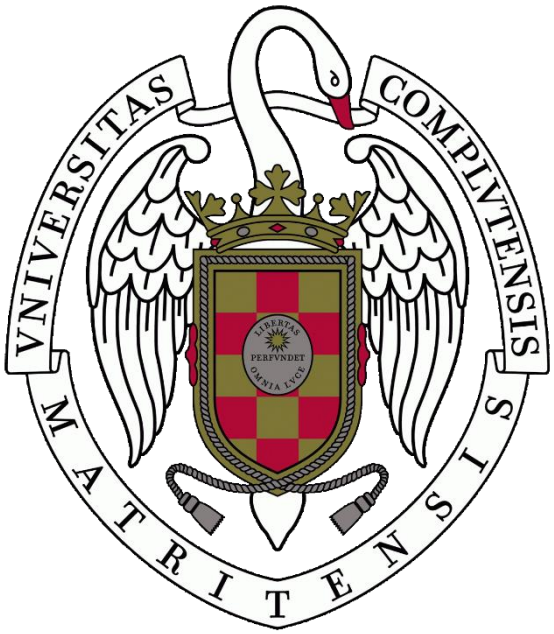
Universidad Complutense de Madrid

Facultad de Informática

Detección de vehículos en movimiento en vídeos mediante técnicas de aprendizaje profundo

Detection of moving vehicles in videos using deep learning techniques



TRABAJO DE FIN DE GRADO EN INGENIERÍA DEL SOFTWARE

Curso académico 2020 – 2021

Marcos Avilés Camarmas

Álvaro Berzosa Tordesillas

Jesús Granizo Egido

Director: Gonzalo Pajares Martinsanz

Resumen

En este trabajo se presenta una aplicación para la identificación de vehículos en vías urbanas mediante técnicas de detección del movimiento en imágenes y aprendizaje profundo para su identificación.

Mediante el entrenamiento de redes neuronales convolucionales como AlexNet y GoogLeNet se procede a la detección de vehículos en una serie de videos previamente cargados por el usuario.

La aplicación permite al usuario ajustar los parámetros de entrenamiento de las redes neuronales, realizar dicho entrenamiento y cargar un video para la posterior detección de los vehículos. Adicionalmente, se procede a la carga de dichos resultados a la red social Twitter para su difusión.

En el trabajo se realiza un análisis y valoración de resultados, tanto en la fase de entrenamiento como en la de decisión.

Palabras clave

- Aprendizaje profundo.
- Segmentación de regiones.
- Redes neuronales convolucionales.
- Flujo óptico.
- Internet de las cosas.
- Ciudades inteligentes.

Abstract

Identification of vehicles on urban roads using motion detection techniques in images and deep learning for their identification.

By training convolutional neural networks such as AlexNet or GoogLeNet, vehicles are detected in a series of videos previously uploaded by the user.

The present work will consist on the creation of an application in which the user is allowed to adjust the training parameters of the neural networks, carry out said training and upload a video for the subsequent detection of the vehicles. Additionally, these results are uploaded to the social network Twitter for dissemination.

In this work, an analysis and evaluation of results is carried out, both in the training phase and in the decision phase.

Keywords

- Deep Learning.
- Region Segmentation.
- Convolutional Neural Network.
- Optical Flow.
- Internet of the Things.
- Smart Cities.

Índice

1. Introducción

El ser humano siempre ha buscado la mejora en los procesos mediante la automatización de los mismos. Desde que el ser humano inventó la primera máquina que evitaba o disminuía el trabajo físico, hasta nuestros días, se ha llevado a cabo un largo proceso para mejorar la calidad de nuestras vidas.

Tras una época de ligeras mejoras, con la creación de herramientas simples, el uso del reloj es un momento clave, ya que permite el control del tiempo y con ello una buena medida para valorar la calidad de los procesos.

El siguiente hito lo tenemos en la segunda mitad del siglo XVIII. Se mecanizan los procesos, gracias a la Revolución Industrial y apoyado por las teorías de los economistas Adam Smith (ley del valor del trabajo) o Frederick Taylor (taylorismo).

A principios del siglo XX aparece el concepto de producción en cadena por parte de Henry Ford que especializa a máquinas y trabajadores en la realización de una sola función. Con ello se crean especialistas en cada una de las tareas.

Desde mediados del siglo XX debido a la Revolución Informática, se da un nuevo impulso a dichas mejoras. Se introducen computadores que permiten la medición de los procesos y su seguimiento, además de automatizar procesos que tienen que ver con actividades basadas en esfuerzos mentales y no solo físicos como lo hacía la antigua mecanización industrial.

Se alcanza un paso más con la introducción de la Inteligencia Artificial y el Aprendizaje Automático (*Machine Learning*), de forma que los computadores pueden manejar grandes volúmenes de datos, utilizar algoritmos mediante los cuales aprenda cuestiones sobre dichos datos y utilizarlos para optimizar la toma de decisiones. Este es el punto en el que arranca nuestro trabajo.

