisito se debe a que el sistema actuará en un área que es crítica para la seguridad vial, dónde la notificaciones deben ser fiables para no causar accidentes. Por ejemplo: a la hora de detectar una señal de límite de velocidad de 40, el sistema no debe confundirse con una de tipo 120, pues podría causar un siniestro. No obstante, también es de suma importancia prestar atención a los falsos negativos, todas aquellas señales que el sistema no ha sido capaz de detectar o clasificar. Al igual que en el caso anterior, no notificar de una señal de stop cuando se encontraba en la carretera podría afectar de manera crítica a la seguridad vial.

En último lugar, para que las notificaciones sean valiosas para el usuario, se deberá hacer seguimiento de los elementos detectados en tiempo real. Si el sistema emitiese una notificación cada vez que detecte una señal, se emitirían mas de 30 notificaciones por segundo, lo que no sería ni entendible ni usable. Por ello, en la transición de una imagen a otra, se deberá detectar que elementos son nuevos o cuales ya se habían visto. Una vez se consiga realizar esto, solo se notificaría al usuario ante nuevas señales, no de las que ya se le había avisado.

1.5. Antecedentes teóricos

Se ha realizado un estudio de los artículos mas recientes en el ámbito de reconocimiento de señales de tráfico y a continuación se presentan algunos de los trabajos mas relevantes.

1.5.1. Traffic Sign Recognition – How far are we from the solution?

En este primer artículo [23] se estudia el campo de reconocimiento de señales de tráfico y se muestra como se comportan algunas tecnologías de Visión por Computador modernas en la detección de dos conjuntos de datos de detección y de clasificación, con imágenes provenientes de Bélgica [39] y Alemania [15].

Para la detección las imágenes de ambos datasets se dividen en tres categorías principales basadas en su forma y color: obligación, peligro y prohibición. Tras realizar la detección de la categoría, las señales se clasifican en subclases. Entre las tecnologías que utilizan destacan el histograma de gradiente orientado para la tarea de detección y el clasificador de representación dispersa para la tarea de clasificación.

Como resultado, sin modificaciones específicas para la aplicación, utilizando métodos similares al de detección de peatones o clasificación de dígitos, alcanzan una precisión de reconocimiento de señales de tráfico en el rango de entre 95 % a 99 %. Cabe destacar que

no es un problema en el contexto de tiempo real, por ello, el tiempo de reconocimiento no está por debajo de los 0.5 segundos por imagen.

1.5.2. Evaluating State-of-the-art Object Detector on Challenging Traffic Light Data

Este artículo [17] aplica el sistema de detección de objetos YOLO (You Only Look Once) en el conjunto de datos público *LISA Traffic Light dataset* [16], el cual contiene gran número de anotaciones de semáforos en diferentes condiciones de tiempo. Este dataset es de acceso abierto gracias al VIVA-Challenge, en el que se fomenta la investigación en el campo de Visión Por Computador.

Además, en el desarrollo del estudio se comparan las distintas versiones del algoritmo YOLO y como se comportan en las mútiples secuencias del dataset. Este trabajo se asimila al primer objetivo perseguido con SaferAuto, únicamente enfocado en detectar y clasificar las distintas clases de semáforos.

Finalmente, el sistema de detección utilizado alcanza un 90.49 % de precisión sobre la primera secuencia del dataset LISA-TL, lo que supuso una mejora del 50.32 % del que había sido la entrada con mayor precisión del reto VIVA.

1.5.3. Real-Time Object Detection For Autonomous Driving Based On Deep Learning

En ésta tesis de Guangrui Liu, se estudia la historia de los sistemas de detección de objetos y cual ha sido su evolución. Expone varios métodos basados en Redes Neuronales Convolucionales y se centra en el estudio del modelo de detección YOLO, que permite detección en tiempo real.

Durante el desarrollo, se utiliza este modelo en tres conjuntos de datos distintos para testear su aplicabilidad en diferentes contextos. Junto a ello, se realizan varias pruebas sobre el conjunto de datos *KITTI*, con el objetivo de analizar el rendimiento del algoritmo de detección sobre el campo de la conducción autónoma [22].

Su contribución se basa en proponer una técnica de mapeado de memoria para mejorar la habilidad de detección de YOLO en el contexto de conducción. A su vez, se exploran otros potenciales de ésta tecnología como su aplicación para estimar la orientación de los objetos, alcanzando resultados bastante positivos.

1.6. Interés y Actualidad del campo

Recientemente, multitud de compañías líderes del sector automovilístico como Google, Tesla o Uber están invirtiendo una gran suma de capital en los sistemas de conducción autónoma, así como en los sistemas avanzados de asistencia a la conducción (ADAS) [21]. La diferencia entre estos dos sistemas radica en que el primero permite al vehículo

la conducción por si mismo sin requerir intervención de un ser humano. Sin embargo, el segundo se trata de un paso intermedio a la conducción autónoma en el que se incluyen distintos niveles de automatización:

- **Nivel 0: Sin automatización** - El conductor se encarga de realizar todas las funciones referentes a la conducción.
- **Nivel 1: Asistencia al conductor** - El sistema emite alertas y control parcial en aparcamiento, dirección y acelerador.
- **Nivel 2: Automatización parcial** - El sistema controla la dirección y aceleración y el conductor las tareas restantes, como por ejemplo los cambios de línea.
- **Nivel 3: Asistencia condicional** - El sistema es capaz de realizar todas las tareas necesarias para la conducción, pero un humano debe estar presente para tomar el control en ocasiones específicas.
- **Nivel 4: Gran automatización** - La automatización del sistema equivale a la del nivel 3 pero se requerirá en menor número de ocasiones de intervención humana.
- **Nivel 5: Automatización completa** - El sistema toma control de todas las tareas necesarias para la conducción y no requiere ayuda humana en ninguna situación.

Un ejemplo de la inversión en el campo de conducción autónoma es la compra con valor de 15.3 mil millones de dólares de Mobileye por parte de Intel [21]. La empresa en cuestión es líder en el campo de Visión por Computador y de tecnologías para la conducción autónoma.

Es tal el auge en este mercado que cuando en una entrevista le preguntaron a Elon Musk, CEO de Tesla, que opinaba sobre el piloto automático respondió: *"En 20 años la razón de tener un coche no autónomo será la misma que por quien tenga un caballo, por razones sentimentales"*. También predijo que en alrededor de 2 años Tesla estará construyendo coches sin volante ni pedales, pues serán totalmente autónomos.

Por último, informes del 'Autonomous Vehicle Global Study 2017' muestran que, en el año 2035, se espera que se vendan en el mercado automovilístico alrededor de 18 millones de vehículos con funciones autónomas, lo que supondría más de un 25 % de cuota de mercado. Con ello, el precio de este mercado crecería hasta 77 mil millones de dólares [21].

1.7. Estado del Arte

Al ser un campo tan atractivo, existen variedad de productos finales desarrollados por empresas de gran tamaño. En este apartado se presentarán algunos de los mas populares y avanzados.

1.7.1. EyeSight

Se trata de una tecnología de asistencia a la conducción que provee la marca automovilística japonesa Subaru, con la que han sido equipados y vendidos alrededor de 1 millón de vehículos [33]. Sus funciones principales son:

- Asistencia al mantenimiento de carril.
- Adaptación de la distancia con el vehículo de delante. Permite la selección de la distancia que el conductor desee preservar y ajusta la velocidad para mantenerse a esa distancia.
- Frenado pre-colisión. El sistema ayuda evitando o reduciendo impactos frontales a través del completo frenado en situaciones de emergencia. Según Subaru, este sistema reduce un 35 % de los daños a peatones.

1.7.2. Tecnología Ford

La multinacional americana Ford incluye una serie de tecnologías para la asistencia a la conducción en sus vehículos de alta gama [8]. Entre sus características destacan:

- Contacto con servicios de emergencia en caso de accidente.
- Limitador de velocidad inteligente. El sistema permite mantener al vehículo dentro de los límites de velocidad legales gracias a la detección de señales verticales de este tipo y el correspondiente ajuste.
- Sistema de información de ángulos muertos. Emplea dos sensores de proximidad ocultos que avisan al conductor en caso de que se detecten vehículos en los puntos ciegos.
- Sistema de reconocimiento de señales de tráfico. Detecta las señales permanentes o temporales situadas junto a la carretera. El sistema muestra la señal detectada sobre el salpicadero. Esta característica está íntimamente relacionada con los objetivos de SaferAuto y se perseguirá desarrollar una interfaz de estilo similar para que sea visual y útil.

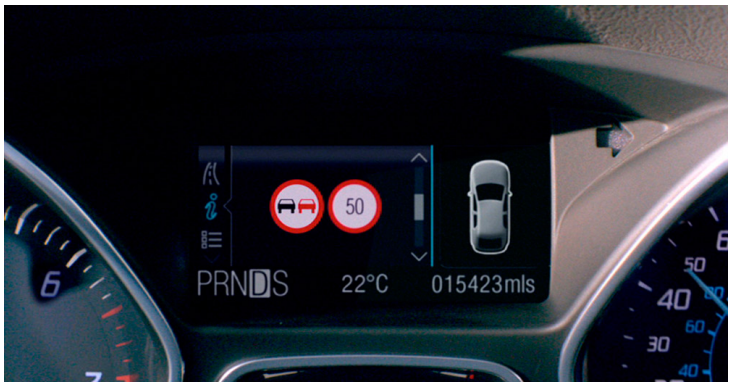


Figura 1.1: Ford - Sistema de reconocimiento de señales de tráfico [8]

1.7.3. Mazda Active Driving Display

La compañía automovilística Mazda incluye también funciones innovadoras en algunos de sus nuevos vehículos. La característica mas relevante en lo relativo a la conducción autónoma es su Visualización Activa de la Conducción o Active Driving Display [3]. En esencia, se trata de una pantalla de control que proporciona información relevante al conductor como la velocidad actual o las direcciones de navegación.

Figura 1.2: Mazda Active Driving Display [3]

La Visualización Activa de la Conducción colecciona la mayoría de información importante para la conducción como la velocidad, dirección o incluso señales de tráfico. Tiene como objetivo agrupar todos estos datos en una localización que resulte cómoda al conductor, con el fin de que su visión se concentre en la carretera.

En la actualidad, existen dos versiones de esta pantalla de control dependiendo del modelo de coche que se elija. La primera, utiliza una pequeña pieza de plástico transparente que se despliega de detrás del volante y un sistema de proyección visualiza información relevante en la pantalla. Por otra parte, en algunos modelos la proyección se realiza en una pequeña sección del parabrisas, siendo esta mas sensible a la luminosidad o los ángulos de visión.

1.7.4. Tesla Autopilot

Para finalizar se expondrá el estado del arte en sistemas de conducción autónoma. Tesla se trata de una compañía estadounidense, dirigida por Elon Musk, que diseña y construye principalmente vehículos eléctricos. Se diferencia de las demás marcas en el propio diseño de sus coches, el cuál rompe con los estándares tradicionales y está pensado en términos de eficiencia y usabilidad.

Tesla ofrece una tecnología llamada Autopilot [37] instalada en sus vehículos. Se trata de un hardware avanzado capaz de ofrecer las funciones de piloto automático en el presente y futuro, actualizándose para mejorar su funcionalidad con el tiempo. Para hacer posible ésta tecnología, sus vehículos poseen una visión panorámica de 360 grados gracias a llevar instaladas ocho cámaras, cada una con un alcance de hasta 250 metros. También utilizan doce sensores ultrasónicos, y un radar delantero para brindar datos adicionales sobre el mundo a través de una longitud de onda redundante, como pueden ser la lluvia intensa, neblina o el vehículo que les precede.

Para poder procesar todos los datos recibidos de cámaras y sensores, los vehículos de Tesla contienen un ordenador integrado que ejecuta una red neuronal desarrollada por la compañía. Este sistema, llamado Tesla Vision, proporciona una vista única del mundo, pues ve en todas las direcciones de manera simultánea, yendo mas allá de los sentidos humanos.

Una lista de algunas de las funciones del piloto automático son:

- Permitir que el coche gire, acelere y frene automáticamente dentro de su carril.
- Proponer cambios de carril para optimizar su trayecto, evitando quedarse atascado tras camiones o vehículos lentos.

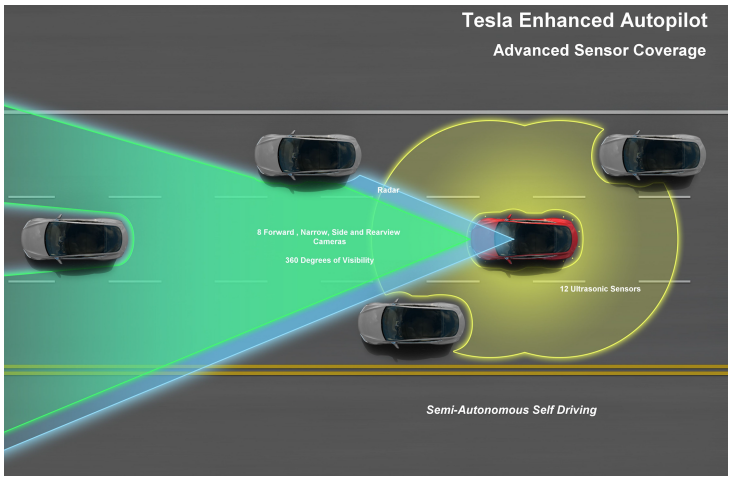


Figura 1.3: Tesla - EL futuro de la conducción [35]

Por último, la opción de conducción autónoma total esta siendo desarrollada. En abril de 2019, Tesla ha dejado ver algunos avances de este sistema en producción, aunque todavía no se puede activar en todos los contextos o lugares.

La idea de esta característica es que el coche sea capaz de realizar viajes de corta y larga distancia sin requerir acción ninguna por parte de la persona en el asiento del conductor. El usuario solo necesitaría acceder al vehículo e indicar a su Tesla a donde quiere ir, este averiguaría la ruta óptima, conduciría por calles urbanas respetando los semáforos o señales verticales como stop y sería capaz de navegar por autopista con alto tráfico. Aunque parezca idílico, en la actualidad ya se encuentran publicados varios vídeos de trayectos en los que el conductor no ha necesitado intervenir [37].

Capítulo 2

Fundamentos y Tecnologías a usar

2.1. Introducción

Este capítulo tiene como objetivo presentar la investigación realizada sobre el estado del arte en el reconocimiento de imágenes, así como explicar el funcionamiento de las tecnologías que se van a utilizar en este proyecto.

2.1.1. Tareas a desarrollar

Para realizar un estudio de las tecnologías que se utilizarán en este trabajo, previamente se deben diferenciar las dos tareas principales que habrá que llevar a cabo: la tarea de detección y la tarea de clasificación.

