UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Programación y simulación de sistema de reconocimiento de señales de tráfico en carretera para la automatización de vehículos

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Aparisi Pérez, Daniel

Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO: 2021/2022

ÍNDICE DOCUMENTO Nº1 MEMORIA

ÍNDICE FIGURAS DOCUMENTO Nº1 MEMORIA

ÍNDICE TABLAS DOCUMENTO Nº1 MEMORIA

ÍNDICE CÓDIGOS DOCUMENTO Nº1 MEMORIA

1. Objeto

1.1. Objetivo principal

El objeto de este proyecto es el modelado y programación de un sistema de visión artificial para el reconocimiento de señales de tráfico en carretera. El sistema estará basado en una inteligencia artificial construida mediante una red neuronal convolucional.

Además, se diseñará un modelo del sistema en funcionamiento en el entorno de desarrollo 3D de Unity. Esta simulación tendrá por objetivo obtener los datos para el entrenamiento de la inteligencia artificial en diferentes condiciones y servir de prueba para los experimentos con el programa ya terminado.

1.2. Otros objetivos

Además del objeto principal general del proyecto, también se buscará completar otros objetivos más específicos.

El primero de ellos será la toma y justificación de un número de muestras suficiente que permita el desarrollo de un sistema de identificación robusto. La cantidad final deberá ser el resultado de la evidencia experimental y demostrar ser suficiente para llevar a cabo la creación del producto final.

Otro de los objetivos será la elección de una arquitectura para la red neuronal que constituya un modelo de identificación con un equilibrio adecuado entre rendimiento y costes computacionales. Se tratará de optimizar la cantidad y complejidad de capas utilizadas.

Finalmente, será obligatoria la elaboración de un sistema de evaluación objetivo y completo para la comparativa entre los distintos modelos producidos durante la creación. Dicho sistema de evaluación deberá de contar con diversos indicadores y ser lo suficientemente precisa y clara para que resulte sencillo diferenciar el modelo con el mejor comportamiento. Esta evaluación se deberá de realizar tomando las predicciones de un número elevado de ensayos de los diferentes modelos sobre un conjunto de datos común. De esta forma se pretende que la comparativa se lleve a cabo de la manera más homogenea posible.

2. Objetivo de Desarrollo Sostenible

Será también objeto de este proyecto la elaboración de un sistema de asistencia a la conducción que pueda sentar las bases de los sistemas de conducción autónoma. Dichos sistemas buscan hacer de la conducción una actvidad segura, eficiente y fluida, con lo que se logra que los vehículos saquen el mayor partido posible a los carburantes y disminuyan la congestión de las carreteras, a fin de reducir el tiempo de operación de los vehículos, y por tanto, las emisiones.

Este punto se alinea con el Objetivo de Desarrollo Sostenible número 11, *Ciudades y Comunidades Sostenibles.*

Figura 1 ODS 11

Concretamente los puntos 11.2 y 11.3:

11.2 De aquí a 2030, proporcionar acceso a sistemas de transporte seguros, asequibles, accesibles y sostenibles para todos y mejorar la seguridad vial, en particular mediante la ampliación del transporte público, prestando especial atención a las necesidades de las personas en situación de vulnerabilidad, las mujeres, los niños, las personas con discapacidad y las personas de edad

11.3 De aquí a 2030, aumentar la urbanización inclusiva y sostenible y la capacidad para la planificación y la gestión participativas, integradas y sostenibles de los asentamientos humanos en todos los países

Figura 2 ODS 11.2 y 11.3

Y el 11.6:

11.6 De aquí a 2030, reducir el impacto ambiental negativo per capita de las ciudades, incluso prestando especial atención a la calidad del aire y la gestión de los desechos municipales y de otro tipo

Figura 3 ODS 11.6

3. Resumen

3.1. Castellano

Los sistemas de asistencia a la conducción son cada vez más empleados en todos los medios de transporte actuales, dispositivos encargados de mejorar el desempeño de los vehículos en aspectos como la seguridad, la fiabilidad, el consumo eficiente de carburantes, entre muchos otros. Desde sistemas más sencillos como el control de velocidad de crucero de los automóviles hasta los complejos sistemas de navegación de las aeronaves. El avance en la tecnología de los sistemas de monitorización ha logrado ofrecer una gran variedad de métodos para el análisis del entorno por parte de los sistemas de asistencia a la conducción, se incluyen aquí sensores térmicos, tecnología RADAR y SONAR, geolocalización satélite, etc. De entre todas estas opciones las que mejor se asemejan a la forma que tienen los humanos de interpretar el entorno son las que se basan en la visión artificial.

La motivación de este proyecto es la aplicación de los sistemas de asistencia a la conducción basados en la visión artificial para el reconocimiento de señales de tráfico en carretera. Este sistema deberá ser entrenado para identificar los distintos patrones que caracterizan cada una de las señales de tráfico dispuestas en la simulación. Es por ello que se necesitará recopilar los datos tomados por el vehículo simulado en forma de imágenes, imágenes que serán entregadas al algoritmo del programa para su entrenamiento.

El proyecto por lo tanto ha conllevado, en primer lugar, el modelado 3D de un entorno simulado que cuenta con el vehículo a asistir, la cámara de captación de imágenes, las señales de tráfico y distintos elementos ambientales que buscan reproducir algunos de los obstáculos reales a los que se ven expuestos estos sistemas, véase vegetación, gradientes lumínicos, posicionamiento variable del vehículo en el carril entre otros. En segundo lugar, se han recogido las imágenes tomadas por la cámara de captación y se han clasificado en clases codificadas, una por cada señal a reconocer. A continuación, se realizó el programa que debía tomar las imágenes, procesarlas y utilizarlas para crear un modelo entrenado. Por último, se puso a prueba el modelo en el mismo entorno simulado utilizado para la captación de imágenes. En esta fase era el modelo el que debía clasificar las diferentes señales. Con los resultados obtenidos se pudo finalmente evaluar el programa.

3.2. Valenciano

Els sistemes d'assistència a la conducció cada vegada són més emprats en tots els mitjans de transport actuals, dispositius encarregats de millorar l'acompliment dels vehicles en aspectes com la seguretat, la fiabilitat, el consum eficient de carburants, entre molts d'altres. Des de sistemes més senzills com el control de velocitat de creuer dels automòbils fins als complexos sistemes de navegació de les aeronaus. L'avançament en la tecnologia dels sistemes de monitoratge ha aconseguit oferir una gran varietat de mètodes per a l'anàlisi de l'entorn per part dels sistemes d'assistència a la conducció, açí s'inclouen sensors tèrmics, tecnologia RADAR i SONAR, geolocalització satèl·lit, etc. D'entre totes aquestes opcions, les que s'assemblen millor a la forma que tenen els humans d'interpretar l'entorn són les que es basen en la visió artificial.

La motivació daquest projecte és l'aplicació dels sistemes d'assistència a la conducció basats en la visió artificial per al reconeixement de senyals de trànsit a la carretera. Aquest sistema haurà de ser entrenat per identificar els diferents patrons que caracteritzen cadascun dels senyals de trànsit disposats a la simulació. És per això que

caldrà recopilar les dades preses pel vehicle simulat en forma d'imatges, imatges que seran lliurades a l'algoritme del programa per al seu entrenament.

El projecte ha comportat, en primer lloc, el modelatge 3D d'un entorn simulat que compta amb el vehicle a assistir, la càmera de captació d'imatges, els senyals de trànsit i diferents elements ambientals que busquen reproduir alguns dels obstacles reals als que es veuen exposats aquests sistemes, vegeu vegetació, gradients lumínics, posicionament variable del vehicle al carril entre altres. En segon lloc, s'han recollit les imatges preses per la càmera de captació i s'han classificat a classes codificades, una per cada senyal a reconèixer. A continuació, es va fer el programa que havia de prendre les imatges, processar-les i utilitzar-les per crear un model entrenat. Finalment, es va posar a prova el model al mateix entorn simulat utilitzat per a la captació d'imatges. En aquesta fase era el model el que havia de classificar els diferents senyals. Amb els resultats obtinguts finalment es va poder avaluar el programa.

3.3. Inglés

Driving assistance systems are increasingly used in all means of transport today, devices that improve vehicle performance in areas such as safety, reliability, fuel efficiency, among many others. From the simplest systems such as cruise control in cars to complex navigation systems in aircraft. Advances in monitoring system technology have provided a wide variety of methods for the analysis of the environment by driver assistance systems, including thermal sensors, RADAR and SONAR technology, satellite geolocation, etc. Of all these options, the ones that best resemble the way humans interpret the environment are those based on artificial vision.

The motivation of this project is the application of driving assistance systems based on artificial vision for the recognition of road traffic signs. This system will have to be trained to identify the different patterns that characterise each of the traffic signs arranged in the simulation. It will therefore need to collect the data taken by the simulated vehicle in the form of images, which will be delivered to the algorithm for training.

The project has therefore involved, firstly, the 3D modelling of a simulated environment with the vehicle to be assisted, the image capture camera, the traffic signs and different environmental elements that seek to reproduce some of the real obstacles to which these systems are exposed, such as vegetation, light gradients, variable positioning of the vehicle in the lane, among others. Secondly, the images taken by the capturing camera were collected and classified into coded classes, one for each signal to be recognised. Next, the software was developed to take the images, process them and use them to create a trained model. Finally, the model was tested in the same simulated environment used for image acquisition. In this phase it was up to the model to classify the different signals. With the results obtained, the programme could finally be evaluated.

4. Antecedentes

Para este proyecto se pretende lograr un producto que pueda ser de utilidad en el avance de las tecnologías de automatización y modernización de los vehículos para transporte de pasajeros.

Uno de los aspectos más importantes cuando se habla de la conducción es la seguridad.

Todo medio de transporte de personas conlleva un riesgo inherente a la actividad, no obstante, este riesgo puede ser minimizado cuando se aplican las técnicas y tecnologías necesarias.

El sector de la automatización requiere minimizar el impacto del mayor factor de riesgo en la conducción: las personas. Es por ello que de un tiempo a esta parte se han multiplicado los esfuerzos por lograr un automóvil autónomo, un vehículo que no esté sujeto a distracciones, estados de ánimo, dolencias físicas, y demás problemas asociados a la conducción humana.

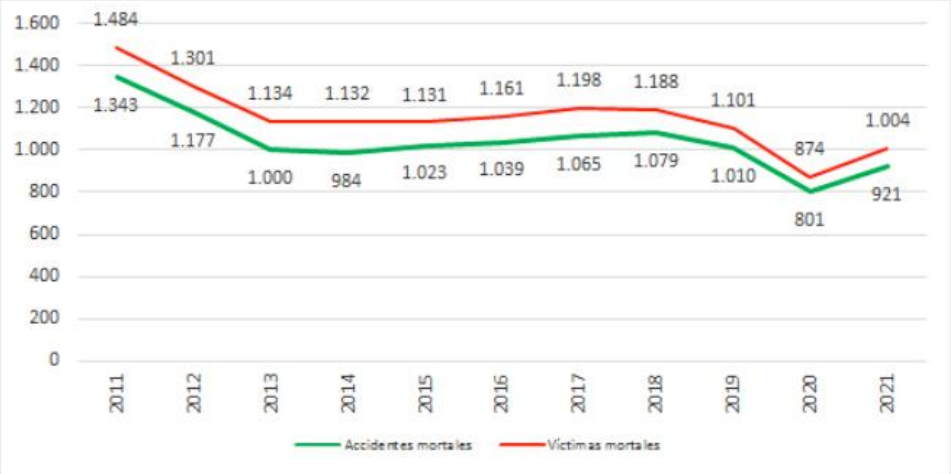

Figura 4 Accidentes y víctimas mortales en España por año

https://revista.dgt.es/es/noticias/nacional/2022/01ENERO/0107-Balance-prov-accidentalidad-2021.shtml

Por otro lado, la sociedad contemporánea tiene, desde hace unas décadas, un nuevo desafío al que enfrentarse. La consecuencia de un sistema industrializado y globalizado. La contaminación del medioambiente.

Como se expuso con anterioridad (Ver apartado 2, Objetivo de Desarrollo Sostenible), uno de los mayores motores de impulso para la realización de vehículos automatizados es la ecologización de las ciudades y los medios de transporte. Las emisiónes producidas por los vehículos de combustión suponen una parte relevante de la contaminación del aire en los núcleos urbanos. Es por ello que resulta importante buscar sistemas y tecnologías que permitan reducir la distancia y duración de los recorridos a la vez que se mejora la fluidez en las carreteras.

Con el problema a resolver presente y explicado, se expondrá el estado en el que se encuentran algunas de las tecnologías en las que se basa este proyecto.

4.1. Visión artificial

La visión es quizás el más importante de los sentidos humanos. Nos proporciona, aparentemente sin esfuerzo, una detallada descripción tridimensional (3-D) de un mundo complejo y rápidamente cambiante.

La visión por ordenador, el estudio que permite a los ordenadores comprender e interpretar la información visual a partir de imágenes estáticas y secuencias de vídeo, surgió a finales de los años cincuenta y principios de los sesenta y se está expandiendo rápidamente por todo el mundo. Pertenece al campo más amplio de la computación relacionada con la imagen y se relaciona con áreas como el procesamiento de imágenes, la visión robótica, las imágenes médicas, las bases de datos de imágenes, el reconocimiento de patrones, los gráficos por ordenador y la realidad virtual. (G. Bebis et al., 2003)

La detección de objetos es una tarea fundamental y desafiante en la visión por ordenador. Puede tratarse como una combinación de clasificación y localización, pero deben detectarse y clasificarse al mismo tiempo múltiples objetos con diferentes escalas dentro de una imagen.

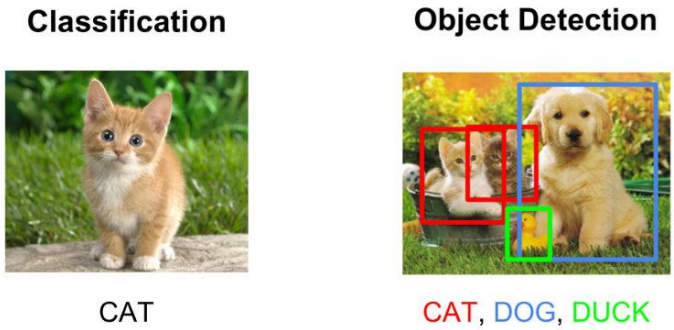

Figura 5 Diferenciación entre clasificación y detección de objetos

https://www.datacamp.com/tutorial/object-detection-guide

La detección de objetos ha recibido una amplia atención y se ha aplicado en muchos otros campos como la conducción autónoma, la vigilancia, etc. (Zhang et al., 2019).

Los avances en la potencia de cálculo y la disponibilidad de datos digitales han propiciado un progreso significativo en los algoritmos de inteligencia artificial (IA). Por ello, tanto la comunidad científica como la prensa no especializada siguen publicando a gran velocidad nuevas e innovadoras aplicaciones de la IA en el ámbito sanitario. La IA es el campo de la informática que se centra en el desarrollo de algoritmos que permiten a las máquinas dar respuestas racionales de alto nivel, interactuar y realizar funciones cognitivas y perceptivas avanzadas. Un área de la IA que se ha desarrollado especialmente en la última década es la visión por ordenador, un campo científico interdisciplinar que trata de cómo los ordenadores pueden obtener una comprensión de alto nivel de las imágenes digitales o los vídeos y la capacidad de realizar funciones, como la identificación y el seguimiento de objetos y el reconocimiento de escenas.

Varios campos de la medicina han tenido un éxito significativo en el desarrollo de modelos de IA capaces de realizar una serie de funciones de diagnóstico utilizando la visión artificial (por ejemplo, la identificación de anomalías en el diagnóstico radiológico, la identificación de lesiones cutáneas malignas y la interpretación de electrocardiogramas), y existe un potencial de éxito similar en especialidades de procedimiento como la cirugía.

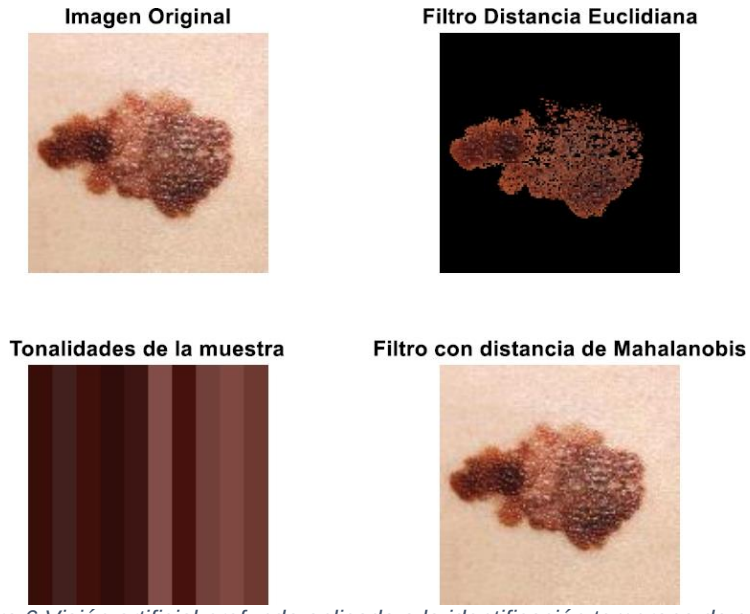

Figura 6 Visión artificial profunda aplicada a la identificación temprana de cáncer

https://www.scielo.org.mx/scielo.php?pid=S1405-55462020000200751&script=sci_arttext_plus&tlng=es

Tanto los médicos como los innovadores han tratado de desarrollar algoritmos de IA capaces de mejorar nuestra capacidad de proporcionar intervenciones terapéuticas, como por ejemplo con el apoyo a la toma de decisiones en tiempo real y la cirugía asistida por ordenador (Kitaguchi et al., 2022).

4.2. Redes neuronales

Mucho antes de la Edad Media, los filósofos y matemáticos chinos, indios y griegos ya habían empezado a investigar el razonamiento formal. En la década de 1930, varios matemáticos y lógicos intentaron formalizar la noción de computabilidad, es decir, la capacidad de resolver un problema con eficacia. En la década de 1940, los científicos empezaron a investigar si las máquinas podían pensar como los humanos. Un colaborador notable fue Alan Turing, que publicó una serie de trabajos sobre el tema de las máquinas inteligentes desde finales de los años 40 hasta principios de los 50.

