Universidad Europea

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN

ANÁLISIS DE DATOS MASIVOS (BIG DATA)

TRABAJO FIN DE MÁSTER

ESTIMACIÓN DEL TRÁFICO RODADO A PARTIR DEL ANÁLISIS VISUAL DE CÁMARAS URBANAS

NOMBRE:

IVÁN SAN MARTÍN FERNÁNDEZ

CURSO 2022-2023

TÍTULO: ESTIMACIÓN DEL TRÁFICO RODADO A PARTIR DEL ANÁLISISIS VISUAL DE CÁMARAS URBANAS

AUTOR: IVÁN SAN MARTÍN FERNÁNDEZ

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANÁLISIS DE DATOS MASIVOS (BIG DATA)

DIRECTOR DEL PROYECTO: Sergio Bemposta

FECHA: Septiembre de 2023

RESUMEN

Este proyecto consiste en estimar el tráfico a partir de imágenes obtenidas por cámaras urbanas. Esto se realiza con el objetivo de probar el uso de redes convolucionales para controlar el tráfico y poder mejorarlo así en un futuro reduciendo el número de accidentes y la mortalidad de estos considerablemente.

Para realizar el proyecto se realizan 2 propuestas:

- El uso de una herramienta que permita el reconocimiento de objetos para contar el número de vehículos y estimar así la cantidad de tráfico o lo congestionada que se encuentra la carretera captada por la cámara.
- El uso de una red neuronal convolucional (CNN).

Tras estas propuestas hemos encontrado 2 principales problemas: la falta de tiempo y la escasez de datos disponibles. Para estos problemas se han propuesto soluciones como técnicas como el Data Augmentation y la monitorización del overfitting.

Tras el estudio y la experimentación llevada a cabo se ha logrado reconocer el tráfico de las imágenes con una precisión del 94%, un resultado que cumple con los objetivos de nuestro proyecto.

Palabras clave: Red Neuronal Convolucional, estimación de tráfico, cámara urbana, aumentado de datos.
Código fuente: https://github.com/hivansm/TFM_repo

ABSTRACT

This project consists of estimating traffic from images obtained by urban cameras. This is done with the aim of testing the use of convolutional networks to control traffic and thus be able to improve it in the future, reducing the number of accidents and their mortality considerably.

To carry out the project, 2 proposals are made:

- The use of a tool that allows object recognition to count the number of vehicles and thus estimate the amount of traffic or how congested the road captured by the camera is.
- The use of a convolutional neural network (CNN).

After these proposals we have found 2 main problems: the lack of time and the scarcity of available data. For these problems, solutions have been proposed as techniques such as Data Augmentation and overfitting monitoring.

After the study and experimentation carried out, it has been possible to recognize the traffic of the images with an accuracy of 94%, a result that meets the objectives of our project.

Key words: Convolutional Neural Network, traffic estimation, urban camera, data augmentation.
Source code: https://github.com/hivansm/TFM_repo

AGRADECIMIENTOS

Hace años, cuando empecé a estudiar ingeniería informática, en 2017, no había oído hablar de aprendizaje automático. Había oído hablar de la conducción autónoma o del reconocimiento de personas, pero no sabía lo que había detrás y, siendo sincero, tampoco es algo que me interesase demasiado. Tiempo después, 3 años después exactamente, asistí a mi primer curso de aprendizaje automático y pude comprender la importancia y la magia en este campo, fue entonces cuando me di cuenta de que quería profundizar en esto. Aprendiendo sobre el Deep Learning e investigando nuevas técnicas me di cuenta de que era mi vocación.
En esto se basa mi motivación en esta tesis. Para mí es una oportunidad de aprender cómo empezar a trabajar en este mundo y cómo funciona.

Universidad Europea

Índice

Índice de Figuras

Imagen 4.4.1. Funcionamiento de la aplicación.

Índice de Tablas

Capítulo 1. INTRODUCCIÓN

Este primer capítulo está dedicado a introducir el tema principal de la tesis y su contexto. En primer lugar, repasamos el trabajo realizado. Luego presentamos nuestra motivación, propuesta y objetivos, seguido de una línea temporal que define los pasos para desarrollar este trabajo y el esquema del proyecto.

1.1 Visión General

En este trabajo, nos enfocamos en estimar el tráfico rodado a partir de cámaras urbanas. Nuestro modelo centra su atención en la cantidad de "ruido" que encontramos en la carretera captada por las cámaras de tráfico con el objetivo de calcular la cantidad de tráfico en varias coordenadas distintas y poder así, en futuros proyectos, realizar las acciones oportunas, logrando prevenir accidentes de tráfico, optimizar el funcionamiento de los semáforos o incluso ahorrar energía apagando temporalmente la iluminación no aprovechada.

1.2 Motivación

Los accidentes de tráfico se cobraron la vida de 1.004 personas el pasado año. En 2021 se produjeron 921 siniestros mortales en las carreteras españolas en los que fallecieron 1.004 personas y otras 3.728 resultaron heridas graves.

Esto son unas cifras escandalosas que podrían reducirse considerablemente planteando una conducción óptima y un aprovechamiento de las carreteras.

En este trabajo proponemos una posible solución analizando el tráfico en varios puntos de nuestro país y clasificando en función de si hay o no retenciones para, en futuros proyectos, redirigirlo de la manera más óptima.

1.3 Objetivos

El principal objetivo de este proyecto es el de ser capaces de analizar el tráfico a partir de cámaras urbanas en varias zonas de control.

Como objetivos secundarios podemos considerar los siguientes:

- Analizar la aplicación de Redes Neuronales Convolucionales para la clasificación del tráfico en una imagen.
- Revisar el estado del arte en cuanto a datasets existentes relacionados con la detección y clasificación de vehículos en vías urbanas.
- Investigar proyectos relacionados y soluciones ya estudiadas anteriormente.

1.4 Timeline

Hemos dividido nuestra línea de tiempo (ver Imagen 1.4.1) en 4 fases que describimos en detalle a continuación.

		Nombre	Duración	Inicio	Terminado
1		⊞Fase inicial: Configuración y análisis del proyecto	3 days	1/06/22 8:00	3/06/22 17:00
3		⊞Fase de preparación: Revisión de aspectos iniciales y búsqueda de proyeyctos relacionados	29 days	6/06/22 8:00	14/07/22 17:00
16		⊞Fase de desarrollo: Revisión de fases anteriores e implementación del código	31 days	15/07/22 8:00	26/08/22 17:00
19		⊞Fase de transición: Experimentación y refinamiento del proyecto	14 days	29/08/22 8:00	15/09/22 17:00

Imagen 1.4.1. Timeline general.

1.4.1 Fase inicial

Para comenzar, dibujamos y analizamos el proyecto. Esta fase incluye una reunión inicial, donde analizamos el proyecto, y fijamos objetivos generales y específicos, las limitaciones del proyecto y su alcance. Además, durante esta primera etapa elegimos las herramientas y marcos adecuados. La duración de esta fase inicial fue de 3 días hábiles.

1.4.2 Fase de preparación

Nombre	Duración	Inicio	Terminado
⊞Fase inicial: Configuración y análisis del proyecto	3 days	1/06/22 8:00	3/06/22 17:00
⊟Fase de preparación: Revisión de aspectos iniciales y	29 days	6/06/22 8:00	14/07/22 17:00
⊟Iteración EM-1: Reconocimiento de objetos	23 days	6/06/22 8:00	6/07/22 17:00
Búsqueda de proyectos relacionados	10 days	6/06/22 8:00	17/06/22 17:00
Actualización del tiempo estimado	1 day	20/06/22 8:00	20/06/22 17:00
Establecer plazos	1 day	21/06/22 8:00	21/06/22 17:00
Configurar herramientas	1 day	22/06/22 8:00	22/06/22 17:00
Búsqueda y estudio de datasets	10 days	23/06/22 8:00	6/07/22 17:00
Búsqueda y estudio de modelos de detección de objetos	10 days	23/06/22 8:00	6/07/22 17:00
⊟Iteración EM-2: CNN	6 days	7/07/22 8:00	14/07/22 17:00
Actulización de la estimación de tiempos	1 day	7/07/22 8:00	7/07/22 17:00
Actualización de plazos	1 day	8/07/22 8:00	8/07/22 17:00
Búsqueda y estudio de funcionamiento de CNN	3 days	11/07/22 8:00	13/07/22 17:00
Refinar arquitectura	1 day	14/07/22 8:00	14/07/22 17:00

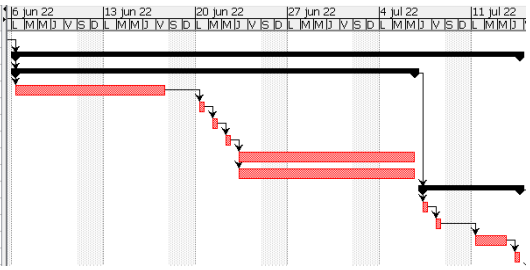


Imagen 1.4.2.1. Fase de preparación.

Esta segunda fase (Imagen 1.4.2.1) consiste en revisar los modelos existentes. Primero, revisamos
obras relacionadas. Luego, determinamos unos plazos y comenzamos con el proyecto. Incluye las herramientas de configuración seleccionadas en la fase inicial y la búsqueda de conjuntos de datos y modelos que pueden ser útil para nuestro proyecto. Estimamos 29 días hábiles para esta fase, debido a que la tarea de investigación es la más larga.

1.4.3 Fase de desarrollo

Nombre	Duración	Inicio	Terminado
⊞Fase inicial: Configuración y análisis del proy	3 days	1/06/22 8:00	3/06/22 17:00
⊞Fase de preparación: Revisión de aspectos	29 days	6/06/22 8:00	14/07/22 17:00
⊟Fase de desarrollo: Revisión de fases anter	31 days	15/07/22 8:00	26/08/22 17:00
Iteración CM-1: Reconocimiento de objetos	15 days	15/07/22 8:00	4/08/22 17:00
Iteración CM-2: CNN	16 days	5/08/22 8:00	26/08/22 17:00
⊞Fase de transición: Experimentación y refin	14 days	29/08/22 8:00	15/09/22 17:00

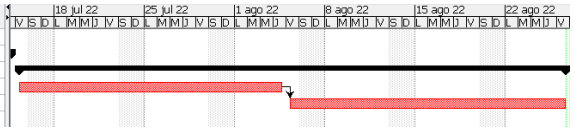


Imagen 1.4.3.1. Fase de desarrollo.

Una vez que hayamos elegido cada modelo y conjunto de datos, podemos comenzar a implementar el código. En esta fase (Imagen 1.4.3.1), codificamos todos los métodos propuestos y generamos una pequeña prueba para verificar que funcionen correctamente. Estimamos alrededor de 31 días para esta fase.

1.4.4 Fase de transición

Nombre	Duración	Inicio	Terminado
⊞Fase inicial: Configuración y análisis del proy	3 days	1/06/22 8:00	3/06/22 17:00
⊞Fase de preparación: Revisión de aspectos	29 days	6/06/22 8:00	14/07/22 17:00
⊞Fase de desarrollo: Revisión de fases anter	31 days	15/07/22 8:00	26/08/22 17:00
⊟Fase de transición: Experimentación y refina	14 days	29/08/22 8:00	15/09/22 17:00
⊟Iteración TM-1	14 days	29/08/22 8:00	15/09/22 17:00
Experimentación CNN	5 days	29/08/22 8:00	2/09/22 17:00
Solución de errores	4 days	5/09/22 8:00	8/09/22 17:00
Revisión de la memoria	5 days	9/09/22 8:00	15/09/22 17:00

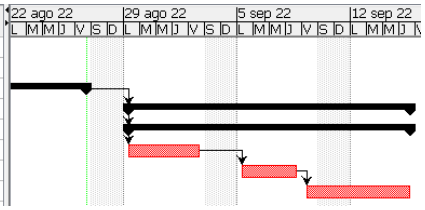


Imagen 1.4.4.1 Fase de transición

En esta última fase (Imagen 1.4.4.1), el propósito principal es hacer experimentos y analizar el código y los métodos que hemos implementado. Lo más importante en esta fase es extraer conclusiones de nuestro proyecto. Estimamos 14 días para esta fase.

1.5 Estructura del documento

Este trabajo consta de 6 capítulos: En el capítulo 1 presentamos la motivación, metas y calendario. En el capítulo 2 explicamos la estructura del algoritmo utilizado. En el capítulo 3 revisamos los trabajos relacionados. Se presentan los materiales y métodos utilizados. en el capítulo 4. Se describe la propuesta de nuestra arquitectura con experimentos cuantitativos en el capítulo 5 y experimentos cualitativos con imágenes reales. Resumimos las conclusiones y trabajos adicionales en el capítulo 6.

Capítulo 2. Estado del Arte

En este capítulo se explicará de forma general el funcionamiento del algoritmo a utilizar en el proyecto, las redes neuronales convolucionales. Para esto, tendremos en cuenta varios proyectos en los que podremos basarnos que serán descritos en el punto 2.1 y los datasets que pueden ser de utilidad en nuestro proyecto (2.2).

2.1 Proyectos Relacionados

Hoy en día, la tarea de reconocimiento de imágenes y vídeos está cobrando una gran importancia. La razón de esto es su aplicabilidad en muchos campos y entornos, como ayudar a prevenir peligrosos accidentes de tráfico en carreteras concurridas. Por esto aparecieron muchos estudios y proyectos que propusieron soluciones a este problema en diferentes campos de aplicación. En las próximas secciones revisamos en detalle algunos de los proyectos más interesantes.

2.1.1 Learning Traffic as Images: A Deep Convolutional Neural Network for Large-Scale Transportation Network Speed Prediction

Este proyecto está estrechamente relacionado a nuestro estudio, por lo que hemos visto interesante incluirlo en este apartado. El proyecto estudia la posibilidad de utilizar una red neuronal convolucional con una entrada de un tipo diferente a cualquier imagen. El objetivo es el de predecir el tráfico en función de la velocidad de los vehículos en varios segmentos de la carretera.

Para esto, lo primero es obtener esta velocidad junto a la localización del vehículo haciendo uso del GPS del teléfono móvil. A continuación, crea una matriz espacio-temporal de dimensiones X e Y, siendo X el tiempo en el que es obtenida la velocidad e Y el segmento de la carretera estudiado (Imagen 2.1.1.1).

Una vez obtenida esta matriz se utiliza de entrada a la red neuronal convolucional.