En primer lugar, la tarea de clasificación es el proceso de recibir una imagen y devolver una etiqueta de la clase a la que pertenece la imagen, dentro de un conjunto de etiquetas. Por ejemplo, si el conjunto de etiquetas se trata de [Perro, Gato] y el sistema clasificador recibe la imagen 2.1, la salida sería Gato.

Figura 2.1: Ejemplo tarea clasificación.

Por el contrario, la tarea de detección es el proceso en el que se recibe una imagen y se devuelve un conjunto de los objetos encontrados dentro de la imagen, representados

mediante su etiqueta y el cuadro delimitador que lo rodea. Por ejemplo, si recibimos una imagen en la que aparecen varios objetos como un jarrón, un ordenador portátil y una taza y el sistema de detección está entrenado para detectar éstos objetos, la salida que esperamos será un conjunto de etiqueta/cuadro delimitador por cada objeto detectado, tal y como se observa en la figura 2.2.

Figura 2.2: Ejemplo tarea detección.

Para el desarrollo de SaferAuto, se deberán realizar las dos tareas, la tarea de detección para localizar los distintos elementos de la carretera junto a su categoría y la tarea de clasificación para averiguar a que subclase pertenece el elemento detectado de una categoría. Esto se explicará con mas detalle en secciones posteriores. Previamente, es importante hacer un estudio de las tecnologías que se requerirán para hacer posible la ejecución de ambas tareas.

2.1.2. Métodos de Detección de Objetos

Tras realizar un estudio de los métodos utilizados para la de detección de objetos, pueden ser divididos en dos categorías: basados en modelo y basados en aprendizaje [17].

Los métodos basados en modelo se centran en analizar la información del color o la forma de los elementos, lo que se trata de un enfoque muy intuitivo y directo para la detección de semáforos, por ejemplo. Estos detectores se basan en un umbral heurístico para cierto espacio de colores, con el objetivo de localizar los diferentes objetos de la imagen. Sin embargo, el color no es una variable muy fiable, dado que el tono puede variar significativamente de una escena a otra, por lo que un umbral estático no se comportaría demasiado bien.

Por otro lado, los métodos basados en aprendizaje se encargan de deducir una función a partir de datos de entrenamiento. La salida de la función puede ser una etiqueta de clase o un valor numérico, dependiendo del problema. Un ejemplo sería la combinación de los histogramas de gradientes orientados (HOG) y de las máquinas de soporte vectorial (SVM), cuyo objetivo es separar los elementos de la imagen según sus características. A su vez, otro método popular es el detector de características de canales integrados, conocido

como ChnFtrs, el cual utiliza imágenes integrales para extraer datos como sumas locales o histogramas de distintos canales de la imagen principal y procesarlos, con el objetivo de la clasificación de imágenes.

Finalmente el reconocimiento de objetos utilizando Redes Neuronales Convolucionales (CNNs) ha ido ganando bastante popularidad en los últimos años por sus sobresalientes resultados. En la competición ImageNet de Reconocimiento Visual a Gran Escala (ILSVRC, 2015), los ordenadores se comportaban mejor que los humanos en las tareas de clasificación de imágenes. A su vez, en 2016, un rápido detector de objetos llamado YOLO fue propuesto para expandir la detección de objetos en el ámbito del tiempo real. Está basado en una Red Neuronal Convolucional y se trata del estado del arte en detección de objetos a tiempo real, por lo que es la tecnología que se utilizará para este trabajo.

Antes de pasar a estudiar como funciona YOLO en profundidad, es necesario introducir algunos conceptos básicos para su entendimiento.

2.2. Redes Neuronales

2.2.1. Neurona

Una neurona se trata de la unidad básica de procesamiento que se puede encontrar dentro de una red neuronal. Esta recibe un conjunto de entradas, realiza una serie de cálculos internos y devuelve un valor de salida. En esencia, una neurona calcula una suma ponderada de las entradas. La razón por la que es ponderada se debe a que cada entrada se multiplica por un peso asociado, como se ilustra en la siguiente figura.

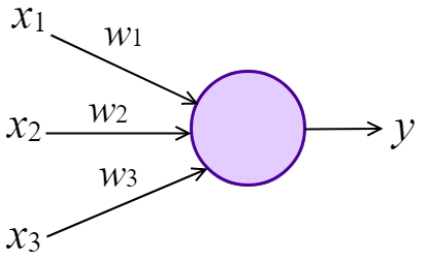


Figura 2.3: Esquema de red neuronal simple.

En la neurona ejemplo de la figura 2.3 las entradas son x_1, x_2 y x_3 y los pesos asociados w_1, w_2 y w_3. El cálculo que realizaría la misma, también llamado z, sería:

$$z = x_1w_1 + x_2w_2 + x_3w_3 = X^\top W$$

En ésta formula $X^\top W$ corresponde a la matriz de las 3 entradas multiplicadas por los 3 pesos asociados. Sin embargo, sería difícil tomar decisiones con esa salida dada que se trata de un valor continuo entre $[-\infty, \infty]$. Por ello, se suele utilizar un tipo de las llamadas *funciones de activación* dependiendo del resultado deseado. Por ejemplo, si se quiere calcular la probabilidad de que haga buen tiempo o no, se debería convertir el

rango de la salida del cálculo de $[-\infty, \infty]$ a $[0, 1]$. Esto se podría hacer utilizando la función de activación sigmoide, ilustrada en la figura 2.4.

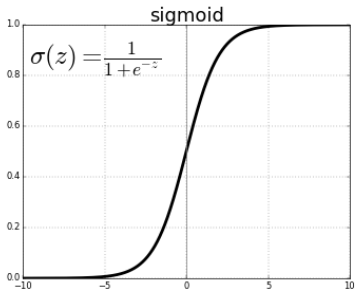


Figura 2.4: Función de activación sigmoide.

2.2.2. Estructura de Red Neuronal

Las redes neuronales consisten en un conjunto de neuronas organizadas en capas, como se puede apreciar en la figura 2.3. A la primera capa se le llama capa de entrada y a la última, capa de salida. A todas las capas del medio se les conoce como *capas ocultas*.

En esta estructura, todas las neuronas se encuentran completamente conectadas, es decir, por cada dos capas de neuronas adyacentes, cada par de neuronas tiene una conexión, por la que se transmite información. Por ejemplo, si una capa tiene m neuronas y la siguiente n, el número total de conexiones será $m * n$. Esto significa que las salidas de una capa son las entradas de cada una de las neuronas de la capa adyacente, por lo que el cálculo de una neurona dependerá de los valores de salida de su capa anterior, realizándose este una única vez.

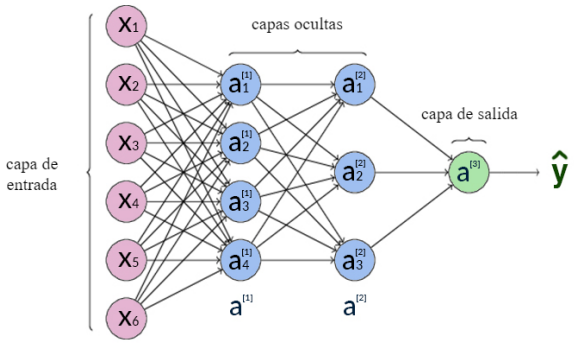


Figura 2.5: Estructura de una red neuronal de 3 capas.

Cabe añadir que las capas pueden ser de distintos tamaños y las neuronas pueden tener distintas funciones de activación incluso dentro de la misma capa, dependiendo del problema. La capa de entrada posee tantas neuronas como componentes tenga el vector X y la capa de salida tantas neuronas como valores se quieran generar. En el proceso,

cada capa va tomando una serie de decisiones. Cuanta mas profunda sea la red, mas inteligentes y complejas serán las decisiones, pues se van basando en las decisiones que han tomado las anteriores capas. Esto se conoce como aprendizaje profundo y gracias a este se han logrado resolver problemas complejos como el de reconocimiento del habla, el procesado de lenguaje natural o la creación de sistemas de recomendación.

2.2.3. Imágenes como entrada

Para que una Red Neuronal fuese capaz de recibir una imagen como entrada, esta se debe representar como una vector X. Recordemos que una imagen no es mas que una matriz de píxeles de dimensión *Alto* x *Ancho* de la imagen. Además, si está en color, existirán tres matrices con las mismas dimensiones, dado que cada píxel es una combinación de los valores rojo, verde y azul (RGB).

Con el fin de representar la imagen en un vector, se añadirá cada matriz convertida a un vector de tamaño *Alto x Ancho*, añadiendo primero la matriz de píxeles rojos, seguida por la de píxeles verdes y finalmente la de píxeles azules. El resultado final será un vector de dimensiones *Alto* x *Ancho* x 3. A este proceso se le conoce como normalización de imagen (figura 2.6). Cabe añadir que en el caso de que la imagen estuviese en escala de grises, el vector tendría dimensiones *Alto* x *Ancho*.

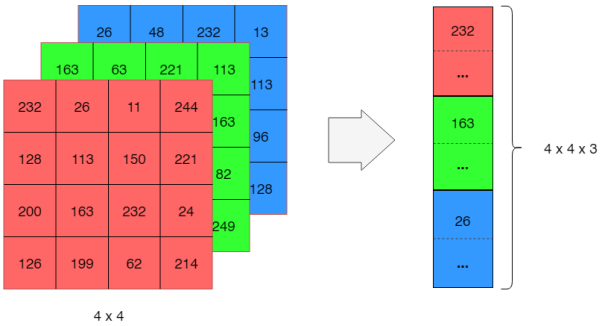


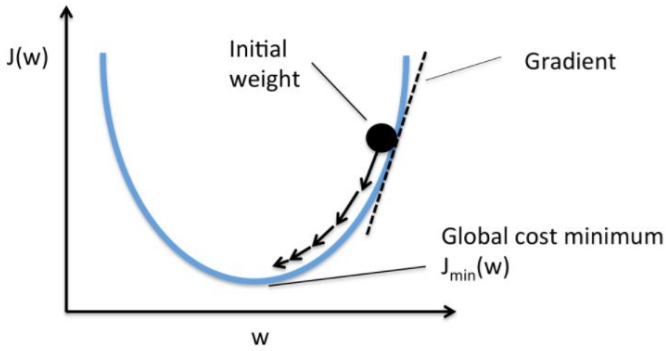
Figura 2.6: Normalización de una imagen de 4 x 4 píxeles.

2.2.4. Entrenamiento de una Red Neuronal

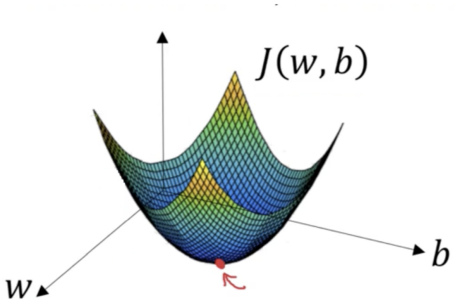
El entrenamiento es el proceso por el cual una red neuronal aprende como llevar a cabo una tarea. En el caso de que se trate de una tarea de clasificación de una imagen, la red neuronal aprendería de los errores entre la clase predicha y la real y, tras el estudio de las características propias de la imagen, ajustaría sus pesos para reducir ese error al mínimo. Esta diferencia entre la clase predicha y la real es calculada haciendo uso de la *Función de Loss o Error*, la cual se explicará en la sección del desarrollo del proyecto.

Durante el entrenamiento, la red neuronal se encarga de reducir gradualmente el error hasta que este sea mínimo. Esto lo hace gracias al descenso por gradiente (gradient descent). Para ilustrar el cambio del error predicho durante el entrenamiento, en la figura 2.7b

se observa una malla que representa la superficie de error. El objetivo del entrenamiento consiste en encontrar el mínimo global de esa superficie.



(a) Descenso por gradiente.



(b) Superficie de error en entrenamiento.

2.2.5. Estadísticos

Durante el desarrollo del proyecto, se generarán gráficas con cada conjunto de entrada utilizado para entrenamiento, con el fin de interpretar el comportamiento de la red. Por ello, previamente es necesario explicar varios estadísticos del campo de Redes Neuronales que serán utilizados para comparar las distintas gráficas.

- Verdadero positivo o *True Positive (TP)*: Acierto de detectar señal que existe en la imagen con su clase y localización correcta.
- Falso positivo o *False Positive (FP)*: Error que se comete cuando se detecta una señal que no existe dentro de la imagen o que si existe pero pertenece a una clase distinta a la clasificada.
- Falso negativo o *False Negative (FN)*: Error que aparece cuando no se detecta una señal que aparece en la imagen.
- Intersección sobre la unión o *Intersection over union (IoU)*: Se encarga de medir que superficie del cuadro predicho se encuentra por encima del cuadro real. Por ejemplo, si se establece un umbral del 50 % y el cuadro predicho no está por encima de este porcentaje del cuadro real, se considera FP.

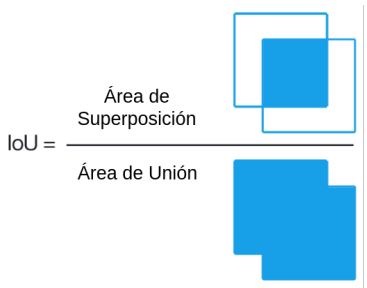


Figura 2.8: Intersección sobre la unión.

- Precisión: Mide cuanto de precisas son las predicciones, es decir, el porcentaje de predicciones correctas. La predicción de una clase se considera correcta si supera un cierto umbral del IoU, en el caso habitual se trata del 0.5, por lo que la predicción es correcta (TP) si $IoU \geq 0{,}5$.

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

- Recall: Mide como de bien se han encontrado todos los positivos. Por ejemplo, si se pueden encontrar un 80 % de las señales que aparecían en la imagen en el conjunto de señales predichas.

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

- Función de Error o *Loss Function*: Se trata del estadístico mas importante para conocer el comportamiento de la red con las imágenes de entrenamiento. En el paper de YOLOv3 [29] se explica como está calculado en dos partes: la función de error de localización ($\mathcal{L}_{loc}$) que calcula la diferencia de predicción del cuadro delimitador del objeto y la función de error de clasificación ($\mathcal{L}_{cls}$) que calcula el error para las clases a detectar.

- Precisión media o Average Precision (AP): Área de la curva precisión-recall.

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

- Promedio de precisión media o Mean Average Precision (mAP): Mide la precisión media para cada clase y realiza un promedio.

Recordemos que el entrenamiento de una red neuronal consiste en modificar los pesos de las neuronas de la red para que la función de error sea mínima. Sin embargo, debemos de tener en cuenta el sobreajuste de pesos o overfitting, que se produce cuando los pesos se ajustan demasiado al conjunto de entrenamiento, lo que implica tener un mAP muy alto para el conjunto de entrenamiento pero, al estar sobreajustado y no generalizar correctamente, se tendría un mAP bajo para el conjunto de validación.