En los últimos años también se utiliza el análisis predictivo para poder anticiparnos a posibles escenarios adversos o poder maximizar escenarios ventajosos. La utilización de *Big Data* y *Deep Learning* como técnicas específicas de aprendizaje en estos casos mejora la posibilidad de realizar predicciones precisas, al contar con una gran cantidad de datos que mediante su estudio y clasificación permiten simular situaciones previsibles y su probabilidad concreta. Los sistemas de reconocimiento o los coches autónomos son ejemplos satisfactorios de estos sistemas de aprendizaje.

Finalmente cabe destacar la aparición del nuevo paradigma conocido como Internet de las Cosas (*IoT, Internet of Things*), término acuñado por Kevin Ashton en la primera década del siglo XXI. Busca la integración de internet en cualquier objeto de la vida cotidiana, de forma que la conexión entre distintos objetos y su intercambio de información genere conocimientos para lograr un beneficio.

En consecuencia, bajo los conceptos y paradigmas anteriores, en el presente trabajo se propone una aplicación inteligente, utilizando técnicas de visión por computador y aprendizaje profundo, para el reconocimiento de vehículos en movimiento en vías de

acceso a las ciudades, publicando los resultados del cómputo e identificación de los mismos aprovechando los recursos de IoT. Este desarrollo se sitúa en el ámbito de las ciudades inteligentes (*smart cities*) de futuro, ofreciendo una solución de concepto.

1.1 Soluciones similares

Con el fin de adelantarse a las necesidades de los consumidores o de las ciudades, algunas empresas ya han desarrollado una serie de proyectos para controlar y monitorizar el tráfico.

Mirame.net (2021)[1] es una empresa de soluciones tecnológicas de gestión inteligente, que ofrece a sus clientes diferentes productos como el control de aforos en tiendas o centros comerciales, el conteo de coches o la gestión de aparcamientos. Este último, denominado *Mall Parking Manager*[2], se basa en el procesamiento y análisis de video capturado a través de cámaras instaladas en el techo de las entradas del aparcamiento, realizando un conteo bidireccional de los vehículos.

La empresa Bosonit[3] situada en Logroño, trata también diversos temas relacionados con la aplicación de la Inteligencia Artificial para mejorar la vida en las ciudades. Posee diversos proyectos destinados a gestionar de una forma eficiente y responsable los distintos bienes de una ciudad. Gracias a la utilización de *Machine Learning* y *Big Data*, gestiona el tratamiento de agua, electricidad, energía solar, seguridad, sanidad y gestión administrativa de la misma. Además, trabaja en edificios inteligentes y en el concepto de movilidad en la ciudad, que es justamente el ámbito de aplicación que se presenta en este trabajo. Centrándonos en esto último, buscan la reducción de las emisiones y coste de combustible perdido debido a los atascos, así como la integración del vehículo eléctrico.

En línea con lo anterior, y en el ámbito de la gestión inteligente de la movilidad en la ciudad, es donde se enmarca el presente proyecto, centrándose, como se ha dicho previamente en plantear una solución de concepto para determinar el flujo de vehículos y su categoría con vistas a su integración en sistemas de control eficiente del flujo de vehículos. En esta línea el propio Ayuntamiento de Madrid, ha lanzado recientemente la aplicación Madrid Mobility 360 (2021)[4] con fines de gestión inteligente, donde sin duda alguna cabría perfectamente un planteamiento como el propuesto en este trabajo. Por otra parte, en el trabajo fin de grado presentado por Corral-Descargue y col. (2020)[5] se propuso un planteamiento similar y en la misma línea que el realizado en este trabajo.

1.2 Objetivos

Así pues, mediante este proyecto se busca el desarrollo de una aplicación a nivel conceptual para la automatización y control del flujo de vehículos en las ciudades. Este objetivo se engloba dentro de las llamadas Ciudades Inteligentes, que buscan la eficiencia y mejora de soluciones en todos sus ámbitos.

Nuestra aportación buscará, mediante dicho control del flujo de tráfico, optimizar el sistema de transportes urbano. De esta forma, se pueden reducir emisiones contaminantes, uno de los puntos clave actualmente y en el que más está incidiendo la Unión Europea en sus políticas. También se puede llevar un control exhaustivo de la circulación en las ciudades, para de esta forma controlar posibles atascos y determinar problemas en ciertas vías. Este proyecto puede desplegarse no solo para la gestión de ciudades, sino a un espectro más amplio, si se utiliza en carreteras nacionales o autopistas.

Además, utilizaremos una plataforma IoT, para poder procesar los datos obtenidos procedentes del tratamiento de los vídeos y transmitirlos a Twitter para que los usuarios puedan acceder a ellos a tiempo real.

Con tal propósito, se generan los siguientes objetivos específicos:

1. Utilización de las redes neuronales convolucionales AlexNet y GoogLeNet como dos modelos base dentro del Aprendizaje Profundo.
2. Digitalización de capturas de imágenes procedentes de los vídeos.
3. Detección y distinción de imágenes en movimiento.
4. Tratamiento de dichas imágenes para clasificar objetos y lograr un conteo de los distintos tipos de vehículos en circulación.
5. Elaboración de una interfaz amigable, sencilla y atractiva.
6. Transferencia de los datos desde la aplicación a la plataforma IoT.
7. Publicación de los distintos datos resultantes en Twitter para facilitar el acceso a los usuarios.
8. Almacenamiento de los datos obtenidos en base de datos relacional.