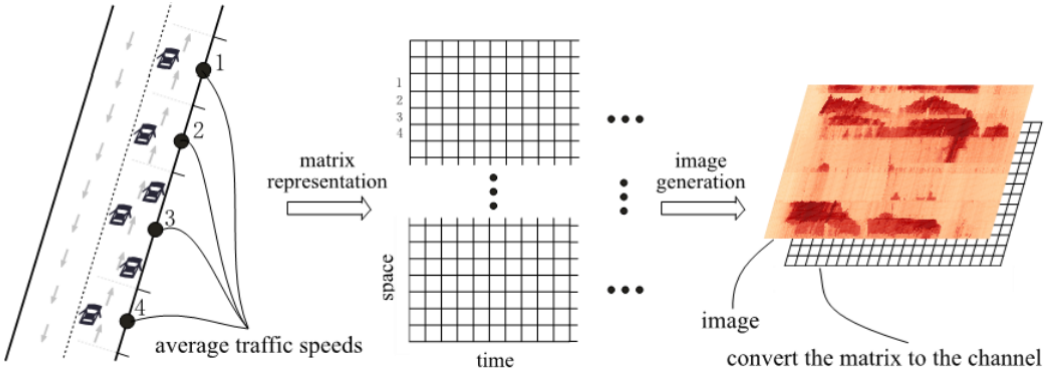
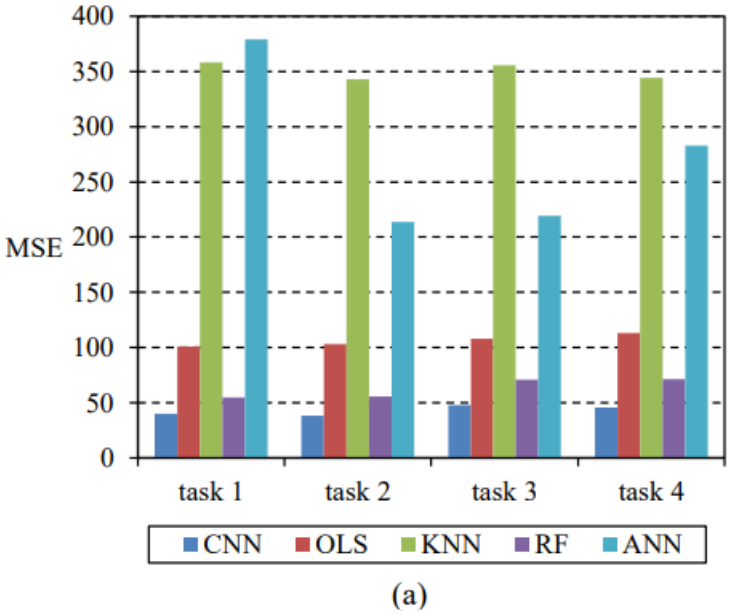


Imagen 2.1.1.1 Learning traffic images. [viii -> i]

Los resultados obtenidos es que esta técnica mejora el uso de otras técnicas como las KNN, OLS, RF o ANN estimando el tráfico a partir de la velocidad de los últimos minutos.



Task 1: 10-minute traffic prediction using last 30-minute traffic speeds;
Task 2: 10-minute traffic prediction using last 40-minute traffic speeds;
Task 3: 20-minute traffic prediction using last 30-minute traffic speeds;
Task 4: 20-minute traffic prediction using last 40-minute traffic speeds.

Imagen 2.1.1.2. Comparativa de algoritmos. [viii -> i]

2.1.2 Estimating Urban Traffic Density Using Street Camera Images

Este proyecto [ix] es muy interesante para nuestro proyecto ya que consiste en la estimación del tráfico a partir de las imágenes de cámaras de tráfico. Este estudio tiene la particularidad de que aplica una máscara, es decir, detecta la carretera y aplica esta máscara a las imágenes, de tal manera que a la CNN únicamente se le envía información en los píxeles que interesan. De esta forma, se facilitará mucho el aprendizaje de la red.

En este caso, se clasifica el tráfico en 5 niveles y tiene una precisión del 57% sin usar una red preentrenada y del 72% haciendo uso de ResNet50 (Tabla 2.1.2.1).

Model	Number of Epochs	Time taken for 1 Epoch	Training Accuracy	Testing Accuracy
Basic Model (0.25 Zoom)	25	~3 minutes	70	57
Inception-v3 (0.75 Zoom)	10	~39 minutes	63	65
ResNet50 (1.0 Zoom)	10	~67 minutes	83	72
SqueezeNet (0.25 Zoom)	30	~2.4 mins	68	68

Tabla 2.1.2.1. Comparativa de modelos. [ix -> ii]

Es una técnica muy interesante y que podría ser de gran utilidad en nuestro proyecto. No se podrá implementar debido al factor limitante del tiempo, pero será una buena opción para proyectos futuros.

2.2 Datasets

Para nuestro estudio, hemos comparado muchos conjuntos de datos (Figura 3.2) y hemos elegido los mejores para nuestro proyecto. En las siguientes secciones, los revisamos y describimos brevemente.

2.2.1 Big Data for Traffic Estimation and Prediction

Cameras Florida

Uno de los conjuntos evaluados fue "Cameras Florida", que podría ser muy útil para el proyecto. Este conjunto de datos contiene datos recopilados de 3.668 cámaras en Florida, lo que lo convierte en una fuente valiosa de información para nuestro proyecto.

La principal ventaja de "Cameras Florida" x es la cantidad de imágenes disponibles de cada una de las cámaras registradas. Este conjunto de datos podría proporcionar una muestra amplia y diversa de situaciones de tráfico en varios lugares, lo que permitiría un entrenamiento del modelo más completo y preciso. Sin embargo, a pesar de su potencial, se optó por no utilizar este conjunto de datos para el trabajo actual debido a la complejidad de etiquetar una cantidad tan significativa de datos en el tiempo disponible para el TFM.

Es importante destacar que "Cameras Florida" x es una opción prometedora para futuras ampliaciones y desarrollos del proyecto, aunque no se ha utilizado en esta ocasión. La tarea de etiquetar los datos de esta colección sería abordable con más tiempo y recursos. Esto permitiría incorporar una amplia gama de ejemplos y mejorar la capacidad del modelo para generalizar y adaptarse a diferentes condiciones de tráfico.

En resumen, el conjunto "Cameras Florida" es un conjunto de datos con un gran potencial que, aunque no se utilizó en el TFM actual debido a limitaciones de tiempo, se considera una opción

atractiva y adecuada para futuras expansiones y mejoras del proyecto. La amplia colección de registros de cámaras de Florida podría mejorar significativamente el entrenamiento del modelo y, por lo tanto, mejorar la eficacia y aplicación de la solución propuesta.

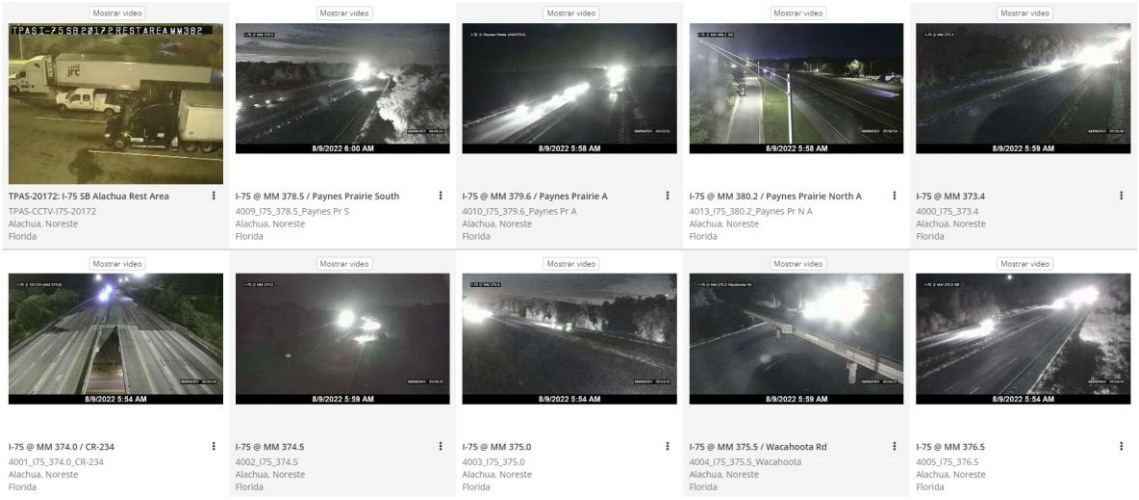


Imagen 2.2.1.1. Cameras Florida. [x->iii]

Al igual que para Florida podemos obtener datasets de otros estados de Estados Unidos de páginas similares, por lo que la cantidad de datos disponible es inmensa.

2.2.2 Traffic Congestion Dataset

Este dataset ha sido elegido debido a su contenido y etiquetado únicos. Este conjunto de datos contiene 4.238 imágenes de cámaras urbanas con una resolución de 500 x 500 píxeles cada una. La importancia radica en que cada imagen ha sido etiquetada en función de si hay mucho tránsito o no.

La elección de este conjunto de datos se basa en una serie de aspectos importantes que lo hacen adecuado para el progreso de nuestro proyecto:

1. Etiquetado específico: la etiquetación de imágenes según la presencia de congestión proporciona información útil para el aprendizaje del modelo. El modelo tendrá la capacidad de reconocer patrones relacionados con situaciones de alta congestión al tener ejemplos claros y definidos. Esto permitirá realizar predicciones más precisas y relevantes.

2. Tamaño del dataset: el conjunto de datos xi proporciona una gran cantidad de datos para el entrenamiento del modelo, con más de 4.000 frames. Esto es crucial porque un conjunto de datos más grande permite una generalización más efectiva y evita los problemas de sobreajuste que podrían surgir con conjuntos de datos más pequeños.

3. Resolución de imágenes: Las imágenes de 500 x 500 píxeles brindan suficiente detalle y contexto para capturar características importantes del tráfico urbano y las situaciones de congestión. Esta resolución correcta garantiza que el modelo tenga la información necesaria para realizar una clasificación precisa.

4. Aplicación al proyecto: el etiquetado de este conjunto de datos es extremadamente relevante porque el objetivo principal de nuestro proyecto es identificar y clasificar situaciones de tráfico. Esperamos que al entrenar al modelo con imágenes que representan situaciones de alta congestión, pueda aprender patrones distintivos y ayudar a detectar y prevenir condiciones de tráfico complicadas.

En conclusión, gracias a su etiquetado específico, su tamaño suficientemente grande y la resolución adecuada de las imágenes, el conjunto xi es una elección adecuada para el entrenamiento de nuestro modelo. Esperamos mejorar la capacidad de nuestro modelo para clasificar y predecir situaciones de gran congestión en las carreteras urbanas con este conjunto de datos.

Imagen 2.2.2.1 Traffic Congestión Dataset [xi -> iv]

2.2.3 Photole and Plain Road Dataset

Este dataset [xiii] incluye 370 imágenes de diferentes carreteras en diferentes estados. Tras el análisis del dataset podemos comprobar que ninguna de ellas incluye tráfico. Por tanto, podría ser de gran utilidad como complemento a algún dataset desbalanceado, es decir, con una mayor cantidad de imágenes de carreteras congestionadas.

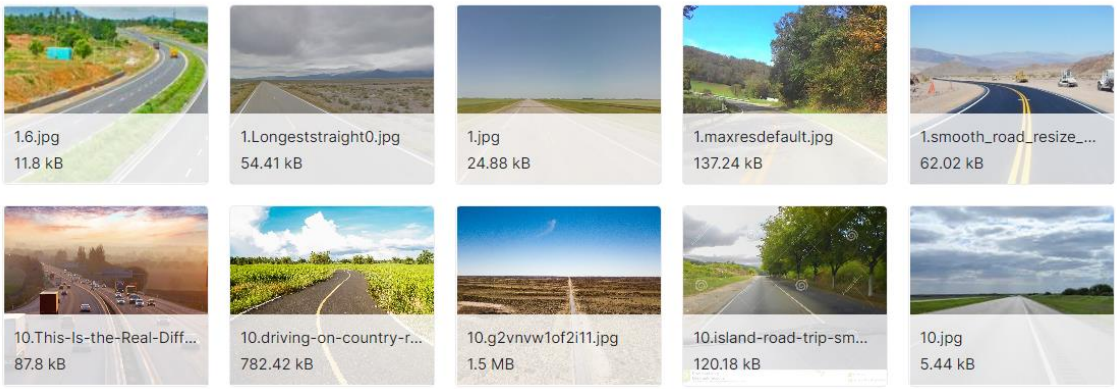


Imagen 2.2.3.1. Photole and Plain Road Dataset [xii ->v]

2.2.4 Cámaras DGT XIV->vi

El conjunto de imágenes a tiempo real capturadas por las cámaras de tráfico proporcionadas por la Dirección General de Tráfico (DGT) en este proyecto se ha utilizado para presentar los resultados.

Para este propósito, se utiliza el conjunto de datos para mostrar cómo funciona y qué ha logrado nuestra aplicación. Podemos demostrar cómo nuestra solución procesa y analiza información del tráfico en situaciones reales y dinámicas utilizando imágenes en tiempo real.

Las visualizaciones y las ilustraciones se utilizan para presentar los resultados, que destacan cómo la aplicación mejora la gestión del tráfico y proporciona información relevante a los usuarios en tiempo real. Además, este método nos permite destacar la capacidad de la aplicación para adaptarse a las condiciones del tráfico cambiantes y cómo se comporta en situaciones de congestión, accidentes y otras situaciones de tráfico.

Para presentar los resultados, el uso de imágenes a tiempo real de DGT ofrece una perspectiva más auténtica y realista del impacto y la utilidad de nuestra solución en comparación con datos estáticos o conjuntos de datos creados artificialmente. Por lo tanto, al representar situaciones reales a las que se enfrentaría la aplicación en un escenario práctico, nuestros resultados adquieren mayor relevancia y credibilidad.

Capítulo 3. Contexto teórico

Para comenzar es importante conocer el significado y la utilidad de las Redes Neuronales Convolucionales o Convolutional Neural Network (CNN).

A lo largo de los años se ha investigado mucho sobre cómo automatizar ciertas tareas monótonas para el ser humano, entre las que se encuentran algunas como vigilancia a través de cámaras de seguridad, lograr una conducción autónoma o la detección de anomalías en imágenes. El problema de estas tareas es que requieren de entender datos algo complejos debido a su estructura espacial, como imágenes o vídeos.

Para solventar este problema nacen las CNN, redes neuronales utilizadas para problemas que contengan datos con una estructura espacial.

3.1 Funcionamiento General

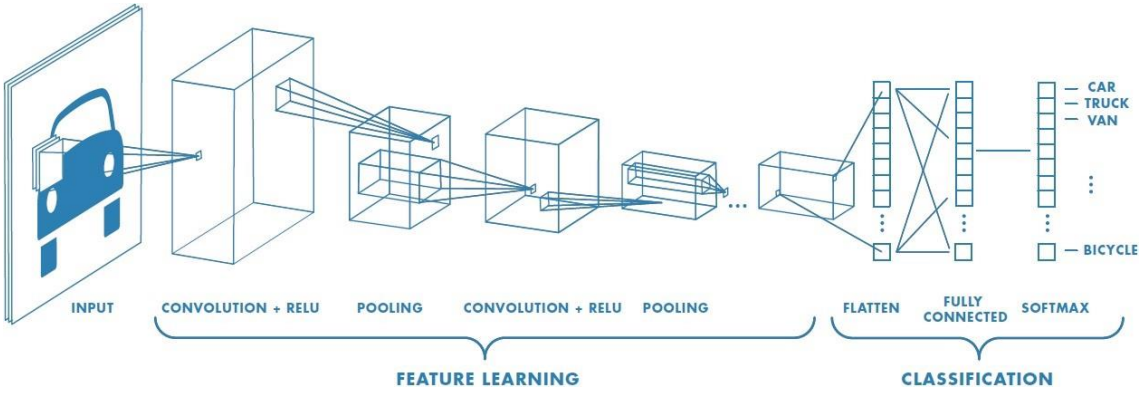


Imagen 3.1.1. Estructura de una CNN. [i] ->vii

Para entender el funcionamiento de estas redes comenzaremos explicando cómo se estructuran. Las CNN se componen de 2 partes principales (Imagen 3.1.1):

3.1.1 Extracción de características

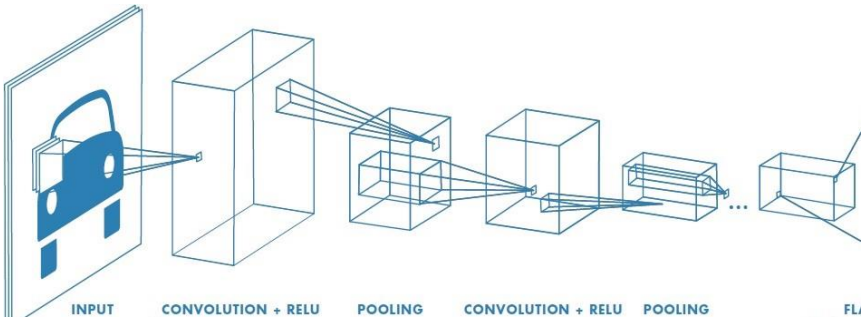


Imagen 3.1.1.1. Extracción de características. [i]->vii

En esta primera parte se extraen las características principales de la imagen a través de capas convolucionales y de pooling, explicadas en los puntos 3.2.1 y 3.2.2. (Imagen 3.1.1.1)

Tras esta extracción de características, se adaptan los datos para su estudio a través de la capa de Flattening que, simplemente, consiste en una capa que redimensiona los datos en una única dimensión.