En las gráficas que se generarán durante el entrenamiento se mostrará la función de error y el mAP calculado sobre el conjunto de validación cada 1000 iteraciones. El estadístico que utilizaremos para comparar los pesos entrenados con distintos conjuntos de datos será principalmente el mAP, pues es un buen indicador del rendimiento de la red al calcular la precisión media para cada señal detectada. Cabe añadir, que como segundo factor decisivo se utilizará la tasa de FP, pues buscamos minimizar los errores en clasificación al tratarse de un contexto que afecta a la seguridad vial. Aunque también se estudiarán los FN, al poder detectar mas de 30 imágenes por segundo, habrá mas probabilidad de detectar la mayoría de señales que aparecen en una escena, por lo que sería el último factor de decisión.

2.3. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales o CNN (Convolutional Neural Networks) [34] se tratan de una modificación en la arquitectura de redes neuronales, especializada en el procesamiento de imágenes. Un requisito en las mismas es que la entrada a la red sea una imagen, lo que permite codificar ciertas propiedades de la arquitectura y hacerla mas eficiente para ésta tarea.

Por ejemplo, en una red neuronal de 1 neurona con una imagen de $30\,x\,30$ píxeles, se encontrarían $30\,x\,30 = 900$ conexiones con esa neurona, pues recordemos que cada píxel de la imagen actuaría como entrada en cada neurona de la primera capa. En el caso de que la imagen fuera mas grande, el número de conexiones crecería demasiado, conduciendo una explotación desmesurada de recursos.

El funcionamiento de las CNN es analizar de forma consecutiva pequeñas piezas de información como pueden ser la detección de aristas o de círculos en la primera capa. A medida que se va profundizando en la red, los patrones detectados se van combinando en formas mas complejas como detección de caras, perros o números, por ejemplo. Las capas finales calcularán una suma ponderada de todas los patrones detectados en la imagen para devolver una predicción final.

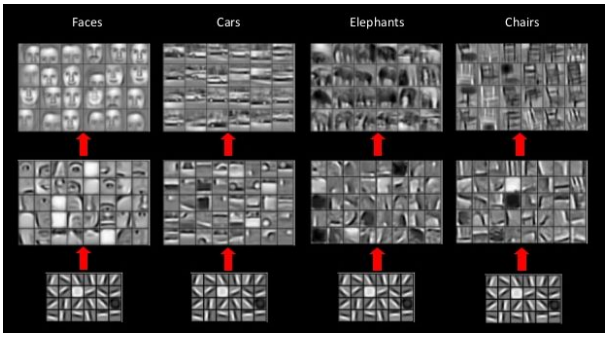


Figura 2.9: Funcionamiento de una Red Neuronal Convolucional.

2.3.1. Tipos de capas de las Redes Neuronales Convolucionales

Generalmente, la estructura de las CNN posee tres tipos de capas [34].

- **Capa convolucional:** Es la capa de entrada y se utiliza en capas sucesivas a esta. Realiza la operación llamada convolución, dando nombre al tipo de red neuronal. Cada neurona de una capa convolucional posee un filtro de convolución, que contiene inicialmente valores aleatorios. Durante el entrenamiento, estos filtros se irán ajustando para detectar patrones dentro de la imagen y ser capaces de reflejarlos en la matriz de salida. La primera capa es responsable de capturar patrones a bajo nivel, como aristas, color, etcétera. Además, dado que se aplica un filtro de menor tamaño que la entrada, la cantidad de parámetros de salida y cálculos se reducirá significativamente.

En la figura 2.10 se aprecian tres matrices: la matriz de la imagen de entrada, el filtro de convolución y la matriz resultante. En este caso el filtro tiene una dimensión de 3 x 3 píxeles. Para calcular la matriz resultante, se deberá ir multiplicando cada matriz de 3 x 3 de la imagen de entrada por el filtro de convolución, tal y como aparece en las operaciones de la ilustración. El resultado será una matriz mas reducida que filtre una serie de características de la matriz de entrada.

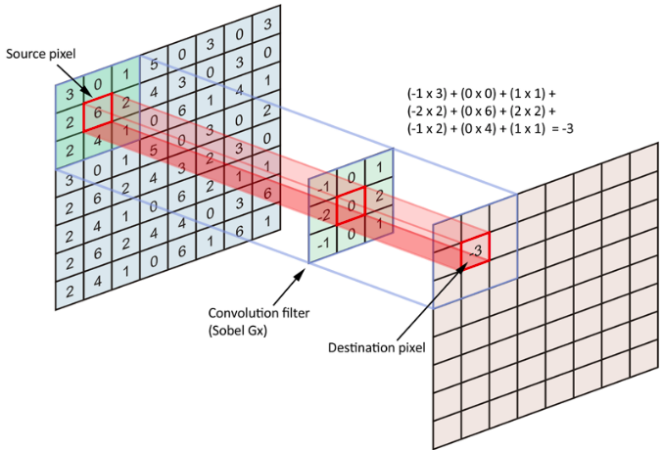


Figura 2.10: Operación de convolución [34].

- **Capa de reducción o pooling:** Está generalmente colocada tras la capa convolucional. Se encarga de reducir las dimensiones espaciales (alto x ancho) del mapa de características recibido de la capa de convolución, de cara a evitar cálculos innecesarios en la próxima operación. Los tipos de operaciones mas usados son el *max-pooling* y el *average-pooling*. Como se ilustra en la figura 2.11, estas operaciones se encargan de dividir la imagen de entrada en un conjunto de rectángulos e ir cogiendo el valor mayor o medio, respectivamente.

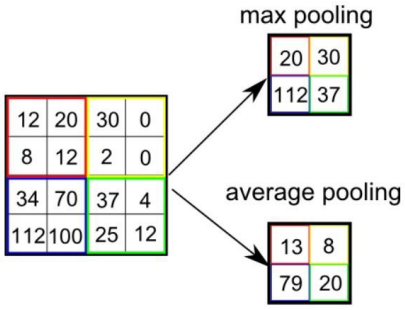


Figura 2.11: Tipos de operaciones de pooling [34].

- **Capa clasificadora:** Al igual que en cualquier Red Neuronal, esta capa se encuentra al final de la red y contiene tantas neuronas como clases se deben predecir. Su objetivo es analizar las formas complejas que se han creado en la red y clasificarlas según la clase con mas formas en común.

2.4. Reconocimiento de Imágenes

En esta sección se estudiará la evolución de las tecnologías usadas para el reconocimiento de imágenes y se expondrán las que han sido mas relevantes hasta llegar al estado del arte.

2.4.1. Historia

Antes del año 2012, los métodos de detección de objetos se enfocaban en la extracción de características de una imagen. Se requería definir las características específicas de una objeto para poder detectarlo. Por ejemplo, en el caso de un sistema de reconocimiento facial, las características específicas podrían ser las distancias entre los elementos de la cara que se quiere reconocer. Tras la extracción de suficientes características del conjunto de datos de entrenamiento, el objeto se representaba como un mapa de características, utilizado para entrenar al clasificador. Las principales desventajas eran la complejidad en la creación del mapa de características y la dificultad de expandir el modelo para añadir nuevos objetos a la detección [22].

En la Competición de Reconocimiento Visual a Gran Escala de 2012 (ILSVRC, 2012), se presentó un modelo basado en Redes Neuronales Convolucionales que superó a los demás en rendimiento con un gran margen. A pesar de que la estructura de CNN modernas existe desde el año 1998 [19], no se había podido alcanzar su potencial hasta poseer grandes conjuntos de datos y un hardware potente. Desde esa competición, las CNN se convirtieron en un herramienta popular para la tarea de clasificación de imágenes.

La principal ventaja de uso de las CNN frente a los métodos tradicionales es que no se necesita la tarea de ingeniería de características, gracias a la habilidad de las CNN del aprendizaje automático. A su vez, esto permite la extensión a otras categorías si se proporcionan datos suficientes. Debido a esto, muchos investigadores comenzaron a estudiar las tareas de detección de objetos dentro de una imagen utilizando CNN. La idea principal de la mayoría de enfoques era utilizar propuestas de regiones de la imagen que pudiesen contener objetos y clasificar estas.

Partiendo de la idea anterior, se podría plantear un enfoque de fuerza bruta en el cual tuviésemos una ventana de por ejemplo un tamaño $3\,x\,3$ y se fuera clasificando cada cuadrado por el que la se desplazara la ventana en la imagen. No obstante, esta solución es computacionalmente muy cara y lenta. A continuación, se explicarán algunos de los enfoques mas populares que optimizan esta tarea.

2.4.2. R-CNN

Para solventar el problema de seleccionar un gran número de regiones, Ross Girshick et al. [9] propone en 2014 un enfoque multi-etapa. Se presenta un método en el que se utiliza una búsqueda selectiva para elegir 2000 propuestas de regiones de la imagen.

El algoritmo de búsqueda selectiva es el siguiente:

- Generar divisiones en la imagen mediante fuerza bruta, consideradas como regiones candidatas.
- Utilizar un algoritmo *greedy* para ir recursivamente combinando regiones con características similares en regiones mas grandes.
- Utilizar las regiones que resulten del algoritmo como las propuestas de regiones finales.

Una vez se tengan las propuestas de regiones finales, pasarán por una CNN que las convertirá en vectores de características. Finalmente estos vectores se enviarán como entrada para entrenar a un clasificador SVM (Support Vector Machine) con el fin de detectar la presencia de objetos en las propuestas candidatas.

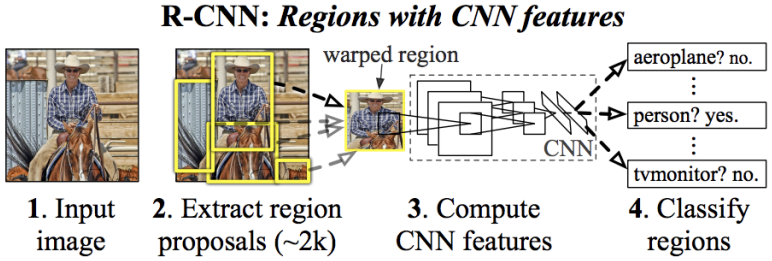


Figura 2.12: R-CNN [9]

Problemas de R-CNN:

- Lleva gran cantidad de tiempo entrenar una red que reciba como entrada 2000 propuestas de regiones por imagen.
- No puede ser implementado en tiempo real dado que detectar cada imagen tomaría alrededor de 47 segundos en un hardware moderno.

2.4.3. Fast R-CNN

Este enfoque fue presentado por el mismo autor del algoritmo anterior (R-CNN) y viene a solucionar las carencias de este, con el fin de aumentar la velocidad en la detección de objetos.

La diferencia con su predecesor es que en FAST R-CNN la Red Neuronal Convolucional no recibe las propuestas de regiones, si no directamente la imagen de entrada, con el fin de generar un mapa de características convolucionales. Desde este mapa se identifican las propuestas de regiones y se pasan por una capa de *poooling* para redimensionarlas. Finalmente, se utiliza una capa *softmax* con el fin de normalizar los resultados y obtener las probabilidades para las distintas clases de la región propuesta.

Al no tener que enviar como entrada las 2000 propuestas de regiones a la Red Neuronal Convolucional, Fast R-CNN resulta mucho mas rápido en entrenamiento y validación,

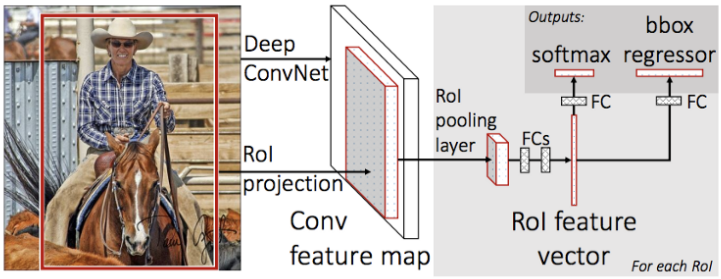


Figura 2.13: Fast R-CNN [31]

llegando a un tiempo de clasificación de cerca de 2 segundos [31]. No obstante, sigue sin poder alcanzar detección de objetos en tiempo real.

2.4.4. Faster R-CNN

El cuello de botella de los dos algoritmos anteriores es la utilización de una búsqueda selectiva para la propuesta de regiones. En el artículo Faster R-CNN [30] publicado por Shaoqing Ren et al. en 2016 se presenta un algoritmo que evita utilizar la búsqueda selectiva de propuestas de regiones y permite que la red las genere.

El funcionamiento inicial es el mismo que el de Fast R-CNN, la imagen de entrada se envía a una CNN para obtener el mapa de características. A continuación, se reemplaza el algoritmo de búsqueda selectiva por una nueva red neuronal que predice propuestas de regiones. Finalmente, estas propuestas se redimensionan utilizando una capa de *pooling* que se utiliza para clasificar la región propuesta de la imagen.

Como se puede apreciar en la siguiente ilustración, Faster R-CNN consigue alcanzar un tiempo de detección de 0.2 segundos por imagen, lo que implica una tasa de 5 imágenes analizadas por segundo, factible para su uso en tiempo real.

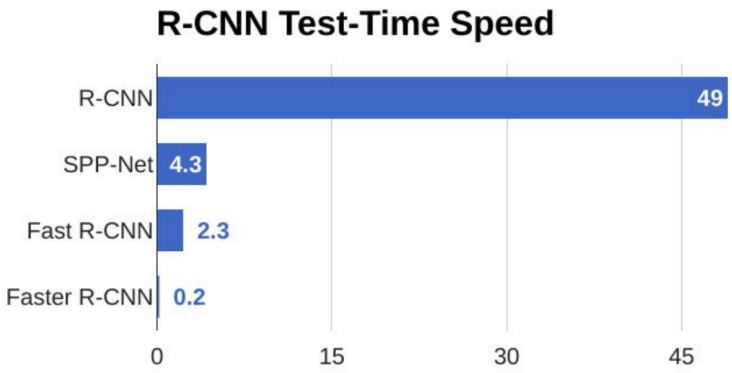


Figura 2.14: Faster R-CNN [30]

2.4.5. YOLO

Los algoritmos previamente expuestos están enfocados en utilizar propuestas de regiones para localizar objetos dentro de una imagen, provocando que la red nunca reciba la imagen completa, sino propuestas de regiones con gran probabilidad de ser objetos. Esto se debe a que los anteriores enfoques son clasificadores rediseñados para ser adaptados a la tarea de detección.

YOLO (You Only Look Once) toma un enfoque completamente distinto, está pensado como un sistema de detección y no como una adaptación de un clasificador tradicional. En esencia, YOLO aplica una sola Red Neuronal Convolucional a la imagen completa. La red divide la imagen en regiones y por cada una de ellas predice cuadros delimitadores y probabilidades de clase [29].

