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 2021-2022

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: Raúl Pérula Martínez

FECHA: Septiembre de 2022

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

RESUMEN 4

ABSTRACT 5

Capítulo 1. INTRODUCCIÓN 12

1.1 Visión General 12

1.2 Motivación 12

1.3 Objetivos 12

1.4 Timeline 12

1.4.1 Fase inicial 13

1.4.2 Fase de preparación 13

1.4.3 Fase de desarrollo 13

1.4.4 Fase de transición 13

1.5 Estructura del documento 13

Capítulo 2. Estado del Arte 15

2.1 Proyectos Relacionados 15

2.1.1 Learning Traffic as Images: A Deep Convolutional Neural Network for Large-Scale Transportation Network Speed Prediction 15

2.1.2 Estimating Urban Traffic Density Using Street Camera Images 16

2.1.3 Boosting Multi-Vehicle Tracking with a Joint Object Detection and Viewpoint Estimation Sensor ..... **¡Error! Marcador no definido.**

2.2 Datasets 17

2.2.1 Big Data for Traffic Estimation and Prediction 17

2.2.2 Traffic Congestion Dataset 18

2.2.3 Photole and Plain Road Dataset 18

Capítulo 3. IMPLEMENTACIÓN DE LA SOLUCIÓN 20

3.1 Arquitectura 20

3.1.2 Método propuesto 22

3.2 Datos 22

3.2.1 Ingesta de datos 22

3.2.2 Preprocesado 23

3.2.3 Aumentado de datos 24
3.3 Modelo y parámetros 24
3.3.1 Arquitectura de la red 24
3.3.2 Overfitting 26
3.4 Experimentación 28
3.4.1 SHAP 28
3.4.2 Comparación de modelos 29
Capítulo 4. Resultados 31
4.1 Resultados generales 31
4.1.1 Prueba parcial 31
4.1.2 Resultado parcial 31
4.1.3 Conclusión parcial 31
4.2 Casos subrealistas 31
4.2.1 Prueba parcial 31
4.2.2 Resultado parcial 32
4.2.3 Conclusión parcial 32
4.3 Elementos poco frecuentes 32
4.3.1 Prueba parcial 32
4.3.2 Resultado parcial 33
4.3.3 Conclusión parcial 33
4.4 Funcionamiento 33
Capítulo 5. CONCLUSIONES 35
Capítulo 6. Futuras mejoras 36
ANEXOS 37
BIBLIOGRAFÍA 39

Índice de Figuras

Imagen 1.4.1. Timeline general.

Imagen 1.4.2.1. Fase de preparación.

Imagen 1.4.3.1. Fase de desarrollo.

Imagen 1.4.4.1 Fase de transición

Imagen 2.1.1.1 Learning traffic images.

Imagen 2.1.1.2. Comparativa de algoritmos.

Imagen 2.2.1.1. Cameras Florida.

Imagen 2.2.2.1 Traffic Congestión Dataset

Imagen 2.2.3.1. Photole and Plain Road Dataset

Imagen 3.1.1. Arquitectura utilizada.

Imagen 3.1.1.2.1. Funcionamiento SHAP.

Imagen 3.2.1.1 Jerarquía de los datos.

Imagen 3.2.2.1. Reducción de la imagen.

Imagen 3.2.3.1. Aumentado de datos.

Imagen 3.3.1.1.1 Red sin preentrenar.

Imagen 3.3.1.1.2. Extracción de características.

Imagen 3.3.1.1.3. Estudio de las características.

Imagen 3.3.2.1. 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. [vii]

Imagen 3.3.2.2. Comparación entre el accuracy del conjunto de entrenamiento y de validación.

Imagen 3.4.1.1. Resultados SHAP.

Imagen 3.4.2.1. Funcionamiento del cross-validation.

Imagen 3.4.2.2. Resultados del test de Wilcoxon.

Imagen 4.2.2.3. Resultado de caso subrealista.

Imagen 4.3.2.1. Resultados de pruebas con elementos poco frecuentes.

Imagen 4.4.1. Funcionamiento de la aplicación.

Índice de Tablas

Tabla 2.1.2.1. Comparativa de modelos.

Tabla 4.1.2.1. Resultado de la prueba general.

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

1.4 Timeline

Hemos dividido nuestra línea de tiempo (ver Figura 1.1) 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

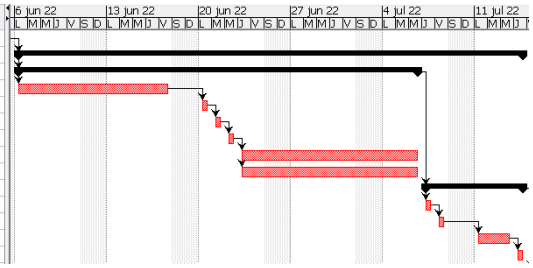


Imagen 1.4.2.1. Fase de preparación.

Esta segunda fase (Figura 1.2) 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 refina	14 days	29/08/22 8:00	15/09/22 17:00

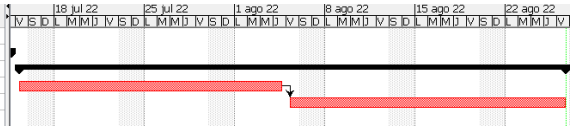


Imagen 1.4.3.1. Fase de desarrollo.

Una vez que hayamos elegido cada modelo y conjunto de datos, podemos comenzar a implementar el código. En esta fase(Figura 1.3), 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

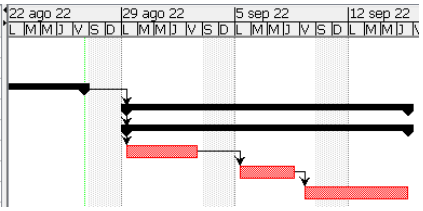


Imagen 1.4.4.1 Fase de transición

En esta última fase (Figura 1.4), 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.