3.1.2 Estudio de características

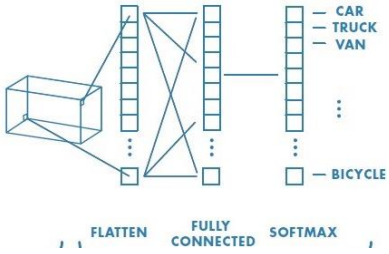


Imagen 3.1.2.1. Estudio de características. [i]->vii

Por último, tras extraer las características de la imagen se realiza un estudio de estas características haciendo uso de una red neuronal básica compuesta de capas densas. (Imagen 3.1.2.1)

3.2 Extracción de características

Para la extracción de características hace falta entender 2 pasos importantes. Por un lado se extraen las características de una imagen y, por otro lado, se resumen esas características.

3.2.1 Capa Convolucional

La capa convolucional es la encargada de extraer las características de una imagen. Para esto hace uso de filtros convolucionales o kernel que funcionan de la siguiente manera:

Para facilitar la comprensión supondremos que únicamente usaremos 1 filtro, de tal forma que partimos de una imagen y del filtro que utilizaremos para realizar la convolución (Imagen 3.2.1.1).

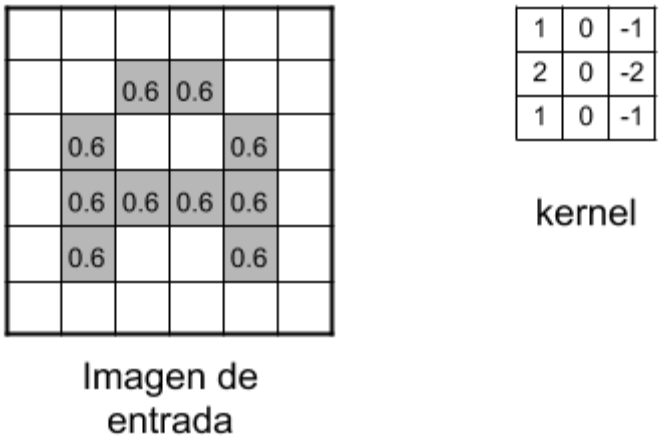


Imagen 3.2.1.1. Parámetros de entrada en capa convolucional. **¡Error! No se encuentra el origen de la referencia.**->viii

Para llevar a cabo la convolución se coloca el kernel sobre la imagen y se multiplica cada uno de los píxeles que abarca por sus respectivas posiciones en el kernel, a continuación, se suman sus puntuaciones y se coloca el resultado en el píxel central (Imagen 3.2.1.2).

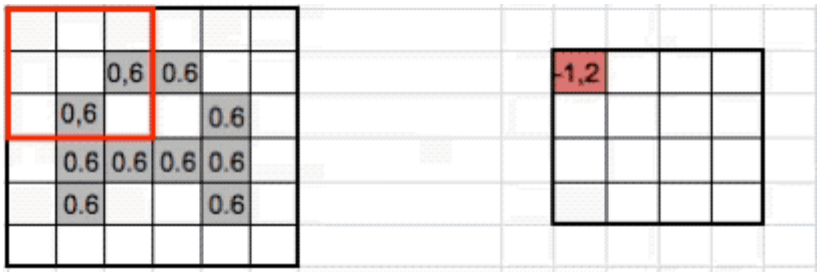


Imagen 3.2.1.2. Convolución 3x3. **¡Error! No se encuentra el origen de la referencia.**->viii

Por ejemplo, para extraer las líneas verticales utilizaríamos el siguiente filtro:

0	1	0
0	2	0
0	1	0

Hasta hace unos años todo este proceso se realizaba de forma manual eligiendo los filtros en función de las características que buscábamos. Pero, gracias a la aparición de las redes neuronales, estos filtros ya pueden calcularse de forma automática de igual forma que actualizamos los pesos en una red perceptrón multicapa (Punto 4.1).

3.2.2 Capa de Pooling

Tras realizar esta extracción, se resumen estas características aplicando el denominado Pooling. Para esta explicación supondremos que realizaremos un Max-Pooling de 2x2.

Esto, simplemente, consiste en calcular el valor máximo de varios píxeles vecinos de tal forma que, si tenemos una matriz de 4x4 y aplicamos este pooling, nos quedaría una matriz resultado de 2x2. (Imagen 3.2.2.1)

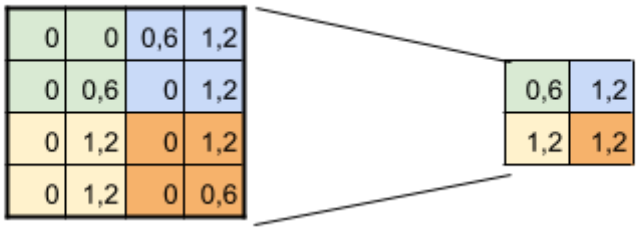


Imagen 3.2.2.1. Max-Pooling 2x2. **¡Error! No se encuentra el origen de la referencia.**->ix

3.2.3 Capa de Flattening

Tras varias capas convolucionales y de pooling nos falta realizar el estudio de las características obtenidas. Para esto antes deberemos adaptar los datos a la entrada de las siguientes capas (Capas Densas). Para esto, simplemente, redimensionaremos los datos a una única dimensión.

3.3 Estudio de las características

3.3.1 Capas Densas

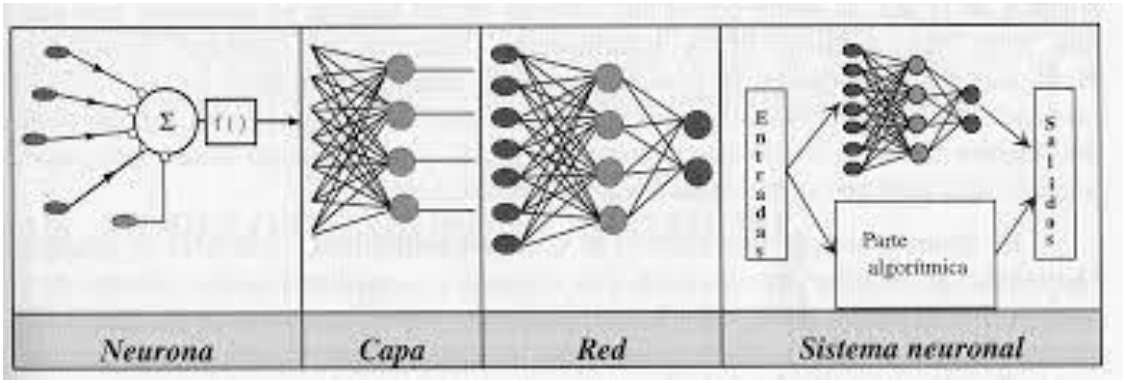


Imagen 3.3.1.1. Perceptrón multicapa. **¡Error! No se encuentra el origen de la referencia.** ->x

Son las capas principales a la hora de realizar el estudio de estas características.

Estas capas son simplemente funciones con varios parámetros o pesos a actualizar. Es decir, cada una de las capas contiene varias neuronas (Imagen 3.3.1.2) que consisten en funciones lineales a las cuales se les aplica una función de activación, como puede ser una función ReLu, sigmoidal, etc.

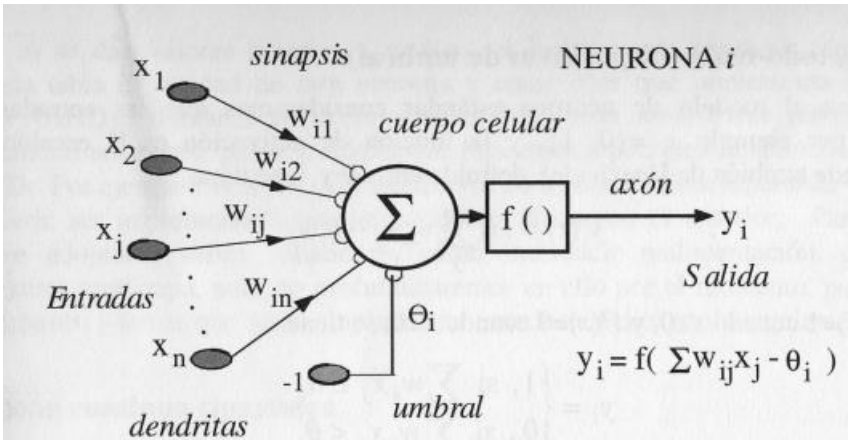


Imagen 3.3.1.2. Neurona. **¡Error! No se encuentra el origen de la referencia.**->x

Una vez tenemos una de estas funciones por cada neurona se introduce una serie de entradas a cada una de estas neuronas. A continuación, se utiliza el resultado obtenido en cada una de ellas como dato de entrada a las neuronas de la siguiente capa. (Imagen 3.3.1.3)

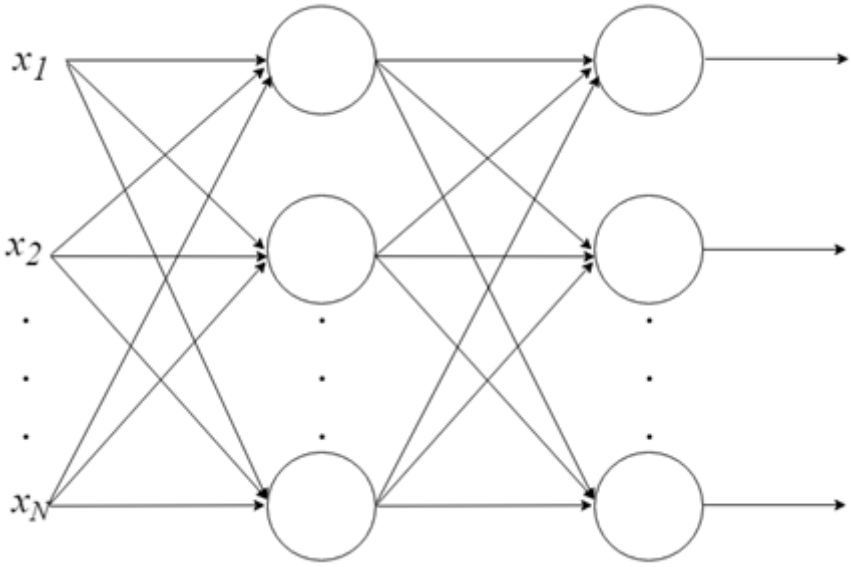


Imagen 3.3.1.3. Esquema Perceptrón Multicapa. [i]->xi

Este proceso se repite hasta finalizar con la capa de salida que, como su propio nombre indica, nos aporta el resultado final de la red neuronal. Este resultado puede ser una clasificación o un número (regresión).

Una vez obtenemos este resultado, se compara con el resultado real y se calcula cómo de grande ha sido el error para, en función del error, adaptar los pesos de cada una de las neuronas optimizando la función con el objetivo de minimizar este error y lograr que, con los datos reales, el error sea mínimo. A este proceso se le llama back propagation.

3.4 Preprocesado

Ahora que ya hemos visto cómo funcionan las redes convolucionales vamos a explicar algunos métodos de preprocesado para mejorar el rendimiento de la red.

3.4.1 Ruido Gaussiano

En todas las imágenes tenemos, en mayor o menor medida, el llamado ruido gaussiano (Imagen 3.4.1.1) debido a la radiación electromagnética o a falta de iluminación y/o temperatura. Esto confunde a la red a la hora de entrenar y de predecir el resultado.

Imagen 3.4.1.1. Ruido gaussiano. [xxi]->xii

Este ruido podemos reducirlo aplicando un filtro gaussiano (Imagen 12). Que consiste en aplicar una máscara similar al filtro de media (Imagen 3.4.1.2).

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Imagen 3.4.1.2. Máscara de filtro gaussiano. [xxi]->xii

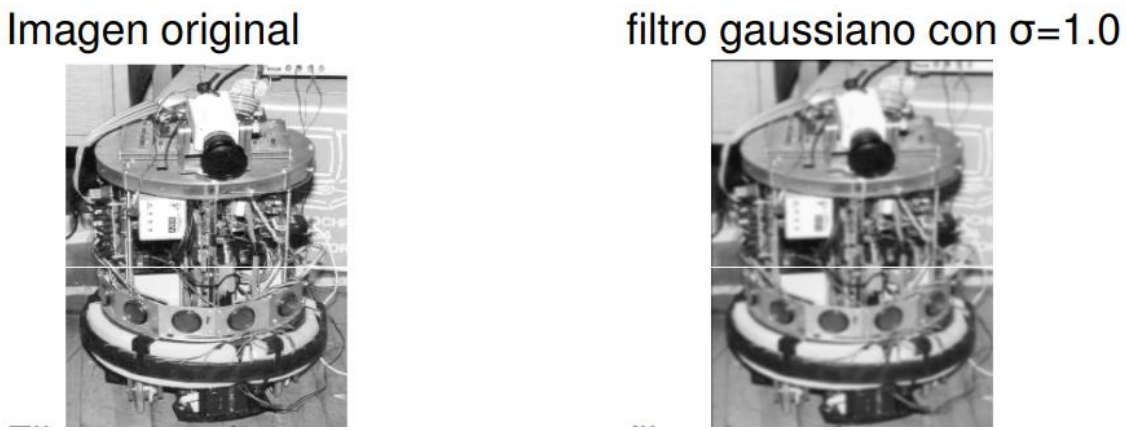


Imagen 12. Filtro gaussiano. [xxi]->xii

3.4.2 Ruido Impulsivo

Por otro lado, tenemos el ruido impulsivo, que se produce normalmente en la cuantificación que se realiza en el proceso de digitalización (Imagen 3.4.2.1).

