

“Diseño e implementación de un
prototipo de sistema de ayuda a la
conducción en vehículos basado en
detección de objetos empleando
visión artificial”

Diego Álvarez Ponce

Tutor: Julio Villena Román

Leganés, Octubre 2020



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento
– No Comercial – Sin Obra Derivada**

RESUMEN

El proyecto descrito en este documento se corresponde con el trabajo realizado para la implementación de un prototipo de sistema de ayuda a la conducción mediante visión artificial para vehículos autónomos. Este campo se encuentra en pleno auge gracias a los avances introducidos en el ámbito de la detección de objetos y el comienzo de su aplicación en automóviles.

El algoritmo utilizado para la construcción de este sistema ha sido You Only Look Once (YOLO) en su versión más reciente, desarrollado por la compañía Ultralytics. Este algoritmo proporciona un gran rendimiento en la detección de objetos en tiempo real, haciendo uso de capas convolucionales. Las ventajas que lo diferencian del resto de modelos son su rapidez en el procesado y su alta precisión en la detección final.

Para la implementación del prototipo se ha utilizado un triple detector sobre cada fotograma. Cada uno de dichos detectores se encarga del reconocimiento de un grupo de objetos: el primero hace uso del conjunto de datos COCO y sus correspondientes pesos pre-entrenados para la detección de elementos dinámicos como vehículos y personas; el segundo se ha entrenado con el conjunto de datos de Bosch con el fin de diferenciar la luz iluminada en semáforos y su flecha de dirección y, finalmente, el tercer detector se aplica para el reconocimiento de señales de tráfico. De cara a la obtención de las imágenes necesarias para realizar el entrenamiento de este último detector, se ha combinado el conjunto de datos de German Traffic Sign Detection Benchmark (GTSDB) con imágenes etiquetadas manualmente mediante la herramienta LabelImg.

Para concluir, una vez comprobado el correcto funcionamiento, se ha incorporado un módulo encargado del sistema de ayuda a la conducción. Este muestra por pantalla la velocidad máxima de la vía con la ayuda de las señales de velocidad detectadas. En caso de encontrar una señal de Stop, una señal de paso de peatones junto con la detección de un viandante, un semáforo en rojo o en caso de detectarse la presencia de obstáculos en la trayectoria, este valor se reduce. Además, se muestran los iconos de las señales detectadas en el lado correspondiente de la vía, así como los semáforos en la parte superior. Adicionalmente, se ha agregado un sistema de alerta por proximidad.

Palabras clave: Detección de obstáculos; YOLO; Vehículo autónomo; Ayuda a la conducción.

AGRADECIMIENTOS

Gracias a mis padres y mi hermana por el apoyo y cariño que me han mostrado durante toda mi vida. Por darme la oportunidad de estudiar lo que me gustaba. Desde el inicio de esta etapa habéis mostrado una gran confianza en mí y en todo lo que hago, empujándome a seguir adelante. Sin vosotros no hubiese podido llegar hasta aquí.

Gracias, por supuesto a mis abuelos por cuidarme y quererme como lo hacéis. Por haberme criado desde pequeño y por todo lo que me habéis enseñado. Jamás conoceré a unas personas con tanto corazón, tan luchadoras y con tanta bondad como vosotros. Vuestro niño se hace mayor.

Gracias a mi tutor, Julio, no sólo por la ayuda y atención mostrada para la realización de este proyecto, sino por tu dedicación y pasión por la enseñanza que demuestras con cada clase. Se nota que te interesa y disfrutas con lo que haces y eso llega a tus alumnos.

A ti, Jimena, por la paciencia y confianza que has tenido conmigo. Gracias por estar ahí cuando más lo necesitaba, arropándome en todo momento. Gracias por tantos momentos buenos. Por el apoyo y la ayuda que me has dado desde que te conocí. Sin ti no lo hubiese logrado.

Y por supuesto, gracias a los amigos que me han acompañado en esta etapa de mi vida, dentro y fuera de la universidad, pero especialmente, a mi grupo de la carrera. Mi segunda familia. Gracias a Alba, Alex, Ayuso, Juancar, Laura y Richi por estos cinco años juntos. Que suerte haberos conocido y haber podido recorrer este camino junto. Por todos los viajes, por todas las risas y buenos momentos vividos y los que están por llegar. GRACIAS.

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN	1
1.1. Contexto socioeconómico	1
1.2. Motivación	2
1.3. Objetivos	3
1.4. Marco regulador	3
1.4.1. Protección y tratamiento de los datos	4
1.4.2. Legislación en sistemas de conducción autónoma.....	5
1.5. Estructura del documento	6
2. ESTADO DEL ARTE.....	8
2.1. Terminología	8
2.2. Aprendizaje profundo	9
2.2.1. Historia del aprendizaje profundo.....	10
2.2.2. Redes neuronales	12
2.2.3. Redes neuronales convolucionales	15
3. DETECCIÓN DE OBJETOS	20
3.1. Introducción	20
3.2. Algoritmos principales	21
4. ELECCIÓN DEL ALGORITMO	25
5. ALGORITMO YOLO.....	28
5.1. Estructura de YOLO.....	28
5.2. Funcionamiento del algoritmo.....	30
5.3. Evoluciones de YOLO	33
5.3.1. YOLO v1	34
5.3.2. YOLO v2.....	34
5.3.3. YOLO v3	35
5.3.4. YOLO v4.....	36
5.3.5. YOLO v5	37
6. TRATAMIENTO DE LOS DATOS.....	39
6.1. Requisitos de los datos de entrenamiento	39
6.2. Elección del conjunto de datos	40
6.2.1. Dataset de vehículos y personas	41
6.2.2. Dataset de semáforos	42
6.2.3. Dataset de señales de tráfico.....	44
7. ENTRENAMIENTO DEL MODELO	49

7.1.	Preparación del entrenamiento	49
7.2.	Proceso de entrenamiento.....	50
7.2.1.	Indicadores de validación	51
7.3.	Resultados obtenidos	52
7.4.	Validación	55
8.	SISTEMA DE AYUDA A LA CONDUCCIÓN	60
8.1.	Control de velocidad	60
8.2.	Control de proximidad.....	63
8.3.	Representación visual de señales.....	64
8.4.	Validación del prototipo	66
9.	PLANIFICACIÓN Y PRESUPUESTO DEL PROYECTO	68
9.1.	Planificación.....	68
9.2.	Presupuesto.....	70
9.2.1.	Costes materiales	70
9.2.2.	Costes personales.....	71
9.2.3.	Costes totales	72
9.2.4.	Presupuesto para la integración	73
10.	CONCLUSIONES Y LÍNEAS FUTURAS DE DESARROLLO	75
10.1.	Conclusiones	75
10.2.	Líneas futuras de desarrollo.....	76
	BIBLIOGRAFÍA	79
	EXTENDED ABSTRACT	85

ÍNDICE DE FIGURAS

Fig. 2.1. Estructura de las diferentes ramas de la inteligencia artificial [8]	8
Fig. 2.2. Representación de la Neurona desarrollada por McCulloch-Pitts [14]	10
Fig. 2.3. Representación del modelo Adaline desarrollado por Bernard Widrow [16]	11
Fig. 2.4. Representación de la estructura de las redes neuronales artificiales [20]	12
Fig. 2.5. Curva descrita por las ecuaciones de las funciones Sigmoide (izq.) y SoftMax (dcha.) [24]	14
Fig. 2.6. Representación de la curva descrita por la función de activación tangente hiperbólica [25]	14
Fig. 2.7. Representación gráfica de la función ReLU junto con su ecuación correspondiente [26]	15
Fig. 2.8. Representación gráfica de las funciones Leaky ReLU y Parametric ReLU [27]	15
Fig. 2.9. Convolución en imágenes de entrada de tres canales (RGB) [29]	16
Fig. 2.10. Representación del desplazamiento del filtro convolucional según el valor de paso [29]	17
Fig. 2.11. Resultado obtenido al añadir relleno a la imagen [29]	18
Fig. 2.12. Diferencias entre las salidas de Max Pooling y Average Pooling [29]	18
Fig. 2.13. Estructura final de una red neuronal convolucional	19
Fig. 3.1. Diferencias entre clasificación de imágenes y detección de objetos [31]	20
Fig. 3.2. Diferencias entre detección de objetos y segmentación de instancias [31]	21
Fig. 3.3. Estructura y funcionamiento de la red R-CNN [32]	22
Fig. 3.4. Estructura del modelo Fast R-CNN [33]	22
Fig. 3.5. Comparación entre los tres modelos de la familia R-CNN [35]	23
Fig. 3.6. Representación de la estructura del modelo RetinaNet [37]	23
Fig. 3.7. Estructura de la red utilizada por Single Shot Detection [38]	24
Fig. 4.1. Gráfica comparativa entre precisión y tiempo de procesamiento para los algoritmos [42]	27
Fig. 5.1. Estructura de los detectores de una única etapa [44]	28
Fig. 5.2. Estructura de capas en los bloques densos [45]	29
Fig. 5.3. Estructura de la etapa de columna vertebral de YOLO [46]	29
Fig. 5.4. Estructura y caminos seguidos por las imágenes en el módulo PAN [47]	30
Fig. 5.5. Estructura del vector de salida del algoritmo de detección de YOLO [48]	31
Fig. 5.6. Representación de las cajas de anclaje sobre una imagen [50]	32
Fig. 5.7. Cálculo de las dimensiones de las cajas delimitadoras [51]	32
Fig. 5.8. Proceso seguido para la detección de objetos en el algoritmo YOLO [50]	33
Fig. 5.9. Comparativa de YOLOv2 en términos de velocidad y precisión [51]	34
Fig. 5.10. Comparativa de YOLOv3 en términos de velocidad y precisión [54]	35
Fig. 5.11. Comparativa de YOLOv4 en términos de velocidad y precisión [44]	36
Fig. 5.12. Representación de la curva descrita por la función de activación Mish	37
Fig. 5.13. Comparativa de YOLOv5 en términos de velocidad y precisión [43]	38
Fig. 6.1. Código utilizado para la detección mediante tres detectores independientes	40
Fig. 6.2. Formato de las etiquetas para Darknet [55]	40
Fig. 6.3. Ejemplos de imágenes dentro del conjunto de imágenes COCO junto con sus etiquetas	42
Fig. 6.4. Fragmento de código encargado de la detección de las clases de COCO seleccionadas	42

Fig. 6.5. Ejemplos de imágenes dentro del conjunto de Bosch junto con sus etiquetas	43
Fig. 6.6. Distribución de etiquetas en el dataset Bosch	43
Fig. 6.7. Formato de las etiquetas en el conjunto de imágenes GTSDb	45
Fig. 6.8. Código necesario para el cálculo del centro, ancho y alto del rectángulo delimitador	46
Fig. 6.9. Ejemplo del proceso de etiquetado de señales con LabelImg	47
Fig. 6.10. Ejemplos de imágenes dentro del conjunto de señales de tráfico junto con sus etiquetas	47
Fig. 7.1. Ficheros para los conjuntos de imágenes de semáforos (izq.) y señales (der.)	49
Fig. 7.2. Comandos para el entrenamiento de los modelos de semáforos (superior) y señales (inferior) ...	50
Fig. 7.3. Distribución de las muestras para los conjuntos de semáforos (izq.) y señales (der.)	52
Fig. 7.4. Comparación de tamaño de objetos entre semáforos (izq.) y señales (der.)	53
Fig. 7.5. Comparación de los indicadores de rendimiento para semáforos (izq.) y señales (der.)	54
Fig. 8.1. Ejemplo de actualización de la velocidad máxima de la vía	61
Fig. 8.2. Actualización del indicador de velocidad ante una señal de Stop	61
Fig. 8.3. Actualización de la velocidad ante un paso de peatones.	62
Fig. 8.4. Sistema de frenado ante semáforo rojo o ámbar	62
Fig. 8.5. Ejemplo del sistema de detección de proximidad.	63
Fig. 8.6. Código para la representación de las señales en el lado correspondiente	65
Fig. 8.7. Ejemplos de la visualización de señales en el sistema de ayuda a la conducción	65
Fig. 8.8. Ejemplos de la visualización de semáforos en el sistema de ayuda a la conducción	66
Fig. 9.1. Representación del diagrama de Gantt seguido en el proyecto	69

ÍNDICE DE ECUACIONES

Ecuación 7.1. Representación del cálculo del indicador IoU	51
Ecuación 7.2. Ecuación para el cálculo de la precisión del modelo	51
Ecuación 7.3. Ecuación del cálculo del indicador de Recall	51
Ecuación 7.4. Ecuación para el cálculo del indicador mAP	52
Ecuación 9.1. Fórmula para el cálculo de la amortización	70

ÍNDICE DE TABLAS

Tabla 4.1. Comparativa de los valores de precisión media para los diferentes algoritmos [42].....	26
Tabla 6.1. Comparación de las diferentes competiciones sobre la detección de objetos	41
Tabla 6.2. Distribución final de las clases utilizadas junto con su correspondiente etiqueta.....	44
Tabla 6.3. Distribución de las clases del conjunto de datos de señales de tráfico	48
Tabla 7.1. Tabla con los mejores valores de indicadores obtenidos en ambos entrenamientos.....	55
Tabla 7.2. Tabla de detecciones durante el día y la noche con el mismo umbral	56
Tabla 7.3. Cálculo de la precisión y el recall para cada vídeo bajo el mismo umbral	57
Tabla 7.4. Tabla de detecciones durante el día y la noche reduciendo el umbral	58
Tabla 7.5. Cálculo de la precisión y el <i>recall</i> para cada vídeo disminuyendo el umbral	59
Tabla 9.1. Descripción de las tareas realizadas durante la realización del proyecto	69
Tabla 9.2. Desglose de gastos en costes materiales	71
Tabla 9.3. Desglose de gastos en costes personales	72
Tabla 9.4. Desglose de gastos totales del proyecto.....	72
Tabla 9.5. Desglose de costes para posible integración del sistema desarrollado	74

1. INTRODUCCIÓN

En este primer capítulo se describen los aspectos relacionados con la situación actual de las tecnologías que se aplican en el reconocimiento de objetos en tiempo real, así como su implementación en vehículos autónomos y el marco regulador existente. Asimismo, se explican tanto las motivaciones para la realización de este Trabajo de Fin de Grado (TFG), como los objetivos a alcanzar con el sistema desarrollado.

1.1. Contexto socioeconómico

Los continuos desarrollos y mejoras en el ámbito de la inteligencia artificial y la aparición de nuevos campos de estudio están impulsando una gran transformación en la sociedad. Es tal el avance, que se ha ido integrando en la vida cotidiana de las personas, compaginándose a la perfección con su día a día y mejorando la calidad de vida de los usuarios que interactúan con estas tecnologías.

El ámbito de la conducción no ha sido ajeno a estos cambios. La primera aparición del vehículo autónomo data del año 1939, durante la feria Futurama [1], en la ciudad de Nueva York. El objetivo de este acontecimiento era mostrar a la sociedad los avances y transformaciones que iba a sufrir el mundo durante los siguientes 30 años. La visión de Norman Bel Geddes, diseñador del prototipo, no iba desencaminada. Sin embargo, no será hasta las últimas décadas cuando se produzca una gran revolución en este campo.

La producción en serie de vehículos equipados con sistemas de conducción autónoma, como los desarrollados por la multinacional Tesla, surge gracias a los avances en el campo de la inteligencia artificial, más concretamente en los modelos de visión artificial. Estos sistemas incluyen algoritmos de detección de objetos capaces de reconocer obstáculos, carriles, señales, etc., capturados mediante cámaras integradas en el propio vehículo. Con esta información y la recopilada por otros dispositivos y sensores, el vehículo puede efectuar una conducción sin la intervención humana, es decir, autónoma.

No obstante, al seguir estos modelos en desarrollo, presentan errores, especialmente ante situaciones adversas o poco frecuentes. Es por ello que se ha establecido una escala de 6 niveles para medir el grado de desarrollo del sistema de navegación autónoma [2],

comenzando por un nivel 0, correspondiente a automóviles sin asistencia a la conducción, y otorgando un valor de 5 a los vehículos con autonomía completa. Cabe destacar el sistema utilizado en el Tesla Autopilot [3], que, pese a ser uno de los modelos más desarrollados, aún se encuentra posicionado en el nivel 2 con autonomía parcial.

Como se puede deducir de las ideas expuestas, la sociedad se encuentra todavía lejos de ese sistema completamente autónomo. Sin embargo, cada vez son más las compañías que se suman al reto de la conducción autónoma para el futuro. Es la llegada de estas empresas la que ha desatado una revolución en el sector automovilístico. Hoy en día se destinan elevados presupuestos a su investigación, lo que promueve los avances, una alta competencia y el resultante abaratamiento de estos productos.

1.2. Motivación

La inteligencia artificial se ha posicionado a lo largo de los últimos años como una de las áreas más importantes en el desarrollo tecnológico. Con la mejora de hardware y software, así como con la aparición de grandes conjuntos de datos, este campo se ha podido desarrollar, abriendo diferentes vías de investigación y de avance. La inteligencia artificial abre nuevas puertas, permitiendo la aparición de distintas tecnologías y la construcción de algoritmos capaces de facilitar tareas anteriormente sólo realizables por un ser humano.

Uno de los cometidos en los que se ha comenzado a poner un mayor interés es en la conducción de vehículos. Gracias a estos avances se ha visto la posibilidad de implementar sistemas que ayuden a los conductores, haciendo su labor más sencilla. Las técnicas introducidas van desde la integración de limitadores de velocidad o sensores para evitar que el vehículo se salga de su carril, hasta el intento de construir un vehículo completamente autónomo.

El potencial de las tecnologías aplicadas a vehículos es inmenso, pudiéndose reducir situaciones peligrosas y accidentes con la simple adición de algún sistema en el automóvil. Con este trabajo se busca, por consiguiente, el desarrollo de una tecnología capaz de alertar al conductor frente a estímulos que se puedan presentar durante los desplazamientos realizados, mediante la aplicación de inteligencia artificial y el uso de nuevas tecnologías.

1.3. Objetivos

El objetivo principal de este trabajo es la implementación de un prototipo de sistema de ayuda a la conducción para vehículos autónomos que sea capaz de reconocer obstáculos, así como señales y semáforos. La idea consiste en, a partir de una imagen de entrada proporcionada por una cámara situada en la parte delantera del vehículo y sin hacer uso de otros elementos como sensores o señales GPS, elaborar un sistema capaz de facilitar la conducción. Para su realización se establecen los siguientes objetivos:

- Análisis del Estado del Arte respecto a los algoritmos de visión artificial, recopilando las metodologías más importantes usadas actualmente para la tarea de detección de objetos en tiempo real.
- Comparación de los algoritmos candidatos a ser aplicados, teniendo en cuenta los puntos fuertes y débiles de cada uno, bajo unos criterios de velocidad de procesamiento y precisión.
- Elección de las categorías a detectar y búsqueda de los conjuntos de imágenes correspondientes para el proceso de entrenamiento, ajustando los datos al formato requerido por el modelo seleccionado.
- Entrenamiento del modelo con los valores de parámetros adecuados para la obtención del mejor rendimiento posible.
- Desarrollo del sistema de ayuda a la conducción en base a los objetos detectados.
- Análisis y validación de los resultados obtenidos mediante grabaciones propias desde el vehículo.

1.4. Marco regulador

El prototipo a implementar, como ha sido mencionado en el epígrafe anterior, pretende hacer uso de una cámara que grabe el entorno durante la conducción. Esto implica la aparición en el vídeo de personas, vehículos y otra clase de datos sensibles protegidos por la ley. A su vez, la implementación de un sistema de conducción autónoma también puede verse afectada por diferentes normativas según la zona geográfica en la que uno se encuentre.

1.4.1. Protección y tratamiento de los datos

En Europa existe el Reglamento General de Protección de Datos (RGPD) [4], que fue introducido en el año 2016 y puesto en vigor el 25 de mayo de 2018. Una de las grandes novedades dentro de este reglamento, frente a normativas anteriores, es la aplicación de las normas a cualquier persona o empresa externa a la Unión Europea, siempre que se trate de manipulación de datos de individuos residentes dentro de ella. Además, se tratará como dato personal toda aquella información que permita la identificación del individuo.

Por otro lado, en el territorio español, la Agencia Española de Protección de Datos (AEPD) es la entidad encargada de concretar ciertos aspectos dentro del RGPD europeo. En octubre de 2018 se aprueba en España la nueva Ley Orgánica de Protección de Datos y Garantía de los Derechos Digitales (LOPDGDD). Junto a ella se publican diversas guías informativas sobre los puntos recogidos en esta ley.

La normativa que concierne al prototipo implementado para este trabajo queda recogida en la Guía de Videovigilancia [5]. En este documento se diferencia entre dos tipos de supuestos: el primero, referido a grabaciones en el ámbito doméstico y el segundo, para la obtención de pruebas ante un suceso.

El caso referido a grabaciones domésticas abarca todas aquellas grabaciones que no requieran de su almacenaje o que se utilicen para un propósito personal sin ánimo de lucro. En la guía se incluyen en esta categoría, dentro del capítulo dedicado a las tecnologías emergentes, las grabaciones realizadas con las denominadas cámaras “*on board*”.

En lo que concierne al segundo caso referido, se considerará videovigilancia cualquier situación en la que se produzca un almacenaje de las imágenes recopiladas con búsqueda de objetivos concretos. Este caso estará autorizado solamente para las Fuerzas y Cuerpos de Seguridad del Estado. Sin embargo, algunos coches autónomos almacenan las imágenes recopiladas durante la conducción para posteriormente ser analizados por la compañía de cara al entrenamiento de modelos de conducción autónoma. De esta forma, la empresa se abastece de gran cantidad de datos y situaciones dadas en la conducción que permiten el ajuste de los modelos. Por tanto, las grabaciones realizadas se verán sometidas a la normativa de videovigilancia, quedando limitadas en España.

En lo que concierne a las normativas que afectan al prototipo implementado, se verán limitadas únicamente al caso de vídeos de uso doméstico. Esto se debe a que con él no se pretende realizar un almacenaje de los datos capturados por la cámara, sino, simplemente, procesar en tiempo real la imagen recogida desde el vehículo. Además de esto, el entrenamiento del sistema no se ha realizado mediante la toma de imágenes por parte del usuario, sino mediante el uso de *datasets* disponibles en internet.

1.4.2. Legislación en sistemas de conducción autónoma

Respecto a la conducción y su regulación, en Europa se establecen algunas de las leyes más restrictivas aplicadas al uso de vehículos autónomos en su territorio. Dentro de las directivas de la Unión Europea se puede encontrar, entre otros, el Reglamento número 79 de la Comisión Económica para Europa de las Naciones Unidas (CEPE) [6], en el cual se regula la homologación del mecanismo de dirección de los vehículos.

Un ejemplo de ello podrían ser los sistemas dotados de cambio de carril automático, que, en este continente, disponen de un tiempo de 5 segundos para efectuar la maniobra. En caso de no poder realizarse la acción en ese tiempo, el vehículo debe abandonar el intento. No obstante, esta normativa ha generado debate dado que un tiempo tan pequeño puede causar movimientos bruscos que afecten a la seguridad e integridad de la persona. Por este y otros motivos, las normativas vigentes se encuentran en constante actualización.

Otro ejemplo dentro del mismo reglamento es la limitación en Europa de la toma de desvíos para incorporaciones o salidas de carreteras. Se impone un límite de 63 km/h para que el vehículo proceda con el giro de manera autónoma. En caso de sobrepasar esa velocidad, el sistema debe desconectarse y la maniobra debe llevarse a cabo por parte del conductor.

Además de las limitaciones ya mencionadas, existen numerosas normas adicionales que afectan a este tipo de conducción autónoma. Así se debe tener en cuenta que la legislación está aún en pleno desarrollo. Debido a la falta de información y la novedad de los sistemas autónomos, ciertas funcionalidades como el Smart Summon [7] de Tesla, mediante el cual se permite el control del vehículo desde un dispositivo móvil, han sido bloqueadas en este continente por prevención.

1.5. Estructura del documento

La memoria de este TFG se ha estructurado teniendo en cuenta los objetivos marcados en el epígrafe 1.3, siguiendo una estructura cronológica del trabajo realizado. El documento se compone de 10 capítulos:

1. En este primer capítulo se ha realizado una breve explicación sobre el proyecto desarrollado, el motivo de la realización del mismo, los objetivos que se pretenden alcanzar y el análisis de los ámbitos a los que repercute directamente como pueden ser el entorno social, el económico y el legislativo.
2. En el segundo capítulo se realiza un estudio y análisis teórico del Estado del Arte, detallando la evolución histórica de la inteligencia artificial y el aprendizaje profundo. Además, se ha descrito la estructura básica de las redes neuronales, así como los distintos elementos que la componen. Finalmente, se explica el funcionamiento de las redes neuronales convolucionales y su uso en modelos basados en el procesamiento de imágenes.
3. En el tercer capítulo se ha expuesto la importancia de la detección de objetos para sistemas de reconocimiento en tiempo real. Además, se explican los algoritmos de una y dos fases más destacados en este campo.
4. El cuarto capítulo está dedicado al análisis de las ventajas e inconvenientes de cada uno de los sistemas detallados en el capítulo 3, comparando los valores de tiempo de procesamiento por fotograma y precisión en la detección, de cara a la elección del algoritmo.
5. En el quinto capítulo se hace una explicación teórica de la estructura y funcionamiento interno del algoritmo seleccionado, partiendo de la información presentada sobre redes neuronales y redes neuronales convolucionales en el segundo capítulo. Finalmente, se profundiza en las distintas evoluciones que ha sufrido el algoritmo hasta la última versión.
6. En el sexto capítulo se explican los puntos a tener en cuenta en la elección de los conjuntos de datos. También, se indican los elementos seleccionados para su detección y se detalla la metodología seguida para la estandarización de los conjuntos de imágenes seleccionados.
7. El séptimo capítulo muestra el proceso seguido para la obtención de los pesos usados en los detectores. Consta de la creación de los ficheros de nombres de clase

y configuración, así como de la elección de los parámetros introducidos en la ejecución del entrenamiento. Por último, se ha realizado una validación del sistema implementado.