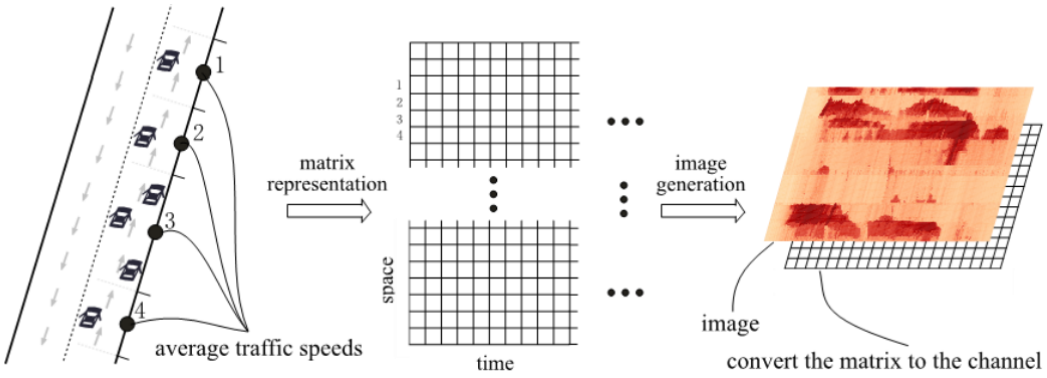
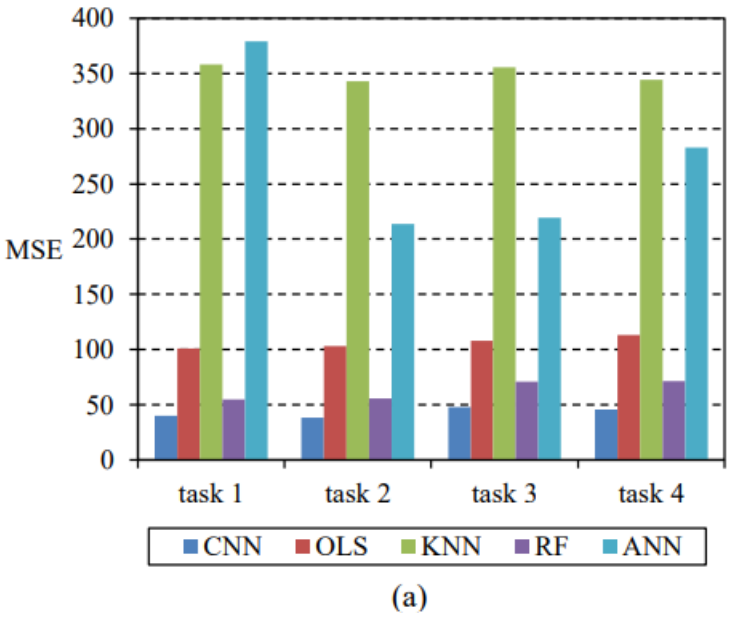


Imagen 2.1.1.1 Learning traffic images. [viii]

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]

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]

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

"Cameras Florida" [x] consiste en un dataset que puede ser de gran utilidad en nuestro proyecto. Este dataset contiene los registros de 3.668 cámaras ubicadas en Florida, por lo que es una importante fuente de datos de la cual podríamos obtener varios frames de cada una de las cámaras para llevar a cabo el entrenamiento de nuestro modelo.

Este dataset no lo hemos elegido debido a la complicación de etiquetar esa cantidad de datos con el tiempo del que disponemos, pero es una buena opción para futuras ampliaciones del proyecto.

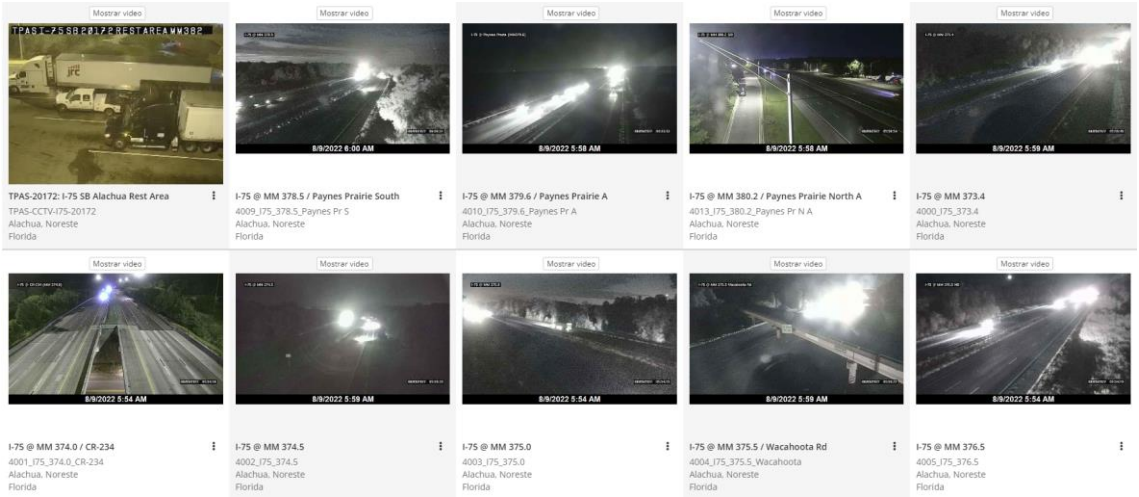


Imagen 2.2.1.1. Cameras Florida. [x]

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 [xi] consiste en 4.238 frames de cámaras urbanas de 500 x 500 etiquetadas en función de si encontramos una gran congestión en la carretera o no.