El funcionamiento es el siguiente:

1. YOLO divide la imagen de entrada en una matriz de x por y celdas. En cada celda se podrán detectar hasta un número z de objetos distintos.

2. En cada celda de la matriz se predice la posibilidad de que haya un objeto dentro de la misma y se crea un cuadro delimitador que rodearía a ese posible objeto junto a las celdas adyacentes. El cuadro delimitador no da información sobre la clase del objeto en cuestión, pero su grosor aumenta cuanto mayor sea esta probabilidad.

3. Por cada cuadro delimitador, se predice el objeto que puede haber dentro del mismo. Ahora estos pasarán a estar etiquetados según la clase que contienen, como se observa en la imagen del medio de la figura 2.15.

4. Finalmente, se fija un umbral mínimo de probabilidad que debe tener un cuadro delimitador para que realmente contenga una clase y se eliminan todos los cuadros que no superen ese umbral. Esto devolvería el resultado esperado.

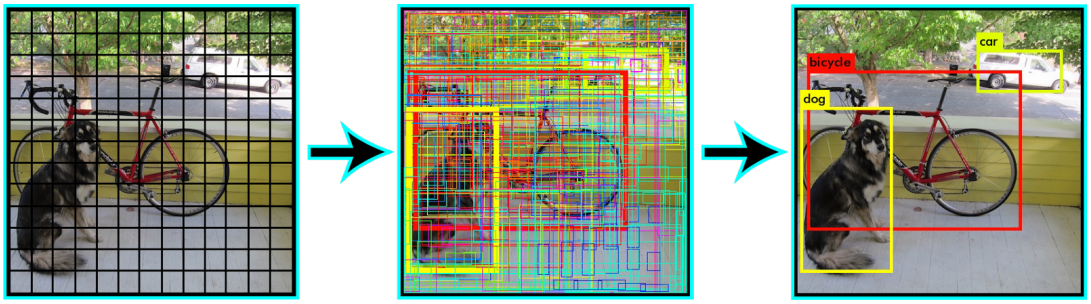


Figura 2.15: YOLO [29]

YOLO ofrece varias configuraciones distintas de su arquitectura, dependiendo de cual sea el objetivo de la tarea, si velocidad o precisión. La arquitectura principal de YOLO consiste en 25 capas convolucionales seguidas de 2 capas completamente conectadas. Tras cada grupo de capas de convolución ilustrado en la figura 2.16, se encuentra una capa de *max-pooling*.

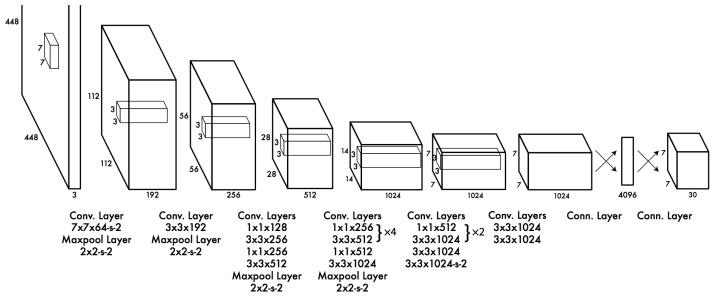


Figura 2.16: Arquitectura de YOLO [29]

Cabe añadir que YOLO se encuentra en la versión 3 [29] y será la que se utilizará en este proyecto. Las diferencias con las versiones anteriores son principalmente mejoras en cuanto a la eficiencia del algoritmo y una mayor configuración del mismo. Haciendo referencia al tiempo de detección por imagen, YOLO supera significativamente a los enfoques anteriores y consigue alcanzar una tasa de 45 imágenes detectadas por segundo en su versión principal (v3-320) y una tasa de 220 en su versión ligera (v3-tiny).

Las principales diferencias entre ambas arquitecturas (320 y tiny) son la cantidad de capas que las componen y el número de operaciones de coma flotante que realizan por segundo (flops), calculando el primero 38.97 billones y la versión ligera 5.56 billones. También, la diferencia en el número de capas provoca que la versión ligera obtenga una precisión mas baja pero consigue una tasa de detección de imágenes por segundo casi 5 veces mas rápida. Por último, el comportamiento de la versión ligera es peor cuanto mayor sea el número de clases diferentes a detectar (la versión normal puede detectar hasta 9000 clases distintas). No obstante, como se justificará en el desarrollo, dado que no necesitaremos detectar una gran cantidad de clases se utilizará la versión Tiny para la tarea de detección dado que ofrece una mejora significativa en el rendimiento.

En la figura 2.17 se puede observar una gráfica comparando los distintos modelos de YOLO con los sistemas de detección mas populares y potentes de la actualidad, reflejando los primeros por debajo del punto de origen x para denotar su diferencia de velocidad.

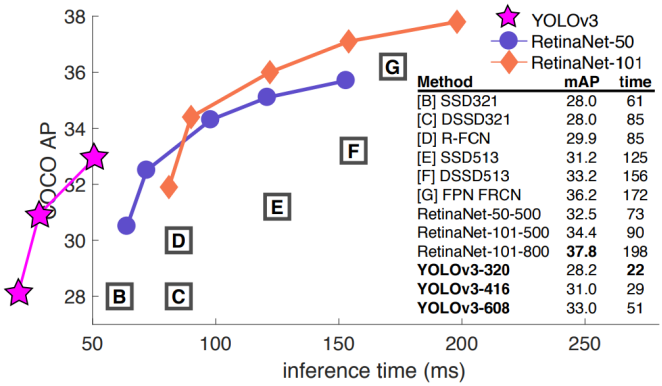


Figura 2.17: YOLO Time Comparision [29]

2.5. Entorno de Trabajo

A continuación se expondrán las diferentes herramientas utilizadas durante el desarrollo del proyecto, como pueden ser APIs o frameworks y su justificación de uso.

2.5.1. Tarea de tratamiento de datos

Esta primera etapa del desarrollo se concentra en recopilar conjuntos de datos de los elementos viales que se persiguen detectar, como señales de tráfico verticales o semáforos. En futuras secciones se especificarán cuales han sido estos datos y porque se han elegido.

No obstante, en lo que a la sección de entorno de trabajo concierne, no se han utilizado APIs externas para el tratamiento de datos. Por el contrario, se ha desarrollado Datasets2Darknet [11], un programa modular escrito en Python, al ser muy cómodo y eficiente para el manejo de imágenes. Este permite combinar varios conjuntos de datos y convertirlos a un formato entendible para YOLO, como se explicará en el desarrollo.

2.5.2. Tarea de Detección

La tarea de detección consiste en la creación de un modelo destinado a la localización y clasificación de señales de tráfico verticales y de semáforos. Para llevarla a cabo se requerirá entrenar un sistema de detección con conjuntos de datos de los objetos que queramos detectar. Como se ha expuesto anteriormente, YOLO se trata del estado del arte en la detección de objetos a tiempo real, por lo que se utilizará para esta tarea.

El sistema de detección YOLO forma parte del conjunto de módulos que ofrece el framework Darknet, un entorno de código libre bastante rápido que soporta computación por CPU y GPU. Para poder hacer uso de este, se ofrece el código en un repositorio de GitHub junto a una guia de utilización. No obstante, en este proyecto se ha decidido utilizar un fork del proyecto principal creado por AlexeyAB [1] al encontrarse mas actualizado, contener código mejor optimizado y ofrecer un mayor número de funcionalidades.

Respecto a la manera de utilización de Darknet, al encontrarse escrito en C y CUDA, su uso por defecto es mediante la línea de comandos. No obstante, resultaría mas cómodo poder utilizar YOLO a través desde una API desde el programa que se desarrolle durante el proyecto, por lo que se realizó una búsqueda de las extensiones que existían para Yolov3. Entre ellas destacaron:

- **TensorFlow Yolov3 [25] -** Programa capaz de convertir los ficheros que resulten del entrenamiento de Darknet en formato entendible para TensorFlow.
- **API en C [1] -** Existen ejemplos en el fork de AlexeyAB que utilizan una API de Darknet escrita en C desde programas Python, de una manera muy cómoda.

- **API en C++ [1] -** Al igual que en el anterior, existen ejemplos escritos en C++ utilizando una API de Darknet escrita en C++.

De todas las opciones disponibles, las dos últimas resultaron las más interesantes para este proyecto dado que son las que podemos encontrar mas actualizadas, lo que resulta en una mayor estabilidad y eficiencia. Tras realizar múltiples pruebas entre estas dos APIs, a pesar de que la API utilizada en Python era muy cómoda, se decidió elegir la API en C++ para este proyecto, esencialmente por su rendimiento superior en tasa de detección (10-15 fps).

2.5.3. Tarea de Clasificación

La tarea de clasificación consistirá en crear un segundo modelo que será usado para la clasificación de señales de tráfico de limite de velocidad. Se deberá desarollar un sistema que, dada una imagen donde aparezca una señal de este tipo, sea capaz de clasificarla según la máxima velocidad que indica.

Dado que el programa a desarrollar estará escrito en C++ (para poder utilizar la API expuesta en el punto anterior), se buscará principalmente un sistema de clasificación escrito en C++, frente a desarrollar uno en TensorFlow y utilizar un programa externo.

En ésta búsqueda se encontró otro módulo de Darknet que ofrece clasificación de imágenes, también escrito en C y CUDA. Para utilizar el mismo, solo se podría interactuar mediante línea de comandos o a través de la API de Python, dado que en la API de C++ no está soportado el módulo de clasificación por el momento. A pesar de que se intentó extender esta para soportar clasificación, debido a su escasa documentación y las grandes dimensiones del código, los resultados obtenidos no fueron satisfactorios.

La solución fue realizar un programa en Python que fuese capaz de inicializar el modelo, recibir rutas de imágenes por línea de comandos, clasificarlas y devolver la clase a la que pertenecían junto a su probabilidad. Una vez desarrollado este, se utilizará desde la aplicación de C++ mediante línea de comandos. Como se verá posteriormente, los resultados fueron positivos.

2.5.4. Tareas de interfaz gráfica de usuario

Finalmente, ésta tarea consistiría en el desarrollo de una interfaz gráfica para mostrar al usuario las detecciones de SaferAuto de una manera visualmente atractiva. Por defecto, Darknet ofrece una visualización simple de las detecciones de YOLO si se activa OpenCV, no obstante, se busca crear una interfaz mas estructurada y consistente, por lo que se optó por utilizar Qt para C++.

Qt se trata de una herramienta multiplataforma que permite desarrollar aplicaciones de escritorio, móviles o web utilizando C++ [4]. Como ventajas destacan su facilidad de uso, la documentación de la herramienta y la rapidez de la misma. Además, ofrece una versión comunidad de código libre, por lo que además de seguro, el uso será gratuito.

2.5.5. Hardware y Software

El hardware utilizado para el desarrollo de las tareas expuestas es el siguiente:

- Procesador Ryzen 7 1700.
- Tarjeta gráfica Nvidia GTX 1060 de 6GB. Importante que contenga núcleos CUDA para utilizar la aceleración por GPU de Darknet.
- RAM DDR3 de 16GB a 3200mhz.

En cuanto al software, se decidió usar únicamente código libre:

- Sistema Operativo: Ubuntu en su versión 18.04.
- IDE para Python y Darknet: Visual Studio Code.
- IDE para QT: QT Creator versión community.

Capítulo 3

Desarrollo del proyecto

3.1. Tratamiento de datos

Para poder entrenar cualquier red neuronal, lo primero que de lo que se precisa es de datos. En este apartado se presentarán los conjuntos de datos que se han recopilado junto a su justificación de uso. Además, dado que se utilizará YOLO tanto en la tarea de detección como de clasificación, necesitaremos convertir las anotaciones de los conjuntos de datos obtenidos en un formato compatible.

Cabe destacar que los conjuntos que se utilicen para ambas tareas serán distintos, al tratase de tareas con diferentes objetivos. En la tarea de detección se necesitarán imágenes de escenas completas con los objetos que queramos detectar, correctamente anotados a través de cuadros delimitadores y la clase a la que pertenezcan (figura 3.1a). Por otro lado, en la tarea de clasificación únicamente se necesitará el contenido de estos cuadros delimitadores (figura 3.2.3), es decir, imágenes de las distintas señales que se quieran clasificar.

(a) Imagen para tarea de detección.

(b) Imagen para tarea de clasificación.

Es importante tener en cuenta esta diferencia pues no se podría entrenar el sistema de detección con conjuntos de datos adaptados a la tarea de clasificación, por ejemplo.

3.2. Origen de los datos

Otro factor a tener en cuenta es el origen de los conjuntos de datos que se van a utilizar. Dependiendo del país en el que nos encontremos, las señales de tráfico pueden variar, por lo que se debe fijar la región en la que se desea que el programa detecte las señales de tráfico.

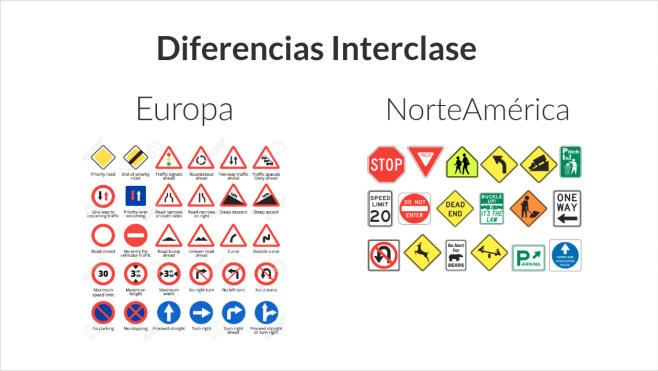


Figura 3.2: Diferencias Interclase.

Dado que el objetivo de este proyecto es que pueda ser utilizado en Europa, se recopilarán conjuntos de datos provenientes de países de esta región. Para ello, es importante saber de la existencia de la Convención de Viena sobre Señales de Carretera, cuyo objetivo es estandarizar el sistema de señalización vial (señales, semáforos y marcas viales) con el fin de aumentar la seguridad. En la actualidad, la mayor parte de la Unión Europea ha adoptado este estándar por lo que se han establecido tamaños, formas y colores comunes.

A pesar de este convenio, cada país puede elegir sus propios símbolos e inscripciones, generando diferencia intraclase. Un ejemplo se puede observar en la ilustración 3.3, donde se comparan señales de Croacia, Francia y Alemania. A pesar de que la diferencia no es significativa para rechazar conjuntos de datos de alguno de éstos países, es importante tener en cuenta que pueden aparecer símbolos diferentes (como el de paso de peatones).

Figura 3.3: Diferencias Intraclase.

3.2.1. Conjuntos de datos para sistema de detección

En este apartado se presentarán los dataset que se encontraron a nivel Europeo con anotaciones de señales de tráfico o de semáforos. Cabe añadir que durante el entrenamiento se hicieron pruebas con diferentes combinaciones de los conjuntos de datos que se indicarán a continuación y algunos de ellos fueron descartados para conseguir una mejor precisión en el sistema.