Por otro lado, los fundamentos de las redes neuronales biológicas fueron construidos a finales del siglo XIX por filósofos y psicólogos como Alexander Bain y William James. Bain y James propusieron de forma independiente que las interacciones entre las neuronas del cerebro dan lugar a pensamientos y actividades corporales. Los estudios de lógica matemática y psicología acabaron conduciendo a la creación de la IA. En 1943, el neurofisiólogo Warren Sturgis McCulloch y el lógico Walter Pitts implementaron las primeras neuronas artificiales que modelaban las unidades lógicas básicas de un cerebro. Un conjunto de estas neuronas artificiales se denomina Red Neural Artificial (RNA), que se inspira en las redes neuronales biológicas que constituyen los cerebros de los animales. (T. Hong et al., 2022)

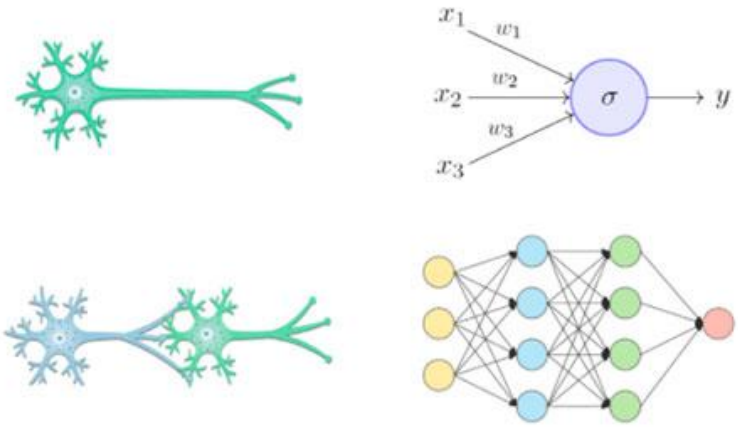

Figura 7 Analogía entre conexiones sinápticas y redes neuronales computacionales

https://www.futurespace.es/redes-neuronales-y-deep-learning-capitulo-2-la-neurona/

Dentro del campo de la informática, el aprendizaje automático, que es un subconjunto de la IA, se ha aplicado ampliamente a áreas como el reconocimiento del habla, el procesamiento del lenguaje natural, el control de robots y la visión por ordenador. Los enfoques convencionales de aprendizaje automático están limitados en su capacidad para procesar datos en su forma bruta. La incapacidad para procesar datos se debe a que se requiere una cantidad considerable de conocimientos de ingeniería y de dominio para diseñar un extractor de características.

El aprendizaje profundo es un método de representación que puede utilizarse para extraer características sofisticadas a altos niveles de abstracción de forma automática. Este método también puede aprender de datos con múltiples niveles de representaciones de extremo a extremo. Combinando los métodos de aprendizaje profundo (por ejemplo, las redes neuronales) con la visión por ordenador, se pueden extraer automáticamente las características de la imagen y utilizarlas para aprender de los datos de entrenamiento.

Un tipo particular de método de aprendizaje profundo que se ha utilizado ampliamente es la CNN (Redes Neuronales Convolucionales), ya que ha sido capaz de superar con precisión y fiabilidad a otras redes neuronales profundas en áreas como la clasificación de imágenes, la detección de objetos y la segmentación.

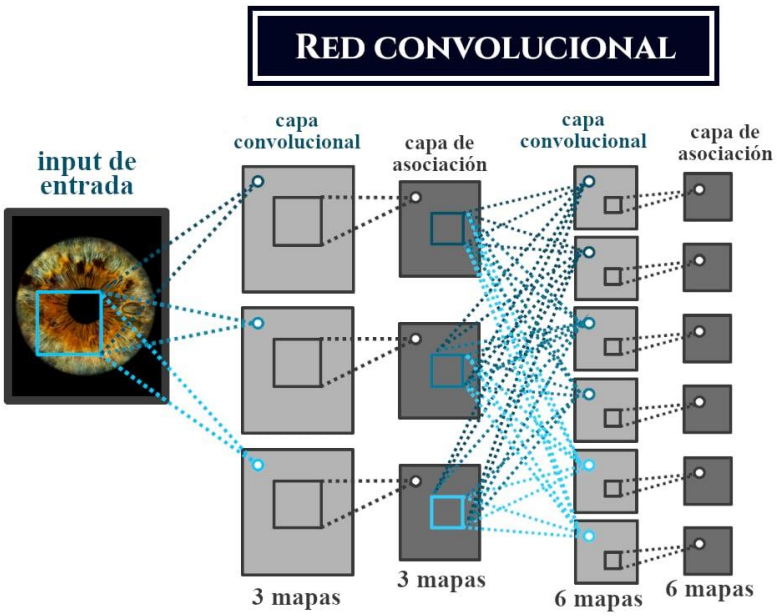

Figura 8 Esquema de ejemplo de la arquitectura de una CNN simple

https://lamaquinaoraculo.com/computacion/redes-neuronales-convolucionales/

El aprendizaje profundo ha permitido que prosperen los desarrollos de aplicaciones basadas en la visión por ordenador por ejemplo, los vehículos autónomos y el diagnóstico automático del cáncer de mama o de piel. (Weili Fang et al., 2020)

Los fundamentos de las redes neuronales convolucionales comenzaron con el descubrimiento de Hubel y Wisel en 1959 . Según ellos, las células de la corteza visual animal reconocen la luz en el pequeño campo receptivo.

En 1980, inspirado en este trabajo, Kunihiko Fukusima propuso el neocognitrón. Esta red se considera el primer modelo teórico de CNN. En 1990, LeCun et al. desarrollaron el marco moderno de CNN llamado LeNet-5 para reconocer dígitos escritos a mano. El entrenamiento mediante el algoritmo de retropropagación ayudó a LeNet-5 a reconocer patrones visuales a partir de imágenes sin procesar directamente, sin utilizar ninguna ingeniería de características por separado. Pero en aquellos días, a pesar de los diversos méritos, el rendimiento de la CNN en problemas complejos se vio afectado por los limitados datos de entrenamiento, la falta de innovación en el algoritmo y la insuficiente potencia de cálculo. Recientemente disponemos de grandes conjuntos de datos etiquetados, algoritmos innovadores y potentes máquinas GPU.

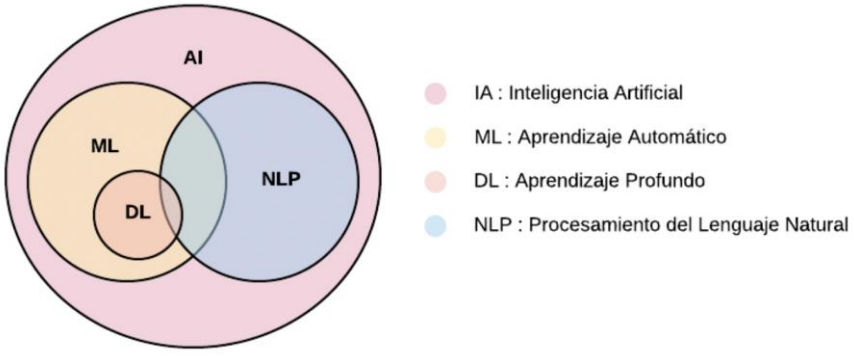

Figura 9 Procesamiento del Lenguaje Natural dentro del contexto de la inteligencia artificial

En 2012, con estas mejoras, una gran CNN profunda, llamada AlexNet, diseñada por Krizhevsky et al. mostró un excelente rendimiento en el ILSVRC. El éxito de AlexNet allanó el camino para inventar diferentes modelos de CNN, así como para aplicar esos modelos en diferentes campos de la visión por ordenador y el procesamiento del lenguaje natural (Ghosh et al., 2020).

4.3. Vehículos asistidos y autónomos

Se considera que la industria del automóvil está llevando a cabo un monumental cambio de paradigma, pasando de los vehículos manuales a los semiautónomos y a los totalmente autónomos.

La demanda de vehículos autónomos ha crecido enormemente con el creciente número de accidentes de tráfico causados por la distracción del conductor.

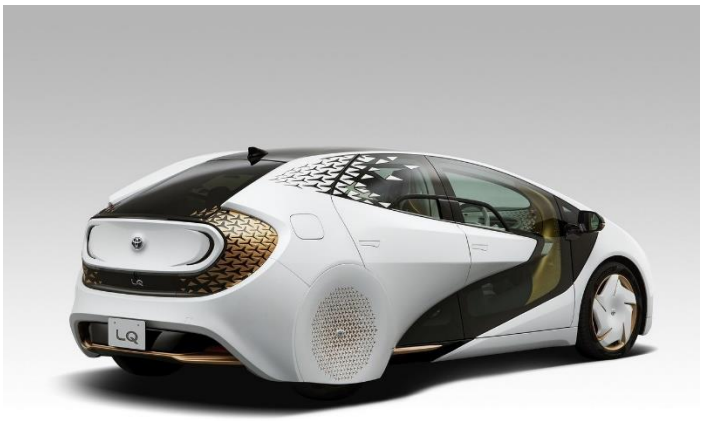

Figura 10 Prototipo de coche autónomo del fabricante Toyota

En función del nivel de autonomía de los vehículos, es posible contar con sistemas autónomos que se limiten a asistir al conductor para un funcionamiento seguro del vehículo o que puedan asumir el control total del mismo.

Los vehículos autónomos parecen tener numerosas ventajas para la seguridad vial, la optimización del tráfico, las aplicaciones militares o médicas en las que sólo es deseable una mínima intervención humana, etc. (Kavya P. Divakarla et al., 2019)

Para conducir sin intervención humana, un vehículo autónomo debe tener en cuenta la situación general, lo que requiere cinco funciones principales: localización, percepción, planificación, control del vehículo y gestión del sistema.

El módulo de localización se encarga de estimar la posición del vehículo, y el módulo de percepción crea un modelo del entorno de conducción a partir de datos fusionados con información de múltiples sensores. A partir de la información de localización y percepción, el módulo de planificación determina las maniobras del vehículo para una navegación segura.

El módulo de control del vehículo sigue la orden deseada por el módulo de planificación controlando la dirección, la aceleración y el frenado. Por último, el módulo de gestión del sistema supervisa todo el sistema de conducción autónoma. Sin embargo, el proceso se vuelve mucho más sofisticado cuando hay más elementos que considerar, como otros vehículos, peatones o ciclistas en la carretera.

Por lo tanto, para permitir la comunicación entre un vehículo autónomo y otros elementos de la carretera, un sistema de comunicación del vehículo se convierte en un componente indispensable y crítico en un vehículo autónomo. Este sistema de comunicación se conoce comúnmente como comunicación de vehículo a todo, que incluye varios escenarios como el de vehículo a vehículo, vehículo a infraestructura, vehículo a peatón y vehículo a red. (Chan, T.K. et al., 2021)

5. Materiales y métodos

El diseño y la implementación de un sistema de asistencia a la conducción basado en la visión artificial requiere de una serie materiales y métodos a seguir.

A continuación, se hará una distinción entre los materiales y métodos que han sido necesarios para desarrollar el sistema simulado y los materiales y métodos que se requerirían en caso de que se quisiera llevar a cabo para un producto real.

Esta diferenciación nace de las limitaciones y características que separan un modelo realizado por simulación y un sistema real. Algunos factores que afectan a los sistemas reales y que deben ser tenidos en cuenta para la elección de los materiales y los métodos son los siguientes: Ambientales (Iluminación, humedad, viento), Físicos (Desgaste de los componentes, manchas en las lentes o recubrimientos, vibraciones provocadas por la conducción), Agentes externos (Animales, otros vehículos, señales en mal estado), entre otros.

Es por ello que, pese a que algunos elementos se intentan reproducir en la simulación mediante modelos (Por ejemplo, distintos gradientes de luz), es necesario tener presentes las diferencias a la hora de elegir y establecer los materiales y los métodos.

5.1. Materiales para el proyecto

5.1.1. Entorno de desarrollo Unity

La herramienta Unity, muy utilizada y conocida en el mundo de los videojuegos, es un entorno de desarrollo y motor gráfico multiplataforma 2D y 3D en el cuál los usuarios pueden crear, animar y renderizar todo tipo de modelos para diversas aplicaciones.

Pese a que su fama proviene del desarrollo de videojuegos en los últimos años ha cobrado relevancia en otros sectores debido a su potencia y su comunidad que cada día crea y distribuye nuevas mejoras, accesorios, modelos y código. Uno de ellos es el de la ingeniería, concretamente en la simulación, ya que con esta herramienta es posible cargar modelos elaborados con cualquier programa de diseño 3D y realizar diferentes pruebas y simulaciones rápidamente.

En este proyecto Unity nos servirá para simular un entorno conformado por distintos elementos que tratarán de recrear al ambiente real en el que se aplicaría un sistema de asistencia a la conducción.

5.1.1.1. Escenario

Para la simulación del escenario se ha elegido recrear un paisaje natural con diferentes elementos potencialmente obstaculizadores como árboles de diferente clase, rocas, luces naturales, etc.

Dentro de esta categoría también se incluiría la carretera escogida para la simulación. Se trata de una vía de doble sentido con un ancho de carril de aproximadamente 3,50 metros.

Este elemento tiene una especial relevancia ya que la posición que toma un vehículo dentro de un mismo carril afecta de forma inmediata a la visión que tiene la cámara de obtención de imágenes sobre las señales de tráfico.

5.1.1.2. Señales de tráfico

Las señales de tráfico escogidas pretenden ser una muestra representativa de los diferentes elementos que se pueden encontrar en la señalización real.

Las señales que se simularán serán las siguientes:

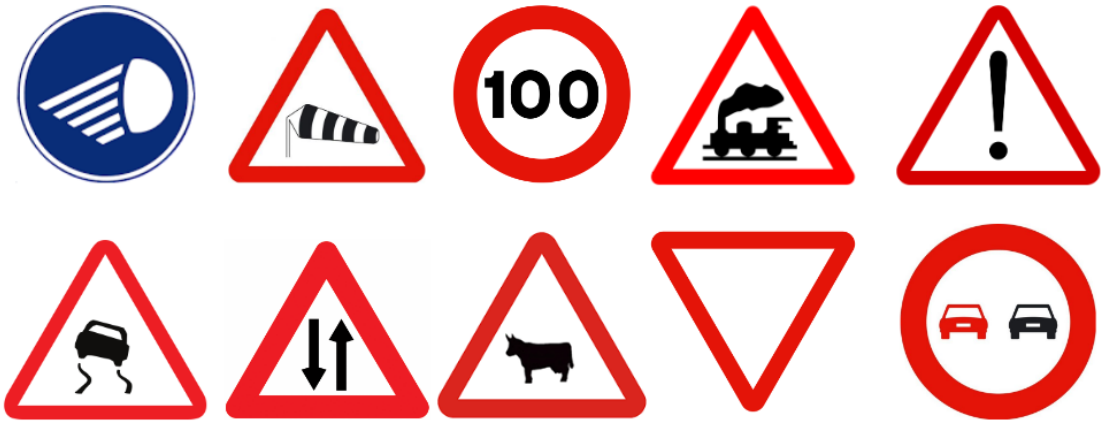

Figura 11 Señales elegidas

Como se puede ver se han elegido señales que difieran en varios elementos como el color, el tamaño, la forma y los símbolos. Esta decisión tiene dos motivaciones, por un lado, comprobar que el programa es capaz de identificar señalizaciones complejas y por otro, que logre discernir señales que representan avisos distintos pero que comparten varios elementos visuales comunes.

5.1.1.3. Vehículo

Para el vehículo se ha tomado un modelo 3D de un automóvil tipo SUV dado que cuenta con una altura cercana tanto a los coches urbanos como a los de altura superior, como los todoterrenos. Pese a la simplificación realizada sobre el modelo, es posible obtener de él alturas de referencia, distancias y medidas que sí que son muy similares a las que se obtendrían de un vehículo real.

La reducción de detalles no necesarios en el modelo 3D nos permite una simulación más ligera que requiere menos esfuerzo de computación ya que se prescinde de elementos que no son relevantes para las imágenes captadas. Esto supone que se puedan hacer más iteraciones en un menor tiempo y se puedan realizar simulaciones con mayor cantidad de fotogramas por segundo.

5.1.1.4. Visor

Para la captación de las imágenes se ha modelado una cámara dentro de la propia simulación. Este dispositivo toma imágenes con una resolución de 64x64 píxeles centradas sobre una zona reducida, las cuales serán almacenadas en diferentes directorios dependiendo de la señal de tráfico correspondiente.

Existen dos motivos que han llevado a la elección de una resolución tan pequeña para el dispositivo simulado. En primer lugar, una cuestión de tiempo de procesado, cada imagen en formato *png* tomada tiene 4096 píxeles y 3 canales de color. Para el entrenamiento del programa de identificación de señales se requieren grandes cantidades de imágenes las cuales implican un esfuerzo de computación elevado que se incrementa exponencialmente cuanto mayor es la resolución. En segundo lugar, una imagen obtenida mediante simulación no contempla todo el ruido que sí que puede

existir en una foto tomada por un visor real ni el reducido tamaño que puede tener una señal dentro de una imagen de mayor resolución, es por ello que se ha optado por disminuir la resolución y aumentar el foco del visor.

5.1.2.Programa de diseño y renderizado 3D Blender

Blender es una herramienta de diseño y renderizado 3D de software libre. Con esta herramienta se han podido editar y ajustar algunos modelos de señales de tráfico que no cumplían las proporciones o medidas reales.

5.1.3.Editor de código Pycharm

Pycharm es un entorno de desarrollo integrado dedicado al lenguaje de programación Python. Esta herramienta ha sido elegida por la rápida instalación y gestión de librerías que ofrece además de permitir al usuario ejecutar el programa dentro del entorno.

5.1.4. Bibliotecas Python

5.1.4.1. TensorFlow

TensorFlow es una de las bibliotecas de código abierto más relevantes en lo que a aprendizaje automático se refiere. Desarrollada y mantenida por Google, esta herramienta facilita la construcción y entrenamiento de redes neuronales.

Estas redes neuronales están especializadas en la detección e identificación de patrones y correlaciones, con un proceso de computación que es análogo al aprendizaje y razonamiento utilizado por los humanos. TensorFlow añade una serie de algoritmos y modelos que vuelven la tarea más sencilla.

Esta biblioteca será fundamental para la creación y entrenamiento del modelo encargado de identificar las señales de tráfico.

Para un mejor entendimiento de la biblioteca y el papel que desarrolla en la creación de inteligencias artificiales visitar el siguiente enlace:

https://www.tensorflow.org/about

5.1.4.2. Keras

Esta biblioteca también es utilizada en los proyectos de desarrollo de redes neuronales, al igual que la anterior. De hecho, Keras es capaz de ejecutarse sobre TensorFlow.

Esta herramienta, diseñada y especializada en la experimentación eficiente de redes de aprendizaje profundo, vuelve más sencillo la programación del modelo al aportar código más intuitivo y fácil de interpretar.

Los métodos que proporciona esta biblioteca para este proyecto son los siguientes:

Métodos	**Función**
Conv2D()	Crea una capa convolucional con el número y tamaño de filtros deseados.
Dense()	Crea una capa de cálculo. Esta capa toma los valores de salida de la capa anterior, realiza una serie de operaciones y entrega el resultado a la siguiente capa.
Dropout()	Crea una capa de tipo *dropout*. Esta capa desactiva de forma aleatoria un porcentaje fijo de neuronas en cada iteración. Esto se utiliza para evitar el sobreentrenamiento del modelo.
fit()	Entrena el algoritmo del modelo con un conjunto de datos de entrenamiento.
fit_generator()	Igual que el anterior. Se utiliza cuando se tiene un mayor conjunto de datos y no se necesita hacer un aumento.
Flatten()	Crea una capa que convierte las matrices multidimensionales de entradas en una matriz unidimensional.
flow()	Carga las imágenes generadas por el proceso de aumentación en memoria.
ImageDataGenerator()	Crea una función que genera nuevas imágenes alteradas a partir de un conjunto existente. En este proyecto se utilizará para lograr una cantidad mayor de imágenes distintas. Se pueden modificar diversos

	parámetros para la creación de nuevas imágenes: desplazamientos horizontales y verticales, rotaciónes, aumentos o decrementos del acercamiento, etc. A este proceso se le conoce como aumentación de imágenes.
MaxPooling2D()	Crea una capa de tipo *pooling* que se utiliza para reducir las dimeniones de un conjunto de datos manteniendo las carácteristicas más relevantes.
Sequential()	Crea un modelo de tipo secuencial. Este tipo de modelo recibe un input y predice un output después de que la información transcurra entre sus capas intermedias. El resultado de cada capa intermedia es la entrada de la siguiente.