8. En el octavo capítulo se describe el desarrollo del sistema de detección para ajustarlo a un sistema de ayuda a la conducción, basado en el reconocimiento de señales y objetos dinámicos, junto con un indicador de proximidad de obstáculos. Finalmente, se ha vuelto a realizar un análisis del rendimiento del prototipo completo.
9. En el noveno capítulo se detalla la planificación seguida para la elaboración del proyecto, junto con los costes materiales y personales del mismo y un posible presupuesto para su futura implementación.
10. El último capítulo recoge las conclusiones obtenidas durante la realización del proyecto y las posibles futuras líneas de desarrollo del sistema implementado.

2. ESTADO DEL ARTE

En el capítulo actual se realiza un análisis del Estado del Arte, la evolución y desarrollo que ha ido experimentando la inteligencia artificial (IA) a lo largo de los años y su situación actual. Se ha hecho una distinción entre las diferentes ramas de la IA, profundizando en la explicación del aprendizaje profundo y el uso de redes neuronales. Finalmente, se habla sobre las redes neuronales convolucionales y su utilidad en modelos basados en el procesamiento de imágenes.

2.1. Terminología

En ocasiones, los términos inteligencia artificial, aprendizaje máquina y aprendizaje profundo son utilizados como sinónimos. Este pensamiento es erróneo ya que, aunque exista una relación entre ellos, no son equiparables. Esto se debe a que hablamos de un término más genérico que engloba a uno cada vez más específico. La estructura que se sigue se puede apreciar en la figura 2.1.

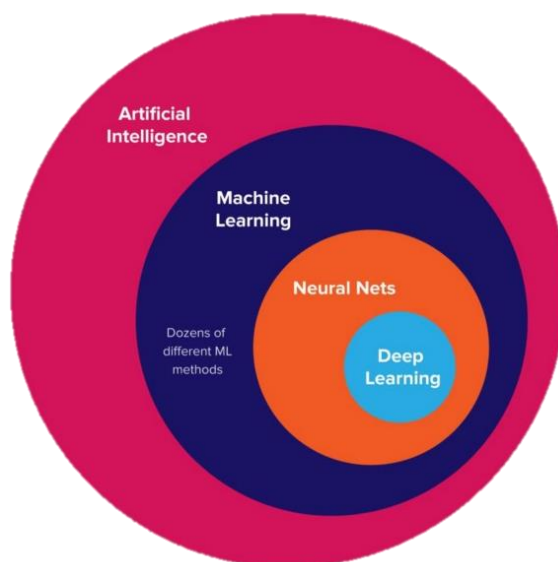


Fig. 2.1. Estructura de las diferentes ramas de la inteligencia artificial [8]

La Real Academia Española (RAE) define la inteligencia artificial como “una disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico” [9]. Dicho de otra manera, la inteligencia artificial es la ciencia que abarca

cualquier metodología implicada en el desarrollo de máquinas capaces de resolver problemas considerados únicamente solucionables por humanos, como son el aprendizaje profundo o el aprendizaje máquina.

El aprendizaje computacional [10] o *Machine Learning* (ML) es un subgrupo dentro de la inteligencia artificial. Se enfoca principalmente en el desarrollo de modelos capaces de aprender a realizar tareas sin haber sido programados explícitamente para ellas. Para llevar a cabo esta labor son necesarios tres componentes: la elección y desarrollo de un algoritmo, el entrenamiento del modelo sobre un conjunto de datos y, finalmente, una extracción de características para que la máquina comprenda a qué datos prestar especial atención.

Pasando al siguiente subgrupo encontramos las redes neuronales o *Neural Networks* (NN) [11]. Estas son modelos informáticos de aprendizaje computacional que se asemejan al funcionamiento y estructura del cerebro humano, capaces de aprender y ajustarse con cada iteración a una mejor realización de las tareas especificadas. Debido a esta semejanza de comportamiento con el procesamiento racional humano, se aplican en el desarrollo de sistemas de inteligencia artificial.

En último lugar, vamos a hablar del grupo más reducido dentro del campo de la inteligencia artificial. Las redes neuronales se han desarrollado en la última década hacia un tipo de red cuya estructura utiliza varias capas internas encargadas de la extracción de características de grandes cantidades de datos. El ámbito encargado del estudio y desarrollo de dichas redes será el aprendizaje profundo o *Deep Learning* (DL), del que se habla en mayor profundidad en el siguiente epígrafe.

2.2. Aprendizaje profundo

El aprendizaje profundo [12] pertenece a un subgrupo del aprendizaje computacional y basa su funcionamiento en la extracción de características a través del procesamiento de gran cantidad de muestras. Al contrario que para el *Machine Learning*, para el que es necesario saber la tarea a realizar y en el que es el ser humano el encargado de la extracción de características, en el *Deep Learning* sólo es necesario un conjunto de muestras para que la máquina pueda ser entrenada y pueda realizar esta tarea de manera

automática. Sin embargo, es necesario realizar un análisis más elaborado de este campo de cara a poder entenderlo correctamente.

2.2.1. Historia del aprendizaje profundo

Los avances en los modelos de aprendizaje profundo de los últimos años han supuesto un gran cambio en la sociedad y ha hecho que se posicione como una de las tecnologías con mayor potencial para el futuro. No obstante, aunque pueda parecer un acontecimiento reciente, estos avances son fruto de un largo proceso histórico de desarrollo.

La primera aparición del término *Deep Learning* data de los años 40. En su momento no fue una de las ramas de la inteligencia artificial más populares, pero, gracias a las reestructuraciones y evoluciones que ha ido sufriendo con el paso del tiempo, se ha convertido en uno de los métodos más utilizados para la resolución de problemas de procesamiento de imágenes por máquinas.

Las bases del *Deep Learning* como se entiende en la actualidad se asentaron durante la época comprendida entre 1940 y 1980. En esta época aparece la idea del aprendizaje biológico, que trata de imitar la estructura y funcionamiento del cerebro humano [13]. En 1943 aparece el concepto de Neurona de McCulloch-Pitts [14], gráficamente representada como aparece en la figura 2.2.

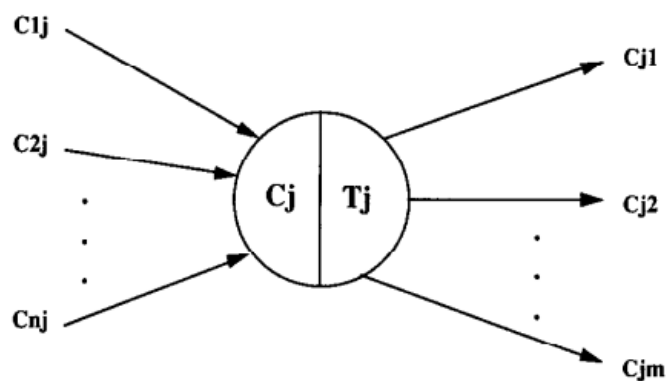


Fig. 2.2. Representación de la Neurona desarrollada por McCulloch-Pitts [14]

Ahora bien, ¿qué es una neurona? La neurona es la unidad básica de procesamiento de las redes neuronales. De forma similar al funcionamiento del cerebro humano, posee conexiones de entrada a través de las que recibe los valores externos. Con estos valores, las neuronas realizan los cálculos internos necesarios para proyectar un valor a la salida.

Sin embargo, se trata de una estructura aún muy simple, capaz solamente de trabajar con transformaciones lineales y que devuelve valores correspondientes a Verdadero o Falso, o lo que es lo mismo, 1 o 0.

El siguiente paso importante en la evolución hacia el aprendizaje profundo ocurre en los años 50. Tomando la neurona como base, Frank Rosenblatt desarrolla el Perceptrón [15], una red neuronal prealimentada (*feed-forward*) que introduce por primera vez el concepto de pesos¹ de entrada. En paralelo, aparece Adaline [16], desarrollado por Bernard Widrow, red capaz de adaptar los pesos de entrada mediante la suma ponderada en la fase de aprendizaje, siguiendo la estructura de la figura 2.3. También, aparece con Adaline el concepto de función de activación². Estos dos modelos sin embargo tuvieron poca popularidad debido a las limitaciones surgidas por la utilización de funciones lineales.

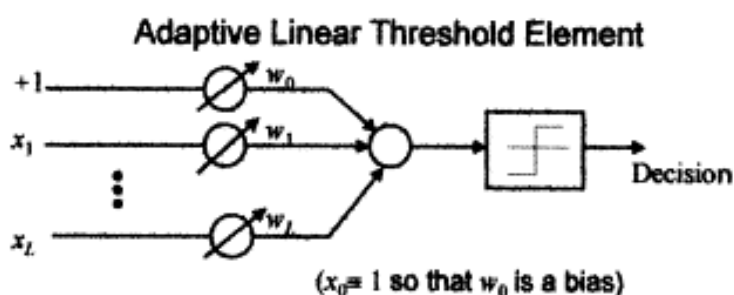


Fig. 2.3. Representación del modelo Adaline desarrollado por Bernard Widrow [16]

Décadas más tarde, gracias al desarrollo de otras ciencias, se cambia el enfoque del problema de los modelos neuronales. Las redes pasan en esta época a componerse de varias neuronas y la información atraviesa varias capas antes de obtenerse un resultado final. Así, sobre los años 80, comienza la Etapa del Conexionismo [17]. Este cambio de paradigma ocurre tras la aparición de las Redes Neuronales Artificiales o *Artificial Neural Networks* (ANN).

Este tipo de redes, ANN, presenta ya una estructura muy similar a la actual, representada en la figura 2.4. Las redes neuronales artificiales basan su funcionamiento en la interconexión de neuronas independientes, buscando conseguir una mayor abstracción y obtener modelos capaces de generalizar ante nuevas muestras. Con ellas, se introduce por

¹ Los pesos son parámetros encontrados en las redes neuronales que sirven para ajustar los datos de entrada a través de la red, de cara a obtener la salida adecuada. Estos definen la importancia de las conexiones entre neuronas, y por tanto, la influencia de las mismas en el resultado final.

² Las funciones de activación son funciones que se encuentran en las redes neuronales, encargadas de calcular la suma ponderada de las entradas y decidir a partir del resultado si la neurona debe activarse o no.

primera vez el concepto de capa oculta y se comienzan a desarrollar nuevos algoritmos como el de *Back Propagation* [18] (retropropagación), indispensables para el salto al periodo actual en el año 2006. Esta última época pasa finalmente a conocerse como como Etapa del Aprendizaje Profundo [19]. Esta rama de la inteligencia artificial basa sus modelos en la implementación de redes neuronales cada vez más y más profundas, es decir, con más capas.

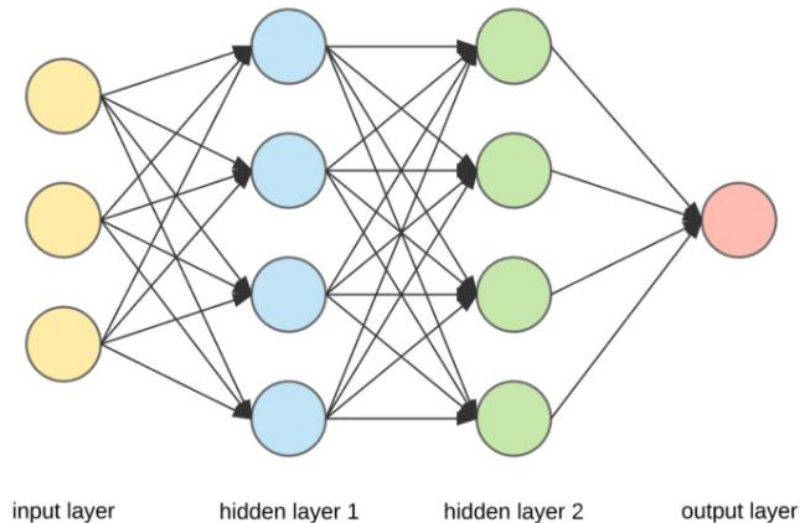


Fig. 2.4. Representación de la estructura de las redes neuronales artificiales [20]

2.2.2. Redes neuronales

El uso de redes neuronales ha ganado gran importancia debido a los buenos resultados obtenidos y su potencial de mejora. Esta metodología se caracteriza por inspirarse en la estructura y el funcionamiento del cerebro humano. Estas redes están compuestas por sucesiones de capas interconectadas entre sí. Cada una de dichas capas está construida con un número concreto de neuronas y la información se procesa pasando los datos a través de la red hasta alcanzar la última capa.

Consecuentemente, conociendo la estructura de las redes neuronales [21], se organizan sus capas en tres grupos: la capa de entrada, primera capa en la red que recibe los datos del exterior a procesar; las capas ocultas, capas intermedias cada vez más complejas encargadas del procesamiento de la información para extraer sus características y del aprendizaje; y finalmente, la capa de salida, donde se obtiene el resultado que previamente

ha sido procesado en las capas ocultas. En general, el número de neuronas en la última capa debe ser igual al número de posibles salidas que tiene el sistema.

Como se comentó anteriormente, la estructura básica de la red neuronal es la neurona. Cada neurona se encarga de la realización de los cálculos internos necesarios para proyectar un valor a la salida. Estos cálculos provienen de la realización de una suma ponderada con los valores recibidos y los pesos que se han asignado a cada una de las conexiones de entrada. Si se observa la ecuación 2.1 se presenta un modelo de regresión lineal en el que se pueden ajustar la inclinación y sesgo (bias) del hiperplano³ en función de los valores de entrada. Para ajustar este hiperplano a la distribución de los datos, será necesario aplicar transformaciones no lineales mediante funciones de activación.

$$f(X, W) = W_0 + X_1 \cdot W_1 + X_2 \cdot W_2 + \dots + X_n \cdot W_n$$

Ecuación 2.1. Suma ponderada realizada por la neurona [22]

Continuando con el concepto de función de activación [23] cabe remarcar la necesidad de su aparición: si en un sistema tan solo se usasen funciones lineales, éste colapsaría sobre sí mismo, ya que la concatenación de varias funciones lineales devuelve otra función lineal. Esto se correspondería con eliminar todas las neuronas y capas introducidas en la red, reduciéndose a una única función lineal resultante.

Existen diferentes tipos de funciones de activación dependiendo de la tarea a realizar y las características de la red. Las más utilizadas son las siguientes:

- **Sigmoide:** se trata de la opción más utilizada en la última capa de la red, ya que es capaz de transformar cualquier valor de entrada en uno de salida que puede variar entre 0 y 1. Esta característica la hace muy interesante para modelos que tengan que arrojar a la salida valores de probabilidad. Dada la forma que presenta, los valores de los extremos tenderán a las asíntotas 0 o 1. Los valores intermedios evolucionan de forma suave y convergen lentamente. Además posee características que la hacen interesante matemáticamente como la monotonía y la diferenciabilidad. Una variante es la función SoftMax, muy utilizada en modelos con múltiples clases ya que la salida también representa probabilidades, de tal manera que el sumatorio de todas ellas sea 1. Esta función presenta un crecimiento

³ Un hiperplano se puede definir como un sub-espacio lineal de un vector, caracterizado por poseer una dimensión menos que el vector que lo contiene.

más lento para valores de entrada pequeños, produciendo un desplazamiento hacia la derecha del punto medio de la curva. Ambas funciones de activación aparecen comparadas en la figura 2.5.

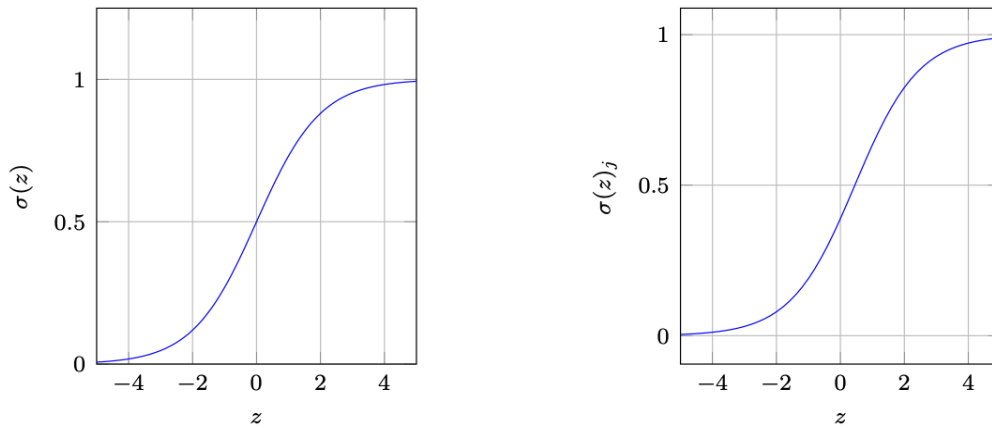


Fig. 2.5. Curva descrita por las ecuaciones de las funciones Sigmoide (izq.) y SoftMax (dcha.) [24]

- **Tangente hiperbólica:** funciona de forma similar a la sigmoide, pero en esta ocasión, la asíntota inferior se encuentra en -1 en lugar de en 0. Además, se encuentra centrada en el origen de coordenadas, como se aprecia en la figura 2.6. Sigue siendo monótona y diferenciable. Se suele utilizar especialmente en redes recurrentes y en modelos cuya salida trate de decidir entre una opción o la contraria.

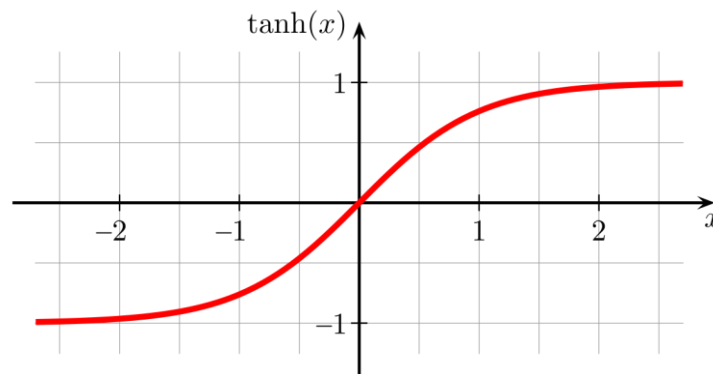


Fig. 2.6. Representación de la curva descrita por la función de activación tangente hiperbólica [25]

- **ReLU (Unidad Lineal Rectificada):** se trata de la función de activación más usada actualmente en el mundo de las redes neuronales. Esto se debe a que ofrece un funcionamiento adecuado frente a imágenes. Su gran potencial reside en la linealidad de la función en el eje de las x positivas, que no distorsiona el valor de entrada. A su vez, esto elimina los valores negativos y sólo se activa para los positivos, pudiendo tomar cualquier valor dado que la función no está acotada

superiormente. Su principal desventaja es que los valores negativos se transforman instantáneamente en 0, haciendo difícil el ajuste de cualquier modelo para los mismos. Su forma y ecuación aparecen en la figura 2.7.

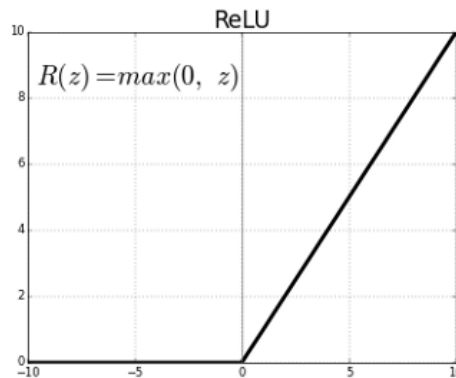


Fig. 2.7. Representación gráfica de la función ReLU junto con su ecuación correspondiente [26]

- **Leaky ReLU:** se trata de la función ReLU modificada ligeramente en el eje negativo, de cara a solucionar el problema comentado en el caso anterior. De esta forma, los valores menores que 0, quedan rectificadas en cierto factor. Generalmente, este valor es 0.01, aunque podrá variar, pasando a llamarse Parametric ReLU como aparece en la figura 2.8.

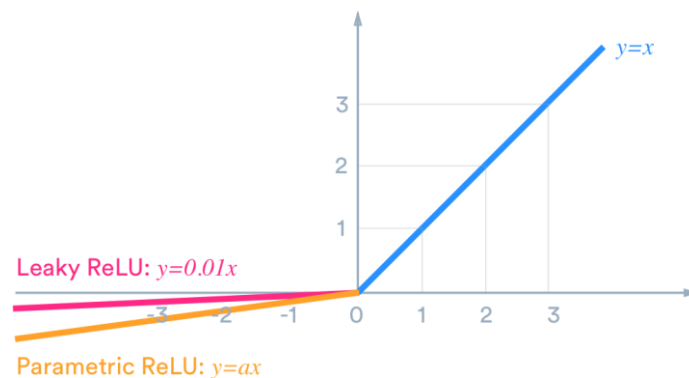


Fig. 2.8. Representación gráfica de las funciones Leaky ReLU y Parametric ReLU [27]

2.2.3. Redes neuronales convolucionales

Dentro de las múltiples posibilidades existentes en estructuras de redes neuronales, en el ámbito de la visión por ordenador han ganado protagonismo las Redes Neuronales Convolucionales o *Convolutional Neural Networks* (CNN) [28], especializadas en el procesamiento de imágenes. Estas siguen la estructura descrita en el apartado anterior,

pero con la particularidad de que alguna de las capas ocultas de la red realiza funciones muy específicas. Una de estas capas singulares es la capa convolucional, la cual otorga el nombre a este tipo de red.

El funcionamiento de las CNN [29] consiste en el análisis por regiones de la imagen de entrada. Estas regiones en las que se divide la imagen sufren transformaciones mediante la aplicación de filtros de convolución, que se van ajustando durante el entrenamiento con el objetivo de detectar patrones dentro de la imagen. La convolución permite al algoritmo reducir el número de parámetros, extrayendo los más relevantes. En las primeras etapas se pueden detectar las características de menor nivel, como son los bordes, los colores y los gradientes, y, a medida que se avanza dentro de la red, irá aumentando la complejidad de las detecciones realizadas en la imagen.

En el caso de redes destinadas al procesamiento de imágenes, donde además de las dos dimensiones espaciales se deben tener en cuenta los tres canales de color por cada píxel (rojo, verde, azul), será necesario aplicar un filtro específico por canal conocido como *kernel*. Este filtro es una matriz de tamaño menor al de la imagen de entrada con pesos establecidos para cada posición, que se superpone con la imagen original. La convolución se hace sumando la multiplicación de cada peso del *kernel* por el valor que tenga el píxel de la imagen en la misma posición. Finalmente se suman todos los valores de cada canal, se añade el término del sesgo y se guarda en la posición del mapa de características de la salida. Visualmente se puede apreciar en la figura 2.9.

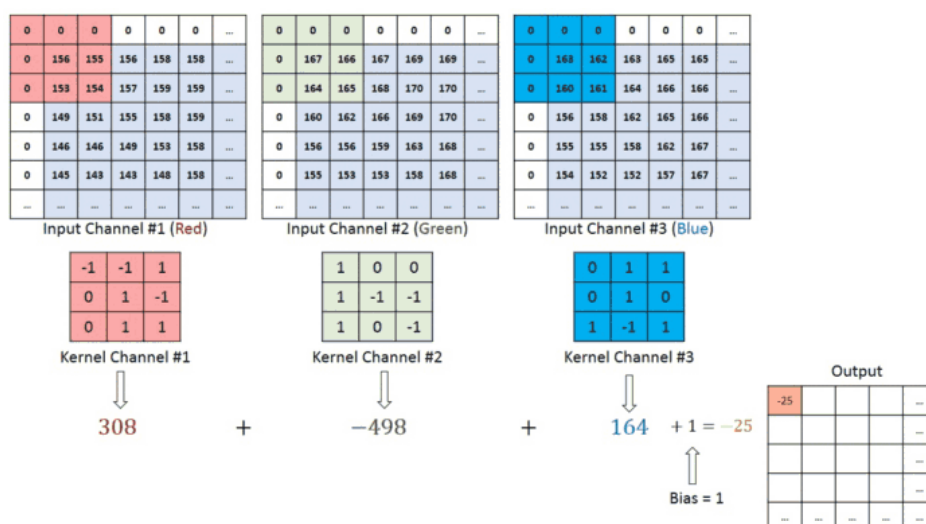


Fig. 2.9. Convolución en imágenes de entrada de tres canales (RGB) [29]

Como se ha mencionado, la imagen se procesa por regiones, siendo necesario desplazar el filtro de convolución. Este desplazamiento se produce horizontalmente, generalmente desde la esquina superior izquierda hasta la esquina inferior derecha. En esta operación intervienen dos parámetros esenciales: el paso y el relleno.

- **Paso (*stride*):** indica la distancia en píxeles que debe desplazarse el filtro convolucional sobre la imagen. En la figura 2.10 se representa el primer desplazamiento que realiza el filtro acorde al valor de paso. Como se puede comprobar, cuanto mayor sea el número asignado al paso, más pequeño será el mapa de características de salida. Aunque la intención del bloque convolucional sea la de reducir la imagen, un valor demasiado alto de paso produciría saltos en el mapa de características y se podría perder demasiada información de la imagen original. Por ello, es necesario marcar un valor de paso con el que se obtenga un balance entre reducción de resolución y baja pérdida de información.