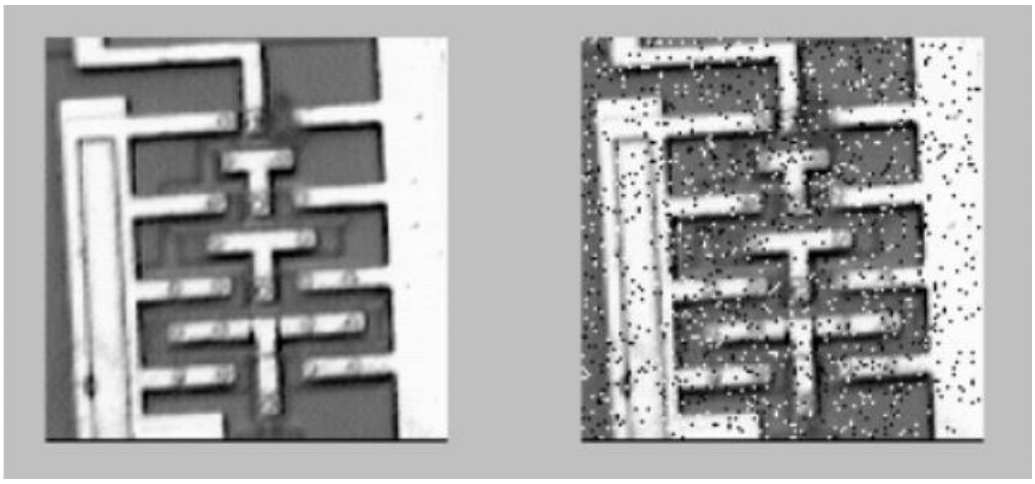


Imagen 3.4.2.1. Ruido impulsivo. [xxi]->xii

Para reducir este ruido podemos aplicar el filtro de la mediana (Imagen 3.4.2.2), que consiste en aplicar la mediana para cada píxel de sus píxeles vecinos.

Imagen 3.4.2.2. Filtro de la mediana. [xxi]->xii

3.4.3 Aumento de Datos

Por otro lado, tenemos el aumento de datos (data augmented). Actualmente, uno de los principales problemas a la hora de entrenar cualquier red neuronal es la falta de datos. Para entrenar estos modelos es necesario disponer de un mínimo de datos que, normalmente, cuesta obtener.

Para esto se aplica este preprocesado, el cual consiste en generar una mayor cantidad de datos de los que tenemos en nuestro dataset original. Para esto, en el caso de las imágenes, podemos aplicar rotaciones, zooms, volteos y/o desplazamientos.

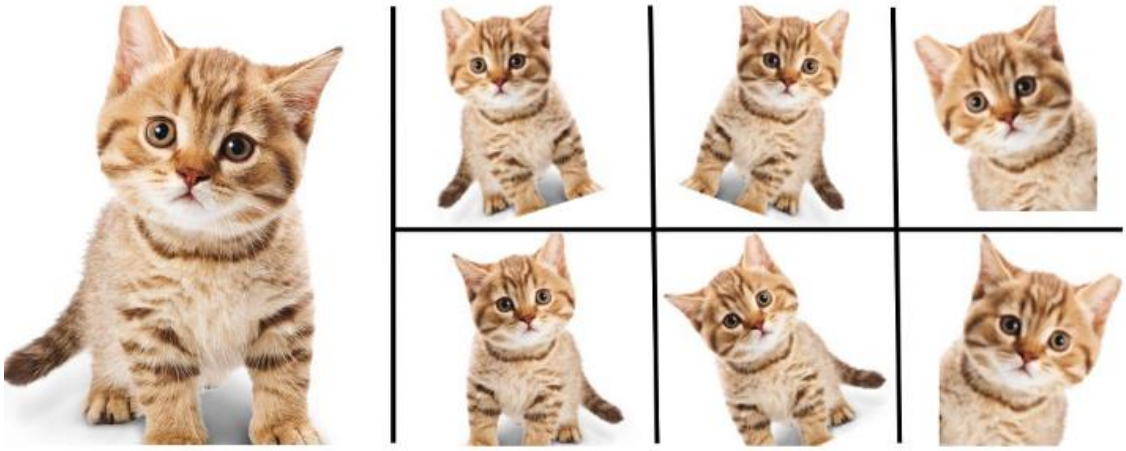


Imagen 3.4.3.1. Aumento de datos. **¡Error! No se encuentra el origen de la referencia.**->xiii

Capítulo 4. IMPLEMENTACIÓN DE LA SOLUCIÓN

4.1 Metodología de trabajo

En esta sección se comentará varias de las metodologías más utilizadas y por cual de ellas nos hemos decantado dando una serie de motivos que nos han llevado a tomar esta decisión. En nuestro caso hemos considerado la metodología SCRUM, es espiral y una metodología híbrida entre ambas analizando cada una de ellas junto a sus ventajas y desventajas para, finalmente, poder decidir cuál es la que mejor se adapta a nuestro proyecto.

4.1.1 SCRUM

En el desarrollo de software y otras industrias que requieren flexibilidad y adaptabilidad, SCRUM es una metodología de gestión de proyectos ágil. Se basa en la idea de hacer entregas incrementales y regulares, dividiendo el trabajo en sprints, que son períodos de tiempo definidos en los que se desarrollan y entregan funcionalidades del producto.

SCRUM se destaca por su enfoque en el trabajo en equipo y la comunicación efectiva. El Product Owner, que es responsable de definir y priorizar los requisitos del producto, el Scrum Master, que es responsable de facilitar el proceso y eliminar obstáculos, y el Equipo de Desarrollo, que es responsable de crear y entregar las funcionalidades, son los principales roles en SCRUM.

La autogestión del equipo en SCRUM permite que los miembros organicen y tomen decisiones de manera independiente. Además, fomenta la transparencia mediante tableros Kanban y reuniones periódicas, como reuniones de planificación de sprint, revisiones de sprint y reuniones diarias de seguimiento (Imagen 4.1.1.1).

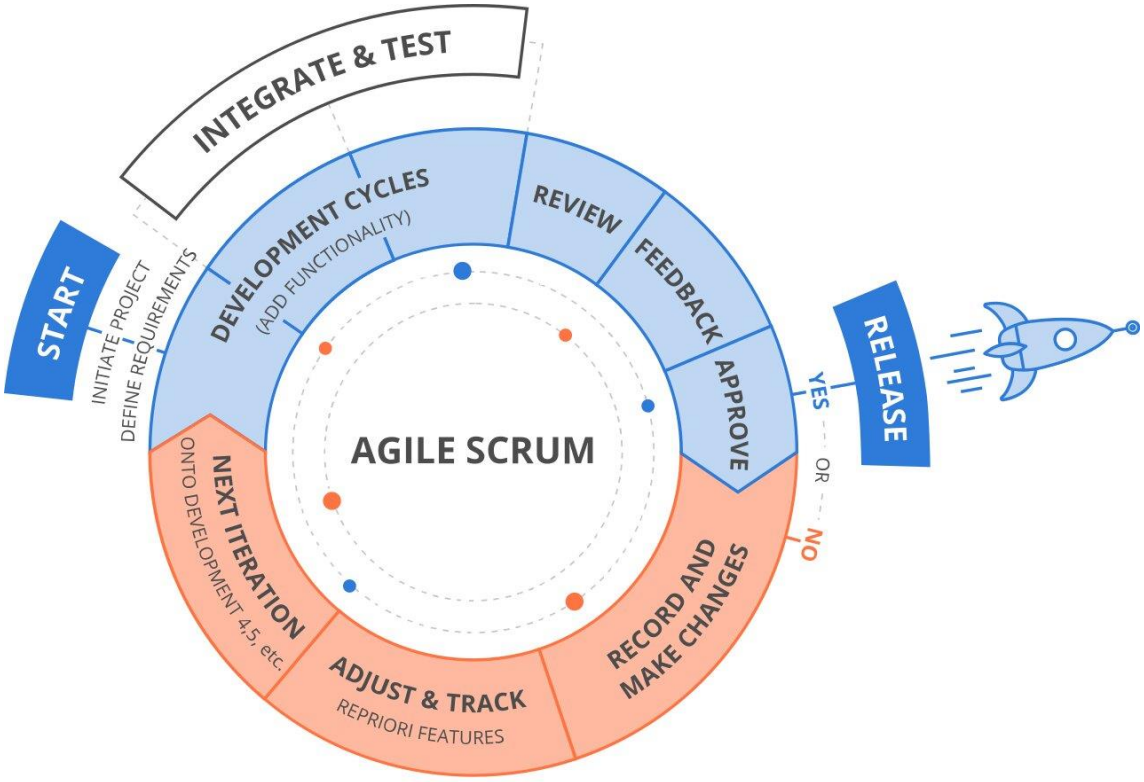


Imagen 4.1.1.1. Descripción.

Esto funciona bien para proyectos cuyos requisitos pueden cambiar o cambiar mientras se desarrollan. Además, permite una entrega temprana de funcionalidades y la posibilidad de recibir retroalimentación constante del cliente para realizar ajustes en el producto al dividir el trabajo en sprints y priorizarlo en función del valor para el cliente.

4.1.2 En espiral

La metodología en espiral combina la gestión de riesgos con un ciclo de vida iterativo. La metodología en espiral permite evaluar y reducir los riesgos del desarrollo del software a medida que el equipo avanza en el proyecto.

La metodología en espiral se divide en pasos que se repiten en un ciclo iterativo. La determinación de objetivos, la identificación de riesgos, la evaluación y resolución de problemas y la planificación de la siguiente fase son ejemplos de estas etapas (Imagen 4.1.2.1). El conocimiento y las lecciones aprendidas de las iteraciones anteriores sustentan cada iteración, lo que permite una mejora continua del producto y una mayor flexibilidad para adaptarse a los cambios.

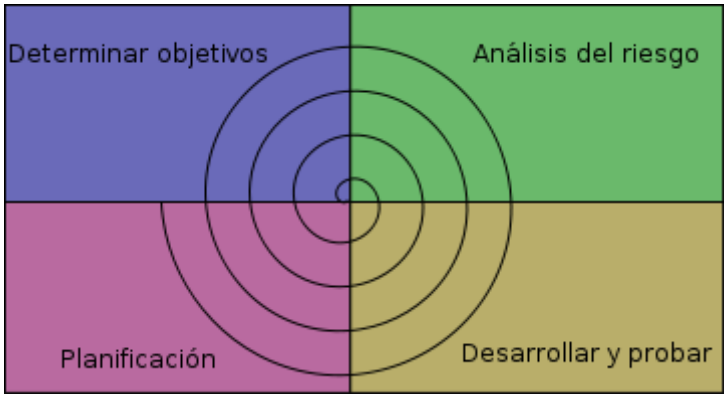


Imagen 4.1.2.1.

La principal ventaja de la metodología en espiral es que se enfoca en la gestión de riesgos. Al evaluar y abordar los riesgos desde el inicio del proyecto, se pueden mitigar los riesgos y evitar problemas más graves en el futuro. Esto reduce la incertidumbre y aumenta la probabilidad de que el proyecto tenga éxito.

Pero la metodología en espiral tiene algunas limitaciones. El análisis de riesgos puede ser más difícil de implementar y requiere más tiempo y recursos. Además, el enfoque iterativo puede aumentar los costos y el tiempo de desarrollo, lo que lo hace menos adecuado para proyectos pequeños o con requisitos estables.

4.1.3 Híbrida

La metodología híbrida combina una variedad de enfoques y prácticas de gestión de proyectos para adaptarse a las necesidades particulares de un proyecto o equipo. En este caso, la metodología híbrida combinaría SCRUM y espiral.

Se pueden aprovechar las ventajas de ambos métodos al utilizar una metodología híbrida. Por ejemplo, la estructura y los roles de SCRUM, como los sprints y los roles de Product Owner, Scrum Master y Equipo de Desarrollo, se podrían aplicar. Esto aumentaría la entrega de funcionalidades y la colaboración del equipo.

Al mismo tiempo, elementos de la metodología en espiral, como el análisis de riesgos y la evaluación continua, se podrían incorporar a lo largo del ciclo de vida del proyecto. Esto permitiría abordar los riesgos y los cambios de manera más proactiva, lo que garantizaría una mayor adaptabilidad y flexibilidad en la gestión del proyecto.

El enfoque de gestión de proyectos personalizado con la metodología híbrida permite adaptarse a diferentes contextos y necesidades. Sin embargo, es importante tener en cuenta que el uso de una metodología híbrida requiere una comprensión sólida de los principios y prácticas de cada enfoque, así como una planificación cuidadosa para garantizar una integración efectiva de ambos.

4.1.4 Metodología utilizada

Tras el análisis de las ventajas y desventajas de cada una de las metodologías optamos por la híbrida. Mientras SCRUM se centra en la entrega incremental y en la adaptabilidad a los cambios, la metodología en espiral se enfoca en la gestión de riesgos y la evaluación continua. Por tanto, integramos la metodología SCRUM haciendo uso de sus reuniones y entregas

incrementales con el cliente (tutor) para estar siempre al día de los progresos que se llevan en el proyecto y poder corregir cualquier error antes de que se agrave con el tiempo y se vuelva aún más difícil de mitigar. Además, debido a que es un proyecto de investigación existen varios riesgos como una infravaloración del trabajo y el tiempo necesario para investigar e implementar todas las funcionalidades planeadas, por lo que la metodología en espiral nos será de gran ayuda en esta tarea. Integraremos por tanto la metodología en espiral determinando y reevaluando continuamente objetivos y analizando sus riesgos.

4.2 Arquitectura

En esta sección se comentará la arquitectura de nuestra propuesta y cómo trascurre su flujo de datos.

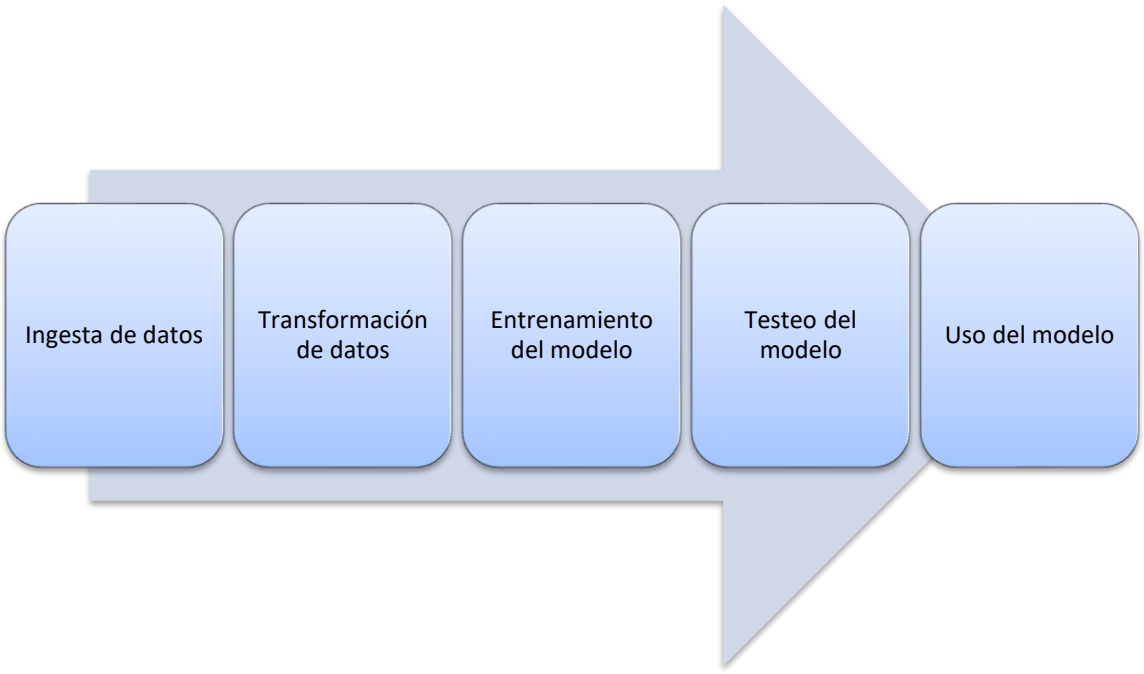


Imagen 4.2.1. Arquitectura utilizada.