No obstante, para poder indicar las razones de estas decisiones durante el entrenamiento, se precisará de exponer las especificaciones de cada conjunto de datos con el que se realizaron pruebas. También resultará de utilidad a los investigadores en el campo que se encuentren en la búsqueda de conjuntos de datos con anotaciones de elementos viales.

GTSDB - German Traffic Sign Detection Benchmark

El conjunto de datos alemán de señales de tráfico se trata de uno de los datasets mas populares del campo en la actualidad [10]. Fue introducido en la Conferencia Internacional Conjunta sobre Redes Neuronales de 2013 con el objetivo de actuar como evaluación para los investigadores interesados en el campo de Visión por Computador y asistencia a la conducción basada en imágenes. Sus características principales son:

Característica	Valor
Formato de imágenes	*PPM*
Formato de anotación	*filename;left_x;bottom_y;right_x;top_y;class_id*
Tamaño señales	*16x16 a 128x128*
Clases	*43*
Imágenes de entrenamiento	*600*
Imágenes de validación	*300*
Total Imágenes	*900*

Cuadro 3.1: Características GTSDB

BTSD - Belgium Traffic Sign Dataset

Este conjunto de datos tiene origen en Bélgica y se encuentra abierto gracias a la universidad ETH Zurich [23]. A pesar de que no existe demasiada información sobre el dataset, se han encontrado las siguientes características:

Característica	Valor
Formato de imágenes	*JPG*
Formato de anotación	*folder/filename;left_x;bottom_y;right_x;top_y;class_id;superclass_id*
Clases	*62*
Imágenes de entrenamiento	*5.905*
Imágenes de validación	*3.101*
Imágenes de fondo	*16.628*
Total Imágenes	*9.006 + 16.628*

Cuadro 3.2: Características BTSD

MASTIF - Mapping and Assessing the State of Traffic InFrastructure

MASTIF es el resultado de un proyecto de investigación en el que se recolectaron múltiples conjuntos de datos de señales de tráfico. Consiste en la unión de 3 datasets: TS2009, TS2010 y TS2011, correspondiendo al año en el que fueron anotados [6]. En la página oficial solo se encuentran disponibles los conjuntos de datos TS2010 y TS2011. No obstante, tras contactar con Siniša Šegvić (colaboradora de MASTIF) y explicarle el objetivo de SaferAuto, se pudieron obtener los datos TS2009.

La características principales de estos datasets combinados son:

Característica	Valor
Formato de imágenes	*BMP - 720x576*
Formato de anotación	*[filename]:class_id@(left_x,bottom_y,width, height)*
Clases por categoría	*A(50) + B(62) + C(133) + D(17) + E(50) = 312*
Total Imágenes	*TS2009(6.000) + TS2010(3.000) + TS2011(1.000) = 10.000*

Cuadro 3.3: Características MASTIF

RTSD - Russian traffic sign images dataset

Este conjunto de datos ruso destaca por su colosal tamaño de 104.358 imágenes de señales verticales, superando en 10 la cantidad de datos de MASTIF, el mayor de los dataset anteriormente expuestos. Es de acceso público y está destinado al entrenamiento y validación de algoritmos de reconocimiento de señales de tráfico [32]. Cabe añadir que

a pesar de que Rusia no pertenece a la Unión Europea, sigue la convención de Viena de señales, por lo que las señales son compatibles con las que se encuentran en la Unión Europea.

Sus características principales son:

Característica	**Valor**
Formato de imágenes	*JPG - 1280x720*
Formato de anotación	*filename,left_x,bottom_y,width,height,class_id,sign_id*
Clases	*156 divididas en 8 superclases*
Total Anotaciones	*179.138*
Total Imágenes	*104.358*

Cuadro 3.4: Características RTSD

LISATS - LISA Traffic Sign Dataset

LISATS es un conjunto de vídeos e imágenes anotadas con señales de tráfico de Estados Unidos. Como se explicó en las diferencias interclase entre las señales de Europa y NorteAmérica (figura 3.2), estos elementos varían en casi todos los tipos menos en las señales de ceda el paso y stop.

Debido a la falta de anotaciones de la señal de stop en los anteriores dataset (el mayor solo cuenta con 552) se ha decidido hacer uso únicamente de las señales de stop de LISATS [24], dado que posee 1821 anotaciones, aumentando significativamente la cantidad de imágenes de este tipo.

LISATL - LISA Traffic Light Dataset

En los anteriores conjuntos de datos, solo se han encontrado anotaciones de los distintos tipos de señales de tráfico. No obstante, además de la detección de estas señales, SaferAuto tiene como objetivo localizar y clasificar los distintos tipos de semáforos.

El laboratorio para vehículos seguros e inteligentes LISA (el mismo que publicó el dataset anterior) ofrece gran cantidad de anotaciones de semáforos de Estados Unidos. A pesar de algunas diferencias con los tipos de semáforos en Europa, se utilizará LISATL como fuente principal para la detección de semáforos [24]. Sus características son:

Característica	Valor
Formato de imágenes	*PNG*
Formato de anotación	*filename;left_x;bottom_y;right_x;top_y;class_id*
Clases	*6*
Total Imágenes	*Día(40.000) + Noche(14.000) = 54.000*

Cuadro 3.5: Características LISATL

3.2.2. Tratamiento de datos para sistema de detección

Como se ha podido observar en las tablas de los conjuntos de datos que se utilizarán para el sistema de detección, existen múltiples diferencias entre estos dataset. Las principales son el formato de las imágenes, el formato de las anotaciones y el número de clases e imágenes de cada uno.

Dataset	Formato Imágenes	Formato de anotación	Clases	Total Imágenes
GTSDB	*PPM*	*fn;left_x;bottom_y;right_x;top_y;class*	*43*	*900*
BTSD	*JPG*	*fn;left_x;bottom_y;right_x;top_y;class;sup*	*62*	*9.006*
MASTIF	*BMP*	*[fn]:class@(left_x,bottom_y,width,height)*	*312*	*10.000*
RTSD	*JPG*	*fn,left_x,bottom_y,right_x,top_y,class,sign*	*156*	*104.358*
LISATL	*PNG*	*fn;left_x;bottom_y;right_x;top_y;class*	*6*	*54.000*

Cuadro 3.6: Características Datasets Combinados

Sin embargo, se requiere utilizar un conjunto de datos unificado dónde todas las imágenes se encuentren en el mismo formato, las anotaciones sean compatibles con YOLO y las tipos de clases iguales tengan el mismo identificador. Para ello, se ha desarrollado un programa propio en Python de código abierto llamado Datasets2Darknet [11].

Previo a desarrollar el mismo, se debió resolver el problema de los distintos identificadores entre los datasets para los mismos tipos de clase. Es decir, mientras que en una clase la señal de stop corresponde con el identificador 7 (BTSD), en otra puede corresponder con un identificador en texto "stop"(LISATS) o "B02"(MASTIF). A su vez, mientras que en algunos conjuntos de datos existen 50 señales para la categoría de prohibición, otros no superan la mitad de clases.

Por ello, para poder utilizar varios datasets de entrenamiento, se deben estudiar las clases que existen en todos los datasets y agrupar las señales en categorías atendiendo a su forma y color. De esta manera aunque en la misma categoría de distintos datasets no existan las mismas señales, nos aseguramos de que las categorías existan en todos, a base de indicar al programa cuales son las señales que pertenecen a cada superclase.

A su vez, esto permite recopilar solo un tipo específico de categoría de un dataset y evita incorporar imágenes en las que aparezcan otros tipos (como es el caso de LISATS de donde solo se necesita la categoría de stop). Cabe añadir que al elegir categorías con el mismo color y forma (figura 3.4), aumentará en gran medida la precisión del clasificador, al existir una mayor diferencia entre los objetos a reconocer.

(a) Señales de Tráfico Vérticales.

(b) Semáforos.

Figura 3.4: Categorías para detección de SaferAuto.

En esencia, esto significa que SaferAuto será capaz de detectar las distintas categorías de elementos, pero no las clases específicas a las que pertenecen. Si se quisiera averiguar a que subclase pertenece una señal dentro de una categoría, se deberían clasificar las diferentes imágenes que existen para esa categoría, como es el caso de las señales de límite de velocidad, explicado en la siguiente sección.

Datasets2Darknet

El objetivo de este programa es unificar varios conjuntos de datos, leyendo los datos de entrada de distintas localizaciones y convirtiendo las anotaciones e imágenes a un formato común, compatible para el sistema de detección YOLO.

Está compuesto por tres componentes principales:

- **Parser general (general_parser.py):** Fichero principal del programa. Se encarga de importar todos los parsers de los datasets específicos e iterar sobre ellos llamando a *read_dataset*, método que todos los parsers que se creen deben tener. Finalmente muestra el número total de imágenes por cada clase en el conjunto de datos unificado.
- **Configuración común (common_config.py):** Clase ayudante que contiene todos los métodos comúnes para los parsers de los datasets específicos. En ella se encuentran, por ejemplo, utilidades para leer, redimensionar o escribir una imagen. A su vez contiene algunas constantes comúnes como:
  - *SHOW_IMG:* Modo verbose para mostrar las imágenes de salida con sus anotaciones.

- *OUTPUT_IMG_EXTENSION:* Extensión de las imágenes de salida.
- *COLOR MODE:* Formato de color de las imágenes de salida.
- *TRAIN_PROB, TEST_PROB:* Porcentaje de entrenamiento y validación que se cogerán de las imágenes de entrada totales. En vez de utilizar la división de train/test que ofrecen la mayoría de datasets, se situarán los datos en un conjunto total y se dividirá de manera personalizada ese conjunto, con el fin de que se utilice la misma proporción en todos los conjuntos de datos.

- **Parser específico (dataset_parsers/specific_parser.py):** Los parsers de cada dataset del que queramos extraer información y unificarla en el *general_parser.py* deberán estar situados en la carpeta *dataset_parsers*. Este debe de poseer una serie de métodos para que se pueda utilizar:

  - *initialize_traffic_sign_classes:* Este método crea las relaciones entre la clase de un objeto en su dataset específico y la clase real que se utiliza en el programa. Por ejemplo, si la clase del "stop" en un dataset es "B02", necesitaremos crear una relación tal que:

```
def initialize_traffic_sign_classes():
    traffic_sign_classes["5-stop"] = ["B02"]
```

  - *calculate_darknet_format:* Este método se encarga de convertir la anotación específica de un dataset al formato compatible con Darknet. Para ello, se especifica en que orden se encuentran los datos dentro de la variable *data* y se pasan de manera ordenada al método común *parse_darknet_format* para que los convierta al formato de Darknet.

```
def calculate_darknet_format(img, width, height, data):
    img_width, img_height = get_img_dim_plt(input_img)
    width_proportion = (img_width / MAX_WIDTH)
    height_proportion = (img_height / MAX_HEIGHT)

    left_x = float(data[1]) / width_proportion
    bottom_y = float(data[2]) / height_proportion
    right_x = float(data[3]) / width_proportion
    top_y = float(data[4]) / height_proportion

    adjusted_class = adjust_object_class(data[6])

    return parse_darknet_format(adjusted_class,
                                width, height,
                                left_x, bottom_y,
                                right_x, top_y)
```

  - *read_dataset:* Por último, quedaría el método principal. Este contiene las rutas especificas del dataset, como la localización de las anotaciones o imágenes. Se

encarga de leer estas anotaciones, convertirlas al formato de Darknet haciendo uso del método anterior y escribirlas en las rutas de salida que recibe como parámetros. A su vez, convierte los formatos de las imágenes de entrada a un formato común definido en la variable *OUTPUT_IMG_EXTENSION* y los sitúa en la ruta de imágenes de salida.

Cabe resaltar que se utilizará el programa expuesto Datasets2Darknet para preparar los datos para el entrenamiento y validación de los sistemas tanto de detección como de clasificación. Este facilitará el proceso de eliminar o añadir datasets a nuestro conjunto de datos unificado.

3.2.3. Conjuntos de datos para sistema de clasificación

En este apartado se describirán los datos utilizados para entrenar y validar el sistema de clasificación. Como se había expuesto en el inicio de esta sección, no se podrán utilizar los mismos datos que en el sistema de detección, si no que se buscarán imágenes en las que aparezca únicamente una señal, para poder clasificarla atendiendo a su categoría.

Respecto a las clases que se perseguirán clasificar en SaferAuto son las señales de límite de velocidad. Se desea desarrollar un sistema que reciba una imagen perteneciente a la categoría de prohibición (figura 3.4a) y devuelva la clase a la que pertenece la señal. Únicamente se entrenará el sistema de clasificación con señales de límite de velocidad, por lo que si el sistema recibe una señal de prohibición que no sea de esta clase, se deberá devolver que no existe una subclase o que si pero con una probabilidad muy baja, que discriminaríamos utilizando un umbral de clasificación.

A la hora de realizar la búsqueda de los conjuntos de datos de clasificación, se encontró un inconveniente en múltiples datasets: las distintas clases de señales de límite de velocidad (40, 50, 60...) tenían el mismo identificador de clase. Estos debieron ser descartados y solo permanecieron los que tenían esta clase subdivida, como se expone a continuación.

GTSRB - German Traffic Sign Recognition Benchmark

GTSRB se trata del equivalente al GTSDB en la tarea de clasificación, siendo uno de los conjuntos de datos europeos mas populares del campo [10]. Fue introducido en la Conferencia Internacional Conjunta sobre Redes Neuronales de 2011 como parte de un reto para clasificación de imágenes.

Sus características principales son:

Característica	Valor
Número de Clases	*40*
Clases de Límite de Velocidad	*20, 30, 50, 60, 70, 80, 100, 120*
Número de Imágenes Límite de Velocidad	*13.200*

Cuadro 3.7: Características GTSRB

rMASTIF Traffic Sign Classification Dataset

Este conjuntos de datos proviene del dataset MASTIF [6], expuesto en los datos para detección. Ha sido adaptado para la tarea de clasificación (recortando las imágenes según sus cuadros delimitadores). Es de acceso abierto y sus características son:

Característica	Valor
Número de Clases	*31*
Clases de Límite de Velocidad	*30, 40, 50, 60, 70*
Número de Imágenes Límite de Velocidad	*415*

Cuadro 3.8: Características rMASTIF

RTSD - Russian traffic sign images dataset

El dataset RTSD [32] expuesto en la anterior sección contiene un subconjunto de datos para la tarea de clasificación. Esta formado por gran cantidad de las imágenes utilizadas para detección, pero adaptadas para entrenar un sistema de clasificación. Sus características relevantes para esta tarea son:

Característica	Valor
Número de Clases	*156*
Clases de Límite de Velocidad	*10, 20, 30, 40, 50, 60, 70, 80*
Número de Imágenes Límite de Velocidad	*7.704*

Cuadro 3.9: Características RTSD-C

3.2.4. Tratamiento de datos para sistema de clasificación

Para el tratamiento de los datos del sistema de clasificación se usó Datasets2Darknet [11], al igual que en la tarea de detección. Para seleccionar los datasets a unificar, se

estudió la combinación que mas cantidad de tipos de señales de límite de velocidad diferentes pudiese clasificar.