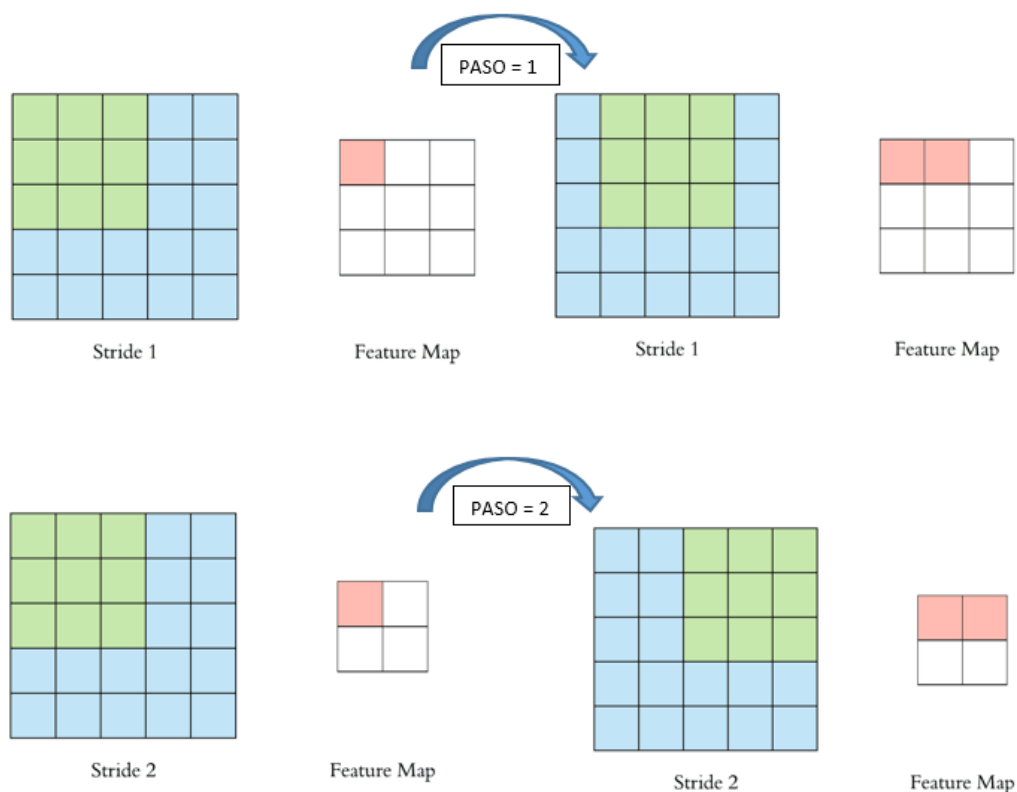


Fig. 2.10. Representación del desplazamiento del filtro convolucional según el valor de paso [29]

- **Relleno (*padding*):** se trata de un recurso usado con el objetivo de mantener el tamaño del mapa de características. Como se ha comentado, tras el proceso de convolución, el tamaño del mapa de características se ve reducido. Sin embargo, puede interesar mantener un tamaño constante. Con el relleno se añaden ceros

alrededor de la imagen de entrada, permitiendo la realización de un mayor número de desplazamientos del filtro y, por consiguiente, mayor resolución a la salida. El proceso seguido con el añadido del relleno se puede observar en la figura 2.11, donde la zona coloreada en gris se corresponde con los ceros.

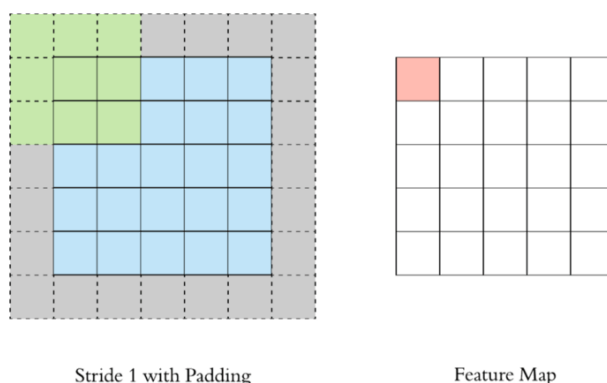


Fig. 2.11. Resultado obtenido al añadir relleno a la imagen [29]

Una vez planteado el funcionamiento de las CNN, se puede profundizar en su estructura. Tratándose de una red neuronal con un funcionamiento más específico, al igual que éstas, posee distintas capas. Dentro de la red, aunque las CNN reciben su nombre por la operación de convolución, existen en la jerarquía otro tipo de capas indispensables para el proceso de detección.

Un ejemplo de capas dentro de las CNN son las capas de agrupación (*pooling*), cuyo objetivo, al igual que la capa convolucional, es el de reducir el tamaño de la muestra convolucionada. En este caso se realiza una agrupación por regiones del mapa de características para simplificarlas a un único valor. Existen dos tipos de capas de agrupación: *Max Pooling*, donde el valor devuelto será el máximo de esa región y *Average Pooling*, devolviendo la media de los valores alcanzados en la agrupación. En términos generales, *Max Pooling* produce mejores resultados, ya que además de la reducción descrita, también cancela las activaciones producidas por ruido. Su funcionamiento y diferencias se pueden comprobar en la figura 2.12.

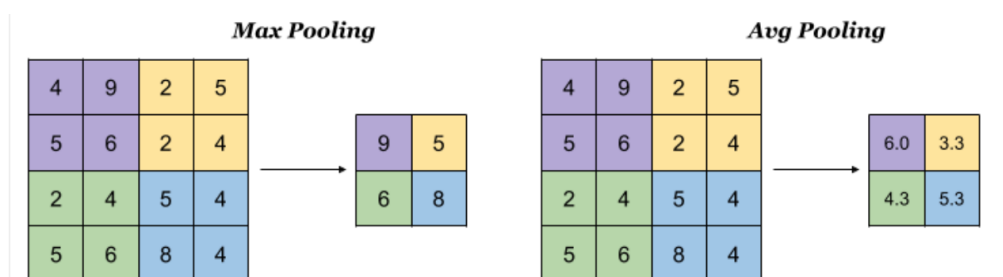


Fig. 2.12. Diferencias entre las salidas de Max Pooling y Average Pooling [29]

Otro ejemplo son las capas conocidas como completamente conectadas (*Fully-Connected Layer*), que suelen implementarse justo antes de la salida de la red. Estas capas son capaces de realizar una clasificación de la entrada en función de las características extraídas en las capas de *pooling* y convolución. Sin embargo, para su funcionamiento, es necesario aplanar la imagen de entrada, convirtiéndola en un vector columna con sus características más relevantes. Gracias a la información contenida en dicho vector, el modelo es capaz de distinguir las características más relevantes para realizar finalmente el proceso de clasificación. La estructura típica de una CNN se puede ver en la figura 2.13.

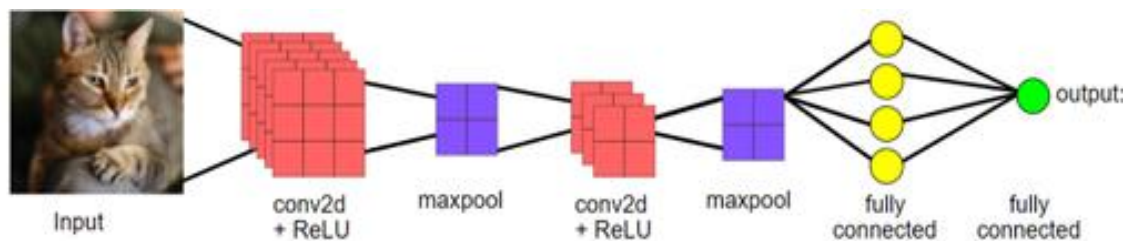


Fig. 2.13. Estructura final de una red neuronal convolucional

3. DETECCIÓN DE OBJETOS

En este tercer capítulo se describe la tecnología existente en el ámbito de la detección de objetos, una rama de la visión artificial encargada del procesamiento de imágenes. Además, se explican brevemente los principales algoritmos existentes en este campo y se hace una comparativa de ellos de cara a la elección del algoritmo a utilizar en el sistema.

3.1. Introducción

En ocasiones se suelen confundir los términos de clasificación de imágenes con detección de objetos (*Object Detection*) [30]. La clasificación tiene como objetivo la asignación de la etiqueta correspondiente a una imagen en función del objeto que contiene. En el ejemplo de la figura 3.1, la imagen de la izquierda contiene únicamente un gato, y esta será la etiqueta que se le asigne. Por el contrario, la detección de objetos se encarga de la localización del objeto dentro de la imagen junto a la clasificación en la clase correspondiente. La gran ventaja de la detección de objetos reside en la capacidad de detección de múltiples clases simultáneamente, como se puede comprobar en la imagen de la derecha en la figura 3.1.

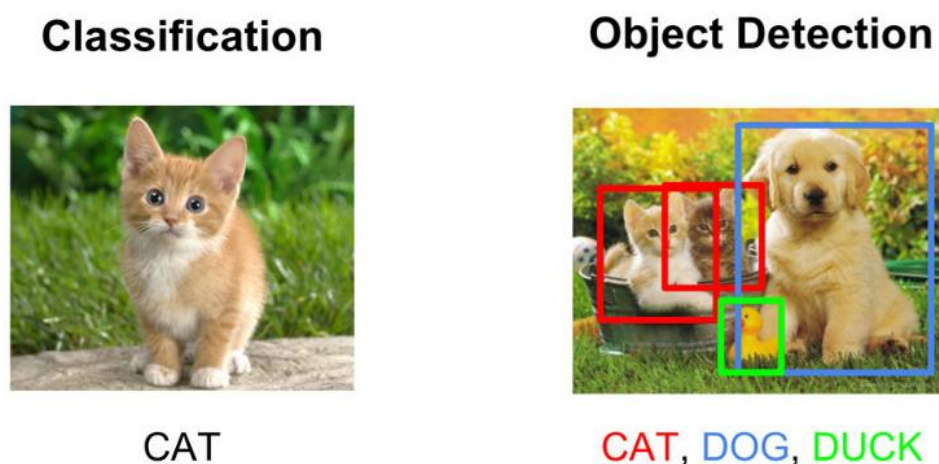


Fig. 3.1. Diferencias entre clasificación de imágenes y detección de objetos [31]

Para el prototipo que se quiere implementar es necesario conocer la posición de los objetos detectados dentro de la imagen, por lo que se ha decidido usar esta segunda tecnología. Como se puede ver, se hace uso de rectángulos delimitadores (*bounding boxes*) para indicar la posición del objeto. Ahora bien, existe una variante de este método

que es capaz de reconocer las siluetas y aplicar una máscara sobre el objeto. Esta tecnología es conocida como segmentación de instancias (*Instance Segmentation*) y la base de su funcionamiento es la clasificación de cada píxel de la imagen en su clase correspondiente. De esta forma, el salto de una clase a otra se corresponderá con el borde del objeto. Ambas tecnologías aparecen comparadas en la figura 3.2. El uso de esta segunda técnica, es decir, la segmentación de instancias, no es recomendable para nuestro objetivo, ya que los algoritmos utilizados necesitan mayor tiempo de procesamiento y haría imposible la detección en tiempo real durante la conducción.

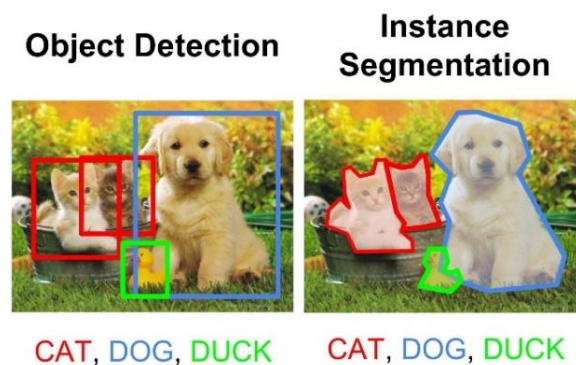


Fig. 3.2. Diferencias entre detección de objetos y segmentación de instancias [31]

3.2. Algoritmos principales

Actualmente existen dos tipos de estructuras diferentes en los algoritmos de detección de objetos. El primer funcionamiento se basa en algoritmos de dos fases en los que, en primera instancia, se generan diferentes rectángulos con posibles detecciones para después ser clasificadas a través de redes neuronales para obtener la detección final. El segundo tipo de algoritmo consta de una única fase en la que la división en regiones y el paso de estas a través de la red neuronal se realizan en una única instancia. Estos algoritmos, dado que simplifican su estructura respecto a los de dos fases, darán en general resultados con menor precisión pero reduciendo considerablemente el tiempo de ejecución.

En este epígrafe se procede a la explicación de los principales algoritmos existentes en el campo de la detección de objetos, pertenecientes a ambos tipos de estructura. Estos son:

- **R-CNN:** este tipo de redes [32] basa su funcionamiento en la división de la imagen de entrada mediante fuerza bruta en múltiples regiones donde pudieran

encontrarse los objetos de mayor interés. Con todas las regiones propuestas se inicia un proceso de agrupamiento de aquellas similares. Cada región final se procesa por separado introduciéndola en una red neuronal convolucional. Este proceso da como resultado un vector de características correspondiente con la entrada suministrada a una SVM (*Support Vector Machine*) encargada de la detección de los objetos en las regiones propuestas. La estructura queda detallada en la figura 3.3.

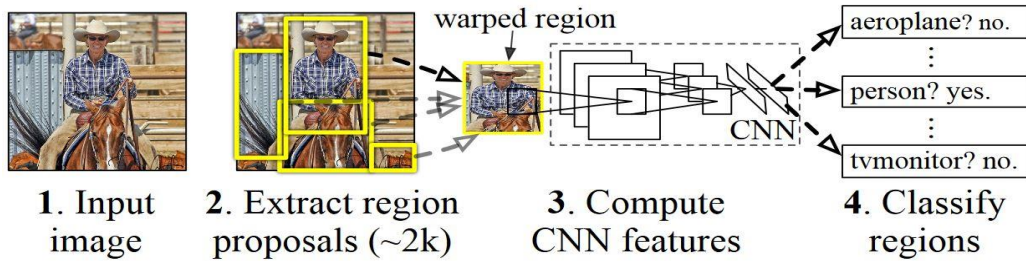


Fig. 3.3. Estructura y funcionamiento de la red R-CNN [32]

- **Fast R-CNN:** este algoritmo [33] aparece años más tarde como evolución del algoritmo R-CNN, por parte del mismo creador. Con el fin de agilizar el procesamiento se descartó el uso de las regiones de interés por fuerza bruta al comienzo del proceso, cambiando a un modelo cuya CNN recibía la imagen de entrada original. Tras su paso por la red se obtenía un mapa de características del que se obtenían las propuestas de regiones. Finalmente, se hacían pasar estas regiones por una capa de *pooling* y otra *SoftMax* para obtener las probabilidades de cada clase. La nueva estructura aparece presentada en la figura 3.4. Dado que no se procesan en un principio todas las regiones de interés como sí ocurría con R-CNN, el tiempo de entrenamiento se vio reducido drásticamente.

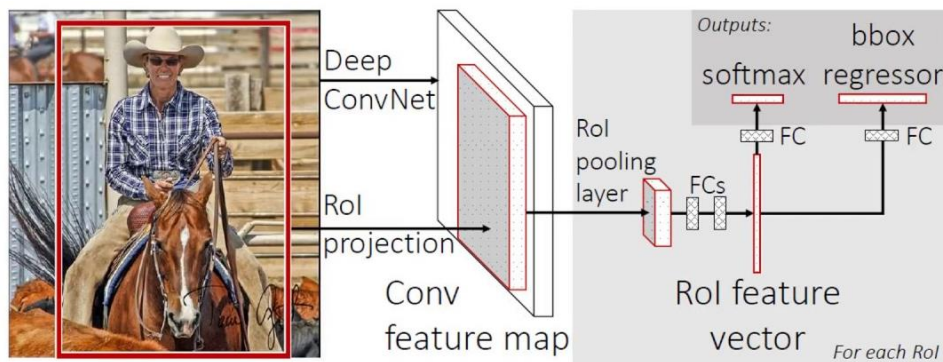


Fig. 3.4. Estructura del modelo Fast R-CNN [33]

- **Faster R-CNN:** como última evolución [34] en esta familia de algoritmos llega Faster R-CNN, que rebaja aún más el tiempo de procesado. Presenta un cambio fundamental en su estructura, sustituyendo la CNN inicial por una red de propuesta de regiones o *Region Proposal Network* (RPN). Con la nueva red se elimina la búsqueda selectiva de regiones, pasando a proponerlas directamente. Esta nueva estructura es capaz de realizar la detección de los objetos junto con su probabilidad, en un único paso. Como se puede observar en la figura 3.5, el tiempo de procesamiento se ha ido reduciendo considerablemente con las evoluciones.

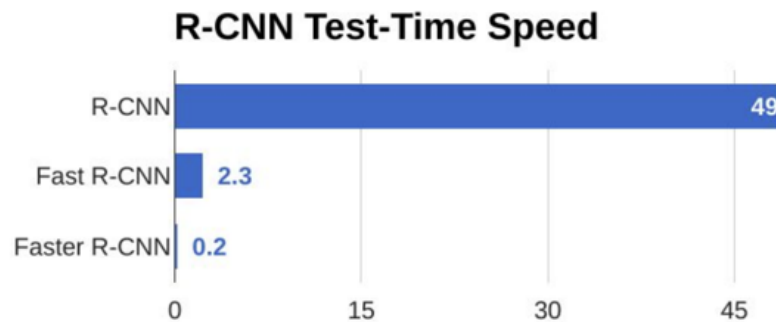


Fig. 3.5. Comparación entre los tres modelos de la familia R-CNN [35]

- **RetinaNet:** se trata de un modelo de una única etapa, compuesto por dos redes unificadas [36]: una primera parte compuesta por una red neuronal piramidal y una segunda, que utiliza una subred de clasificación. El sistema hace uso de cajas delimitadoras preestablecidas conocidas como cajas de anclaje (*anchors*). Estos rectángulos delimitadores pasan a través de una subred de regresión de cajas proyectando a la salida el cuadrado final sobre el objeto, acompañado de la probabilidad de la clase a la que pertenece. La estructura de la red se representa en la figura 3.6.

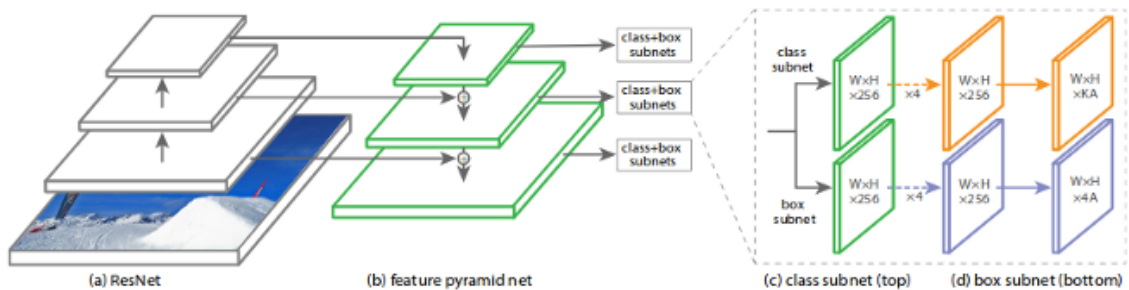


Fig. 3.6. Representación de la estructura del modelo RetinaNet [37]

- **Single Shot Detector (SSD):** se trata de un algoritmo de detección de objetos basado en una única fase. Este algoritmo [38] elimina el proceso de proposición

de regiones, aplicando en su lugar cajas delimitadoras predefinidas a diferentes escalas. Comienza utilizando la red convolucional VGG16 permitiendo la extracción del mapa de características usando celdas. En cada una de las celdas se produce la detección y el cálculo de la probabilidad de cada clase mediante pequeños filtros convolucionales. La salida de cada filtro posee 25 canales, 20 para las probabilidades de las clases, otro reservado para el caso de no contener ningún objeto, y los últimos 4 para las dimensiones del rectángulo delimitador. Dado que los objetos suelen tener una relación de tamaño constante, por ejemplo los autobuses son más largos que altos y los viandantes al contrario, se definen varias cajas de distintos tamaños sobre los posibles objetos de interés. Mediante un algoritmo de agrupación (*clustering*) se eligen los tamaños de las cajas que más se adapten para finalmente, agruparlas en una única caja que se obtendrá a la salida.

Para poder detectar distintas clases independientemente, SSD incorpora 6 capas convolucionales más para generar mapas de características de menor resolución en los que aparecen los objetos más grandes. El resultado de la estructura del modelo sería el representado en la figura 3.7.

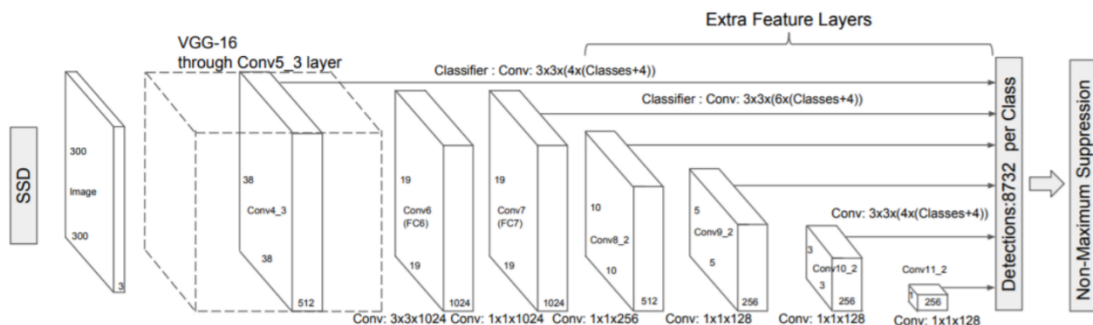


Fig. 3.7. Estructura de la red utilizada por Single Shot Detection [38]

- **You Only Look Once (YOLO):** este algoritmo [39] presenta una estructura similar a la descrita en el algoritmo SSD mediante una única etapa de detección y clasificación. El modelo se compone de capas convolucionales, que permiten reducir considerablemente la resolución de las muestras de entrada. El mapa de características se hace más pequeño en cada paso, extrayéndose los rasgos más significativos de los objetos a detectar. La imagen es visible en todo momento, eliminando el uso de regiones de interés y, por lo tanto, reduciendo el tiempo de procesamiento. Este algoritmo se explica en mayor profundidad en el capítulo 5.

4. ELECCIÓN DEL ALGORITMO

Una vez expuestos los modelos de visión artificial más importantes, se va a realizar en este cuarto capítulo una comparativa con el objetivo de la selección del algoritmo con el que se obtengan mejores prestaciones en rapidez de procesamiento y precisión en la detección.

Uno de los principales retos del procesamiento de imágenes es el poder obtener tiempos de ejecución suficientemente pequeños. Por ello, es necesario implementar un algoritmo que requiera poco tiempo para analizar las imágenes, pudiéndose realizar la detección a medida que se introducen los datos en la red. De esta manera se podrá procesar cada fotograma casi a tiempo real, lo cual es indispensable en vehículos en circulación, ya que durante el tiempo de análisis de cada fotograma se recorren varios metros en los que el entorno puede cambiar.

Existen numerosas técnicas y modelos capaces de llevar a cabo la tarea de reconocimiento de objetos, descritos en el capítulo anterior. Entre los algoritmos de aprendizaje computacional existen algunos que son capaces de detectar formas y siluetas y, a partir de ello, clasificar el objeto en alguna de las clases. Sin embargo, presentan problemas a la hora del reconocimiento de formas cambiantes y, difícilmente, son capaces de abstraer y extrapolar los conocimientos adquiridos a nuevas entradas no vistas anteriormente. Por tanto, dada la variabilidad existente en el entorno de la conducción, se ha descartado este tipo de soluciones para la implementación a realizar.

Con objeto de poder desarrollar el sistema que se plantea en este TFG, se va a elegir el algoritmo que mejor resultados arroje dentro del campo del *Deep Learning* y más concretamente, aquellos diseñados específicamente para la tarea de detección de objetos. Como se ha comentado anteriormente, existen dos grandes grupos estructurales en este ámbito: modelos de una o de dos etapas [40].

El grupo definido por los modelos de dos etapas incluye los algoritmos pertenecientes a la familia R-CNN (R-CNN, Fast R-CNN y Faster R-CNN), junto con su versión dedicada a la segmentación, Mask R-CNN. Este tipo de algoritmos se caracteriza por su robustez y su precisión, pero el hecho de tener que procesar la imagen cientos de veces hace que sean muy pesados computacionalmente y que el tiempo de procesado de cada fotograma sea muy lento.

Por otro lado, el grupo con una única etapa de localización y clasificación procesa la imagen una única vez, siendo visible al completo por la red en todo momento. Gracias a esto, a la hora de realizar la detección, el sistema posee además un cierto contexto que facilita la labor de clasificación. También en este caso es posible obtener buenos resultados con una menor resolución en la imagen de entrada. De esta forma, no son necesarias cámaras de una gran resolución para poder construir un sistema de conducción autónoma. Aun así, esa ventaja que se obtiene en rapidez computacional queda sacrificada por una menor precisión en la detección. En este campo destacan los sistemas de RetinaNet, SSD y YOLO.

Teniendo en cuenta las ventajas y desventajas de los algoritmos propuestos, hay que realizar un balance para seleccionar un algoritmo cuya rapidez y precisión sean lo suficientemente buenas. Para ello se ha realizado una comparación entre todos con el fin de comprobar el rendimiento de cada uno bajo las mismas circunstancias. La comparación de prestaciones sobre precisión, aparece en la tabla 4.1. Para la prueba se ha utilizado una tarjeta gráfica Nvidia Titan X y se ha entrenado con el mismo conjunto de datos, COCO [41].

	Sistema	Backbone	AP	AP ₅₀	AP ₇₅
Dos etapas	Faster R-CNN +++	ResNet-101-C4	34.9	55.7	37.4
	Faster R-CNN con FPN	ResNet-101-FPN	36.2	59.9	39.0
	Faster R-CNN con TDM	Inception-ResNet-TDM	36.8	57.7	39.2
Una etapa	SSD	ResNet-101-SSD	31.2	50.4	33.3
	RetinaNet	ResNet-101-FPN	39.1	59.1	42.3
	YOLOv3	Darknet-53	33.0	57.9	34.4

Tabla 4.1. Comparativa de los valores de precisión media para los diferentes algoritmos [42]

En la tabla se comparan los valores promedio de precisión. Como se puede comprobar, los mejores resultados en todos los campos se obtienen para el modelo de RetinaNet. Sin embargo, si se analiza la gráfica mostrada en la figura 4.1, donde además de la precisión se muestra la velocidad de procesamiento en el eje de abscisas, podemos observar que YOLO destaca por delante del resto con un tiempo de procesamiento de tan solo 22 ms por fotograma.

