**UNIVERSIDAD AUTÓNOMA DE MADRID**
**ESCUELA POLITÉCNICA SUPERIOR**



**Grado en Ingeniería Informática**

# TRABAJO FIN DE GRADO

**Reconocimiento de situaciones mediante modelos de inteligencia artificial**

**Autor: Diego Tapia Sánchez**
**Tutor: Juan Jesús Roldan Gómez**

**07 2024**

# Resumen

En este proyecto se ha realizado análisis de la capacidad de detección de situaciones por medio de la Inteligencia Artificial, específicamente, por medio de redes neuronales. Dicho análisis se centró en las redes convolucionales, las cuales están diseñadas para el análisis de imágenes. A su vez se crearon tres conjuntos de datos distintos para abordar tres problemas de clasificación, detección de accidentes, detección de tipos de accidentes y detección de congestión de carreteras.

Antes del entrenamiento se aplicaron dos tipos de normalización, una en escala de grises y otra a color. También se realizó un análisis del nivel de predicción y tiempo de ejecución de cada uno de los modelos dependiendo del tipo de normalización y conjunto de datos presente.
Por otra parte, se estudió el impacto del sobreajuste sobre los modelos, principalmente causado por el tamaño de los conjuntos de datos. Esto llevó a implementar medidas tanto en los modelos como sobre la ejecución, para así, aumentar la capacidad de predicción y evitar el sobreajuste. Dichas medidas son: capas de Drop Out, Batch Normalization, Validación Cruzada y Aumento de datos.

Por último, se ha comparado todo el estudio realizado respecto a los modelos y los problemas que presentan con una Red Neuronal de YOLO.

Tanto el código como las imágenes generadas se pueden consultar en https://drive.google.com/drive/u/0/folders/1vmwSG2XV1wzND0iCptiagfvCL1Wt8Y5k

# Palabras Clave

Red Neurona, Inteligencia Artificial Red Convolucional, YOLO, dataset, Convolución, Validación Cruzada, Aumento de Datos, Normalización, entrenamiento, validación, regresión lineal, Max Pooling, Batch Normalization y Dropout

# Abstract

In this project, an analysis of the detection capacity of situations using Artificial Intelligence, specifically through neural networks, was carried out. The analysis focused on convolutional networks, which are designed for image analysis. Additionally, three different datasets were created to address three classification problems: accident detection, accident type detection, and road congestion detection.

Before training, two types of normalization were applied: one in grayscale and another in color. An analysis of the prediction level and execution time of each model was also conducted depending on the type of normalization and dataset present. Furthermore, the impact of overfitting on the models, mainly caused by the size of the datasets, was studied. This led to the implementation of measures both in the models and in the execution to increase the prediction capacity and avoid overfitting. These measures include Dropout layers, Batch Normalization, Cross-Validation, and Data Augmentation.

Finally, the entire study conducted on the models and the issues they present was compared with a YOLO Neural Network.

Both the code and the generated images can be accessed at https://drive.google.com/drive/u/0/folders/1vmwSG2XV1wzND0iCptiagfvCL1Wt8Y5k

# KeyWords

Neural Network, Artificial Intelligence, Convolutional Network, YOLO, dataset, Convolution, Cross-Validation, Data Augmentation, Normalization, training, validation, linear regression, Max Pooling, Batch Normalization, and Dropout

# Índice


***Resumen*** ..... ***5***
***Abstract*** ..... ***7***
***Índice*** ..... ***9***
***Índice de figuras*** ..... ***11***
**1 Introducción** ..... **1**
1.1. Motivación ..... 3
1.2. Objetivos ..... 4
**2 Estado del arte** ..... **6**
2.1. Inteligencia Artificial ..... 6
2.2. Regresión Lineal ..... 6
2.3. Redes Neuronales ..... 7
2.4. Red Convolucional ..... 8
2.5. Normalización ..... 10
**3 Creación del Dataset** ..... **12**
**4 Entrenamiento y Validación de las Redes Neuronales** ..... **16**
4.1. Entorno y lenguaje de programación ..... 16
4.2. Cargar información desde Drive ..... 16
4.3. Normalización de imágenes y etiquetas ..... 17
4.4. Modelos de Redes Neuronales ..... 19
4.5. Compilación y ejecución de modelos ..... 22
4.5.1 Ejecución de Modelos Densos: ..... 23
4.5.2. Ejecución de Modelos CNN: ..... 25
4.5.3. Ejecución de Modelos CNN2 y CNN3: ..... 27
4.6. Validación Cruzada ..... 28
4.6.1. Definición y desarrollo ..... 28
4.6.2. Ejecución Modelos CNN2 y CNN3: ..... 29
4.7. Aumento de datos ..... 31
4.7.1 Ejecución de Detección de accidentes: ..... 32
4.7.2. Ejecución de Detección de Tipo de accidentes y Congestión de Carretera: ..... 34
4.7.3. Conclusión del aumento de datos: ..... 35
4.8. YOLO ..... 36
**5 Conclusiones** ..... **38**
**6 Bibliografía** ..... **41**

# Índice de figuras


Figura 1: Gráfica accidentes de tráfico 1
Figura 2: Reducción del índice de mortalidad 2
Figura 3: Arquitectura Red Neuronal 6
Figura 4: Función Escalón 6
Figura 5: Binary Cross Entropy 7
Figura 6: Esquema Red Neuronal 7
Figura 7: Esquema de convolución (9) 8
Figura 8: Imagen normal frente a imagen convolucionada (10) 8
Figura 9: Ejemplo de matriz de convolución (10) 8
Figura 10: Representación de Max Pooling (11) 9
Figura 11: Escenario generado para los datasets 10
Figura 12: colisión entre dos coches Figura 13: Vehículo accidentado 11
Figura 14: Atropello Figura 15: Persona cruzando paso de cebra 11
Figura 16: Colisión y vuelco 12
Figura 17: Nivel Verde de congestión Figura 18: Nivel Amarillo de congestión 13
Figura 19: Nivel Rojo de congestión Figura 20: Nivel Negro de congestión 13
Figura 21: Proceso de redimensión y normalización en 3 Canales (RGB). 16
Figura 22: Proceso de redimensión y normalización en 1 Canal (Escala de grises). 16
Figura 23: Gráfica evolutiva sobre la validación y entrenamiento en modelos densos de detección de accidentes. 21
Figura 24: Gráfica evolutiva sobre el error generado en Modelos Densos de detección de accidentes. 22
Figura 25: Gráfica evolutiva sobre el entrenamiento y validación en Modelos Densos de detección de tipos accidentes y Congestión de carretera. 23
Figura 26: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN de detección de accidentes. 24
Figura 27: Gráfica evolutiva sobre el error generado en Modelos CNN de detección de accidentes. 24
Figura 28: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN de detección de tipos accidentes y Congestión de carretera. 25
Figura 29: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de accidentes. 25
Figura 30: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN3 de detección de accidentes. 26
Figura 31: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de accidentes aplicando Validación Cruzada. 27
Figura 32: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de Congestión de Carreteras aplicando Validación Cruzada. 28
Figura 33: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN3 de detección de detección de accidentes aplicando Validación Cruzada. 29
Figura 34: Distorsión de imagen para aumento de datos. 30
Figura 35: Gráfica evolutiva sobre el entrenamiento y validación en Modelos Densos de detección de accidentes aplicando Validación Cruzada. 31

Figura 36: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de accidentes aplicando Validación Cruzada y Aumento de datos. 31
Figura 37: Ampliación Gráfica figura 34. 32
Figura 38: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de Tipos de accidentes aplicando Validación Cruzada y Aumento de datos. 32
Figura 39: Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de Congestión de carreteras aplicando Validación Cruzada y Aumento de datos. 33
Figura 40: Gráfica evolutiva sobre el entrenamiento en YOLO. 35
Figura 41: Detección de clases en imágenes con cuadros delimitadores 36

# 1

# Introducción

El proyecto se centra en el reconocimiento de situaciones por medio de la inteligencia artificial, específicamente por medio de redes neuronales. A lo largo del tiempo han surgido diversas tecnologías que perfeccionan estos modelos, una de las mas interesantes, y en la que se centra el proyecto, es en la convolución.

Las redes neuronales convolucionales están diseñadas para la detección de patrones en imágenes. De esta manera, se ha realizado un estudio respecto a dichos modelos a la hora de analizar situaciones de tráfico, centrándose en situaciones de riesgo y de accidente. Para ello se han desarrollado tanto los modelos a entrenar como como el conjunto de imágenes que van a servir para el entrenamiento (datasets). Se ha introducido en dichas fotos situaciones de accidente, circulación normal y riesgo de accidente. Además, para cada imagen se ha contado con un conjunto de diferentes actores (vehículos varios, peatones...).

Para garantizar la seguridad y accesibilidad del transporte por carretera en el territorio de los países miembros de la Unión, la UE estableció, en 2018, el llamado Plan de Acción Estratégico de la Comisión sobre Seguridad Vial. Más adelante publicó el Marco político de la UE en materia de seguridad vial de 2021-2030. En este se detalla un nuevo enfoque para la política de seguridad vial, junto con un Plan de Acción Estratégico a medio plazo, con el fin de disminuir la mortalidad en carretera [1,2].

Este tipo de medidas vienen dadas por el incremento global de mortalidad en accidentes de tráfico, como se puede ver en la **Figura 1**. Según la información remitida por la Organización Mundial de la Salud, en este último año (2023) han fallecido más de un millón de personas en carretera. La mayoría de los fallecimientos se producen en países con ingresos medio-bajos, y son la principal causa de muerte entre los jóvenes (individuos de entre 15 a 29 años). En concreto, el mayor índice de mortalidad se observa en Asia Sudoriental y África, con tendencia al alza, y en el Pacífico Occidental [3].

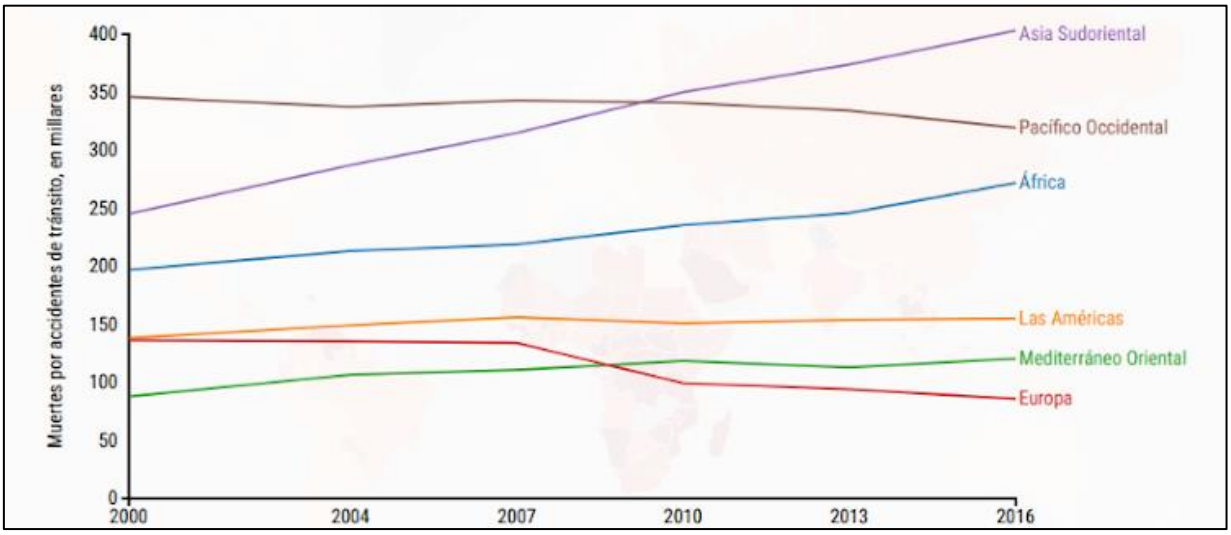



**Figura 1:** Gráfica accidentes de tráfico

Gracias a las medidas aplicadas en el territorio de los países miembros de la Unión, Europa consiguió reducir el índice de mortalidad en carretera un 43% entre 2001 y 2010, y un 21% entre 2010 y 2018, valores reflejados en la **Figura 2**. Sin embargo, en los últimos años no se ha visto un decremento significativo de la mortalidad en carretera. En 2018, 25.150 personas fallecieron en territorio de la UE y 135.000 sufrieron heridas graves, lo que supone una importante pérdida de vidas humanas [2].

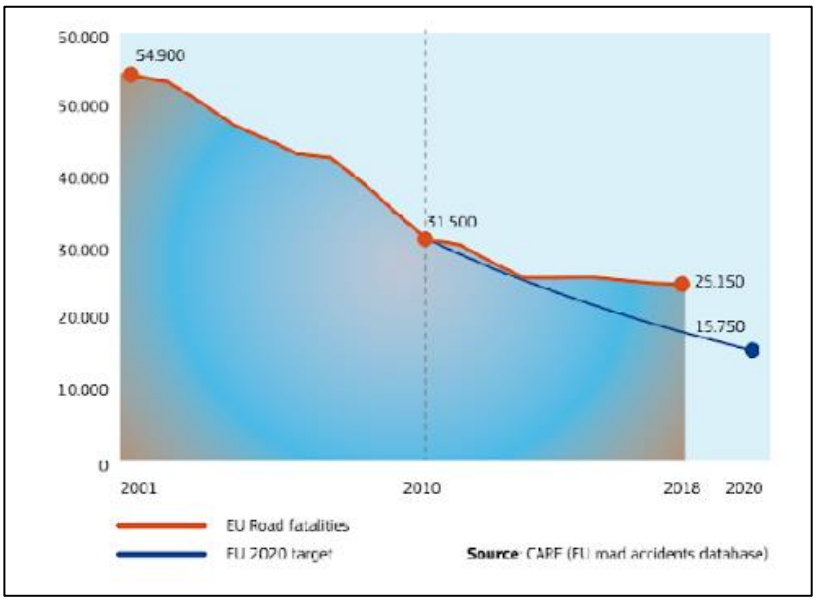



**Figura 2:** Reducción del índice de mortalidad

Según los datos disponibles de la UE, se observa un preocupante incremento de ciclistas fallecidos en los últimos años, principalmente debido a la falta de infraestructuras adecuadas y a acciones peligrosas llevadas a cabo por los propios usuarios, como el exceso de velocidad en la vía pública o la conducción bajo los efectos del alcohol y/o drogas [1].

Si profundizamos en los grupos poblacionales más afectados por este tipo de accidentes, se observa que tres de cada cuatro fallecidos en carretera son hombres. Además, las personas mayores de 65 años y los jóvenes de entre 18 y 24 años experimentan un riesgo considerablemente mayor de sufrir

este tipo de accidentes. Los usuarios vulnerables son los peatones, los ciclistas y los conductores de vehículos a motor de dos ruedas, que constituyen el 70% de las víctimas mortales [1].

Desde un punto de vista puramente económico, se estima que los costes derivados de los accidentes de tráfico en la UE ascienden a 280 mil millones de euros, lo que supone un 2% del PIB [2].

En España, en 2023, se observó un incremento del número de siniestros mortales registrados con respecto a 2022. Sin embargo, el número de fallecidos se mantiene constante, lo que indica que se ha producido una disminución de la letalidad media (número de personas fallecidas por siniestro). Resulta reseñable que 3 de cada 4 fallecimientos se produce en siniestros que tienen lugar en la salida de vías y el incremento del índice de mortalidad (en un 9%) de los usuarios vulnerables [4].

Desde un punto de vista geográfico, las comunidades autónomas más afectadas son Andalucía y Cataluña, aunque en ambas se experimenta una reducción de la tasa de fallecimientos con respecto al año anterior [4].

Por ello, la DGT y el Ministerio del Interior consideran relevante incrementar los niveles de seguridad en las salidas de vía y la revisión de las normas de circulación y seguridad que atañen a los motoristas, con el objetivo de reducir así las cifras de mortalidad vial [4].

Con el avance en la tecnología se plantean nuevas estrategias para hacer frente a esta problemática. Una posibilidad es la utilización de la Inteligencia Artificial para la detección de accidentes de tráfico a partir de imágenes.

Se han planteado diferentes metodologías para la detección automática de incidentes a través del procesamiento de imágenes de tráfico. Una de ellas es el conteo de vehículos, en la que se observa una eficiencia elevada en situaciones de flujo libre y en condiciones climáticas y de iluminación favorables (durante el día y en ambientes soleados o nublados). Este sistema, además, posibilita el análisis en tiempo real de las imágenes analizadas [5].

Sin embargo, en condiciones de baja visibilidad (noche y condiciones climáticas desfavorables) el método propuesto anteriormente no resulta eficiente, dado el número considerable de falsos positivos que genera. La lluvia también parece suponer un problema en el análisis de detección y seguimiento [5].

Otra posibilidad es la aplicación de técnicas de detección de bordes, que posibilita el conteo de vehículos en diferentes condiciones de iluminación con una tasa de detección óptima. Sin embargo, este sistema conlleva ciertos inconvenientes en condiciones de congestión de vehículos [5].

La Inteligencia Artificial también tiene posibilidades en otros contextos relacionados con la seguridad vial. Un ejemplo de ello lo tenemos en el trabajo de fin de máster realizado por Jon Ayuso Hernández. En este, se detallan los problemas que sufre Madrid en sus carreteras debido a la gran afluencia de vehículos. Esto conlleva la congestión de sus vías y el incremento del nivel de contaminación en la capital. Por ello, un posible uso de interés público sería la predicción de atascos en función de diferentes variables [6].

## 1.1. Motivación

La tasa de fallecimientos de tráfico a nivel global ha sufrido un notable incremento, que afecta principalmente a los países menos favorecidos económicamente. Si bien en la Unión Europea se han experimentado grandes avances durante la última década, resulta imprescindible seguir trabajando en medidas que garanticen la seguridad de los usuarios más vulnerables. El uso de la Inteligencia Artificial para la detección rápida de accidentes, mediante el procesamiento de las imágenes de las cámaras de tráfico, podría suponer un importante avance en este contexto.

## 1.2. Objetivos

El objetivo principal es la optimización del proceso de detección de accidentes de tráfico, tipos de accidentes y situaciones de congestión de carreteras, con el fin de agilizar la movilización de los activos necesarios para contener el problema y tratar de disminuir el índice de víctimas mortales de los accidentes de tráfico. Para ello, realizaremos un análisis en profundidad de los resultados de las distintas redes neuronales desarrolladas y de los obtenidos con YOLO.

# 2

# Estado del arte

## 2.1. Inteligencia Artificial

La inteligencia, como concepto, ha sido estudiada desde la antigüedad clásica. Platón y Aristóteles definieron la inteligencia (nous) como la parte analítica del alma humana, capaz de alcanzar las verdades y entender las formas que tejen la realidad. Más adelante, con el Racionalismo, se plantearon definiciones de inteligencia más alejadas del Esoterismo y del contexto religioso. Descartes fue de los primeros en abordar este concepto desde un punto de vista más analítico, incidiendo en la importancia de la razón y el pensamiento crítico [7].

Marvin Minski, uno de los grandes pioneros de la Inteligencia Artificial, la definió como la ciencia de construir máquinas que realizan actividades, simulando el razonamiento humano y que, por tanto, precisan de inteligencia [8]. Actualmente, la Inteligencia Artificial está en boca de todos los medios de comunicación y supone un gran impacto en campos como la educación, la investigación científica o el arte [8].