Tabla 1 Métodos de la biblioteca Keras

Más información sobre la biblioteca en el siguiente enlace:

https://keras.io/examples/vision/image_classification_from_scratch/

5.1.4.3. Sklearn

Esta librería contiene una serie de rutinas para realizar análisis predictivo. Incluye clasificadores, algoritmos de agrupamiento, etc.

Los métodos que proporciona esta biblioteca para este proyecto son los siguientes:

Métodos	**Función**
test_train_split()	Divide un conjunto de datos en subconjuntos destinados a entrenar y a testear el modelo.

Tabla 2 Métodos de la biblioteca Sklearn

5.1.4.4. Pickle

Esta biblioteca se utiliza para serializar o deserializar una estructura de objetos Python. O lo que es lo mismo, es el proceso por el cual se convierte un objeto programado en un flujo de datos de tipo *byte.* Esto permite poder almacenar objetos en archivos.

Con la ayuda de esta biblioteca será posible almacenar el modelo en un fichero para más tarde poder ejecutarlo.

Los métodos que proporciona esta biblioteca para este proyecto son los siguientes:

Métodos	**Función**
dump(obj, file)	Toma el objeto de la variable *obj* y lo almacena en la variable de tipo archivo *file.* El archivo abierto corresponidente a la variable *file* deberá estar configurado en *"wb"* o escritura en bytes.
load(file)	Lee un objeto almacenado en el archivo *file*.

Tabla 3 Métodos de la biblioteca Pickle

Usos y ejemplos de la librería en el siguiente enlace:

https://wiki.python.org/moin/UsingPickle

5.1.4.5. OpenCV

OpenCV es una biblioteca de visión artificial y código libre desarrollada por la compañía Intel.

Esta biblioteca aporta multitud de funciones y métodos útiles para el reconocimiento facial, robótica, reconocimiento gestual, identificación de objetos, etc.

Los métodos que proporciona esta biblioteca para este proyecto son los siguientes:

Métodos	Función
cvtColor(img, color)	Toma la imagen almacenada en la variable *img* y le asigna un nuevo espacio de color almacenado en la variable *color*. En este proyecto servirá para convertir una imagen a escala de grises.
equalizeHist(img)	Toma la imagen almacenada en la variable *img* y la somete a un proceso de equalización. De esta forma, la imagen resultante tendrá un mayor contraste y por lo tanto será más sencillo identificar patrones.
imread(path)	Carga la imagen que se encuentre en la ruta *path*.
imshow(img)	Muestra una imagen almacenada en la variable *img*.
resize(obj, dim)	Modifica las dimensiones de una imagen. Toma el objeto *obj* y lo redimensiona con los valores de la variable *dim*.
set(param)	Establece los parámetros de la figura en la que se mostrará el archivo. Algunos de los parámtros que son modificables son: altura, ancho brillo, etc.
putText(img, text)	Añade el texto almacenado en la variable *text* a la imagen *img*. Este método se utilizará para mostrar la clase que predice el modelo en cada instante durante la experimentación.
VideoCapture(file)	Abre el archivo de tipo video almacenado en la variable *file*. En caso de que el argumento sea un número entero, el método accederá a la cámara del sistema.
waitKey(key)	La ventana abierta permanece a la espera de que se pulse la tecla *key.*

Tabla 4 Métodos de la biblioteca OpenCV

Más información sobre todas las herramientas que ofrece la librería en el siguiente enlace:

https://docs.opencv.org/4.x/d7/da8/tutorial_table_of_content_imgproc.html

5.1.4.6. NumPy

La biblioteca NumPy es la que da soporte al programa para poder crear grandes vectores y matrices multidimensionales. Además, también cuenta con diversas funciones matemáticas para operar con ellas.

Al igual que las anteriores, se trata de una biblioteca de código abierto que resulta imprescindible para poder trabajar con redes neuronales.

Esta herramienta es la encargada de almacenar las distintas imágenes en matrices para facilitar al modelo trabajar con los datos.

Los métodos que proporciona esta biblioteca para este proyecto son los siguientes:

Métodos	Función
amax(array)	Devuelve el valor máximo dentro de una variable *array* de tipo matriz. En este proyecto se utilizará para encontrar la predicción con mayor probabilidad ofrecida por el modelo.
append(obj)	Añade un nuevo objeto almacenado en la variable *obj* a una matriz.
array(list)	Convierte un objeto Python de tipo lista en una matriz Numpy.
asarray(obj)	Convierte un objeto en una matriz Numpy.

Tabla 5 Métodos de la biblioteca NumPy

Para mayor información sobre las distintas herramientas y funcionas que ofrece visitar en enlace:

https://numpy.org/doc/stable/reference/generated/numpy.info.html

5.1.4.7. Pandas

Pandas es una biblioteca que extiende algunas de las funciones de NumPy para la realización de análisis y manipulación de datos.

Los métodos que proporciona esta biblioteca para este proyecto son los siguientes:

Métodos	Función
read_csv(file)	Abre el archivo en formato *csv* almacenado en la variable *file* para su lectura y/o escritura.

Tabla 6 Métodos de la biblioteca Pandas

Más información acerca de la librería en el siguiente enlace:

https://pandas.pydata.org/docs/user_guide/index.html

5.1.4.8. Matplotlib

Esta biblioteca la utilizaremos para la graficar los datos y resultados obtenidos durante el entrenamiento del modelo.

La herramienta será imprescindible para el análisis de los modelos y sus respectivos desempeños, en el apartado de conclusiones.

El objetivo principal será valorar la recopilación de datos aportados para cada una de las señales de tráfico y comparar los diferentes valores de pérdida y precisión de los modelos en relación al número de iteraciones completa (Epoch) que se realiza al conjunto de las imágenes de entrenamiento.

Los métodos que proporciona esta biblioteca para este proyecto son los siguientes:

Métodos	Función
axis()	Establece los límites de los ejes de coordenadas.
bar()	Crea una gráfica de barras para mostrar los datos.

close()	Cierra la figura que esté activa. Es útil para ahorrar memoria al programa.
figure(num, figsize)	Crea una figura con el identificador de la variable *num* y las dimensiones *figsize*. Si ya existe una figura con ese identificador la activa de nuevo.
legend()	Añade una leyenda en el gráfico creado.
show()	Muestra las figuras creadas.
subplots(rows, columns)	Divide la figura en un diversos gráficos.
title(text)	Establece el título del gráfico almacenado en la variable *text*.
xlabel(text), ylabel(text)	Establece la cadena de carácteres almacenada en la variable *text* como nombre de los ejes x o y.

Tabla 7 Métodos de la biblioteca Matplotlib

Para un mejor entendimiento de todas las herramientas que ofrece Matplotlib visitar el enlace:

https://matplotlib.org/stable/tutorials/index.html

5.1.5.Sony Vegas Pro

Por último, para la comprobación del modelo ya entrenado, se utilizará un video tomado por el visor simulado. Este video se conformará con un conjunto de todos los fotogramas capturados a lo largo del recorrido de un circuito diferente al utilizado para los datos de entrenamiento.

Para renderizar el video se utilizará Sony Vegas Pro ya que permite tomar una secuencia de imágenes y convertirlo en un vídeo con la resolución y formato deseados.

Esta herramienta también será útil para cambiar la velocidad de reproducción de las imágenes sin tener que cambiar la velocidad de los elementos en el entorno de simulación. Esto puede ser necesario en las ocasiones en las que el reproductor de vídeo no sea capaz de reproducir con fluidez a la velocidad original.

5.2. Métodos para el proyecto

Para la elaboración del proyecto se ha seguido el siguiente mapa de procedimientos:

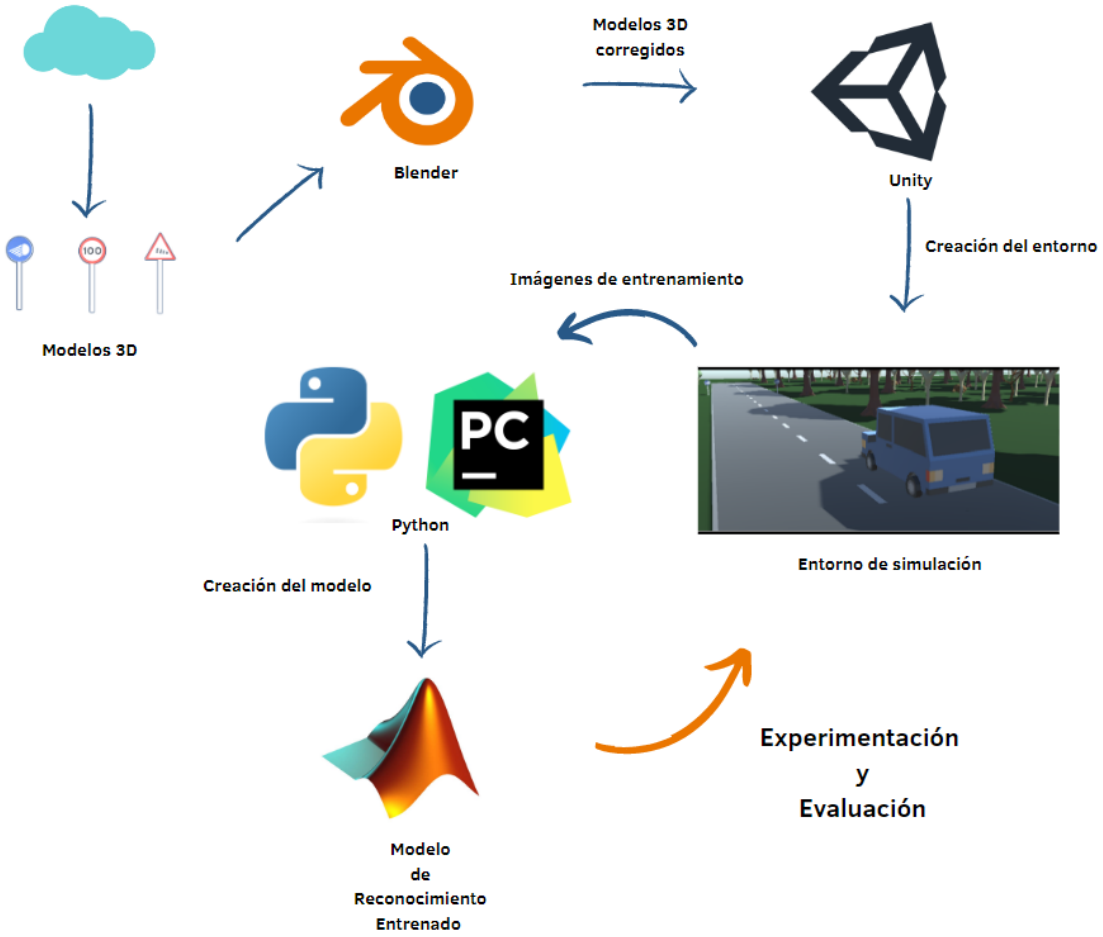

Figura 12 Desarrollo del proyecto simulado

5.2.1.Diseño de la simulación

El primer paso del proyecto será el diseño de un entorno de simulación que pueda servir de representación de un ambiente real. Como ya se ha mencionado con anterioridad se ha optado por un paisaje natural con una carretera de doble sentido y un vehículo SUV.

Figura 13 Escenario simulado

El proceso de diseño deberá incluir la creación del escenario que mantenga unas proporciones y dimensiones lo más similares posibles a los de la realidad, la programación del entorno para la obtención de las imágenes en movimiento y la incorporación de un visor que pueda tomar capturas de múltiples fotogramas consecutivos de una forma rápida y sencilla.

5.2.1.1. Creación del escenario

Para la creación del escenario se han tomado modelos 3D de libre acceso elaborados por la comunidad. Este paso ahorrará mucho tiempo ya que las medidas de las señales y las carreteras son valores estandarizados y normativos, por lo tanto, son comunes en muchos de los proyectos de simulación automovilística.

Figura 14 Señales de tráfico prefabricadas

Por otro lado, para los elementos obstaculizadores como vegetación, rocas y luces no se ha buscado tanto la reproducción de modelos reales de vegetación sino la variedad de tonalidades y formas. Esta variabilidad aportará robustez al modelo ya que el programa se habrá entrenado con imágenes más dispares las unas de las otras.

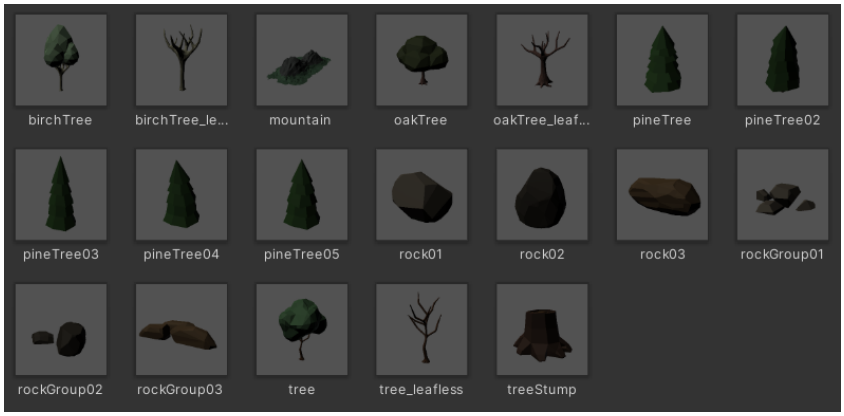

Figura 15 Elementos naturales prefabricados

Además de utilizar distintos elementos, se transformarán añadiendo rotación y escala para lograr más variedad.

5.2.1.2. Programación

Dentro del entorno es necesario controlar distintos aspectos de la simulación. Para ello se utilizará el editor de texto Microsoft Visual Studio y se programará en el lenguaje C#, un lenguaje de alto nivel.

En primer lugar, se programará el movimiento de los elementos para crear la sensación de conducción. Aquí existen dos alternativas principalmente, manipular el vehículo para que se desplace a lo largo del circuito o manipular el entorno para que se mueva alrededor del automóvil.

En este proyecto se ha optado por la segunda opción ya que proporciona algunos beneficios interesantes para la aplicación que se va a llevar a cabo. Si el vehículo, junto con el visor, se encuentran parados y el escenario es el que se desplaza se reduce el riesgo de fallo por parte de la cámara de captación de imágenes ya que pasa a ser un objeto estático. Este elemento es mucho más susceptible a errores de lo que lo son los distintos componentes del entorno.

Además, para la simulación se ha construido una cinemática constante, es decir, los diferentes elementos del escenario reaparecen en la escena una vez pasan de largo el vehículo estacionario. Esto reduce el esfuerzo de computación ya que no se generan nuevas instancias de elementos prediseñados, sino que se reubican los ya existentes.

Este enfoque sería mucho más complejo de realizar si se decidiera manipular el vehículo ya que cuando se reubicara, se observaría una falta de continuidad por el visor.

La velocidad de desplazamiento será común entre todos los elementos del entorno de simulación. Para ello se tomará una variable pública de un objeto de referencia y se utilizará en los códigos de los distintos elementos.

Para esta simulación se ha tomado de objeto de referencia uno de los segmentos de carretera. Es por eso que en los códigos de los distintos elementos del entorno se utilizará este objeto para tomar la velocidad.

El primer código será el de los segmentos de carretera:

```
Script de Unity (1 referencia de recurso) | 6 referencias
public class ControlCarretera : MonoBehaviour
{
    public float floorVel = 7.0f;
    float iniPos;
    public float roadSize;

    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    void Start()
    {
        iniPos = transform.position.x;

    }

    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void Update()
    {
        roadSize = GetComponent<Collider>().bounds.size.z;
        transform.Translate(new Vector3(-1,0,0) * Time.deltaTime * floorVel);

        if (transform.position.z <= iniPos -0.7 * GetComponent<Collider>().bounds.size.x)
        {
            transform.Translate(20*GetComponent<Collider>().bounds.size.z, 0, 0);
            iniPos = transform.position.x;
        }
    }
}
```

Código 1 Programación de la carretera

En este programa se declaran las siguientes variables:

Variables	Uso
floorVel	Variable que recoge la velocidad a la que se desplazan los segmentos de carretera. Se declara con el modificador de acceso *public* para que pueda ser accedida desde otros objetos.

iniPos	Almacena la posición inicial de la carretera. Este valor se toma de la propiedad *position.x* dentro del componente *transform* de la carretera. Se actualiza antes de comenzar la simulación y cada vez que se reubica el objeto.
roadSize	Esta variable almacena el valor de la anchura de la carretera. Se toma de la propiedad *bounds.side.z* dentro del componente *Collider*.

Tabla 8 Variables del código de la carretera

Este programa realiza dos funciones, por un lado, desplaza el objeto carretera en cada instante para darle movimiento, eso lo hace con el siguiente comando:

```
transform.Translate(new Vector3(-1,0,0) * Time.deltaTime * floorVel);
```

Código 2 Comando para el desplazamiento de la carretera

Esta línea de código aplica una traslación al objeto tomando tres parámetros.
Por un lado, un vector unitario en la dirección de movimiento (-1,0,0).
En segundo lugar, un parámetro *Time.detaTime* que tiene un valor de 1/fotogramas por segundo. Esto sirve para que la velocidad sea independiente de la cantidad de fotogramas por segundo del dispositivo en el que se realiza la simulación.
Por último, se multiplica el valor de velocidad declarado con anterioridad y que se almacena en la variable *floorVel*.
A continuación, el código del entorno:

```
Script de Unity (6 referencias de recurso) | 0 referencias
public class ControlPaisaje : MonoBehaviour
{
    // Start is called before the first frame update
    private float velocity;
    private ControlCarretera road_script;
    public GameObject road;
    private float finPos = -20;
    Mensaje de Unity | 0 referencias
    void Start()
    {
        road_script = road.GetComponent<ControlCarretera>();

    }

    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void Update()
    {
        velocity = road_script.floorVel;
        transform.Translate(new Vector3(0, 0, -1) * Time.deltaTime * velocity);
        if (transform.position.z <= finPos)
        {
            transform.Translate(0, 0, 6 * 25);

        }
    }
}
```

Código 3 Programación del entorno

Las variables utilizadas son las siguientes:

Variables	Uso
velocity	Variable que tomará la velocidad del objeto carretera para utilizarse de forma local en este código.
road_script	Almacena un componente de tipo código. Se utilizará para almacenar el código del objeto carretera. Una vez se tiene el código en esta variable se puede extraer el valor de la velocidad

	de la carretera *floorVel* para asignarlo a la variable de velocidad local *velocity*.
road	Esta variable es de tipo objeto. En este programa es la encargada de almacenar el objeto carretera.
finPos	Esta variable sirve para delimitar la posición a la que puede llegar el escenario antes de reubicarse.

Tabla 9 Variables del código del entorno

La funcionalidad de este código es la misma que el de la carretera. El objetivo es crear un entorno que se desplace a velocidad constante y vuelva a aparecer al inicio del recorrido una vez traspasado el vehículo.

Por último, el código de las señales de tráfico:

```
Script de Unity (11 referencias de recurso) | 0 referencias
public class ControlSeñales : MonoBehaviour
{
    // Start is called before the first frame update
    private float velocity;
    private ControlCarretera road_script;
    public GameObject road;
    Mensaje de Unity | 0 referencias
    void Start()
    {
        road_script = road.GetComponent<ControlCarretera>();
    }

    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void Update()
    {
        velocity = road_script.floorVel;
        if (transform.position.z <= -15.0f)
        {
            velocity = 0;
        }
        transform.Translate(new Vector3(0, 0, -1) * Time.deltaTime * velocity);
    }
}
```

Código 4 Programación de las señales de tráfico

Este programa es idéntico al anterior, la única diferencia es que las señales de tráfico se detienen al llegar a un punto en vez de reaparecer al inicio. Esto es así porque en cada circuito de entrenamiento el recorrido se graba hasta que se capturan todas las señales. Una vez las imágenes de cada señal son almacenadas se hace alguna variación sobre el entorno y se vuelve a realizar la captura de imágenes.