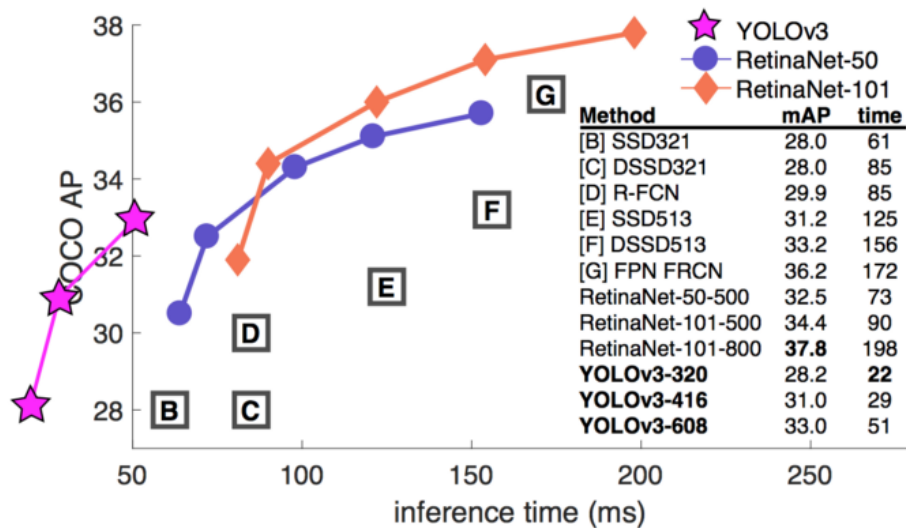


Fig. 4.1. Gráfica comparativa entre precisión y tiempo de procesamiento para los algoritmos [42]

Hay que tener en cuenta que los resultados representados en cada una de las figuras han sido obtenidos bajo configuraciones distintas. Esta es una de las grandes dificultades que aparecen a la hora de realizar una comparación entre los diferentes modelos: no hay resultados estandarizados.

A partir de la información recopilada, la elección del algoritmo se ha realizado bajo dos criterios. El primero, y más importante, ha sido el tiempo de procesamiento de imágenes. Es indispensable que este valor sea lo más bajo posible para poder realizar la detección de objetos en tiempo real. Bajo este criterio, los mejores resultados se obtienen para el algoritmo de YOLO. Si atendemos al segundo criterio utilizado, la precisión, se puede comprobar que existe una gran variabilidad en función de la configuración utilizada dentro de cada algoritmo. Sin embargo, los valores obtenidos no presentan grandes diferencias (como mucho, alrededor de 10 puntos de precisión). Asimismo, la precisión en el prototipo a implementar no se considera de vital importancia, ya que los sistemas de ayuda a la conducción además incorporan elementos como sensores y radares que permiten mejorar sus prestaciones.

Por consiguiente, el algoritmo elegido para la implementación en el sistema será YOLO, y más específicamente, la versión 5 recientemente desarrollada, con la que se obtienen los mejores resultados de velocidad de procesamiento y precisión. El código del algoritmo seleccionado puede encontrarse en el repositorio de YOLO Ultralytics [43], estando implementado para PyTorch y la versión más reciente de Python.

5. ALGORITMO YOLO

En este capítulo se explica en profundidad la estructura y funcionamiento del algoritmo escogido: You Only Look Once (YOLO). Además, se describen las características y mejoras de cada una de las cinco versiones del algoritmo hasta el modelo actual.

5.1. Estructura de YOLO

El algoritmo escogido, You Only Look Once [39], pertenece al grupo de modelos de detección de objetos en el que el procesamiento se realiza en un único paso. Su particularidad frente a otros modelos pertenecientes a este grupo, y por la que se ha seleccionado, reside en que este algoritmo es capaz de obtener una precisión similar a la del resto con una velocidad de procesamiento mucho mayor.

Dado que existen varias versiones del algoritmo YOLO, y cada de ellas posee una estructura diferente, en este apartado no se describirá el funcionamiento detallado de cada una. Esta explicación se realizará en el epígrafe 5.3, junto con la de las diferencias y mejoras entre versiones. En este caso, se procederá a la descripción de la última estructura conocida del modelo: la cuarta versión del algoritmo, ya que la quinta se encuentra todavía en desarrollo y no se ha publicado.

En primer lugar, es necesario comprender que, aunque se conozca como un algoritmo de una única etapa, ésta a su vez se divide en tres partes. La entrada de la red se conoce como columna vertebral o *backbone* y es la encargada de la extracción de características. La segunda región es conocida como el cuello y actúa de nexo entre la etapa de entrada y la de salida. Esta última parte, la etapa de salida, es conocida como cabeza de la red y es la encargada de la detección final. Esta estructura se presenta en la figura 5.1.

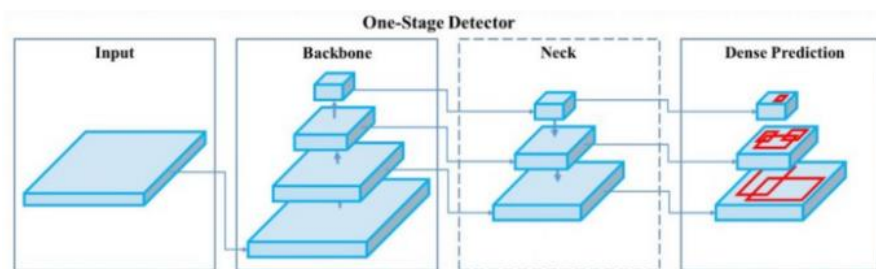


Fig. 5.1. Estructura de los detectores de una única etapa [44]

En la primera etapa de YOLO se utiliza la red conocida como CSPDarknet-53 [44]. Es por esto que muchas veces se encontrarán referencias a esta red como el *backbone* de YOLO. La red tiene una estructura que hace uso de bloques densos para expandir el campo de recepción y aumentar la complejidad del modelo. Cada bloque denso está constituido por capas convolucionales en las que se utilizan todas las salidas de las capas anteriores para aumentar el número de mapas de características. Su estructura se puede apreciar en la figura 5.2.

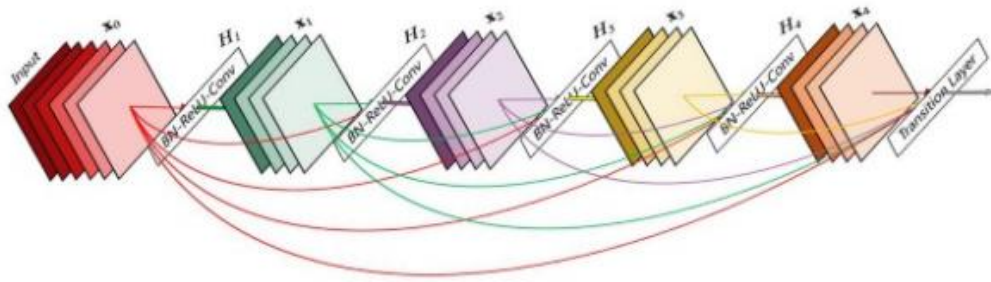


Fig. 5.2. Estructura de capas en los bloques densos [45]

A continuación, a la salida de estos bloques, se pasan los datos por el módulo CSP (*Cross Stage Partial Connections*), donde se separan en dos partes para reducir su complejidad computacional. Finalmente, se introducen estos datos en la red Darknet-53 para la extracción de características. Además de todo esto, se ha incorporado a la salida de esta red una capa SPP (*Spatial Pyramid Pooling*) encargada de la separación de los mapas de características más relevantes. La integración de todos los módulos comentados se puede ver en la figura 5.3.

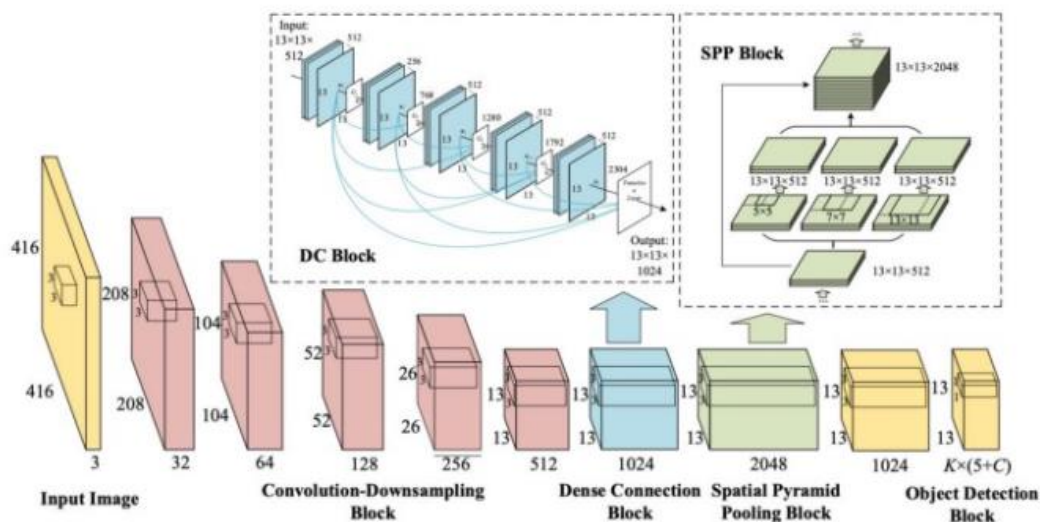


Fig. 5.3. Estructura de la etapa de columna vertebral de YOLO [46]

En la segunda etapa se atraviesa el cuello de la red neuronal, donde se ha incorporado una red de agregación de caminos (*Path Aggregation Network*, PAN). En la estructura de la PAN existe un camino de subida encargado de la propagación en las capas superiores de las características de las capas de menor nivel. Asimismo, posee otros atajos que conectan las primeras capas con otras más avanzadas. Su estructura se muestra en la figura 5.4.

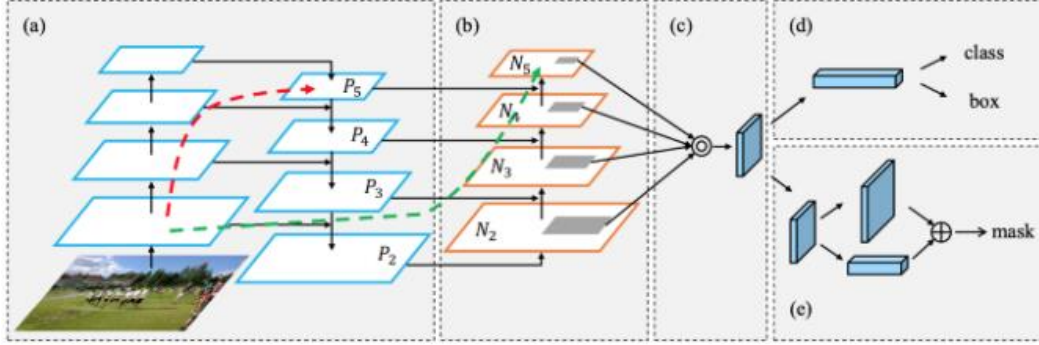


Fig. 5.4. Estructura y caminos seguidos por las imágenes en el módulo PAN [47]

En última instancia, los datos que han ido atravesando la red llegan a la cabeza. Es en este paso en el que finalmente se produce la detección de los objetos presentes en la imagen.

Cabe destacar que gracias a esta estructura, el algoritmo es capaz de ver la imagen de entrada al completo durante todo el proceso, a diferencia de otros algoritmos basados en ventanas deslizantes o regiones de interés donde la detección se realiza únicamente sobre ciertas partes de la imagen. Como resultado, YOLO posee una mayor capacidad de codificación de información contextual extra sobre cada una de las clases.

5.2. Funcionamiento del algoritmo

Durante la detección existen varios procesos que permiten el funcionamiento del sistema [48], reduciendo el problema de la detección de objetos a un simple problema de regresión.

En primer lugar, se divide la imagen de entrada en una cuadrícula de $M_1 \times M_2$ celdas que podrán predecir hasta un máximo de B cuadros delimitadores cada una, además de un valor de confianza P para estos delimitadores y un número C de probabilidades para las clases. De esta manera, las predicciones serán codificadas como un tensor de dimensiones $M_1 \times M_2 \times (B \cdot (5 + C))$. Hay que tener en cuenta que un mismo objeto puede abarcar

varias celdas, por lo que será aquella en la que caiga el centro del rectángulo la encargada de realizar la detección.

El algoritmo de localización en cada celda funciona de la siguiente manera:

- En primer lugar, es necesario obtener las coordenadas t_x y t_y del centro del objeto detectado. Además, se debe calcular el par t_w y t_h que indican el ancho y el alto del rectángulo respectivamente. Todos estos valores deben ser normalizados entre las dimensiones de entrada, por lo que t_x y t_y siempre estarán entre los valores 0 y 1, mientras que t_w y t_h pueden superarlos en caso de no entrar el objeto en la celda.
- A continuación, se añade por utilidad un indicador de confianza P que muestre cómo de probable es que esté el objeto dentro de esa imagen. El rango de valores que puede tomar va de 0 a 1, donde el mínimo indicará que el modelo está seguro de que el objeto no se encuentra en la imagen y el máximo, por el contrario, asegurará que se encuentra dentro de ella.
- Por último, se necesitará clasificar el objeto en una de las N clases posibles. Para codificar esta información se hará uso de la función *one hot encoding*, cuyo propósito es devolver un vector de tantos ceros como clases haya y colocar un 1 en la posición que se corresponde con la clase detectada. Todos estos valores se concatenan en un único vector como se muestra en la figura 5.5.

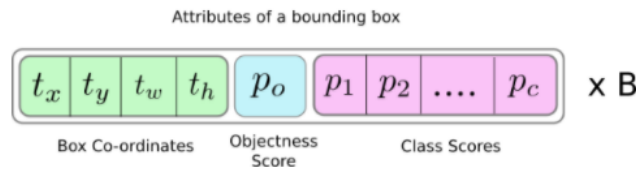


Fig. 5.5. Estructura del vector de salida del algoritmo de detección de YOLO [48]

De esta manera, se define el algoritmo de forma que cada celda únicamente puede detectar una clase. El problema de este procedimiento surge cuando en una misma celda aparecen distintos objetos, ya que un sistema con la estructura descrita no podrá detectarlos todos. Además, en ocasiones estos objetos quedan superpuestos en la imagen, y los que se encuentran detrás quedan ocultos tras los objetos más cercanos. Esta situación complica su detección. Para solucionar los conflictos ante los que se encuentra el funcionamiento del algoritmo, se define el sistema las cajas de anclaje (*anchor boxes*).

Las cajas de anclaje pueden definirse como un conjunto de cajas delimitadoras con una altura y anchura específicas [49]. Estas cajas capturan ratios del tamaño definido

previamente para cada clase, mediante los cuales se puede abarcar el objeto que ha quedado oculto en la celda. Las orientaciones y tamaños de las cajas varían de acuerdo con el objeto, y van asociadas a una probabilidad que indica cómo de probable es que un objeto entre en esa caja de anclaje. Sobre una imagen quedarían superpuestas como en la figura 5.6.

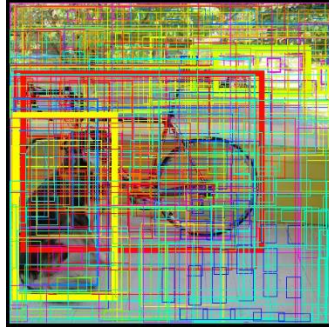


Fig. 5.6. Representación de las cajas de anclaje sobre una imagen [50]

A partir de las cajas de anclaje, se obtiene más tarde el tamaño del rectángulo delimitador (*bounding box*) de la detección. Sus dimensiones vendrán dadas por una transformación de las dimensiones que la red neuronal ha devuelto previamente. La transformación realizada sigue la estructura de las funciones presentes en la figura 5.4.

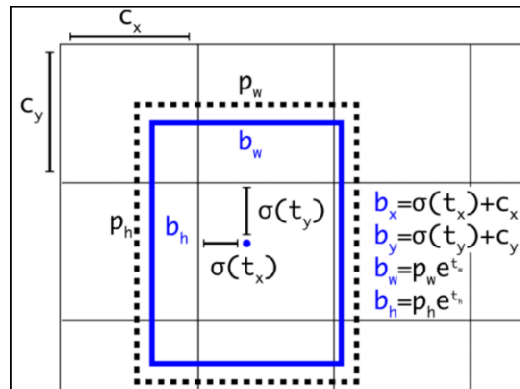


Fig. 5.7. Cálculo de las dimensiones de las cajas delimitadoras [51]

Tras añadir las cajas de anclaje, se obtiene un vector de predicción donde se proponen 5 parámetros fijos: el centro (x, y) del recuadro detectado, el ancho y el alto del mismo y su indicador de confianza. A esto hay que sumarle el número total de clases del sistema, C , por lo que obtenemos una dimensión de $(5 + C)$. Por último, como por cada celda podremos detectar varios objetos, el tamaño final del vector será $B \cdot (5 + C)$, donde B se correspondía con el número de cajas delimitadoras por celda.

Para solucionar el problema de las cajas de predicción superpuestas se aplica un proceso de depurado. Con este proceso se irán eliminando aquellas cajas cuya probabilidad de predicción P sea menor a un umbral definido. Este es el primer paso que sigue el algoritmo conocido como *Non-Maximum Suppression* (NMS) [52]. Aun así, tras la depuración, pueden quedar recuadros superpuestos, haciendo necesario la selección del que posea mayor probabilidad de detección. El resto de rectángulos cuya Intersección Sobre la Unión (IoU) respecto a la caja seleccionada sea mayor que otro segundo umbral, serán también eliminados. De esta forma se garantiza siempre la selección del recuadro con mayor probabilidad, eliminando predicciones similares. Este proceso se repite nuevamente tantas veces como sea necesario hasta que quede una única caja, correspondiente con la detección final.

Resumiendo, la imagen atraviesa la red buscándose todas las posibles detecciones, y aplicando diversos algoritmos de cara a reducir el número de las mismas. Así se obtiene finalmente una imagen en la cual se detectan todos los objetos presentes que pertenezcan a la selección, sin importar superposiciones, y evitando duplicados. Este funcionamiento queda recogido gráficamente en la figura 5.8.

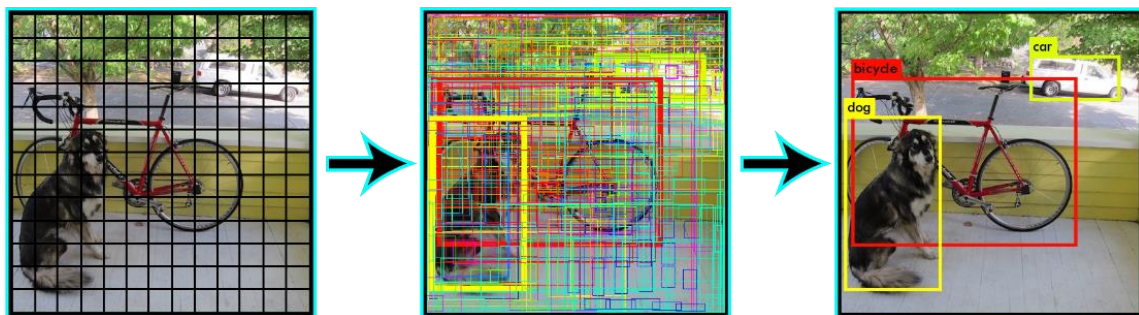


Fig. 5.8. Proceso seguido para la detección de objetos en el algoritmo YOLO [50]

5.3. Evoluciones de YOLO

Desde la publicación de la primera versión de YOLO en mayo de 2016, el algoritmo ha despuntado sobre el resto de métodos de visión artificial. Es por ello que existe un afán por la mejora de este modelo desde su aparición, con la existencia hasta el momento de 5 versiones.

5.3.1. YOLO v1

En su primera versión [53], estructuralmente se inspiró en el modelo GoogleNet con 24 capas convolucionales y dos capas completamente conectadas. Cada una de sus capas ocultas utiliza la función de activación ReLU, mientras que la capa final utiliza una función lineal. Aunque haya sido entrenado con imágenes de 224x224 píxeles, acepta como entrada para la detección, imágenes de tres canales con una resolución de 448x448 píxeles. Esta configuración, junto a su estructura en una única fase, ofrece un alto rendimiento, consiguiendo el procesamiento de imágenes en tiempo real a 40 fps.

Una de las grandes ventajas que presentaba este algoritmo frente al resto es la alta capacidad de aprendizaje de características de objetos y la posibilidad de detección de varias instancias. Gracias a ello, el sistema posee un alto nivel de precisión, junto con una disminución de los falsos positivos. Sin embargo, posee una alta tasa de errores en la localización frente a sus competidores basados en propuestas de regiones.

5.3.2. YOLO v2

En su segunda versión [51] lanzada en diciembre de 2016, Joseph Redmon y Ali Farhadi, desarrolladores de la primera versión del algoritmo, proponen una red que puede ser entrenada para la detección y clasificación con el fin de mejorar los resultados obtenidos en localización, manteniendo la precisión. Se puede ver la comparación con otros algoritmos en la figura 5.9. Esta versión también fue conocida como YOLO 9000.

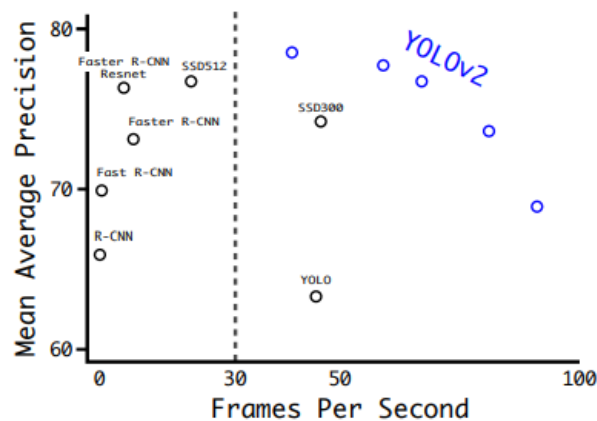


Fig. 5.9. Comparativa de YOLOv2 en términos de velocidad y precisión [51]

Estructuralmente, en esta ocasión se reduce el número de capas convolucionales a 23 y cambia la función de activación utilizada en las capas internas a Leaky ReLU, manteniendo la función lineal a la salida de la red.

Se introducen además mejoras en el funcionamiento como la normalización de los valores de entrada. De esta forma se regula y estabiliza el modelo, reduciendo el tiempo de entrenamiento y llegando a aumentar la precisión en dos puntos. También cambia el entrenamiento de la red que anteriormente se hacía con una resolución de 224x224. En esta ocasión se entrena directamente con imágenes de 448x448, evitando así el cambio de escala entre el entrenamiento y la detección producido en la versión 1. Con esta modificación se obtiene un aumento de 4 puntos en la precisión media.

Otra nueva característica que aparece en esta segunda versión son las cajas de anclaje mencionadas en epígrafes anteriores. Aunque a priori el uso de estas regiones supone una ligera disminución en la precisión, el valor de cobertura (*Recall* en inglés) aumenta en mayor medida. Para solucionar esa caída en precisión, se usa un algoritmo de agrupamiento para la búsqueda de las dimensiones adecuadas de las cajas.

5.3.3. YOLO v3

No será hasta abril de 2018 cuando aparezca una nueva versión del algoritmo [54]. Los mismos desarrolladores de los modelos anteriores consiguen posicionarlo como una tecnología puntera en el ámbito de la detección en tiempo real puesto que con esta evolución se mejoran las prestaciones de precisión y rapidez como se aprecia en la figura 5.10.

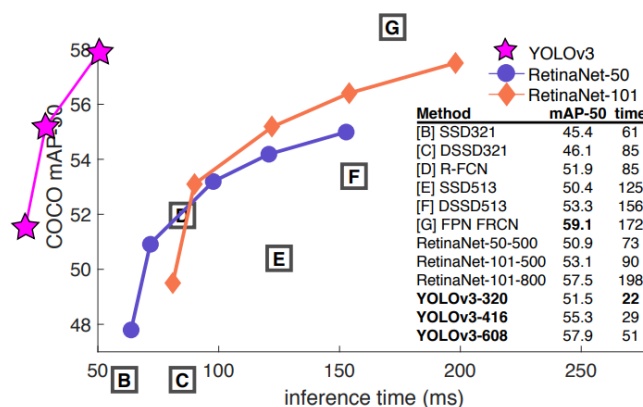


Fig. 5.10. Comparativa de YOLOv3 en términos de velocidad y precisión [54]

Su estructura interna cambia drásticamente pasando a tener 106 capas, de las cuales 75 son capas convolucionales. Además, para la extracción de características se utiliza la CNN Darknet-53 que hará la función de columna vertebral del modelo.

Dentro de las mejoras introducidas destaca el método de obtención de los rectángulos delimitadores. En este caso, se asigna a cada uno de los posibles candidatos un valor de objetividad calculado mediante regresión logística. Sumado a esto, el ancho y alto se calcularán a partir del desplazamiento respecto a los centroides de cada *cluster*. Por otro lado, el centro de la caja se obtiene haciendo pasar estos valores por una función sigmoide.

Finalmente, destaca en su funcionamiento el uso de redes piramidales (*Feature Pyramid Network*, FPN) mediante las cuales se consigue la detección de objetos a diferentes escalas. En concreto, esta estructura permite realizar detecciones en los mapas de características a diferentes profundidades dentro de la red. Su potencial radica en la posibilidad de aumentar el mapa de características uniendo el de la capa actual con las características extraídas en capas futuras. De esta forma la red es capaz de contrastar ambos mapas para realizar una predicción más precisa.

5.3.4. YOLO v4

Durante el desarrollo de la versión 4, Joseph Redmon abandona el proyecto debido al posible mal uso que se podría dar a esta tecnología, sobre todo en aplicaciones militares. Es por ello que la aparición de la nueva versión [44] no ocurrirá hasta 2 años más tarde, durante abril de 2020, gracias a la investigación de Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao. Las prestaciones obtenidas se aprecian en la figura 5.11.