## 2.2. Regresión Lineal

La regresión lineal es una técnica de análisis y detección de relaciones entre variables. Dicha técnica nos permite entender como una variable puede influir en el valor de otra. Por otra parte, cuando se conoce una posible relación entre dos variables, la regresión lineal permitirá predecir el valor de una de ellas basándonos en el valor conocido de la otra variable [9].

Matemáticamente, para comprender si dos variables están relacionadas, podemos representar los valores que alcanzan por medio de puntos en un mapa cartesiano, generando una “nube de puntos”. Si la representación de estos valores en la gráfica nos genera puntos muy dispersos entre sí, podremos determinar que es poco probable que haya una relación entre estas dos variables. Sin embargo, si los puntos están situados de tal manera que se puede trazar una recta a lo largo de los ejes que represente su tendencia, entonces podremos indicar que dichas variables están relacionadas [9].

Una recta está formada por una pendiente y un término independiente. La pendiente define la inclinación que va a tener una recta, por medio de la relación que se dará entre el valor de entrada ***X*** y la salida ***Y***. El término independiente indica el punto en el que una recta corta el eje ***Y*** [9].

$$Y = W0 + W1*X \text{ (recta)}$$

Aquellos modelos en los que una única variable determina el valor de salida pueden ser representados por medio de una recta y serán considerados de regresión simple. Por otro lado, aquellos modelos en los cuales más de una variable afecte al valor de salida serán representados por planos o hiperplanos en su defecto y serán considerados modelos de regresión múltiple [9].

$$Y = W0 + W1*X1 + W2*X2 \text{ (plano)}$$

## 2.3. Redes Neuronales

Las redes neuronales artificiales son sistemas computacionales inspirados en el funcionamiento de las redes neuronales humanas. Cada red neuronal está conformada por un conjunto de unidades básicas de cálculo a las que llamamos neuronas [10].

Cada neurona consta de un conjunto de valores de entrada, al que se le aplicará una función matemática para obtener un valor de salida o resultado. Dicha función utiliza los valores de entrada para realizar una suma ponderada, que viene dada por cada uno de los pesos (***W***) que se asignan a las conexiones de entrada a la neurona, esta arquitectura se puede comprobar de una forma mas esquemática por medio de la **Figura 3** [10].

De esta manera, podemos definir el comportamiento interno de una neurona como el de un modelo de regresión lineal, en el que vamos a tener un nuevo *término independiente* al que denominaremos ***sesgo*** y representaremos por medio de una ***b*** *[10,11]*.

***Y = W1*X1 + W2*X2 + b*** (neurona)

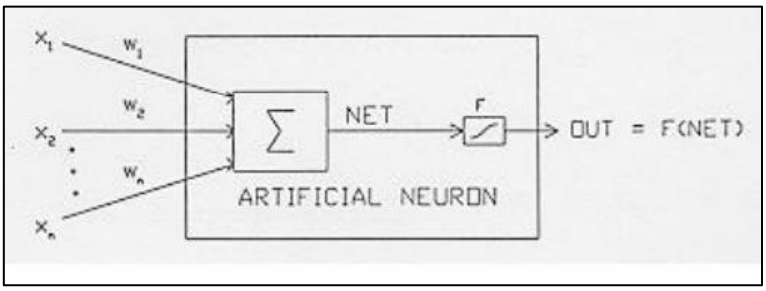



**Figura 3:** Arquitectura Red Neuronal

Una de las principales limitaciones de una única neurona es la incapacidad de resolver problemas complejos por sí misma, como es el ejemplo de la puerta XOR. Es por ello que este tipo de problemas se abordan por medio de un conjunto de neuronas que conformarán una red neuronal [11].

Un componente indispensable en una neurona es su función de activación. Dicha función permite que las neuronas aprendan relaciones y reglas específicas para la clasificación correcta de los resultados [10].

Una de las funciones de activación más conocida es la función escalón, que se muestra en la **Figura 4** (12):

$$f(x) = \{1 \; si \; x \geq 0, 0 \; si \; x < 0$$

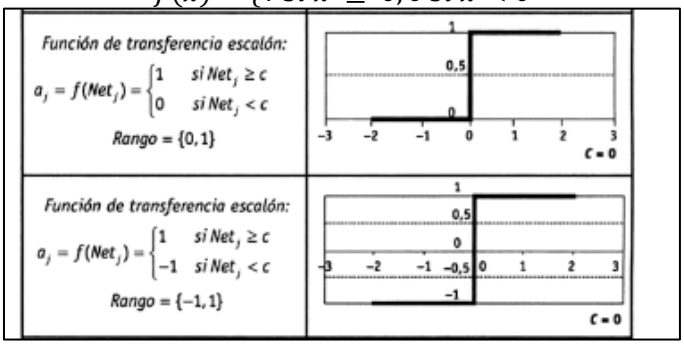



**Figura 4:** Función Escalón

En este trabajo se hizo uso de la función de activación RELU (Unidad Lineal Rectificada), que se explicará más adelante.

El proceso de entrenamiento de una red neuronal está dividido en un conjunto de épocas, en cada una de ellas se hace una predicción de clasificación sobre las imágenes de entrenamiento. Sobre estas predicciones se calcula el valor de la función de pérdida. Este proceso consiste en comparar los valores predecidos con los valores reales de la muestra, la discrepancia entre datos se cuantifica y utiliza para ajustar los pesos del modelo con el fin de disminuir la función de pérdida en la siguiente época. Esta tarea la lleva a cabo el optimizador. De esta manera, según se vayan ejecutando las épocas de entrenamiento, la predicción de las clases será más precisa [13].

Para la clasificación binaria, es decir, aquellos problemas de clasificación que solo constan de dos posibles resultados para una predicción, utilizamos la función de pérdida "Binary Cross Entropy", que está diseñada para este tipo de problemas. Su fórmula es está especificada en la **Figura 5** [14].

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * \log(\hat{y}_i) + (1-y) * (1-\hat{y}_i))$$

**Figura 5:** Binary Cross Entropy

Las redes neuronales se pueden clasificar en función de la distribución de sus neuronas. Si todas las neuronas se encuentran en la misma capa hablaremos de redes neuronales monocapa. Por otro lado, si están distribuidas entre varias capas en cascada, hablaremos de redes neuronales multicapa, en las que la salida de una neurona será el valor de entrada de la neurona de la capa siguiente. Por medio de la **Figura 6,** podemos ver de manera esquemática cómo podría ser una distribución de red neuronal. [11].

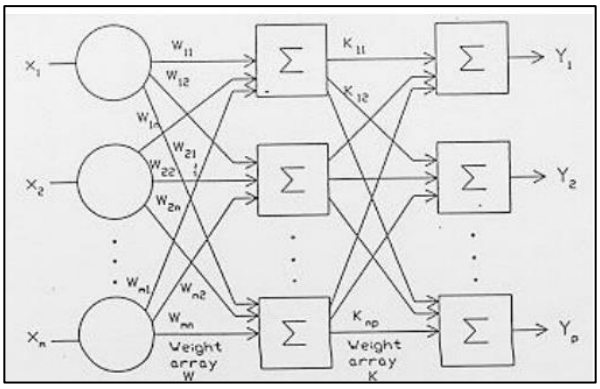



**Figura 6:** Esquema Red Neuronal

## 2.4. Red Convolucional

Las redes neuronales convolucionales son redes multicapa, útiles para la identificación de patrones e inspiradas en el funcionamiento del córtex visual del cerebro. Constan de varias capas ocultas y están organizadas según la capacidad de reconocimiento de patrones complejos. Las primeras capas tendrán la capacidad de reconocer formas simples, mientras que las capas de nivel más profundo nos permitirán reconocer formas de mayor complejidad [15].

Las redes neuronales convolucionales cuentan con dos tipos nuevos de capas: capas de convolución y capas de agrupación. Estas capas se encargan de extraer las características de los valores de entrada para así poder clasificarlas por medio del resto de capas [16].

A la hora de analizar una imagen, necesitamos distinguir el conjunto de formas que la conforman. Para ello es preciso reconocer los ejes que componen y dan sentido a cada uno de los objetos o siluetas. Esta tarea la realizan las capas de convolución [17].

La convolución de una imagen consiste en su distorsión. Para ello, definiremos una matriz 3x3 llamada Kernel, y a cada una de las posiciones de esta matriz le asignamos un peso específico. El Kernel irá recorriendo los píxeles de la imagen teniendo en cuenta aquellos que lo rodean y multiplicará el valor de los pesos con el de los píxeles asignados. Finalmente, se realiza la suma de los valores generados. Este resultado se guardará en la misma posición del píxel analizado, pero en una nueva imagen. En la **Figura 7** se muestra un esquema del funcionamiento del proceso de convolución [17].

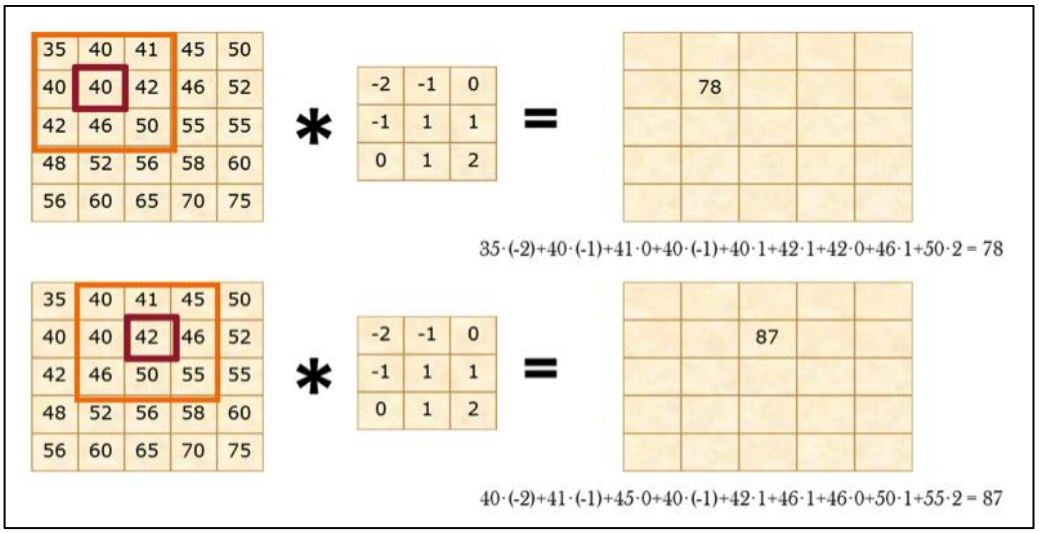



**Figura 7:** Esquema de convolución [17]

Gracias a esta operación podemos asignar unos "pesos" específicos a la matriz para que resalte los ejes de las figuras de la imagen, como es el caso del operador de Sobel. A continuación, en la **Figura 8**, se muestra un claro proceso de convolución en imágenes [18].

**Figura 8:** Imagen normal frente a imagen convolucionada [18]

El operador de Sobel se basa en la utilización de dos máscaras o matrices Kernel, una para la detección de ejes verticales y otra para la detección de ejes horizontales. Esta técnica de detección se aplica en imágenes en escala de grises. En la **Figura 9** se puede apreciar un ejemplo de operador sobel [18].

$$M_h = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad M_v = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

**Figura 9:** Ejemplo de matriz de convolución [18]

Para imágenes a color la convolución contará con tres matrices o kernel, uno para cada canal de color, sease: rojo, verde y azul (RGB). La imagen se dividirá entre los tres canales mencionados y, a

cada una, se le aplicará una matriz de convolución. El resultado de cada una de estas convoluciones se sumará con el del resto de canales, generando una única imagen.

Una vez tenemos la información correspondiente a la capa de convolución, la agrupamos, disminuimos el tamaño de la imagen y resaltamos las características de mayor relevancia. Estas tareas las realiza la llamada capa de agrupación [19].

Al igual que hay distintos tipos de convolución, hay diferentes métodos de agrupación. La técnica de agrupación más utilizada es la Agrupación Máxima o *MaxPooling*. Dicha técnica consiste en generar una matriz de un tamaño determinado e ir recorriendo la imagen por medio de movimiento o saltos acordes al tamaño de la matriz. Este tipo de "salto" se llama Zancada o Stride. Por cada posición que vaya recorriendo la matriz, seleccionamos el píxel de mayor valor y descartamos el resto. Los píxeles seleccionados se guardarán en una nueva imagen de menor resolución, todo este proceso se puede ver reflejado en la **Figura 10** [19].

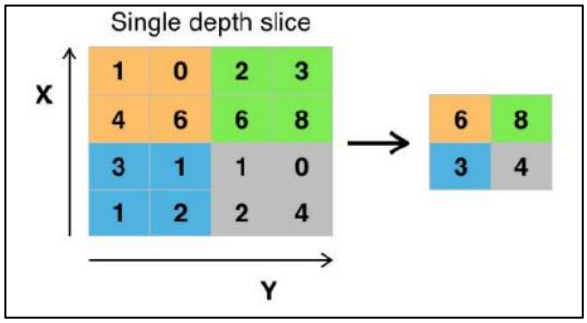



**Figura 10:** Representación de Max Pooling [19]

## 2.5. Normalización

Es común que en el proceso de análisis y clasificación de imágenes se muestren alteraciones dependiendo de la iluminación entre zonas, lo que provoca que los modelos de entrenamiento no sean tan certeros y puedan llegar a fallar. Los principales problemas que se dan a la hora de analizar las imágenes son la *Explosión de Gradiente* y el *Desvanecimiento de Gradiente* [20].
La Explosión de Gradiente ocurre tras una recursiva multiplicación de gradientes de valor muy elevado. Dichos gradientes pueden ocasionar cambios drásticos en los pesos, generando que la convergencia del modelo sea más lenta e incluso que no llegue a converger.

Por otro lado, el Desvanecimiento de Gradiente ocurre tras una recursiva multiplicación de gradientes muy pequeños. Específicamente, se da cuando los gradientes son menores a 1 y genera, al igual que la Explosión de Gradiente, cambios significativos en los pesos, que pueden disminuir el grado de aprendizaje provocando que no sea eficiente aplicar el modelo.

Para evitar estos problemas derivados de los datos originales realizamos un proceso de normalización. La normalización es una técnica que busca ajustar un conjunto de datos a una escala común sin generar distorsiones en los rangos de los valores ni pérdidas de información [20].

Hay varias técnicas que nos permiten normalizar de manera correcta nuestros datos. Entre las distintas técnicas aplicadas en este trabajo, destacaremos la Normalización Min-Max y la técnica de Batch Normalization [20].

Min-Max es una técnica que busca reducir el rango entre los píxeles, de tal manera que ajusta el valor de estos para que se encuentre en el rango [0, 1]. Dicha técnica facilita el entrenamiento y mejora la estabilidad numérica [20].

Batch Normalization es una técnica que realiza la normalización de los datos en una capa de la propia red neuronal. Busca modelizar la media y desviación de la distribución para, posteriormente, realizar la propia normalización. Esta técnica consta de dos fases, la fase de entrenamiento y la fase de

Inferencia. Durante la fase de Entrenamiento, se calcula la media y la varianza del subconjunto de datos actual, tras esto se realiza la normalización para dichos datos. En la fase de Inferencia, se utilizan las estadísticas generadas en la fase de entrenamiento para así normalizar los datos de entrada [20].

# 3

# Creación del Dataset

Este proyecto de inteligencia artificial se enfoca en la predicción de situaciones por medio de imágenes, específicamente detección de posibles accidentes de tráfico, así como la clasificación de sus distintos tipos y detección de factores de riesgo en las carreteras. Para la correcta detección de las situaciones necesitamos un conjunto de imágenes adecuado con las que entrenar a nuestra red neuronal.

Para este propósito, se decidió utilizar Unity como herramienta de generación de escenas, asegurando que se crea un dataset (conjunto de situaciones) correcto y completo para un entrenamiento satisfactorio.

Unity es una plataforma de desarrollo de videojuegos 3D y 2D, que permite la creación de proyectos para distintas plataformas, como puede ser PC, consolas y dispositivos móviles. Unity utiliza C# como lenguaje de programación y es una plataforma con un entorno visual sólido y una interfaz amigable.

En este proyecto se ha trabajado con las distintas herramientas que nos proporciona dicha aplicación para generar los escenarios de los videojuegos en 3D. Unity cuenta con una gran biblioteca de recursos (Assets), objetos y texturas a utilizar. Si bien muchos de estos recursos son de pago, se disponen de algunos gratuitos que se han utilizado para plantear este proyecto.

Para el desarrollo de los dataset se ha creado un gran escenario, como se puede observar en la **Figura 11**, en el que he introducido carreteras, árboles, pasos peatonales, edificios, personas y, como no, una gran cantidad de vehículos.

**Figura 11:** Escenario generado para los datasets

Uno de los problemas principales a la hora de entrenar una red neuronal es seleccionar el conjunto de imágenes correcto para que el modelo entienda cómo hacer la clasificación de manera acertada. Si el dataset no está bien construido, no se producirá un entrenamiento productivo y, por ende, el modelo no convergerá. Por ello, se ha utilizado una gran cantidad de vehículos, tanto accidentados como realizando una circulación normal, para que el entrenamiento no detectase que los accidentes solo se producen en vehículos de cierta forma o color.

A la hora de simular accidentes de tráfico me basé en situaciones reales de colisiones, choques frontales, atropellos y vuelcos. A su vez, introduje un conjunto de vehículos en mal estado, con cristales rotos, carrocería oxidada y abollada, entre otros, para así indicar que independientemente del vehículo a estudio, si este tiene graves desperfectos, podemos encontrarnos ante un accidente de tráfico. Las **Figuras 12 y 13** muestran un ejemplo de alguna de situaciones de accidentes que se han realizado para el proyecto.

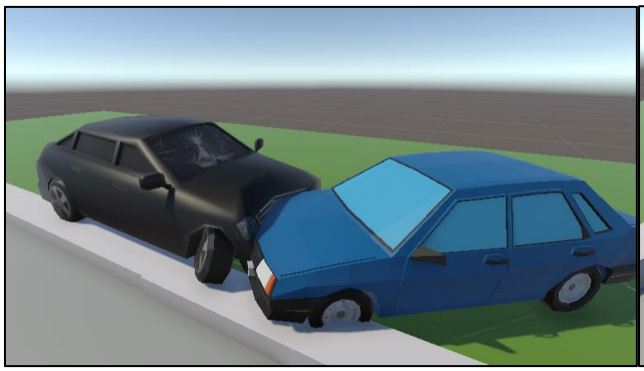


**Figura 12:** colisión entre dos coches

**Figura 13:** Vehículo accidentado

Para simular los atropellos utilicé ciertos assets gratuitos que se encontraban en la biblioteca de Unity. Dichos assets eran modelos de personas completamente articulables, de tal manera que podía simular perfectamente la colisión de un coche con una persona basándose en la posición corporal y la cercanía con el vehículo, como se puede observar en la **Figura 14**. Este mismo asset también se ha utilizado para generar situaciones normales de tráfico, como es el caso de la **Figura 15.** En este caso específico, simulando que las personas están cruzando un paso de cebra sin recibir ningún impacto de los vehículos cercanos.

**Figura 14:** Atropello

**Figura 15:** Persona cruzando paso de cebra

Se han generado imágenes en distintos ángulos y con gran variedad de vehículos, modificando el número de actores principales y situaciones en las que se encuentran para así, conseguir un total de tres datasets correctos y completos, que se detallarán a continuación.