1.3 Plan de trabajo

Para la realización del proyecto se han establecido las siguientes tareas según el orden secuencial indicado:

1. Utilización de redes neuronales convolucionales para el aprendizaje

Para el proceso de aprendizaje a partir de los datos disponibles y tras el análisis de distintos modelos de redes, se ha decidido utilizar dos tipos de redes neuronales convolucionales. Estas son AlexNet y GoogLeNet que se encuentran previamente entrenadas.

2. Elaboración de la interfaz

Desarrollo de una interfaz detallada y amigable, desde la que se accede a los distintos menús disponibles. Desde ella será posible elegir la red neuronal a utilizar, añadir nuevas

imágenes al entrenamiento, reentrenar las redes, tratar los videos elegidos y enviar sus resultados a la plataforma IoT para su posterior publicación en Twitter.

3. Detección y etiquetado de imágenes en movimiento

Esta tarea se plantea para procesar imágenes donde aparecen objetos en movimiento, con el objetivo de su identificación por los modelos de red. Se trabaja a partir de un vídeo, que se procesa *frame* a *frame*, para detectar regiones en movimiento para delimitarlas, recortarlas y procesarlas por las redes para identificar esa región como objeto conocido en movimiento, esto es, los vehículos.

4. Conteo de vehículos

Se determina teniendo en cuenta la posición de un vehículo en un *frame* y el siguiente si se trata del mismo o es uno distinto, de forma que el cómputo se lleve a cabo de la forma más precisa posible a la vez que se determina el tipo de vehículos circulando por una vía determinada.

5. Transferencia de datos a ThingSpeak[6]

Una vez obtenidos los datos relativos al tipo de vehículos y el número que ha circulado, son enviados al canal correspondiente en ThingSpeak[6] quedando la información publicada y disponible para consulta y análisis en su caso. Y posterior creación de la plataforma propia al no ajustarse Thingspeak a las necesidades del proyecto.

6. Entrenamiento de las redes neuronales

Se trata de una tarea relevante en tanto en cuanto permite configurar y establecer los distintos parámetros de aprendizaje de los modelos de red, con el fin de que la detección de los vehículos se lleve a cabo con la mayor precisión posible.

7. Inserción de nuevas imágenes para entrenamiento

Es una tarea que permite añadir nuevas imágenes al directorio en el que se encuentran aquellas utilizadas para el entrenamiento, cuyo objetivo consiste en realizar operaciones de transformación para añadir ruido, realizar pequeñas rotaciones, incluir objetos nuevos, entre otras, para incluir más ejemplos posibles de cara a mejorar el rendimiento de las redes en futuros entrenamientos.

8. Visualización de las imágenes disponibles para el entrenamiento

Acceso al directorio de imágenes para entrenamiento.

9. Persistencia en base de datos relacional y realización de consultas

Se almacenarán los datos en una base de datos relacional, y existirá un menú desde el cual el usuario podrá visualizar dichos datos según los parámetros especificados para su consulta.

10. Grabación de vídeos de tráfico

Consistente en realizar grabaciones reales de vías de tráfico desde distintos puntos de Madrid y Guadalajara, asignando a cada uno de ellos un canal específico de ThingSpeak[6] para el tratamiento de sus datos asociados. En todos los casos quedan suficientemente preservados todos los aspectos relacionados con la privacidad de los datos capturados atendiendo a la normativa vigente.

1.4 Contribuciones personales

1.4.1 Marcos Avilés

Labor de investigación:

- Elección sobre el modelo de gestión de proyecto a seguir, comprobando la idoneidad de cada uno de ellos para nuestro proyecto y seleccionando el tipo de metodología a aplicar. Una vez elegida la metodología, estudio de los distintos artefactos, selección y adaptación de los mismos al proyecto.
- Adquisición de conocimientos sobre programación en Matlab[7], al no haber tratado con ello durante el Grado.
- Búsqueda de aplicación para realizar el seguimiento del Sprint Backlog, tratando de localizar información sobre diversas aplicaciones: Trello, Jira y Pivotal Tracker.
- Búsqueda de herramientas para la elaboración de Mockups sobre los que se basarán las interfaces. En concreto buscando información y probando las herramientas: Balsamiq, Frame Box, LucidChart y Microsoft Visio.
- Búsqueda de información y comprensión sobre los modelos de redes neuronales a integrar en la aplicación: AlexNet y GoogLeNet.
- Trabajo de investigación sobre ThingSpeak[6] y todos sus componentes, concretamente aquellos que son necesarios para poder enviar la información a Twitter, que son React, ThingTweet y Matlab Analysis.