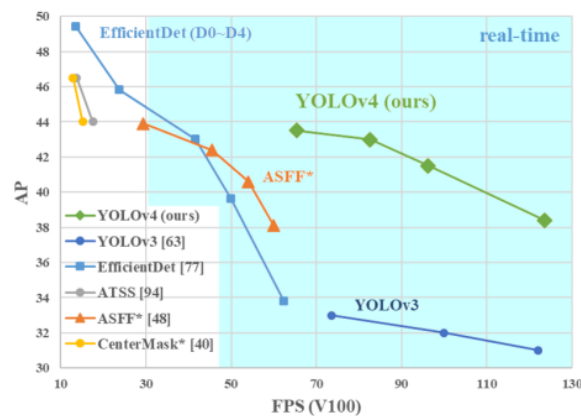


Fig. 5.11. Comparativa de YOLOv4 en términos de velocidad y precisión [44]

En la construcción de la red se ha utilizado en este caso el modelo CSPDarknet-53 como extractor de características, concatenándolo con un bloque denominado *Spatial Pyramid Pooling* (SPP). De esta forma se incrementa el campo de recepción, separando claramente las características referidas al contexto de cada clase, sin reducir la velocidad de operación de la red. Además, se ha sustituido el uso de FPN por un método de agregación de parámetros conocido como PANet (*Path Agregation Network*).

Con los cambios aplicados sobre el sistema se ha conseguido optimizar el proceso de entrenamiento de la red, usándose distintas estrategias y metodologías sin suponer los cambios efectuados ningún coste computacional extra. A este conjunto de procedimientos se les ha denominado *bag of freebies* (bolsa de bienes o regalos). Entre ellos destaca el aumento de datos mediante variaciones de las imágenes de entrada (cambio de color, tamaño, posición, etc.) junto con el uso de multitud de cajas de anclaje para un mismo objeto. Con la integración de las mejoras comentadas y una mejor elección de los hiperparámetros usados, se ha conseguido un sistema de detección mucho más robusto.

Además, se han incluido otras mejoras que, aunque sacrifican algo el rendimiento, suponen un gran incremento de la precisión del sistema. A este conjunto de medidas se las ha denominado *bag of specials*. Destaca el uso de una nueva función de activación, MISH, cuya forma aparece en la figura 5.12. Junto con el resto de las mejoras, se ha conseguido adecuar el modelo para poder ser entrenado en una única GPU.

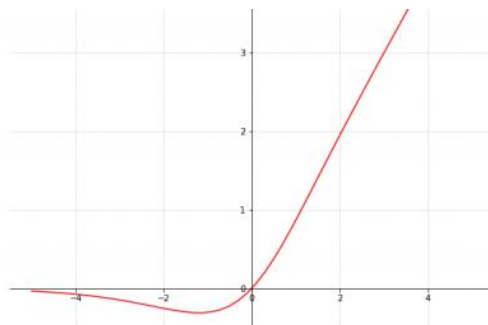


Fig. 5.12. Representación de la curva descrita por la función de activación Mish

5.3.5. YOLO v5

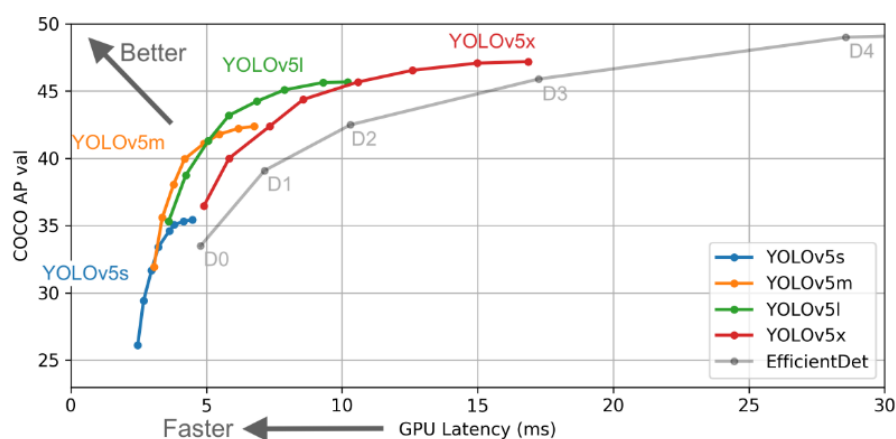
Tan solo unos meses tras del lanzamiento de la versión 4, el 10 de junio de 2020, aparece en el repositorio de Glenn Jocher [43], fundador de la compañía Ultralytics, una nueva

implementación del algoritmo con el nombre YOLOv5. Sin embargo, la falta de documentación explicativa de su desarrollo y la carencia de resultados han suscitado un gran debate al respecto de su mejoría.

Los datos aportados por el desarrollador muestran una tasa de procesamiento de 140 fps haciendo uso de una tarjeta gráfica como la Tesla P100. Estos resultados son mucho mejores a los 50 fps obtenidos bajo las mismas condiciones con su versión anterior. Además, se trata de un modelo mucho más ligero respecto a su antecesor, que con un tamaño de 27 MB es capaz de mantener una precisión similar.

A pesar de los resultados proporcionados, la controversia surgida alrededor de este sistema proviene de la comparativa entre algoritmos publicada por la propia compañía. El problema surge con la realización de las pruebas bajo diferentes criterios para cada una de las versiones, haciendo una comparativa poco precisa. Otro punto de conflicto es la elección del conjunto de datos usado en las pruebas. En este tipo de algoritmos se había estandarizado en los últimos años el uso del *dataset* de COCO pero para la versión 5 de YOLO se ha optado por el uso de un conjunto diferente, Blood Cell Count and Detection (BCCD).

Teniendo en cuenta lo comentado y valorando positivamente el buen rendimiento obtenido en precisión y velocidad que se observa en la figura 5.13, se ha elegido la versión 5 de YOLO para la implementación como base para el desarrollo del proyecto descrito en este documento.



6. TRATAMIENTO DE LOS DATOS

Como se indica en el repositorio de YOLO Ultralytics, existe la opción de poder realizar un entrenamiento mediante un *dataset* propio. En este capítulo se describe la búsqueda y el tratamiento de los subconjuntos de datos necesarios para el entrenamiento del sistema de ayuda a la conducción.

6.1. Requisitos de los datos de entrenamiento

Una de las tareas de mayor importancia a la hora de obtener buenos resultados en la detección se trata de la elección correcta de los datos de entrenamiento. Las características que se han elegido como indispensables son las siguientes:

- Es necesario seleccionar un *dataset* que posea entre todas sus clases aquellas que se necesiten para el sistema diseñado. Esta tarea es compleja, ya que es difícil que un único conjunto de datos tenga todas las clases que se necesitan. Es por ello que una de las soluciones que se deben tomar es la de la búsqueda por separado de distintos grupos de imágenes y la unificación de los mismos en uno solo.
- Las muestras deben ser uniformes. Esto implica que haya un número de muestras similar en cada clase para que el entrenamiento del sistema sea balanceado. Además, el conjunto de datos debe presentar variantes de cada etiqueta como por ejemplo, muestras desde distintos ángulos del mismo objeto, a diferentes distancias o en distintas condiciones de iluminación.
- Debe poseer una gran cantidad de muestras correctamente etiquetadas. Este apartado es fundamental ya que todos los sistemas de reconocimiento de objetos necesitan numerosos datos de entrada para poder extraer la información y de esta manera extrapolar lo aprendido a nuevas muestras.
- Mantener un tamaño constante en las imágenes de entrada para mejorar los resultados obtenidos ya que el algoritmo puede aplicar recortes o transformaciones del tamaño que harán que los objetos aparezcan distorsionados.
- Cada sistema utiliza un formato de etiquetado diferente, por lo que será necesario transformar los ficheros con la información de las *bounding boxes* al formato utilizado en el proyecto.

6.2. Elección del conjunto de datos

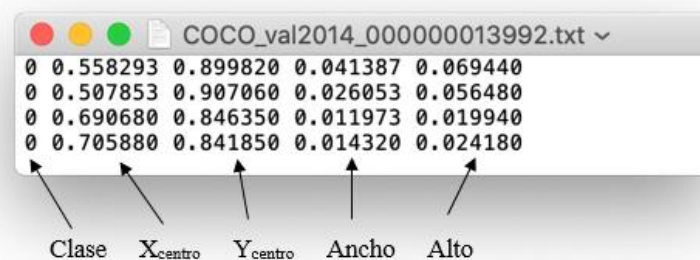
Teniendo en cuenta los apartados anteriores, se ha realizado la búsqueda de un *dataset* que pueda satisfacer la mayoría de los requisitos mencionados. Ya que es difícil encontrar un conjunto de datos que cumpla todos los puntos, se intenta encontrar un balance entre todos los candidatos que permita obtener buenos resultados para el sistema a diseñar.

En el caso propuesto, la implementación del prototipo de sistema de ayuda a la conducción, se ha optado por la realización de tres detectores por separado. Cada uno de estos detectores estará enfocado a la detección de un grupo de objetos. El primer detector se aplicará para la detección de elementos móviles como son vehículos o personas. Habrá un segundo detector encargado del reconocimiento de semáforos. Por último, el tercer detector será necesario para la detección de señales de tráfico. El motivo de la utilización de tres detectores por separado se debe a los problemas surgidos durante el entrenamiento con un único conjunto de datos unificado. Al tratarse de elementos completamente distintos tanto en forma como en tamaño, el algoritmo no es capaz de ajustar el modelo para la detección de todas las clases, tendiendo a localizar únicamente aquellas más grandes y de forma invariante como son el conjunto de señales de tráfico. Dentro del código, los detectores se crean como aparecen en la figura 6.1.

```
# Load model
model_1 = attempt_load(weights1, map_location=device)
model_2 = attempt_load(weights2, map_location=device)
model_3 = attempt_load(weights3, map_location=device)
```

Fig. 6.1. Código utilizado para la detección mediante tres detectores independientes

Previo al entrenamiento es necesario poseer los ficheros de etiquetas de cada conjunto de datos en el formato correcto utilizado por el algoritmo. En el caso de YOLO, se utiliza el formato Darknet, que sigue la forma descrita en la figura 6.2.



Clase	Xcentro	Ycentro	Ancho	Alto
0	0.558293	0.899820	0.041387	0.069440
0	0.507853	0.907060	0.026053	0.056480
0	0.690680	0.846350	0.011973	0.019940
0	0.705880	0.841850	0.014320	0.024180

Fig. 6.2. Formato de las etiquetas para Darknet [55]

6.2.1. Dataset de vehículos y personas

En primer lugar, se va a elegir un conjunto de datos que permita la detección de gran cantidad de objetos móviles. Este se utilizará en el sistema como base para la detección de vehículos y personas que se avisten durante la conducción. Existen varios conjuntos de imágenes surgidos a raíz de distintas competiciones realizadas en el ámbito de la detección de objetos. En la tabla 6.1 se presenta la comparación de distintos conjuntos de imágenes.

Competición	Clases	Entrenamiento	Prueba
VOC07	20	6.301	6.307
VOC12	20	13.609	13.841
ILSVRC17	200	478.807	55.502
COCO18	80	860.001	36.781
OID18	500	12.195.144	-

Tabla 6.1. Comparación de las diferentes competiciones sobre la detección de objetos

Si se tienen en cuenta los requisitos impuestos en la elección de los *datasets*, uno es la obtención de un gran conjunto de imágenes etiquetadas. Comprobando el número de anotaciones de cada *dataset*, aquellos que cumplen este objetivo en mayor medida son COCO y OID. Sin embargo, aunque el segundo posee más muestras, el número de clases es demasiado grande para el sistema deseado, donde únicamente interesan las clases asociadas a vehículos y viandantes. Es por este motivo que se elegirá COCO [41] como el *dataset* base para la localización y clasificación de objetos móviles.

Dentro del conjunto de imágenes de COCO se encuentran etiquetadas 80 clases distintas. Las que pueden resultar interesantes en el contexto de la conducción son: persona, bicicleta, coche, moto, bus, tren y camión. Por lo tanto, de las 80 clases disponibles, el conjunto de datos se aplicará para la detección de un grupo reducido de las mismas. En la figura 6.3 se muestran algunos ejemplos de imágenes pertenecientes a estas categorías junto con sus etiquetas correspondientes.



Fig. 6.3. Ejemplos de imágenes dentro del conjunto de imágenes COCO junto con sus etiquetas

Dado que desde el repositorio de YOLO Ultralytics se proporciona el fichero con los pesos pre-entrenados para las mejores prestaciones de YOLOv5 con COCO, no será necesario volver a entrenar con las clases seleccionadas, sino que se utilizará el conjunto completo. Una vez obtenido el fichero con los pesos, mediante código, se depurarán las detecciones para limitarlas al conjunto de los 7 objetos relevantes, descartando las clases restantes. El fragmento de código encargado de la selección de las clases que interesan de cara al desarrollo del primer detector se muestra en la figura 6.4.

```
# Print results
labels_coco = ['person', 'bicycle', 'car', 'motorcycle', 'bus', 'train', 'truck']
for c in det[:, -1].unique():
    if names_1[int(c)] in labels_coco:
        n = (det[:, -1] == c).sum() # detections per class
        s += '%g %ss, ' % (n, names_1[int(c)]) # add to string
```

Fig. 6.4. Fragmento de código encargado de la detección de las clases de COCO seleccionadas

Cabe mencionar a su vez, que los archivos con las etiquetas correspondientes a los objetos detectados y sus coordenadas vienen creados en formato Darknet. Dado que el algoritmo usado para el entrenamiento acepta datos en este formato, no será necesario modificar la información de la clase, las coordenadas del centro de la *bounding box* y las medidas normalizadas del ancho y el alto.

6.2.2. Dataset de semáforos

Para la detección de semáforos se ha elegido el conjunto de datos Bosch [56]. Las 13427 imágenes que contiene han sido tomadas en las carreteras de Estados Unidos. Esto, sin embargo, no será un problema ya que no hay grandes diferencias entre los semáforos que se pueden encontrar en Europa con los de EEUU. En la figura 6.5 se muestran algunos

ejemplos de imágenes que se pueden encontrar en el *dataset*. Además se han agregado las cajas delimitadoras originales sobre la imagen.



Fig. 6.5. Ejemplos de imágenes dentro del conjunto de Bosch junto con sus etiquetas

En este caso, a diferencia de COCO, el formato de las anotaciones disponibles no es el correspondiente a Darknet, por lo que ha sido necesario un *script* [57] que convierta el formato de los datos de Bosch (.xml) al formato aplicable en YOLO (.txt). Tras ello se ha comprobado si la transformación realizada se ha producido con éxito.

Este conjunto de datos posee 8 clases de semáforos: rojo izquierda, rojo, rojo derecha, verde izquierda, verde, verde derecha, ámbar y apagado. Sin embargo, no se utilizarán todas ellas ya que algunas no tienen un número suficiente de muestras que permita obtener buenos resultados. Por este motivo se ha establecido el número mínimo de etiquetas necesarias para el entrenamiento alrededor de las 100. Mediante código se ha realizado un recuento para conocer la distribución de las clases del conjunto de datos. Esta se muestra en la figura 6.6, por lo que las únicas clases que cumplen este requisito son: rojo izquierda, rojo, verde izquierda, verde, ámbar y apagado.

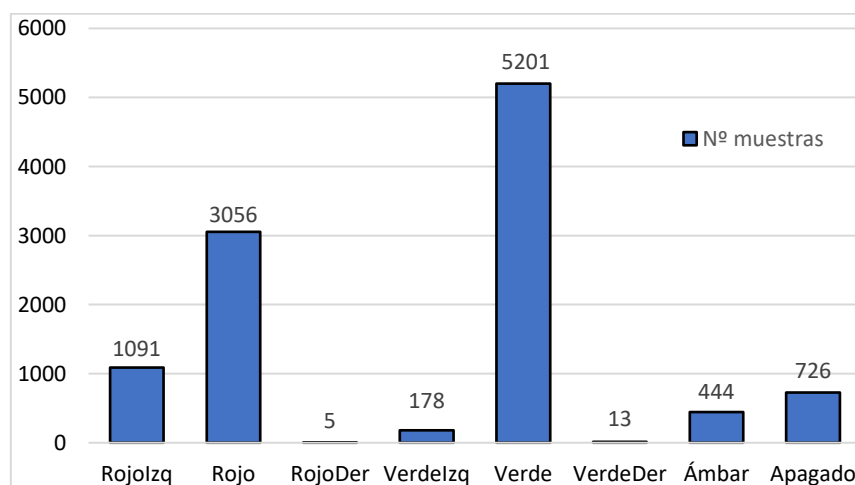


Fig. 6.6. Distribución de etiquetas en el dataset Bosch

Por último, ha sido necesaria la creación de un nuevo *script* encargado de la ordenación consecutiva de las clases seleccionadas. Es decir, una vez eliminadas las clases de rojo derecha y verde derecha es necesario cambiar el identificador asignado a cada clase creando una lista consecutiva en la que no existan los huecos que se corresponderían con dichas clases. Finalmente, la nueva asignación queda de la forma presentada en la tabla 6.2.

Semáforo	Etiqueta	Nº de muestras
Rojo Izquierda	0	1091
Rojo	1	3056
Verde Izquierda	2	178
Verde	3	5201
Ámbar	4	444
Apagado	5	726

Tabla 6.2. Distribución final de las clases utilizadas junto con su correspondiente etiqueta

6.2.3. Dataset de señales de tráfico

El tercer detector necesario para la posterior implementación del prototipo de ayuda a la conducción se corresponde con un detector de señales de tráfico. Una vez más, se ha realizado un cribado: de todas las señales existentes se ha optado por el reconocimiento de señales que puedan ser útiles, aportando información interesante para el sistema, y que a su vez creen una necesidad de interacción. Bajo este criterio se ha determinado como primordial la obtención de una base de datos con señales de velocidad, que marcarán la velocidad máxima de la vía, y señales que obliguen indirectamente a modificar este valor, como pueden ser una señal de ceda el paso o un Stop, requiriendo reducir la velocidad del vehículo o que se detenga. Otra señal que puede llevar a una disminución de velocidad será la de paso de peatones en caso de encontrarse conjuntamente con la detección de un

viandante. De esta forma, las clases seleccionadas son: Límite 20, 30, 40, 50, 60, 80, 90, 100, 120, Ceda el paso, Stop y Paso de cebra.

Para realizar el entrenamiento de este bloque detector, en primer lugar, se utilizarán como imágenes de entrenamiento para la señal de Stop las que vienen proporcionadas en el *dataset* de COCO. Será por lo tanto necesario la elaboración de un *script* que busque entre todos los ficheros con etiquetas aquellas en las que aparezca la clase correspondiente al Stop (clase número 11). En caso de existir en la misma imagen más objetos detectados pertenecientes a otras clases, será necesario eliminar la fila correspondiente para quedarnos únicamente con la de Stop. Ante la presencia de un fichero de texto con la clase elegida, se deberá buscar la imagen con el mismo nombre y se guardará en la carpeta donde quedarán almacenados todos los datos de entrenamiento para el detector de señales.

Para distinguir y etiquetar el resto de señales de tráfico se va a utilizar el conjunto de datos German Traffic Sign Detection Benchmark (GTSDB) [58], que dispone de 900 imágenes de distintas señales que se pueden encontrar en las carreteras alemanas. De todas las imágenes recopiladas tan solo será necesario seleccionar aquellas en las que aparezca un límite de velocidad, una señal de ceda o un paso de cebra.

Una vez más, habrá que seleccionar sólo las categorías relevantes al sistema. De esta manera se actuará de forma similar a la obtención de las señales de Stop a partir del *dataset* COCO; es decir, mediante un *script* se realizará un filtrado de las clases necesarias y se descartará el resto. Cabe remarcar que en esta ocasión el formato presentado en las etiquetas varía mínimamente al necesario para Darknet. En el *dataset* GTSDB las etiquetas aparecen como se muestra en la figura 6.7.

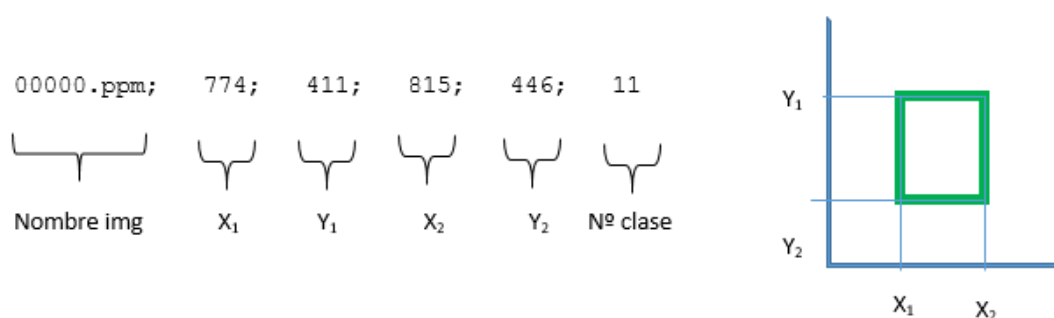


Fig. 6.7. Formato de las etiquetas en el conjunto de imágenes GTSDB

Dado que no es necesaria una modificación drástica del formato disponible, con los datos de las etiquetas existentes se puede realizar una reorganización de los mismos o, a partir de ellos, calcular los restantes. Para realizar esta operación se ha elaborado otro *script*. La

clase pasa entonces a colocarse al inicio, en lugar de ser el último dato de la línea. Tomando esa clase, se analiza si se encuentra entre las seleccionadas o debe ignorarse. En caso de tratarse de una clase relevante en el detector, es necesario buscar la imagen entre todas las presentes en el *dataset*. Adicionalmente, se han requerido los cálculos presentados en la figura 6.8 para encontrar el centro del recuadro, y el ancho y el alto del mismo.

```
def convert_coordinates(size, box):
    dw = 1.0/size[0]
    dh = 1.0/size[1]
    x = (box[0]+box[1])/2.0
    y = (box[2]+box[3])/2.0
    w = box[1]-box[0]
    h = box[3]-box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)
```

Fig. 6.8. Código necesario para el cálculo del centro, ancho y alto del rectángulo delimitador

El último paso necesario antes del entrenamiento del detector será la conversión del formato de imagen. Estas se encuentran originalmente en formato *.ppm*, formato no reconocido por el algoritmo. Debido a ello, las imágenes se han de convertir a *.jpg*, formato ya usado en el *dataset* de COCO, para que todas las imágenes presenten un tipo de archivo unificado.

Un problema encontrado en la recopilación de imágenes a usar en el entrenamiento ha sido la falta en algunas clases de muestras, dado que no se alcanzaba el número mínimo establecido previamente (100 muestras). Para completarlas, se hará un etiquetado manual a partir de imágenes obtenidas de Google. La búsqueda se limitará a señales españolas, ya que en otros países de la Unión Europea algunas de ellas presentan variaciones. En este proceso se ha prestado atención de cara a no repetir ninguna imagen, e incluso incluir fotografías en las que apareciesen varias señales pertenecientes a distintas clases. Con todo esto, y tratando de encontrar imágenes no demasiado pesadas, se reduce el número de imágenes que aportar al sistema para el entrenamiento y se reduce el tiempo del mismo.

En este proceso de etiquetado manual el paso final será la creación de los recuadros alrededor de las señales, y la consiguiente creación de las etiquetas a ellas correspondientes. Para agilizar el proceso es útil la ayuda de herramientas informáticas, habiéndose usado en este caso la herramienta de uso libre LabelImg [59]. Herramientas

como esta permiten, tras crear un fichero con los nombres de las clases y la posición ordenada de estas, importar las imágenes y etiquetarlas automáticamente sólo marcando la región de interés. El programa calcula de manera autónoma posteriormente el centro del recuadro, el ancho y el alto y lo escribe junto al número de clase en un fichero de texto con el nombre igual al de la imagen original. Un ejemplo del proceso de etiquetado mediante este programa se muestra en la figura 6.9.

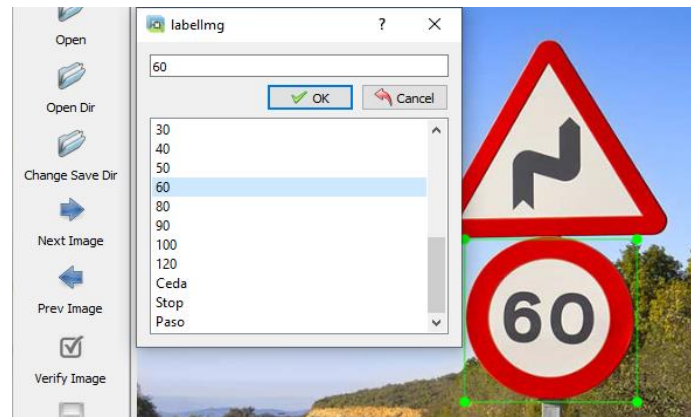


Fig. 6.9. Ejemplo del proceso de etiquetado de señales con LabelImg

Con el procedimiento seguido se ha conseguido un *dataset* uniforme, con un número similar de muestras para cada señal de tráfico. Tras la verificación del correcto etiquetado de imágenes de forma manual, se han agregado estas al conjunto de GTSDDB existente junto con las muestras correspondientes a la clase Stop, obtenidas de COCO. Algunos ejemplos de imágenes que se pueden encontrar en el nuevo *dataset* creado se muestran en la figura 6.10.



Fig. 6.10. Ejemplos de imágenes dentro del conjunto de señales de tráfico junto con sus etiquetas

Una vez completado el conjunto de datos para señales de tráfico con muestras etiquetadas a mano, la unificación del formato de sus etiquetas y la reorganización de las clases, el conjunto de datos obtenido tiene la distribución mostrada en la tabla 6.3.