El primer dataset planteado consta de 240 imágenes y permite adiestrar a la red neuronal para detectar la presencia o ausencia de accidentes de tráfico. Originalmente se planteó con 80 imágenes, pero a la hora de validarlo con la red neuronal, se consideró que era un conjunto de imágenes insuficiente. Por ello, se amplió el tamaño del dataset hasta llegar a las 240. En concreto, se decidió aumentar el número de situaciones que mostraban accidentes por atropello, debido a su elevada incidencia según los últimos informes presentados por la DGT en España [4].

Para las imágenes de atropellos, tanto de este primer dataset como de los siguientes explicados a continuación, se decidió utilizar como criterio para su detección el análisis del estado de la persona afectada. Así, si la figura humana se encuentra en posiciones anómalas, tumbada sobre el asfalto o en el capó del coche, con extremidades en posiciones antinaturales, se considera que se ha producido un atropello.

El segundo dataset planteado permite un adiestramiento para la identificación del tipo de accidente, entre los que se incluyen accidentes por colisión, vuelco y atropello. Se seleccionaron estos tres tipos por su gran incidencia, en el caso concreto de los accidentes por colisión y atropellos [4], y por su peligrosidad y letalidad, en el caso concreto de los accidentes por vuelco de vehículo.

Para garantizar un correcto adiestramiento de la red neuronal, en este segundo dataset se introdujeron imágenes en las que se mostraban varios de estos tipos de accidentes simultáneamente, como es el caso de la **Figura 16**. Por ejemplo, en una única imagen puede aparecer un vehículo volcado y otro que haya sufrido una colisión con este. Para este tipo de imágenes, que corresponden a dos categorías a la vez, se ha creado una sección específica.

**Figura 16:** Colisión y vuelco

El segundo dataset consta de 60 imágenes mostrando situaciones de atropellos, 40 de vuelcos, 40 de colisiones y, finalmente, de otras 20 de situaciones mixtas, de vuelco con colisión, obteniendo un total de 160 imágenes para el adiestramiento.

El tercer y último dataset planteado permite la detección del grado de congestión de las carreteras. Para su planteamiento, se tuvieron en consideración los niveles de congestión vehicular utilizados por la DGT, que consta de cinco niveles. El primer nivel, llamado nivel blanco, indica una circulación fluida. El segundo nivel, el nivel verde, indica la existencia de una concentración vehicular considerable, que impide a los vehículos alcanzar la velocidad máxima permitida en la vía. En el siguiente nivel, el nivel amarillo, la circulación es discontinúa, con existencia de detenciones intermitentes. En el cuarto nivel, el nivel rojo, indica la existencia de saturación vehicular, con paradas intermitentes y prolongadas. Por último, en el nivel más elevado de congestión, tenemos el nivel negro, en el que la circulación se encuentra completamente interrumpida.

Como se puede apreciar, la DGT clasifica el estado de congestión en función, no solo del volumen vehicular, sino también de la velocidad de los vehículos en cuestión. Dado que esto no puede ser

valorado en imágenes estáticas, en este proyecto se realizará esta categorización en función de la distancia de separación entre los vehículos.

De esta forma, se ha hecho una adaptación sobre los propios niveles estipulados por la DGT, creado 4 niveles nuevos. El primer nivel será el verde, que es una versión intermedia entre el nivel blanco y verde estipulado por la DGT. En dicho nivel se da circulación de coches por las carreteras, pero la distancia entre cada uno de ellos es bastante amplia y no se deberían producir interrupciones de tráfico **(Figura 17).** El segundo nivel es el amarillo, y la distancia entre vehículos es mucho menor y simula situaciones en la que es probable que se produzcan interrupciones de tráfico **(Figura 18).** En el nivel rojo, la distancia entre los vehículos se acorta aún más, simulando situaciones de circulación con tráfico sumamente congestionado **(Figura 19)**. Por último, tenemos el nivel negro, que al igual que en lo estipulado por la DGT, engloba aquellas situaciones en las que se dé atasco, gran concentración de vehículos con tráfico completamente interrumpido **(Figura 20)**.

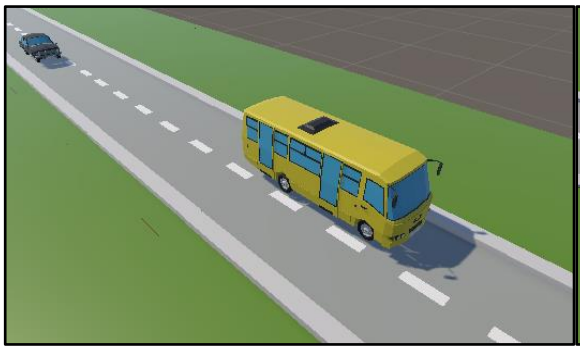


**Figura 17:** Nivel Verde de congestión

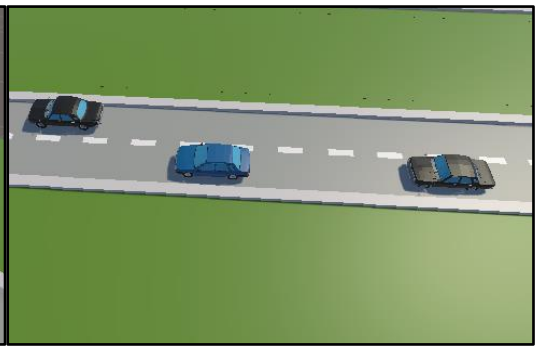


**Figura 18:** Nivel Amarillo de congestión

**Figura 19:** Nivel Rojo de congestión

**Figura 20:** Nivel Negro de congestión

Este último dataset, que consta de 80 imágenes que comprenden los diferentes niveles mencionados, se planteó debido a que existe una correlación entre el nivel de congestión vehicular y el aumento del riesgo de accidentes de tránsito, entre otros como el incremento de contaminación y el consumo de combustible [22]. En casos, por ejemplo, de que el nivel de congestión vehicular detectado sea negro, se tomará como una situación de alto riesgo en lo que respecta a la probabilidad de aparición de accidentes de tráfico.

La elección del tamaño de cada uno de los datasets se ha ido modificando y adaptando a lo largo del desarrollo del TFG, con el fin, no solo de detectar correctamente las situaciones, sino también de simular los posible problemas o desventajas que tienen cada una de las redes neuronales a la hora de su entrenamiento y validación. Es por ello que cada dataset muestra una situación distinta. En este caso, el dataset óptimo es el de detección de accidentes de tráfico, ya que cuenta con 120 imágenes para cada uno de los tipos. Además, solo consta de dos clases distintas, “accidente” o “normal”. El siguiente dataset con mejores resultados, dada la cantidad y variedad de las imágenes, debería ser el de detección de tipo de accidente que consta en este caso con tres clases. Por último,

a nivel de variedad y cantidad de situaciones que representen cada clase, el peor dataset deberá de ser el de congestión de tráfico.

A lo largo del desarrollo de este proyecto se han implementado técnicas que favorecen el entrenamiento e incrementa la eficiencia de aquellos datasets menos óptimos.

Respecto a los assets utilizados para la detección de situaciones, se han utilizado diseños menos realistas que las situaciones reales de accidente. Los accidentes tienen muchas variables entre ellos, y dado que se está haciendo un estudio limitado y no se tienen miles de fotos desarrolladas para realizar un estudio completo, se ha buscado simplificar los modelos, detectando así las características mas esenciales a la hora de clasificar situaciones. En un modelo preparado para el uso en carreteras se debería de utilizar modelos más realistas y ampliar enormemente los datos y posibles situaciones. Sin embargo, a nivel de detectar situaciones, el resultado del análisis sería el mismo, en mayor o menor escala de detalle.

# 4 Entrenamiento y Validación de las Redes Neuronales

Una vez creado el conjunto de datos, damos inicio a la etapa de entrenamiento, para ello debemos codificar los modelos de redes neuronales que vamos a usar a la hora de hacer las predicciones. A su vez, tenemos que escoger un entorno y lenguaje de programación para el posterior desarrollo de dichos sistemas.

## 4.1. Entorno y lenguaje de programación

Para este proyecto he optado por utilizar Google Colab como entorno de programación, una plataforma gratuita basada en la nube que nos permite ejecutar código python. He optado por este entorno de programación basándome en su sencillez y compatibilidad con el conjunto de librerías que se van a implementar a lo largo del proyecto. Además, colab permite variar el entorno de ejecución del proyecto. De esta forma, podemos utilizar un entorno basado en el uso de la CPU o bien usar la GPU.

El único inconveniente de dicha plataforma es que el uso de GPU está limitado por ciertas políticas sujetas al tiempo de sesión, uso diario, uso semanal y disponibilidad. Para disminuir las restricciones de colab es necesaria una suscripción no gratuita a colab pro.

## 4.2. Cargar información desde Drive

El conjunto de las imágenes que conforman cada uno de los tres datasets se ha agrupado y subido a drive. Gracias a la biblioteca de drive en python, se ha cargado el contenido de dichos dataset en un notebook de Colab.

Se han creado 3 funciones para cargar cada uno de los datasets de drive: *cargar_imagenes()*, que corresponde al dataset de detección de accidentes; *cargar_imagenes_tipo_accidente()*, que corresponde al dataset de clasificación de accidentes según su tipo y *cargar_imagenes_congestion_carretera()*, que corresponde al dataset que clasifica las situaciones según el flujo de vehículos e interrupciones que se dan en las carreteras.

Las imágenes almacenadas en drive, están nombradas especificando el tipo de situación y el número de la foto, de tal manera, que en cada una de las funciones de carga se detecta el tipo de situación, analizando el nombre del archivo para así generar las etiquetas de manera correcta. Finalmente, las funciones de carga devolverán 2 listas, una con las imágenes cargadas y otra con las etiquetas de cada una de las imágenes.

A la hora de generar las etiquetas, se tiene en cuenta como va a ser el análisis de sus valores por parte de la red neuronal, de esta forma, se ha diferenciado tres tipos de etiquetas según el dataset utilizado:

- Detección de accidentes:
  - 1: corresponde a una situación de accidente.
  - 0: corresponde a una situación de circulación normal.
- Detección de tipo de accidente:
  - [1,0,0]: corresponde a una colisión.
  - [0,1,0]: corresponde a situaciones de vuelco de uno o más vehículos.
  - [0,0,1]: corresponde a situaciones de atropello.
  - [1,1,0]: corresponde a situaciones de vuelco y colisión simultánea.
- Detección de congestión en carretera:
  - [1,0,0,0]: verde, situaciones con tráfico escaso.
  - [0,1,0,0]: amarillo, situaciones de mayor número de tráfico, sin generar problemas en la circulación.
  - [0,0,1,0]: rojo, situaciones de tráfico condensado con riesgos de interrupciones en la circulación.
  - [0,0,0,1]: negro, situaciones de atasco.

## 4.3. Normalización de imágenes y etiquetas

Tras crear las funciones de carga, se ha realizado la normalización sobre los datos correspondientes a cada uno de los futuros datasets. El objetivo de esta normalización es ajustar el conjunto de datos a una escala común sin distorsionar los rangos de los valores ni perder información. Los tipos de normalización aplicados son dos: normalización a color (RGB) y normalización en escala de grises.

Para realizar dicha normalización, se han obtenido las imágenes y etiquetas correspondientes a cada conjunto de situaciones por medio de las funciones de carga. Cabe destacar que los modelos de redes neuronales que se desarrollarán a lo largo del proyecto se basan en la biblioteca de Tensor Flow. Es por ello que se necesita crear un dataset de tensor flow, que permita un manejo sencillo y óptimo de los datos durante la normalización. El primer paso para ello es convertir tanto las imágenes como las etiquetas en *tensores*. *D*ichos *tensores* se han introducido como valores de entrada al método de creación del dataset. Los tipos de datos correspondientes a los tensores son: *float32* para las imágenes y *int32* para las etiquetas.

Todas las imágenes pertenecientes al dataset deben tener las mismas dimensiones antes de pasar a la fase de entrenamiento. Con motivo de este ajuste, se ha hecho una redimensión de cada una de las imágenes a tamaños de 200x200 píxeles. Además, se han convertido tanto las imágenes como las etiquetas en arrays de tipo *numpy*.

El proceso de normalización es siempre el mismo, dividimos las imágenes entre 255.0, para que los valores estén en un rango de entre 0 y 1. La diferencia principal es sobre qué imágenes se realiza dicha división, ya que para la normalización en escala de grises, antes de realizar dicha división, se actualizan las imágenes para que solo cuenten con un canal de color.

Una vez se ha realizado el proceso completo de normalización se muestran las imágenes para comprobar que no haya ningún error en los datos.

Cabe destacar que, a la hora de mostrar y guardar las imágenes a color, estas se muestran en formato BGR y no RGB. Por ello, los colores de las imágenes son alterados a la hora de mostrarse por pantalla. Esto no afecta a la calidad de la normalización ni al propio entrenamiento es un aspecto visual, que solventamos usando la biblioteca OpenCV para cambiar de BGR a RGB.

En las **Figuras 21 y 22** se puede ver un ejemplo de normalización a color y en escala de grises.

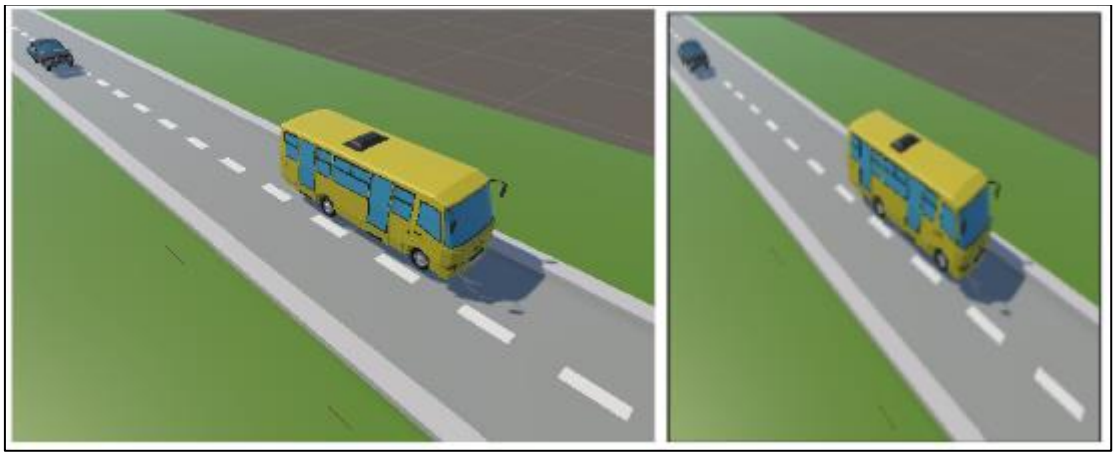


**Figura 21:** Proceso de redimensión y normalización en 3 Canales (RGB).

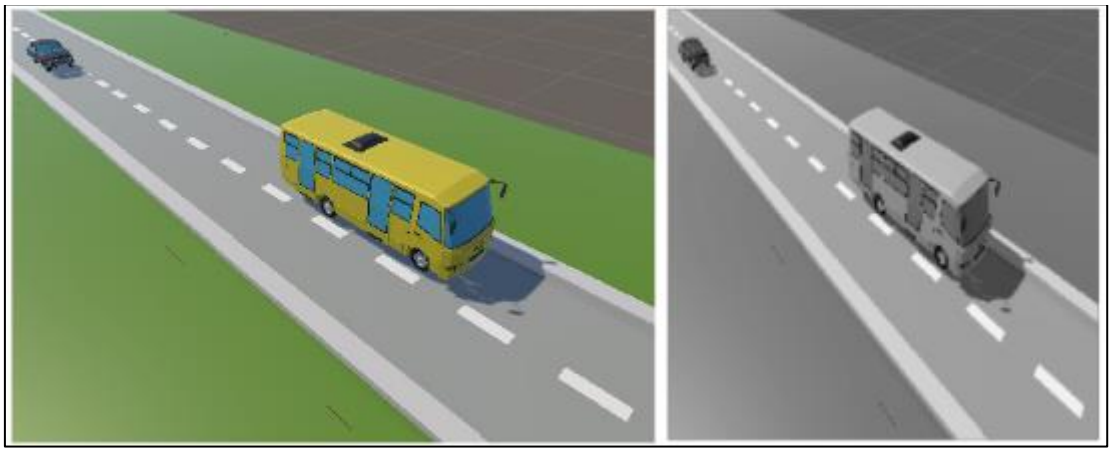


**Figura 22:** Proceso de redimensión y normalización en 1 Canal (Escala de grises).

Tras ejecutar para cada dataset los dos tipos de normalización y almacenar sus resultados, también guardaremos la información correspondiente a las imágenes y etiquetas sin normalizar, para posteriormente aplicar Batch Normalization.

## 4.4. Modelos de Redes Neuronales

Para el entrenamiento y validación de los datos se han creado siete modelos de redes neuronales: tres de ellos dedicados a imágenes en escala de grises, otros tres dedicados a imágenes RGB y un último modelo que cuenta con normalización de tipo Batch Normalization. Dependiendo de la arquitectura de estos modelos, podemos clasificarlos en 4 grupos:

**Modelos densos**: Contamos con dos modelos "densos" por cada dataset, dependiendo del tipo de normalización que se vaya a utilizar. Este modelo consta de 4 capas:

1. Capa de Aplanamiento: Esta capa toma imágenes 2D y las "aplana" en una array 1D, su función es transformar las imágenes a un formato correcto para las capas densas
2. Primera Capa Densa (Capa Oculta): esta primera capa oculta consta de 150 neuronas, su función de activación es ReLU (Rectified Linear Unit), es decir, si la entrada es negativa, devuelve 0 y si la entrada es positiva devuelve el propio número. La función de esta capa es aprender las características de los datos de entrada, específicamente, aprenderá un total de 150 características.
3. Segunda Capa Densa (Capa Oculta): Esta capa es igual que la primera capa densa, consta de 150 neuronas con una función de activación ReLU, en este caso las características que se aprenden en esta capa están basadas en las características obtenidas en la capa anterior.
4. Capa de salida: Esta capa densa consta de una sola neurona en el caso de que la clasificación sea binaria, es decir, que solo haya dos clases a predecir en el dataset. En el caso que que fuesen más de dos clases aumentaría el número de neuronas. Por ejemplo, en el caso de los datasets de clasificación de tipo de accidente y congestión de carreteras tendremos, para esta capa, 3 y 4 neuronas respectivamente. Esta última capa se encarga de determinar a qué clase pertenecen las imágenes.

```
model_dense = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape = (200,200,1)),
    tf.keras.layers.Dense(150, activation = 'relu'),
    tf.keras.layers.Dense(150, activation = 'relu'),
    tf.keras.layers.Dense(1, activation = 'sigmoid'),
])
```

**Modelos convolucionales CNN:** Contamos con dos modelos "CNN" por cada dataset, dependiendo del tipo de normalización que se vaya a utilizar. Este modelo consta de 9 capas:

1. Capas Convolucionales: Corresponden a la primera, tercera y quinta capa de la red neuronal y aplican 32, 64 y 128 filtros distintos respectivamente. Cuenta con matrices de 3x3, su función de activación es ReLU. Como ya se ha indicado anteriormente, estos filtros, que corresponden a "pesos" específicos que le asignaremos a dicha matriz para generar la convolución. La función principal de estas capas es detectar características, bordes y texturas en cada una de las imágenes. La entrada de la primera capa convolucional depende del número de canales de color que tengan las imágenes a procesar.
   El motivo sujeto al incremento del número de filtros por cada capa convolucional es que, inicialmente la primera capa convolucional detecta características simples como bordes, texturas, etc. Cada nueva capa debe detectar características más complejas y de alto nivel. Incrementar la cantidad de filtros a utilizar permite que la red neuronal aprenda una mayor cantidad de características sobre las imágenes.
2. Capas Max Pooling: Corresponden a la segunda, cuarta y sexta capa de la red neuronal. Estas capas generan matrices 2x2 que van recorriendo todos los píxeles de la imagen

desplazándose por medio de saltos llamados “Zancadas”. Sobre la posición en la que se encuentran, obtienen el píxel de mayor valor de la matriz y el resto los descartan, reduciendo el tamaño de las imágenes maximizando sus características principales.