Imagen 2.2.2.1 Traffic Congestión Dataset [xi]

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.

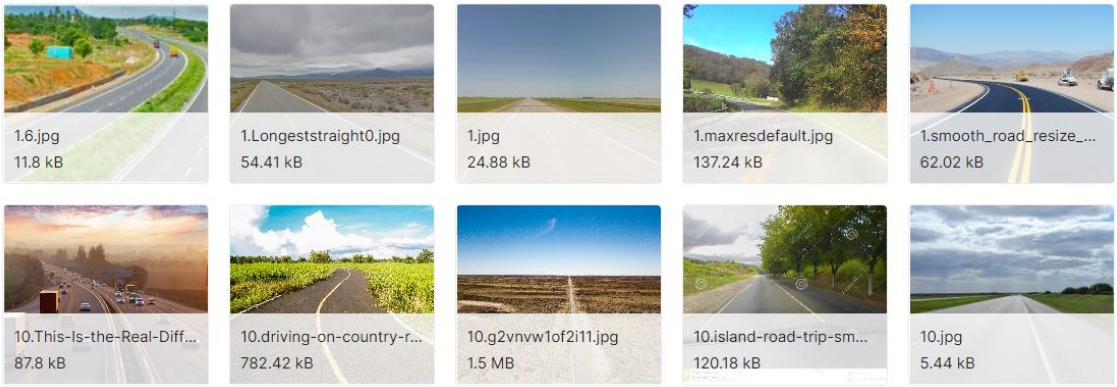


Imagen 2.2.3.1. Photole and Plain Road Dataset [xii]

Capítulo 3. IMPLEMENTACIÓN DE LA SOLUCIÓN

3.1 Arquitectura

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

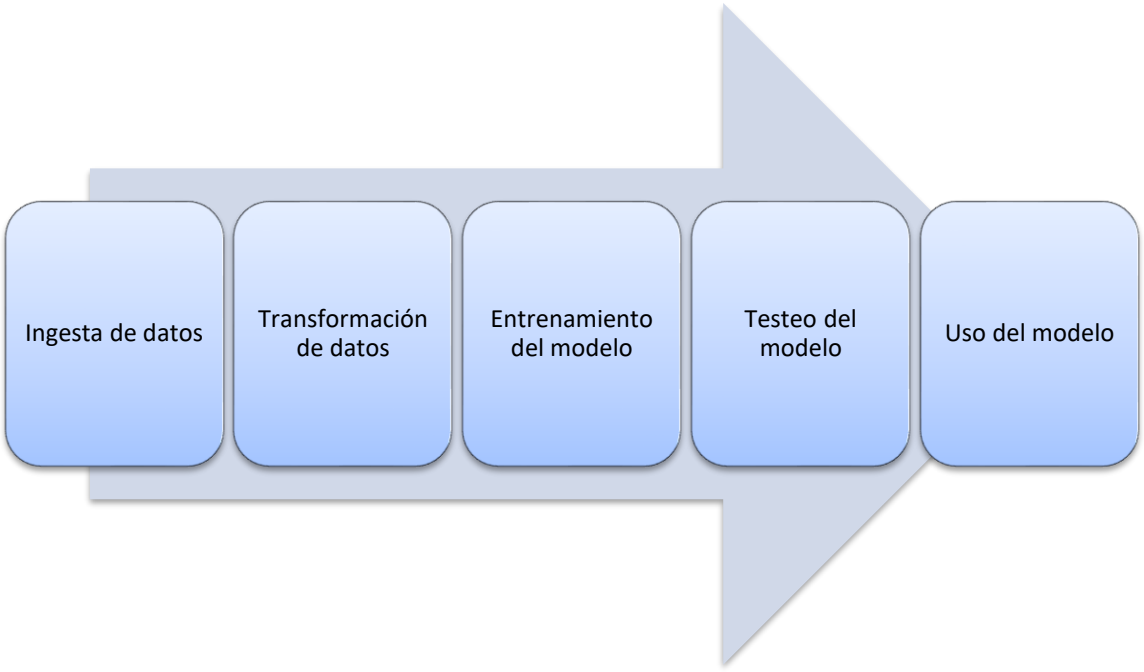


Imagen 3.1.1. Arquitectura utilizada.

Comenzaremos ingestando los datos comentados en el capítulo 3 “Datasets”.

3.1.1.1 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], un importante y poderoso framework destinado al Aprendizaje Automático.

Hemos seleccionado "Keras" [2] porque es muy popular, flexible y habíamos trabajado con él antes. 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 estás transformaciones se incluyen reducción del tamaño, aumentado de datos y demás transformaciones explicadas en el punto 3.2.

Una vez lo datos están preparados, se introducirán en la infraestructura propuesta en el punto 3.3, 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.

3.1.1.2 SHAP

Imagen 3.1.1.2.1. Funcionamiento SHAP.

Ya con el modelo entrenado, se testeará su funcionamiento aplicando técnicas como SHAP [xxv], 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 3.1.1.2.1). 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.

3.1.1.3 Google Colaboratory

Para ejecutar este flujo de datos ha sido necesario utilizar un hardware que nos permita llevarlo a cabo [xxii]. 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

3.1.1.4 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

3.1.1.5 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.

3.1.2 Método propuesto

El primer método que se propuso no fue llevado a cabo. Consiste en hacer uso de YOLOv3, 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 la 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.

3.2 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”.

3.2.1 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 3.2.1.1 Jerarquía de los datos.

3.2.2 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.