Señal	Etiqueta	Nº de muestras
Límite 20 km/h	0	94
Límite 30 km/h	1	105
Límite 40 km/h	2	108
Límite 50 km/h	3	114
Límite 60 km/h	4	97
Límite 80 km/h	5	104
Límite 90 km/h	6	124
Límite 100 km/h	7	123
Límite 120 km/h	8	111
Ceda el paso	9	116
Stop	10	100
Paso de peatones	11	114

Tabla 6.3. Distribución de las clases del conjunto de datos de señales de tráfico

7. ENTRENAMIENTO DEL MODELO

En este séptimo capítulo se expone el proceso de entrenamiento realizado para los detectores de semáforos y señales. Asimismo, se indican los valores de los parámetros elegidos para ello y los resultados obtenidos.

7.1. Preparación del entrenamiento

Como se ha comentado anteriormente en este documento, se van a realizar dos entrenamientos por separado. El primero tiene como objetivo la obtención de los pesos utilizados para la detección de semáforos y el segundo, posee la información necesaria para el reconocimiento y clasificación de las distintas señales viales seleccionadas.

Para comenzar con el proceso de entrenamiento será necesario crear un par de ficheros por cada conjunto de datos. Esto sirve para que el algoritmo pueda realizar la configuración interna de sus capas y conocer las clases a entrenar, así como la ruta del conjunto de datos que se le va a suministrar. El primer fichero, cuya forma se muestra en la figura 7.1, será de tipo *.yaml* y poseerá la ruta con las imágenes de entrenamiento y validación, el número de clases con el que se va a entrenar y los nombres de las etiquetas como vector. En este caso, ambos conjuntos de datos, entrenamiento y validación, serán iguales. El segundo fichero será el de configuración del algoritmo, también en formato *.yaml*. Únicamente se necesitará modificar el número de clases y el resto de parámetros se mantendrán como aparecen en el fichero descargado del repositorio de Ultralytics. Con estos valores el algoritmo podrá calcular los valores óptimos para las capas de la red.

<pre># Ruta de los set de datos de entrenamiento y validación train: ../SEMAFORO/images/train/ val: ../SEMAFORO/images/train/ # Numero de clases nc: 6 # Nombre de las clases names: ['rojoIzq', 'rojo', 'verdeIzq', 'verde', 'ambar', 'OFF']</pre>	<pre># Ruta de los set de datos de entrenamiento y validación train: ../SEÑAL/images/train/ val: ../SEÑAL/images/train/ # Numero de clases nc: 12 # Nombre de las clases names: ['20', '30', '40', '50', '60', '80', '90', '100', '120', 'ceda', 'stop', 'paso']</pre>
---	--

Fig. 7.1. Ficheros para los conjuntos de imágenes de semáforos (izq.) y señales (der.)

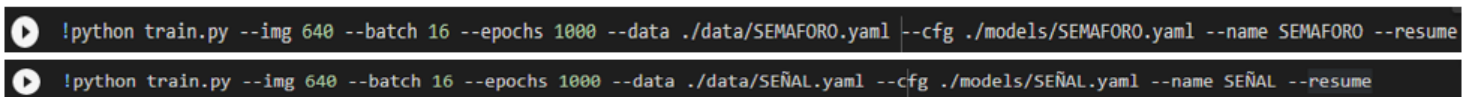
Como se comentó anteriormente, el número de clases para el conjunto de imágenes de semáforos es de 6, mientras que el de señales tiene 12 objetos a detectar. Existirán

entonces dos ficheros que hagan referencia a las imágenes de semáforos y sus 6 clases y otros dos referentes a las 12 clases de señales y las rutas a las imágenes correspondientes.

7.2. Proceso de entrenamiento

Para ejecutar el código necesario para realizar el entrenamiento de los detectores, se han utilizado las máquinas que Google pone a disposición de los usuarios de Google Colaboratory. Se trata de ordenadores en la nube más potentes que los ordenadores de los que cualquier usuario medio dispone, con gran potencia de procesamiento de imágenes gracias a las tarjetas gráficas que poseen. No obstante, existen límites de uso para no sobrecargar los servidores y evitar su mal uso. Debido a los límites establecidos por Google de 12 horas al día, ha sido necesario dividir en varias sesiones el entrenamiento mediante el comando `--resume`, que permite retomar la ejecución del proceso desde la última interrupción. De entre todas las opciones existentes, siempre que ha sido posible, se ha utilizado la tarjeta gráfica Tesla P100 de 16 GB, una de las mejores que posee Google Colab.

El comando introducido para la ejecución del fichero de entrenamiento para ambos detectores aparece en la figura 7.2. Uno de los parámetros elegidos es un tamaño de imagen de 640 píxeles con el parámetro `--img`, para ajustar los datos a la misma resolución que tiene COCO. Además, se trata de un tamaño de imagen que no es excesivamente grande, reduciendo el tiempo de procesamiento necesario para realizar cada iteración. Hay que tener en cuenta también que un tamaño de imagen constante arroja mejores resultados para el modelo.



```
!python train.py --img 640 --batch 16 --epochs 1000 --data ./data/SEMAFORO.yaml --cfg ./models/SEMAFORO.yaml --name SEMAFORO --resume
!python train.py --img 640 --batch 16 --epochs 1000 --data ./data/SEÑAL.yaml --cfg ./models/SEÑAL.yaml --name SEÑAL --resume
```

Fig. 7.2. Comandos para el entrenamiento de los modelos de semáforos (superior) y señales (inferior)

Entre el resto de parámetros introducidos en la ejecución se encuentra el valor de `--batch`, que marca el número de imágenes procesadas en cada iteración, con un valor de 16. De igual manera, se establece el número de épocas con `--epoch`, eligiendo un valor de 1000. Esto indica que se realizarán 1000 iteraciones para el ajuste de pesos. Finalmente, se adjunta la ruta del fichero de datos y de configuración ya mencionados.

7.2.1. Indicadores de validación

Para comprobar la evolución del entrenamiento se van a utilizar diferentes indicadores. Todos los que se han tenido en cuenta son útiles de cara a reconocer las prestaciones de los detectores. Estos indicadores son:

- **IoU:** son las siglas en inglés de Intersección sobre la Unión [60] (*Intersection over Union*). Este parámetro se calcula como la relación entre el área que comparte el rectángulo delimitador original (*ground truth*) y la caja predicha por el sistema, dividida por el área total de estas figuras juntas. La fórmula utilizada se observa en la ecuación 7.1. Este parámetro indica con qué precisión es capaz de localizar los objetos el modelo. Cuanto mayor sea este valor, mejores prestaciones.

$$\text{IoU} = \frac{\text{Área de la superposición}}{\text{Área de la unión}}$$

Ecuación 7.1. Representación del cálculo del indicador IoU

- **Precisión:** este parámetro [60] tiene en cuenta el número de aciertos realizados en la clasificación de una clase, respecto al número de objetos pertenecientes a esa clase entre todas las detecciones realizadas. Su fórmula se representa en la ecuación 7.2. y aporta la información relevante sobre lo acertado que es el modelo a la hora de clasificar los objetos.

$$\text{Precisión} = \frac{\text{Nº detecciones correctas de la clase C}}{\text{Nº objetos detectados para la clase C}}$$

Ecuación 7.2. Ecuación para el cálculo de la precisión del modelo

- **Cobertura:** de forma similar a la precisión, la cobertura [60] (*recall* en inglés) se formula como la relación entre el número de detecciones correctas de una cierta clase respecto al total de objetos de dicha clase. Su fórmula aparece en la ecuación 7.3.

$$\text{Recall} = \frac{\text{Nº detecciones correctas de la clase C}}{\text{Nº objetos totales pertenecientes a la clase C}}$$

Ecuación 7.3. Ecuación del cálculo del indicador de Recall

- **Mean Average Precision (mAP):** se trata del parámetro más importante [60] ya que calcula la precisión media de todos los valores promedio de precisión obtenidos para cada clase. Su formulación sigue la forma de la ecuación 7.4.

Además se puede calcular este valor para ciertos umbrales de IoU, calculándose en el modelo implementado para un valor de IoU de 0.5 y otro para el intervalo entre 0.5 y 0.95.

$$\text{mAP} = \frac{\sum \text{Precisiones promedio}}{\text{Nº de clases}}$$

Ecuación 7.4. Ecuación para el cálculo del indicador mAP

7.3. Resultados obtenidos

El proceso de entrenamiento ha sido relativamente largo. Como se ha comentado anteriormente, siempre se ha intentado realizar el entrenamiento con la misma instancia de Colab. En este caso ha sido una máquina equipada con la tarjeta gráfica Tesla P100-PCIE de 16 GB. Se trata de una de las más potentes dentro de todas las disponibles en Google Colaboratory. En el caso del conjunto de imágenes de señales, el tiempo por cada iteración, en media, ha sido de 3 minutos. Como se han completado 1000 ciclos, el tiempo total de entrenamiento para este *dataset* es de 50 horas. Para el conjunto de semáforos, el entrenamiento ha sido más rápido. El tiempo medio por iteración se ha reducido a 40 segundos, lo que hace un total de 11 horas.

Tras el proceso de entrenamiento se han obtenido los dos ficheros de pesos que utilizarán los detectores de señales y semáforos respectivamente. Estos dos ficheros, junto a los obtenidos del repositorio de Ultralytics para el *dataset* de COCO, serán los necesarios para el funcionamiento de las detecciones en el sistema.

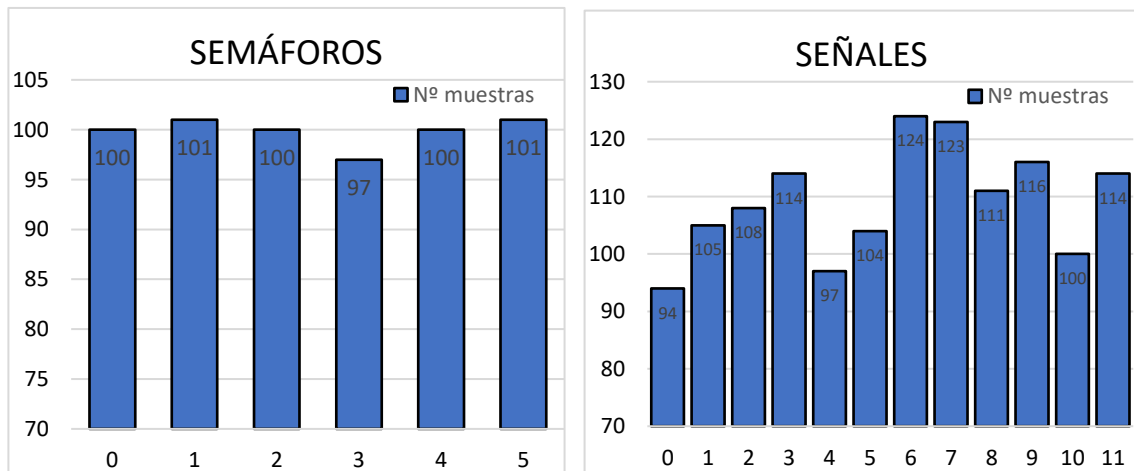


Fig. 7.3. Distribución de las muestras para los conjuntos de semáforos (izq.) y señales (der.)

Previo al proceso de validación se puede corroborar que el entrenamiento se haya hecho con la configuración correcta. En primer lugar se puede comprobar en la figura 7.3 que la distribución del número de muestras por cada clase ha sido el correspondiente con el mínimo establecido, cerca de las 100 muestras pertenecientes a cada una de las clases a detectar.

Adicionalmente, se puede comprobar la distribución de los tamaños de las cajas delimitadoras para cada uno de los conjuntos de imágenes. En la figura 7.4, como se puede apreciar, los tamaños de los rectángulos delimitadores correspondientes al conjunto de datos de señales son considerablemente mayores a aquellos obtenidos para el conjunto de semáforos. Es por ello que es una buena idea el uso de dos detectores distintos, garantizando que cada uno se pueda ajustar correctamente al tamaño de los objetos de los que se encarga.

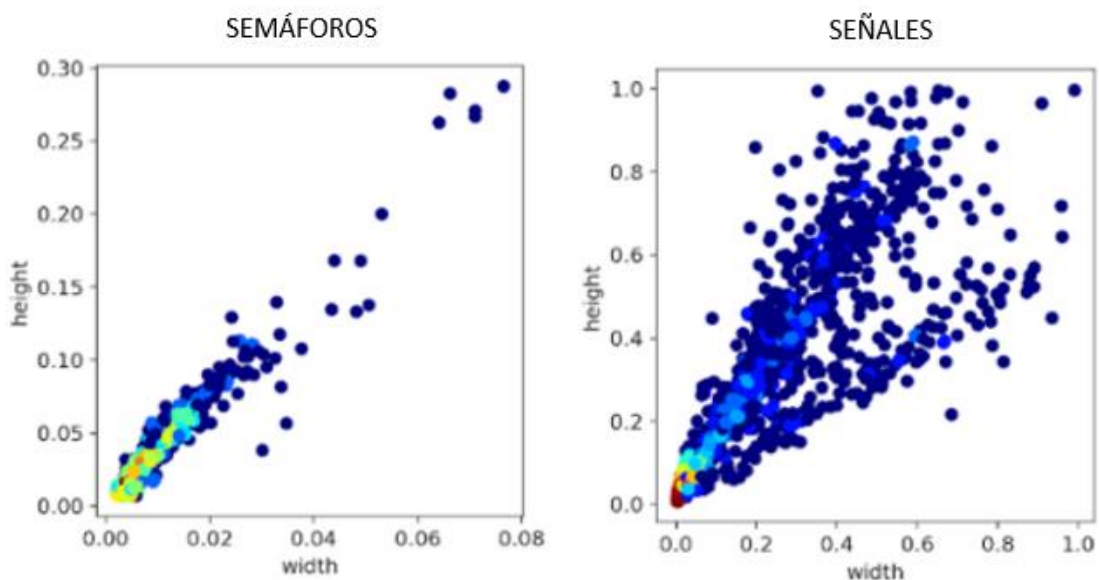


Fig. 7.4. Comparación de tamaño de objetos entre semáforos (izq.) y señales (der.)

Una vez comprobado que el entrenamiento ha sido correcto teóricamente, es necesario realizar una evaluación de los resultados mediante los valores de los indicadores descritos en el epígrafe 7.2.1. Los resultados obtenidos para ambos entrenamientos se muestran en las gráficas de la figura 7.5.

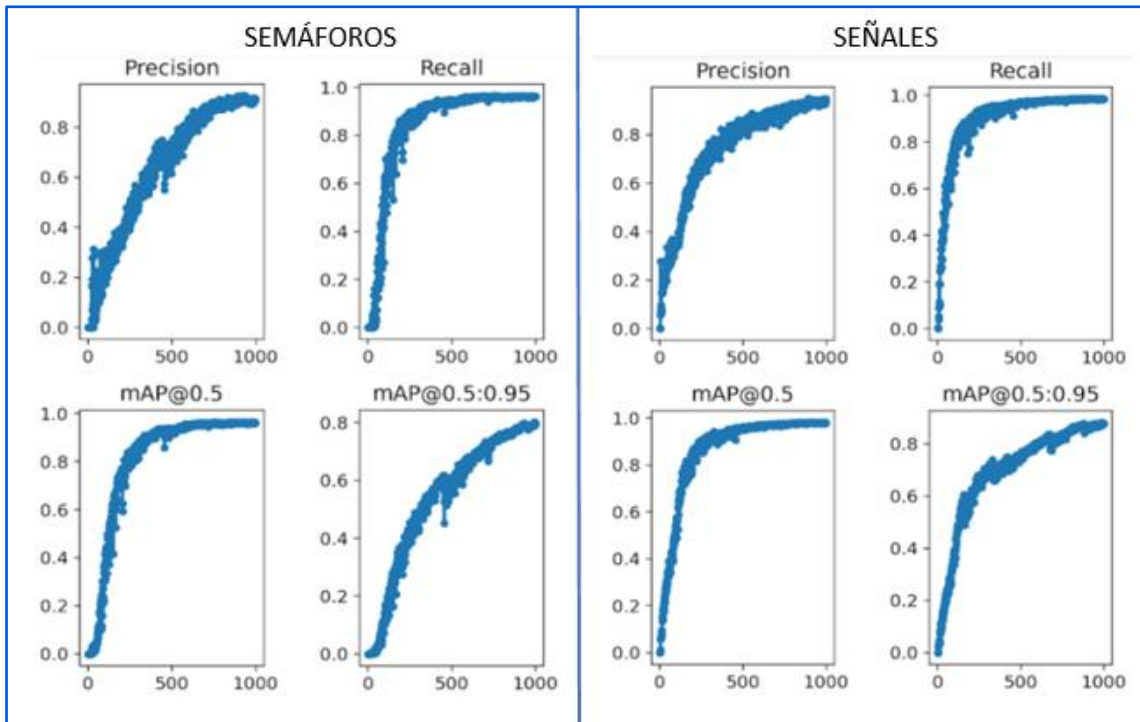


Fig. 7.5. Comparación de los indicadores de rendimiento para semáforos (izq.) y señales (der.)

Comparando las gráficas de ambos conjuntos se puede apreciar que la evolución de los indicadores ha sido más rápida en el *dataset* de señales. Se observa que para el detector de señales de tráfico el crecimiento es más pronunciado y se consigue una estabilidad en menor número de iteraciones, alcanzándose valores similares al caso de semáforos en un tiempo menor. Esto se debe principalmente al tamaño de las muestras, ya que al encontrarse cajas de mayores dimensiones en el *dataset* de señales, el algoritmo reconoce detalles con mucha mayor rapidez y es capaz de ajustar sus pesos rápidamente.

Otro aspecto a destacar de las gráficas propuestas en la figura 7.9 es la evolución del indicador mAP en el rango de IoU entre 0.5 y 0.95. Como se puede comprobar, el valor es menor que el obtenido para un IoU de 0.5 para ambos conjuntos de datos. Esto indica que hay cierta variabilidad entre la posición de la caja delimitadora y la posición correcta. Mientras que para una IoU de 0.5 se alcanza rápidamente una precisión cercana a 1, para el rango de IoU de entre 0.5 y 0.95 este parámetro no se acerca al mismo valor hasta las últimas iteraciones. Si se quisiese obtener mejores resultados, de acuerdo a lo explicado, sería necesaria la asignación de un mayor número de iteraciones a cada entrenamiento.

Cabe mencionar que, uno de los grandes problemas que pueden surgir en el entrenamiento de redes neuronales es el sobreajuste (*overfitting*). Esto ocurre cuando el modelo se amolda en exceso a los datos con los que ha sido entrenado, de tal forma que

al recibir una nueva muestra desconocida, la red no es capaz de generalizar el conocimiento adquirido para la correcta clasificación de la misma. Gráficamente, se puede comprobar mediante los indicadores de precisión y *recall*, los cuales, en caso de haber sobreajuste, se mantendrían constantes durante un gran número de iteraciones. En la figura 7.5 se puede comprobar que, aunque el *recall* alcanza un valor constante durante gran parte del entrenamiento, el valor de la precisión continua creciendo con cada iteración. En caso de haber notado indicios de sobreajuste habría sido necesario utilizar métodos como ‘*EarlyStopping*’ que interrumpirá la ejecución del entrenamiento si los valores de precisión no evolucionan con cada iteración.

Finalmente, se muestran en la tabla los mejores valores obtenidos para cada uno de los indicadores en ambos entrenamientos.

	Precisión	Recall	mAP @ 0.5	mAP @ 0.5:0.95
Semáforos	0.914	0.9634	0.9625	0.7961
Señales	0.9494	0.9827	0.9807	0.8783

Tabla 7.1. Tabla con los mejores valores de indicadores obtenidos en ambos entrenamientos

7.4. Validación

Una vez finalizado el entrenamiento y la integración de los tres detectores, es necesario realizar el proceso de validación para la comprobación del correcto funcionamiento. Para ello, se van a analizar a mano varios vídeos ante distintas condiciones para la anotación de los errores y aciertos del sistema. Dado que los pesos del *dataset* de COCO han sido ya entrenados para las mejores prestaciones, únicamente se comprobarán las prestaciones de los conjuntos de señales y semáforos.

En primer lugar, se van a analizar dos vídeos grabados en el mismo recorrido a una resolución de 1280x720 píxeles, el primero durante el día y el segundo de noche. En ellos aparecen señales de distintos tipos y semáforos. Previo a la validación se ha comprobado la velocidad de procesamiento del sistema. Con esta resolución y la tarjeta gráfica Tesla P100, el tiempo de procesamiento por cada imagen es de 0.1 segundo, dando una tasa de 10 fps. Dado que una cámara común suele grabar a 30 fps, esto supondría la pérdida de 2 de cada

3 fotogramas. Sin embargo, en el contexto de la conducción, la pérdida de fotogramas no supone un problema, ya que el entorno no varía tan repentinamente. Por ello, bajo estas condiciones, es posible la detección en tiempo real aplicando el prototipo desarrollado.

Comenzando con el proceso de validación, se van a establecer ciertos umbrales como punto de partida para el cálculo de prestaciones. Estos umbrales serán iguales para ambos vídeos e indicarán a partir de qué probabilidad arrojada por el sistema aparecerá la detección en el vídeo final. El umbral para el *dataset* de semáforos tiene el valor de 0.6, mientras que el de señales se establece en 0.85. De cara a la elección de dichos valores, se ha analizado previamente el comportamiento de los detectores bajo distintos umbrales, seleccionándose los que a primera vista ofrecían mejores resultados como punto de partida. Los datos recogidos aparecen en la tabla 7.2 para ambos *datasets*.

	Dataset	Detección Correcta	Detección Incorrecta	No detectado
Día	Señales @ 0.85	362	57	39
	Semáforos @ 0.6	259	171	221
Noche	Señales @ 0.85	276	89	134
	Semáforos @ 0.6	56	73	374

Tabla 7.2. Tabla de detecciones durante el día y la noche con el mismo umbral

A partir de los datos de la tabla, se pueden calcular parámetros de validación de precisión y *recall* mediante las fórmulas presentadas en las ecuaciones 7.2 y 7.3. Los resultados obtenidos aparecen en la tabla 7.3.

	Dataset	Precisión	Recall
Día	Señales @ 0.85	0.864	0.903
	Semáforos @ 0.6	0.602	0.539
	Sistema completo	0.731	0.705

Noche	Señales @ 0.85	0.756	0.673
	Semáforos @ 0.6	0.434	0.130
	Sistema completo	0.672	0.395

Tabla 7.3. Cálculo de la precisión y el recall para cada vídeo bajo el mismo umbral

Observando la tabla 7.2, se puede comprobar que durante el día se han realizado aproximadamente el mismo número de detecciones para ambos *datasets*. Sin embargo, es remarcable la diferencia en cuanto a las detecciones correctas e incorrectas en cada caso. En el conjunto de semáforos existe una elevada cantidad de detecciones erróneas en relación a las correctas. De igual manera, hay un elevado número de muestras no identificadas. Estos datos se hacen más visibles en la tabla 7.3, donde se muestran la precisión y el *recall* de ambos detectores, así como del sistema completo. Aunque el detector de señales ofrece una alta fiabilidad, los indicadores obtenidos para el conjunto de semáforos provocan que los resultados del sistema completo rondan un valor de 0.7.

En caso de realizar la comparación con los valores nocturnos, es visible la reducción del número de muestras detectadas, especialmente en el detector de semáforos. El número de objetos contabilizados en el campo de detecciones correctas es 200 puntos menor al de aquellos contabilizados durante el día. A su vez, se observa una diferencia de igual magnitud entre las muestras de semáforos detectados correctamente y las muestras de señales correctamente localizadas durante la noche. Estos resultados se ven representados nuevamente en la tabla 7.3, donde precisión y *recall* disminuyen considerablemente en todos los casos, siendo mayor la diferencia siempre para el *recall*.

Teniendo en cuenta los párrafos anteriores, se pueden sacar ciertas conclusiones:

- Durante el día, el sistema ofrece altas prestaciones pese a obtenerse unos resultados peores en la detección de semáforos.
- Durante la noche, las detecciones decrecen para ambos detectores, debido a que la falta de luz provoca una mayor confusión en el sistema.
- El detector de semáforos ha sido entrenado a partir de imágenes nítidas. Dado que una luz capturada en un vídeo, especialmente durante la noche, aparece distorsionada, el reconocimiento de estos objetos se dificulta.

Para tratar de mejorar los resultados previamente obtenidos, se han modificado los umbrales de detección usados para cada conjunto de datos. Se ha intentado obtener un mayor reconocimiento de objetos bajando el umbral con el que éstos se detectan. Durante el día, la disminución ha sido menor que la aplicada al caso nocturno. Esto se debe al visible empeoramiento de las detecciones ante la falta de luz. Asimismo, para el detector de semáforos se ha optado por hacer una reducción mayor en el umbral ya que estos objetos se tienden a reconocer con una menor confianza. Los nuevos resultados se recogen en la tabla 7.4.

	Dataset	Detección Correcta	Detección Incorrecta	No detectado
Día	Señales @ 0.8	398	71	31
	Semáforos @ 0.45	449	217	103
Noche	Señales @ 0.7	388	88	113
	Semáforos @ 0.3	76	81	354

Tabla 7.4. Tabla de detecciones durante el día y la noche reduciendo el umbral

Con la bajada del umbral, como se esperaba, ha aumentado el número de detecciones totales en cada vídeo. La diferencia es drástica para el caso del detector de semáforos en el vídeo grabado durante el día, donde el número de detecciones se ve prácticamente doblado.

En este caso, se ha optado por proceder de la misma manera que con los datos recogidos en la tabla 7.2, es decir, calcular los parámetros de validación del sistema. Los nuevos resultados se recogen en la tabla 7.5.

	Dataset	Precisión	Recall
Día	Señales @ 0.8	0.849	0.928
	Semáforos @ 0.45	0.674	0.813
	Sistema completo	0.746	0.863

Noche	Señales @ 0.7	0.815	0.774
	Semáforos @ 0.3	0.484	0.177
	Sistema completo	0.733	0.498

Tabla 7.5. Cálculo de la precisión y el *recall* para cada vídeo disminuyendo el umbral

Si se realiza un análisis de los nuevos resultados, y se comparan con los que se obtenían con umbrales mayores, los puntos con una mayor importancia serán:

- Las precisiones con umbrales menores tienden a aumentar. Sin embargo, si se reduce demasiado el umbral, podría disminuir la precisión dado que existirá un mayor número de detecciones incorrectas.
- Con umbrales menores se consigue un mayor *recall*. Esto se debe a que el número de muestras no detectadas disminuye y el de detecciones correctas aumenta.