3. Capa de Aplanamiento: Esta capa transforma el valor de entra proporcionado por su capa anterior a un array 1D. Prepara los datos para que puedan ser utilizados en las siguientes capas densas.
4. Primera Capa Densa: Esta capa consta de 100 neuronas, su función de activación es ReLU. Se encarga de aprender combinaciones no lineales correspondientes a las características detectadas en las capas de convolución.
5. Segunda Capa Densa (Capa de Salida): Esta última capa se encarga de determinar a qué clase pertenecen las imágenes.

```
model_CNN = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, (3,3), activation = 'relu', input_shape= (200,200,1)),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(64, (3,3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(100, activation = 'relu'),
        tf.keras.layers.Dense(1, activation = 'sigmoid')
])
```

**Modelos convolucionales CNN2:** Contamos con dos modelos “CNN2” por cada dataset, dependiendo del tipo de normalización que se vaya a utilizar. Este modelo consta de 13 capas:

1. Capas Convolucionales: Corresponden a la primera, cuarta, séptima capa en la red neuronal y aplican 32, 64 y 128 filtros distintos respectivamente. Cuenta con matrices de 3x3 y su función de activación es ReLU. La función principal de estas capas es detectar características, bordes y texturas en cada una de las imágenes.
2. Capas Max Pooling: Corresponden a la segunda, quinta y octava capa de la red neuronal. Estas capas generan matrices 2x2, reducen el tamaño de las imágenes y maximizan sus características principales.
3. Capas de Dropout: Corresponden a la tercera, sexta, novena y duodécima capa de la red neuronal. Aplican un dropout del 20%, 30%, 40% y 50% respectivamente. Estas capas se encargan de prevenir el sobreajuste, apagando aleatoriamente un porcentaje de las neuronas. El motivo de que se produzca un aumento del dropout en cada capa es debido a que el riesgo de sobreajuste es más pronunciado según se va profundizando en las capas, y por eso es necesario apagar un mayor porcentaje de neuronas.
4. Capa de Aplanamiento: Esta capa transforma el valor de entra proporcionado por su capa anterior a un array 1D. Prepara los datos para que puedan ser utilizados en las siguientes capas densas.
5. Primera Capa Densa: Esta capa consta de 250 neuronas. Su función de activación es ReLU. Se encarga de aprender combinaciones no lineales correspondientes a las características detectadas en las capas de convolución. Gracias al incremento de neuronas respecto al modelo anterior, de 100 a 250, esta red neuronal será más útil a la hora de aprender características complejas. Sin embargo, dependiendo de los datos de entrenamiento que aportemos, puede no resultar en una mejora significativa y ser más eficiente disminuir el número de neuronas, evitando así el riesgo de sobreajuste.
6. Segunda Capa Densa (Capa de Salida): Esta última capa se encarga de determinar a qué clase pertenecen las imágenes.

```
model_CNN2 = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(200,200,1)),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Dropout(0.4),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(250, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(1, activation='sigmoid')
])
```

**Modelos Convolucionales CNN3:** Contamos con un modelo "CNN3" por cada dataset. Este modelo consta de la misma arquitectura incremental que se ha planteado en los otros modelos convolucionales (CNN y CNN2). En este caso, se añade una nueva capa, posterior a cada una de las capas convolucionales, y también posterior a la primera capa densa. Esta capa se encarga de realizar la normalización (Batch Normalization). Este tipo de normalización se realiza en una capa de la red neuronal y ayuda a estabilizar el entrenamiento.

El proceso de Batch Normalization consiste en:

1. Calcular la media y la varianza sobre el Mini-Batch, el cual es una pequeña parte del conjunto de datos total, con el objetivo de mejorar la eficiencia computacional, aprovechando mejor el hardware de la GPU. Esto aumenta la estabilidad del entrenamiento y reduce los requisitos de memoria.

$$\mu B = 1/m\sum xi \qquad \textit{(media)}$$
$$(\sigma B)^2 = 1/m\sum m(xi - \mu B)^2 \qquad \textit{(varianza)}$$

2. Se realiza la normalización de las activaciones utilizando la varianza y la media que se han calculado.

$$x\hat{}i = xi \; - \; \mu B/raiz((\sigma B)^2 \; + \epsilon \,) \qquad \textit{(normalización)}$$

3. Se aplica Escalado y Desplazamiento para recuperar cualquier transformación lineal eliminada durante el proceso de normalización.

$$yi = \gamma x\hat{}i + \beta \qquad \textit{(escalado y desplazamiento)}$$

Para realizar el proceso de Batch Normalization, por cada uno de los dataset se han guardado las imágenes y etiquetas sin normalizar, con el objetivo de que dicha normalización se realice en las capas del modelo que se muestra a continuación.

```
model_CNN3 = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32, (3,3), input_shape=(200,200,1)),
        BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        tf.keras.layers.MaxPooling2D(2,2),
```

```
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv2D(64, (3,3)),
BatchNormalization(),
tf.keras.layers.Activation('relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Dropout(0.3),
tf.keras.layers.Conv2D(128, (3,3)),
BatchNormalization(),
tf.keras.layers.Activation('relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Dropout(0.4),
tf.keras.layers.Flatten(),tf.keras.layers.Dense(250, activation='relu'),
BatchNormalization(),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(1, activation='sigmoid')])
```

## 4.5. Compilación y ejecución de modelos

Una vez generados los modelos de redes neuronales, se realiza su compilación y posterior ejecución.

A la hora de compilar cada uno de los modelos debemos especificar ciertos valores de entrada que influyen directamente en el entrenamiento. Estos valores corresponden al optimizador, función de pérdida y métricas a utilizar.

El optimizador se encarga de actualizar los pesos del modelo realizado basándose en el gradiente de error frente a los dichos pesos. En el proyecto se ha implementado Adam como tipo de optimizador (Adaptative Moment Estimation), que maneja tasas de aprendizaje adaptativas, es decir, a diferencia de otros optimizadores, que utilizan tasas de aprendizaje constante, Adam ajusta individualmente la tasa de aprendizaje para cada uno de los parámetros del modelo.

La función de pérdida, como ya se ha explicado anteriormente, calcula el error en cada predicción respecto al valor real de la etiqueta. En este proyecto hemos utilizado dos funciones de pérdida:

- Binary Cross Entropy: función de pérdida utilizada principalmente para problemas de clasificación binaria, es decir, entre 0 y 1. Este enfoque es perfecto para datasets como el de detección de accidentes. Por otra parte, también se ha utilizado para abordar problemas de clasificación multi-etiqueta, como es el caso del dataset de detección de tipo de accidente. Los problemas multi-etiqueta son aquellos en los que cada imagen puede pertenecer a más de una clase, en nuestro caso, puede darse accidentes de colisión y vuelco de manera simultánea. Al utilizar Binary Cross Entropy como función de pérdida, cada etiqueta será procesada de manera independiente, tratando el problema como una clasificación binaria.
- Categorical Cross Entropy: función de pérdida utilizada principalmente en problemas de clasificación multi-clase, es decir, aquellos problemas en los que hay más de dos posibles clases para las etiquetas y en el que estas son excluyentes. Por tanto, una imagen no puede pertenecer a más de una clase a la vez. Este tipo de función de pérdida se ha utilizado para la compilación del dataset de congestión de carreteras, el cual consta de 4 clases excluyentes.

Las métricas son utilizadas para evaluar el rendimiento de la red neuronal durante el proceso de entrenamiento y evaluación, aportando información útil para el posterior análisis del modelo. Permite visualizar el porcentaje de acierto a la hora de realizar las predicciones (*Accuracy*).

```
model_dense.compile(optimizer = 'adam',loss= 'binary_crossentropy', metrics=['accuracy'])
```

Para ejecutar cada una de las redes neuronales se llama a la función fit correspondiente al modelo en cuestión. En dicha función, hemos especificado como valores de entrada el conjunto de las imágenes y etiquetas. Además, hemos establecido un Batch size o tamaño de lote, que indica durante el entrenamiento, el número de imágenes que se van a procesar antes de que se actualicen los pesos. Respecto a los datos a analizar, se han dividido en dos grupos, dividiendo la información entre entrenamiento y validación. Por lo general se ha utilizado un 85% para el entrenamiento y un 15% para la validación. Por último, se ha establecido un conjunto de épocas, que indican el número de veces que se va a ejecutar todos los lotes hasta recorrer por completo los datos.

Una vez explicados los parámetros de entrada y su finalidad, procedemos al análisis de los resultados tanto del entrenamiento como de la validación.

### ■ 4.5.1 Ejecución de Modelos Densos:

Se han analizado los resultados del entrenamiento y validación correspondientes a cada uno de los modelos densos, diferenciando cada ejecución según su tipo de normalización y dataset utilizado.

Detección de accidentes de tráfico:

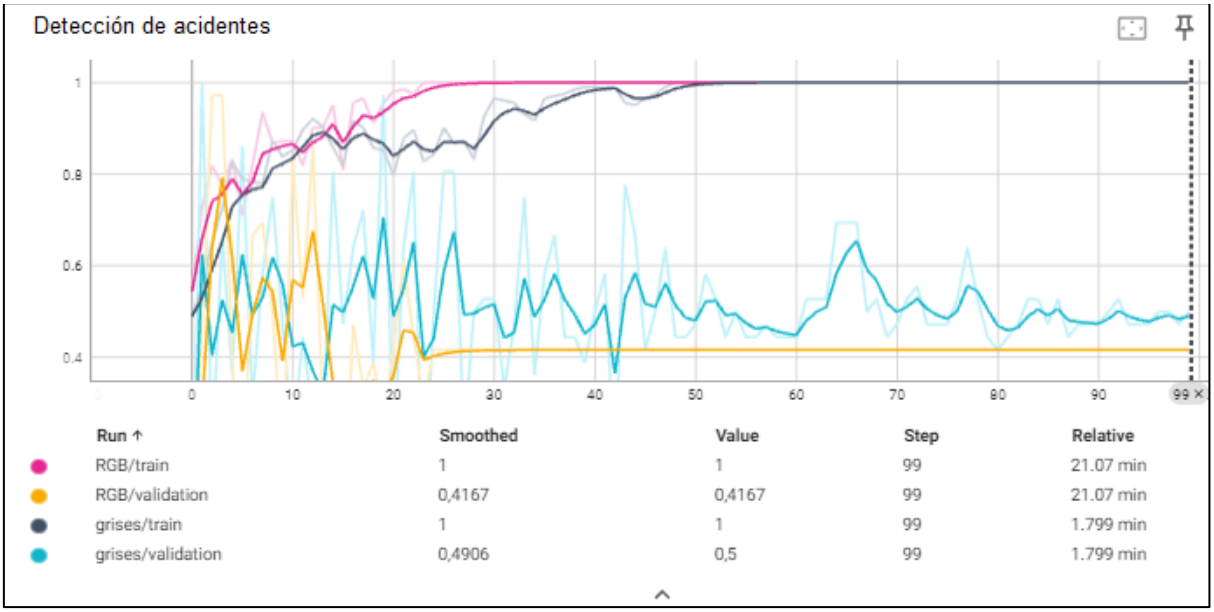
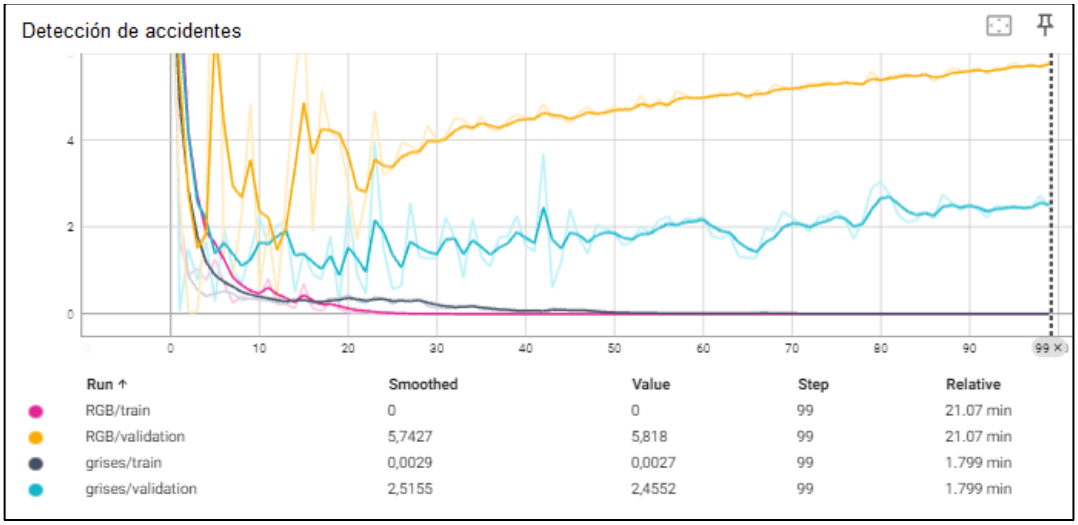



**Figura 23:** Gráfica evolutiva sobre la validación y entrenamiento en modelos densos de detección de accidentes.

Como se observa en la **Figura 23**, para este dataset se alcanza un nivel de acierto en predicciones sobre los datos de entrenamiento del 100%, a partir de la época 50 en escala de grises y de la época 25 en imágenes a color (RGB). Sin embargo, la validación no aumenta acorde a los valores del entrenamiento. La validación del modelo muestra una fluctuación de los parámetros, sin llegar a apreciarse una correlación entre dichos resultados frente a los del entrenamiento, lo que indica que se está generando sobre entrenamiento. Esto significa que la red neuronal se está aprendiendo los datos de entrenamiento por completo, generando predicciones del 100% sobre estos datos. Pero, al aprenderse los parámetros, a la hora de la validación las predicciones son bastante bajas y no mejoran con el paso de las épocas.

Por otra parte, podemos analizar el rendimiento de cada uno de los entrenamiento según los canales de color aplicados. En el caso de imágenes en escala de grises, el tiempo de entrenamiento y validación es de 1 minuto, mientras que en RGB el tiempo de entrenamiento y validación es de 21. El motivo de que una normalización a color tarde mucho más, radica en el número de píxeles a procesar. Para la escala de grises, inicialmente, hay una primera capa de aplanamiento a la que le llega una entrada de 200 x 200 x 1 = 40.000 píxeles. Tras esta capa, se encuentra la primera capa densa que consta de 150 neuronas, por tanto, se procesa un total de 40.000 x 150, lo que da un total de 6.000.000 píxeles. En el caso de la normalización RGB, a la capa de aplanamiento le llega una entrada de 200 x 200 x 3 = 120.000 píxeles. Si multiplicamos estos píxeles por las 150 neuronas de la primera capa densa, nos da un total de 18.000.000 píxeles a procesar, una cantidad significativamente más grande que en la escala de grises.

Además, las predicciones de validación generan resultados más altos en escala de grises, por lo que podemos entender que para este modelo es mucho más eficiente aplicar normalización con un solo canal de color.




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| RGB/train | 0 | 0 | 99 | 21.07 min |
| RGB/validation | 5,7427 | 5,818 | 99 | 21.07 min |
| grises/train | 0,0029 | 0,0027 | 99 | 1.799 min |
| grises/validation | 2,5155 | 2,4552 | 99 | 1.799 min |

**Figura 24:** Gráfica evolutiva sobre el error generado en Modelos Densos de detección de accidentes.

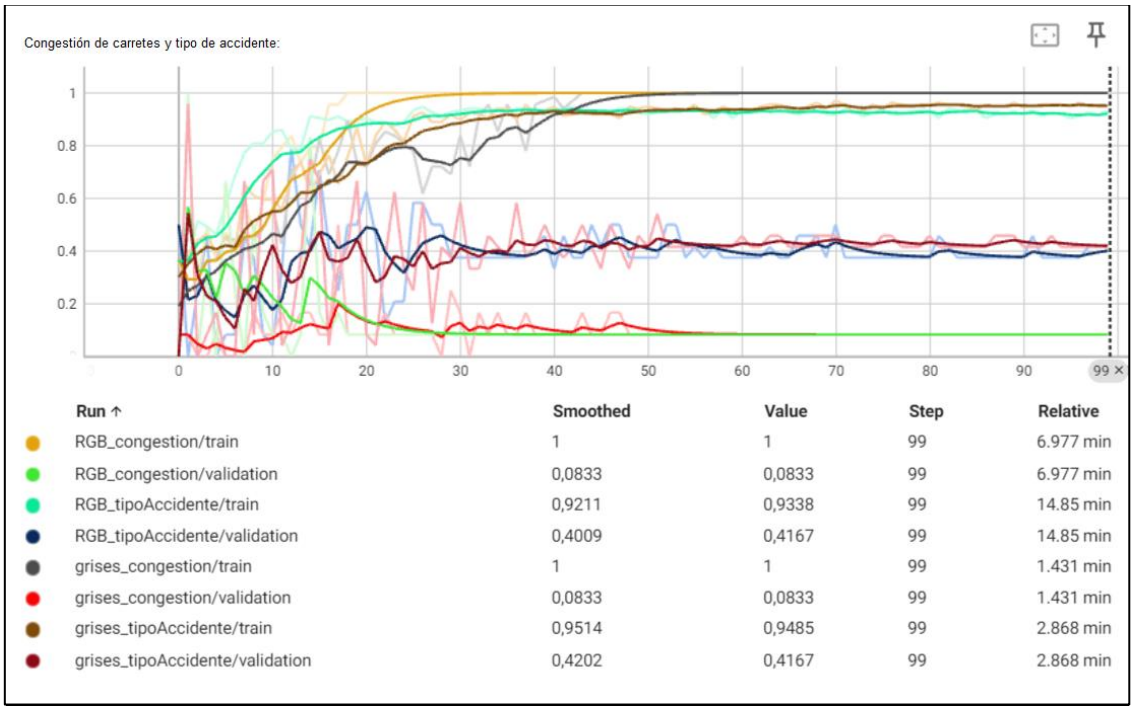
En **Figura 24**, se aprecia perfectamente como el número de errores disminuye constantemente para el entrenamiento. Sin embargo, en las etapas de validación, no sólo no disminuye, sino que, para imágenes a color, aumenta el número de errores según van pasando las épocas.

Principalmente, los problemas enunciados para este modelo radican en el tamaño del dataset y en el conjunto de capas que conforman dicha red neuronal.

Por parte del dataset, las redes neuronales suelen trabajar con conjuntos de datos muy grandes. De esta forma, se garantiza que se detecten correctamente los patrones y que tanto el entrenamiento como la validación vayan mejorando con el paso de las épocas. Sin embargo, nuestro conjunto de datos no es muy grande, por lo que se acaba aprendiendo de memoria los datos a la hora de entrenar.