Comenzaremos ingestando los datos comentados en el capítulo 2.2 “Datasets”.

4.2.1 Keras

Tras la ingestión se aplicarán una serie de transformaciones que ayudarán al aprendizaje de nuestro modelo con el objetivo de aumentar su rendimiento. Para esto haremos uso del framework “Keras” [xi->xiv], un importante y poderoso framework sencillo e intuitivo destinado al Aprendizaje Automático.

Por otro lado, TensorFlow es una biblioteca de aprendizaje automático de código abierto creada por Google Brain. Se enfoca en crear y entrenar modelos de aprendizaje automático a gran escala, como redes neuronales y otros algoritmos. A través de su arquitectura escalable y adaptable, TensorFlow permite el despliegue eficiente de modelos en una variedad de plataformas, como CPUs, GPUs y dispositivos móviles.

La conexión entre Keras y TensorFlow es que Keras puede ser utilizado como una interfaz de usuario avanzada para TensorFlow. A partir de TensorFlow 2.0, Keras se convirtió en la API principal predeterminada de TensorFlow, lo que nos permite utilizar Keras directamente para crear y entrenar modelos de aprendizaje automático.

La combinación de Keras y TensorFlow permite a los usuarios aprovechar la potencia y escalabilidad de Keras al mismo tiempo que se benefician de su facilidad de uso y escalabilidad. Esto permite disfrutar de la sintaxis fácil de entender y la facilidad de construcción de modelos de Keras mientras se aprovecha la amplia gama de funcionalidades y optimizaciones disponibles en TensorFlow.

Por esto hemos seleccionado "Keras Tensorflow" ->xv, porque es muy popular, flexible y sencillo, además ya habíamos trabajado antes con él. Este framework permite crear modelos lineales añadiendo capas como una secuencia y es con el que hemos desarrollado nuestra red convolucional. Además, ofrece muchas facilidades a la hora de transformar los datos utilizados.

Entre estas transformaciones se incluyen reducción del tamaño, aumentado de datos y demás transformaciones explicadas en el punto 3.4.

Una vez lo datos están preparados, se introducirán en la infraestructura propuesta en el punto 3.1, que nos devolverá un modelo ya entrenado para detectar el tráfico a partir de las imágenes dadas. Al igual que las transformaciones, nos ayudaremos del framework “Keras” para llevar a cabo la implementación de esta infraestructura.

4.2.2 SHAP

Imagen 4.2.2.1. Funcionamiento SHAP.

Ya con el modelo entrenado, se testeará su funcionamiento aplicando técnicas como SHAP [xxv->xvi], que ayuda a explicar cualquier modelo de aprendizaje automático, y estará listo para ser utilizado con imágenes reales.

SHAP (SHapley Additive exPlanations) es un enfoque teórico utilizado para explicar el resultado de cualquier modelo de aprendizaje automático (Figura 4.2.2.1). Conecta la asignación óptima de créditos con explicaciones locales utilizando los valores clásicos de Shapley de la teoría de juegos y sus extensiones relacionadas.

Esta herramienta nos será muy útil para poder estudiar el funcionamiento de nuestro modelo y si éste funciona como esperamos.

4.2.3 Google Colaboratory

Para ejecutar este flujo de datos ha sido necesario utilizar un hardware que nos permita llevarlo a cabo [xxii->xvii]. Para nuestro caso, debido a la limitación del presupuesto, hemos decidido aprovechar los recursos que nos ofrece Google de manera gratuita haciendo uso de su GPU para agilizar los cálculos. Estos tienen las siguientes propiedades:

- RAM: 12.68 GB
- Disco: 78.19 GB

Como hemos comentado una de las principales ventajas de Google Colab es su acceso gratuito. La plataforma no requiere configuraciones ni tarifas de suscripción, lo que nos permite aprovechar sus funcionalidades sin ningún coste. Además, de esto proporciona potentes recursos computacionales, como unidades de procesamiento gráfico (GPU) y unidades de procesamiento tensorial (TPU). Estos recursos aceleran significativamente el entrenamiento y el procesamiento de las CNN. Esto es particularmente ventajoso a la hora de entrenar nuestro modelo para analizar el tráfico.

Otra ventaja que nos impulsa a decidirnos por esta herramienta es que basa su funcionamiento en Jupyter Notebook, una interfaz interactiva que facilita la creación, ejecución y documentación de código. La creación de celdas de código, la visualización de resultados intermedios y la documentación de los pasos realizados nos van a facilitar enormemente el trabajo para el desarrollo de estas redes. En conclusión, nos facilitará la experimentación y la depuración del desarrollo.

4.2.4 Rog Strix

Además de los recursos ofrecidos por Google Colab también se han utilizado los recursos del ordenador personal Rog Strix que consiste en:

- RAM: 16 GB
- Disco: 0.5 TB
- Procesador: AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz
- GPU: Geforce GTX

4.2.5 GitHub

Además de todo esto, ha sido necesario el uso de un gestor de versiones, por lo que hemos hecho uso de GitHub, un gestor de versiones muy popular en los últimos años debido a la comodidad y facilidad de uso.

En estos repositorios hemos guardado cada uno de los cambios realizados tanto en el código como en la documentación.

El repositorio de GitHub utilizado ha sido https://github.com/hivansm/TFM_repo.

4.2.6 Método propuesto

El primer método que se propuso no fue llevado a cabo. Consiste en hacer uso de YOLO, un framework que detecta los objetos existentes en una imagen a tiempo real.

Este método simplemente trata de contar el número de vehículos que encontramos en una imagen y, en función de si es un número grande o pequeño, calcular si la carretera de la imagen está congestionada o no.

Sin embargo, nos encontramos con que el método tiene un problema añadido. No todas las imágenes están tomadas del mismo punto de visita ni son de la misma carretera, lo que implica que, para realizar el cálculo, debemos tener en cuenta la extensión de la carretera captada por la cámara. Por esto decidimos buscar otra solución más sencilla.

Tras la complicación que supone el primer método propuesto pensamos que, a veces, lo más sencillo es lo más eficaz. Por tanto, decidimos implementar una red neuronal convolucional y ver el resultado que obtenemos.

Nuestra idea final es implementar nuestro modelo convolucional, realizar un aumentado de datos del dataset “Traffic Congestión Dataset” (Punto 2.2.2) y entrenar con ello nuestro modelo de aprendizaje. Una vez hecho esto, evaluaremos su fiabilidad y lo utilizaremos para analizar si existe congestión de tráfico en 4 carreteras muy concurridas de Madrid.

4.3 Datos

El dataset que hemos elegido el “Traffic Congestión Dataset” debido a la comodidad que nos proporciona el tener todos los frames ya etiquetados.

Tiene un formato de etiquetado diferente al que estamos acostumbrados. Es decir, por cada imagen disponible “nombreImagen.jpg” tenemos un documento de texto “nombreImagen.txt” en el que en la primera línea tendremos un ‘0’ en caso de existir tráfico en la carretera y un ‘1’ en caso contrario.

Es por esto que tendremos que adaptar el formato como se explica en el punto “X.Y.Z. Ingesta de Datos”.

4.3.1 Ingesta de datos

Antes de empezar a implementar el modelo debemos comenzar cargando los datos, para lo que utilizaremos Keras.

Este apartado es simple pero imprescindible, ya que los datos del dataset elegido no tenían un formato compatible con Keras, por lo que, antes de cargar los datos, hemos tenido que adaptar el formato.

Keras obtiene el etiquetado en función de la jerarquía de carpetas, por lo que hemos tenido que redefinir su estructura en función del etiquetado de cada imagen de la siguiente forma (Imagen 4.3.1.1):

Imagen 4.3.1.1 Jerarquía de los datos.

4.3.2 Preprocesado

Al implementar la red nos damos cuenta de que el tamaño de las imágenes (500 x 500) puede darnos 2 problemas principales:

- Reducirá la velocidad de entrenamiento y de predicción de la red considerablemente.
- Necesitaremos de un modelo mucho más complejo, lo que implica una mayor cantidad de datos necesaria para el entrenamiento del mismo.

Por tanto, al cargar los datos redimensionamos estas imágenes a 32 x 32. Es decir, reduciremos los píxeles de 250.000 a 1.024.

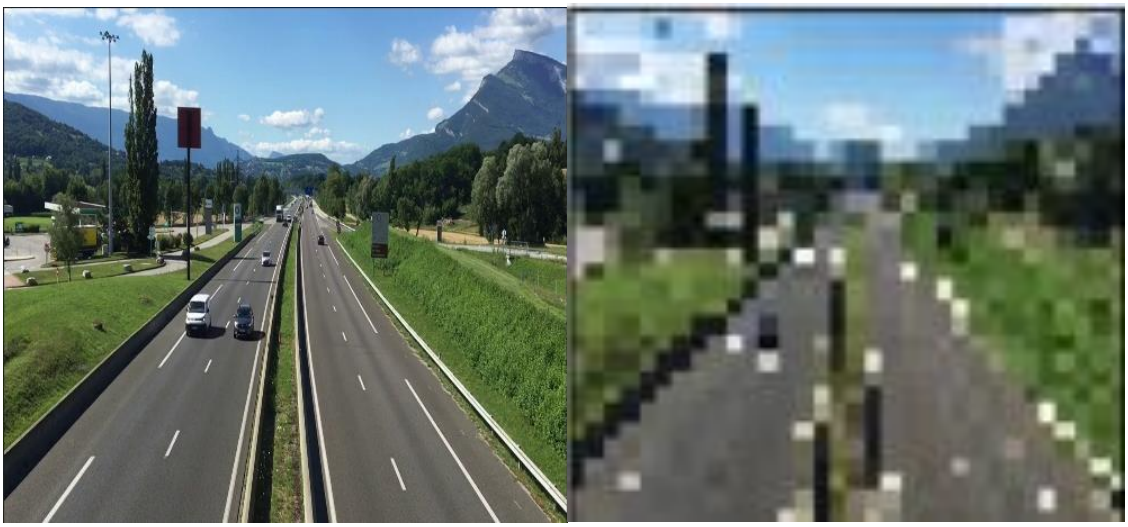


Imagen 4.3.2.1. Reducción de la imagen.

Además, normalizamos el valor de los píxeles, de tal forma que, en lugar de valores entre 0-255, tengan valores entre 0-1. (Anexo 1)

```
IMAGE_SIZE = 32
BATCH_SIZE = 64

# Data augmentation
datagen = ImageDataGenerator(rescale=(1/255))
```

```
train_generator = datagen.flow_from_directory(
  dir_img_only,
  target_size=(IMAGE_SIZE, IMAGE_SIZE),
  batch_size=-1,
  shuffle = False,
  class_mode = 'binary',
  classes = ['no_traffic','traffic']
)
```

Anexo 1

4.3.3 Aumentado de datos

Actualmente, tenemos 2.119 imágenes de cada tipo en nuestro dataset, lo que puede ser algo escaso. Por tanto, decidimos obtener una mayor cantidad de datos aumentándolos. Esto nos será muy útil para reducir el overffitting y, de esta forma, poder entrenar durante más tiempo la red logrando que obtenga una mayor precisión.

Hemos hecho uso del framework "Keras", explicado en el punto 4.2.1, para realizar pequeñas modificaciones en cada una de las fotos como realizar zoom, volteos, desplazamientos (Anexo 2)… De esta forma hemos obtenido una mayor cantidad de datos y haciendo nuestro modelo más efectivo.

Imagen 4.3.3.1. Aumentado de datos.

```
def data_augmentation(X_train,y_train,batch_size):
  image_gen=ImageDataGenerator(
    rotation_range=40,  #Rota imagenes aleatoriamente en un rango
de 40° [-40 - 40]
    width_shift_range=0.2,  #Desplaza imagenes aleatoriamente
    height_shift_range=0.2, #Desplaza imagenes aleatoriamente
    shear_range=0.2,  #Recorta imagenes aleatoriamente
    zoom_range=0.2, #Zoom en imagenes aleatoriamente
    horizontal_flip=True,
```

```
    fill_mode='nearest')

  data_augmented=image_gen.flow(X_train, y_train,
batch_size=batch_size,seed=1)

  return data_augmented
```

Anexo 2

4.4 Modelo y parámetros

4.4.1 Arquitectura de la red

Para obtener la arquitectura final del proyecto hemos probado 2 opciones. Por un lado hemos implementado una CNN desde 0 como baseline y, por otro lado, hemos aprovechado una red ya entrenada (ResNet50) para adaptarla a nuestro código.

Para entender esto es importante entender lo que es una CNN. A lo largo de los años se ha investigado mucho sobre cómo automatizar ciertas tareas monótonas para el ser humano, entre las que se encuentran algunas como vigilancia a través de cámaras de seguridad, lograr una conducción autónoma o la detección de anomalías en imágenes. El problema de estas tareas es que requieren de entender datos algo complejos debido a su estructura espacial, como imágenes o vídeos.

Para solventar este problema nacen las CNN, redes neuronales utilizadas para problemas que contengan datos con una estructura espacial. Es por esto que este tipo de red será la que utilicemos en nuestro proyecto.