Para la incorporación del visor se deberá añadir un objeto de tipo cámara, esta cámara se colocará a la altura de la luna del vehículo simulando la posición que suelen adoptar este tipo de dispositivos.

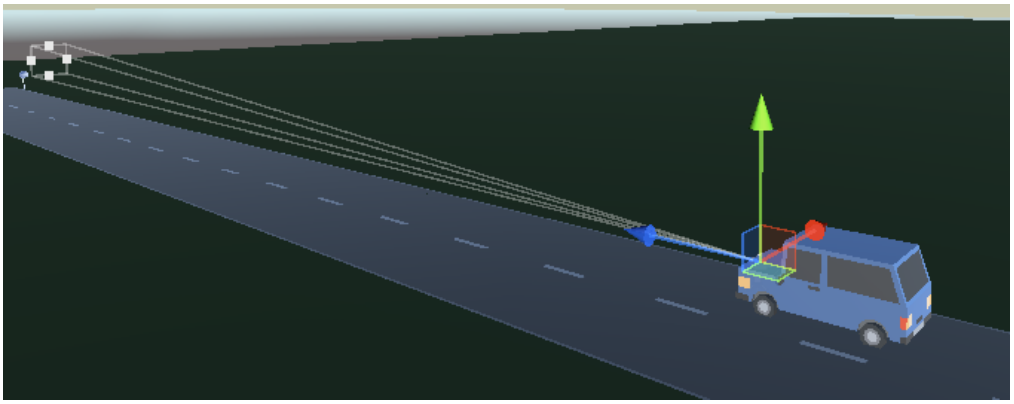

Figura 16 Visor en la escena de simulación

Figura 17 Punto de vista del visor

La configuración de dicha cámara es la siguiente:

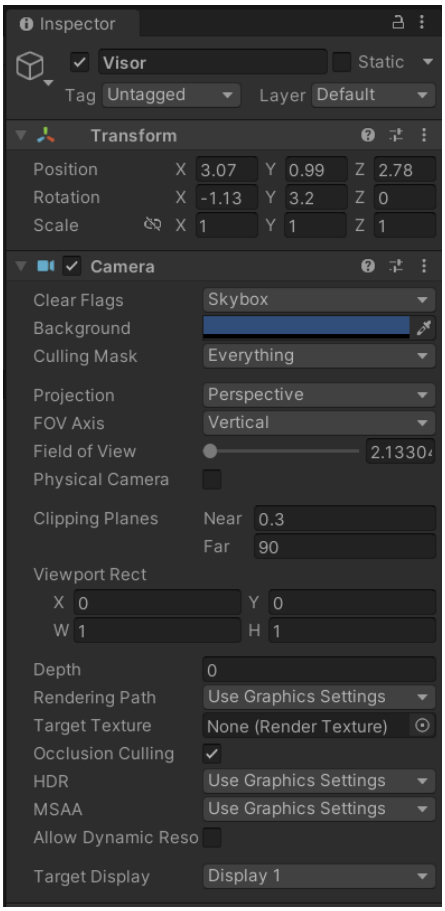

Figura 18 Configuración del visor

Con estos parámetros se logran imágenes de baja resolución pero que captan toda la superficie de las señales. Gracias a esto se consigue representar el proceso que se llevaría a cabo después de hacer un aumentado de una imagen de mayor resolución.

Para la captura de imágenes por parte del visor se empleará el siguiente código:

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class Screenshots : MonoBehaviour
{
    // Start is called before the first frame update
    public int _shotindex;
    public int _classindex;
    private bool record = false;

    Mensaje de Unity | 0 referencias
    private void Start()
    {

    }
    Mensaje de Unity | 0 referencias
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.F))
        {
            record = !record;
        }
        if(record)
        {
            ScreenCapture.CaptureScreenshot(@$"Capturas\{_classindex}\" + $"{new string('0', (2 - _classindex.ToString().Length))}{_classindex}" +
                $"{new string('0', (3 - _shotindex.ToString().Length))}{_shotindex}.png",1);
            _shotindex++;
        }
        if (Input.GetKeyDown(KeyCode.C))
        {
            //_shotindex = 0;
            _classindex++;
        }
    }
}
```

Código 5 Programación del visor

Las variables utilizadas en este programa son las siguientes:

Variables	Uso
_shotindex	Variable que almacena el índice de cada una de las imágenes dentro de una misma clase. Se incrementa después de realizar una captura para poder tomar múltiples imágenes consecutivas.
_classindex	Variable que almacena el índice de cada una de las clases, es decir, cada una de las señales de tráfico. Por comodidad se ha programado que la tecla "*C"* aumente en una unidad el valor de esta variable para poder tomar capturas de varias clases sin detener la simulación.
record	Esta variable de tipo boleana es la que se utiliza para decidir si los fotogramas se capturan.

El programa realiza dos tareas, por un lado, hace una captura en cada fotograma en caso de que se presione la tecla "*F*", esta función se mantiene hasta que se vuelva a accionar la tecla.

Por otro lado, las imágenes se codifican con dos valores numéricos, en primer lugar, se añade la clase, es decir la señal, y a continuación se añade el índice, el cual se incrementa con cada fotograma.

Para mantener el formato se han utilizado las siguientes líneas de código:

```
{new string('0', (2 - _classindex.ToString().Length))}
```

Código 6 Corrección de formato de clase

```
{new string('0', (3 - _shotindex.ToString().Length))}
```

Código 7 Corrección de formato de captura

Esta permite que se mantenga la cantidad de cifras en el nombre de cada imagen. En el caso de las clases dos cifras y en el de las imágenes tres.

Los fotogramas se almacenan en directorios específicos dependiendo de la clase:

```
@$"Capturas\{_classindex}\"
```

Código 8 Almacenamiento de capturas por clases

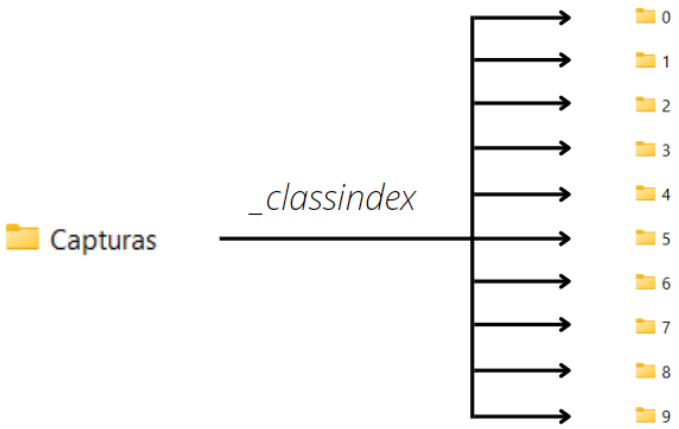

Figura 19 Clasificación en directorios

El código anterior se vinculará a un nuevo objeto vacío llamado *CamHandler*. Desde este objeto se podrá elegir la clase, es decir la señal de tráfico, y el índice de la fotografía que se va a hacer:

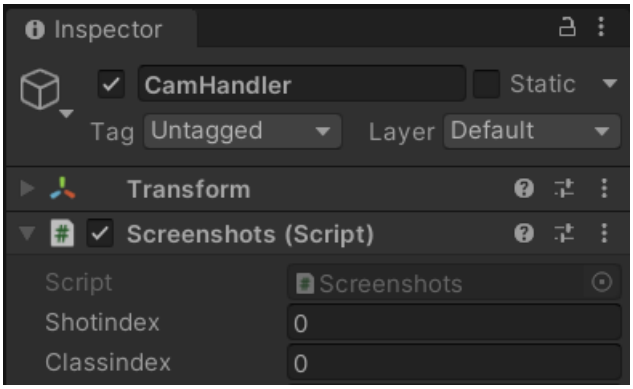

Figura 20 Interfaz con el controlador del visor

Una vez ya se han recopilado las suficientes imágenes se puede pasar a diseñar el modelo. Para este proyecto se han tomado unas 600 imágenes de cada señal de tráfico.

5.2.2.Programación del modelo

La programación necesaria para el diseño y evaluación del modelo será dividida en dos archivos distintos.

Por un lado, se creará un archivo que recoja las imágenes, cree el modelo, y lo entrene tomando en cuenta unos parámetros de diseño. Este proceso es el más complejo y el que requiere un mayor tiempo tanto de diseño como de computación

Un segundo programa será utilizado para comprobar el modelo sobre el video obtenido de la simulación.

5.2.2.1. Creación

El primer paso será importar las bibliotecas necesarias (Ver apartado 5.1.4, Bibliotecas Python):

```
import cv2
import pickle
import random
import numpy as np
import os
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

Figura 21 Importación de bibliotecas

A continuación, definiremos algunos de los parámetros que se utilizarán a lo largo del programa, de esta manera será más fácil realizar pequeñas modificaciones generales sin la necesidad de modificar todos los lugares en los que aparece:

La explicación de los parámetros es la siguiente:

Parámetros	**Significado**
batchSize	Es el tamaño de cada paquete de datos. Para este proyecto se ha utilizado 50 en el modelo final. Eso quiere decir que cada vez que se hace una entrega de datos al modelo se le envian 50 imágenes.
epochs	Es la cantidad de veces que se le entrega el conjunto de todos los datos de entrenamiento al modelo. Una cantidad reducida significa menos tiempo de entrenamiento y menos precisión. Una mayor cantidad requerirá un mayor tiempo de computación a cambio de una mejor precisión. Un número demasiado elevado puede ocasionar sobreentrenamiento, esto hace que el modelo solo sea capaz de identificar correctamente las imágenes usadas en el entrenamiento pero no sea efectivo con nuevas imágenes.
stepsPerEpoch	En cuantos gradientes o *steps* se va a entregar la cantidad de imágenes de entrenamiento en cada *epoch*. En cada *step* una cantidad de imágenes definida por el *batchSize* es entregada al modelo.
testRatio	Es el porcentaje de las imágenes totales que se va a destinar a testear el modelo una vez ya se encuentra entrenado y validado. Para este proyecto se ha tomado un valor del 25%.
valRatio	Es el porcentaje de las imágenes restantes, después de separar las de test, que se destinan a validar el modelo entrenado. En la fase de validación se ajustan los hyperparámetros del modelo, aspectos como el número de capas y sus dimensiones. Para este proyecto se ha tomado un porcentaje de 15%.

Tabla 10 Páramteros del código del modelo

Una vez los parámetros están ajustados para el modelo se procederá a importar las clases que el modelo deberá reconocer. Para ello, el primer paso será almacenar las imágenes en una matriz. Se utilizarán dos bucles *for* anidados para iterar por cada clase y cada fotografía:

```
for i in range(0, len(myList)):
    myPicList = os.listdir(picsPath + "/" + str(count))
    for j in myPicList:
        curImg = cv2.imread(picsPath + "/" + str(count) + "/" + j)
        images.append(curImg)
        classNo.append(count)
    print(count, end=" ")
    count += 1
print(" ")
```

Código 9 Importación de las imágenes de cada clase

El programa dará respuesta a través de la consola informando del proceso de importación:

```
Clases Totales Detectadas: 10
Importando Clases.....
```

Figura 22 Detección de clases

```
Clases Totales Detectadas: 10
Importando Clases.....
0 1 2 3
```

```
Clases Totales Detectadas: 10
Importando Clases.....
0 1 2 3 4 5
```

Figura 23 Seguimiento de las imágenes importadas

Una vez las imágenes se encuentran cargadas se procederá a su separación en tres grupos: entrenamiento, validación y test. Para ello se utiliza el comando *train_test_split* visto con anterioridad en el apartado de las bibliotecas (Ver apartado 5.1.4.3, Sklearn).

Cuando las imágenes ya se encuentran separadas es momento de tratarlas para facilitar el entrenamiento del modelo. Para este proyecto se aplicarán dos preprocesos, la conversión a escala de grises y la equalización del histograma.

La conversión a escala de grises es un proceso por el cual se toma el espacio de color de una imagen y se convierte a escala de grises:

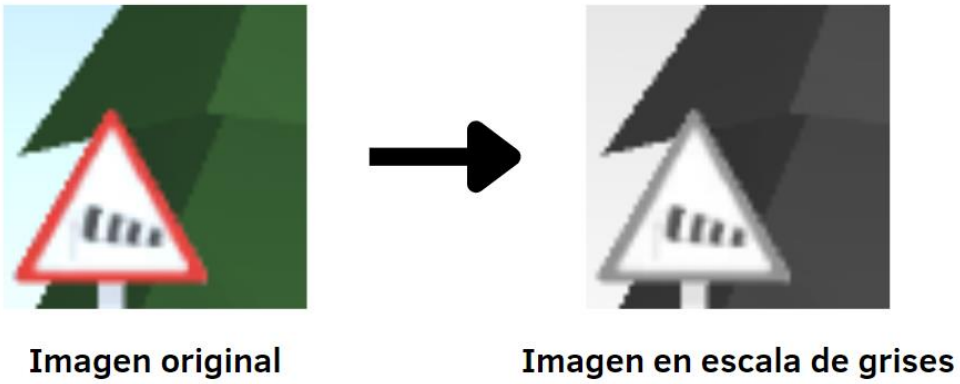

Figura 24 Conversión a escala de grises

Esto permite reducir los canales de color de la imagen y por tanto el tamaño de los datos de entrenamiento, lo cual conduce a un mejor tiempo de procesado. Además, de esta forma la red neuronal se ve obligada a reconocer patrones basados en los cambios de luminosidad y no de color. El modelo debe ser capaz de reconocer una imagen en blanco y negro de igual manera que lo haría una persona.

El siguiente proceso que se aplica es la equalización del histograma. El histograma es una representación gráfica de la cantidad de píxeles que tiene una imagen por cada uno de los valores de luminosidad:

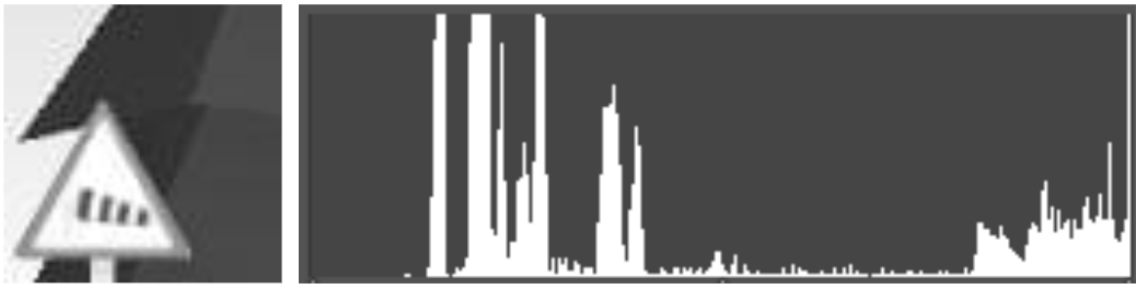

Figura 25 Imagen con su histograma

La ecualización consiste en la redistribución de la iluminación de los píxeles para ocupar un mayor rango. De esta forma, al aumentarse la distancia entre los valores de iluminosidad, resulta más sencillo reconocer patrones.

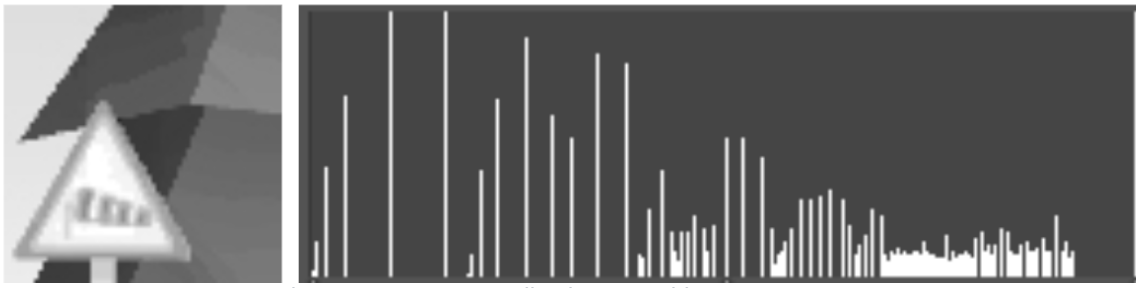

Figura 26 Imagen equalizada con su histograma

Cómo se puede observar, es posible diferenciar los distintos tonos lumínicos que había en las hojas del árbol, mientras que en la primera imagen parecía ser un único tono sólido.

Una vez las imágenes ya se encuentran preprocesadas se recurrirá a un método de aumentación. Los métodos de aumentación permiten generar nuevas imágenes a partir de un conjunto. Estas imágenes se transforman alterando en pequeña medida parámetros como el acercamiento a la imagen, la rotación, el desplazamiento horizontal y vertical, entre otros:

Figura 27 Ejemplo de aumentación de imágenes

Como resultado se logra una mayor cantidad de imágenes “nuevas” que se le entregan al modelo. Este método de obtención de imágenes nace de la premisa de que el modelo debe poder reconocer los patrones independientemente de su escala y su orientación. Para la generación de estas se utiliza el comando *ImageDataGenerator* (Ver apartado 5.1.4.2).

Una vez terminados los preprocesos y la aumentación de imágenes se podrá proceder a la creación del modelo. Para este proyecto se ha escogido el modelo LeNet.

El modelo LeNet está compuesto de dos sets de capas convolucionales, activación y *pooling* seguidas de dos capas densas totalmente conectadas.

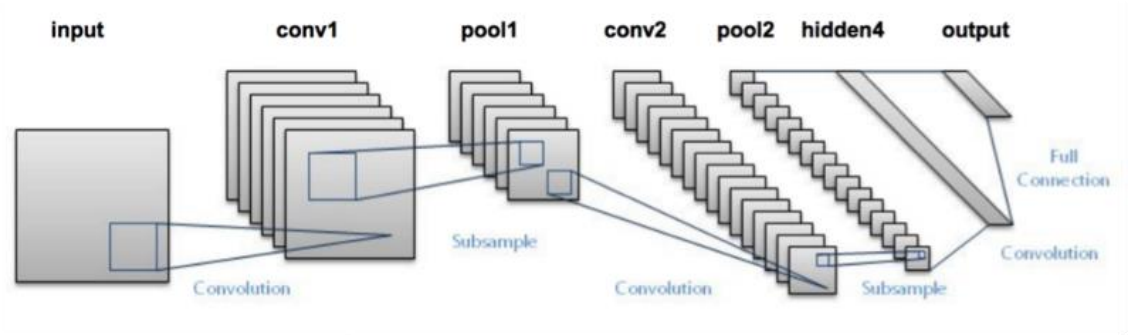

Figura 28 Modelo de red neuronal convolucional LeNet

https://pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/

Este modelo fue diseñado en sus inicios para el reconocimiento de digitos escritos a mano y reconocimiento de lenguaje. No obstante, su facilidad de construcción y su buena relación rendimiento-coste computacional, lo vuelven una gran opción a la hora de elegir un modelo de red neuronal convolucional.

A continuación, se explicará en detalle la tarea que lleva a cabo cada elemento del modelo.