Por otra parte, esta red neuronal está conformada únicamente por capas densas y de aplanamiento. No consta de capas convolucionales, técnicas de dropout ni batch normalization que podrían mejorar sus resultados.

Tipos de accidentes y congestión de carreteras:




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| RGB_congestion/train | 1 | 1 | 99 | 6.977 min |
| RGB_congestion/validation | 0,0833 | 0,0833 | 99 | 6.977 min |
| RGB_tipoAccidente/train | 0,9211 | 0,9338 | 99 | 14.85 min |
| RGB_tipoAccidente/validation | 0,4009 | 0,4167 | 99 | 14.85 min |
| grises_congestion/train | 1 | 1 | 99 | 1.431 min |
| grises_congestion/validation | 0,0833 | 0,0833 | 99 | 1.431 min |
| grises_tipoAccidente/train | 0,9514 | 0,9485 | 99 | 2.868 min |
| grises_tipoAccidente/validation | 0,4202 | 0,4167 | 99 | 2.868 min |

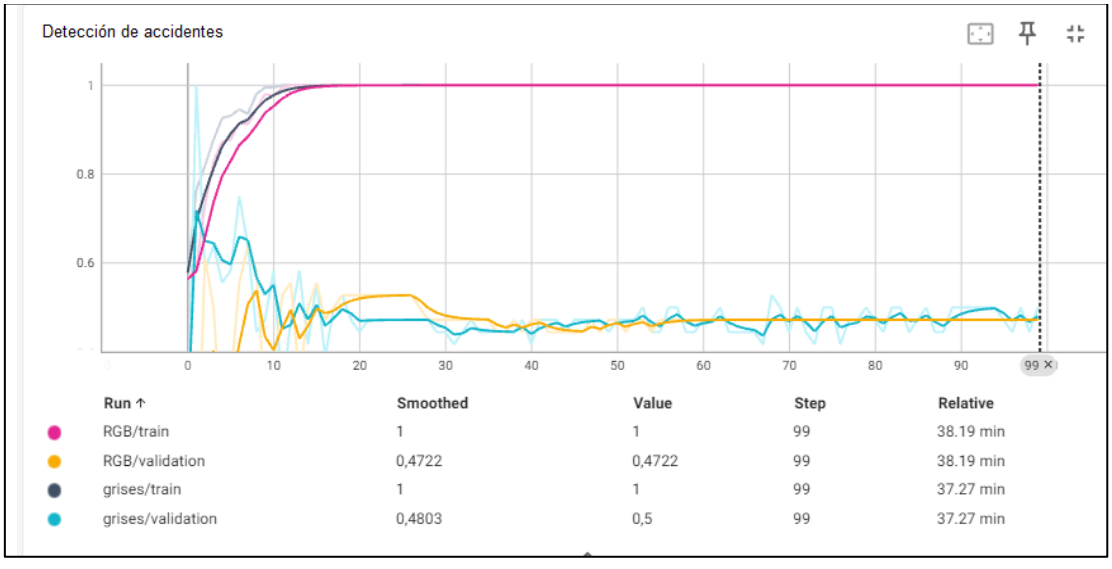
**Figura 25:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos Densos de detección de tipos accidentes y Congestión de carretera.

Como se observa en la **Figura 25**, para los otros dos datasets se han detectado los mismos problemas, derivados de la propia estructura de la red y de una cantidad todavía menor de datos para el entrenamiento y validación, siendo aún más críticos los valores correspondientes a la congestión en carretera, dado que el dataset contiene el menor número de imágenes y el mayor número de clases, con un total de 4.

### ■ 4.5.2. Ejecución de Modelos CNN:

Se han analizado los resultados del entrenamiento y validación correspondientes a cada uno de los modelos CNN, diferenciando cada ejecución según su tipo de normalización y dataset utilizado.

Detección de accidentes de tráfico:




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| RGB/train | 1 | 1 | 99 | 38.19 min |
| RGB/validation | 0,4722 | 0,4722 | 99 | 38.19 min |
| grises/train | 1 | 1 | 99 | 37.27 min |
| grises/validation | 0,4803 | 0,5 | 99 | 37.27 min |

**Figura 26:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN de detección de accidentes.

Como podemos observar en la **Figura 26**, los resultados son muy parecidos a los observado en la red neuronal densa. Esto se debe a que las redes convolucionales son herramientas pensadas para manejar grandes cantidades de datos. Al ejecutar un conjunto de datos pequeño, la mejora de rendimiento relacionada con la capacidad de detección de patrones en imágenes no es lo suficientemente relevante como para variar notablemente los resultados respecto a la red densa. Además, podemos notar que el sobre entrenamiento de las imágenes se genera mucho antes que en las redes densas.

Por último, cabe destacar el tiempo de ejecución, que es significativamente distinto en la red densa que en la red CNN. Por un lado, en la red densa, los tiempos de ejecución son: 1 minuto para escala de grises y 21 minutos en RGB. En el caso de la red CNN los tiempos de ejecución son: 38 minutos para escala de grises y 39 minutos en RGB.

En este caso, los tiempos de ejecución de cada tipo de normalización son prácticamente los mismos. Esto se produce debido a que las capas convolucionales están diseñadas para procesar imágenes de manera más eficiente, independientemente del número de canales de color. Sin embargo, la complejidad computacional que se genera al crear dichas capas hace que, respecto al modelo denso, la ejecución sea más costosa a nivel de tiempo.

Este análisis permite comprender cómo para un conjunto de datos reducido, es más eficiente aplicar redes neuronales menos complejas, como es el caso del modelo denso.

Esto ocurre para cada uno de los datasets sin importar el tipo de normalización aplicada.

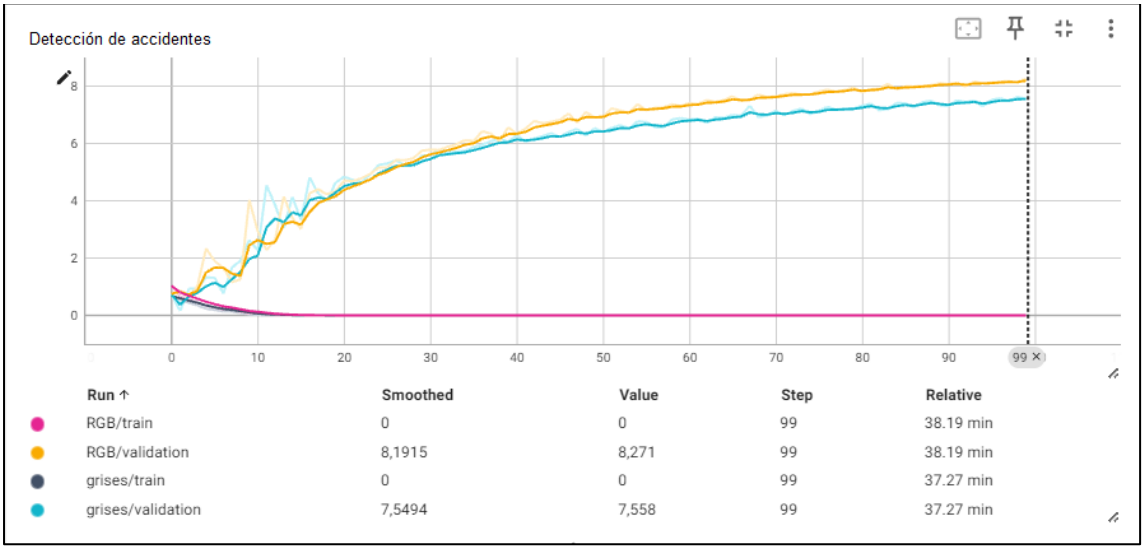
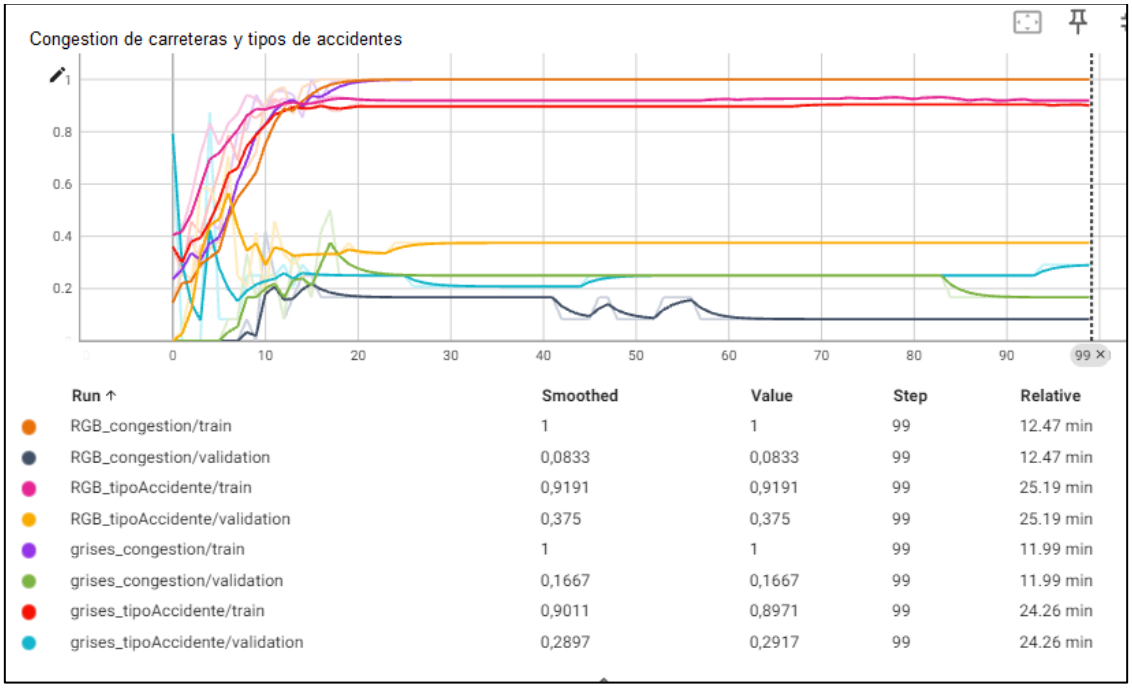



**Figura 27:** Gráfica evolutiva sobre el error generado en Modelos CNN de detección de accidentes.

Como se puede observar en la **Figura 27**, los errores de entrenamiento van bajando por cada época mientras que los errores de validación tienden a incrementarse hasta finalizar la ejecución.

Tipos de accidentes y congestión de carreteras:

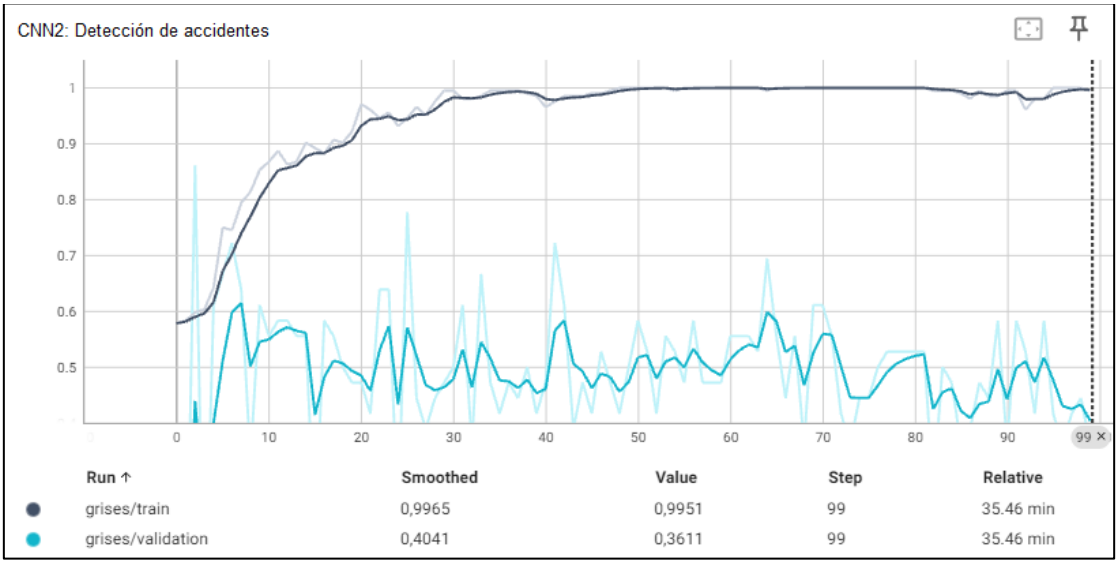



| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| RGB_congestion/train | 1 | 1 | 99 | 12.47 min |
| RGB_congestion/validation | 0,0833 | 0,0833 | 99 | 12.47 min |
| RGB_tipoAccidente/train | 0,9191 | 0,9191 | 99 | 25.19 min |
| RGB_tipoAccidente/validation | 0,375 | 0,375 | 99 | 25.19 min |
| grises_congestion/train | 1 | 1 | 99 | 11.99 min |
| grises_congestion/validation | 0,1667 | 0,1667 | 99 | 11.99 min |
| grises_tipoAccidente/train | 0,9011 | 0,8971 | 99 | 24.26 min |
| grises_tipoAccidente/validation | 0,2897 | 0,2917 | 99 | 24.26 min |

**Figura 28:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN de detección de tipos accidentes y Congestión de carretera.

En la **Figura 28** se muestra que, al igual que en el modelo denso, tanto para el dataset de congestión de carreteras como el de detección de tipo de accidentes generan resultados muy similares a los del dataset detección de accidentes. Presentan un claro sobre entrenamiento, prediciendo completamente el entrenamiento y fallando en gran medida en las predicciones de la validación.

### ■ 4.5.3. Ejecución de Modelos CNN2 y CNN3:




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| grises/train | 0,9965 | 0,9951 | 99 | 35.46 min |
| grises/validation | 0,4041 | 0,3611 | 99 | 35.46 min |

**Figura 29:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de accidentes.

La **Figura 29** nos muestra que al igual que en los modelos anteriormente descritos, en redes convolucionales con dropouts también se genera sobreajuste, esto se aprecia fácilmente viendo que las predicciones sobre el entrenamiento alcanzan valores cercanos al 100% y las predicciones de la validación son bajas y no aumentan con el paso de las épocas. Cabe destacar que las predicciones en validación respecto a los otros modelos son mayores, pero todavía insuficientes.

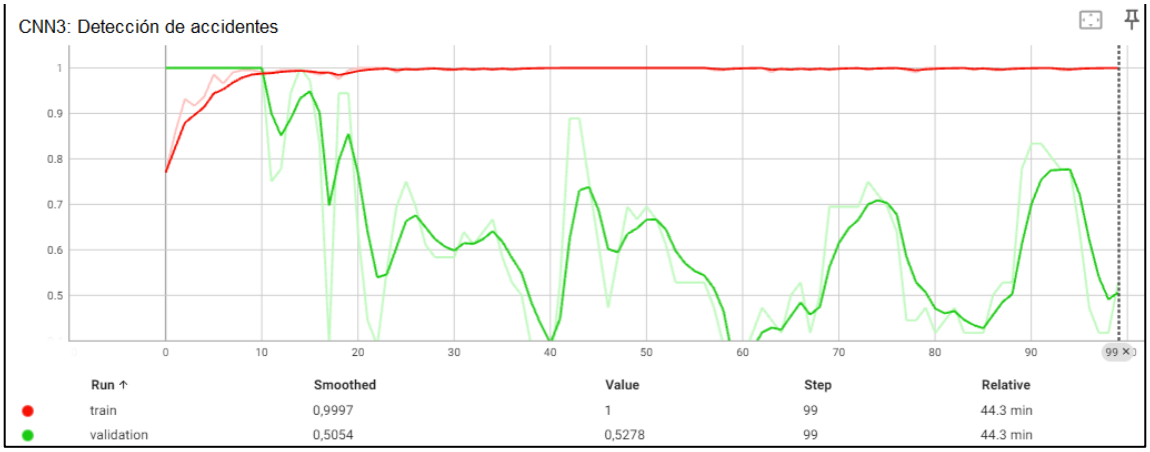



**Figura 30:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN3 de detección de accidentes.

Respecto al modelo CNN3, en la **Figura 30**, se aprecia un elevado nivel de predicción en el entrenamiento, tomando una tendencia creciente según pasan épocas. La predicción de la validación también es sumamente alta, llegando en numerosas ocasiones a predecir correctamente el 100% de los datos. Sin embargo, como en todos los modelos estudiados, llegado cierto número de épocas, el modelo acaba aprendiendo de memoria los parámetros y empieza a fallar en sus predicciones de validación. Una medida a aplicar para evitar dicho sobre entrenamiento sería reducir el número de épocas, evitando así que se aprenda los datos de entrada, pero reduciendo drásticamente el aprendizaje, como este modelo llega a predicciones sumamente elevadas en la validación, es muy probable que no suponga ningún inconveniente reducir dichas épocas

Como se ha podido apreciar, las medidas aplicadas en la arquitectura de los modelos no han acabado con el problema de sobreajuste. Es por ello que se ha optado por complementar dichas arquitecturas con técnicas como la validación cruzada, disminución de épocas y el aumento de datos, que permiten mejorar la eficiencia y nivel de predicción de los modelos generados, evitando que surja el sobreajuste.

## 4.6. Validación Cruzada

### 4.6.1. Definición y desarrollo

La validación cruzada es una técnica utilizada a la hora de evaluar modelos, en nuestro caso, modelos de redes neuronales. Gracias a esta técnica podemos garantizar la correcta generalización de los datos y evitar en gran medida el sobreajuste.

Para aplicar validación cruzada a nuestro entrenamiento, inicialmente debemos de dividir nuestros datos en subconjuntos. El modelo se ejecutará tantas veces como subconjuntos creemos. Además, por cada ejecución (fold), seleccionaremos uno de los subconjuntos como datos de validación y el resto los usaremos para entrenamiento.

Es importante tener en cuenta el número de subconjuntos a crear, ya que pocos subconjuntos generarían un conjunto de datos de entrenamiento muy pequeño y un conjunto de datos de validación demasiado grande. Por el contrario, dividir los datos en muchos subconjuntos, además de aumentar el tiempo de ejecución, también afectará negativamente a la validación, que contendrá muy pocos datos para esta.

Finalmente, los resultados obtenidos se promedian para obtener una estimación final de rendimiento.