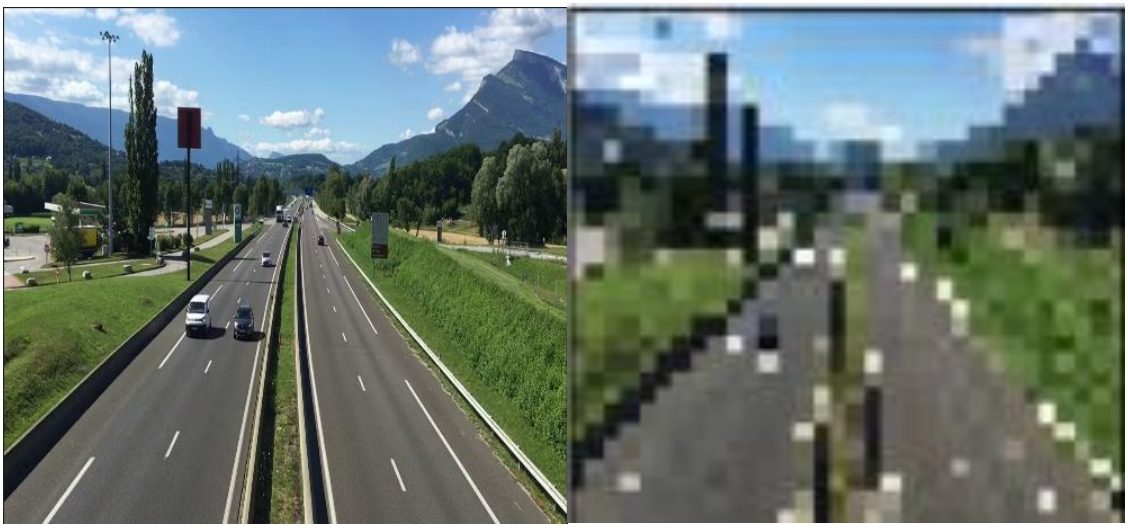


Imagen 3.2.2.1. Reducción de la imagen.

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

3.2.3 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 3.1.1.1, para realizar pequeñas modificaciones en cada una de las fotos como realizar zoom, volteos, desplazamientos… De esta forma hemos obtenido una mayor cantidad de datos y haciendo nuestro modelo más efectivo.

Imagen 3.2.3.1. Aumentado de datos.

3.3 Modelo y parámetros

3.3.1 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.

3.3.1.1 Red sin preentrenar

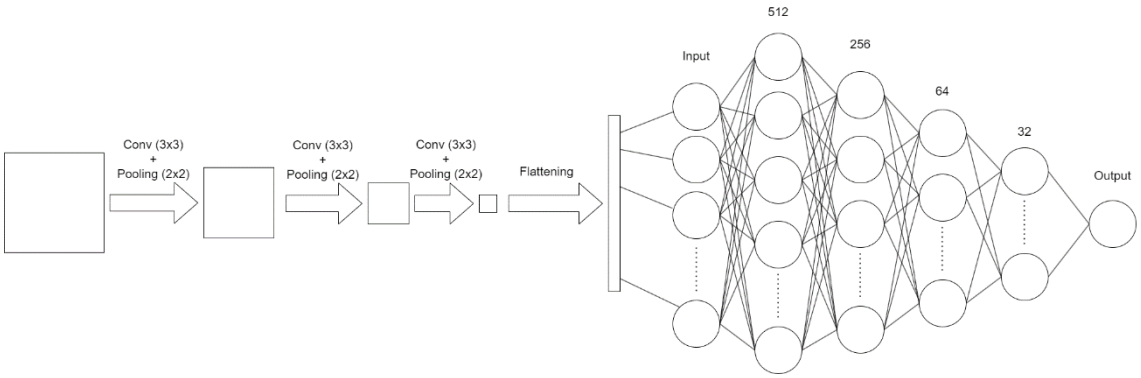


Imagen 3.3.1.1.1 Red sin preentrenar.

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 1)

Extracción de características

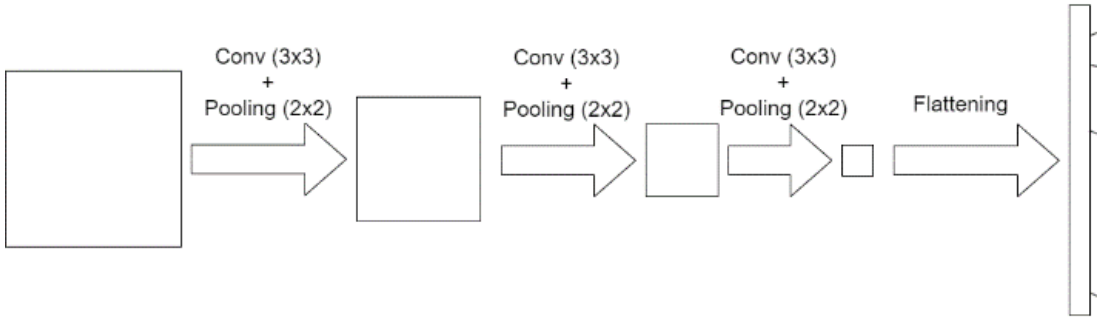


Imagen 3.3.1.1.2. Extracción de características.