En primer lugar, están las capas convolucionales, estas trabajan de forma consecutiva las unas con las otras. Comienza con la detección de rasgos generales y a medida que se avanza se configuran para identificar patrones más concretos.

El funcionamiento de las capas convolucionales se basa en la aplicación de filtros con pesos ponderativos que sirven para crear mapas de características a partir de imágenes.

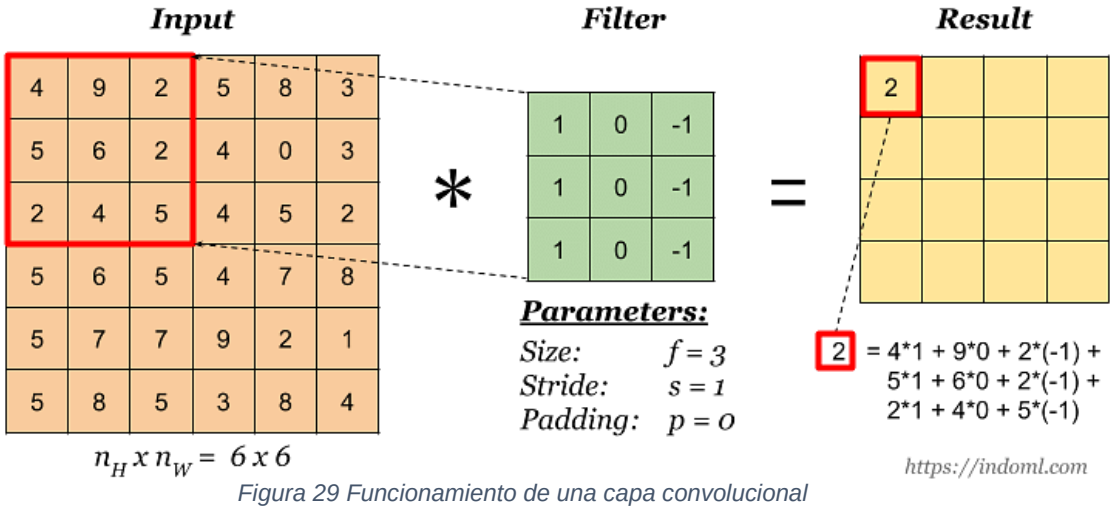

Figura 29 Funcionamiento de una capa convolucional

https://www.projectpro.io/article/introduction-to-convolutional-neural-networks-algorithm-architecture/560

Cada uno de los filtros aplicados dará lugar a un mapa de características. La aplicación de diversas capas convolucionales con sus correspondientes filtros o *kernel* lleva el modelo a reconocer bordes y gradientes de luminosidad en los primeros pasos y patrones más complejos a medida que avanza.

Junto a las capas convolucionales se encuentra la función de activación. Una función de activación tiene la tarea de limitar o condicionar la salida de una capa neuronal. Hay de distintos tipos, pero la que se aplicará en las capas convolucionales de este proyecto será la función *ReLu* (Rectifyer Linear Unit).

Esta función convierte los valores negativos en cero y los positivos los mantiene. Al ser una operación sencilla resulta rápida de implementar y procesar. Su objetivo es hacer que la siguiente capa solo reciba valores positivos, lo cual facilita el trabajo de computación.

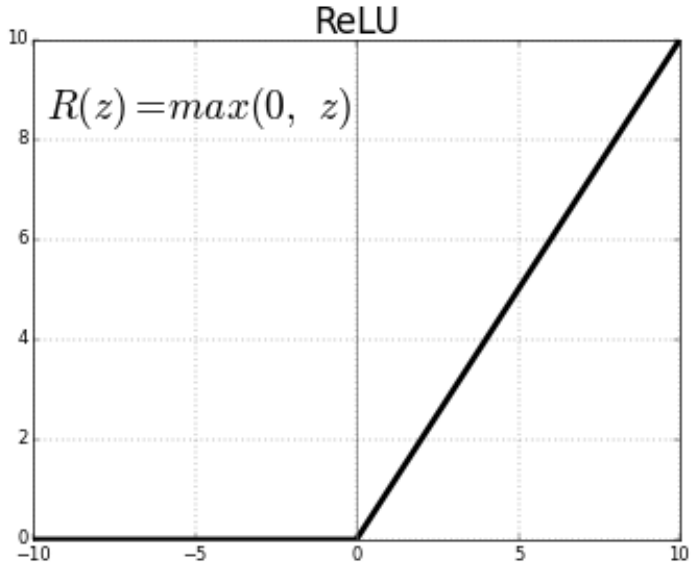

Figura 30 Función de activación ReLu

Seguidamente se encuentran las capas *pooling*. Estas cumplen una tarea sencilla pero necesaria para reducir el esfuerzo de computación. Las capas *pooling* toman el mapa de características producido por las convolucionales y reducen sus dimensiones tomando algún criterio para elegir los píxeles con la información más relevante. Una de las opciones más comunes, y la aplicada en este proyecto, es la capa *pooling* de máximos.

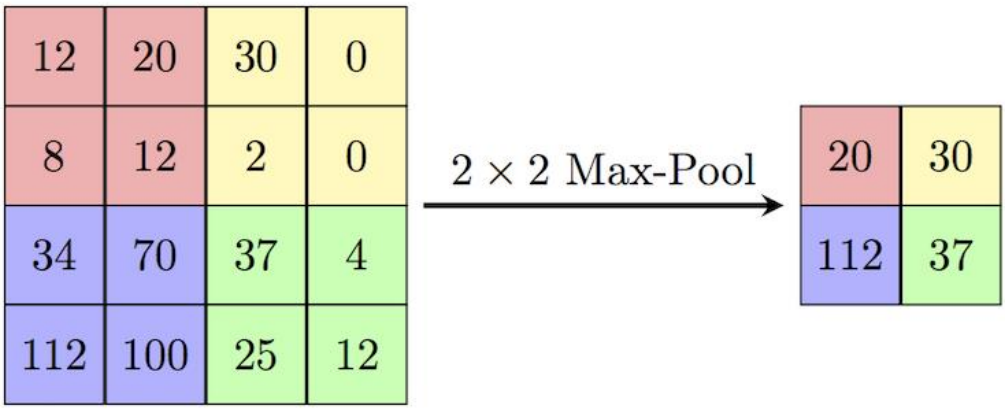

Figura 31 Funcionamiento de una capa pooling

https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

Como se puede ver en el ejemplo, la capa pooling divide la información recibida en submatrices y toma de ellas el valor más relevante. En este caso el criterio es el valor máximo. El resultado es un conjunto de valores que mantiene las características más relevantes, pero con una menor dimensión. La aplicación de una capa *pooling* conlleva una pérdida de información a cambio de un menor coste computacional.

Finalmente, se encuentran las capas densas. Estas capas son la base de toda red neuronal y se basan en la configuración de una serie de pesos ponderados asociados a los vinculos entre las diferentes capas. Las capas densas están totalmente conectadas con la capa siguiente y su salida es el resultado de aplicar un producto escalar a su entrada.

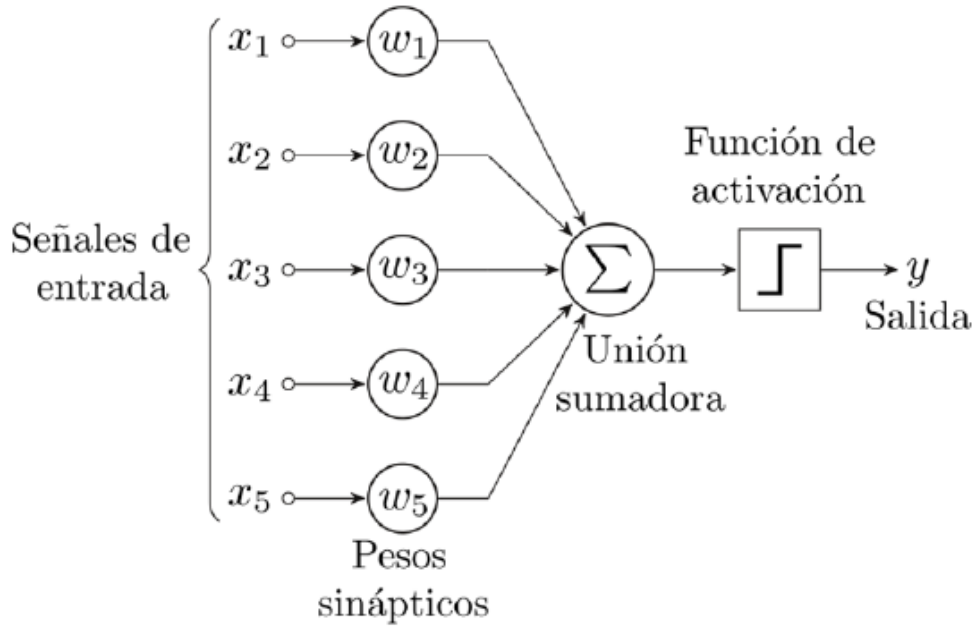

Figura 32 Funcionamiento de una capa densa

https://math.libretexts.org/Courses/Universidad_Complutense_de_Madrid/Las_matematicas_de_la_inteligencia_artificial/06%3A_Redes_Neuronales/6.01%3A_Perceptron

Una vez el modelo está creado se puede compilar y entrenar.

Durante el entrenamiento del modelo también se tendrá un seguimiento del proceso. De esta forma, se podrá saber los pasos o *steps* restantes, el *epoch* actual, el tiempo estimado, el valor actual de pérdida del modelo o *loss* y el valor actual de precisión o *accuracy*.

```
Epoch 1/3

   1/2000 [..............................] - ETA: 39:44 - loss: 2.3133 - accuracy: 0.1000
   2/2000 [..............................] - ETA: 37:22 - loss: 2.3064 - accuracy: 0.1000
31/2000 [..............................] - ETA: 36:21 - loss: 2.3017 - accuracy: 0.1194
32/2000 [..............................] - ETA: 36:28 - loss: 2.3010 - accuracy: 0.1181
33/2000 [..............................] - ETA: 36:29 - loss: 2.3003 - accuracy: 0.1200
34/2000 [..............................] - ETA: 36:34 - loss: 2.2991 - accuracy: 0.1224
35/2000 [..............................] - ETA: 36:38 - loss: 2.2969 - accuracy: 0.1240
36/2000 [..............................] - ETA: 36:50 - loss: 2.2987 - accuracy: 0.1233
```

Figura 33 Seguimiento del proceso de entrenamiento del modelo

La pérdida es una función que cuantifica la desviación entre predicciones consecutivas. Es por eso que a medida que avanza el entrenamiento este valor comienza a decaer.

Por otro lado la precisión es el porcentaje de predicciones acertadas sobre el total que realiza el modelo. Este valor debe aumentar a medida que el entrenamiento avanza.

Una vez este proceso llega a su fin se obtiene un modelo entrenado que se encuentra listo para ser probado. Para poder almacenarlo en un fichero y utilizarlo en futuros programas se hará uso del siguiente comando (Ver apartado 5.1.4.4, Pickle):

```
modelToSave = open("modelo_entrenado.p", "wb")
pickle.dump(model, modelToSave)
modelToSave.close()
cv2.waitKey(0)
```

Código 10 Almacenamiento del modelo en un fichero

5.2.2.2. Experimentación

Hay diversas formas para poner a prueba un modelo entrenado. Es posible transmitirle imágenes para recibir predicciones, conectar una cámara al ordenador y procesar imágenes del mundo real, etc. Para este proyecto se le entregará una grabación del entorno de simulación.

El primer paso será modificar el entorno simulado para que las imágenes capturadas por el visor virtual resulten novedosas para el modelo, de lo contrario solo estaría recordando imágenes del entrenamiento.

Las modificaciones que se le pueden hacer al ambiente vienen condicionadas por el rango de imágenes distintas que se tomaron para el entrenamiento. Esto significa que el modelo podrá reconocer imágenes en las que las señales sufran modificaciones para las que ya ha elaborado patrones de reconocimiento. Si en las imágenes de entrenamiento las señales estaban rotadas, inclinadas, volteadas o desplazadas y la muestra era suficientemente amplia el modelo debería de ser capaz de identificar nuevas imágenes con el mismo tipo de alteraciones.

No obstante, si el modelo no hubiera sido entrenado con imágenes de señales alejadas o distorsionadas, sería complicado obtener predicciones precisas, ya que estaría expuesto a patrones que nunca ha aprendido.

Con los fotogramas tomados del nuevo circuito se creará un vídeo. Para ello se utilizará el programa de edición de vídeo SonyVegas Pro (Ver apartado 5.1.5, Sony Vegas Pro). Se seleccionará la primera imagen y se activará la opción de “Abrir secuencia”.

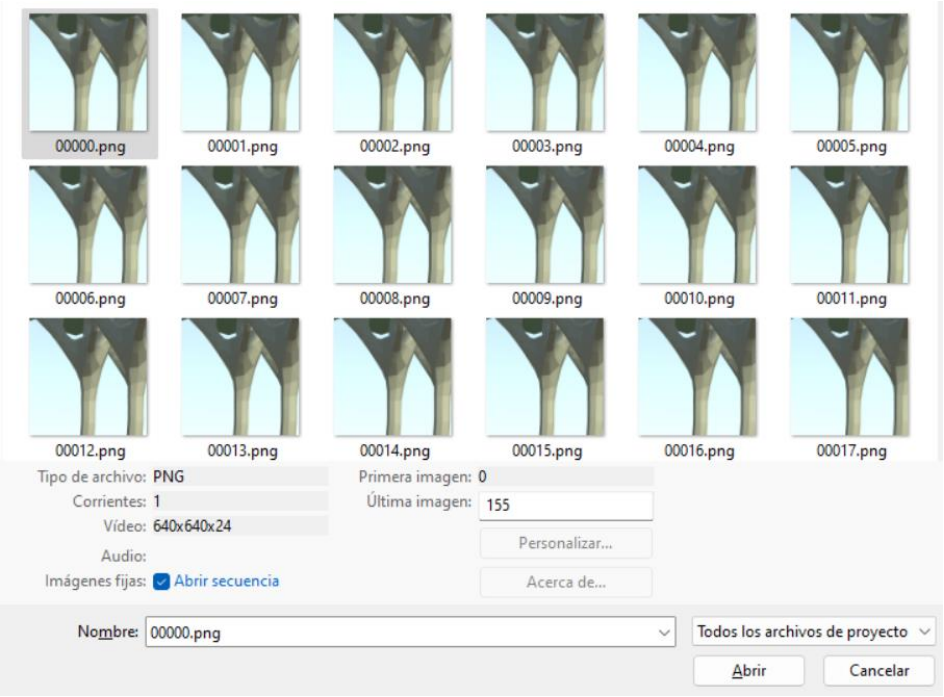

Figura 34 Importación de fotogramas para secuencia

Una vez seleccionada se importará la secuencia a la pista de vídeo. Los fotogramas separados apareceran de forma consecutiva por lo que solo será necesario renderizarlo.

Para la el código de experimentación es necesario establecer algunos parámetros de la cámara que se utilizarán más adelante para la configuración.

Parámetros	Significado
brightness	Brillo de la imagen capturada que se muestra por el reproductor.
frameHeight	Altura de la ventana del reproductor de video.
frameWidth	Ancho de la ventana del reproductor de video.

Tabla 11 Párametros de la ventana del reproductor

Además de estos parámetros también se configurará el límite de precisión o *treshold* del experimento, este es el valor a partir del cual el modelo considera que una predicción es válida.

Un límite demasiado bajo generará que el modelo ofrezca predicciones de las que no tiene mucha certeza, por el contrario, un límite demasiado alto podría hacer que el modelo no ofreciera las predicciones que sí son ciertas.

Se tomará entonces el video generado mediante el comando VideoCapture (Ver apartado 5.1.4.5, OpenCV) y se almacenará en una variable.

```
video = cv2.VideoCapture("Circuito_Prueba.mp4")
```

Código 11 Comando para abrir el video de prueba y asignarlo a una variable

A continuación, se añaden los parámtetros del reproductor que se habían especificado anteriormente.

```
video.set(3, frameWidth)
video.set(4, frameHeight)
video.set(10, brightness)
```

Código 12 Configuración del reproductor de video

El siguiente paso consistirá en listar e identificar los diferentes resultados aceptados dependiendo de las clases. Esto se realizará con una estructura de *else if*, en pyhton *elif*, puesto que en cada fotograma la predicción solo puede resultar en una única señal.

```
def obtenerClase(classNo):
    if classNo == 0:
        return 'Encender luces'
    elif classNo == 1:
        return 'Vientos fuertes laterales'
    elif classNo == 2:
        return 'Maximo 100 Km/h'
    elif classNo == 3:
        return 'Paso de tren'
    elif classNo == 4:
        return 'Atencion'
    elif classNo == 5:
        return 'Suelo resbaladizo'
    elif classNo == 6:
        return '2 Direcciones'
    elif classNo == 7:
        return 'Paso de animales'
    elif classNo == 8:
        return 'Ceda el paso'
    elif classNo == 9:
        return 'No adelantar'
```

Código 13 Identificación del nombre de la clase

Dependiendo de identificador que prediga el modelo, almacenado en la variable *classNo*, el texto que devolverá esta función variará.

Finalmente se creará un bucle continuo con una estructura *while* que mantendrá el modelo continuamente prediciendo señales. A su vez, se mostrarán dos cadenas de texto en las que se especificará la clase que se está prediciendo y la precisión con la que el modelo la identifica.

```
cv2.putText(imgOrignal, "CLASE: ", (20, 35), font, 0.75, (255, 0, 0), 2, cv2.LINE_AA)
cv2.putText(imgOrignal, "PROBABILIDAD: ", (20, 75), font, 0.75, (0, 255, 0), 2, cv2.LINE_AA)
```

Código 14 Cadenas de texto para clase y probabilidad

Con esto solo queda ejecutar el programa:

Figura 35 Predicción de ejemplo 1

Figura 36 Predicción de ejemplo 2

Figura 37 Predicción de ejemplo 3

5.3. Materiales reales

Se procederá a realizar también una descripción de cuáles habrían sido los materiales necesarios para la implementación real de este proyecto.

5.3.1. Cámara de captación de imágenes

Para la toma de datos en un proyecto real es necesario contar con una herramienta que resulte fácilmente transportable, ligera y que a su vez pueda almacenar una gran cantidad de imágenes.

Es por eso que se recurrirá a una cámara reflex ya que estas ofrecen una gran calidad de captura y diversas funcionalidades para cambiar parámetros como el enfoque, la iluminosidad, etc. Esto permite una mayor variabilidad de imágenes en capa toma.

Figura 38 Cámara fotográfica reflex

Además, estas cámaras pueden almacenar grandes cantidades de imágenes en pequeñas memorias portátiles.

Cómo última ventaja, este tipo de dispositivos cuentan con un software que almacena las imágenes con una codificación indexada. Esto permite visualizar rápidamente las imágenes que se han tomado en un día concreto en caso de que haya que descartar las imágenes de un periodo conocido.

5.3.2. Visor

El visor es la cámara que se utiliza en la aplicación concreta, esta cámara no necesita disponer de una gran memoria o ajustes de imagen ya que las capturas tomadas son enviadas de forma casi inmediata a una unidad de procesamiento que es la encargada de procesar la información con el modelo y transmitir el resultado al ordenador del vehículo para que tome realice la acción correspondiente.