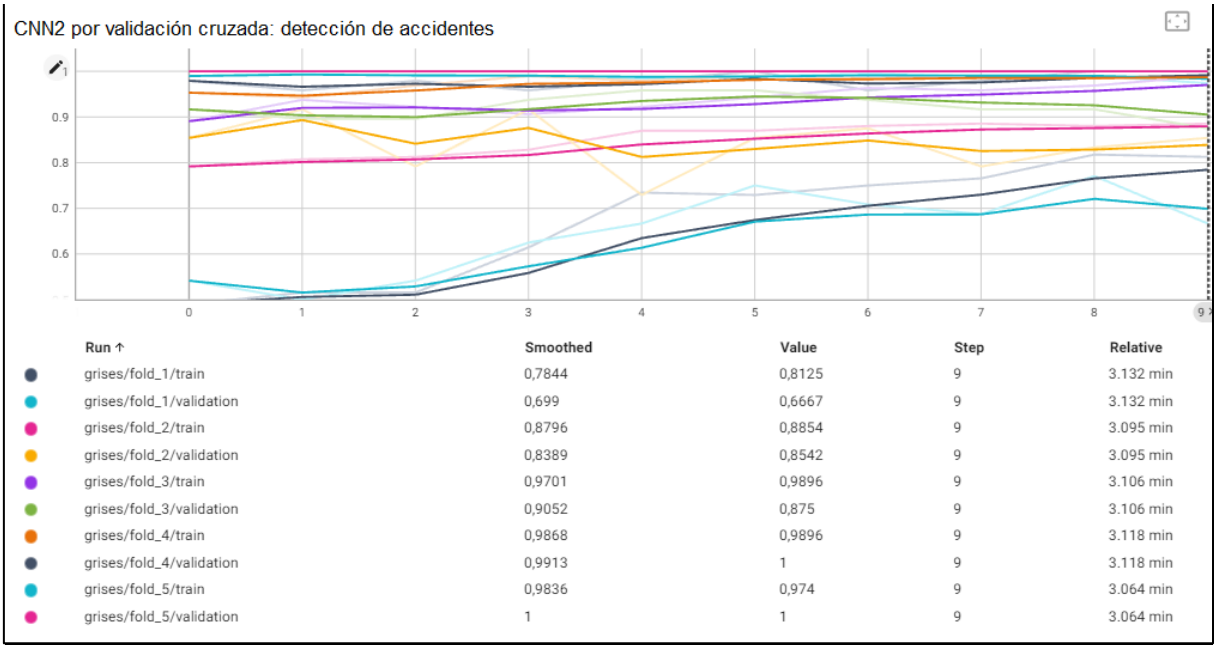
Respecto al problema del sobreajuste, se ha variado la composición de los conjuntos de entrenamiento y validación por cada fold y se ha disminuido el número de épocas a 10. Estas medidas buscan mitigar en gran medida este problema, principalmente, para aquellos modelos que ya cuentan con herramientas que buscan disminuir el sobreajuste, como es el caso de CNN2 y CNN3.

Respecto a las predicciones de validación, cada vez que se ejecuta el modelo sobre un fold, los pesos de la última época se utilizan como valores de inicio para la siguiente iteración. Esto garantiza que el entrenamiento no empiece siempre desde 0 y que cada vez que se ejecute el modelo, aumente tanto las predicciones de entrenamiento como las predicciones de validación.

Se ha probado esta herramienta de validación con cada uno de los dataset, por simplicidad, se ha tomado como punto de partida tanto el modelo CNN2 como el modelo CNN3, que son aquellos que cuentan con técnicas para evitar el sobreajuste.

### 4.6.2. Ejecución Modelos CNN2 y CNN3:

Detección de accidentes:




| Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| grises/fold_1/train | 0,7844 | 0,8125 | 9 | 3.132 min |
| grises/fold_1/validation | 0,699 | 0,6667 | 9 | 3.132 min |
| grises/fold_2/train | 0,8796 | 0,8854 | 9 | 3.095 min |
| grises/fold_2/validation | 0,8389 | 0,8542 | 9 | 3.095 min |
| grises/fold_3/train | 0,9701 | 0,9896 | 9 | 3.106 min |
| grises/fold_3/validation | 0,9052 | 0,875 | 9 | 3.106 min |
| grises/fold_4/train | 0,9868 | 0,9896 | 9 | 3.118 min |
| grises/fold_4/validation | 0,9913 | 1 | 9 | 3.118 min |
| grises/fold_5/train | 0,9836 | 0,974 | 9 | 3.064 min |
| grises/fold_5/validation | 1 | 1 | 9 | 3.064 min |

**Figura 31:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de accidentes aplicando Validación Cruzada.

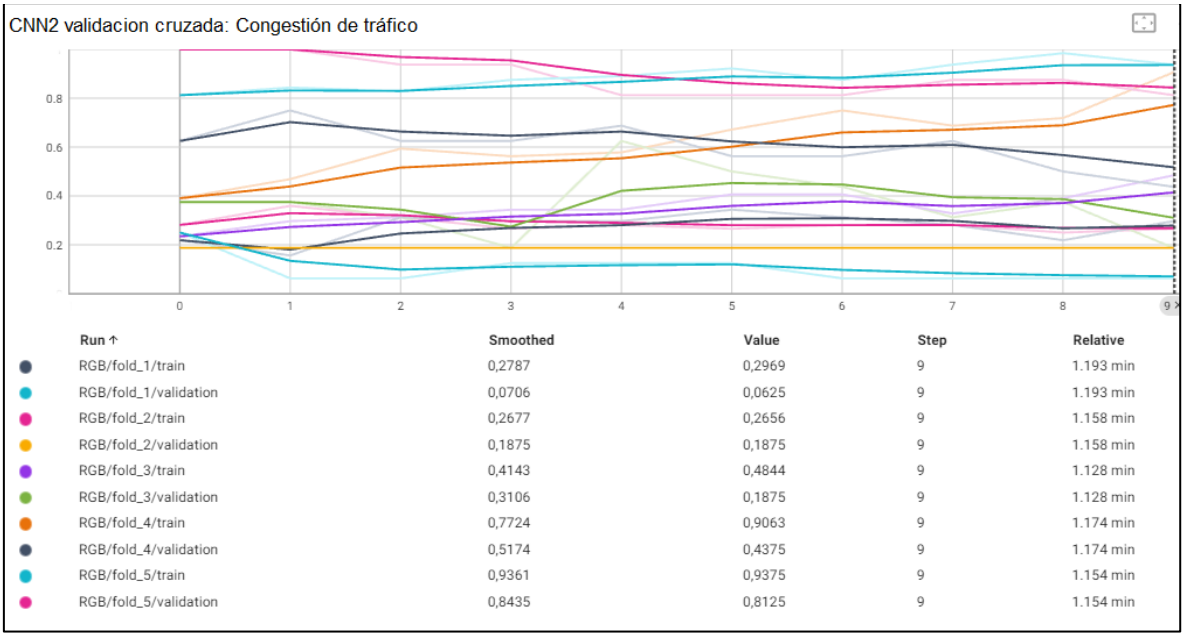
Se ha ejecutado el modelo CNN2 respecto al dataset de detección de accidentes, buscando las condiciones óptimas para generar un entrenamiento robusto en las predicciones de validación y que

no tienda al sobreajuste. Para ello, se ha implementado un total de 5 folds y únicamente 10 épocas, con el objetivo de mitigar por completo dicho sobreajuste.

Como se observa en la **Figura 31**, las predicciones de entrenamiento y validación toman valores similares respecto a la época y fold específico. Además, por cada incremento (fold) se observa un nivel de predicción aún más elevado, asegurando que el modelo clasifica correctamente las imágenes. Al analizar los últimos folds se ha detectado que las predicciones son prácticamente perfectas lo que es una clara evidencia de que se está produciendo sobreajuste.

Congestión de carreteras:

También se ha ejecutado dicho modelo aplicado al resto de datasets. Principalmente, el análisis se enfocó en el dataset de congestión en carreteras, ya que cuenta con el conjunto de datos más pequeño, además del mayor número de clases entre los datasets generados. Este modelo, es el más difícil de predecir, por lo tanto, para el mismo número de folds y épocas, debería de mostrar unos valores distintos al modelo de detección de accidentes.

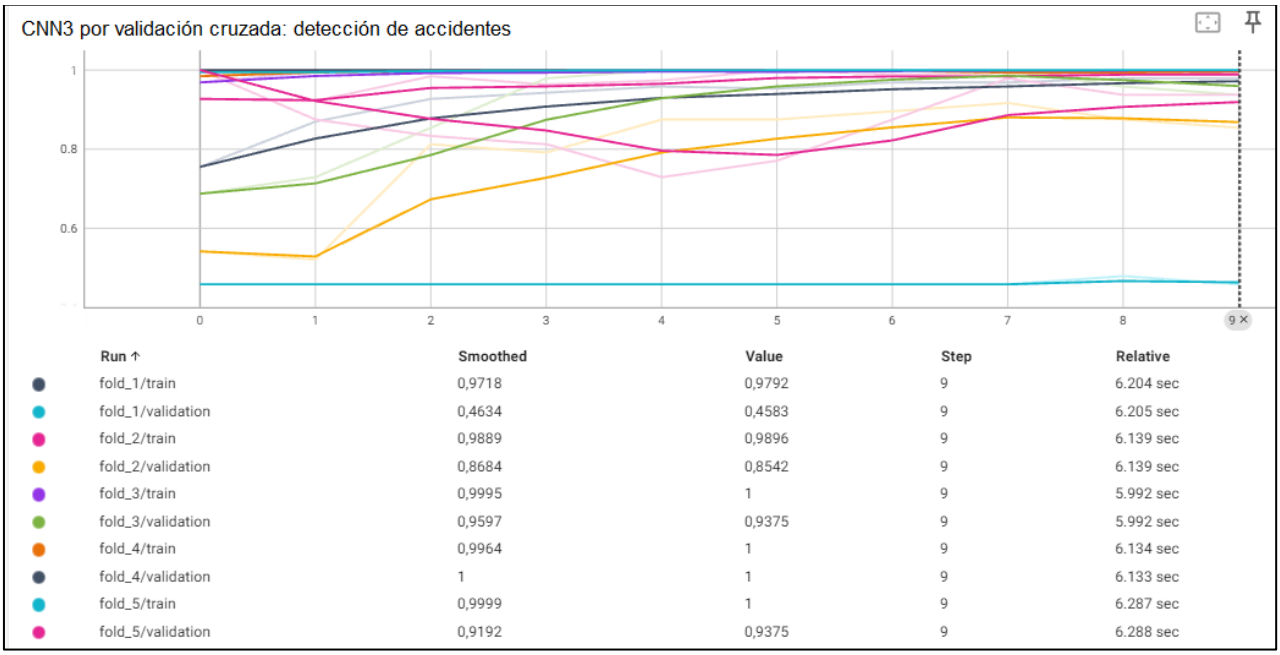



| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| RGB/fold_1/train | 0,2787 | 0,2969 | 9 | 1.193 min |
| RGB/fold_1/validation | 0,0706 | 0,0625 | 9 | 1.193 min |
| RGB/fold_2/train | 0,2677 | 0,2656 | 9 | 1.158 min |
| RGB/fold_2/validation | 0,1875 | 0,1875 | 9 | 1.158 min |
| RGB/fold_3/train | 0,4143 | 0,4844 | 9 | 1.128 min |
| RGB/fold_3/validation | 0,3106 | 0,1875 | 9 | 1.128 min |
| RGB/fold_4/train | 0,7724 | 0,9063 | 9 | 1.174 min |
| RGB/fold_4/validation | 0,5174 | 0,4375 | 9 | 1.174 min |
| RGB/fold_5/train | 0,9361 | 0,9375 | 9 | 1.154 min |
| RGB/fold_5/validation | 0,8435 | 0,8125 | 9 | 1.154 min |

**Figura 32:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de Congestión de Carreteras aplicando Validación Cruzada.

Como se observa en la **Figura 32**, para este dataset las predicciones no son tan buenas. Sin embargo, para el último fold, tanto las predicciones del entrenamiento como de la validación superan siempre el 80%.

Tras esta ejecución, se han realizado los entrenamientos del modelo CNN3 con validación cruzada, los resultados de dichas ejecuciones seguían mostrando sobreajuste y predicciones sobre la validación altas y acordes a las realizadas en el entrenamiento.




| Run ↑ | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| fold_1/train | 0,9718 | 0,9792 | 9 | 6.204 sec |
| fold_1/validation | 0,4634 | 0,4583 | 9 | 6.205 sec |
| fold_2/train | 0,9889 | 0,9896 | 9 | 6.139 sec |
| fold_2/validation | 0,8684 | 0,8542 | 9 | 6.139 sec |
| fold_3/train | 0,9995 | 1 | 9 | 5.992 sec |
| fold_3/validation | 0,9597 | 0,9375 | 9 | 5.992 sec |
| fold_4/train | 0,9964 | 1 | 9 | 6.134 sec |
| fold_4/validation | 1 | 1 | 9 | 6.133 sec |
| fold_5/train | 0,9999 | 1 | 9 | 6.287 sec |
| fold_5/validation | 0,9192 | 0,9375 | 9 | 6.288 sec |

**Figura 33:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN3 de detección de detección de accidentes aplicando Validación Cruzada.

Respecto a parámetros mostrados en la **Figura 33**, cabe destacar el sobreajuste, principalmente notorio en el primer incremento de la validación. La generalización aumenta en los incrementos 2, 3, 4 y 5, donde la precisión de validación es alta.

Respecto al estudio realizado sobre la Validación Cruzada, se concluye que esta técnica disminuye el sobreajuste a la vez que mejora el índice de predicción de la validación, ajustándose mucho más a los valores correspondientes a la predicción del entrenamiento. Sin embargo, no es suficiente para evitar el sobreajuste completamente. Una posible medida para acabar con este problema consistiría en reducir aún más las épocas de ejecución. Se ha considerado que llegado a este punto, dicha medida no sería óptima para el entrenamiento y por lo tanto se probará a aplicar otras medidas, como es el aumento de datos.

## 4.7. Aumento de datos

El aumento de datos es una técnica que permite aumentar la cantidad y forma de los datos. Es muy útil para conjuntos de datos pequeños, como es el caso de los datasets realizados en el proyecto. Permite mejorar la generalización del modelo, además de reducir el sobreajuste.

Para conseguir aumentar la información se han realizado transformaciones sobre las imágenes originales, entre dichas transformaciones encontramos rotaciones, traslaciones, escalado, etc. El objetivo es añadir dichas imágenes distorsionadas junto a sus etiquetas al conjunto de información original.

```
datagen = ImageDataGenerator(
        rotation_range=40,
        height_shift_range=0.2,
        width_shift_range=0.2,
        shear_range=15,
        zoom_range=[0.7, 1.4],
```

```
                horizontal_flip=True,
                vertical_flip=True
)
```

Como resultado, se han obtenido tanto las imágenes en escala de grises como las imágenes RGB distorsionadas, como se puede apreciar en la **Figura 34**.

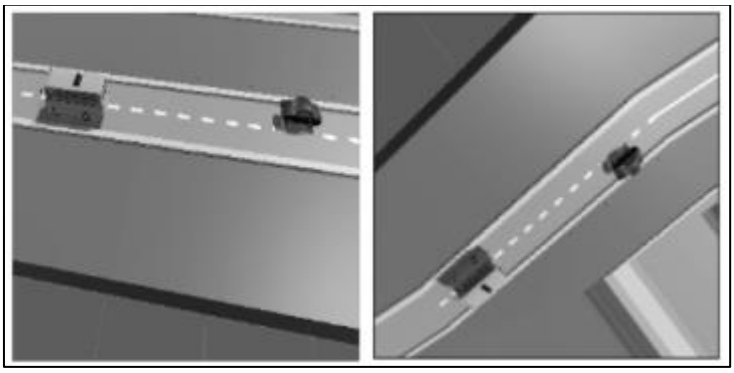


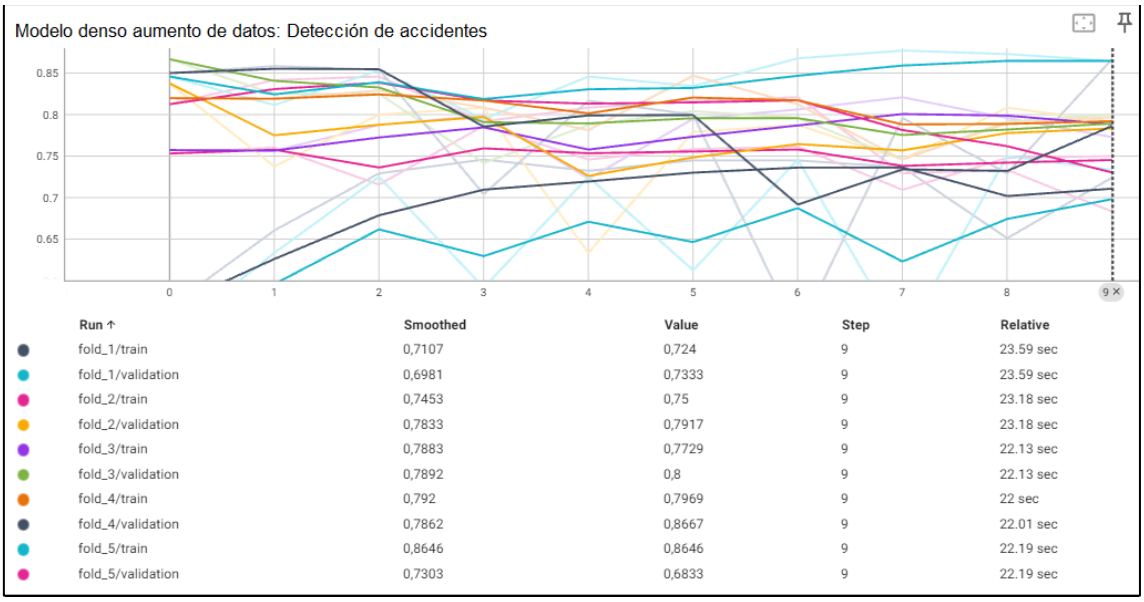
**Figura 34:** Distorsión de imagen para aumento de datos.

Por otro lado, realizar aumento de datos tiene sus desventajas frente al uso de los datasets originales. Al aumentar en gran medida la información, también va a aumentar notoriamente el tiempo de ejecución. Además, no todas las transformaciones son correctas, modificar las imágenes puede llevar a que dejen de tener sentido respecto al contexto de análisis, es por esta razón que es muy importante tener un conjunto de datos fuertemente ligado al contexto de clasificación.

Teniendo en cuenta toda la información obtenida hasta el momento por cada uno de los modelos, dependiendo de su dataset, normalización y técnicas para evitar el sobreajuste, se ha llegado a la conclusión que, llegado a este punto, tanto el modelo CNN2 como el modelo CNN3 manejan predicciones de entrenamiento y validación muy parecidas. Además, para dichos modelos, el tipo de normalización utilizado no va a modificar la calidad de las predicciones. Por ello, el análisis del aumento de datos se va a hacer sobre los modelos denso y CNN2 para cada uno de los datasets.