Teniendo en cuenta las ideas mencionadas y los valores de los parámetros de validación, se optará por la utilización de los umbrales aplicados en la segunda prueba.

8. SISTEMA DE AYUDA A LA CONDUCCIÓN

En el octavo capítulo se va a explicar finalmente la construcción del prototipo de sistema de ayuda a la conducción, especificando su funcionalidad y el proceso llevado a cabo para la integración de cada parte.

Una vez obtenidos los pesos, se va a proceder a la implementación de un prototipo de ayuda a la conducción basado en el reconocimiento de los elementos seleccionados. Se busca con el programa que el sistema sea capaz de mostrar la velocidad máxima de la vía, alertar al conductor de la necesidad de reducir la velocidad del vehículo y detectar posibles obstáculos que afecten a la conducción.

Como método para evitar errores se ha establecido un *threshold* diferente para cada detector. De esta forma, cualquier posible detección con una probabilidad menor a la establecida será descartada, evitando todas aquellas detecciones incorrectas que pueden aparecer a lo largo de la ejecución. Aun así, en ocasiones, el sistema confunde ciertos objetos con una probabilidad muy alta, haciendo detecciones esporádicas que duran un fotograma y desaparecen. Para evitar estos casos se generará un contador, el cual se incrementará con cada fotograma consecutivo en el que aparezca una clase. De este modo, cuando aparezca una detección continuada durante varios fotogramas, se dará por buena esa detección.

8.1. Control de velocidad

En primer lugar, se ha creado un sistema de control de velocidad basado en la lectura de señales de tráfico de velocidad. El sistema comenzará con un valor de cero que se irá actualizando con cada señal de velocidad diferente que se detecte. En la esquina inferior derecha se mostrará gráficamente la velocidad máxima identificada. Para lograr mostrar la velocidad reconocida se utilizará una variable en el código, inicializada a 0. Su valor se irá actualizando a medida que se detecte una señal de velocidad. En la figura 8.1 se muestran las líneas donde se actualiza el indicador de velocidad máxima de la vía.



Fig. 8.1. Ejemplo de actualización de la velocidad máxima de la vía

A su vez, esta velocidad debe ser modificada en ciertas circunstancias, alertando al conductor de la necesidad de reducir su velocidad. Los casos recogidos para esta condición en la que el contador volverá a su posición inicial, es decir, valor nulo, son los siguientes:

- En caso de encontrarse una señal de Stop, se indicará la necesidad de parar el vehículo cuando el área de la señal sea suficientemente grande, ya que esto indicará su proximidad. Esta área se calcula como la multiplicación del ancho y del alto de la caja delimitadora. Si el producto de ambos supera un umbral de 15000 píxeles, se considerará lo suficientemente cercano como para que el sistema actúe. Cabe mencionar que este valor debe elegirse dependiendo de la situación, pudiendo modificarse el umbral en función de la resolución de la cámara, el tamaño de imagen, etc. La velocidad, por consiguiente, se colocará a 0 y saldrá el indicador gráfico correspondiente. Un ejemplo del funcionamiento relatado se puede observar en la figura 8.2.



Fig. 8.2. Actualización del indicador de velocidad ante una señal de Stop

- En caso de encontrarse una señal de paso de peatones, si en el área central de la imagen se detecta una persona, el sistema actuará nuevamente. De esa forma el

conductor sólo deberá reducir la velocidad por precaución en el paso de peatones, llegando a pararse, si detecta gente cruzando. Si no se detectan personas cruzando, el vehículo no modificará su velocidad y el sistema no mostrará ningún indicador para modificar el comportamiento. Ambos ejemplos se muestran en la figura 8.3.



Fig. 8.3. Actualización de la velocidad ante un paso de peatones.

- Si durante la conducción se detecta un semáforo con luz ámbar o roja, el sistema mostrará una alerta gráfica para indicar al conductor que ante esta situación también debe reducir la velocidad de conducción. Al cambiar el semáforo a luz verde, el indicador de velocidad volverá a la última señal de velocidad detectada. Un ejemplo de esto se muestra en la figura 8.4.



Fig. 8.4. Sistema de frenado ante semáforo rojo o ámbar

- Ante situaciones peligrosas que puedan llevar a una colisión con un objeto también existirá un aviso que alerte al conductor, como se hace visible en la figura 8.5. Aunque en este caso también se pondrá el contador de velocidad a cero ya que el conductor deberá frenar, el aviso será distinto, mostrándose un texto de alerta. Para controlar estas situaciones se ha de implementar un sistema de control de proximidad, que entrará en funcionamiento únicamente al realizar una detección muy cercana a nuestro vehículo, de cara a evitar una colisión. Su funcionamiento se explica con mayor detalle en el epígrafe 8.2.



Fig. 8.5. Ejemplo del sistema de detección de proximidad.

8.2. Control de proximidad

Para poder controlar la proximidad de los objetos detectados y evitar posibles colisiones, se utilizará la visión artificial. El proceso seguido para lograr esta finalidad es el siguiente:

- En primer lugar, se ha dibujado una línea poligonal sobre el vídeo en la zona correspondiente al carril por el que circula el vehículo. Esta área será la zona sensible a tener en cuenta para la detección de proximidad. Una vez conocida la región, se aplicará el proceso para generar la alerta.
- En la comprobación de la proximidad, primero se ha de calcular el centro de la caja delimitadora y comprobar que se encuentra dentro de la zona de riesgo. Este procedimiento no otorga suficiente información al respecto de la distancia a la que se encuentra el objeto, es por ello que a continuación será necesario calcular el tamaño del área de la detección.
- Una vez obtenido el área, si el valor resultante de multiplicar el ancho por el alto es mayor que cierto umbral (elegido por el desarrollador del sistema), significará que el objeto se encuentra cerca y la velocidad debe reducirse. El valor fijado en este caso ha sido de 60000 píxeles.

8.3. Representación visual de señales

Para finalizar el sistema, se ha añadido un módulo encargado de la representación de las señales detectadas. Este módulo se encargará de mostrar por pantalla las señales detectadas mediante una representación gráfica. Cabe mencionar que su funcionalidad solo afecta a los semáforos y a las señales que no indican la velocidad máxima de la vía, dado que éstas se utilizan únicamente para el control de velocidad

Para el caso de las señales de tráfico, se ha dividido la pantalla en dos mitades, derecha e izquierda. Ante la detección de una de las señales seleccionadas, se calcula el centro de la detección. Más tarde, atendiendo a la posición que tenga respecto a la división de pantalla, se asigna la representación a uno de los dos lados. De esta manera, se puede observar con mayor claridad en qué lado de la vía se encuentra la señal detectada. En ocasiones, se pueden encontrar señales a ambos lados de la carretera, por lo que será necesario en dichas situaciones mostrar la representación de cada una a la derecha e izquierda de la imagen.

El proceso de impresión por pantalla de la representación de las señales se ha realizado de la siguiente forma:

- Se han creado láminas transparentes con la misma resolución que el vídeo (1280x720), colocando las diferentes señales en distintas posiciones. Las señales se mostrarán siempre, sin importar el lado en el que se detecten, en sentido descendente en el siguiente orden: Stop, Ceda y Paso de peatones. Se ha prestado especial atención a la posición asignada de tal forma que no se solapen al superponer varias láminas. De este modo, ha sido necesario elaborar 6 láminas distintas, una para cada caso, indicando en su nombre la señal y el lado.
- Con la información relativa a la posición en la imagen y la clase a la que pertenece la detección, se ha generado una cadena en la cual se concatenan el nombre de clase y la subcadena “Der” o “Izq” en función del lado. A través de esa cadena, se busca la lámina correspondiente a la señal a mostrar, situada en el lateral correcto. El fragmento de código encargado de la detección del lado y la generación de la cadena correspondiente se puede observar en la figura 8.6.

```

c1 = (int(xyxy[0]), int(xyxy[1]))
c2 = (int(xyxy[2]), int(xyxy[3]))
center = (int((c2[0]+c1[0])/2), int((c2[1]+c1[1])/2))
if center[0] < 640:
    slide = Image.open('signals/' + str(name) + 'Izq.png')
    im0 = cv2.cvtColor(im0,cv2.COLOR_BGR2RGB)
    im0 = Image.fromarray(im0)
    im0.paste(slide, (0,0),slide)
    im0 = np.array(im0)
    im0 = cv2.cvtColor(im0,cv2.COLOR_RGB2BGR)

if center[0] > 640:
    slide = Image.open('signals/' + str(name) + 'Der.png')
    im0 = cv2.cvtColor(im0,cv2.COLOR_BGR2RGB)
    im0 = Image.fromarray(im0)
    im0.paste(slide, (0,0),slide)
    im0 = np.array(im0)
    im0 = cv2.cvtColor(im0,cv2.COLOR_RGB2BGR)

```

Fig. 8.6. Código para la representación de las señales en el lado correspondiente

- Una vez recuperada la lámina, se ha optado por superponerla sobre el fotograma procesado, de cara a que en el resultado final se pueda ver la imagen original con los diferentes avisos sobre ella. En la figura 8.7. se muestra un ejemplo del funcionamiento ante la detección de señales.

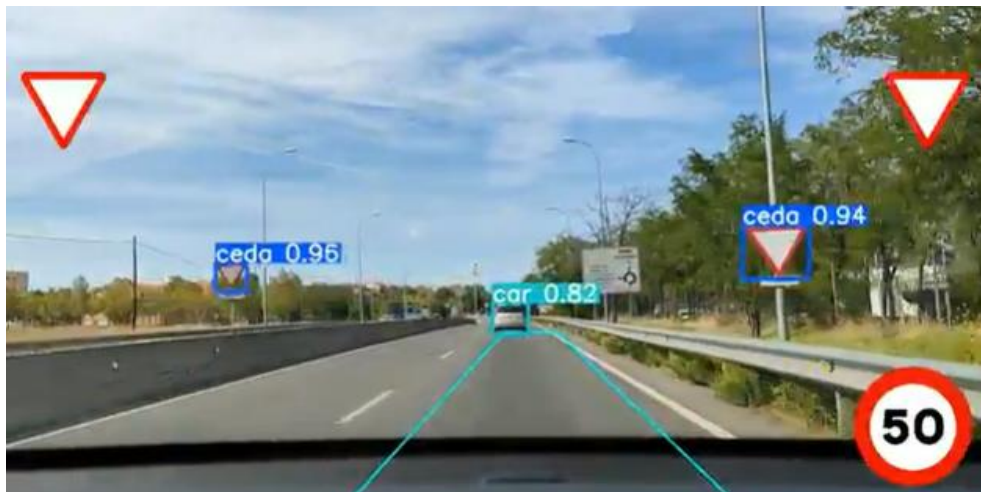


Fig. 8.7. Ejemplos de la visualización de señales en el sistema de ayuda a la conducción

Por otro lado, en el caso de la representación de los semáforos, se ha optado por operar de una manera distinta. En esta situación, en lugar de colocarse el aviso en los laterales, se ha decidido mostrarlo centrado en la parte superior de la pantalla. Es por este motivo que el procedimiento ha variado ligeramente. Aunque ha sido necesario crear láminas para cada detección de clases de semáforos, sólo existirá una por cada tipo. Además, no es necesario generar una cadena con información acerca de la posición para poder superponerla. Un ejemplo del indicador de semáforo se muestra en la figura 8.8.



Fig. 8.8. Ejemplos de la visualización de semáforos en el sistema de ayuda a la conducción

8.4. Validación del prototipo

Tras la incorporación del módulo encargado del prototipo de sistema de ayuda a la conducción, se ha realizado una validación del funcionamiento y las prestaciones obtenidas. Para ello, se han vuelto a procesar los mismos vídeos utilizados en la validación del entrenamiento, comparando los resultados entre una conducción diurna y nocturna.

En este caso, gracias a la incorporación de un contador, se consigue mostrar únicamente la señal cuando aparece en varios fotogramas consecutivos. De esta forma se han conseguido reducir considerablemente los efectos de parpadeo surgidos de detecciones erróneas. Dado que la precisión para cada grupo de clases es distinta, se ha realizado un análisis de cada conjunto para fijar el número mínimo de fotogramas necesarios para la representación de la señal:

- Para el caso del *dataset* de objetos dinámicos, es decir, COCO, no ha sido necesaria la incorporación de ningún contador de fotogramas, ya que con los pesos descargados el sistema aporta una gran precisión en las detecciones. Además, como este conjunto de clases no necesita ser representado sobre el vídeo, no cabe la posibilidad de problemas de parpadeo.
- Respecto al conjunto de señales de tráfico, el principal problema se concentra en la detección de señales de velocidad. Estas, al presentar la misma forma, independientemente del valor de velocidad de su interior, pueden ser confundidas

con mayor facilidad. Es por ello que se ha establecido un valor elevado de fotogramas necesarios para su representación. En este caso se ha fijado en 15 fotogramas.

- Como se ha comentado en el epígrafe 7.4, el detector de semáforos es el que peores resultados arroja. Además de los errores que presenta en la clasificación en la clase correcta, el prototipo es propenso a la localización de semáforos en lugares donde no existen, soliendo confundirse con otros puntos de luz o reflejos en la luna delantera. Sin embargo, estos errores no se suelen dar de manera consecutiva durante varios fotogramas, sino que suelen ser en momentos esporádicos. Es por ello que no será necesario fijar un valor tan elevado al contador como el asignado a señales. En este caso, el número mínimo de fotogramas es de 10 consecutivos.

Gracias a este mecanismo que se ha incorporado en el prototipo, se han disminuido considerablemente los errores, consiguiendo un sistema que se confunda en menor medida y que sea más robusto.

9. PLANIFICACIÓN Y PRESUPUESTO DEL PROYECTO

En este capítulo se expone la planificación realizada para la realización del proyecto en los plazos marcados junto con los costes del desarrollo del mismo y un posible presupuesto para su implementación.

9.1. Planificación

Tras la elección del TFG y el fin de las clases en el mes de mayo, se estableció como objetivo la defensa del trabajo para la convocatoria de octubre, sin embargo, debido a la situación excepcional provocada por la pandemia del Covid19, el cierre de las universidades y la falta de medios, ha sido imposible el cumplimiento de los plazos. Finalmente, se ha fijado la defensa del proyecto para la convocatoria excepcional abierta para el mes de noviembre. En la tabla 9.1 se han detallado las tareas a realizar, siguiendo como referencia la lista de objetivos descrita en el epígrafe 1.4.

ID	Actividad	Descripción	Duración
A	Estudio del problema	Se ha analizado el estado actual de las tecnologías, la recopilación de documentos y la adquisición de conocimientos previos.	20 días
B	Memoria	La escritura de la memoria se ha ido realizando progresivamente a medida que se hacían avances en el desarrollo del proyecto.	118 días
C	Búsqueda algoritmo	Se ha buscado una implementación que permita el entrenamiento y la instalación de requisitos en la máquina para su funcionamiento.	8 días
D	Elección de dataset	Se han establecido las clases a detectar y la búsqueda de conjuntos de datos interesantes para ello y se ha establecido la estructura de detección mediante tres detectores independientes.	14 días
E	Tratamiento de datos	Se ha realizado el tratamiento de los datos. Se ha dado el formato correspondiente y el etiquetado y balanceo de clases.	17 días
F.1	Entrenamiento señales	Se ha realizado el entrenamiento con el conjunto de datos de señales mediante distintas pruebas para la obtención de los mejores resultados.	14 días

F.2	Entrenamiento semáforos	Se ha realizado el entrenamiento con el conjunto de datos de semáforos.	12 días
G.1	Pruebas COCO	Se ha realizado una serie de pruebas del funcionamiento del detector con los datos de COCO y se ha validado las prestaciones con vídeos propios.	5 días
G.2	Pruebas señales	Se han realizado diversas pruebas del conjunto de datos de señales bajo distintas condiciones.	7 días
G.3	Pruebas semáforos	Se ha realizado el ajuste del conjunto de datos de semáforos para su correcta detección.	7 días
H	Ayuda conducción	Se ha desarrollado el sistema de ayuda a la conducción junto con la creación de las interfaces gráficas a mostrar.	15 días
I	Pruebas del sistema	Finalmente, se han realizado las pruebas y la validación del sistema al completo bajo grabaciones propias en distintas circunstancias de la conducción.	15 días

Tabla 9.1. Descripción de las tareas realizadas durante la realización del proyecto

Gráficamente, se detalla en el diagrama de Gantt de la figura 9.1 los procesos y el orden cronológico seguidos para la realización del proyecto al completo.

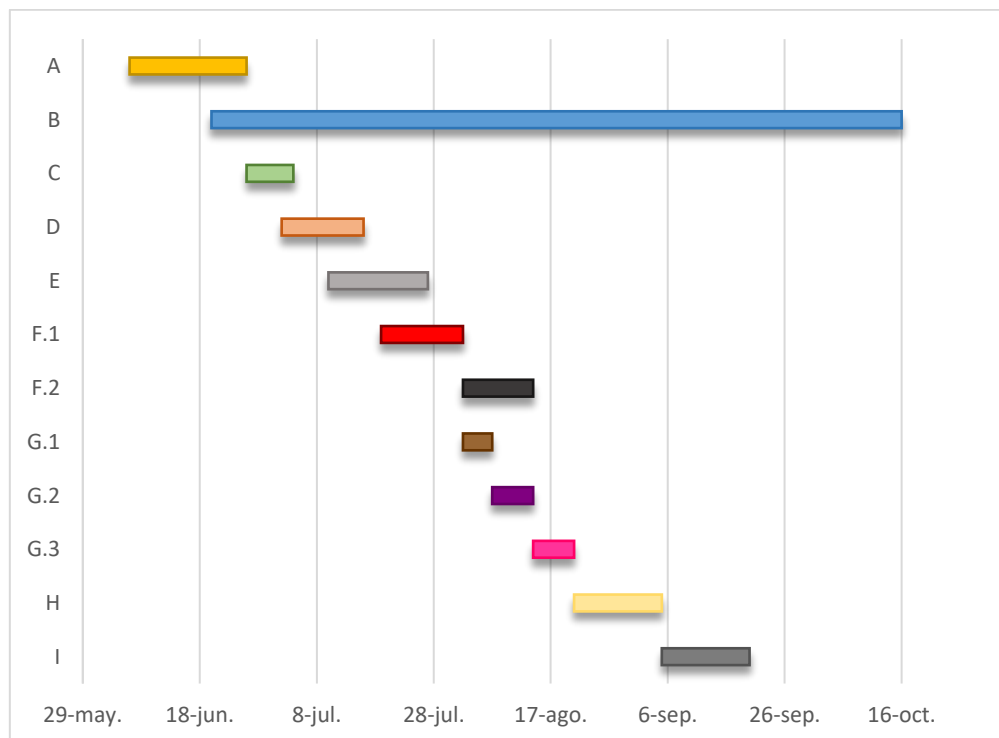


Fig. 9.1. Representación del diagrama de Gantt seguido en el proyecto

9.2. Presupuesto

En este epígrafe se va a realizar un desglose de los recursos dedicados al desarrollo del proyecto, incluyendo los costes materiales y personales junto con un posible presupuesto para la implementación del sistema.

9.2.1. Costes materiales

Para el desarrollo de la memoria y la programación del código se ha utilizado un ordenador portátil personal de gran potencia computacional. La elección de esta máquina surge de la necesidad de realización de pruebas con grandes conjuntos de imágenes. El ordenador utilizado se trata de un MSI valorado alrededor de 1500€. Teniendo en cuenta la longevidad media de un portátil de estas características y el tiempo utilizado en el desarrollo del proyecto se ha calculado una amortización de 80€. Este valor ha sido calculado mediante la fórmula mostrada en la ecuación 9.1.

$$\begin{aligned} \text{Amortización} &= \frac{(\text{Precio} - \text{Precio al final de la vida útil})}{\text{Vida útil}} \cdot \text{Duración proyecto} \\ &= \frac{1500\text{€} - 100\text{€}}{6 \cdot 12 \text{ meses}} \cdot 4 \text{ meses} \cong 80\text{€} \end{aligned}$$

Ecuación 9.1. Fórmula para el cálculo de la amortización

Más concretamente, el ordenador utilizado es el MSI Dominator GE62. Las características de este portátil se presentan en la siguiente lista:

- Procesador: Intel ® Core i7-6700HQ
- Tarjeta gráfica: Nvidia GeForce GTX 970M 3GB
- Memoria RAM: 16GB DDR4
- Espacio de almacenamiento: 1TB

Respecto a la grabación de imágenes ha sido posible gracias al uso de un vehículo ya en propiedad. Dado que su compra no fue realizada con la finalidad de uso en la elaboración del TFG, no se tendrá en cuenta su coste. Sin embargo, sí se incluirán los litros consumidos y su importe durante la grabación. Se ha estimado un gasto de 10€ en

gasolina. Del mismo modo que para el vehículo, no se tendrán en cuenta los gastos asociados al dispositivo de grabación.

Cabe comentar que el coste en software ha sido nulo, ya que tanto Python como sus librerías son de código abierto y se permite el uso libre y gratuito de esta plataforma. Para la realización de los entrenamientos de los detectores se barajó la posibilidad del alquiler de servidores para esta tarea, pero finalmente se optó por la opción gratuita mediante el uso de Google Colaboratory. Finalmente, para la realización de la memoria se ha usado Microsoft Office, programa que se puede adquirir gratuitamente mediante las licencias proporcionadas por la Universidad.

Por último se detalla el gasto en consumo eléctrico e Internet. Debido al gran periodo de tiempo dedicado al entrenamiento del algoritmo, este coste será más elevado de lo habitual. El precio estimado en factura eléctrica e Internet durante estos 4 meses es de 200€. El desglose total se detalla en la tabla 9.2.

Tipo de gasto	Coste
Hardware y software	80 €
Grabación y conducción	20 €
Suministros domésticos	200 €
TOTAL	290 €

Tabla 9.2. Desglose de gastos en costes materiales

9.2.2. Costes personales

Para el cálculo de los costes personales es necesario conocer el tiempo empleado en el proyecto por los distintos integrantes y el salario medio en función de la tarea desempeñada.

Como se ha comentado anteriormente, la duración del proyecto ronda los 4 meses, dedicando en torno a 9 horas diarias. El salario medio en España de un ingeniero junior de nueva incorporación se ha estimado en 12€/hora. Este presupuesto se ha aplicado también al copiloto encargado de la toma de imágenes, que ha requerido 4 horas.

Además, es necesario tener en cuenta el tiempo dedicado a la ayuda y supervisión del proyecto por parte del tutor. Entre reuniones, tutorías y tiempo dedicado a la corrección del documento, se ha empleado 30 horas. El salario medio de un ingeniero senior junto con la dirección de trabajos de fin de grado alcanza los 27 €/hora.

Teniendo todo esto en cuenta, se ha realizado en la tabla 9.3 el presupuesto dedicado a los costes personales.

Persona	Precio por hora	Horas dedicadas	Coste
Estudiante	12 €	1.080 h.	12.960 €
Copiloto	12 €	5 h.	60 €
Tutor	27 €	30 h.	810 €
TOTAL	-	-	13.830 €

Tabla 9.3. Desglose de gastos en costes personales

9.2.3. Costes totales

Si se realiza la suma de los gastos incluidos en el coste material y personal, teniendo en cuenta el 21% de IVA aplicado en España, se ha obtenido el presupuesto total dedicado al desarrollo del proyecto. Este desglose de gastos se muestra en la tabla 9.4.

Tipo de gasto	Coste
Coste material	290 €
Coste personal	13.830 €
Total	14.120 €
21% IVA	3.194 €
COSTE TOTAL con IVA	17.024 €

Tabla 9.4. Desglose de gastos totales del proyecto

9.2.4. Presupuesto para la integración

En este epígrafe se ha detallado un posible presupuesto necesario para la integración del sistema desarrollado en un vehículo.

En primer lugar, será necesario un procesador encargado de la administración de las tareas a realizar por el sistema. Además, se debe incorporar una tarjeta gráfica potente para el procesamiento de la imagen de entrada y su detección en tiempo real. Finalmente, es necesario un elemento para el almacenaje de datos. En este caso, al no guardar las imágenes recopiladas, no será necesario mucho espacio en el disco.

Además de los elementos utilizados para el procesamiento de las imágenes, serán necesarios ciertos periféricos. Para la toma de imágenes será necesario el uso de una pequeña cámara. Esta no requiere de gran resolución, ya que las imágenes de entrada al sistema serían demasiado pesadas y dificultarían la detección en tiempo real. También es necesaria la incorporación de un sistema de representación visual de imágenes. En este caso se optará por una pantalla de 7 pulgadas.

Para la conexión e integración final del sistema en el vehículo serán necesarios diversos elementos. Entre ellos se contará indispensablemente con varios metros de cableado, así como con un soporte para la fijación de la cámara delantera.

Otro punto a tener en cuenta en este apartado es la necesidad de mano de obra que realice el trabajo requerido. Se debe contar con un ingeniero cualificado que sea capaz de montar el sistema sobre el vehículo. El tiempo de instalación de los elementos supondrá alrededor de 6 horas.

Los datos referentes al presupuesto desglosado de la integración final del sistema en el vehículo se pueden observar en la tabla 9.5.