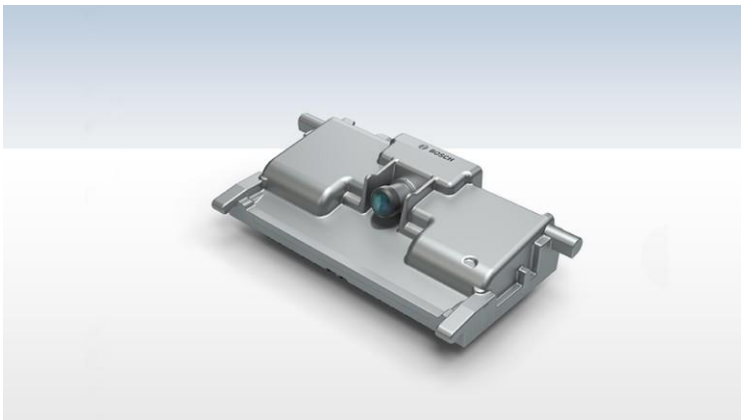

Figura 39 Cámara multipropósito de Bosch

Si se conocen las características del ambiente en el que va a trabajar el dispositivo se puede lograr un visor más eficiente y económico. Por ejemplo, si se va a emplear en un circuito cerrado que no se encuentra al aire libre, como por ejemplo un metro, no

sería necesario elegir un visor que fuera resistente al agua. De igual manera, si se conocen las distancias, tamaños y posicionamiento de los elementos, se puede optar por visores con menor tamaño, baja resolución y mejor rendimiento.

Es necesario tener en cuenta que cuanto más específico sea el visor para la aplicación menos resiliente será a la hora de resolver otros problemas que puedan aparecer en el futuro. Por lo tanto, una vez más resulta imprescindible conocer el entorno en el que va a trabajar el sistema para llegar a un equilibrio entre efectividad y flexibilidad.

5.3.3. Ordenador

Para el entrenamiento de toda red neuronal es necesario un ordenador. Este es el encargado de tomar los datos y proporcionarselos al modelo a medida que lo ajusta para obtener la mejor precisión posible y reducir su función de pérdida (Ver apartado 7.1.2.1, Creación).

A diferencia de la unidad de control que será utilizada durante la aplicación real, el ordenador debe tratar con grandes cantidades de información y realizar muchas operaciones para llegar a un modelo que sea fiable y robusto. Es por eso que los ordenadores dedicados al entrenamiento de inteligencias artificiales necesita una gran potencia de procesamiento y mucha memoria RAM.

El dimensionamiento de este ordenador depende una vez más de la aplicación para la que se quiera diseñar. Una inteligencia artificial que permita una conducción autónoma en carretera necesitará haber pasado por un entrenamiento mucho más preciso y extenso que un modelo que solo debe señalizar al conductor de un tren las señales que se aproximan.

5.3.4. Unidad de Control

La Unidad de Control es aquella encargada de procesar las imágenes obtenidas por el visor en tiempo real y comunicar al ordenador del vehículo la información obtenida. Esta unidad debe contar con diferentes dispositivos tales como puertos de comunicación, procesador, fuente de alimentación, además de elementos auxiliares como el sistema de ventilación o los soportes.

En el caso de una Unidad de Control no es necesario que el programa analice al mismo tiempo una gran cantidad de imágenes, no obstante sí que es necesario que el procesado lo haga velozmente. El tiempo de reacción es un parámtreo crítico en los sistemas de seguridad en el mundo del sector automotriz y se vuelve más relevante cuando el vehículo se encuentra a velocidades elevadas.

5.4. Métodos reales

De la misma forma que se ha expuesto para los materiales, existen particularidades que se deben considerar a la hora de llevar a cabo los métodos en un proyecto real.

5.4.1. Toma de imágenes

La toma de imágenes es el primer paso en todo proyecto que involucre la visión artificial una vez ya se ha concretado cual va a ser la tarea a desempeñar.

Cuando se toman capturas de aquellos elementos que el sistema debe ser capaz de reconocer, es necesario entender que estas se deben ajustar a la aplicación del modelo. Si por ejemplo se pretende entrenar un modelo que va a estar conectado a un visor situado en la luna del vehículo, quizá es poco interesante tomar imágenes de

señales con un plano muy picado, ya que cuando el sistema se encuentre en funcionamiento esta perspectiva de la señal no va a ser común.

De nuevo, es un compromiso entre flexibilidad del modelo y eficiencia. Para una aplicación concreta hay parámetros específicos que tienen una serie de condiciones limitantes ya conocidas de antemano.

En el caso del modelo que identifica señales de tráfico quizá no es tan útil hacer fotografías desde mucha altura porque las señales siempre van a estar a una distancia vertical aproximada. No obstante, sí que puede resultar necesario tomar imágenes a diferentes distancias y con diversa iluminación. No tendría sentido que el sistema solo pudiera identificar las señales cuando se encuentran a una distancia fija, en las horas de más luz y con un tiempo perfectamente soleado.

Figura 40 Señales con mala visibilidad

Por supuesto las condiciones medioambientales siempre afectarán a la capacidad de los sistemas de visión artificial de la misma manera que nuestra propia visión se ve limitada, pero con un entrenamiento a conciencia y una buena selección de las imágenes, se puede lograr mejorar en gran medida las prestaciones del sistema.

5.4.2. Creación y entrenamiento del modelo

La creación del modelo para una aplicación real sería muy similar a la realizada para el proyecto.

La diferencia principal sería la variabilidad de condiciones a la que está sometido un sistema de visión artificial en la realidad. Condiciones que no siempre son replicables en un sistema simulado.

Es por ello que un modelo que fuera a ser utilizado en una aplicación real, como por ejemplo un sistema de visión artificial para la asistencia a la conducción, debería cumplir unos requisitos más estrictos de aquellos elegidos para el modelo simulado.

En primer lugar, como ya se ha mencionado, la recopilación de imágenes de entrenamiento debería ser mucho mayor y en las capturas se debería buscar la máxima variedad posible dentro de los parámetros normales del entorno.

Además, para un sistema en el que se va a confiar en mayor o menor medida para una función de seguridad, sería recomendable crear diversos modelos, con construcciones y configuraciones diferentes, y ponerlos a prueba para elegir el que mejor resultado ofrezca.

Cabe esperar que el entrenamiento también sea más extenso y requiera una mayor capacidad de computación que el que se ha necesitado para el modelo aplicado a la simulación. Es recomendable, además, añadir mensajes que sirvan de monitorización durante el entrenamiento para poder analizar el desarrollo de la inteligencia artificial y reiniciar el proceso en caso de que no se esté obteniendo el resultado esperado. De esta forma es posible optimizar el tiempo empleado en esta fase del desarrollo.

5.4.3.Prueba del modelo

Para comprobar que el modelo funciona correctamente será necesario realizar pruebas en funcionamiento. En la medida de lo posible, se debería intentar realizar pruebas en un entorno controlado, donde la seguridad de los equipos y de las personas no se pusiera en riesgo en caso de un fallo del sistema.

Una vez los resultados fueran satisfactorios y la tasa de acierto estuviera por encima de un umbral prudente se podría pasar a ensayar en un entorno real, siempre bajo vigilancia y sin desempeñar funciones de demasiada seguridad.

Este proceso es iterativo con el entrenamiento. Los resultados deben ser estudiados y contrapuestos a medida que se desarrolla y se pone a prueba el modelo. Aspectos como el número de capas, su tipología, las funciones de activación y demás parámetros vistos durante la creación del modelo simulado (Ver capítulo 5.2.2.1, Creación) pueden ser modificados para conseguir un mejor resultado.

Durante esta fase, un acercamiento a las herramientas de probabilidad y ciencia de datos resulta necesaria cuando se trata de una ciencia como la inteligencia artificial. La cantidad de datos necesaria para elaborar un modelo preciso hace imposible su manejo directo sobre los algoritmos. Es por eso que se tiende a confiar más en las herramientas de alto nivel, con un grado de abstracción mayor, para su control. Un ejemplo podría ser la propia biblioteca de creación de redes neuronales Keras (Ver apartado 5.1.4.2, Keras), o la biblioteca NumPy (Ver apartado 5.1.4.6, NumPy) para el tratamiento de grandes cantidades de datos en estructuras. Ambas proporcionan al usuario comandos de programación abstractos que facilitan tratar con gran cantidad de información sin necesitar conocer lo que sucede a bajo nivel.

6. Conclusión y discusión

En este punto se expondrán algunas de las conclusiones a las que se ha llegado después de haber concluido el desarrollo del proyecto y haber obtenido los resultados de los experimentos.

6.1. Proyecto

Después de haber realizado este trabajo surgen algunas conclusiones que, aun entendiento que son limitadas a las experiencias obtenidas de un solo proyecto, pueden servir de utilidad para entender mejor qué ha sucedido durante todo el proceso y qué información útil se puede extraer.

En primer lugar, cabe recalcar la gran variabilidad en la que esta envuelto el mundo de los modelos basados en redes neuronales y de la ciencia de la visión artificial. Esta ha sido una idea que se ha hecho presente en diversas fases del proyecto. Desde el momento en el que se creaba el entorno de simulación hasta la comprobación y experimentación del modelo, todos las fases del proyecto eran susceptibles a cambios en pequeños parámetros que habrían desenvocado en resultados muy diversos.

El primer ejemplo que se puede tomar para la explicación de este concepto es la simulación. Desde el tamaño de las señales, pasando por la iluminación, los obstáculos, los parámetros de la cámara, la velocidad y distorsión de las capturas. Todos los factores que pudieran afectar en el más mínimo detalle a las imágenes optenidas moldean el resultado final.

De igual manera, la cantidad y criterio a la hora de seleccionar las imágenes, es un aspecto clave a la hora de llevar a cabo un experimento de visión artificial. Unas imágenes muy similares darían unos resultados más precisos y un modelo con una menor tasa de fallo. No obstante, sería poco efectivo si el entorno fuera modificado de alguna forma. De igual manera, una selección demasiado amplia da lugar a un entrenamiento más lento y demandante, y se obtendrá un modelo mucho más pesado.

Finalmente, la creación y entrenamiento del propio modelo también ofrece diversas opciones, cada una con sus propias ventajas y consecuencias. Se podría hacer un modelo más denso, con más capas y mayor concentración de neuronas, pero esto podría conllevar a un modelo sobreentrenado si la variedad de imágenes no es suficiente y siempre se siguen las mismas conexiones. Se podría tomar de igual manera un modelo más ligero, con un menor nombre de capas o filtros, pero el modelo podría perder la capacidad de reconocer patrones que fueran clave para la identificación de los elementos.

Por lo tanto, una de las conclusiones que se puede extraer, una vez concluido el proyecto, es que en caso de realizar de nuevo el proceso, el resultado sería seguramente diverso de lo obtenido.

En segundo lugar, resulta necesario destacar el potencial que tiene esta tecnología.

La ciencia de la visión artificial junto con las herramientas que ofrecen las redes neruonales resultan ser verdaderamente eficientes en la resolución de problemas complejos que requieren de un aprendizaje profundo.

En este proyecto se ha creado un modelo que tenía 600 imágenes de cada señal de tráfico a identificar y se han añadido menos de 10 capas tomando aquellas de convolución, reducción o *pooling* y densas. Además, se han entrenado y comprobado tres modelos distintos, variando los ciclos completos a los datos de entrenamiento o *epochs.* Y con ello se han obtenido resultados muy satisfactorios como se comprobará en el siguiente apartado.

Esto suscita la duda de qué se puede llegar a hacer cuando en lugar de 600 imágenes de cada clase se cuenta con un servidor de datos recogidos a lo largo del

tiempo. O cómo de complejas podrían ser las predicciones si en lugar de este modelo se tuviera uno con muchas más capas convolucionales y densas. De igual manera, es lógico suponer que también se podría lograr una mejor configuración de los parámetros con mejores y más potentes ordenadores iterando el modelo y reajustándolo de forma continua.

El trabajo expuesto en este proyecto ha resultado ser un catalizador para el surgimiento de nuevas ideas y posibilidades. Es innegable conceder que esta tecnología tiene el potencial para desempeñar labores complejas y construir procedimientos que todavía se desconocen.

6.2. Experimentación

Para la experimentación se han construido tres modelos diferentes. Estos modelos comparten las mismas fotografías tomadas por el visor. La única diferencia entre ellos será la cantidad de veces que han sido sometidos al ciclo completo de datos o *epochs*, teniendo en cuenta que en cada iteración las imágenes se entregan en orden distinto y son separadas de forma diferente entre imágenes de entrenamiento, validación y test.

Con los resultados obtenidos de la experimentación será posible obtener conclusiones acerca de los parámetros a tener en cuenta para esta aplicación.

En cada uno de los experimentos se ha realizado la gráfica de función de pérdida y precisión generados por el modelo durante el entrenamiento.

Además, se expondrán los resultados de las predicciones junto con las señales reales. Para ello se han tomado 10 cinemáticas de la simulación en entornos distintos en los que se han modificado parámetros que oscilan desde los tamaños de las señales, hasta la cantidad y tipo de elemento obstaculador, como también la rotación de la cámara.

Cada una de estas cinemáticas será procesada por el modelo y se obtendrán 10 predicciones en cada uno de los tramos en los que aparece una señal. Como resultado, de cada señal se obtendrán 100 predicciones tomadas en 10 entornos diferentes.

Esto servirá para realizar una comparativa real entre los distintos modelos.

6.2.1.Evaluación

Para evaluar los modelos y poder compararlos se seguirá una metodología común.

El primer paso consistirá en anotar las predicciones que se obtienen de los diferentes circuitos de prueba. A continuación, se realizará la matriz de confusión de cada una de las señales de cada uno de los modelos. De esta forma se podrán obtener los indicadores de precisión, exactitud, sensibilidad y especifidad que el modelo presenta para cada una de las señales.

6.2.1.1. Matriz de confusión

La matriz de confusión es una herramienta cuya función es la de visualizar de forma rápida el rendimiento del algoritmo empleado en modelos que son entrenados mediante aprendizaje supervisado.

Para una mejor comprensión de la herramienta se construirá una matriz que sirva de ejemplo. En esta aplicación se desea diseñar un modelo que diferencie si en una imagen aparece una playa o una montaña. Por lo tanto, tanto en las imágenes reales como en las predicciones realizadas por el modelo solo podrá haber playas o montañas.

Se construye la tabla a continuación:

		Valor Real	
		Playa	Montaña
Valor Predicho	Playa	3	2
	Montaña	1	5

Tabla 12 Ejemplo de matriz de confusión

De esta tabla se pueden obtener cuatro parámetros que servirán para calcular los indicadores con los que se evaluará el modelo.

En primer lugar, está la cantidad de verdaderos positivos (**VP**). Este número representa la cantidad de veces en las que el modelo ha predicho correctamente la clase que se toma de referencia. Por ejemplo, si la clase que se estuviera estudiando fuera la de *Playa*, la cantidad de verdaderos positivos sería de *3*.

De igual manera también está la cantidad de falsos positivos (**FP**). Este número representa la cantidad de veces en las que el modelo ha predicho erróneamente la clase que se toma de referencia. Si la clase de referencia fuera la de *Playa*, el número de falsos positivos sería de *2.*

En tercer lugar, se encuentra la cantidad de verdaderos negativos (**VN**). Este número representa la cantidad de veces que el modelo ha predicho correctamente que la clase mostrada no era la de referencia. Para el ejemplo anterior, si se tomara de referencia la clase *Playa*, la cantidad de verdaderos negativos sería de *5.* Cabe recalcar que la cantidad de verdaderos negativos no tiene en cuenta si la predicción de la clase, que no es la de referencia, es correcta o no. Esto quiere decir que si al modelo del ejemplo se le enseñara una imagen de una nueva clase llamada *Desierto*, y el modelo predijera que se trata de *Montaña*, este caso contaría como un verdadero negativo si la referencia fuera de la clase *Playa*, incluso si es una predicción errónea.

Finalmente, está la cantidad de falsos negativos (**FN**). Este valor recoge todos los casos en los que al modelo se le enseña la clase de referencia y éste predice otra clase. En el ejemplo anterior, y tomando una vez más la clase *Playa* como referencia, la cantidad de falsos negativos sería de *1*.

6.2.1.2. Indicadores

Una vez realizada la matriz de confusión y anotados sus parámetros, se pueden calcular sus indicadores. Estos servirán para comparar los modelos entre sí. Es necesario destacar que, al igual que los parámetros de la matriz de confusión (**VP**, **FP**, **VN** y **FN**), los indicadores también son únicos de cada clase que se tome como referencia. Por lo tanto, es posible que un modelo en concreto tuviera una gran precisión en sus predicciones de una cierta clase y, al mismo tiempo, fuera impreciso prediciendo el resto de las clases.

El primer indicador es el de la precisión. Éste indicador mide la cantidad de predicciones correctas (**VP** + **VN**) respecto al total de predicciones (**VP** + **FP** + **VN** + **FN**). Se calcula de la siguiente manera:

$$Precisión = \frac{VP + VN}{VP + FP + VN + FN}$$

$$Precisión = \frac{3 + 5}{3 + 2 + 5 + 1} = 0'73$$

En el ejemplo anterior, el modelo haría predicciones de la clase *Playa* con una precisión de 0’73.

En segundo lugar, se encuentra la exactitud. Este indicador mida la cantidad de veces que un modelo predice de forma correcta la clase de referencia (**VP**) respecto a la cantidad total de veces que predice esa clase, correctas (**VP**) e incorrectas (**FP**). Se calcula de la siguiente manera:

$$Exactitud = \frac{VP}{VP + FP}$$

$$Exactitud = \frac{3}{3 + 2} = 0'6$$

En el ejemplo anterior, el modelo haría predicciones de la clase *Playa* con una exactitud de 0’6.

El tercer indicador a calcular será la sensibilidad. Este indicador mide la cantidad de veces que el modelo predice correctamente una clase (**VP**) respecto a todas las veces que esa misma clase es mostrada, se prediga de forma correcta (**VP**) o no (**FN**). Se calcula de la siguiente manera:

$$Sensibilidad = \frac{VP}{VP + FN}$$

$$Sensibilidad = \frac{3}{3 + 1} = 0'75$$

En el ejemplo anterior, el modelo haría predicciones de la clase *Playa* con una sensibilidad de 0’75.

El último indicador es la especificidad. Este indicador mide la cantidad de veces que al modelo no se le muestra la clase de referencia, y predice de manera correcta que no se trata de esa clase (**VN**), respecto a todas las veces en las que no se le muestra la clase de referencia, tanto si predice que no lo es (**VN**), como si se equivoca y predice falsamente la clase de referencia (**FP**). Se calcula de la siguiente manera:

$$Especificidad = \frac{VN}{VN + FP}$$

$$Especificidad = \frac{5}{5 + 2} = 0'71$$

En el ejemplo anterior, el modelo haría predicciones de la clase *Playa* con una especificidad de 0’71.

Una vez explicados los parámetros e indicadores que se tomarán de referencia se puede proceder con la experimentación de los distintos modelos.

6.2.2.Modelo con 3 ciclos completos

Este es el modelo más ligero de todos.