## 4.7.1 Ejecución de Detección de accidentes:

Modelo denso:



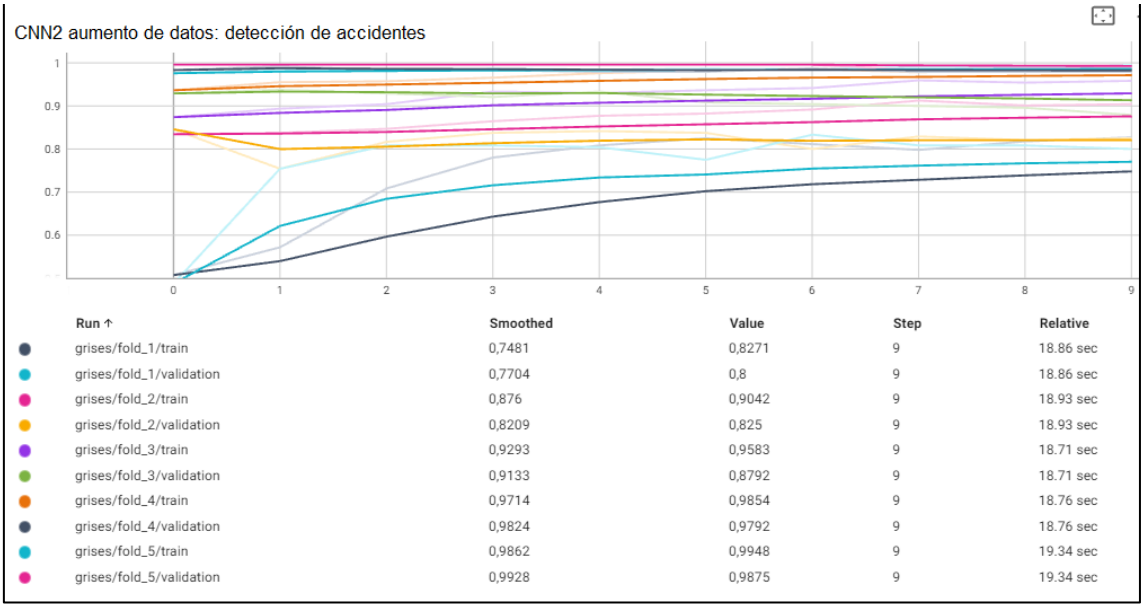

| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| fold_1/train | 0,7107 | 0,724 | 9 | 23.59 sec |
| fold_1/validation | 0,6981 | 0,7333 | 9 | 23.59 sec |
| fold_2/train | 0,7453 | 0,75 | 9 | 23.18 sec |
| fold_2/validation | 0,7833 | 0,7917 | 9 | 23.18 sec |
| fold_3/train | 0,7883 | 0,7729 | 9 | 22.13 sec |
| fold_3/validation | 0,7892 | 0,8 | 9 | 22.13 sec |
| fold_4/train | 0,792 | 0,7969 | 9 | 22 sec |
| fold_4/validation | 0,7862 | 0,8667 | 9 | 22.01 sec |
| fold_5/train | 0,8646 | 0,8646 | 9 | 22.19 sec |
| fold_5/validation | 0,7303 | 0,6833 | 9 | 22.19 sec |

**Figura 35:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos Densos de detección de accidentes aplicando Validación Cruzada.

Como se observa gracias a la **Figura 35**, para el modelo denso con normalización en escala de grises, la precisión de predicción en el entrenamiento es siempre ligeramente superior a la predicción de validación, llegando a alcanzar un total de 85%. En el último fold se ha detectado oscilaciones en las predicciones que son indicativas de un ligero sobreajuste. A nivel de la arquitectura del modelo, esto es algo lógico, ya que no cuenta con herramientas internas para solventar dicho sobreajuste.

Al ejecutar el modelo con una normalización a color, se ha percibido un claro incremento en los costes de ejecución. En este caso, el modelo requiere de una cantidad de recursos superior a la RAM del sistema. Esto causa que la ejecución para los últimos incrementos no llegue a finalizar. Dicho problema está directamente relacionado con el aumento de datos y con la optimización del modelo respecto a imágenes a color, como se ha explicado anteriormente en la memoria.

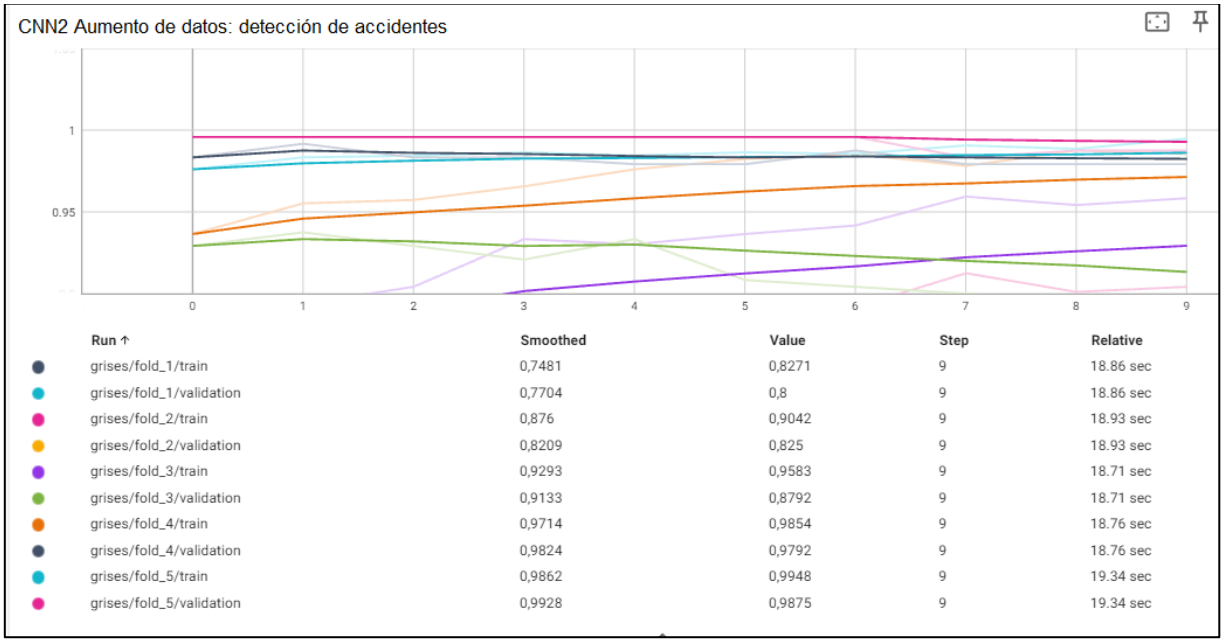
Modelo CNN2:




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| grises/fold_1/train | 0,7481 | 0,8271 | 9 | 18.86 sec |
| grises/fold_1/validation | 0,7704 | 0,8 | 9 | 18.86 sec |
| grises/fold_2/train | 0,876 | 0,9042 | 9 | 18.93 sec |
| grises/fold_2/validation | 0,8209 | 0,825 | 9 | 18.93 sec |
| grises/fold_3/train | 0,9293 | 0,9583 | 9 | 18.71 sec |
| grises/fold_3/validation | 0,9133 | 0,8792 | 9 | 18.71 sec |
| grises/fold_4/train | 0,9714 | 0,9854 | 9 | 18.76 sec |
| grises/fold_4/validation | 0,9824 | 0,9792 | 9 | 18.76 sec |
| grises/fold_5/train | 0,9862 | 0,9948 | 9 | 19.34 sec |
| grises/fold_5/validation | 0,9928 | 0,9875 | 9 | 19.34 sec |

**Figura 36:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de accidentes aplicando Validación Cruzada y Aumento de datos.

La **Figura 36** muestra las predicciones de entrenamiento y validación para el dataset de detección de accidentes con aumento de datos. Se ha apreciado un elevado nivel de predicción en los últimos incrementos, sin llegar al 100% en ninguno de los casos. El modelo no solo predice con una excelente precisión, además, es la primera ejecución que no genera sobreajuste tanto en su normalización en escala de grises como en RGB.




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| grises/fold_1/train | 0,7481 | 0,8271 | 9 | 18.86 sec |
| grises/fold_1/validation | 0,7704 | 0,8 | 9 | 18.86 sec |
| grises/fold_2/train | 0,876 | 0,9042 | 9 | 18.93 sec |
| grises/fold_2/validation | 0,8209 | 0,825 | 9 | 18.93 sec |
| grises/fold_3/train | 0,9293 | 0,9583 | 9 | 18.71 sec |
| grises/fold_3/validation | 0,9133 | 0,8792 | 9 | 18.71 sec |
| grises/fold_4/train | 0,9714 | 0,9854 | 9 | 18.76 sec |
| grises/fold_4/validation | 0,9824 | 0,9792 | 9 | 18.76 sec |
| grises/fold_5/train | 0,9862 | 0,9948 | 9 | 19.34 sec |
| grises/fold_5/validation | 0,9928 | 0,9875 | 9 | 19.34 sec |

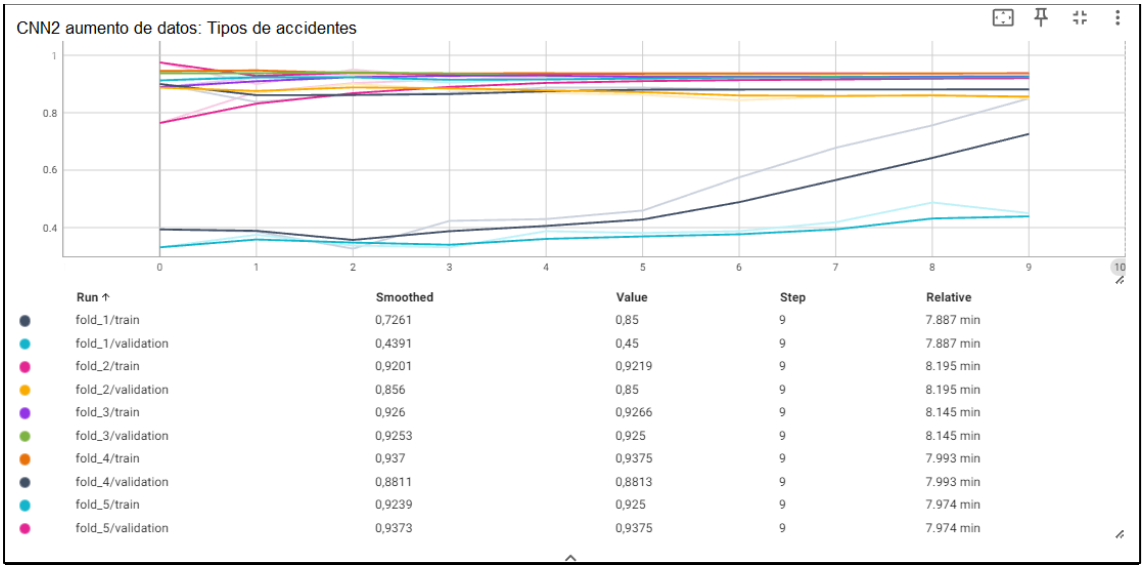
**Figura 37:** Ampliación Gráfica figura 34.

La **Figura 37** muestra cómo efectivamente las predicciones nunca llegan al 100% pero se acercan extremadamente a dicho valor.

### 4.7.2. Ejecución de Detección de Tipo de accidentes y Congestión de Carretera:

Dado que se ha detectado sobreajuste en el modelo denso con el dataset de detección de accidentes, no hay razones para pensar que será diferente con estos dos nuevos datasets. Los resultados obtenidos no solo presentarán el problema del sobreajuste, sino que probablemente lo incrementarán aún más.

Modelos CNN2:

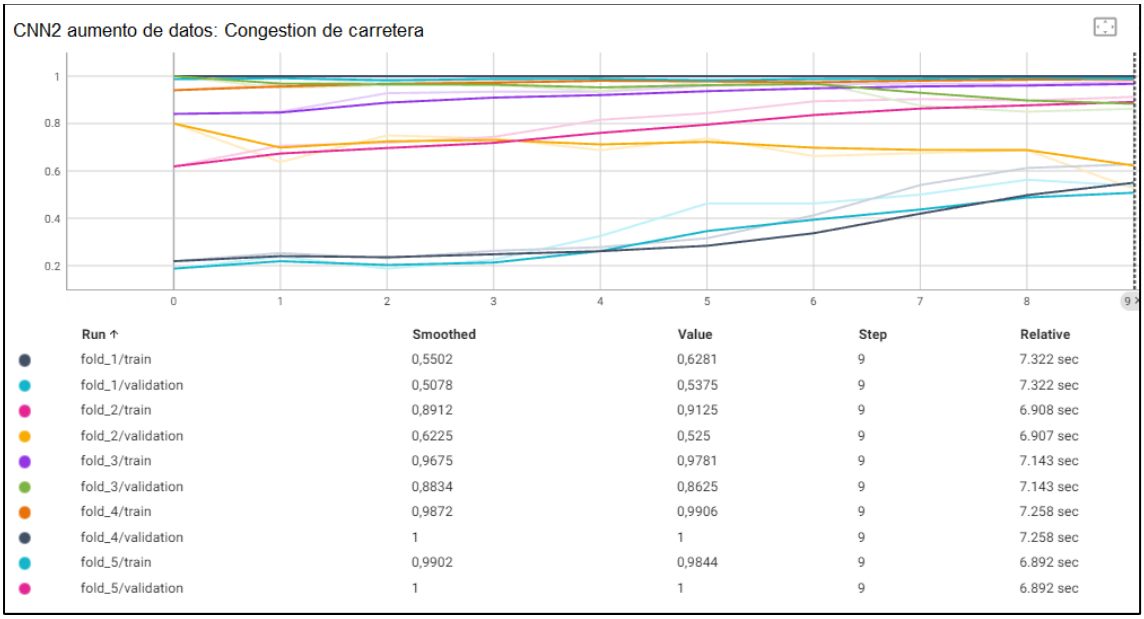



| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| fold_1/train | 0,7261 | 0,85 | 9 | 7.887 min |
| fold_1/validation | 0,4391 | 0,45 | 9 | 7.887 min |
| fold_2/train | 0,9201 | 0,9219 | 9 | 8.195 min |
| fold_2/validation | 0,856 | 0,85 | 9 | 8.195 min |
| fold_3/train | 0,926 | 0,9266 | 9 | 8.145 min |
| fold_3/validation | 0,9253 | 0,925 | 9 | 8.145 min |
| fold_4/train | 0,937 | 0,9375 | 9 | 7.993 min |
| fold_4/validation | 0,8811 | 0,8813 | 9 | 7.993 min |
| fold_5/train | 0,9239 | 0,925 | 9 | 7.974 min |
| fold_5/validation | 0,9373 | 0,9375 | 9 | 7.974 min |

**Figura 38:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de Tipos de accidentes aplicando Validación Cruzada y Aumento de datos.

Como se observa en la **Figura 38**, en los primeros incrementos se muestran indicios de sobreajuste, especialmente al comparar el porcentaje de predicción en el entrenamiento respecto al de validación. Además, en estos primeros incrementos, el porcentaje de predicción no es muy alto. En los

incrementos 3 y 4, el índice de predicción ha aumentado considerablemente, aunque sigue siendo inferior a los valores correspondientes al dataset de detección de accidentes. Es importante destacar que el problema del sobreajuste se ha eliminado prácticamente por completo en estos incrementos, ya que la pérdida de entrenamiento y validación son muy similares.




| Run | Smoothed | Value | Step | Relative |
|---|---|---|---|---|
| fold_1/train | 0,5502 | 0,6281 | 9 | 7.322 sec |
| fold_1/validation | 0,5078 | 0,5375 | 9 | 7.322 sec |
| fold_2/train | 0,8912 | 0,9125 | 9 | 6.908 sec |
| fold_2/validation | 0,6225 | 0,525 | 9 | 6.907 sec |
| fold_3/train | 0,9675 | 0,9781 | 9 | 7.143 sec |
| fold_3/validation | 0,8834 | 0,8625 | 9 | 7.143 sec |
| fold_4/train | 0,9872 | 0,9906 | 9 | 7.258 sec |
| fold_4/validation | 1 | 1 | 9 | 7.258 sec |
| fold_5/train | 0,9902 | 0,9844 | 9 | 6.892 sec |
| fold_5/validation | 1 | 1 | 9 | 6.892 sec |

**Figura 39:** Gráfica evolutiva sobre el entrenamiento y validación en Modelos CNN2 de detección de Congestión de carreteras aplicando Validación Cruzada y Aumento de datos.

Para el dataset de congestión de carretera, tras el aumento de datos para los últimos dos incrementos, se sigue apreciando un sobreajuste, como se puede observar en la **Figura 39**. Este sobreajuste es lógico teniendo en cuenta el número de clases e imágenes originales, tras realizar aumento de datos pasamos de tener 80 imágenes a tener 400. Sin embargo, esta cantidad sigue siendo insuficiente para un correcto entrenamiento.

### 4.7.3. Conclusión del aumento de datos:

Esta técnica es sumamente útil para evitar el sobreajuste, siempre y cuando se tenga un dataset base lo suficientemente grande, en el caso tanto del dataset de detección de accidentes como el de tipos de accidentes. Gracias al aumento de datos, se ha evitado por completo el sobreajuste, sin embargo, en el dataset de congestión de carreteras no se ha podido mitigar por completo. Si se hiciese un aumento de dato aún mayor, acabaríamos con dicho sobreajuste completamente.

El aumento de datos también ha proporcionado predicciones de entrenamiento y validación sumamente altas para los incrementos finales, especialmente en los modelos convolucionales, diseñados para manejar grandes cantidades de datos. Tras este aumento de información, se ha observado que las predicciones de los modelos convolucionales han resultado significativamente mejores que en los modelos densos. Al contrario que con la información original, en las que los dos modelos toman valores muy parecidos. El aumento de datos ha mostrado la superioridad de los modelos convolucionales frente a los densos en datasets más grandes.

## 4.8. YOLO

YOLO (You Only Look Once) es un algoritmo que permite la detección de objetos tanto en imágenes como en videos a tiempo real, dando predicciones sumamente precisas y rápidas. A nivel de arquitectura YOLO está marcado por el uso de capas convolucionales, intercaladas con capas de agrupamiento (Max Pooling) para reducir las dimensiones de las imágenes al igual que en los modelos realizados (CNN, CNN2 y CNN3). Además, YOLO divide las imágenes en un conjunto de celdas, cada una de estas se encargará de detectar los objetos que se encuentren dentro de ellas (21).

Para predecir los objetos que se encuentran encerrados en cada una de las celdas se utilizan los llamados *cuadros delimitadores* o *Bounding boxes*. En cada celda hay un número fijo de dichos cuadros. Cada uno de estos cuenta con un centro marcado por los ejes *(x)* e *(y)*, un ancho del cuadro delimitador *(w)*, una altura del cuadro delimitador *(h)* y por último, la probabilidad de que en un cuadro específico se encuentre un objeto *(p)*. Gracias a estos cuadros, YOLO es capaz de predecir más de una clase dentro de las imágenes, analizando los objetos que encierran cada una de las cuadrículas (21).

YOLO obtiene resultados sumamente precisos tras una única ejecución del entrenamiento, al contrario que en los modelos desarrollados, los cuales requieren de validación cruzada junto a aumento de datos para obtener altos porcentajes en el índice de predicción.

A la hora de trabajar con YOLO, se han creado un conjunto de direcciones donde se guardarán tanto las imágenes como las etiquetas a analizar. Respecto a las etiquetas, se ha tenido que modificar las actuales para que se adecuen al formato estipulado por YOLO. Es por ello que se ha creado la función *generar_etiquetas_yolo()*. Esta función se encarga de crear para cada archivo de etiquetas un cuadro delimitador que cubre el 50% de la foto.

Se han especificado estos parámetros para los cuadros delimitadores:

```
x_center = 0.5
y_center = 0.5
width = 0.5
height = 0.5
etiqueta_yolo = f"{etiqueta} {x_center} {y_center} {width} {height}"
```

Una vez hemos modificado las etiquetas, tenemos que dividir el dataset en conjunto de datos de entrenamiento y conjunto de datos de validación, y mandarlo tanto las imágenes como las etiquetas a su carpeta correspondiente en drive dependiendo de al conjunto que pertenezcan.

Finalmente, procedemos al entrenamiento. Para ello, YOLO saca la información del entrenamiento y validación de las carpetas de drive creadas anteriormente para comprobar la efectividad del modelo se ha introducido un conjunto de datos pequeño, que consta únicamente de 80 imágenes. A la hora de analizar los resultados debemos tener en cuenta ciertas métricas que permiten entender los resultados del entrenamiento:

- Precisión: Mide el acierto sobre las predicciones positivas. Es decir, la proporción de verdaderos positivos entre todas las predicciones positivas se han realizado.
- Recall: Mide la capacidad que tiene el modelo para encontrar todos los verdaderos positivos. Es decir, la proporción de verdaderos positivos entre todos los verdaderos positivos.
- mAP50: Promedio de precisión para un umbral de IoU (Intersection over Union) de 0.50
- mAP50-95: es el promedio de precisión que se da para múltiples umbrales de IoU (desde 0.50 hasta 0.95, incrementando en 0.05).
- Box Loss (box_loss): se encarga de medir la precisión de los cuadros delimitadores, mostrando el porcentaje de error que generan.