Como se puede observar en la tabla 3.7 en el conjunto de datos alemán no hay señal de límite de velocidad de 40, lo que es bastante común en países de la unión Europea como España. Por otra parte, el dataset ruso contiene clases de 10, 20 y 40, no ofrecidas en el GTSRB. Se decidió que un 80 % de la unión de los conjuntos de datos GTSRB y RTSD-Clasificación es la que se utilizaría para entrenar el sistema de detección y el 20 % restante junto al dataset rMASTIF (para datos nuevos) se usaría para la validación.

Con ese dataset unificado, dada una señal perteneciente a la cateogría de prohibición 3.4a, SaferAuto sería capaz de clasificarlas en una de las subclases que se muestran en la figura 3.5.

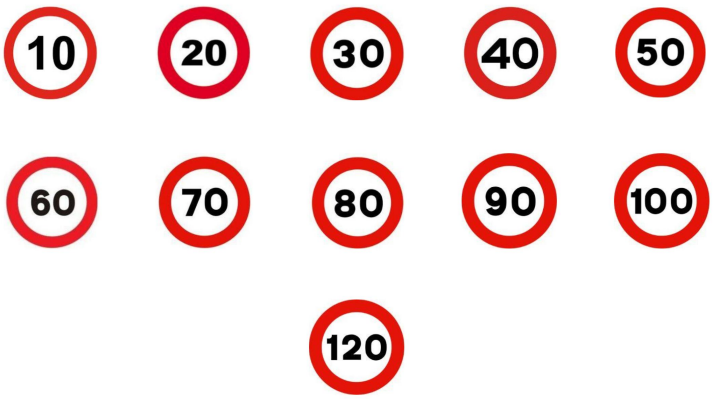


Figura 3.5: Señales clasificadas por SaferAuto.

3.3. Sistema de detección

En esta sección se presentarán las distintas combinaciones de datasets que se han utilizado para el entrenamiento del sistema de detección y cuales han sido los resultados de validación. Como se ha explicado anteriormente, la tarea de detección consiste en localizar los elementos víales dentro de la imágen de entrada y clasificarlos según el tipo de objeto.

Para poder validar el sistema a desarrollar, se debe dividir el conjunto de datos de entrada en un subconjunto para el entrenamiento y otro para la validación. La proporción de división que se ha utilizado es el 80 % de las imágenes de entrada como datos para entrenamiento y el 20 % restante para validación. Esto se ha hecho en cada conjunto de datos por igual, para que la proporción train-test sea equivalente en todos los conjuntos. Cabe destacar que se han dividido las imágenes de manera aleatoria con las probabilidades comentadas haciendo uso de Datasets2Darknet [11].

3.3.1. Modificaciones en Datos

En este apartado se presentarán las modificaciones que se plantearon realizar sobre los datos de entrada y su justificación de llevarlas o no a cabo.

Una de las técnicas utilizadas para reducir el overfitting se conoce como *Data Augmentation* y consiste en hacer una copia de las imágenes de entrada, modificar ciertas de sus características como el tamaño o la rotación de manera aleatoria y añadir las copias modificadas como nuevas imágenes al conjunto de entrada. Esto tiene como ventajas un aumento en las imágenes del conjunto de entrada que puede ser bastante favorable en caso de datasets con pocos datos. También reduce el overfitting, dado que al realizar modificaciones de manera aleatoria es mas difícil que se sobreajusten los pesos a las imágenes de entrada.

YOLO utiliza varios algoritmos de Data Augmentation en cada iteración de su entrenamiento, modificando características de las imágenes de entrada como el tamaño, traslación, exposición, saturación, brillo o rotación. Para poder configurarlo se debe especificar el rango máximo en el que se realizarán modificaciones de cierta característica. Por ejemplo para controlar la modificación de la exposición de la imágen se podría especificar *exposure = 1.5*, lo que indicaría que el algoritmo aumentará o disminuirá aleatoriamente la exposición de la imagen en el rango [-1.5, 1.5]. Se explica con mas detalle en el paper de Yolov1 [28].

Dado que YOLO aplica técnicas para el aumento de datos de las imágenes de entrada, se decidió no hacer modificaciones externas a este conjunto. Los parámetros de Data Augmentation utilizados fueron:

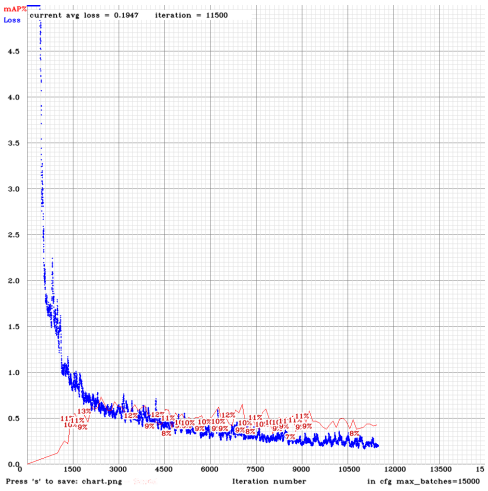
- *angle=7*: Modifica el ángulo de la imagen de manera aleatoria entre -7 y 7 grados.
- *saturation=1*: Modifica la saturación de la imagen.
- *exposure=1.* Modifica la exposición de la imagen.
- *hue=.1*: Modifica el tono de la imagen en un rango 0.1
- *flip=1*: Permite voltear las imágenes en el eje horizontal. Este parámetro es de gran utilidad en el conjunto de señales verticales porque si una señal solo apareciera en el lado derecho en las imágenes de entrenamiento y en la validación la misma señal se encuentra en el lado izquierdo, no se detectaría, por lo que es recomendable entrenarlas desde ambos lados.

En cuanto al tamaño de las imágenes, se hicieron pruebas redimensionando todas las imágenes de entrada a un mismo tamaño, concretamente de 608x608 píxeles. No obstante, tras el estudio del funcionamiento de YOLO se descubrió que la herramienta redimensiona automáticamente las imágenes al tamaño de la red y mantiene la relación de aspecto, por lo que se descartó la opción de convertir las imágenes a una forma cuadrada, pues se estarían utilizando datos no reales. Sin embargo, para acelerar el proceso de entrenamiento y validación, las imágenes se redujeron de tamaño manteniendo la relación de aspecto para que fueran mas rápidas de leer y tratar.

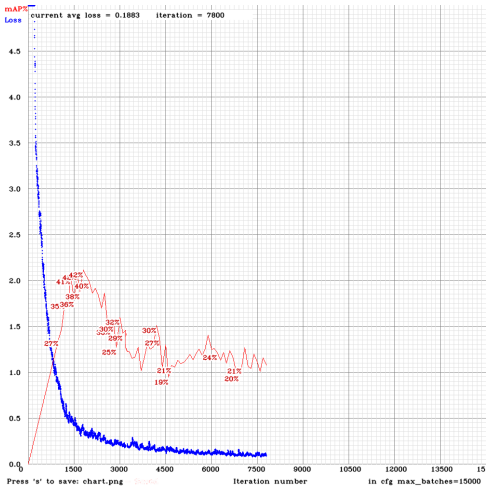
3.3.2. Entrenamiento y Validación

El primer conjunto que se entrenó fue el GTSDB (sección 3.2.1). La arquitectura de YOLO utilizada fue YOLOv3-spp, la cual alcanzaba 20 fps y un mAP de 60.6 en el dataset COCO [5]. Al utilizar solo un dataset no había que agrupar las clases por sus características, por lo que el primer intento fue el de detectar las 43 clases de GTSDB, el resultado del entrenamiento se muestra en la gráfica 3.6b. Se puede observar que los resultados fueron bastante negativos, dado que no consiguió superar un 14 % de mAP.

El segundo intento consistió en agrupar las 43 clases del GTSDB en 5 categorías de señales de tráfico (prohibición, peligro, obligación, stop y ceda el paso) con el que se consiguió alcanzar un mAP de 42 %, una diferencia significativa respecto al entrenamiento anterior.



(a) Yolov3-spp - GTSDB - Entrenamiento de 43 clases.



(b) Yolov3-spp - GTSDB - Entrenamiento de 5 clases.

La principal limitación del conjunto de datos alemán fue su cantidad reducida de datos, lo que provocó no alcanzar un mAP aceptable. Por ello, se decidió probar con otro dataset, el BTSD (sección 3.2.1). Se repitió la misma agrupación de 5 clases que en el conjunto alemán y esta vez se consiguió alcanzar un 86 % de mAP antes de que la red comenzara a sobreajustarse.

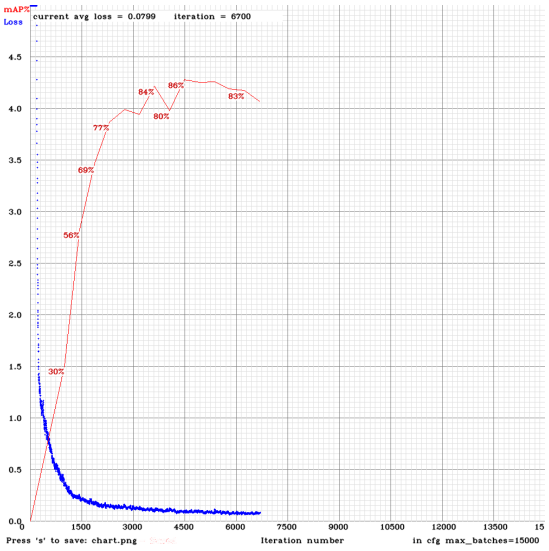


Figura 3.7: Yolov3-spp - BTSD - Entrenamiento de 5 clases.

Esto supuso una diferencia significativa de un 44 % frente al GTSDB. Se comprueba que cuantos mas datos mejor funciona YOLO. De hecho, en el fork de Alexey [1] se recomienda tener un mínimo de 2.000 imágenes de entrenamiento para cada clase. No obstante, en la tabla 3.10 se puede apreciar que la distribución de imágenes de BTSD no alcanza ese requisito. Además, las señales de stop y ceda el paso tienen un número de datos demasiado reducido para alcanzar una buena precisión.

Clase	*Prohibición*	*Peligro*	*Obligación*	*Stop*	*Ceda el Paso*
Numero de Imágenes	*2122*	*1345*	*1596*	*88*	*436*

Cuadro 3.10: BTSD - Imágenes/Clase

Para solucionar el problema del reducido número de señales en las categorías de stop y ceda el paso, se probó a unificar los conjuntos anteriores GTSDB y BTSD junto al conjunto de datos americano LISATS (sección 3.2.1), resultando en un dataset unificado con la distribución imágenes/clases que se ilustra en la tabla 3.11.

Clase	*Prohibición*	*Peligro*	*Obligación*	*Stop*	*Ceda el Paso*
Numero de Imágenes	*2708*	*1564*	*1759*	*1909*	*672*

Cuadro 3.11: GTSDB + BTSD + LISATS - Imágenes/Clase

El resultado del entrenamiento se puede observar en la figura 3.8. A pesar de que el mAP disminuyó de 86 % a 83 %, los AP de las clases stop y ceda el paso aumentaron, por lo que el modelo se encontraba mas equilibrado. Cabe añadir que la tasa de detección de imágenes por segundo era de 30 fps con esta combinación.

La razón del descenso del mAP podría ser la de utilizar un conjunto de datos dónde las únicas señales anotadas sean la de stop y ceda el paso, dado que la red puede confundirse al poseer las señales diferentes formas que en los conjuntos de datos europeos.

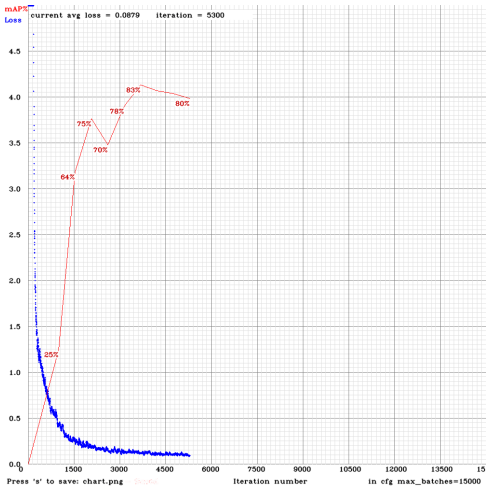


Figura 3.8: Yolov3-spp - GTSDB + BTSD + LISATS - Entrenamiento de 5 clases.

A pesar del aumento en el número de imágenes por clase, se realizó una búsqueda de un dataset mas completo y esta resultó en la incorporación del conjunto MASTIF (sección 3.2.1) a nuestro conjunto de datos unificado. Como se puede observar en la tabla 3.12, los datos ascendieron a una media de 1500 imágenes por clase, lo que permitió alcanzar un mAP de un 89 % (figura 3.9), el máximo hasta ahora.

Clase	*Prohibición*	*Peligro*	*Obligación*	*Stop*	*Ceda el Paso*
Numero de Imágenes	*4839*	*6001*	*2274*	*2548*	*878*

Cuadro 3.12: GTSDB + BTSD + LISATS + MASTIF - Imágenes/Clase

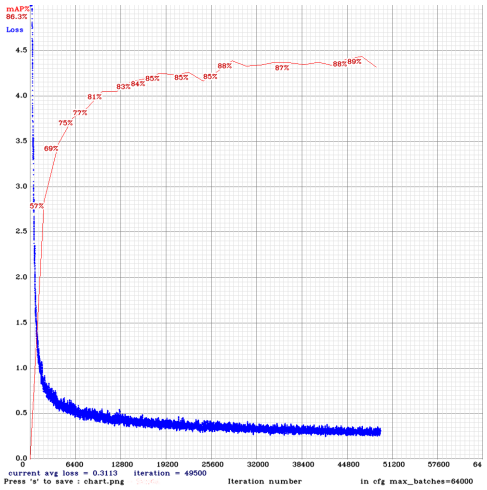


Figura 3.9: Yolov3-spp - GTSDB + BTSD + LISATS + MASTIF - Entrenamiento de 5 clases.

La siguiente prueba realizada fue la de cambiar la arquitectura del modelo que se utiliza para la detección, haciendo uso de *yolov3-tiny* en vez de *yolov3-spp* [27]. Esta arquitectura es mas ligera por lo que permitirá una tasa de detección mas rápida, pero supuestamente bajara la precisión del sistema comparada con *yolov3-spp*.