Su función de pérdida y precisión se pueden observar en los siguientes gráficos:

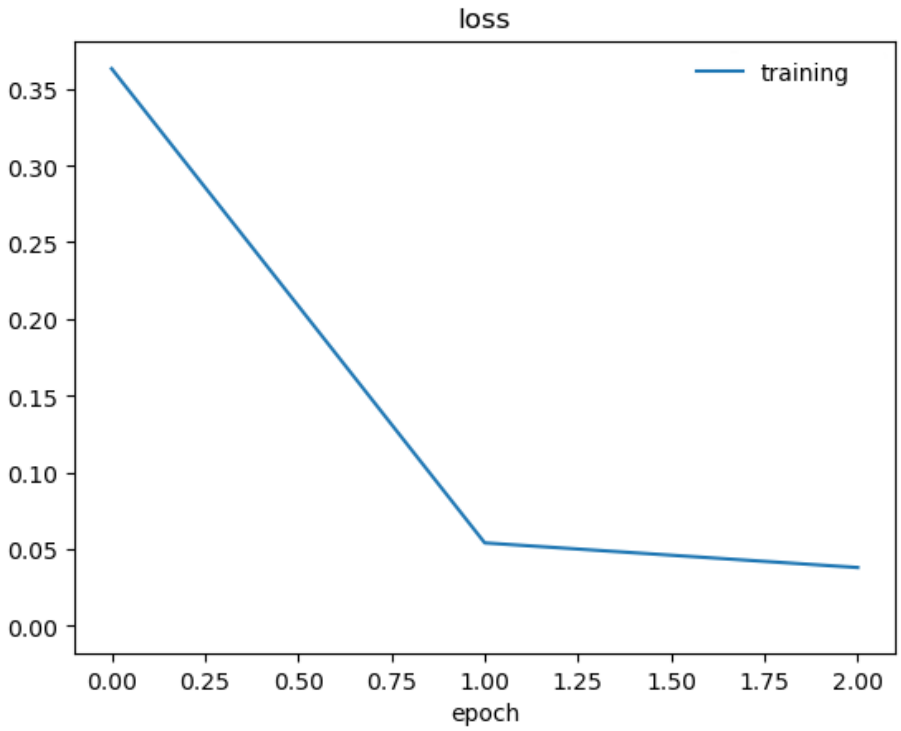

Figura 41 Función de pérdida, modelo de 3 epochs

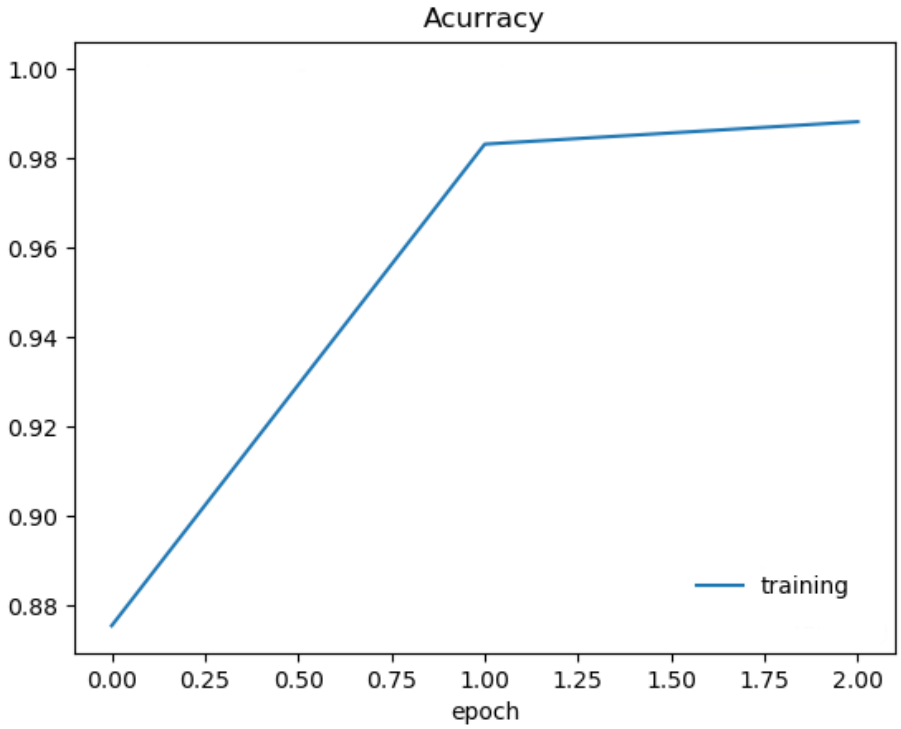

Figura 42 Precisión, modelo de 3 epochs

Al finalizar el proceso de entrenamiento el modelo ofrece los valores finales alcanzados:

```
Test Score: 0.000685452986752183
Test Accuracy: 1.0
```

Figura 43 Resultados del modelo de 3 epochs

Cuando este modelo se pone a prueba con las cinemáticas de examen se obtiene la siguiente matriz de confusión:

		REAL	Encender luces	Vientos fuertes laterales	Maximo 100	Paso de tren	Atencion	Suelo resbaladizo	2 direcciones	Paso de animales	Ceda el paso	No adelantar
P	Encender luces		100									
R	Vientos fuertes laterales			92		2				2	2	2
E	Maximo 100			2	94				2		1	
D	Paso de tren				5	94	1	1	1	3	1	
I	Atencion					2	93		1		3	
C	Suelo resbaladizo			1			1	95		1		3
C	2 direcciones			1			2	1	96			
I	Paso de animales			4		2	1	3		94	1	
O	Ceda el paso						2				92	
N	No adelantar				1							95

Figura 44 Matriz de discusión, modelo de 3 epochs

Con ella se calculan los parámetros:

	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Clase 7	Clase 8	Clase 9	Clase 10
Verdadero Positivo (VP)	100	92	94	94	93	95	96	94	92	95
Falso Positivo (FP)	0	8	5	12	6	6	4	11	2	1
Verdadero Negativo (VN)	900	892	895	888	894	894	896	889	898	899
Falso Negativo (FN)	0	8	6	6	7	5	4	6	8	5

Figura 45 Parámetros del modelo de 3 epochs

Y con ellos se pueden calcular los indicadores:

	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Clase 7	Clase 8	Clase 9	Clase 10
Precisión	1,00	0,98	0,99	0,98	0,99	0,99	0,99	0,98	0,99	0,99
Exactitud	1,00	0,92	0,95	0,89	0,94	0,94	0,96	0,90	0,98	0,99
Sensibilidad	1,00	0,92	0,94	0,94	0,93	0,95	0,96	0,94	0,92	0,95
Especificidad	1,00	0,99	0,99	0,99	0,99	0,99	1,00	0,99	1,00	1,00

Figura 46 Indicadores del modelo de 3 epochs

6.2.3.Modelo con 5 ciclos completos

Este modelo resulta un término medio en cuanto a coste computacional se refiere. Su función de pérdida y precisión se pueden observar en los siguientes gráficos:

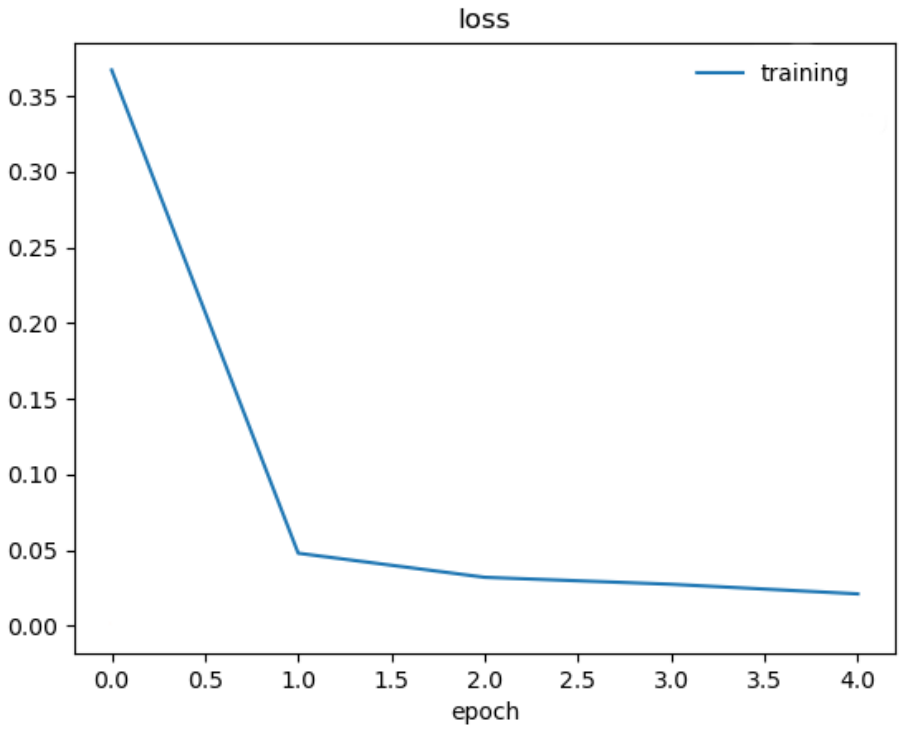

Figura 47 Función de pérdida, modelo de 5 epochs

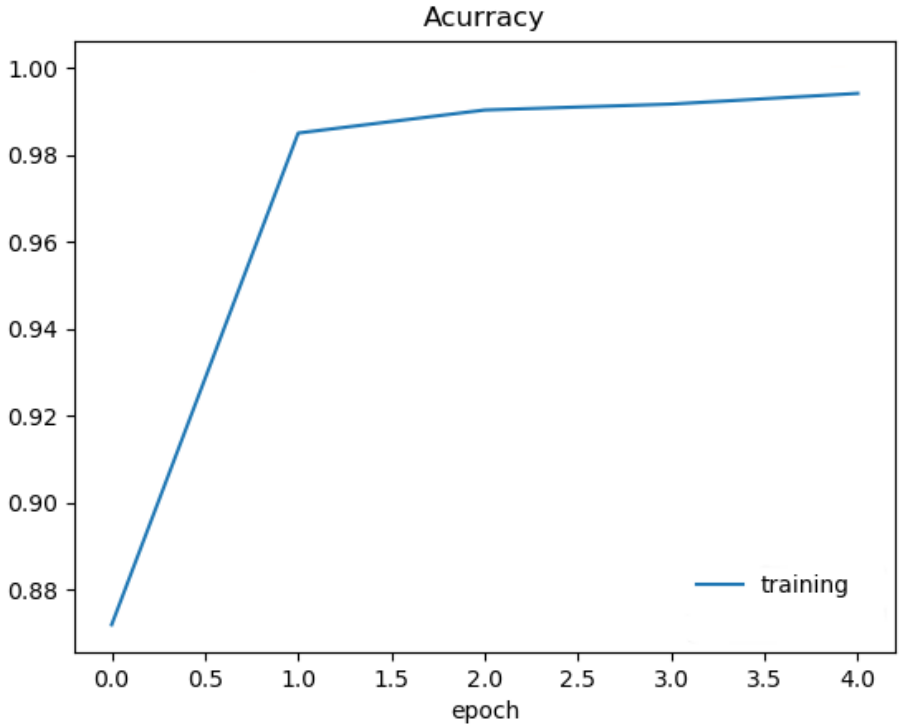

Figura 48 Precisión, modelo de 5 epochs

Al finalizar el proceso de entrenamiento el modelo ofrece los valores finales alcanzados:

```
Test Score: 6.97497453906319e-06
Test Accuracy: 1.0
```

Figura 49 Resultados del modelo de 5 epochs

Cuando este modelo se pone a prueba con las cinemáticas de examen se obtiene la siguiente matriz de confusión:

		REAL: Encender luces	Vientos fuertes laterales	Maximo 100	Paso de tren	Atencion	Suelo resbaladizo	2 direcciones	Paso de animales	Ceda el paso	No adelantar
P	Encender luces	100									
R	Vientos fuertes laterales		94							1	
E	Maximo 100		2	97	1		2	1		2	
D	Paso de tren			2	95		1		3	1	
I	Atencion				2	96		1			
C	Suelo resbaladizo		2	1			95		2		1
C	2 direcciones						1	98			
I	Paso de animales		2		2		1		95	1	
O	Ceda el paso					4				95	1
N	No adelantar										98

Figura 50 Matriz de confusión, modelo de 5 epochs

Con ella se calculan los parámetros:

	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Clase 7	Clase 8	Clase 9	Clase 10
Verdadero Positivo (VP)	100	94	97	95	96	95	98	95	95	98
Falso Positivo (FP)	0	1	8	7	3	6	1	6	5	0
Verdadero Negativo (VN)	900	899	892	893	897	894	899	894	895	900
Falso Negativo (FN)	0	6	3	5	4	5	2	5	5	2

Figura 51 Parámetros del modelo de 5 epochs

Y con ellos se pueden calcular los indicadores:

	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Clase 7	Clase 8	Clase 9	Clase 10
Precisión	1,00	0,99	0,99	0,99	0,99	0,99	1,00	0,99	0,99	1,00
Exactitud	1,00	0,99	0,92	0,93	0,97	0,94	0,99	0,94	0,95	1,00
Sensibilidad	1,00	0,94	0,97	0,95	0,96	0,95	0,98	0,95	0,95	0,98
Especificidad	1,00	1,00	0,99	0,99	1,00	0,99	1,00	0,99	0,99	1,00

Figura 52 Indicadores del modelo de 5 epochs

6.2.4.Modelo con 10 ciclos completos

Este modelo ha sido mucho más costoso de producir.
Su función de pérdida y precisión se pueden observar en los siguientes gráficos:

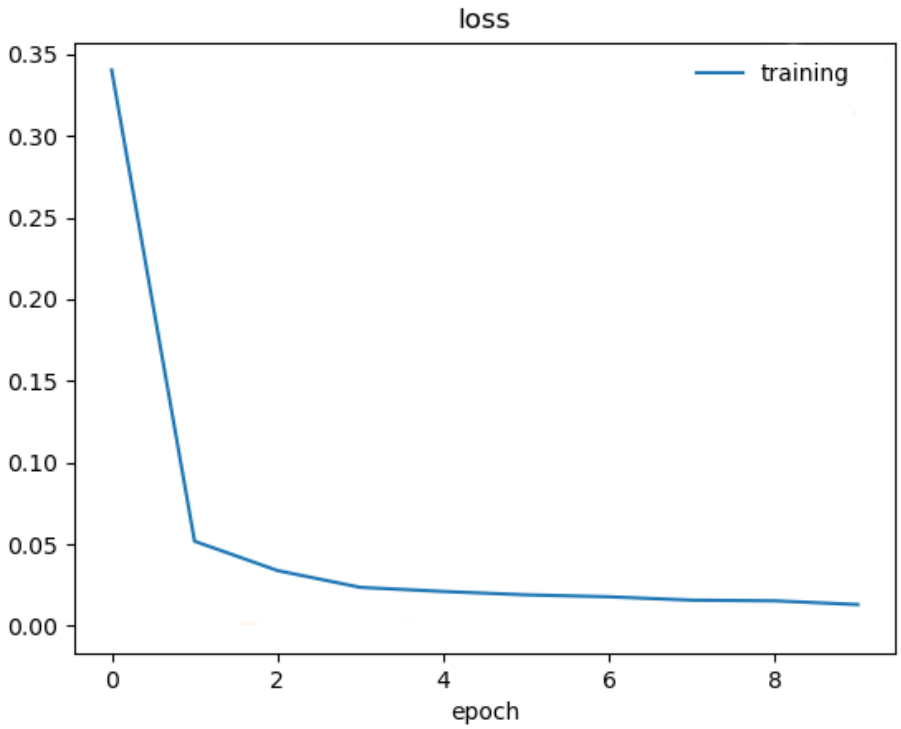

Figura 53 Función de pérdida, modelo con 10 epochs

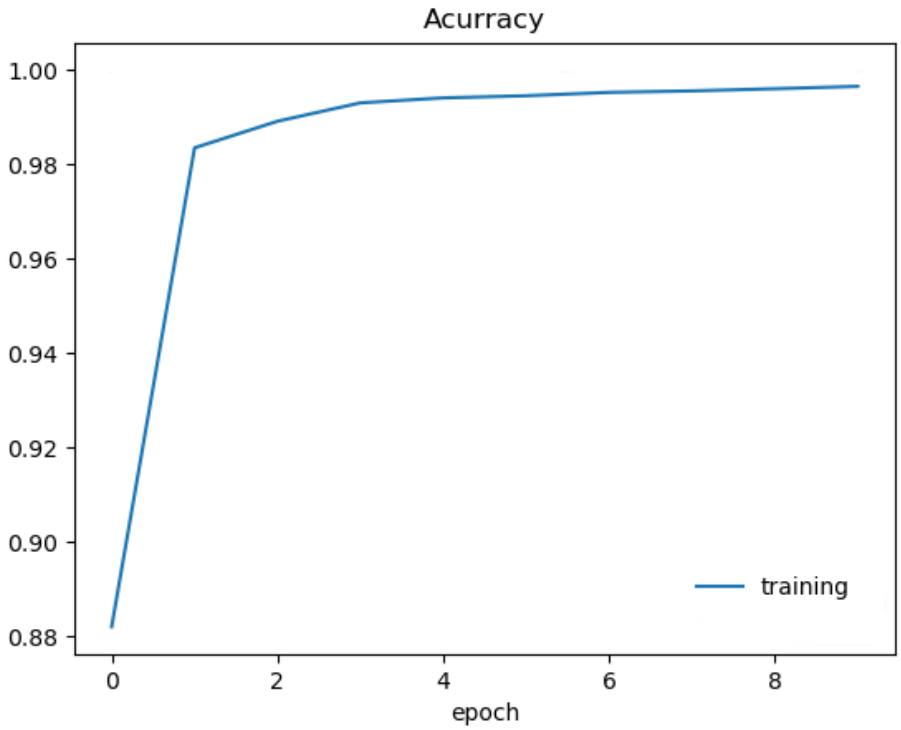

Figura 54 Precisión, modelo con 10 epochs

Al finalizar el proceso de entrenamiento el modelo ofrece los valores finales alcanzados:

```
Test Score: 1.0961761419387574e-08
Test Accuracy: 1.0
```

Figura 55 Resultados del modelo de 10 epochs

Cuando este modelo se pone a prueba con las cinemáticas de examen se obtiene la siguiente matriz de confusión:

		REAL									
		Encender luces	Vientos fuertes laterales	Maximo 100	Paso de tren	Atencion	Suelo resbaladizo	2 direcciones	Paso de animales	Ceda el paso	No adelantar
P	Encender luces	95									
R	Vientos fuertes laterales		92		1				2	2	3
E	Maximo 100	3	2	93	1			2		3	
D	Paso de tren	1		2	92	2	3	1	2	1	
I	Atencion				1	94		1		4	
C	Suelo resbaladizo		1	1		1	92		2		4
C	2 direcciones		3			1	1	95			
I	Paso de animales		2	1	2		2		94	2	
O	Ceda el paso			2	3	1	2			88	
N	No adelantar	1		1		1		1			93

Figura 56 Matriz de confusión, modelo de 10 epochs

Con ella se calculan los parámetros:

	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Clase 7	Clase 8	Clase 9	Clase 10
Verdadero Positivo (VP)	95	92	93	92	94	92	95	94	88	93
Falso Positivo (FP)	0	8	11	12	6	9	5	9	8	4
Verdadero Negativo (VN)	900	892	889	888	894	891	895	891	892	896
Falso Negativo (FN)	5	8	7	8	6	8	5	6	12	7

Figura 57 Parámetros del modelo de 10 epochs

Y con ellos se pueden calcular los indicadores:

	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Clase 7	Clase 8	Clase 9	Clase 10
Precisión	1,00	0,98	0,98	0,98	0,99	0,98	0,99	0,99	0,98	0,99
Exactitud	1,00	0,92	0,89	0,88	0,94	0,91	0,95	0,91	0,92	0,96
Sensibilidad	0,95	0,92	0,93	0,92	0,94	0,92	0,95	0,94	0,88	0,93
Especificidad	1,00	0,99	0,99	0,99	0,99	0,99	0,99	0,99	0,99	1,00

Figura 58 Indicadores del modelo de 10 epochs

6.3. Conclusiones

En este apartado se expondrán las conclusiones derivadas del proceso de experimentación de los diferentes modelos.