Labor de desarrollo:

- Trabajo junto al resto de compañeros en la elaboración de las primeras funcionalidades, la Interfaz principal, el Controller y el Dispatcher.
- Creación del transfer para los parámetros del entrenamiento, que servirán para seleccionar el nivel de completitud del mismo, definiendo así opciones basadas en un menor coste de tiempo frente a otras que primarán el porcentaje de acierto.

- Realización de la lógica necesaria para la redimensión de imágenes, recogiendo las imágenes seleccionadas por el usuario y tratándolas según la red para la que vayan a ser utilizadas. Además, se asegura la unicidad de las mismas al asignarlas un nombre único en base a su tipo y a la cantidad de imágenes del mismo tipo existentes.
- Trabajo en la lógica propia de la visualización de imágenes para el entrenamiento, trabajando a su vez en la configuración de sus eventos y el comando asociado a dicha funcionalidad.
- Colaboración en la lógica de negocio propia de la reproducción de videos y su funcionamiento.
- Realización de pruebas y control de errores en cada una de las distintas funcionalidades.
- Trabajo en integración con ThingSpeak[6], implementando las lecturas y escrituras desde la aplicación a los diferentes canales.
- Elaboración de los canales necesarios en ThingSpeak[6] para el control de datos.
- Realización de los scripts necesarios de Matlab Analysis para obtener las variables a mostrar en Twitter.
- Creación de base de datos relacional con PHPMyAdmin, trabajando tanto en la creación de las tablas como en la de sus relaciones. También en la introducción de los datos propios de la tabla Cámaras.
- Creación de las consultas necesarias en SQL para la obtención de datos, realizando distintas búsquedas en base a los tipos de vehículos seleccionados, la cantidad de los mismos, el rango de fechas y las diferentes cámaras que el usuario seleccione.
- Trabajo junto a Jesús en la integración del API PHP en Matlab[7] para la persistencia de datos.
- Colaboración en distintas interfaces con el resto de integrantes del equipo.

Labor de gestión:

- Control de artefactos propios de Scrum Master, como el Sprint Backlog y el Product Backlog, debido a que ocupé dicho puesto en la asignatura Gestión de Proyectos Software.
- Realización de diversos Mockups para la realización de interfaces, mostrados en el Anexo 2[A2].

Labor de documentación:

- Realización de la documentación del apartado 1.Introducción, buscando información histórica que marca los antecedentes de nuestro proyecto.
- Apartado 3.2 Gestión de proyecto, y colaboración en el apartado 3.3 donde se explican las herramientas utilizadas en el proyecto.
- Redacción del apartado 4.1 Entrenamiento de redes, en la parte de GoogLeNet, realizando del punto 4.3 para el resultado de la clasificación junto a Álvaro realizando tablas en las que se comparan los vehículos detectados con los reales, así como en el apartado 4.5 Errores frecuentes. Realización del apartado 4.6 que explica la integración de los datos.
- Colaboración en la bibliografía.
- Realización del apartado 5.1 Conclusión, al igual que trabajo en el 5.2 Trabajo futuro, en concreto el apartado de nuevas funcionalidades.
- Trabajo en el anexo 1 consistente en el Product Backlog.
- Trabajo junto al resto de compañeros recopilando imágenes y capturando vídeos para probar la aplicación realizando grabaciones de tráfico real en diversos puntos de Madrid.
- Conteo manual de los vehículos registrados en los vídeos correspondientes a Moncloa y Villaverde, para poder comparar dichos resultados a los que determine nuestro entrenamiento en cada una de las redes.
- Realización de los entrenamientos correspondientes a GoogLeNet en los tres tipos propuestos para observar posteriormente el nivel de acierto en cada uno de ellos.
- Estudio sobre el resultado de diferentes ejecuciones para observar el comportamiento de la aplicación.

1.4.2 Álvaro Berzosa

Labor de investigación:

- Búsqueda de información y comprensión de las redes neuronales AlexNet y GoogLeNet con las que se va a trabajar.
- Búsqueda de distintas redes neuronales para comparar el funcionamiento y diferencias respecto a las que vamos a trabajar.
- Investigación del funcionamiento de la herramienta de Matlab[7] llamada “App Designer” para el desarrollo de las interfaces.
- Búsqueda de proyectos y soluciones similares en la literatura.

- Captura de videos de muestra para su posterior uso en la detección de vehículos.
- Creación de la cuenta de Twitter @ControlMadrid para la posterior automatización de la publicación de tweets con los resultados de las detecciones.

Labor de gestión:

- Como miembro del equipo he colaborado en la realización de la lista de tareas propuestas (Sprint Backlog).
- Realización de diversos mockups, mostrados en el Anexo 2[A2], para tener un diseño conceptual en el que basarse a la hora de desarrollar las interfaces.
- Realización de reuniones con los miembros del equipo asi como con el tutor para resolver dudas y valorar el avance del proyecto.