No obstante, con el mismo dataset unificado de la última prueba, al utilizar la arquitectura ligera, los resultados fueron muy positivos, como se observa en la figura 3.10. Se consiguió llegar a un **mAP de un 93 %**, el máximo conseguido. Además, la tasa de imágenes detectadas por segundo subió de 30 a casi 120, una diferencia significativa. Las pruebas realizadas a partir de ahora se realizarían con la arquitectura yolov3-tiny.

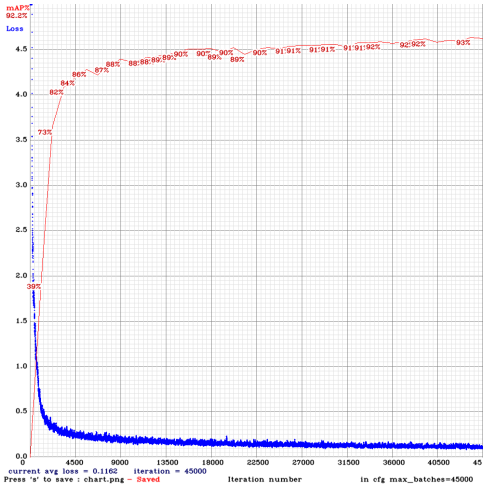


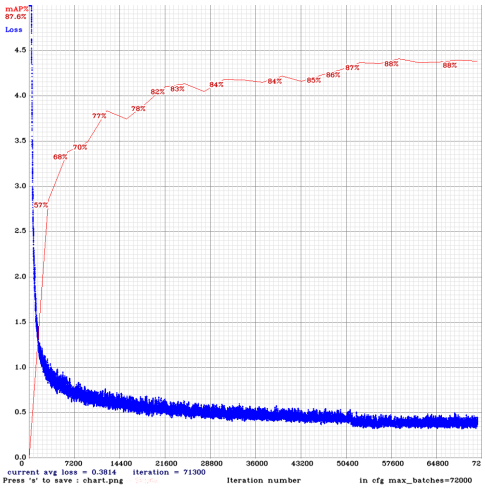
Figura 3.10: Yolov3-tiny - GTSDB + BTSD + LISATS + MASTIF - Entrenamiento de 5 clases.

Como se había comentado en la introducción, el objetivo de SaferAuto, además de detectar señales de tráfico verticales, también es el de detectar y clasificar los distintos

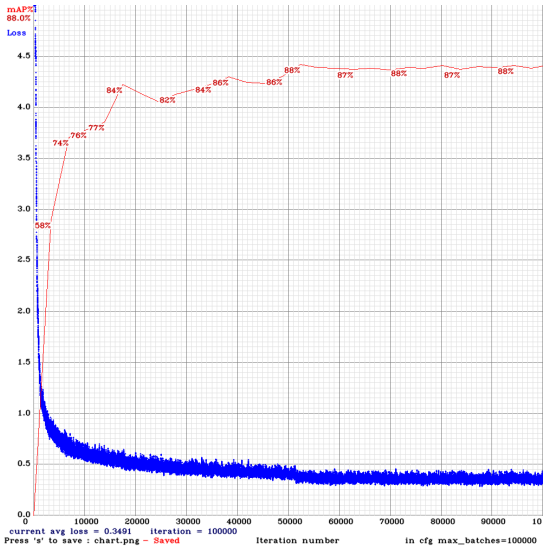
tipos de semáforos. Por ello, se decidió incorporar el dataset LISATL (sección 3.2.1) en nuestro conjunto de datos unificado. Esto supuso que el número de clases pasara de ser 5 a 8.

No obstante, tras encontrar LISATL también se descubrió un nuevo dataset de señales de tráfico verticales que contenía 10 veces mas datos que todos los anteriores, el conjunto de datos ruso RTSD (sección 3.2.1). Con este, se podría intentar prescindir de los anteriores datasets de señales verticales y nos permitiría generar nuevas subdivisiones de clases, dado que las anotaciones estaban dividas en 8 superclases. Al unificar LISATL y RTSD, se decidió un nuevo conjunto de clases a detectar, formado por 10 superclases (figura 3.4): prohibición, peligro, obligación, información, stop, ceda el paso, no entrar, semáforo verde, semáforo ámbar, semáforo rojo...

La primera prueba se realizó uniendo RTSD, LISATL y LISATS (para el aumento de datos de stop y ceda el paso). En la segunda, se incorporó también el dataset croata MASTIF. Los resultados fueron bastante similares, ambos llegando a un 88 % de mAP. Sin embargo, se eligió la segunda configuración dado que había sido entrenado para reconocer señales con una mayor diversidad de imágenes. A pesar de la bajada de mAP respecto a la anterior prueba de yolov3-tiny (figura 3.10), este nuevo modelo es capaz de detectar 10 clases distintas, frente a las 5 del anterior.



(a) Yolov3-tiny - RTSD + LISATS + LISATL - Entrenamiento de 10 clases.



(b) Yolov3-tiny - RTSD + MASTIF + LISATS + LISATL- Entrenamiento de 10 clases.

3.3.3. Configuración Final

Para concluir, en este apartado se presentarán los datos que se han utilizado para el entrenamiento y validación del sistema de detección de SaferAuto y sus correspondientes estadísticos (definidos en la sección 2.2.5) por clase. Como conjunto de validación se utilizó un 20 % del conjunto de entrada que no se utilizó en el entrenamiento.

El mAP del modelo con mejores pesos obtuvo un porcentaje del 88.51 %. No obstante, como se aprecia en la tabla 3.13 existen clases con una tasa de aciertos de mas de un 95 %. A su vez, la tasa de imágenes por segundo que el sistema de detección es capaz de detectar es de 220 fps.

Datasets	**Clase**	**AP**	**TP**	**FP**
RTSD + MASTIF + LISATS + LISATL	*Prohibición*	*90.18 %*	*2660*	*447*
	Peligro	*94.21 %*	*3143*	*329*
	Obligación	*92.95 %*	*1487*	*227*
	Información	*91.35 %*	*9411*	*1794*
	Stop	*90.65 %*	*524*	*50*
	Ceda el Paso	*84.11 %*	*792*	*148*
	No entrar	*83.74 %*	*154*	*29*
	Semáforo Rojo	*96.13 %*	*2369*	*126*
	Semáforo Ámbar	*68.88 %*	*65*	*22*
	Semáforo Verde	*92.93 %*	*1939*	*233*

Cuadro 3.13: Configuración Final Detección.

3.4. Sistema de clasificación

Esta sección tendrá como objetivo explicar el desarrollo de entrenamiento del sistema de clasificación de SaferAuto, presentando los conjuntos de datos que se han utilizado para entrenar y validar y cuales han sido los resultados. Recordemos que la tarea de clasificación consiste en clasificar una imagen según su tipo. Por el momento, se ha decidido solo enfocarse en la clasificación de señales de límite de velocidad, cuya superclase es la de prohibición.

Como se explicó al presentar los datasets que serían utilizados para esta tarea (sección 3.2.3), con el objetivo de conseguir clasificar el máximo número de clases de límite de velocidad distinta, se decidió crear un dataset unificado formado por GTSRB 3.2.3 y RTSD-C 3.2.3. Los datos del conjunto unificado se ilustran en la figura 3.14.

Número de Clases Límite de Velocidad	*10*
Clases Límite de Velocidad	*10, 20, 30, 40, 50, 60, 70, 80, 100, 120*
Número de Imágenes Límite de Velocidad	*20.904*

Cuadro 3.14: GTSRB + RTSD-C - Entrenamiento Sistema de Clasificación

El entrenamiento del módulo de clasificación de Darknet se realiza de manera distinta al módulo de entrenamiento. En la tarea de detección se requerían imágenes con las

anotaciones de los objetos que contengan. Sin embargo, en el sistema de clasificación, al solo aparecer una señal en cada imagen, se requiere nombrar la imagen con su su índice en el conjunto de datos final y su tipo. Por ejemplo, para la imagen número 1000 del dataset con clase "sl_20", se debía renombrar como *"1000_sl_20.jpg"*. Para esto se adaptaron los parsers de Datasets2Darknet [11].

El modelo de YOLO de clasificación que se utilizó fue el llamado *Darknet Reference* [26], al ser el que mejor rendimiento tiene tanto en cantidad de operaciones en punto flotante como en precisión general.

Los resultados del entrenamiento se ilustran en la figura 3.12, con menos de 5000 iteraciones se consiguió alcanzar un valor muy bajo en la función de error. Para poder calcular la precisión de los pesos generados, se desarrolló un programa en Python [12] dado que Darknet no lo tenia implementado. Este utilizaba los datos de validación para generar tanto la precisión por clase como la precisión media del modelo. Cabe añadir que en los datos para validar se incorporó el dataset rMASTIF (sección 3.2.3), de cara a añadir datos completamente nuevos para la red. La precisión general alcanzada fue muy positiva, obteniendo casi en todas las clases mas de un 90 % de precisión.

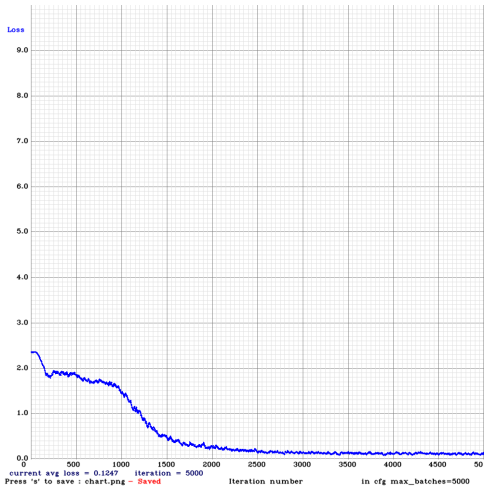


Figura 3.12: Darknet Reference - RTSD-C + MASTIF.

3.4.1. Configuración Final

En este apartado se presentarán los datos utilizados para entrenar y validar el sistema de clasificación, así como su precisión general y precisión por cada tipo de señal de límite de velocidad. Cabe añadir que los datos que se utilizaron para la validación fueron un 20 % de los datos usados para el entrenamiento (seleccionados de manera aleatoria) y el 100 % de los datos contenidos en el dataset rMASTIF.

La precisión media alcanzada fue de 98.23 % y la precisión media por clase se ilustra en la tabla 3.15. A su vez, se consiguió una velocidad de clasificación de imágenes de 90 fps.

Datasets	*Clase*	Precisión	Correctas	Incorrectas
GTSRB + RTSD-C + rMASTIF (Validación)	*10*	*100 %*	*5*	*0*
	20	*98.45 %*	*191*	*3*
	30	*98.57 %*	*344*	*5*
	40	*97.56 %*	*559*	*14*
	50	*98.73 %*	*311*	*4*
	60	*91.73 %*	*233*	*21*
	70	*97.95 %*	*287*	*6*
	80	*100 %*	*202*	*0*
	100	*99.29 %*	*139*	*1*
	120	*100 %*	*140*	*0*

Cuadro 3.15: Configuración Final Clasificación.

3.5. Interfaz gráfica de usuario

Una vez se han creado y elegido los modelos que serán usados para las tareas de detección y de clasificación, se deberá desarrollar una interfaz gráfica que sea capaz de utilizar ambos modelos para detectar señales de tráfico verticales y semáforos y clasificar las señales de límite de velocidad dependiendo de su tipo. Como se expuso en la sección de tecnologías a usar (2.5.4), para la creación de aplicación de interfaz gráfica se utilizará QT con C++.

3.5.1. Vistas de la Aplicación

A continuación se presentarán las dos vistas que componen la interfaz gráfica.

Vista de Configuración

Para que SaferAuto funcione, necesita una serie de archivos de entrada. Esta vista tiene como finalidad permitir al usuario seleccionar esos ficheros de una manera visual y cómoda, en vez de tener que enviarlos por línea de comandos al tener que iniciar la aplicación. Una vez que estén seleccionados, se podrá iniciar la detección.

Como se observa en la figura 3.13, existen 4 botones para seleccionar ficheros y uno para iniciar la detección. Cuando se hace click sobre uno de los botones color turquesa, se abrirá un explorador de archivos donde se podrá seleccionar el fichero correspondiente con el botón. Una vez seleccionado, la ruta aparecerá en el texto del lado derecho del botón. Los archivos que permite seleccionar son:

- *CFG File* o Fichero de configuración: Indica la arquitectura de capas de la red, así como el tamaño de la misma y el número de clases que se quiere detectar.
- *Names File* o Fichero de nombres: Fichero que contiene las etiquetas de las distintas clases que se desean detectar separadas en líneas.

Figura 3.13: SaferAuto - Vista de Configuración.

- *Weights File* o Fichero de pesos: Los pesos son el resultado del entrenamiento. En este fichero se deberán situar los pesos que se quieren utilizar para la detección. Cabe añadir que estos deben haber sido entrenados con la misma arquitectura que la del fichero de configuración.

- *Media File* o Fichero de multimedia: Vídeo o imagen dónde se quiere realizar la detección de señales de tráfico verticales y semáforos.

Cabe destacar que el programa recordaría las rutas seleccionadas para estos ficheros para el próximo inicio de la aplicación. Por último, la función del botón de mayor tamaño es iniciar la vista de detección, ocultando la vista de configuración hasta que el usuario haya cerrado la otra vista o hasta que la detección haya finalizado.

Vista de Detección

Se trata de la vista principal. Si el fichero de entrada es una imagen, mostrará una modificación de la misma, añadiendo las señales que se hayan detectado mediante cuadros delimitadores y etiquetas. Por el contrario, si el fichero de entrada es un vídeo, se empezarán a mostrar las distintas imágenes que lo componen y las señales que se van detectando. La vista está compuesta por tres partes:

- Parte superior: Muestra un indicador de la tasa de imágenes por segundo a la que la red está analizando las imágenes.

- Parte central: Presenta las imágenes del vídeo de entrada a la velocidad de fps indicada en la parte superior. Si se detecta una señal, se señalará rodeándola con un cuadro delimitador de color naranja.

- Parte inferior: Está compuesta por 5 cuadrados donde se mostrarán las señales detectadas. En vez de añadir la señal como recorte de la imagen principal, se de-

cidió mostrar una señal modelo que represente a la clase detectada, para un mejor entendimiento visual de la detección.

Cabe añadir que solo se mostrarán las señales de las que se haya realizado seguimiento, para evitar añadir varias veces la misma señal. Para ello, en la parte superior de la señal se mostrará su número de seguimiento y clase. Este número se trata del índice del conjunto de señales detectadas al que pertenece la señal actual. Como detalle adicional, se presentará la probabilidad con la que se detectó la señal en la parte inferior.

La distribución de las señales en los 5 cuadros depende de su índice de seguimiento, es decir, del momento de aparición de la señal con respecto a las demás. La primera señal detectada se visualizaría en el primer cuadro, la segunda en el segundo cuadro y así sucesivamente. Cuando los cuadros estén completos se empezará a sustituir las señales por el lado izquierdo.