4.4.1.1 Red sin preentrenar

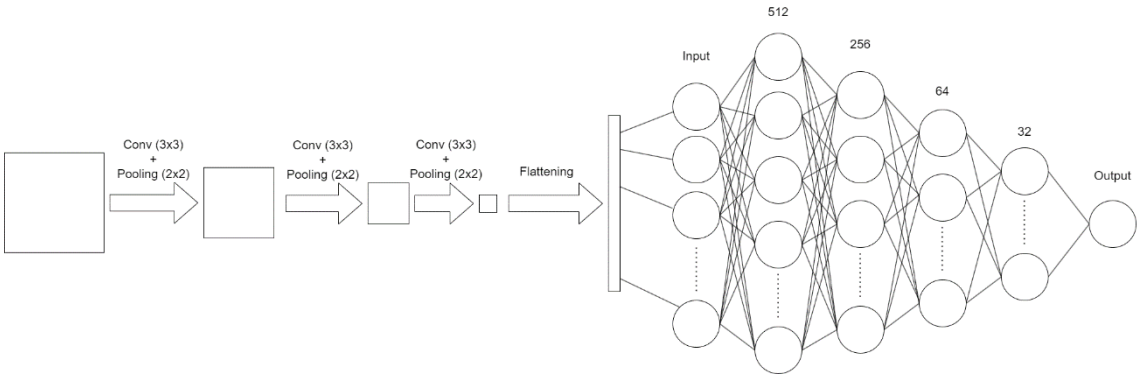


Imagen 4.4.1.1.1 Red sin preentrenar.

```
def cnn_model2():
  visible = layers.Input(shape=X[0].shape)
  x = layers.Conv2D(16,(3,3),padding="same",activation="relu")(visi
ble)
  x = layers.MaxPooling2D((2,2))(x)

  x = layers.Conv2D(32,(3,3),padding="same",activation="relu")(x)
  x = layers.MaxPooling2D((2,2))(x)

  x = layers.Conv2D(64,(3,3),padding="same",activation="relu")(x)
```

```
x = layers.MaxPooling2D((2,2))(x)

x = layers.Flatten()(x)

x = layers.Dense(512,activation="relu")(x)

x = layers.Dense(256,activation="relu")(x)

x = layers.Dense(64,activation="relu")(x)

x = layers.Dense(32,activation="relu")(x)

output = layers.Dense(1, activation='sigmoid')(x)

model = keras.Model(inputs=visible, outputs=output)

return model
```

Anexo 3

Como observamos en la imagen 3.3.1.1.1, nuestra red, como cualquier red convolucional, consta de 2 partes principales: La extracción de características y su estudio. (Anexo 3)

Extracción de características

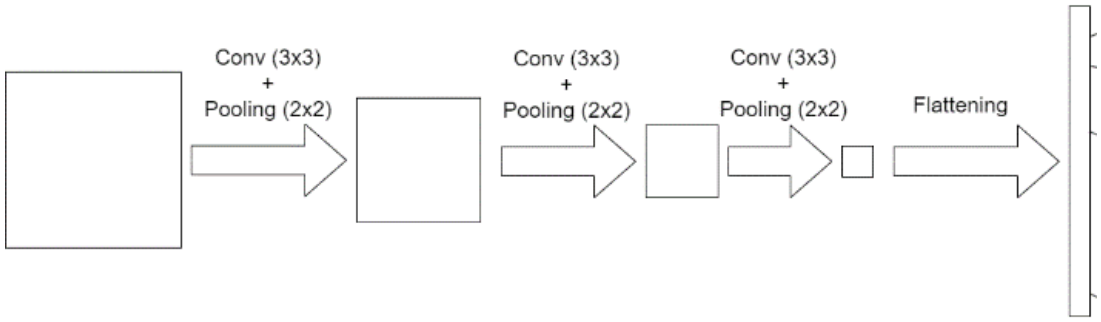


Imagen 4.4.1.1.2. Extracción de características.

En esta primera parte se extraen las características principales de la imagen a través de capas convolucionales y de pooling.

La capa convolucional es la encargada de extraer las características de una imagen haciendo uso de filtros convolucionales o kernel. Mientras que la capa de pooling es la ecargada de resumir esas características en información más compacta.

Esta primera parte se compone de 3 capas de convolución de 3x3. En estas capas se utiliza una función de activación Relu ya que, actualmente, es la que mejor rendimiento obtiene en la gran mayoría de capas intermedias de una red. Por otro lado, hemos utilizado un padding, lo que significa que se añaden píxeles alrededor de la imagen para que, al realizar la convolución,

no se reduzca el tamaño de la información de tal forma que, esta reducción, se lleve a cabo únicamente en sus capas correspondientes, las capas de pooling.

Estas capas convolucionales se intercalan con capa de pooling de 2x2, cuya función, como se ha comentado en puntos anteriores, es resumir las características obtenidas.

Por último, usaremos una capa de Flattening que aplane nuestros datos a una única dimensión que sea compatible con el input de la siguiente fase.

Estudio de las características

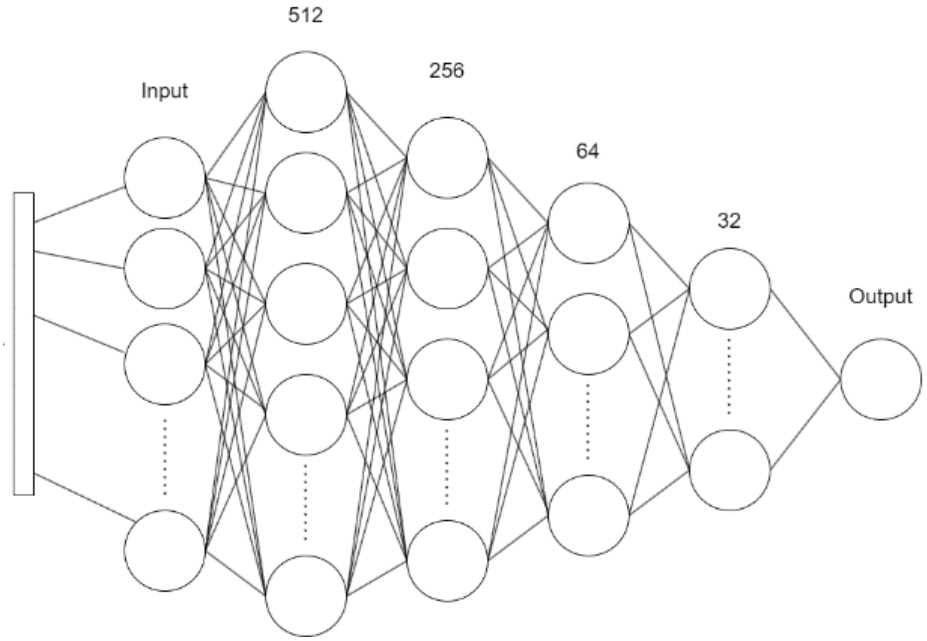


Imagen 4.4.1.1.3. Estudio de las características.

Tras finalizar esta primera parte, utilizamos las características obtenidas como entrada a una red multiperceptrón. Esta red se compone de 4 capas intermedias de 512, 256, 64 y 32 neuronas y en las que también hemos utilizado la función Relu como función de activación.

Por último, utilizamos 1 única neurona que nos proporcionará el resultado en formato binario (0 ó 1). Para esta última neurona utilizamos la función de activación sigmoide, ya que esta función nos ofrece resultados entre 0 y 1 de forma proporcional.

Hemos elegido esta estructura para que la información se comprima más en cada una de las capas hasta llegar al resultado final.

4.4.1.2 Red preentrenada

Otro modelo que hemos probado ha sido haciendo uso de la red preentrenada “ResNet50” (Anexo 4), un modelo explicado en el punto 4.4.1.2.1. Simplemente, hemos añadido una capa de salida de 1 única neurona y con la función de activación sigmoide para adaptarlo a nuestras necesidades.

```
def cnn_model():
  #Descargamos el modelo
```

```
  URL = 'https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vec
tor/4'
  feature_extractor =  hub.KerasLayer(
      URL,
      input_shape=X[0].shape
  )

  #Bloqueamos el entrenamiento de los pesos ya entrenados de esta r
ed
  feature_extractor.trainable=False

  #Adaptamos la red a nuestras necesidades
  model = tf.keras.models.Sequential([
    feature_extractor,
    tf.keras.layers.Dense(1,activation='sigmoid')
  ])

  return model
```

Anexo 4

4.4.1.2.1 ResNet50

ResNet50 es una red neuronal convolucional (CNN) de 50 capas de profundidad entrenada con más de 1 millón de imágenes obtenidas de la base de datos de ImageNet (X). Esta red tiene una entrada de 224x224 y es capaz de clasificar imágenes en 1.000 clases diferentes.

4.5 Experimentación

4.5.1 Overfitting

Lo primero en este punto es entender qué es el overfitting y el undedrfitting. El underfitting consiste en entrenar el modelo muy poco tiempo de forma que no le da tiempo a aprender, mientras el overfitting es lo contrario, es decir, entrenar demasiado tiempo el modelo con los mismos datos de tal forma que en lugar de aprender de estos datos los memoriza, por lo que al utilizar el modelo con datos nuevos no sabrá cómo clasificarlos.

El overfitting puede detectarse observando la gráfica del accuracy (Imagen 4.5.1.1) y comparando el accuracy de los datos de entrenamiento con los datos de test.

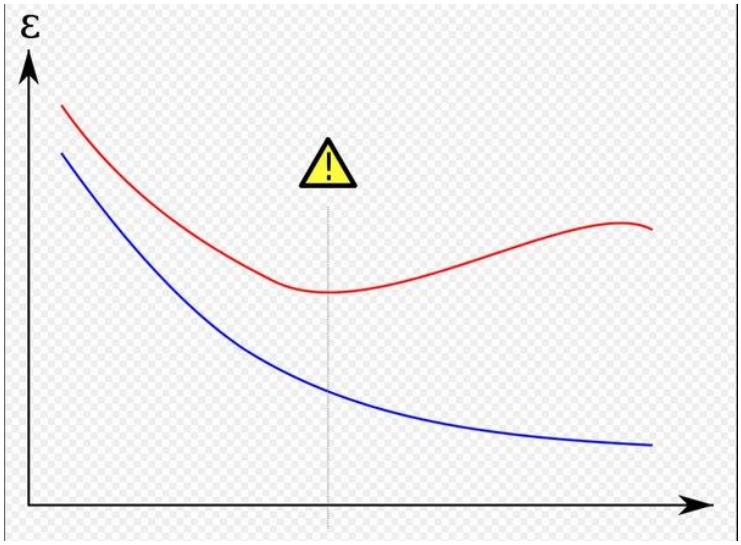


Imagen 4.5.1.1. Detectando el overfitting mediante el accuracy. La línea roja es el accuracy del conjunto de test mientras la línea azul es el accuracy del conjunto de entrenamiento.

El overfitting es un problema muy común en el entrenamiento de modelos. Es por esto que hemos prestado especial atención en detectarlo y evitarlo. Para esto, por un lado, hemos monitorizado cómo evoluciona tanto el accuracy como el loss tanto del conjunto de entrenamiento como del conjunto de validación.

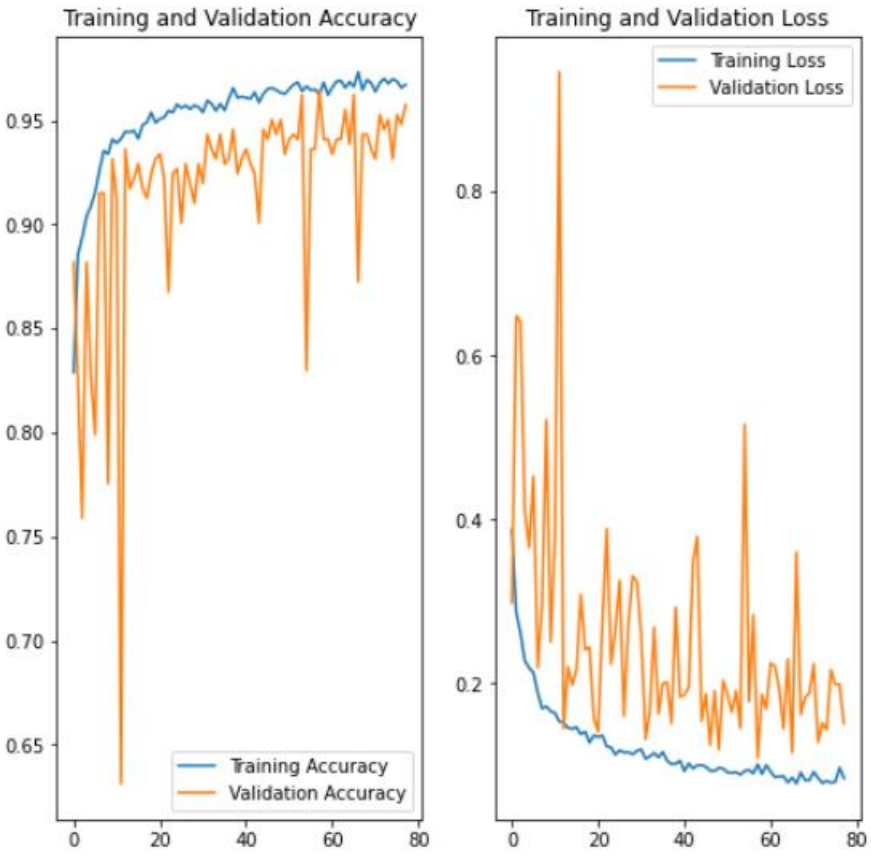


Imagen 4.5.1.2. Comparación entre el accuracy del conjunto de entrenamiento y de validación.

En la Figura 4.5.1.2 podemos observar como el accuracy de ambos conjuntos se mantienen similares, lo que implica que no está memorizando los datos del conjunto de entrenamiento y, por tanto, no existe overfitting.

Por otro lado, comprobamos que el loss del conjunto de validación no tiene una tendencia a aumentar en ningún momento. Esto es otra señal de que el entrenamiento está funcionando correctamente sin sobreajustar los parámetros.

Por último, hemos hecho uso del early_stopping, que pone el foco en el loss del conjunto de validación y, si comienza a empeorar, para automáticamente el entrenamiento.

4.5.2 SHAP

A pesar de todas estas comprobaciones los resultados del modelo podrían no tener una fiabilidad del 100%. Es decir, si los datos utilizados contienen algún sesgo que diferencie ambas clases (con tráfico y sin tráfico) el modelo podría estar aprendiendo a diferenciarlos por este sesgo. Es decir, supongamos que en el conjunto de datos las imágenes con tráfico siempre se encuentran en una misma carretera y las imágenes sin tráfico se encuentran en otra distinta. En este caso, el modelo podría estar aprendiendo a diferenciar las imágenes en función de la carretera captada en lugar de prestando atención al tráfico que encontramos en la imagen.

Para asegurarnos de que esto no ocurre veremos en que está poniendo el foco nuestro modelo con un subconjunto de datos. Para esto haremos uso de la herramienta SHAP (explicada en el punto 4.2.2), una herramienta que nos facilitará esta tarea.

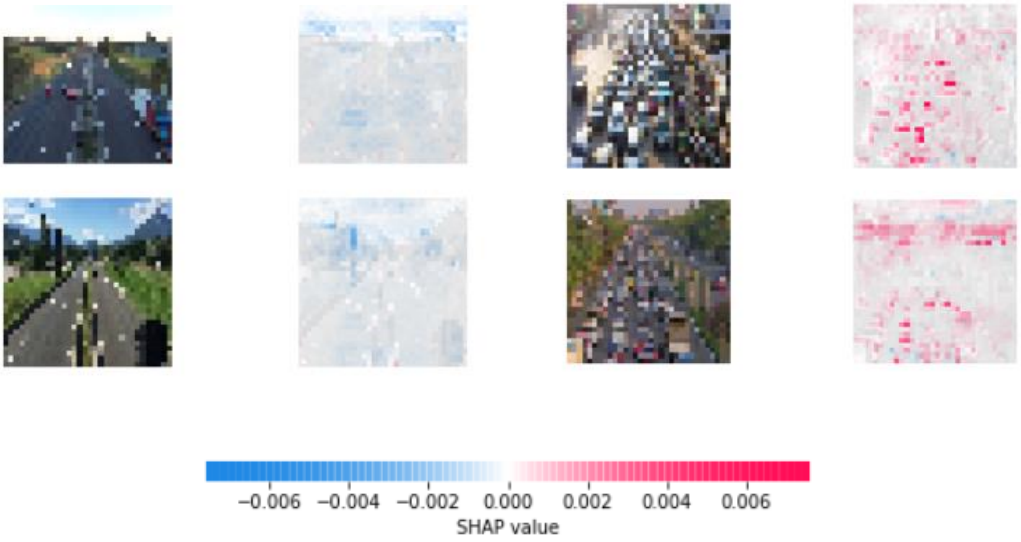


Imagen 4.5.2.1. Resultados SHAP.