Labor de desarrollo:

- Trabajo junto al resto del equipo en la implementación del Dispatcher, Controller y las interfaces iniciales de la aplicación.
- Colaboración con los resto de miembros del equipo en los eventos de subida de imágenes de AlexNet y GoogLeNet.
- Implementación de las interfaces propias de AlexNet y GoogLeNet siguiendo el diseño de los mockups previamente realizados.
- Creación de la interfaz de selección de parámetros del entrenamiento de las redes.
- Creación de la interfaz para las consultas especificando los parámetros seleccionables y mostrando los datos resultantes en una tabla.
- Trabajo con el resto del equipo en la integración de las redes neuronales convolucionales en el proyecto.
- Ayuda en el manejo de los datos resultantes con Thingspeak[6] y en la posterior iteración de este apartado.
- Generación de diagramas de clases de negocio y presentación, incluidos en el Anexo 3[A3].
- Labores de depuración de código para encontrar y solucionar errores.

Labor de documentación:

- Redacción del resumen y consensuando con el resto del equipo las palabras clave.
- Apartado 1.5 correspondiente a la estructrura de la memoria.
- Colaboración con Marcos en el apartado 4.5 de errores frecuentes.
- Colaboro con Jesús en la realización del apartado 3.1 sobre la arquitectura de la aplicación.
- Realización del apartado 3.3 de la memoria referente a las herramientas utilizadas junto con el resto de miembros del equipo.
- Conteo manual de los vehículos correspondientes al video ubicado en la zona de Ventas para la generación de las tablas de resultados del apartado 4.1 y su posterior comparación con los resultados obtenidos.
- Redacción del apartado 4.2.
- Realización de los entrenamientos del apartado 4.3 correspondientes a la red AlexNet así como la ejecución de las detecciones de vehículos posteriores para completar las tablas de resultados.
- Desarrollo en el punto 5.2, correspondiente al trabajo futuro, del apartado de mejoras de la aplicación.
- Traducción del apartado 5 para la posterior realización del apartado 6.
- Inclusión de los Anexos 2 y 3 sobre los diagramas de clases de negocio y presentación, y los mockups.
- Colaboración con los demás miembros del equipo en la generación de la bibliografía.
- Revisión de la memoria.

1.4.3 Jesús Granizo

En la parte de investigación del proyecto:

- Me he encargado de la búsqueda de información y diferentes tutoriales para poder realizar un correcto uso y entender su funcionamiento de los algoritmos[23] AlexNet[21] y GoogLeNet[22,24] utilizados en el proyecto realizado.
- Investigación de distintos alojamientos web con MySQL y PHP y creación de servidores para alojar una API[25] que permitiera realizar las operaciones de guardado y carga de datos de forma correcta.

- Búsqueda de un programa que permitiera desarrollar y ejecutar código en el lenguaje PHP[26] para el diseño de la API[25].
- Investigación y búsqueda de programas que permitieran hacer peticiones a una API[27] local u online y comprobar el correcto funcionamiento del código PHP desarrollado
- Búsqueda de los patrones de diseño software[28,29] que cumplieran los requisitos necesarios para desarrollar nuestro proyecto, y nos ayudaran a simplificar el código e implementando abstracción y encapsulamiento en el código diseñado.

Labor de desarrollo del proyecto:

- Me he encargado de realizar la capa de negocio, principalmente programada en Matlab[7]. He realizado la lectura, comprensión y explicación a mis compañeros de proyecto de los algoritmos tanto de entrenamiento de las redes utilizadas como el de detección de vehículos mediante videos.
- Me he encargado de crear las clases necesarias para realizar una ejecución correcta de los algoritmos, añadiendo los parámetros necesarios para que el usuario pueda elegir como realizar la ejecución con total libertad. Para realizar esta ejecución de forma apropiada me he encargado de la búsqueda e implementación de los patrones de diseño software necesarios para nuestro proyecto.
- Creación de una correcta comunicación entre las interfaces y los datos de la integración, a través del controlador de la aplicación y eventos creados para facilitar el envío de datos desde la capa de presentación, además he realizado la implementación de un controlador de comandos para ejecutar aquel código que no implica el uso de interfaces.
- Creación y desarrollo del algoritmo de conteo y diferenciación de la aplicación que permite identificar si dos vehículos que han sido detectados en distintos *frames* del video son en realidad el mismo. Además de llevar a cabo la idea de cómo realizar dicha detección.
- También me he encargado de la creación de Transfers y conexiones con la capa de dominio, de forma que permita a esta manejar los datos enviados de forma sencilla, en un principio con el envío a Thingspeak[6] y más tarde implementando la conexión vía API REST con el servidor donde se guarda actualmente los datos del entrenamiento realizado.
- Además, he contribuido con la realización de ayudas a la capa de presentación, utilizando programación dinámica obteniendo y tratando datos y elementos que debía renderizar la interfaz, facilitando así su carga.

- He contribuido, además, en la programación y desarrollo de la capa de integración, principalmente en el lenguaje PHP, ya que contaba con experiencia previa y he podido ayudar a mis compañeros, este ha sido el utilizado para el diseño de la API para guardar u obtener datos de las detecciones.