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

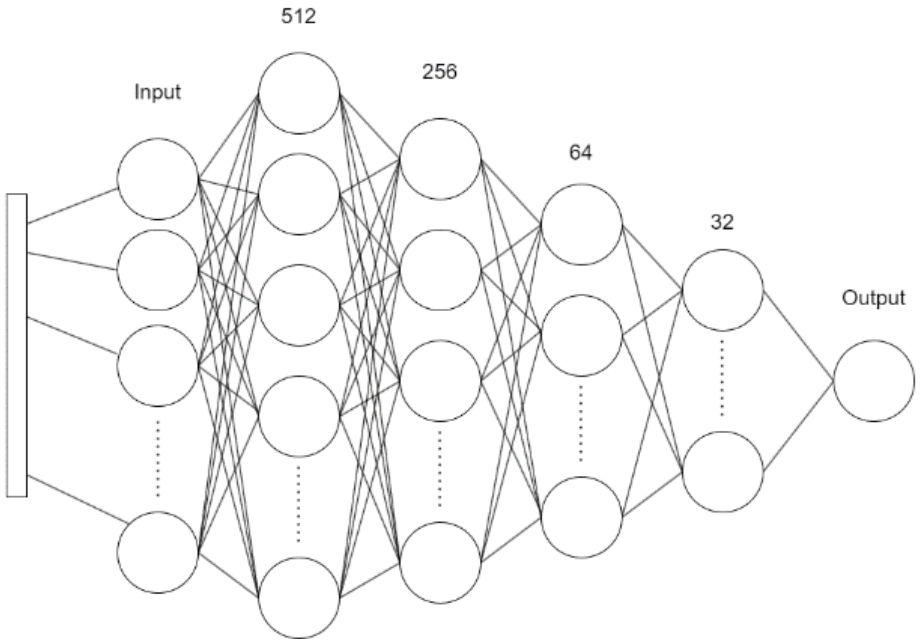


Imagen 3.3.1.1.3. Estudio de las características.

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.

3.3.1.2 Red preentrenada

Otro modelo que hemos probado ha sido haciendo uso de la red preentrenada "ResNet50" (Anexo 2), un modelo explicado en el punto 3.3.1.2.1. Simplemente, hemos añadido una capa de salida de 1 única neurona y con la función de activación sigmoide para adaptarlo a nuestras necesidades.

3.3.1.2.1 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.

3.3.2 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 16) y comparando el accuracy de los datos de entrenamiento con los datos de test.

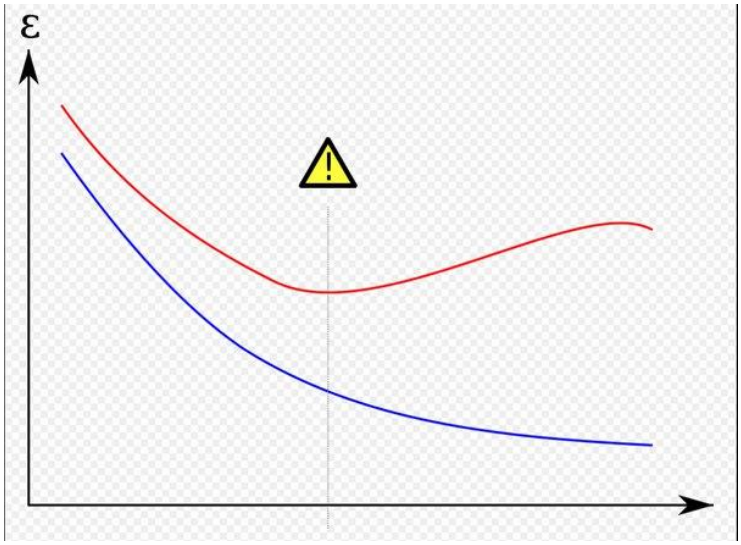


Imagen 3.3.2.1. Detectando el overfitting mediante el accuracy. La línea roja es el accuracy del conjunto de test mientras la línea azul es el accuracy del conjunto de entrenamiento. [vii]

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.

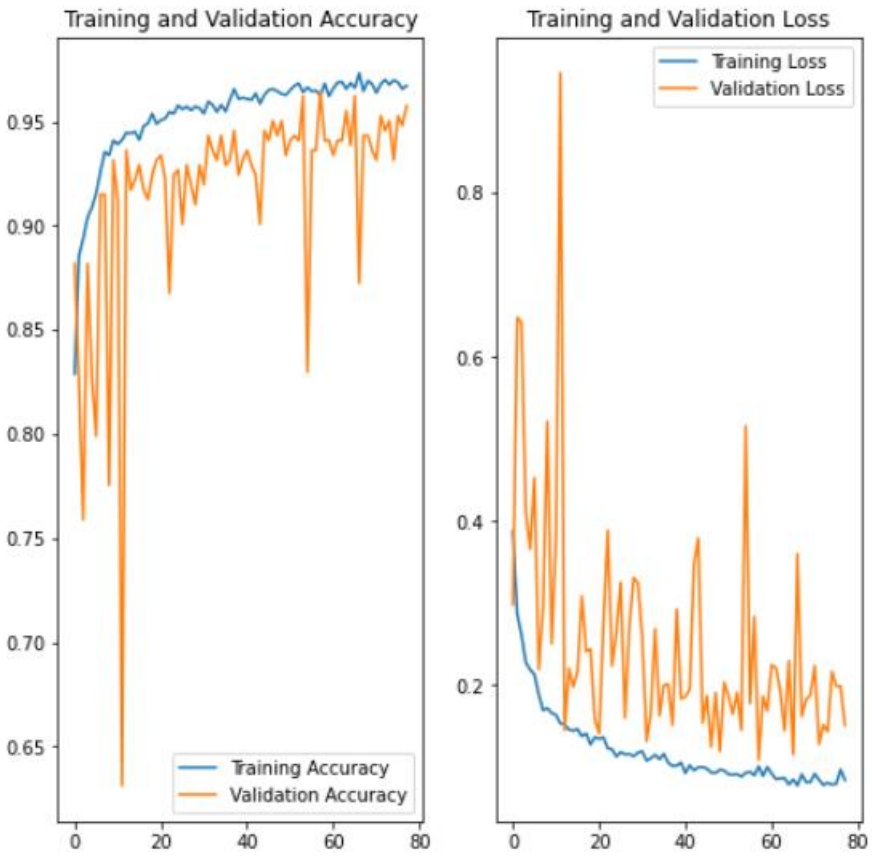


Imagen 3.3.2.2. Comparación entre el accuracy del conjunto de entrenamiento y de validación.