Como vemos en la Imagen 4.5.2.1, esta herramienta pinta cada píxel de un color: Rojo si el píxel decanta la solución hacia la clase “Con tráfico” y azul si la decanta hacia la clase “Sin tráfico”.

Con estas imágenes podemos comprobar que, efectivamente, nuestro modelo está poniendo el foco de atención en los vehículos para clasificar la imagen.

4.5.3 Comparación de modelos

Al tener 2 modelos distintos queremos comprobar cuál de los 2 obtiene mejores resultados. Para esto necesitaremos utilizar unas métricas fiables. Podríamos utilizar el accuracy, pero esta métrica es fiable únicamente si los datos están perfectamente equilibrado, por eso suele utilizarse el AUC, es decir, el área cubierta por la curva ROC. Este parámetro sí tiene en cuenta el posible desequilibrio de datos y, por tanto, es una métrica mucho más fiable.

Una vez decidido el parámetro podemos tener otro problema, y es el problema de la aleatoriedad. Es decir, podría darse que un modelo peor obtenga mejores resultados simplemente porque los datos se hayan distribuido a su favor mejorando, de forma excepcional, su resultado. Es por esto que normalmente se utiliza el "cross validation" o "validación cruzada", simplemente consiste en generar varios subconjuntos y entrenar el modelo varias veces utilizando un subconjunto diferente como conjunto de validación en cada una de las iteraciones, como se muestra en la Imagen 4.5.3.1.

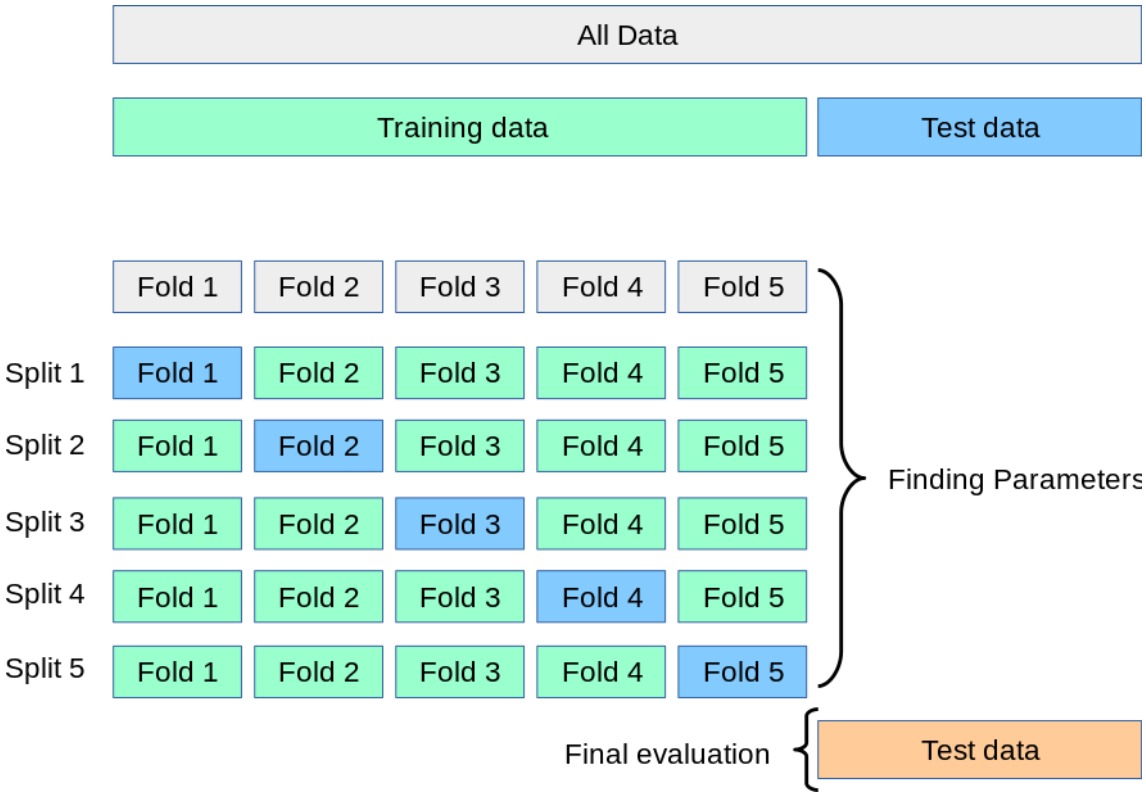


Imagen 4.5.3.1. Funcionamiento del cross-validation. XXVI->xiii

Una vez tenemos todo esto, comprobamos cuál de los 2 modelos obtiene mejores resultados mediante el modelo de Wilcoxon. Este modelo con los valores AUC de las iteraciones del cross validation (Imagen 4.5.3.2) calcula la probabilidad de que uno de los modelos sea más eficaz que el otro.

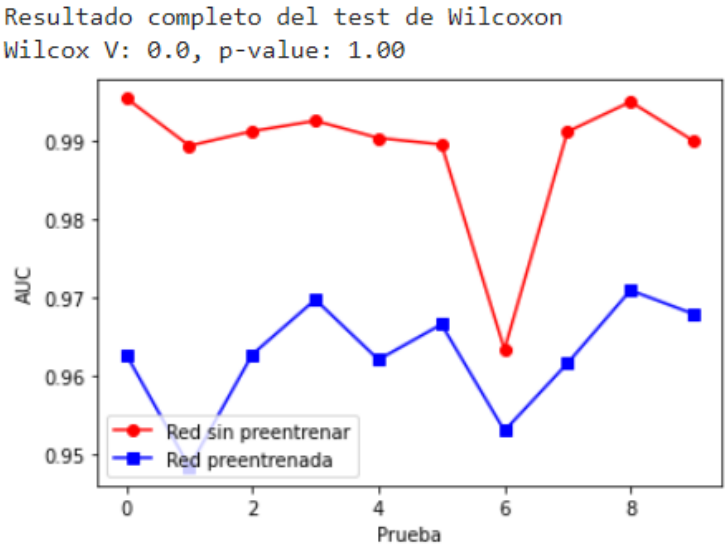


Imagen 4.5.3.2. Resultados del test de Wilcoxon.

Como observamos en la Imagen 4.5.3.2, el valor de p-value es 1, lo que implica que el modelo sin preentrenar tiene una probabilidad del 100% de obtener mejores resultados que el modelo preentrenado, por lo que este será nuestro modelo final.

Capítulo 5. Resultados

5.1 Resultados generales

5.1.1 Prueba parcial

Para obtener los resultados finales de nuestro proyecto utilizaremos el conjunto de test para obtener el rendimiento de nuestro modelo final. Como se ha comentado anteriormente, hemos dividido nuestro conjunto de datos en “train”, “validation” y “test”, por lo que para esta prueba usaremos los datos de test que nos darán un resultado más fiable, ya que son datos nuevos para nuestro modelo.

5.1.2 Resultado parcial

Accuracy: 94% Loss: 0.19 AUC: 0.9888

Tabla 5.1.2.1. Resultado de la prueba general.

5.1.3 Conclusión parcial

Como podemos observar, nuestro modelo únicamente falla 1 imagen de cada 20. Asumiendo que los datos pueden tener imágenes difíciles de clasificar y teniendo en cuenta la escasez de datos, tiempo y material es un resultado muy bueno para el objetivo que persigue el proyecto.

Además, veremos cómo funciona nuestro modelo en casos excepcionales.

5.2 Casos subrealistas

En esta sección comprobaremos los casos subrealistas, es decir situaciones en las que el modelo se enfrenta a escenarios poco realistas o fuera de contexto. Al proporcionar estas imágenes, se busca determinar cómo reacciona el modelo y qué tipo de respuestas produce.

5.2.1 Prueba parcial

En este caso, introduciremos imágenes de gatos como entrada al modelo y evaluaremos los resultados obtenidos. Sin embargo, nuestro modelo no ha sido entrenado para evaluar este tipo de imágenes, por lo que producirá respuestas completamente aleatorias.

5.2.2 Resultado parcial

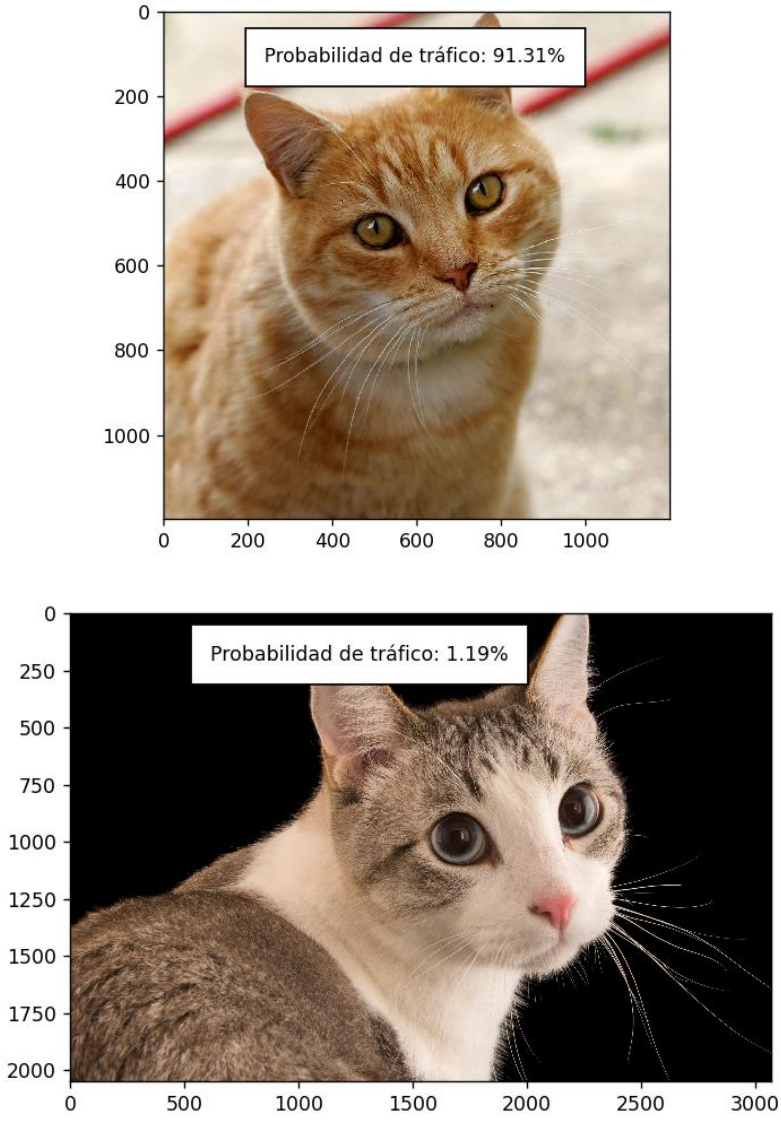


Imagen 5.2.2.1. Resultado de caso subrealista.

Como se muestra en la imagenes 5.2.2.1 nuestro modelo no puede analizar imágenes de gatos. Por tanto, los resultados obtenidos son completamente aleatorios y no están relacionados con la presencia o ausencia de tráfico.

5.2.3 Conclusión parcial

En conclusión, como es de esperar, en estos casos (Imagen 5.2.2.1) nuestro modelo devuelve resultados completamente aleatorios ya que el modelo pierde todas las referencias en las que se apoya para detectar o evaluar la presencia de tráfico en la imagen.

5.3 Elementos poco frecuentes

5.3.1 Prueba parcial

Otro caso para probar es cómo se comporta con imágenes en las que existen elementos que puedan confundir a nuestro modelo. En este caso hemos elegido trenes y graffitis, ya que nuestro modelo podría confundir la pared o la vía de tren con una carretera.

5.3.2 Resultado parcial

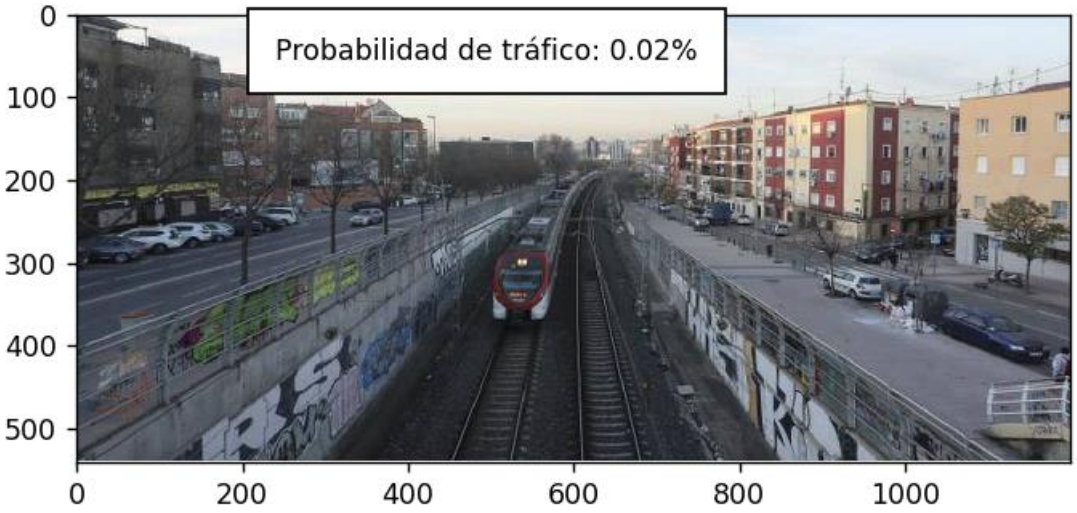


Imagen 5.3.2.1. Resultados de pruebas con elementos poco frecuentes.

5.3.3 Conclusión parcial

Efectivamente, como vemos en las imágenes 5.3.2.1, pueden existir ciertos elementos como trenes, graffitis o las vías del tren que confundan a nuestro modelo. Esto ocurre ya que en los datos con los que se ha entrenado no se han tenido en cuenta estas características.

5.4 Funcionamiento

5.4.1 Streamlit

Streamlit es una herramienta de código abierto que facilita la creación de aplicaciones web interactivas que está diseñada específicamente para el desarrollo de aplicaciones basadas en Python.

Su enfoque en la simplicidad y la facilidad de uso es la principal característica de Streamlit. Permite crear una interfaz de usuario interactiva para sus aplicaciones con Streamlit con poco código, lo que permitirá simplificar el desarrollo de nuestro proyecto.

La capacidad de actualizar automáticamente la interfaz de usuario a medida que se modifican los parámetros o se actualizan los datos es una de las características más importantes en nuestro proyecto, ya que esto permite una experiencia interactiva en tiempo real del tráfico en las carreteras de Madrid que se refleja en la aplicación web sin recargar la página.

Además, Streamlit tiene una función de recopilación de registros, que permite al desarrollador realizar un seguimiento de los eventos y acciones que se realizan en la aplicación. Esto puede ser beneficioso en nuestro proyecto para la depuración y la comprensión del flujo de trabajo de la aplicación mientras se desarrolla.

En resumen, Streamlit es una herramienta que facilita el desarrollo de aplicaciones web interactivas con Python. Ayuda a crear interfaces de usuario rápidas y fáciles, integrar visualizaciones y actualizar automáticamente la aplicación en tiempo real. Todas estas ventajas son las que nos hacen decidirnos por esta herramienta.

(FALTA Añadir bibliografía)

5.4.2 Explicación

(FALTA Añadir pies de imagen)

Para finalizar con este capítulo vamos a mostrar un ejemplo de funcionamiento.