Labor de documentación:

- Realizacion de la documentacion del apartado 1.2: Objetivos, en este punto expliqué los distintos objetivos que queriamos conseguir con el desarrollo de nuestra aplicación y que estrategia seguir para conseguir dicho objetivo.
- Realización del apartado 1.4: Plan de trabajo, en este punto se explicó el plan de trabajo que deberiamos seguir para realizar un correcto desarrollo y llevar un control adecuado de las operaciones y tareas que hemos llevado cada uno de los integrantes del equipo.
- Realización del apartado 3.1: Arquitectura, con la colaboración de los integrantes del grupo explicando la arquitectura del programa, tanto del lado cliente debido a los conocimientos adquiridos en la programación con Matlab[7], y del lado servidor debido a la experiencia que cuento con PHP y SQL. También se explica todos los programas, lenguajes y algoritmos utilizados durante el desarrollo del proyecto.
- Realización del apartado 4.4: Algoritmo de conteo y diferenciación de vehículos, explicando el funcionamiento del algortimo que creamos e indicando las ventajas respecto a otros algoritmos de diferenciación. Tambien la creación de los esquemas e imágenes para este punto.
- Colaboración el los apartados 5.1 y 5.2: Conclusiones y trabajo futuro, donde se ha explicado las distintas ideas que hemos adquirido durante el desarrollo y las principales características a añadir o mejorar para realizar en un futuro.
- Colaboración para la busqueda y realización de los distintos puntos de la bibliografía, asi como añadir los distintos enlaces a ella desde la memoria.
- Realización del Anexo 4: Instalación y ejecución de la aplicación, creando un pequeño manual de usuario para descargarse, instalar y ejecutar el programa creado de una manera correcta.
- Realización del Anexo 5: Licencias de imágenes de la memoria e iconos de la aplicación, realizando la búsqueda de las licencias utilizadas durante el desarrollo del programa y de la creación de la memoria, investigando además como realizar las atribuciones necesarias a los autores de las distintas imágenes cumpliendo con las respectivos derechos de autor.

- Trabajo junto al resto de compañeros recopilando imágenes y capturando vídeos para probar la aplicación realizando grabaciones de tráfico real en diversos puntos de Madrid.
- Grabación de las imágenes utilizadas en las cámaras de tráfico y deteccion de imagenes en Guadalajara debido a la cercanía geográfica de esta zona respecto al resto de compañeros.

1.5 Estructura de la memoria

La memoria se estructura en distintas secciones como sigue:

1. **Introducción:** se describen los antecedentes y cómo está el panorama actual, dando algún ejemplo de soluciones similares realizadas. También se muestran los objetivos del proyecto, el plan de trabajo necesario para completar el proyecto y las contribuciones personales de los miembros del equipo.

2. **Métodos técnicos – teóricos analizados:** en esta sección se describen a nivel teórico las técnicas utilizadas para la detección y procesado de los vehículos.

3. **Diseño y solución desarrollada:** descripción detallada de las herramientas utilizadas, la arquitectura y las conexiones para la transferencia de datos.

4. **Resultados obtenidos:** se exponen los resultados de los modelos. Se explica cómo se detecta el movimiento, los resultados del entrenamiento de las redes, tiempos de entrenamiento, precisión. Los resultados de la clasificación y la subida de datos a Thingspeak[6].
5. **Conclusiones y trabajo futuro:** se expone de un breve resumen de lo que se ha realizado, y lo que quedaría por hacer de cara al futuro.

6. **Bibliografía:** una lista con información de aquellos recursos que se han utilizado para la realización de la memoria.

7. **Anexos:** se incluyen cinco anexos. En el primero de ellos aparece el Product Backlog donde se tienen las distintas funcionalidades de la aplicación, el segundo contiene una muestra de Mockups realizados para trabajar con las interfaces, a continuación, en el tercero se encuentran los diagramas de clases para las capas de negocio y presentación, tras ello, en el cuarto aparece una guía para la instalación y ejecución de la aplicación. Por último, el quinto anexo recoge las licencias de las imágenes utilizadas en la memoria, así como los iconos utilizados en la aplicación.

1.6 Introduction

Human beings have always sought to improve processes by automating them. From the time man invented the first machine that avoided or reduced physical labor, until today, a long process has been carried out to improve the quality of our lives.

After a period of slight improvements, with the creation of simple tools, the use of the clock is a key moment, since it allows the control of time and thus a good measure to assess the quality of processes.

The next milestone was in the second half of the 18th century. Processes are mechanized, thanks to the Industrial Revolution and supported by the theories of the economists Adam Smith (law of the value of labor) or Frederick Taylor (Taylorism).

At the beginning of the 20th century, Henry Ford came up with the concept of chain production, specializing machines and workers in the performance of a single function. This created specialists in each of the tasks.

Since the middle of the 20th century, due to the Computer Revolution, a new impulse is given to these improvements. Computers that allow the measurement of processes and their follow-up are introduced, in addition to automating processes that have to do with activities based on mental efforts and not only physical as the old industrial mechanization did.

A further step is reached with the introduction of Artificial Intelligence and Machine Learning, so that computers can handle large volumes of data, use algorithms to learn questions about these data and use them to optimize decision making. This is the point at which our work starts.