En la Figura 3.3.2.2 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.

3.4 Experimentación

3.4.1 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 3.1.1.2), una herramienta que nos facilitará esta tarea.

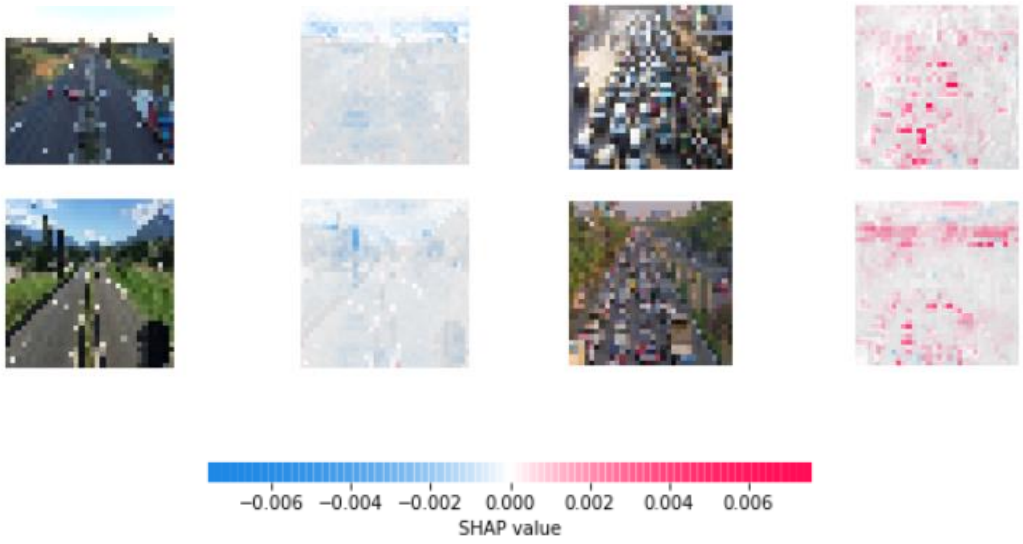


Imagen 3.4.1.1. Resultados SHAP.

Como vemos en la Figura 3.4.1.1, 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.

3.4.2 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 Figura 3.4.2.1.

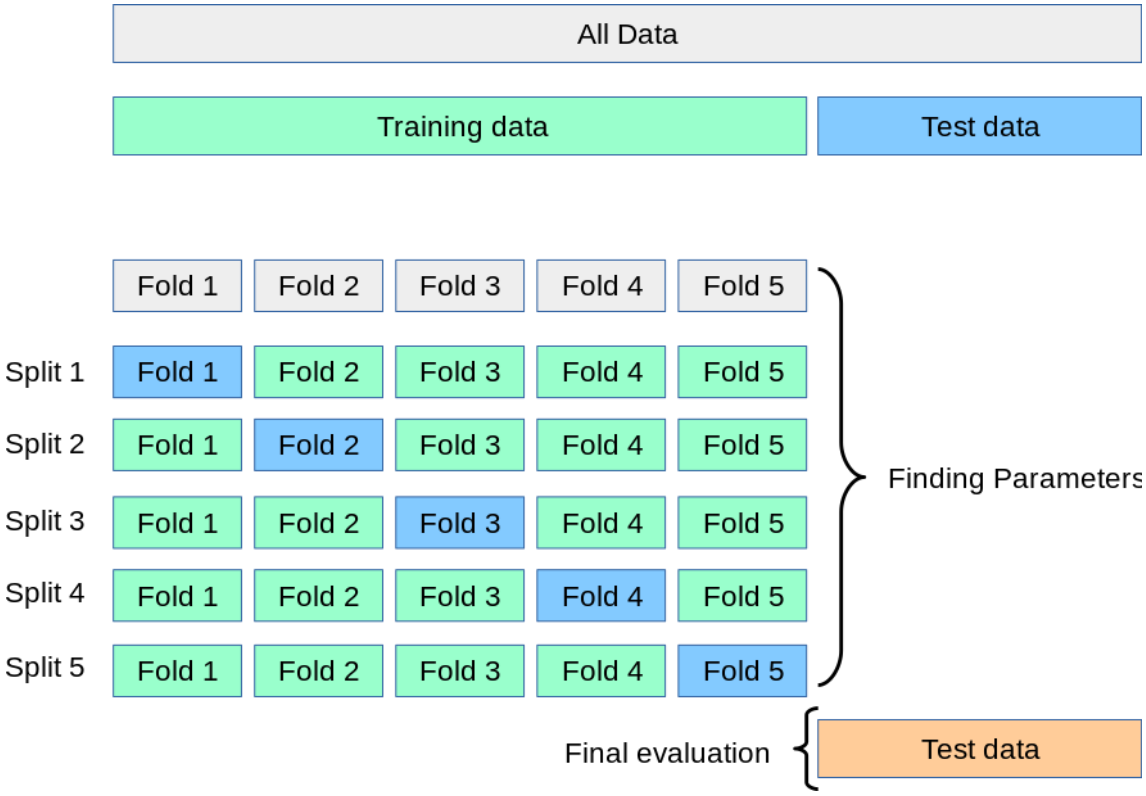


Imagen 3.4.2.1. Funcionamiento del cross-validation. https://scikit-learn.org/stable/modules/cross_validation.html

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 3.4.2.2) calcula la probabilidad de que uno de los modelos sea más eficaz que el otro.