- Object Loss: se encarga de medir la precisión del modelo a la hora de predecir si hay o no un objeto en cada cuadro clasificador.
- Class Loss: se encarga de mostrar el porcentaje de error en la clasificación del entrenamiento.

Para ejecutar el entrenamiento, se han introducido 100 épocas, un batch size de 16 y se han redimensionado las imágenes a tamaños de 200x200.

```
!python train.py --img 200 --batch 16 --epochs 100 --data escenas_tfg.yaml --cfg yolov5s.yaml --weights yolov5s.pt
```

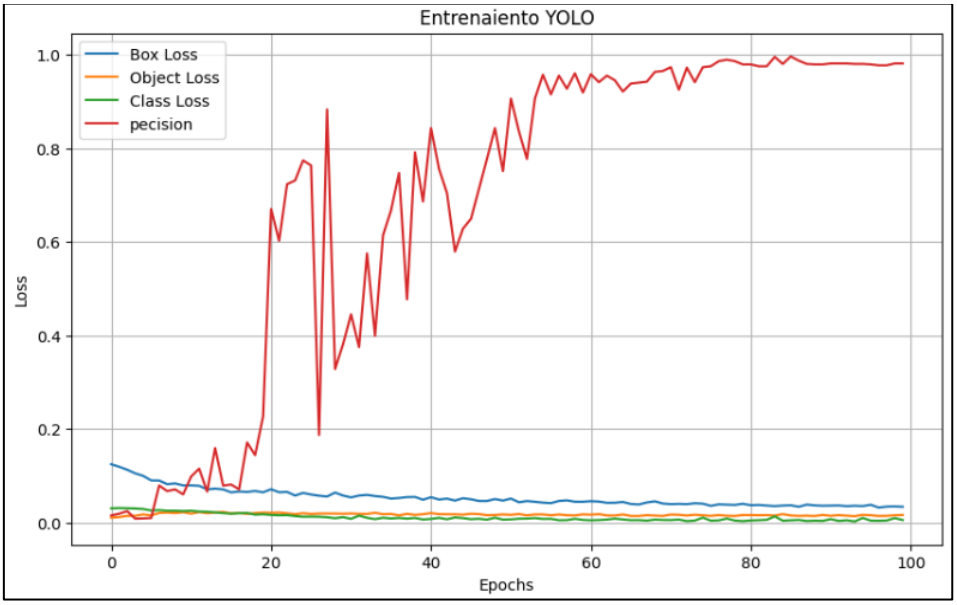



**Figura 40:** Gráfica evolutiva sobre el entrenamiento en YOLO.

Como se observa en la **Figura 40**, el entrenamiento de YOLO cuenta con una mejora progresiva sobre todas las métricas según van pasando las épocas, mostrando una disminución constante en el box_loss, obj_loss y class_loss, lo que indica que el modelo está mejorando las predicciones de los cuadros delimitadores. Por otra parte, la gráfica muestra cómo a lo largo de las épocas la precisión aumenta significativamente. Esto se traduce en una mejora en la capacidad de clasificar correctamente los objetos de las imágenes

Una vez finalizado el entrenamiento, se ha realizado la validación del modelo.

```
!python val.py --weights {model_path} --data '{yaml_path}' --img 224 --batch 16
```

Los resultados muestran validaciones sumamente altas, indicando que el modelo ha entrenado correctamente. La alta precisión y recall del modelo indican fiabilidad y robustez a la hora de realizar las clasificaciones.

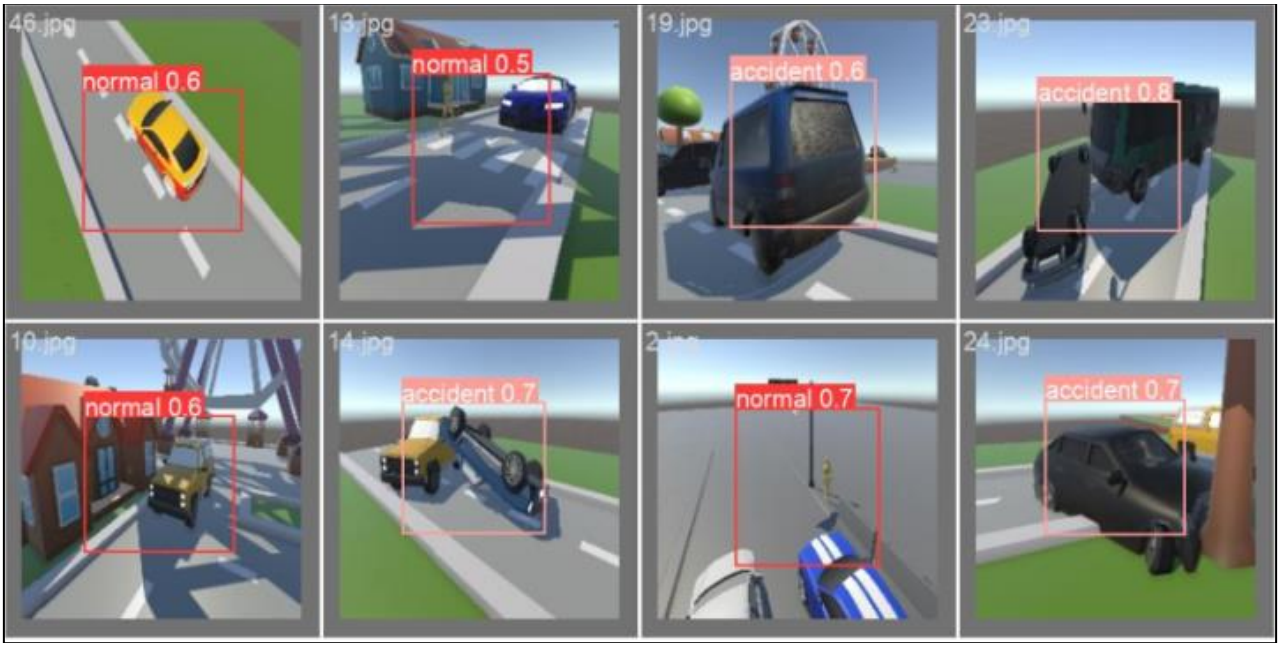



**Figura 41:** Detección de clases en imágenes con cuadros delimitadores

La **Figura 41** muestra varias predicciones sobre las imágenes generadas, junto a su cuadro delimitador.

Se ha concluido que a la hora de aplicar los modelos de redes neuronales, los resultados a nivel de tiempo y precisión siempre van a ser mejores en YOLO debido a su arquitectura.

# 5

# Conclusiones

La mortalidad en carretera es un problema de trascendencia global, que ha experimentado un crecimiento considerable, sobre todo en países menos favorecidos económicamente. Para disminuir estas cifras, la UE lleva años implementando medidas de control y saneamiento de carreteras, así como un marco normativo adecuado para los países de la Unión. Estas han resultado efectivas, a pesar de que las pérdidas humanas siguen siendo considerables. Con el avance de la tecnología, diferentes organismos y profesionales en la materia ven en la Inteligencia Artificial un posible aliado para este propósito.

Para poder detectar y clasificar los accidentes de tráfico fue necesario generar un conjunto de imágenes que sirvieran como entrenamiento para la red neuronal. Concretamente, se elaboraron tres datasets: El primero de ellos, permite discernir entre la presencia o ausencia de accidentes de tráfico; el segundo, garantiza la identificación entre los diferentes tipos de accidentes (colisión, vuelco o atropello) y el tercer y último dataset, muestra a la red neuronal a identificar el nivel de congestión vehicular de las carreteras.

Una vez creado el conjunto de datos se inició la etapa de entrenamiento. Para ello, se codificaron los siete modelos de redes neuronales a utilizar, tres para imágenes a color, tres para imágenes en escala de grises y un último modelo para Batch Normalization. El entorno seleccionado para su elaboración fue Google Colab, una plataforma gratuita que nos permite ejecutar código Python. Se

plantearon las diferentes funciones de carga para los datasets y se normalizaron los datos según el número de canales de color.

Tras la compilación y ejecución de los diferentes modelos, se procedió a realizar un análisis comparativo de los diferentes resultados.

Para los modelos neuronales densos, independientemente del dataset utilizado, se observa un menor tiempo de entrenamiento y validación debido a su arquitectura simple. También se detecta un mayor índice de aciertos en la validación en escala de grises, alcanzando valores de 0,6 y oscilando hasta 0,4. En el caso de la validación a color, los valores se establecían en 0,4 sin generar oscilaciones. A nivel de rendimiento, se observa un mayor tiempo de ejecución con datos a color. Esto se debe a que la simpleza de la arquitectura del modelo denso, que le da ventaja en datasets pequeños, no está optimizada para tratar los datos de la manera más eficiente.

Para los modelos neuronales CNN, se observan resultados similares a los obtenidos en el modelo anterior, aunque ligeramente peores, oscilando entre 0,4 y 0,5. El motivo radica en que las redes convolucionales son herramientas pensadas para manejar grandes cantidades de datos. Al ejecutar un conjunto de datos pequeño, la mejora de rendimiento obtenida no es lo suficientemente relevante como para variar notablemente los resultados con respecto a una red densa. Cabe destacar que el tiempo de ejecución es considerablemente mayor respecto al modelo denso. Por lo tanto, para datasets pequeños no es eficiente aplicar un modelo convolucional (CNN).

A nivel de rendimiento, los tiempos de ejecución dependiendo del tipo de normalización son prácticamente iguales. Esto se debe a que las capas convolucionales están pensadas para tratar de manera más eficiente los datos. Respecto a las predicciones, los errores de entrenamiento disminuyen por cada época, mientras que los errores de validación tienden a incrementarse hasta finalizar la ejecución.

Estos dos modelos, además de no predecir correctamente los datos de validación, constan de un grave problema de sobreajuste. El modelo se aprende de memoria los datos de entrenamiento y no valida correctamente, para evitar dicho problema se generan los modelos CNN2 y CNN3

En los modelos neuronales CNN2 y CNN3, aun tomando medidas como el dopout (apagado de neuronas) o Batch Normalization (Normalización en capa), siguen generando sobreajuste, al igual que en los dos modelos anteriores. Esto es un claro indicativo de lo arraigado que estaba este problema en nuestros datos, principalmente por la cantidad de estos, a nivel de validación los resultados siguen siendo muy parecidos a los mostrados en los modelos anteriores.

Dado que a nivel arquitectónico no se ha conseguido una modificación de los modelos que mejore la validación y evite el sobreajuste, se han tomado medidas externas al modelo.

Como primera medida se ha implementado Validación Cruzada. Esta técnica permite una mayor variedad de datos tanto en el entrenamiento como en la validación debido a que por cada incremento, se modifica el contenido de los conjuntos de entrenamiento y validación. Esto no solo evita que se genere sobreajuste, sino que también incrementa el nivel de predicción en la validación.

Tras ejecutar la validación cruzada, mejoró notablemente las predicciones de validación. Sin embargo, seguía dándose sobreajuste. Esto indicaba que las medidas implantadas no eran suficientes para evitar dicho problema. Dado que la causa principal recae en el tamaño del dataset se implantó aumento de datos.

Para aplicar el aumento de datos, utilizamos las imágenes originales de cada uno de los datasets y realizamos modificaciones sobre estas, tanto para normalización a color como para la normalización en escala de grises. Tras el aumento de datos se puede apreciar que, para aquellos datasets con más cantidad de imágenes, al aplicar modelos CNN2 y CNN3 se conseguía acabar

con el sobreajuste. Para el dataset de congestión de carreteras, las medidas aplicadas eran insuficientes, lo que indica que deberá hacerse un aumento de datos porcentualmente mayor al de los otros datasets. A nivel predictivo, se generan porcentajes sumamente altos tanto en el entrenamiento como en la validación llegando a cifras cercanas al 1 pero sin generar sobreajuste.

Finalmente se ha realizado un estudio con la red neuronal YOLO en el que se especifica su funcionamiento y arquitectura. Posteriormente, se han comprado sus resultados aplicados a los datasets generados. Para dicha red neuronal se ha introducido un número de imágenes que para los modelos realizados supondría un problema, tanto de porcentaje de validación como de sobreajuste. El modelo de YOLO realiza únicamente un entrenamiento al contrario que nuestros modelos al aplicar validación cruzada, por lo tanto, los resultados deben de compararse con los propios modelos sin aumento de datos ni validación cruzada.

Al realizar el entrenamiento y posteriormente la validación se muestra que no se ha generado sobreajuste y que las predicciones tanto en entrenamiento como en validación se acercan extremadamente a 1. Por lo que se concluye que esta arquitectura es la idónea a la hora de utilizar conjuntos de datos no muy grandes.

# 6

# Bibliografía


(1) European Commission. (2024). La Comisión anuncia nuevas políticas de tráfico. Recuperado de https://ec.europa.eu/commission/presscorner/detail/es/ip_24_1361

(2) Publications Office of the European Union. (2020). *Publication detail*. Recuperado de https://op.europa.eu/en/publication-detail/-/publication/d7ee4b58-4bc5-11ea-8aa5-01aa75ed71a1

(3) World Health Organization. (s.f.). *Death on the roads*. Recuperado de https://extranet.who.int/roadsafety/death-on-the-roads/?lang=es#ticker/car_passengers

(4) Dirección General de Tráfico. (2023). *1.145 personas fallecieron en siniestros de tráfico en carretera durante 2023*. Recuperado de https://www.dgt.es/comunicacion/notas-de-prensa/1.145-personas-fallecieron-en-siniestros-de-trafico-en-carretera-durante-2023/#:~:text=Por%20tipo%20de%20v%C3%ADa%3A%20de,disminuido%20respecto%20al%20a%C3%B1o%20anterior.

(5) Universidad Nacional de Colombia. (s.f.). *Repositorio institucional*. Recuperado de https://repositorio.unal.edu.co/handle/unal/12244

(6) Universidad Complutense de Madrid. (s.f.). *Repositorio institucional*. Recuperado de https://docta.ucm.es/entities/publication/e25148b8-0afa-4ffe-8ca7-1e4964621c1f

(7) Universidad de A Coruña. (2013). *RGP 21 artículo 3*. Recuperado de https://ruc.udc.es/dspace/bitstream/handle/2183/12604/RGP_21_2013_art_3.pdf?sequence=1&isAllowed=y

(8) Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Recuperado de https://books.google.es/books?hl=es&lr=&id=_spC6S7UfZgC&oi=fnd&pg=PP1&dq=inteligencia+artificial+modelos+tecnicas&ots=sRftQGTttS&sig=e7PeUSmD7Qu1hmqset0laUdUhoc

(9) Sociedad de Análisis y Computación de Chile. (2014). *Regresión 2*. Recuperado de https://www.sachile.cl/upfiles/revistas/54e63943b5d69_14_regresion-2-2014_edit.pdf

(10) Ofelia y Orquesta. (s.f.). *Redes Neuronales 001*. Recuperado de https://ofeliayorquesta.com/articulos/Redes_Neuronales_001.pdf

(11) Universidad del País Vasco. (s.f.). *Curso de Redes Neuronales*. Recuperado de https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwj9l_vt7s6GAxV5Q6QEHdbEAH0QFnoECA8QAQ&url=https%3A%2F%2Focw.ehu.eus%2Fpluginfile.php%2F40137%2Fmod_resource%2Fcontent%2F1%2Fredes_neuro%2Fcontenidos%2Fpdf%2Flibro-del-curso.pdf&usg=AOvVaw3l5NtpQPyZj3T1clzCEMSq&opi=89978449

(12) Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Recuperado de https://books.google.es/books?hl=es&lr=&id=X0uLwi1Ap4QC&oi=fnd&pg=PA11&dq

=redes+neuronales&ots=gPIEeomq0h&sig=NjRjSdk0iyuiipbpKX0bxs9DTk4#v=onepage&q=redes%20neuronales&f=false

(13) Journal of Applied Research and Technology. (2020). *Paper on neural networks*. Recuperado de https://www.scielo.org.mx/scielo.php?pid=S1405-55462020000401589&script=sci_arttext

(14) Unknown author. (2019). *Skin Lesion Segmentation using SegNet*. Recuperado de https://d1wqtxts1xzle7.cloudfront.net/61319526/paper20191124-21798-1159tha-libre.pdf?1574621527=&response-content-disposition=inline%3B+filename%3DSkin_Lesion_Segmentation_using_SegNet_wi.pdf&Expires=1718034816&Signature=N8GsWQjGd7yL1Znz2xrBdWr5Fc76Zpvoy3py2fyNGTrpKCy23qBIFOIsdny5KPbitWovy6smd53J8n6CE2F04wqv2ugFlOrJq20XhFL8-S-WIPQLFNFi84txIaTmQ84OTdjhc0iITZVuDNAq3yvlKadXTJr7yOWjOrSbMniYfWjUZNi-1aJsrwbKqyuPRHZFkp8iKbxSkDxPQZ5HRb5qWub5VvAiLt9qlddU~6zAk3-mqQLMyiayzb6a1tlpk6Cego8BPszaYGA9ZRCBgDdait4bfJXvCMsX4IJn9eWl-U-cRKJ4XE1oEmcpK0hjl-sjBiR0XfGsFy6hOeD7ixWbOQ__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

(15) Artola, D. (2019). *TFG sobre redes neuronales*. Universidad de Sevilla. Recuperado de https://idus.us.es/bitstream/handle/11441/89506/TFG-2402-ARTOLA.pdf?sequence=1&isAllowed=y

(16) Massiris, M. (2018). *Detección de equipos de protección personal con YOLO*. Recuperado de https://ruc.udc.es/dspace/bitstream/handle/2183/24891/2018_Massiris_Manlio_Detecci%C3%B3n-equipos-protecci%C3%B3n-personal-red-neuronal-convolucional-YOLO.pdf?sequence=3&isAllowed=y

(17) Universitat Politècnica de València. (2016). *Artículo sobre MSEL*. Recuperado de https://polipapers.upv.es/index.php/MSEL/article/view/4524/4724

(18) Universidad Nacional de La Plata. (2020). *Documento completo*. Recuperado de https://sedici.unlp.edu.ar/bitstream/handle/10915/102579/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

(19) Bonilla Carrión, C. (2021). *TFG sobre detección de patrones*. Universidad de Sevilla. Recuperado de https://idus.us.es/bitstream/handle/11441/115221/TFG%20DGMyE%20Bonilla%20Carri%C3%B3n%2c%20Carmelo.pdf?sequence=1&isAllowed=y

(20) Silvestre Llopis, J. (2019). *TFM sobre inteligencia artificial*. Universitat de València. Recuperado de https://www.uv.es/lapeva/Thesis/TFM_2019_Jordi_Silvestre_Llopis.pdf

(21) ScienceDirect. (2022). *Artículo de ScienceDirect*. Recuperado de https://www.sciencedirect.com/science/article/pii/S1877050922001363

(22) Comisión Económica para América Latina y el Caribe (CEPAL). (2020). *Publicación de CEPAL*. Recuperado de https://repositorio.cepal.org/entities/publication/d3bff0bc-1c64-49fa-b5d2-b898b84c4be6