In recent years, predictive analytics has also been used to anticipate possible adverse scenarios or to maximize advantageous scenarios. The use of Big Data and Deep Learning as specific learning techniques in these cases improves the possibility of making accurate predictions, by having a large amount of data that through their study and classification allow us to simulate predictable situations and their specific probability. Recognition systems or autonomous cars are successful examples of these learning systems.

Finally, it is worth mentioning the emergence of the new paradigm known as the Internet of Things (IoT), a term coined by Kevin Ashton in the first decade of the 21st century. It seeks the integration of the Internet in any object of everyday life, so that the connection between different objects and their exchange of information generates knowledge to achieve a benefit.

Consequently, under the above concepts and paradigms, this paper proposes an intelligent application, using computer vision and deep learning techniques, for the recognition of moving vehicles on access roads to cities, publishing the results of the computation and identification of them taking advantage of IoT resources. This development is in the field of smart cities of the future, offering a concept solution.

1.6.1 Objectives

This project seeks to develop a conceptual application for the automation and control of the flow of vehicles in cities. This objective is included within the so-called Smart Cities, which seek efficiency and improved solutions in all areas.

Our contribution will seek, through traffic flow control, to optimize the urban transport system. In this way, polluting emissions can be reduced, one of the key points at present and on which the European Union is currently focusing its policies. It is also possible to carry out an exhaustive control of the traffic in the cities, in order to control possible traffic jams and determine problems on certain roads. This project can be deployed not only for city management, but on a broader spectrum, if used on national roads or highways.

In addition, we will use an IoT platform that will be able to process the data obtained from the processing of the videos and transmit it to Twitter so that users can access them in real time.

For this purpose, the following specific objectives are generated:

1. Use of the convolutional neural networks AlexNet and GoogLeNet as two base models within Deep Learning.
2. Digitization of image captures from the videos.
3. Detection and distinction of moving images.
4. Processing of these images to classify objects and achieve a count of the different types of vehicles in circulation.
5. Elaboration of a friendly, simple and attractive interface.
6. Data transfer from the application to the IoT platform.
7. Publication of the different resulting data on Twitter to facilitate access to users.
8. Storage of the obtained data in a relational database.

1.6.2 Work plan

The following tasks have been established for the realization of the project according to the sequential order indicated:

1. Use of Convolutional Neural Networks for learning

For the learning process from the available data and after the analysis of different network models, it has been decided to use two types of convolutional neural networks. These are AlexNet and GoogLeNet, which have been previously trained.

2. Development of the interface

Development of a detailed and user-friendly interface, from which the different available menus can be accessed. From it will be possible to choose the neural network to be used, add new images to the training, retrain the networks, process the chosen videos and send the results to our IoT platform for subsequent publication on Twitter.

3. Detection and labeling of moving images

This task is proposed to process images where moving objects appear, with the objective of their identification by the network models. We work from a video, which is processed frame by frame, to detect moving regions in order to delimit, crop and process them by the networks to identify that region as a known moving object, i.e., vehicles.

4. Vehicle count

It is determined by taking into account the position of a vehicle in a frame and the next one if it is the same or a different one, so that the computation is carried out as accurately as possible while determining the type of vehicles circulating on a given road.

5. Data transfer to ThingSpeak[6]

Once the data related to the type of vehicles and the number of vehicles that have circulated are obtained, they are sent to the corresponding channel in ThingSpeak[6], leaving the information published and available for consultation and analysis if necessary. And later creation of its own platform when Thingspeak[6] did not meet the needs of the project.

6. Training of the neural networks

This is a relevant task insofar as it allows configuring and establishing the different learning parameters of the network models, so that the detection of vehicles is carried out as accurately as possible.

7. Insertion of new images for training

It is a task that allows adding new images to the directory where the images used for training are located, whose objective is to perform transformation operations to add noise, perform small rotations, include new objects, among others, in order to include more possible examples to improve the performance of the networks in future training.

8. Visualization of the images available for training

Access to the directory of images for training.

9. Persistence in relational database and queries.

The data will be stored in a relational database, and there will be a menu from which the user will be able to visualize the data according to the parameters specified for consultation.

10. Recording of traffic videos

Consisting in making real recordings of traffic routes from different points of Madrid and Guadalajara, assigning to each of them a specific channel of ThingSpeak[6] for the treatment of its associated data. In all cases, all aspects related to the privacy of the captured data are sufficiently preserved in compliance with current regulations.

2. Métodos técnico - teóricos utilizados

2.1 Cálculo del flujo óptico con imágenes

A continuación, se explican, en primer lugar, las técnicas aplicadas para detección de objetos en movimiento en imágenes, cuyo fundamento es el flujo óptico, que permite determinar regiones candidatas como entradas a la red. En segundo lugar, se explican los modelos de redes utilizados (AlexNet y GoogLeNet), junto con las operaciones involucradas en dichos modelos.