Lo que haremos será obtener imágenes en tiempo real de la DGT (https://www.camarasdgt.es/madrid/) (Capítulo 2.2.4 Cámaras DGT), las cuales se actualizan cada varios minutos, para realizar un análisis del tráfico y, en caso de que exista una retención se generará una alarma en la aplicación. Por tanto, nuestra aplicación sigue los siguientes pasos:

1. Descarga las imágenes de la DGT para las carreteras en puntos estratégicos donde puede haber retenciones (Anexo 5):
    a. Carretera A-1 P.k.: Via de Servicio A-121.000
    b. Carretera M-50 P.k.: 17.500
    c. Carretera A-4 P.k.: 49.100 Pasado Aranjuez
    d. Carretera M-40 P.k.: 6.300 Silvano Madrid

Imagen 5.4.2.1

2. Analiza cada una de las imágenes y las clasifica haciendo uso de nuestro modelo en función de si hay o no retención (Anexo 6).
3. Muestra la probabilidad de que haya retención en cada una de las imágenes.

Imagen 5.4.2.2

4. En caso de tener un >90% de probabilidad se considera que, efectivamente, hay retención y salta una alarma en la aplicación (Anexo 7).

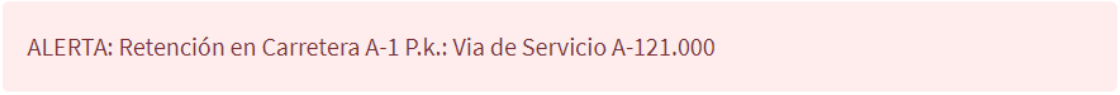


Imagen 5.4.2.3

5. Espera 2 minutos y actualiza las imágenes cargadas. Volvemos al paso 1.

Imagen 5.4.2.4

Para acceder a la aplicación, debemos hacer click en el siguiente enlace: https://hivansm-tfm-repo-srcejecutable-tfm-rlgr9c.streamlit.app/ .

Capítulo 6. CONCLUSIONES

En este proyecto se han estudiado técnicas muy importantes para el reconocimiento de imágenes. Se ha comenzado con un preprocesado de los datos reduciendo el tamaño y normalizando los valores para, más tarde generar más datos haciendo aumentado de datos.
Una vez tenemos los datos, hemos implementado y entrenado el modelo. Por último, hemos comprobado el funcionamiento del modelo y hemos obtenido el rendimiento de nuestro modelo.
Este rendimiento ha sido muy bueno teniendo en cuenta la escasez de datos, tiempo y material del que disponemos.
Podemos concluir que, para la estimación de tráfico, podemos obtener muy buenos resultados haciendo uso de las redes neuronales convolucionales.

Atendiendo a los objetivos propuestos en el capítulo 1 vemos que hemos cumplido todos los objetivos:

- Se ha logrado analizar el tráfico a partir de cámaras urbanas con gran precisión.
- Se ha llegado a la conclusión de que, efectivamente, las CNN son muy útiles en tares como la estimación de tráfico a partir de las imágenes recogidas por cámaras urbanas.
- Se ha llevado a cabo un estudio con los datasets disponibles explicando la utilidad de cada uno de ellos en nuestro proyecto.
- Hemos conseguido implementar un programa capaz de obtener datos que permitan generar estadísticas de tráfico para en un futuro proyecto trazar un plan de redirección de tráfico y preparar los servicios oportunos en las zonas más concurridas.

Como vemos, podemos concluir que el proyecto ha sido un éxito ya que hemos cumplido en gran medida con todos los objetivos propuestos.

Sin embargo, cabe destacar que todavía existen ocasiones en los que nuestro modelo se confunde y falla en la detección de tráfico.

Capítulo 7. Futuras mejoras

A pesar de los buenos resultados de este Trabajo Fin de Máster (TFM), es importante reconocer que el modelo y el enfoque utilizados pueden mejorarse en varios aspectos. Aunque no se han implementado en el trabajo actual debido a limitaciones temporales, estas mejoras se consideran útiles para mejorar el desempeño del sistema en futuros proyectos. Estos son algunos ejemplos de mejoras que podrían ser consideradas en trabajos futuros:

1. Ampliación de la base de datos: una de las mejoras más importantes sería la adquisición de datos de cámaras urbanas de varias carreteras en España y su etiquetado mediante clustering. La incorporación de esta gran cantidad de datos aumentaría la diversidad del conjunto de datos de entrenamiento y, por lo tanto, el rendimiento del modelo. El sistema podrá adquirir patrones más precisos y generalizar mejor en escenarios desconocidos al tener más ejemplos y situaciones diferentes.

2. Integración de modelos avanzados: Se sugiere considerar la incorporación de modelos avanzados como YOLO (You Only Look Once) para mejorar la precisión y la detección de objetos en las imágenes capturadas por cámaras urbanas. El modelo podría ser más capaz de identificar situaciones específicas, como la ausencia de vehículos en áreas clasificadas como "Con tráfico" utilizando técnicas de detección de objetos. Esto aumentaría la confiabilidad de las etiquetas y, por lo tanto, la calidad de los resultados.

3. Generación de estadísticas y planificación de servicios: Después de implementar las mejoras anteriores, el modelo podrá generar estadísticas más precisas y detalladas para una variedad de lugares. Se podrían utilizar estas estadísticas para diseñar planes de redirección de tráfico y crear servicios adecuados en áreas con alta afluencia de vehículos. Uno de los objetivos principales de esta fase de mejora sería reducir el número de accidentes y la mortalidad en áreas de alta concurrencia.

4. Evaluación y comparación de resultados: Será esencial evaluar minuciosamente el modelo mejorado y comparar sus resultados con otros métodos utilizados anteriormente. Esto permitirá establecer métricas de mejora claras y verificar si las implementaciones propuestas funcionan.

En resumen, las mejoras propuestas para el futuro tienen como objetivo mejorar el modelo actual al aumentar la cantidad y la diversidad de datos, incorporar métodos de detección de objetos y utilizar las estadísticas generadas para la planificación de servicios y disminuir los accidentes. Se espera que estas mejoras hagan un gran aporte al progreso del proyecto y al cumplimiento de los objetivos establecidos. Es importante destacar que, aunque se requerirá una planificación cuidadosa y dedicación de recursos para llevar a cabo estas mejoras, los resultados serán de gran valor para futuras aplicaciones en el ámbito del control y la gestión del tráfico.

ANEXOS

```
IMAGE_SIZE = 32
BATCH_SIZE = 64

# Data augmentation
datagen = ImageDataGenerator(rescale=(1/255))

train_generator = datagen.flow_from_directory(
  dir_img_only,
  target_size=(IMAGE_SIZE, IMAGE_SIZE),
  batch_size=-1,
  shuffle = False,
  class_mode = 'binary',
  classes = ['no_traffic','traffic']
)
```

Anexo 1

```
def data_augmentation(X_train,y_train,batch_size):
  image_gen=ImageDataGenerator(
    rotation_range=40,  #Rota imagenes aleatoriamente en un rango
de 40° [-40 - 40]
    width_shift_range=0.2,  #Desplaza imagenes aleatoriamente
    height_shift_range=0.2, #Desplaza imagenes aleatoriamente
    shear_range=0.2,  #Recorta imagenes aleatoriamente
    zoom_range=0.2, #Zoom en imagenes aleatoriamente
    horizontal_flip=True,
    fill_mode='nearest')

  data_augmented=image_gen.flow(X_train, y_train,
batch_size=batch_size,seed=1)

  return data_augmented
```

Anexo 2

```
def cnn_model2():
  visible = layers.Input(shape=X[0].shape)
  x = layers.Conv2D(16,(3,3),padding="same",activation="relu")(visi
ble)
  x = layers.MaxPooling2D((2,2))(x)

  x = layers.Conv2D(32,(3,3),padding="same",activation="relu")(x)
  x = layers.MaxPooling2D((2,2))(x)

  x = layers.Conv2D(64,(3,3),padding="same",activation="relu")(x)
  x = layers.MaxPooling2D((2,2))(x)
```

```
  x = layers.Flatten()(x)

  x = layers.Dense(512,activation="relu")(x)

  x = layers.Dense(256,activation="relu")(x)

  x = layers.Dense(64,activation="relu")(x)

  x = layers.Dense(32,activation="relu")(x)

  output = layers.Dense(1, activation='sigmoid')(x)

  model = keras.Model(inputs=visible, outputs=output)

  return model
```

Anexo 3

```
def cnn_model():
  #Descargamos el modelo
  URL = 'https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/4'
  feature_extractor =  hub.KerasLayer(
      URL,
      input_shape=X[0].shape
  )

  #Bloqueamos el entrenamiento de los pesos ya entrenados de esta red
  feature_extractor.trainable=False

  #Adaptamos la red a nuestras necesidades
  model = tf.keras.models.Sequential([
    feature_extractor,
    tf.keras.layers.Dense(1,activation='sigmoid')
  ])

  return model
```

Anexo 4

```
url = [ 'https://www.camarasdgt.es/madrid/a-1/km-21-000-a-121000/',
            'https://www.camarasdgt.es/madrid/m-50/km-17-500/',
            'https://www.camarasdgt.es/madrid/a-4/km-49-100/',
            'https://www.camarasdgt.es/madrid/m-40/km-6-300/'] # URL de la página web de la DGT


    response = [requests.get(url[0]),
```

```
                requests.get(url[1]),
                requests.get(url[2]),
                requests.get(url[3])] # Hacer una solicitud HTTP GET

    soup = [BeautifulSoup(response[0].content, 'html.parser'),
            BeautifulSoup(response[1].content, 'html.parser'),
            BeautifulSoup(response[2].content, 'html.parser'),
            BeautifulSoup(response[3].content, 'html.parser')] # Analizar
el HTML con BeautifulSoup


    # Encontrar la URL de la imagen más reciente
    img = [ soup[0].find_all('img', {'title': 'Cámara Carretera A-1 P.k.:
 Via de Servicio A-121.000'})[0],
            soup[1].find_all('img', {'title': 'Cámara Carretera M-50
P.k.:  17.500'})[0],
            soup[2].find_all('img', {'title': 'Cámara Carretera A-4 P.k.:
 49.100 Pasado Aranjuez '})[0],
            soup[3].find_all('img', {'title': 'Cámara Carretera M-40
P.k.:  6.300 Silvano madrid'})[0]]


    img_url = [img[0]['src'],
               img[1]['src'],
               img[2]['src'],
               img[3]['src']]

    # Descargar la imagen
    bytes_data = [requests.get(img_url[0]).content,
                  requests.get(img_url[1]).content,
                  requests.get(img_url[2]).content,
                  requests.get(img_url[3]).content]

    image = [Image.open(io.BytesIO(bytes_data[0])),
            Image.open(io.BytesIO(bytes_data[1])),
            Image.open(io.BytesIO(bytes_data[2])),
            Image.open(io.BytesIO(bytes_data[3]))]
```

Anexo 5

```
model = keras.models.load_model('models/model04.h5')
prob_traf = [model.predict(img[0].reshape((1,32,32,3)))[0][0],
             model.predict(img[1].reshape((1,32,32,3)))[0][0],
             model.predict(img[2].reshape((1,32,32,3)))[0][0],
             model.predict(img[3].reshape((1,32,32,3)))[0][0]]
```

Anexo 6

```
    for i in range(4):
        if(prob_traf[i] > 0.9):
            st.error(f'ALERTA: Retención en {carreteras[i]}')
```

Anexo 7

BIBLIOGRAFÍA

- [i] "Understanding of Convolutional Neural Network (CNN) — Deep Learning." *Medium*, 4 March 2018, https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. Accessed 31 July 2022.

- [ii] "Estimation of Traffic Density using Street Images." *UPCommons*, https://upcommons.upc.edu/bitstream/handle/2117/114169/128294.pdf?sequence=1&isAllowed=y. Accessed 22 August 2022.

- [iii] "Florida Traffic Cameras | View Live Florida Traffic Cameras." *FL511*, https://fl511.com/cctv?start=0&length=10&order%5Bi%5D=1&order%5Bdir%5D=asc. Accessed 22 August 2022.

- [iv] Pereira, Jonathan. "Perceptrón Multicapa y Algoritmo de Retropropagación - Artículos sobre MQL5." *MQL5*, https://www.mql5.com/es/articles/8908 . Accessed 31 July 2022.

- [v] "Pothole and Plain Road Images." *Kaggle*, https://www.kaggle.com/datasets/virenbr11/pothole-and-plain-rode-images. Accessed 22 August 2022.

- [vi] dgt, www.dgt.es, Altran. (s. f.-b). DGT - Cámaras de tráfico. https://www.dgt.es/conoce-el-estado-del-trafico/camaras-de-trafico/

- [vii] *Tema 8. Redes Neuronales*, http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf . Accessed 31 July 2022.

- [viii] Ma, Xiaolei & Dai, Zhuang & He, Zhengbing & Max Jihui & Wang, Yong & Wang, Yunpeng. (2017). Learning Traffic as Images: A Deep Convolutional Neural Network for Large-Scale Transportation Network Speed Prediction. Sensors. 17. 818. 10.3390/s17040818.

- [ix] López-Sastre, Roberto & Herranz-Perdiguero, Carlos & Guerrero-Gómez-Olmedo, Ricardo & Oñoro, Daniel & Maldonado-Bascón, Saturnino. (2019). Boosting Multi-Vehicle Tracking with a Joint Object Detection and Viewpoint Estimation Sensor. Sensors. 19. 4062. 10.3390/s19194062.

- [x] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

- [xi] Bekele, Bedada (2020), "Traffic congestion Dataset", Mendeley Data, V1, doi: 10.17632/wtp4ssmwsd.1

- [xii] *ImageNet*. http://www.image-net.org

- [xiii] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

- [xiv] F. Chollet et al. Keras. https://keras.io, 2015.

- [xv] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

- [xvi] Lundberg, Scott M., and Su-In Lee. "A Unified Approach to Interpreting Model Predictions." Advances in Neural Information Processing Systems 30, edited by I. Guyon et al., Curran Associates, Inc., 2017, pp. 4765–74, http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

- [xvii] Bisong, Ekaba. "Google Colaboratory." Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, Apress, 2019, pp. 59–64, doi:10.1007/978-1-4842-4470-8_7.

- [xviii] Cross-validation: evaluating estimator performance. (s. f.). scikit-learn. https://scikit-learn.org/stable/modules/cross_validation.html

- [xix] Bagnato, Juan Ignacio. "Convolutional Neural Networks: La Teoría explicada en Español." *Aprende Machine Learning*, 29 November 2018,

https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/ . Accessed 31 July 2022.

- [xx] Gandhi, Arun. "Data Augmentation | How to use Deep Learning when you have Limited Data — Part 2." *Nanonets*, 19 May 2021, https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/ . Accessed 31 July 2022.

- [xxi] *Tema 3: Filtros*, http://alojamientos.us.es/gtocoma/pid/tema3-1.pdf . Accessed 31 July 2022.

- [xxii] "The train accuracy of a detection model (supervised, NN) after 10 epochs is 100% whereas the test accuracy is only 82%. Am I overfitting the model?" *Quora*, https://www.quora.com/The-train-accuracy-of-a-detection-model-supervised-NN-after-10-epochs-is-100-whereas-the-test-accuracy-is-only-82-Am-I-overfitting-the-model . Accessed 31 July 2022.

-