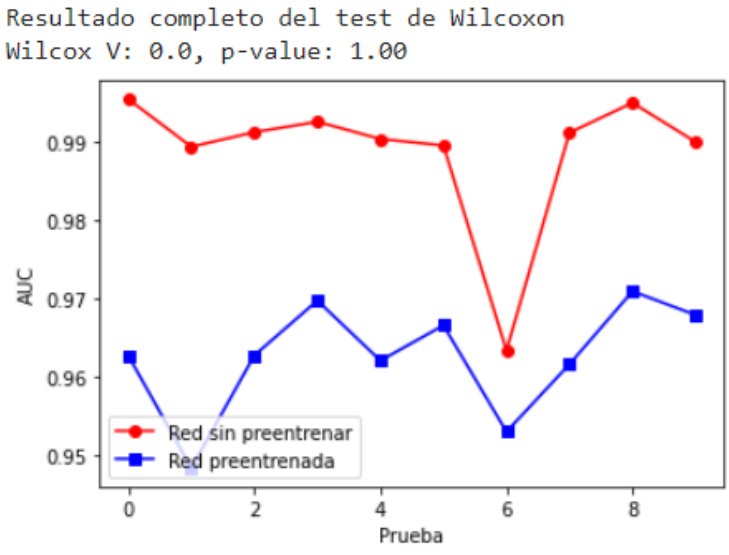


Imagen 3.4.2.2. Resultados del test de Wilcoxon.

Como observamos en la imagen 3.4.2.2, 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 4. Resultados

4.1 Resultados generales

4.1.1 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.

4.1.2 Resultado parcial

Accuracy: 94%
Loss: 0.19
AUC: 0.9888

Tabla 4.1.2.1. Resultado de la prueba general.

4.1.3 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.

4.2 Casos subrealistas

4.2.1 Prueba parcial

Para esto comprobaremos primero casos subrealistas de forma aleatoria. En este caso, introducir imágenes de gatos y comprobaremos el resultado.

4.2.2 Resultado parcial

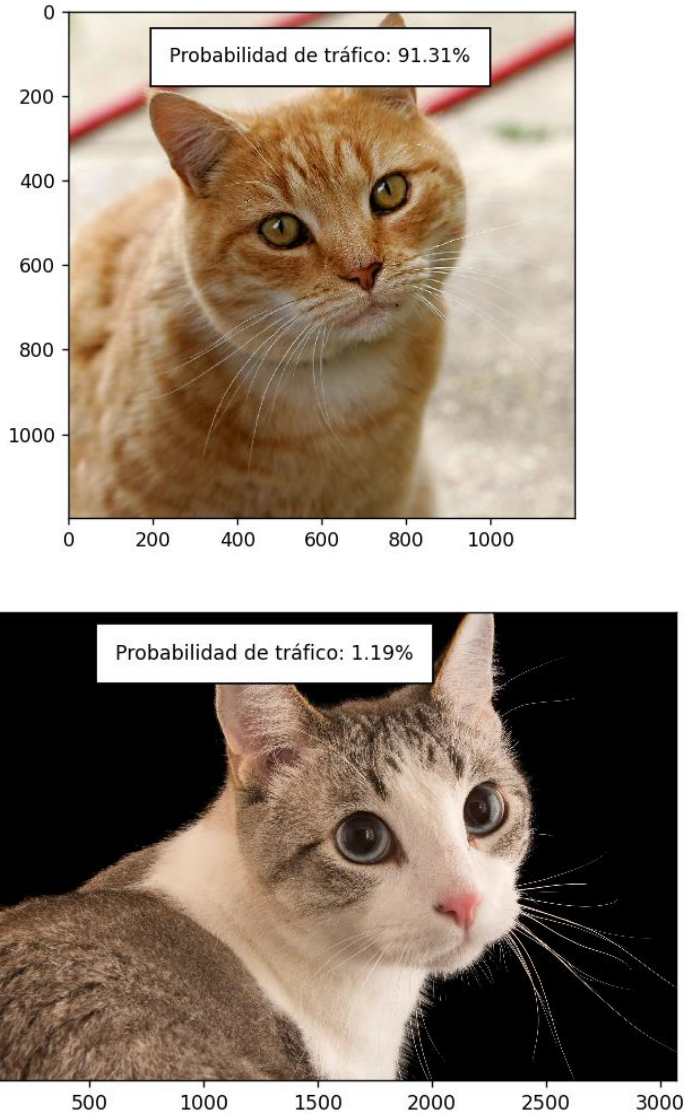


Imagen 4.2.2.3. Resultado de caso subrealista.

4.2.3 Conclusión parcial

Como es de esperar, en estos casos devuelve resultados completamente aleatorios ya que no el modelo pierde todas las referencias en las que se apoya para detectar si existe tráfico en la imagen o no.

4.3 Elementos poco frecuentes

4.3.1 Prueba parcial

Otro caso a 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.