Figura 3.14: SaferAuto - Vista de Detección.

Finalmente se podría cerrar la vista de detección y volver a iniciar otra detección con parámetros distintos sin necesidad de reiniciar la aplicación.

Capítulo 4

Conclusiones y líneas futuras

4.1. Conclusiones

La conducción autónoma probablemente se trate de uno de los campos de estudio mas atractivos que existen en la actualidad. A pesar de que todavía no exista ningún vehículo que sea completamente autónomo sin requerir supervisión de un ser humano, lo mas seguro es que esto sea una realidad en los próximos años. Así lo confirmó Elon Musk, CEO de Tesla, durante el *Día de la Autonomía de Tesla*, asegurando que "probablemente dentro de dos años, Tesla hará un coche sin volante ni pedales" [20].

Para que un vehículo tenga la capacidad de ser autónomo, debe comprender todo lo que se encuentra en la carretera, al igual que lo hace un ser humano. Por ello, es imprescindible que aprenda como reconocer las señales de tráfico, semáforos o demás elementos viales que se encuentren en la escena, detectándolos en tiempo real.

Durante el capítulo 2 de este trabajo de fin de grado se han estudiado algunas de las tecnologías que hacen posible esto, como las Redes Neuronales Convolucionales o el sistema de detección a tiempo real YOLO. En el capítulo 3 se ha presentado el proceso de desarrollo de SaferAuto, un sistema de asistencia a la conducción, capaz de detectar señales de tráfico verticales y semáforos.

4.2. Contribuciones

Desde un principio, se ha desarrollado SaferAuto pensando en la modularidad del proyecto, para que sea sencillo poder implementar nuevos módulos de detección. Por ello, los programas Datasets2Darknet [11] y SaferAuto [13] son de código libre y se encuentran públicos en GitHub, con el fin de que la comunidad pueda incorporar nuevos módulos y se aumente la funcionalidad de SaferAuto.

Se han presentado los conjuntos de datos europeos mas completos que existen en la actualidad para el reconocimiento de señales de tráfico verticales y de semáforos, detallando las características que los diferencian, con el objetivo de que sirva de base a futuras investigaciones en el campo.

Se han ido documentando las tareas mas importantes a realizar para la utilización de YOLO, como la instalación del framework Darknet, utilización del programa Datasets2Darknet [11] y la configuración del sistema de detección. Todo se ha publicado en un blog del proyecto de SaferAuto [14], para contribuir a un mayor uso de esta tecnología.

Se han utilizado combinaciones de estos datasets como entrada de YOLO y se ha conseguido diferenciar 10 tipos de elementos viales (figura 3.4) con una precisión media de un 88 % y una tasa media de imágenes por segundo de 220 fps. Cabe resaltar que algunas categorías como la de peligro o semáforo en rojo alcanzan una precisión superior al 94 %. Con esto se comprueba que YOLO es un excelente modelo para detectar objetos requeridos para la conducción autónoma en tiempo real.

Por último, se ha desarrollado un programa que es capaz de utilizar distintos modelos de detección y de clasificación, permitiendo no solo detectar las señales de límite de velocidad, si no también clasificarlas según su contenido. Este programa ofrece una interfaz gráfica de usuario para que se puedan observar las detecciones de una manera cómoda y notifica estas mediante mensajes de voz.

4.3. Líneas Futuras

Debido a su gran modularidad, el rango de futuras funcionalidades del proyecto SaferAuto es bastante amplio. A continuación se presentan algunas de las mas interesantes para llevar a cabo.

- Ampliación de la clasificación de subclases de señales de tráfico verticales a todas las categorías que SaferAuto es capaz de detectar, persiguiendo solo mostrar señales específicas y no agrupaciones de las mismas.
- Adaptación de SaferAuto a móviles, a través del uso de TensorFlow Lite [36] y proyectos como TensorFlow Yolov3 [25]. Esto permitiría utilizar SaferAuto en cualquier tipo de vehículo, dado que solo se necesitaría el teléfono móvil.
- Detección de peatones. Sería de gran utilidad notificar de la presencia de peatones para asistir en situaciones de baja luminosidad o visión reducida.
- Detección de vehículos y clasificación según su tipo (turismos, furgonetas, camiones...)

4.4. Premios obtenidos

En Abril de 2019, en la fase final del XIII Concurso Universitario de Software Libre, SaferAuto obtuvo el premio nacional al mejor proyecto científico [38].

Capítulo 5

Conclusions and future work

5.1. Conclusions

Self-driving is probably one of the most attractive fields of study nowadays. Although there is not any fully autonomous car yet, it is more likely to become reality in the next few years. In fact, during the "Tesla Autonomy Investor Day", Elon Musk, Tesla CEO, claimed that Tesla cars could be sold without steering wheels and pedals within three years from now [20].

For a vehicle to be able to be autonomous, it must comprehend everything on the road, just as a human being does. Therefore, it is essential that it learns how to recognize traffic signs, traffic lights or any other road element in the scene, detecting them in real-time.

During the chapter 2 of this final-degree project, several technologies that make this possible have been studied, such as Convolutional Neural Networks or the real-time detection system YOLO. Chapter 3 presents the development of SaferAuto, a driving assistance system capable of detecting vertical traffic signs and traffic lights.

5.2. Contributions

From the beginning, SaferAuto has been developed with the modularity of the project in mind, so that is easy to implement new detection modules. Because of that, the developed programs Datasets2Darknet [11] and SaferAuto [13] are open-source and available on GitHub.

The most complete European datasets currently available for detecting traffic signs and traffic lights have been presented, detailing the features that differentiate them, with the aim of serving as basis for future research in the field.

The necessary tasks to use YOLO have been documented, such as the installation of Darknet framework [27], the use of Datasets2Darknet [11] and the configuration of the detection system. Everything has been published in the SaferAuto project blog [14] in order to contribute to a wider use of this technology.

Combinations of the presented datasets have been used as YOLO input and 10 types of road elements (figure 3.4) have been distinguished with an average precision of 88 % and an average image rate per second of 220 fps. It is noteworthy that some categories such as danger or red traffic lights achieve an accuracy higher than 94 %. This proves that YOLO is an excellent model for detecting objects required for autonomous driving in real time.

Finally, a program capable of using different detection and classification models has been developed, allowing not only to detect speed limit signals, but also to classify them according to their limit. This program includes a graphical user interface so that the detections can be displayed in a comfortable way and notifies them by voice messages.

5.3. Future Work

Due to its great modularity, the range of future features for SaferAuto is quite wide. Here are some of the most interesting to implement.

- Extension of the classification of sub-classes of vertical traffic signs to all categories that SaferAuto is able to detect, with the aim of only showing specific signs, instead of categories.
- Adaptation of SaferAuto to mobile phones, through the use of TensorFlow Lite [36] and projects such as TensorFlow Yolov3 [25]. This would allow SaferAuto to be used in any type of vehicle, as only the mobile phone would be needed.
- Pedestrian detection. It would be very useful to notify of the presence of pedestrians to assist in situations of low luminosity or reduced vision.
- Detection of vehicles and classification according to their type (cars, vans, trucks...)

5.4. Awards

In April 2019, in the final phase of the XIII National Open Source University Contest, SaferAuto won the national award of the best scientific project [38].

Capítulo 6

Presupuesto

Este capítulo tiene como objetivo presentar los costes estimados del proyecto.

Dado que todas las tecnologías utilizadas son de código libre y gratuitas, no habrá coste de software, por lo que los costes a estimar serán exclusivamente en hardware y en recursos humanos.

6.1. Costes en Hardware

Para el entrenamiento de YOLO se requiere de una tarjeta gráfica con núcleos CUDA. A mayor cantidad de estos, mayor velocidad de entrenamiento y validación, dado que se utiliza aceleración por GPU en ambas tareas. Atendiendo a esto, se ha elegido la siguiente configuración de Hardware para llevar a cabo SaferAuto.

Tipo	**Componente Hardware**	**Precio**
Placa base	*Asus PRIME X370-PRO*	*130 €*
CPU	*Ryzen 7 1700*	*170 €*
GPU	*Gigabyte GeForce GTX 1060 6G*	*340 €*
RAM	*Ram G Skill F4-3200C14D*	*160 €*
Memoria	*SSD Samsung 960 EVO M.2 250GB*	*137 €*
Fuente de Alimentación	*Be quiet Pure Power 10 600W*	*110 €*
Refrigeración Líquida	*Enermax Liqmax II 240*	*61 €*
Total		*1108 €*

Cuadro 6.1: Costes en Hardware

6.2. Costes en Recursos Humanos

A continuación se presentan las horas dedicadas en cada una de las tareas desarrolladas y su coste asociado. El precio por hora de trabajo se fijará a 20 €, al tratarse de un campo de gran atractivo alto y demanda.

Sección	**Tarea**	**Horas**	**Precio**
Estudio Tecnologías	*Investigación del estado del arte*	*50*	*1.000 €*
	Aprendizaje de la herramienta YOLO	*25*	*500 €*
	Búsqueda de los conjuntos de datos	*25*	*500 €*
Tratamiento de datos	*Formateado de los conjuntos de datos*	*100*	*2.000 €*
	Configuración de los modelos usados	*25*	*500 €*
Sistemas de Detección y Clasificación	*Pruebas de entrenamiento con múltiples combinaciones de datasets*	*200*	*4.000 €*
	Elección de los mejores modelos para detección y clasificación	*50*	*1.000 €*
Interfaz gráfica	*Desarrollo de aplicación gráfica*	*150*	*3.000 €*
Total		*550*	*12.500 €*

Cuadro 6.2: Costes en Recursos Humanos

6.3. Costes Totales

Los costes finales estimados para el proyecto de SaferAuto son:

Costes	**Precio**
Costes en Hardware	*1.108 €*
Costes en Recursos Humanos	*12.500 €*
Total	*13.608 €*

Cuadro 6.3: Costes Totales

Bibliografía

[1] AlexeyAB. Darknet Fork. *https://github.com/AlexeyAB/darknet*.

[2] Bajaj, R. H., and Ramteke, P. Big data–the new era of data. *International Journal of Computer Science and Information Technologies 5*, 2 (2014), 1875–1885.

[3] Blog, M. What is the Mazda Active Driving Display? *https://www.mazdaofgermantown.com/blog/what-is-the-mazda-active-driving-display/*.

[4] Company, T. Q. Qt — Cross-platform software development for embedded desktop. *https://www.qt.io/*.

[5] Consortium, C. COCO - Common Objects in Common. *http://cocodataset.org/home*.

[6] Croatia. MASTIF DATASET. *http://www.zemris.fer.hr/šsegvic/mastif/datasets.shtml*.

[7] DGT. En 2017 fallecieron 1.830 personas en accidente de tráfico. *http://www.dgt.es/es/prensa/notas-de-prensa/2018/20180712-en-2017-fallecieron-1830-personas-en-accidente-de-trafico.shtml*.

[8] Ford. Tecnología para asistencia a la conducción. *https://www.ford.es/compra/explora/tecnologia/experiencia-de-conduccion*.

[9] Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2014), pp. 580–587.

[10] Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., and Igel, C. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks (submitted)* (2013).

[11] Igareta, A. Datasets2Darknet - Tool that extracts data from multiple datasets and parses it to Darknet format. *https://github.com/angeligareta/Datasets2Darknet*.

[12] Igareta, A. Fork of AlexeyAB Darknet Fork. *https://github.com/angeligareta/darknet*.

[13] Igareta, A. SaferAuto - Real-time detection system to assist during driving. *https://github.com/angeligareta/SaferAuto*.

[14] Igareta, A. SaferAuto Blog - Real-time detection system to assist during driving. *https://saferauto.home.blog/*.

[15] INI. German Traffic Sign Detection Benchmark. *https://btsd.ethz.ch/shareddata/*.

[16] Intelligent, L. F., and Automobiles, S. LISA Traffic Light Dataset. *http://cvrr.ucsd.edu/vivachallenge/index.php/traffic-light*.

[17] Jensen, M. B., Nasrollahi, K., and Moeslund, T. B. Evaluating state-of-the-art object detector on challenging traffic light data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2017), pp. 9–15.

[18] LeCun, Y. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.

[19] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[20] Limited, S. T. D. Elon Musk claims Tesla cars could be sold without steering wheels and pedals within three years. *https://www.driving.co.uk/news/elon-musk-claims-tesla-cars-sold-without-steering-wheels-pedals-within-three-years//*.

[21] Lin, S.-C., Zhang, Y., Hsu, C.-H., Skach, M., Haque, M. E., Tang, L., and Mars, J. The architectural implications of autonomous driving: Constraints and acceleration. In *ACM SIGPLAN Notices* (2018), vol. 53, ACM, pp. 751–766.

[22] Liu, G. *Real-time object detection for autonomous driving-based on deep learning*. PhD thesis, 2017.

[23] Mathias, M., Timofte, R., Benenson, R., and Van Gool, L. Traffic sign recognition—how far are we from the solution? In *The 2013 international joint conference on Neural networks (IJCNN)* (2013), IEEE, pp. 1–8.

[24] Mogelmose, A., Trivedi, M. M., and Moeslund, T. B. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems 13*, 4 (2012), 1484–1497.

[25] Mystic123. TensorFlow Yolov3. *https://github.com/mystic123/tensorflow-yolo-v3*.

[26] Pjreddie. Tiny Darknet. *https://pjreddie.com/darknet/tiny-darknet/*.

[27] Pjreddie. YOLO: Real-Time Object Detection. *https://pjreddie.com/darknet/yolo/*.

[28] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 779–788.

[29] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[30] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (2015), pp. 91–99.

[31] Rohith, and Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms. *https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e*.

[32] Shakhuro, V., and Konushin, A. Russian traffic sign images dataset. *Computer Optics 40*, 2 (2016), 294–300.

[33] Subaru of America, I. An extra set of eyes on the road. *https://www.subaru.com/engineering/eyesight.html*.

[34] Sumit, and Saha. A Comprehensive Guide to Convolutional Neural Networks. *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*.

[35] Tabora, V., and Tabora, V. Tesla Enhanced Autopilot Overview. *https://medium.com/self-driving-cars/tesla-enhanced-autopilot-overview-l2-self-driving-hw2-54f09fed11f1*.

[36] TensorFlow. TensorFlow Lite. *https://www.tensorflow.org/lite*.

[37] Tesla, I. Tesla Autopilot. *https://www.tesla.com/autopilot*.

[38] Wikipedia. Concurso Universitario de Software Libre. *https://es.wikipedia.org/wiki/Concurso_Universitario_de_Software_Libre*.

[39] Zurich, E. BelgiumTS Dataset. *https://btsd.ethz.ch/shareddata/*.