En primer lugar, se encuentra el modelo con 3 epochs. Los resultados obtenidos tras su experimentación son razonables. El modelo identifica las señales en la gran mayoría de los casos. Teniendo en cuenta su reducido entrenamiento, y el poco coste computacional que tiene, este modelo cumple con lo esperado. No obstante, se podrían conseguir unas prestaciones mejores si se implementaran algunas de las mejoras que se mencionan en el siguiente apartado (Ver apartado 7.1, Mejoras para este proyecto).

Los resultados muestran que el modelo con mejores prestaciones es aquel que se entrena con 5 epochs (ciclos completos de imágenes de entrenamiento). Su desempeño es ligeramente superior al de 3 epochs. Este modelo refleja la mejora lograda cuando la cantidad de imágenes de entrenamiento es mayor. El algoritmo dispone de un mayor número de ejemplos y por lo tanto es capaz de ajustar y optimizar los parámetros internos de forma que reconozce mejor los patrones.

Por último, se encuentra el modelo con 10 epochs, el cual ha demostrado tener un peor rendimiento respecto a los otros modelos. Esto se debe a un fenómeno conocido como “sobreentrenamiento”. Este sucede cuando durante la fase entrenamiento a un modelo se le ofrecen las mismas imágenes un número elevado de veces. El modelo consolida sus parámetros para lograr la mejor precisión posible con estas imágenes de forma que cuando se le entregan imágenes nuevas le resulta más dificil asociarlas.

Esto sucede por la forma que tienen las redes neuronales de ajustarse. Con cada imagen cambian sus parámetros, de forma más sutil a medida que se suceden las imágenes, para mejorar el rendimiento medio del modelo. Por lo tanto, un modelo que se entrena en demasiadas ocasiones con el mismo ciclo completo de imágenes o epochs, será muy certero reconociendo esas imágenes de la fase de entrenamiento, pero tendrá dificultades para identificar nuevas imágenes.

7. Ideas futuras

7.1. Mejoras para este proyecto

Durante la realización de este proyecto se han necesitado consultar muchas fuentes de información tanto para la parte de simulación como para todo lo referente a la programación del modelo y su entrenamiento.

Es por ello que una vez realizado y terminado este proyecto parece necesario revisar, con los conocimientos adquiridos, aquellos aspectos que se podrían mejorar.

7.1.1.Simulación

Comenzando por la simulación, hay una serie de elementos que se podrían optimizar para lograr obtener un entorno que sea más interesante para entrenar un modelo de visión artificial.

7.1.1.1. Vehículo

El vehículo en esta simulación sirve como soporte para el visor encargado de tomar las imágenes en conducción. No obstante, y pese a que ha servido para dimensionar algunos parámetros acerca del posicionamiento de la cámara, en la realidad el vehículo es un factor clave que afecta en gran medida a la calidad de los datos escogidos.

Curvatura de la luna, transparencia del cristal, distancia de la lente, suciedad, daños, son solo algunos de los elementos que influyen a las imágenes tomadas en movimiento.

Estás desviaciones se podrían llegar a parametrizar y simular. Por ejemplo, se podría añadir ruido aleatorio al visor, hacer alteraciones sobre la perspectiva y las características del punto focal, entre otros.

Con estos cambios se obtendría un vehículo que representara de una forma más fidedigna algunos problemas reales a los que los sistemas de visión en carretera se ven enfrentados.

7.1.1.2. Entorno

La función principal del entorno en este proyecto ha sido la de entorpecer de alguna manera la visión de la cámara, ya sea con cambios de luces o elementos naturales que ocuparan espacio en las imágenes capturadas.

A pesar de que el entorno ha sido útil en este proyecto existen algunas mejoras que se podrían aplicar para obtener una simulación más realista.

La incorporación de otros vehículos con la influencia que ello supone (Faros que reflejan, dificultad para ver las señales), la variación de condiciones ambientales (Lluvia, niebla, etc), o incluso añadir objetos animados, como animales o señales luminosas intermitentes, harían del entorno una representación más interactiva y similar a la realidad.

7.1.1.3. Señales

Las señales de tráfico son el elemento principal dentro del escenario de simulación. A continuación, se expondrán algunas ideas que se podrían implementar en relación con ellas.

En primer lugar, se podría aumentar la cantidad y variedad de señales. El proyecto actual cuenta con 10 clases distintas, el motivo de esta decisión era la búsqueda de suficientes señales como para comprobar la respuesta del modelo a diferentes patrones, pero sin comprometer los tiempos de renderizado y almacenamiento. No obstante, una versión extendida de este proyecto podría buscar de incluir todas las señales que se pueden encontrar en las carreteras, incluso añadir señales de diferentes regiones.

También se podrían implementar señales con desperfectos tales como pintadas, golpes, pérdida del color, etc.

7.1.1.4. Animación

La animación de los distintos elementos también se podría mejorar añadiendo comportamientos aleatorios.

Se podría hacer que el vehículo se desplazara y rotara ligeramente de forma aleatoria, imitando el movimiento natural de los automóviles en carretera, de esta forma el visor no seguiría una trayectoria recta perfecta y por lo tanto tomaría capturas con diferentes perspectivas.

De la misma forma las luces se podrían modificar para provocar reflejos, simular un ciclo diario con distintas luminosidades dependiendo de la hora, incorporar la noche, entre otros.

7.1.2.Modelo de reconocimiento

El modelo de reconocimiento ha demostrado ser efectivo en la identificación de estas señales, no obstante, hay algunos cambios que se podrían implementar para lograr mayores prestaciones y flexibilidad en sus predicciones.

7.1.2.1. Creación

Dentro de los modelos de reconocimiento existen un sinfín de opciones. Algunas más optimizadas para aplicaciones concretas. En este caso se ha optado por una red neuronal convolucional.

Para este proyecto el modelo debía identificar la señal que aparecía en una imagen de reducida resolución dentro de un grupo de 10 opciones.

Alternativamente, se podría diseñar un modelo que pudiera reconocer más de una señal dentro de la misma imagen. Un ejemplo de red neuronal que dispone de esta capacidad es la famosa YOLO, un sistema de detección de objetos programado en C que en los últimos años se ha viralizado por sus notables avances en reconocimiento de imágenes con múltiples elementos en tiempo real.

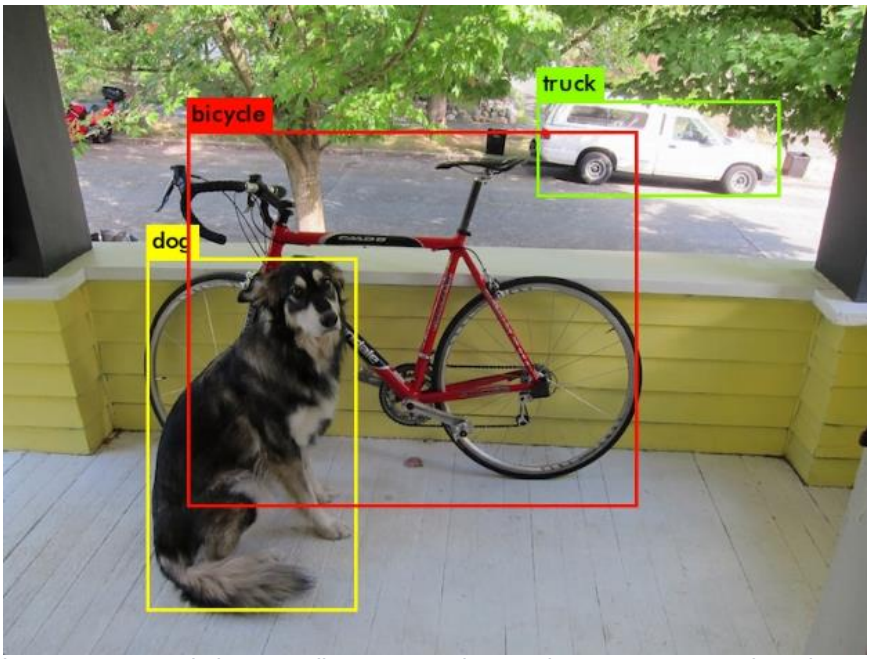

Figura 59 Reconocimiento mediante YOLO de tres elementos en una misma imagen

https://pythondiario.com/2018/09/yolo-you-only-look-once-deteccion-de.html

7.1.2.2. Entrenamiento

A la hora de entrenar un modelo de reconocimiento de imágenes hay algunos factores que son realmente importantes y que se podrían mejorar.

En primer lugar, están las imágenes, para este proyecto se han tomado alrededor de 600 imágenes por clase, esta cantidad es razonable para obtener un resultado satisfactorio con el entorno simulado elegido, y a la vez, poder realizar varios entrenamientos iterando parámetros con el fin de optimizar el modelo.

En cualquier caso, esta cantidad podría ser mucho mayor. Un modelo que se entrena con más imágenes variadas se vuelve más robusto a la hora de enfrentarse a imágenes nuevas. Esto es debido a la forma en que funcionan las redes neuronales, que perfeccionan sus parámetros y reconocen patrones más complejos con cada nuevo conjunto de datos que reciben para su entrenamiento.

Además de la cantidad, también se podrían cambiar las características de los datos entregados. En este proyecto el vehículo se encontraba siempre a una distancia relativamente similar a lo largo de las diferentes simulaciones. No obstante, se podría desear un modelo que fuera capaz de reconocer las señales cuando se encuentran a kilómetros de distancia, señales invertidas o extremadamente ladeadas, reconocer las señales independientemente de su tamaño o de la forma de su base, etc.

8. Bibliografía

-G. Bebis, D. Egbert and M. Shah, "Review of computer vision education," in IEEE Transactions on Education, vol. 46, no. 1, pp. 2-21, Feb. 2003, doi: 10.1109/TE.2002.808280. https://ieeexplore.ieee.org/document/1183662

-T. Hong and P. Wang, "Artificial Intelligence for Load Forecasting: History, Illusions, and Opportunities," in IEEE Power and Energy Magazine, vol. 20, no. 3, pp. 14-23, May-June 2022, doi: 10.1109/MPE.2022.3150808.
https://ieeexplore.ieee.org/document/9761176

-Zhang, H., Hong, X. Recent progresses on object detection: a brief review. Multimed Tools Appl 78, 27809–27847 (2019). https://doi.org/10.1007/s11042-019-07898-2

- Weili Fang, Lieyun Ding, Peter E.D. Love, Hanbin Luo, Heng Li, Feniosky Peña-Mora, Botao Zhong, Cheng Zhou, Computer vision applications in construction safety assurance, Automation in Construction, Volume 110, 2020, 103013, ISSN 0926-5805
https://doi.org/10.1016/j.autcon.2019.103013

-Kavya P. Divakarla, Ali Emadi, Saiedeh Razavi, Saeid Habibi and Fengjun Yan, “A review of autonomous vehicle technology landscape” in InderScience Magazine, Vol.11, No. 4, September 2019.
https://www.inderscienceonline.com/doi/abs/10.1504/IJEHV.2019.102877

-Chan, T.K.; Chin, C.S. Review of Autonomous Intelligent Vehicles for Urban Driving and Parking. Electronics 2021, 10, 1021. https://doi.org/10.3390/electronics10091021

-Kitaguchi, Daichi MD; Watanabe, Yusuke MD, PhD; Madani, Amin MD, PhD; Hashimoto, Daniel A. MD, MS, Meireles, Ozanan R. MD; Takeshita, Nobuyoshi MD, PhD; Mori, Kensaku PhD||; Ito, Masaaki MD, PhD; on behalf of the Computer Vision in Surgery International Collaborative. Artificial Intelligence for Computer Vision in Surgery: A Call for Developing Reporting Guidelines. Annals of Surgery: April 2022 - Volume 275 Issue 4 - p e609-e611 doi: 10.1097/SLA.0000000000005319
https://journals.lww.com/annalsofsurgery/Fulltext/2022/04000/Artificial_Intelligence_for_Computer_Vision_in.29.aspx

- Ghosh, A., Sufian, A., Sultana, F., Chakrabarti, A., De, D. (2020). Fundamental Concepts of Convolutional Neural Network. In: Balas, V., Kumar, R., Srivastava, R. (eds) Recent Trends and Advances in Artificial Intelligence and Internet of Things. Intelligent Systems Reference Library, vol 172. Springer, Cham. https://doi.org/10.1007/978-3-030-32644-9_36

INDICE DOCUMENTO Nº2 PLIEGO DE CONDICIONES

1. Objeto

El presente documento tiene como objeto concretar las pautas de seguimiento y condiciones técnicas a tener en cuenta durante el desarrollo del modelo de identificación de señales de tráfico y la simulación en la que se basa.

2. Normativa

La normativa aplicable para la realización del proyecto es la siguiente:

-IEC 62061. Seguridad funcional de los sistemas de control eléctrico, electrónico y electrónico programable relacionados con la seguridad.

-UNE 199141-1:2013. Equipamiento para la gestión del tráfico. Visión artificial. Lectores de matrículas. Parte 1: Especificaciones funcionales.

-UNE 199141-2:2013. Equipamiento para la gestión del tráfico. Visión artificial. Lectores de matrículas. Parte 2: Protocolos aplicativos.

-IEC 61000-6-1:2005. Compatibilidad electromagnética (CEM) Parte 6-1: Normas genéricas.

3. Entorno de simulación

En caso de que el sistema de identificación de señales de tráfico mediante visión e inteligencia artificial se desarrollara a partir de una simulación, ésta deberá seguir las siguientes directrices para garantizar una representación fidedigna del comportamiento esperable en un entorno real.

La simulación deberá incluir tantos elementos ambientales como hubiera en el entorno real representado a fin de reproducir con la mayor exactitud posible aquellas condiciones susceptibles de propiciar inconvenientes a la toma de imágenes. Algunos ejemplos, pero no todos, son los elementos naturales como vegetación y paisajes, gradientes lumínicos que afecten parcial o totalmente a la superficie reflectante de las señales, entre otros.

Será obligatorio garantizar que en cada simulación de entrenamiento haya parámeteros que se ven variados con el fín de obtener condiciones heterogéneas entre las distintas tomas. Los parámetros a modificar y la magnitud de dichas modificaciones serán ajustadas a las necesidades de la aplicación a la que se encuentre dirigido el proyecto.

En la medida de lo posible, siempre que no afecte a la replicabilidad del proyecto, los elementos que se encuentren tanto en el entorno como en el vehículo de referencia deberán mantener las proporciones y medidas escaladas de sus objetos de referencia reales.

4. Captura de imágenes

El objetivo del proceso de captura de imágenes para el entrenamiento de un modelo de inteligencia artificial debe siempre ser lograr una memoria de datos abundante y completa, que resulte un reflejo fidedigno de las distintas realidades y adversidades a las que se debe exponer el modelo real una vez terminados el entrenamiento y la evaluación.

La toma de imágenes se deberá llevar a cabo en distintas fases del año, horas del día y condiciones medioambientales con el fín de que el conjunto de datos pueda ser considerada una muestra válida del entorno real.

La cantidad de imágenes necesarias para el entrenamiento deberá ser nombroso. Este valor deberá ser fijado dependiendo de la tipología del entorno, la variabilidad de las condiciones en un ambiente real, el rendimiento esperado y necesario, factores de seguridad y riesgo, etc.

Las imágenes no solo deberán ser representativas de las diferentes condiciones a las que se ven expuestas las señales en un entorno real, también deberán presentar diferentes resoluciones a fín de que el modelo resulte más robusto en caso de un descenso de la capacidad de procesado puntual.

5. Modelo de reconocimiento

Para la elaboración, entrenamiento y puesta a prueba del modelo de identificación de señales de tráfico será necesario satisfacer las condiciones técnicas expuestas en el siguiente texto.

El código del algoritmo, y demás programas auxiliares que sean necesarios para realizar el proyecto, deberá seguir las pautas de identado, estructura y organización propias de una buena práctica para un mejor entendimiento por parte de programadores externos. Será también recomendable la adición de comentarios al final o al principio de aquellas líneas de código que puedan ser susceptibles de interpretaciones erróneas.

Con el fin de obtener un algoritmo eficiente y reducir el coste computacional de cada modelo generado será posible realizar pruebas de rendimiento y rastreo de ineficiencias en caso de ser necesario.

La elección de la arquitectura de la red neuronal así como los distintos parámeteros que la carácterizan (número y tamaño de filtros, funciones de activación, orden de las capas, etc) debe ser el resultado de una profunda investigación y comparación para encontrar la mejor solución al problema encontrado. Todas las arquitecturas deberán ser estudiadas y revisadas en caso de no obtener los valores de rendimiento mínimos esperados.

El proceso de entrenamiento deberá finalizar con un diagnosis simplificado que muestre el rendimiento que presenta el modelo una vez diseñado. Será también posible en caso de que se especifique, obtener diagnosis más compleja a lo largo de los diferentes ciclos de entrenamiento para lograr un seguimiento con mayor profundidad y mejorar la trazabilidad de errores.

Será altamente recomendado realizar diversos modelos similares para poder comparar los rendimientos y escoger aquel que ofrezca el comportamiento más deseable.

Es obligatorio el establecimiento de unos indicadores y/o metodologías comúnes a todos los modelos para el proceso de evaluación. La recomendación es realizar una matriz de confusión por cada clase identificable y por cada modelo.

El modelo final debe ser aquel que presente mejores rendimientos en los indicadores que son considerados relevantes para la aplicación.

DOCUMENTO Nº3 PRESUPUESTO

Cuadro de precios elementales

Ref	Unidad	Descripción	Precio (€)
Materiales			
c1	ud	Cámara de captación de imágenes	519,00
c2	ud	Visor	304,11
c3	ud	Unidad de control	2300,00
c4	ud	Kit de cables	30,00
c5	ud	Ordenador	890,00
M.O.D. (Mano de obra directa)			
h1	Técnico de software		35
h2	Técnico auxiliar		20
h3	Técnico instalador		25

Presupuesto de ejecución

Costes de M.O.D.					
Ref	**Unidad**	**Descripción**	**Precio (€/h)**	**Cantidad**	**Parcial (€)**
h2	h	Captación de las imágenes para el entrenamiento del modelo	20,00	80	1600,00
h1	h	Programación del programa para crear el modelo de reconocimiento	35,00	16	560,00
h1	h	Entrenamiento	35,00	40	1400,00
h1	h	Evaluación del modelo con imágenes de muestra	35,00	30	1050,00
h3	h	Instalación del sistema en el vehiculo	25,00	2	50,00
h2	h	Ensayos sobre el sistema de identificación en un entorno real	20,00	160	3200,00
h2	h	Evaluación del modelo en funcionamiento	20,00	30	600,00
Medios auxiliares					
	%	M.A. sobre los costes directos	7	8460,00	592,20
Mediciones					
Ref	**Unidad**	**Descripción**	**Precio (€)**	**Cantidad**	**Total (€)**
c1	ud	Cámara de captación de imágenes	519,00	1	519,00
c2	ud	Visor	304,11	1	304,11
c3	ud	Unidad de control	2300	1	2300,00
c4	ud	Kit de cables	30	1	30,00
c5	ud	Ordenador	890	1	890,00
				Total	**13095,31**