4.3.2 Resultado parcial

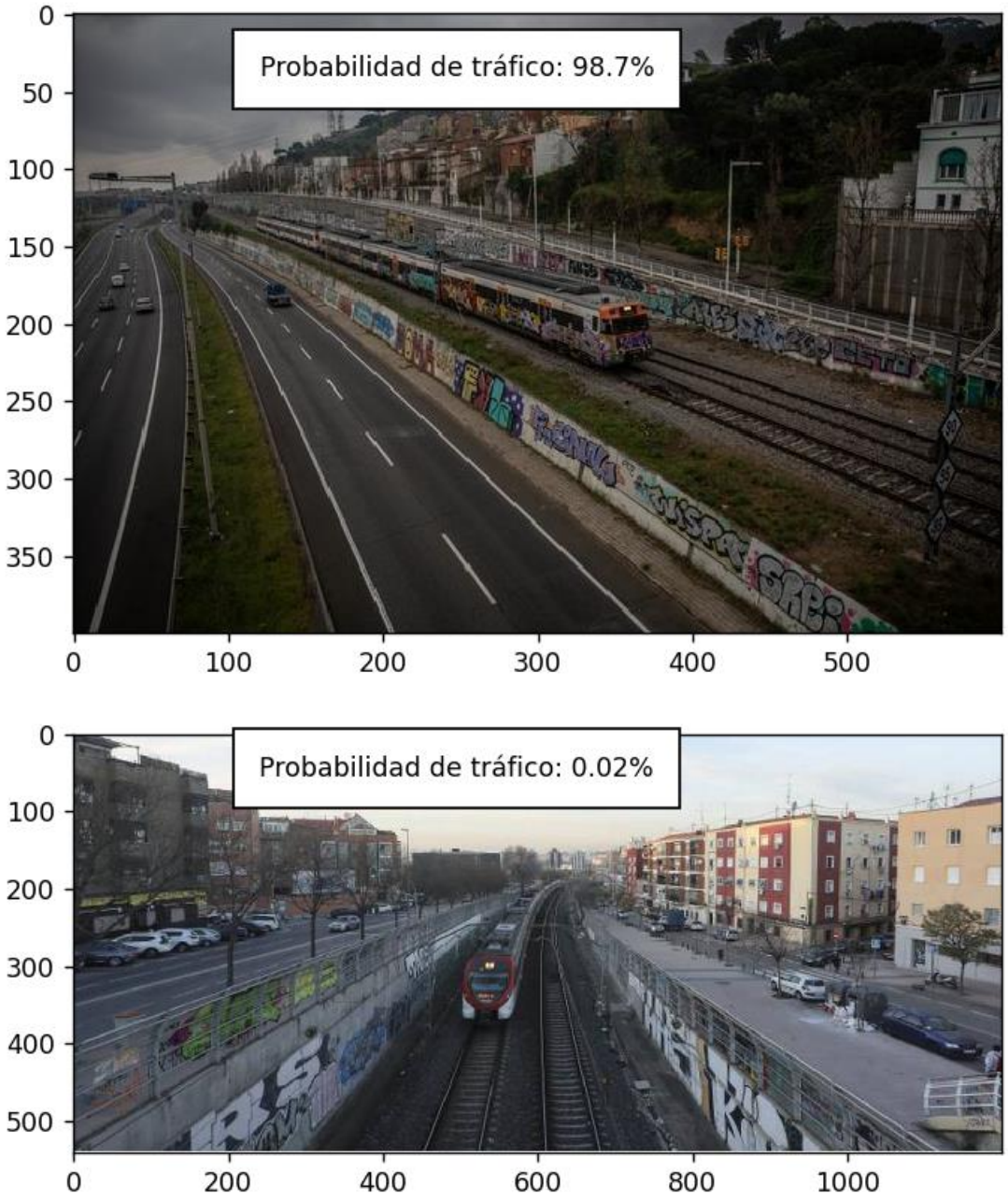


Imagen 4.3.2.1. Resultados de pruebas con elementos poco frecuentes.

4.3.3 Conclusión parcial

Efectivamente, como vemos en las imágenes, 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.

4.4 Funcionamiento

Para finalizar con este capítulo vamos a mostrar un ejemplo de funcionamiento. Simplemente, entraremos en el enlace "https://hivansm-tfm-repo-srcejecutable-tfm-eitvk2.streamlitapp.com/", subiremos una imagen y pulsaremos en el botón de "Predecir".

Predict

Probabilidad de tráfico: 0.024634084547869 86%

Imagen 4.4.1. Funcionamiento de la aplicación.

Capítulo 5. 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.
- Se ha comparado nuestra propuesta con la propuesta de utilizar herramientas de reconocimiento de objetos, como YOLOv3.
- 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 6. Futuras mejoras

A pesar de que los resultados obtenidos son buenos existen posibles mejoras que no se han aplicado debido al factor limitante del tiempo.

Una propuesta para un futuro trabajo es obtener los datos de las cámaras urbanas de varios estados de EEUU y etiquetarlos ayudándonos del clustering. Una vez tengamos los datos, dispondremos de una cantidad inmensa que implicarán una importante mejora del rendimiento. Otra propuesta sería ayudarnos de un modelo como YOLOv3 para terminar de afinar el resultado, por ejemplo si en una imagen clasificada como “Con tráfico” no se detecta ningún vehículo seguramente no estará bien etiquetada.

Tras aplicar estas mejoras a nuestro modelo podemos avanzar en el proyecto y comenzar a generar estadísticas de varias localizaciones distintas, con esto lograremos tener un dataset con el cuál podremos trazar un plan de redirección de tráfico y preparar los servicios oportunos en las zonas más concurridas reduciendo así el número de accidentes y la mortalidad de estos considerablemente.

ANEXOS

```
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 1

```
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 2

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

- [iii] 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.

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

- [v] 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.

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

- [vii] "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.

- [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] "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.

- [x] 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.

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

- [xii] 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.

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

- [xix] "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.

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

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

- [xxii] 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.

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

- [xxiv] 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.

- [xxv] 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.