Elemento	Características	Precio
Procesador	Intell Core i7-8700K 3.7 GHz	370 €
Tarjeta gráfica	NVIDIA RTX 3080	615 €
Memoria RAM	Kingston HyperX Fury 16 GB	73 €
Disco almacenamiento	Sundisk SDSSDP-128G-G25	290 €

Cámara	Blink XT2	83 €
Pantalla	Ankeway 2 DIN 7''	107 €
Soporte	-	3 €
Cableado	-	15 €
Mano de obra	-	650 €
TOTAL	-	2.206 €

Tabla 9.5. Desglose de costes para posible integración del sistema desarrollado

10. CONCLUSIONES Y LÍNEAS FUTURAS DE DESARROLLO

A modo de cierre, en este último capítulo se recogen los aspectos más destacados descritos en el documento, así como las conclusiones obtenidas a partir de los resultados. Además se expone el rendimiento del sistema, junto a sus puntos fuertes y débiles. Finalmente, se comentarán ciertos aspectos y líneas futuras de desarrollo para la mejora del sistema.

10.1. Conclusiones

En el transcurso de este documento se ha detallado el procedimiento seguido para el desarrollo de un prototipo de sistema de ayuda a la conducción. La idea propuesta surge a raíz del desarrollo de nuevas tecnologías aplicables en el campo de la conducción, que permitan simplificar la tarea.

Para el desarrollo del prototipo se ha propuesto una estructura capaz del reconocimiento de elementos dinámicos, semáforos y señales, haciendo uso de uno de los algoritmos de visión artificial más punteros en la actualidad. La elección del algoritmo se ha basado en las cualidades más destacables de la quinta versión de YOLO, que lo colocan como principal candidato gracias a su rapidez de procesado. Esta decisión podría haber variado si en lugar de la velocidad, se hubiese valorado en mayor medida la precisión del sistema. Sin embargo, buscando un procesamiento en tiempo real de las imágenes recopiladas no se le ha dado tanta importancia a este punto, pese a obtenerse buenos resultados de precisión.

A partir de los resultados obtenidos tras la validación del sistema de detección, que se integra dentro del sistema global de ayuda a la conducción, se pueden sacar un par de conclusiones:

- Los detectores son robustos y devuelven la posición de los objetos con una alta precisión del 75% en buenas condiciones. No obstante, hay margen de mejora en los resultados obtenidos. Se pueden modificar tanto los parámetros de entrenamiento como los datos utilizados en el proceso buscando una solución más óptima.

- El tiempo de ejecución del programa de detección es lo suficientemente veloz como para aplicarse en tiempo real durante la conducción. Los tiempos obtenidos rondan los 10 fps. Nuevamente, este punto es mejorable mediante el uso de GPUs más potentes y mejorando la eficiencia del código.

Por otro lado, sin tener en cuenta resultados, el prototipo de sistema de ayuda a la conducción propuesto tiene varios puntos dignos de mención:

- Se implementa con ayuda de una cámara, un procesador y una tarjeta gráfica. No necesita, por tanto, de un hardware complejo y su integración en el vehículo no supondría un cambio drástico en la actual disposición de sus elementos.
- Las tecnologías utilizadas son accesibles a un gran número de personas, ya que con su desarrollo se han abaratado. Es decir, pese a no venir integrado de fábrica en el vehículo, podría ser un sistema que el propio usuario pueda acoplar al automóvil por un coste no muy elevado.
- De cara a garantizar la seguridad del uso del mismo, es necesario someter el sistema a pruebas en las cuales se enfrente a una situación real, no simulaciones realizadas desde un ordenador sobre grabaciones.
- El sistema, según el uso que quiera aplicarse del mismo, debe acogerse sobre la normativa existente. Las regulaciones cambian según el país y antes de utilizarse hay que verificar el correcto cumplimiento de la ley.
- Al igual que las leyes, las señales de tráfico varían dependiendo del territorio. Es por esto que el sistema tal y como se ha implementado solo es viable para su uso en Europa.

10.2. Líneas futuras de desarrollo

Aunque con el sistema implementado se consigan obtener buenos resultados, la tarea de la conducción y en especial, la conducción autónoma debe ir acompañada de grandes garantías en fiabilidad y seguridad. Para cumplir estos objetivos es necesario un continuo proceso de desarrollo y mejora de los sistemas. A continuación, se van a proponer futuras líneas de trabajo como continuación de este proyecto:

- En lo que respecta a la obtención de los conjuntos de datos, se puede realizar una búsqueda más amplia con el fin de obtener más imágenes de cada objeto. El *dataset* de COCO es suficientemente extenso y variado por lo que se obtienen muy buenos resultados en la detección. Sin embargo, los conjuntos de datos de señales y semáforos no son tan amplios, llevando a errores en la localización y clasificación de los objetos.
- Al hilo de lo comentado en el punto anterior, sería interesante realizar una unificación de los distintos *datasets* existentes para la obtención de uno más amplio, y con mayor variabilidad. Aunque las señales puedan parecer objetos inmutables en forma, las pequeñas variaciones puede inducir a errores al sistema, como puede ocurrir en el caso de señales de velocidad (similares en forma, donde solo varía el número en su interior). Otro ejemplo son los detalles que diferencian las señales entre países, como textos añadidos o rebordes.



10.1. Diferencias entre señales en función del país. España a la izquierda y Reino Unido a la derecha

- Lograr el uso de un único detector para todas las categorías, mediante la unificación de todos los detectores en uno solo en lugar de la aplicación por pasos de 3 distintos, haría más rápida la ejecución permitiendo el análisis de un mayor número de fotogramas cada segundo. Este aumento y estandarización de los datos haría también más fácil el ajuste del modelo eliminando la necesidad de múltiples entrenamientos independientes.
- Introducir un mayor número de clases correspondientes al grupo de señales de tráfico puede servir para la obtención de un sistema mucho más complejo, con mayor utilidad que la expuesta. Existen muchas señales que pueden influir en la velocidad del vehículo que se pueden integrar en versiones posteriores.
- Con la implementación de un sistema de detección de carril se podría dar una mayor utilidad a las detecciones de semáforos, llegando a diferenciar aquellos que

influyen al carril por el que se circula frente a los que no. De esta forma se podrían tomar decisiones en función de aquellos elementos que afecten al automóvil de una manera más precisa.

- Se pueden añadir más elementos de hardware para añadir complejidad al sistema, como pueden ser sensores como el LIDAR o mayor número de cámaras. Con mayor número de datos se pueden tomar decisiones más adecuadas de acuerdo a la situación. Además, sería interesante dotar al vehículo con comunicaciones GPS o bases de datos que permitan añadir más información del entorno y la vía por la que se circula.
- Por último, de cara a garantizar una mayor autonomía, se podría programar un sistema que actúe directamente sobre el vehículo, permitiendo que sea el propio automóvil el que actúe sin necesidad de interacción humana.

BIBLIOGRAFÍA

- [1] M. Wachs, “Futurama”, *Issues in Science and Technology*, vol. 26, n.º 4, pp. 90–92, 2010. [En línea]. Disponible en: <https://search.proquest.com/docview/746614600?accountid=14501>. Acceso: 28 Sep. 2020.
- [2] SAE International, “Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles”, *SAE International*, J3016, 2018. doi: https://doi.org/10.4271/J3016_201806.
- [3] “Piloto Automático”. Tesla. https://www.tesla.com/es_ES/autopilot?redirect=no. (acceso: 14 Oct. 2020) .
- [4] Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos). [En línea]. Disponible en: <https://www.boe.es/doue/2016/119/L00001-00088.pdf>. Acceso: 25 Sep. 2020.
- [5] Agencia Española de Protección de Datos, “Guía sobre el uso de videocámaras para seguridad y otras finalidades”, *Agencia Española de Protección de Datos*, 2018. [En línea]. Disponible en: <https://www.aepd.es/sites/default/files/2019-09/guia-videovigilancia.pdf>. Acceso: 25 Sep. 2020.
- [6] Reglamento n.º 79 de la Comisión Económica para Europa de las Naciones Unidas (CEPE). Prescripciones uniformes relativas a la homologación de vehículos por lo que respecta al mecanismo de dirección [2018/1947]. [En línea]. Disponible en: <https://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:42018X1947&from=EN>. Acceso: 25 Sep. 2020.
- [7] B. Braga. “What is Tesla Smart Summon?”. J. D. Power. <https://www.jdpower.com/cars/shopping-guides/what-is-tesla-smart-summon>. (acceso: 14 Oct. 2020).
- [8] Serokell. “Artificial Intelligence vs. Machine Learning vs. Deep Learning: What’s the Difference”. Medium. <https://medium.com/ai-in-plain-english/artificial-intelligence-vs-machine-learning-vs-deep-learning-whats-the-difference-dccce18efe7f> (acceso: 28 Sep. 2020).

- [9] *Diccionario de la Real Academia Española*, 23.^a ed. [En línea]. Disponible en: <https://dle.rae.es/inteligencia>. Acceso: 29 Sep. 2020.
- [10] S. Chandramouli, S. Dutt, y A. K. Das, *Machine Learning*, Pearson Education India, 2018. [En línea]. Disponible en: <https://learning.oreilly.com/library/view/machine-learning/9789389588132/>. Acceso: 29 Sep. 2020.
- [11] I. Casas, “Networks, Neural”, en *International Encyclopedia of Human Geography*, 2.^a ed. Elsevier, 2020, pp. 381–385. doi: <https://doi.org/10.1016/B978-0-08-102295-5.10410-X>.
- [12] Y. LeCun, Y. Bengio, y G. Hinton, “Deep learning”, *Nature*, vol. 521, n.º 7553, pp. 436–444, may. 2015, doi: <https://doi.org/10.1038/nature14539>.
- [13] S. Sieniutycz, “Systems science vs cybernetics”, en *Complexity and Complex Thermo-Economic Systems*. Elsevier, 2020, pp. 1–8. doi: <https://doi.org/10.1016/B978-0-12-818594-0.00001-5>.
- [14] J. M. Sempere y D. López, “A McCulloch-Pitts neural net to characterize even linear languages”, en *Information Processing Letters*, vol. 56, n.º 4, Elsevier, 1995, pp. 201–208. doi: [https://doi.org/10.1016/0020-0190\(95\)00153-4](https://doi.org/10.1016/0020-0190(95)00153-4).
- [15] L. N. Kanal, “Perceptrons”, en *International Encyclopedia of the Social & Behavioral Sciences*. Pergamon, 2001, pp. 11218–11221. doi: <https://doi.org/10.1016/B0-08-043076-7/00572-6>.
- [16] K. Abend, “The 2001 Benjamin Franklin Medal in Engineering presented to Bernard Widrow”, *Journal of the Franklin Institute*, vol. 339, n.º 3, pp. 283–294, 2002. doi: [https://doi.org/10.1016/S0016-0032\(01\)00044-8](https://doi.org/10.1016/S0016-0032(01)00044-8).
- [17] D. A. Medler, “A Brief History of Connectionism”, Proyecto de Computación Biológica, Dpto. de Psicología, Universidad de Alberta, Alberta, Canadá, 1998. [En línea]. Disponible en: <http://www.blutner.de/NeuralNets/Texts/Medler.pdf>. Acceso: 1 Oct. 2020.
- [18] M. Somasundaram, P. Latha, y S. A. S. Pandian, “Curriculum Design Using Artificial Intelligence (AI) Back Propagation Method”, *Procedia Computer Science*, vol. 172, pp. 134–138, 2020. doi: <https://doi.org/10.1016/j.procs.2020.05.020>.

- [19] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, y F. E. Alsaadi, “A survey of deep neural network architectures and their applications”, *Neurocomputing*, vol. 234, pp. 11–26, 2017. doi: <https://doi.org/10.1016/j.neucom.2016.12.038>.
- [20] A. Dertat. “Applied Deep Learning - Part 1: Artificial Neural Networks”. towards data science. <https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6>. (acceso: 2 Oct. 2020).
- [21] F. Marini, “Neural Networks”, en *Comprehensive Chemometrics*. Elsevier, 2009, pp. 477–505. doi: <https://doi.org/10.1016/B978-044452701-1.00128-9>.
- [22] F. B. Tek, “An adaptive locally connected neuron model: Focusing neuron,” *Neurocomputing*, vol. 419, pp. 306–321, 2021, doi: <https://doi.org/10.1016/j.neucom.2020.08.008>.
- [23] V. K. y S. K., “Towards activation function search for long short-term model network: A differential evolution based approach”, *Journal of King Saud University - Computer and Information Sciences*, may. 2020. [En línea]. Disponible en: <https://doi.org/10.1016/j.jksuci.2020.04.015>. Acceso: 2 Oct. 2020.
- [24] StackExchange. <https://tex.stackexchange.com/questions/519666/plotting-softmax-activation-function> (acceso: 6 Oct. 2020).
- [25] “Tangente hiperbólica”. Wikipedia. https://es.wikipedia.org/wiki/Tangente_hiperb%C3%B3lica (acceso: 6 Oct. 2020).
- [26] “Activation Functions: ReLU & Softmax”. MC.AI. <https://mc.ai/activation-functions-relu-softmax/> (acceso: 6 Oct. 2020).
- [27] H. Sharma. “Activation Functions: Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning”. Medium. <https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e> (acceso: 6 Oct. 2020).
- [28] S. Zare y M. Ayati, “Simultaneous fault diagnosis of wind turbine using multichannel convolutional neural networks”, *ISA Transactions*, ago. 2020. [En línea]. Disponible en: <https://doi.org/10.1016/j.isatra.2020.08.021>. Acceso: 2 Oct. 2020.
- [29] A. Dertat. “Applied Deep Learning - Part 4: Convolutional Neural Networks”. towards data science. <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>. (acceso: 3 Oct. 2020).

- [30] U. Michelucci. "Object Classification: An Introduction", en *Advanced Applied Deep Learning*. Apress, Berkeley, CA, 2019, pp. 195–220. doi: https://doi.org/10.1007/978-1-4842-4976-5_6.
- [31] A. Ouaknine. "Review of Deep Learning Algorithms for Object Detection". Medium. <https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852> (acceso: 6 Oct. 2020)
- [32] R. Girshick, J. Donahue, T. Darrell y J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", 2014, arXiv: 1311.2524.
- [33] R. Girshick, "Fast R-CNN", *Proceedings of ICCV*, pp. 1440-1448, 2015. doi: 10.1109/ICCV.2015.169
- [34] S. Ren, K. He, R. Girshick y J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39. n.º 6, pp. 91-99, pp. 1137–1149, 2017. doi: 10.1109/TPAMI.2016.2577031.
- [35] R. Gandhi. "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms". towards data science. <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e> (acceso: 6 Oct. 2020).
- [36] J. Liu et al., "High precision detection algorithm based on improved RetinaNet for defect recognition of transmission lines", *Energy Reports*, vol. 6, pp. 2430–2440, nov. 2020, doi: <https://doi.org/10.1016/j.egy.2020.09.002>.
- [37] P. Jay. "The intuition behind RetinaNet". Medium. <https://medium.com/@14prakash/the-intuition-behind-retinanet-eb636755607d#:~:text=RetinaNet%20is%20a%20single%2C%20unified,%2Dthe%2Dself%20convolution%20network.&text=Backbone%3A%20Feature%20Pyramid%20network%20built%20on%20top%20of%20ResNet50%20or%20ResNet101>. (acceso: 6 Oct. 2020).
- [38] W. Liu et al., "SSD: Single Shot MultiBox Detector", *Proceedings of the European Conference on Computer Vision*, pp. 21-37, sep. 2016, doi: https://doi.org/10.1007/978-3-319-46448-0_2.
- [39] J. Redmon. "YOLO: Real-Time Object Detection". pjreddie. <https://pjreddie.com/darknet/yolo/>. (acceso: 14 Oct. 2020)

- [40] P. Soviany y R. T. Ionescu, “Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction”, *Proceedings of the SYNASC*, pp. 209–214, 2018. doi: 10.1109/SYNASC.2018.00041.
- [41] 2018, “Coco Dataset” COCO Common Objects in Context, doi: <https://cocodataset.org/#download>.
- [42] J. Hui. “Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)”. Medium. https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 (acceso: 3 Oct. 2020).
- [43] G. Jocher. “Ultralytics/yolov5”. Github. <https://github.com/ultralytics/yolov5> (acceso: 7 Oct. 2020).
- [44] A. Bochkovski, C. Wang y H. Mark Liao, “YOLOv4: Optimal Speed and accuracy of Object Detection”, 2020, arXiv: 2004.10934.
- [45] G. Huang, Z. Liu, L. Van Der Maaten y K. Q. Weinberger, "Densely Connected Convolutional Networks", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017, doi: 10.1109/CVPR.2017.243.
- [46] Z. Huang, J. Wang, X. Fu, T. Yu, Y. Guo, y R. Wang, “DC-SPP-YOLO: Dense connection and spatial pyramid pooling based YOLO for object detection”, *Information Sciences*, vol. 522, pp. 241–258, 2020, doi: <https://doi.org/10.1016/j.ins.2020.02.067>.
- [47] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path Aggregation Network for Instance Segmentation”, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8759–8768, 2018, doi: 10.1109/CVPR.2018.00913.
- [48] A. Kathuria. “How to implement a YOLO (v3) object detector from scratch in PyTorch: Part 1”. Paperspace. <https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/> (acceso: 2 Oct. 2020).
- [49] “Anchor Boxes for Object Detection”. MathWorks. <https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html#:~:text=Anchor%20boxes%20are%20a%20set,sizes%20in%20your%20training%20datasets>. (acceso: 30 Sep. 2020).
- [50] “YOLO: Real-Time Object Detection”. TheBinaryNotes. <https://thebinarynotes.com/yolo-realtime-object-detection/>. (acceso: 14 Oct, 2020)

- [51] J. Redmon y A. Farhadi, “YOLO9000: better, faster, stronger.” *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263-7271, 2017, doi: 10.1109/CVPR.2017.690.
- [52] R. Rothe, M. Guillaumin, y L. Van Gool, “Non-Maximum Suppression for Object Detection by Passing Messages between Windows”, *Asian Conference on Computer Vision*, vol. 9003, 2015, doi: https://doi.org/10.1007/978-3-319-16865-4_19.
- [53] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, 2015, arXiv: 1506.02640.
- [54] J. Redmon y A. Farhadi, “Yolov3: An incremental improvement”, 2018, arXiv: 1804.02767.
- [55] G. Jocher. “Ultralytics/yolov3”. Github. <https://github.com/ultralytics/yolov3/issues/102> (acceso: 6 Oct. 2020).
- [56] K. Behrendt, L. Novak y R. Botros. 2017, “A deep learning approach to traffic lights: Detection, tracking, and classification” Heidelberg Collaboratory for Image Processing, doi: 10.1109/ICRA.2017.7989163.
- [57] Berktepebag. “berktepebag/Traffic-light-detection-with-YOLOv3-BOSCH-traffic-light-dataset”. Github. <https://github.com/berktepebag/Traffic-light-detection-with-YOLOv3-BOSCH-traffic-light-dataset> (acceso: 3 Oct. 2020).
- [58] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing y C. Igel. 2013, “Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark” Electronic Research Data Archive, doi: <https://doi.org/10.17894/ucph.358970eb-0474-4d8f-90b5-3f124d9f9bc6>.
- [59] Enicck. “tzutalin/labelImg”. Github. <https://github.com/tzutalin/labelImg> (acceso: 3 Oct. 2020).
- [60] J. Hui. “mAP (mean Average Precision) for Object Detection”. Medium. https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. (acceso: 14 Oct. 2020).

EXTENDED ABSTRACT

Introduction

The motivation behind this project arises from the continuous developments in the artificial intelligence field, as well as from the recent incorporation of this technology in the construction of autonomous vehicles.

Following this idea, with the execution of this final project it is aimed to accomplish an initial development on a driving assistance system. The system will be implemented by means of You Only Look Once (YOLO) algorithm, such algorithm being in charge of the detection of dynamic elements, traffic signs and traffic lights.

First, an analysis on the state of the art has been carried out, distinguishing the different branches of artificial intelligence and focusing on the structure and functioning of neural networks. Also, the importance of neural networks on deep learning architectures has to be taken into account.

Neural Networks

Moving on to the explanation concerning neural networks and how they work some points can be noted. This kind of network is based on an urge to simulate the behaviour of human brains by machines. This way, a computer is able to do chores which were previously only executed by men. Its basic unit consist of the neuron. Neurons are basic processing units with input connexions which receive external data. This information will also be accompanied by some values known as weights, which are used to convey the importance of each of the input characteristics. Inside the neuron, a pondered sum of all the weights in addition with the incoming information is carried out in order to compute the output. Given that the obtained result is a linear function, it will be difficult to divide the variables into different categories if they are scattered. Due to this matter, the hyperplane whose objective is to distinguish the samples is bounded to be flexible so that it can adjust to their distribution. The variability that is aimed to have can be obtained in two different ways: distorting the output by means of non-linear functions or by increasing the number of neurons and grouping them in different layers.

Next, an explanation on both proposed methods can be given. In case the decision is to distort the output by means of non-linear functions, we need to select a function

depending on the network and the final task. Some of the most used ones are the hyperbolic tangent function, the sigmoid function, the ReLU function and Mish function. The last one is the function employed in the selected algorithm.

On the other hand, if the second method is encountered, understanding Deep Learning is required. Deep Learning is a field that groups neurons into various layers. Networks implemented following such method have three parts: input layer, hidden layers and output layer. The data arrive at the input and proceed to the hidden layers, where the feature extraction is done. Finally, all the processed data goes through the output layer and the final classification is done there.

Convolutional Neural Networks

With the numerous developments done in the field, the number of hidden layers used in this kind of network is rapidly increasing. A growth on the complexity level can be accomplished thanks to the application of convolutional layers. On these layers, the feature map is reduced in size. Convolutional Neural Networks receive their name from this type of layers found in their structure. The function of these layers is to convolve, which corresponds with applying smaller size filters to the input data. After that, data can be classified into different regions and its information is reduced to one single feature. The filters used are called kernels and they use some weights as well as they move through the picture depending on some variables: stride and padding. Although convolutional layers may be thought of as the unique layers in CNN, this network uses other type of layers too. Between these layers pooling layers can be found, and they are used to reduce even further the size of the information in the network.

Convolutional Neural Networks can be divided in two groups depending on the structure the algorithm follows. When the inner structure is divided into two different phases, one in charge of obtaining regions of interest and other that returns the detection. In this first group of algorithms the R-CNN family can be highlighted. However, some networks are able to perform the whole task in one phase. They will form the second group. To be found between the one-phase algorithms there are YOLO and SSD.

Algorithm selection

In order to choose the algorithm that will perform the detection in the implemented system, the previous information has to be taken into account. R-CNN algorithms offer high precision results on detection. However, the frame processing time used by the

network is too high to be able to fulfil the requirements. On contrast to these family of algorithms, YOLO belongs to the one-phase structure category. With the data only having to go through one phase, processing time is reduced considerably, making possible real-time object detection. Although the results do not have such a good precision as the ones in R-CNN, there is no remarkable difference. Therefore, the selected algorithm to perform detection will be You Only Look Once.

YOLO

YOLO algorithm has evolved during the past few years, leading to the existence of 5 different versions. The one version to be used in the implementation of the system will be the latest one. Nevertheless, this version is still being developed and there is no much information to be found about its structure. Hence, the analysis of its functioning and structure will be based on information found about YOLOv4.

As it has been mentioned, YOLO's structure is composed of one stage. However, its inner structure is then divided into three parts: backbone, neck and head. At the backbone, a feature extraction is done by means of CSPDarknet-53. This is a network formed by dense blocks, which are in charge of making bigger the reception area. Then, the CSP module is responsible for the division of the output into groups so that Darknet-53 can proceed to the generation of the most relevant feature maps. At the neck, a Path Agregation Network (PAN) is used. There, the lower level features propagate through the upper layers taking an ascending path between the different layers. Also, there can be found some shortcuts that lead to the deeper layers in the network. Finally, at the head, the detection can be done.

The chosen algorithm proceeds with the detection by dividing the input image into a grille. Each of the subdivisions must detect the objects whose centre lays in their corresponding cell. To be able to detect several objects per cell anchor boxes can be used. These are boxes whose size is predetermined depending on the class, usually associated to a probability that shows the certainty of an object laying in them or not. In the end, the proposed bounding boxes can be computed through the size of the anchor boxes.

Driving assistance system

To implement the desired system, we have selected to apply three independent detectors. Each of these detectors will detect a different group of classes. The first detector is based on COCO dataset, using the available pre-trained weights, to detect dynamic objects such

as cars or people. The second detector shall be able to distinguish the different lights in traffic lights and has been trained with Bosh dataset. Finally, the last detector will be used to find traffic signs in the video. It has been trained with GTSDb dataset, combined with manually labelled images.

During the training process the chosen parameters have been the batch size and the epoch. The batch indicates how many images are processed per iteration, and the epoch tells the system how many iterations should be done. To be able to accomplish good results a batch of 16 and 1000 iterations are enough. This can be checked by computing the value of precision, recall and average precision for some IoU values.

The last step of the implementation is to code the functionality of the driving assistance, integrating the different detectors in the process. The main functionality will show the maximum speed by checking the traffic signs detected. This maximum speed will change to 0 to indicate some situations in which the driver should reduce driving speed. The situations in which the system will operate are:

- When there is a Stop sign close enough.
- When there is a pedestrian crossing sign and a pedestrian can be detected in the frame.
- When a yellow or red traffic lights is detected.
- When there is an object too close to the vehicle.

Furthermore, it has been implemented for the system to show on the screen the detected traffic signs as well as the colour of the light in traffic lights. As speed limit signs are already used to show maximum speed, only Stop, pedestrian crossing and give way signs are to be showed.

Conclusions and future development possibilities

Through the described proceedings that can be found in this file, it was possible to develop a basic driving assistance system by means of artificial vision. Thanks to the application of YOLO algorithm to do the detection task the construction of the system was attainable. With this algorithm it is viable to get rather good results during validation of position and classification of the objects detected while driving, as well as to provide results in a short enough time that allows real-time detection.

Meanwhile, although it is bound to bring good results to the table, some new features and functionalities could be added to improve its performance. Among them, the two worth of more attention should be the introduction of a greater number of different traffic signs and being able to unify the three detectors into one. Then, related to a real-life implementations, the introduction of some peripherals could also improve the system. Having a bigger amount of data entering the system, there could be a comparison of different input information in order to increase precision and decrease the failure probability.