El flujo óptico permite identificar el movimiento de los objetos que tiene lugar entre dos secuencias (*frames*) de vídeo consecutivos separados por un intervalo de tiempo reducido (*dt*) (Pajares y Cruz, 2007[15]; Farneback, 2003[10]). La obtención de éste permite determinar la posición, dirección y velocidad de un objeto dentro de un *frame*. Este cálculo se realiza sobre todos los píxeles de un *frame* de vídeo.

El cómputo del gradiente es uno de los métodos clásicos utilizados en el cómputo del gradiente, que representa los cambios del nivel de intensidad en una imagen.

Su fundamento se basa en la obtención de la intensidad de un píxel en la posición (*x, y*) en un instante de tiempo (*t*) a través de la función f(*x,y,t*). Si además la imagen es dinámica en el tiempo, se puede pensar en encontrar el mismo píxel, pero en otro instante de tiempo muy muy cercano (*dt*) en el que apenas ha habido desplazamiento (*dx, dy*). Esto se sintetiza en la ecuación (2.1).

$$f(x + dx, y + dy, t + dt) = f(x, y, t) \tag{2.1}$$

Si se desarrolla el lado izquierdo de la ecuación (2.1) como un polinomio de Taylor de primer orden, se obtiene el resultado mostrado en la ecuación (2.2).

$$\begin{aligned} f(x + dx, y + dy, t + dt) &= f(x, y, t) + \frac{\delta f}{\delta x} dx + \frac{\delta f}{\delta y} dy + \frac{\delta f}{\delta t} dt + O(\delta^2) = \\ &= f(x, y, t) + f_x dx + f_y dy + f_t dt + O(\delta^2) \end{aligned} \tag{2.2}$$

En este desarrollo las derivadas se interpretan como las diferencias existentes entre las intensidades de los píxeles adyacentes.

Ahora podemos despreciar el término de orden superior, obteniendo una igualdad aproximada, luego de hacer esto y despejar la ecuación, llegamos a (2.3).

$$-f_t = f_x \frac{dx}{dt} + f_y \frac{d_y}{dt} \tag{2.3}$$

De esta forma se obtiene una ecuación que indica efectivamente que la diferencia de intensidad *f*, en una misma posición e instante de tiempo se debe a la diferencia de intensidad espacial, que es debida a la cantidad de movimiento.

Asumiendo que la intensidad de la imagen permanezca constante a lo largo del tiempo (δf / δt = 0) y que el vector gradiente de la ecuación (2.3) sea constante, se llega a la conclusión reflejada en la ecuación (2.4).

$$\begin{bmatrix} {\delta^2 f}/{\delta x^2} & {\delta^2 f}/{\delta x \delta y} \\ {\delta^2 f}/{\delta x \delta y} & {\delta^2 f}/{\delta y^2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\frac{\delta(\nabla f)}{\delta t} \tag{2.4}$$

Si además se considera un entorno de vecindad Ω para todos los puntos, entonces la ecuación se transforma en la (2.5).

$$\begin{bmatrix} {\delta^2 f}/{\delta x^2} & {\delta^2 f}/{\delta x \delta y} \\ {\delta^2 f}/{\delta x \delta y} & {\delta^2 f}/{\delta y^2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\frac{\delta(\nabla f)}{\delta t} \tag{2.5}$$

Aplicando mínimos cuadrados a la ecuación (2.5) se llega a la solución, que aparece en la ecuación (2.6).

$$v = (A^T A)^{-1} (A^T b) \tag{2.6}$$

De forma que se puede obtener el flujo óptico V(*u, v*) para todo píxel de una imagen, cuyas componentes horizontal y vertical son u y v, respectivamente.

2.2 Detección de regiones en movimiento

A partir del cálculo del flujo óptico en los puntos de la imagen, se pueden identificar los objetos en movimiento en ella, ya que estos vectores contienen y representan implícitamente la magnitud y sentido del movimiento producido en cada píxel.

El módulo del flujo óptico se utiliza para calcular el valor medio del movimiento y la desviación estándar de éste. La suma de la media y la desviación se utiliza para

determinar un valor umbral, que sirve para realizar un proceso de binarización sobre una imagen, proporcionando como resultado una imagen en blanco y negro (Pajares y Cruz, 2007[15]).

Sobre esta imagen binaria se realiza un proceso de etiquetado de las distintas componentes conexas, teniendo cada una de éstas el mismo identificador para cada uno de los píxeles que las componen.

Para realizar este proceso de etiquetado se aplica el algoritmo de Haralick y Shapiro (1992)[11], un algoritmo de dos fases que emplea una tabla de equivalencias. En la primera fase se realiza un recorrido sobre la imagen binarizada, de todas las filas de arriba hacia abajo y recorriendo los píxeles de izquierda a derecha. Cuando un píxel no posee etiqueta, se evalúan los ocho píxeles adyacentes para determinar si alguno de ellos se la puede propagar a éste.

Al aplicar aplicación de este algoritmo, se distinguen varios casos:

a) Si ningún píxel adyacente tiene etiqueta, entonces se asigna una nueva a éste y se inserta en la tabla de equivalencias un elemento tipo par, cuya clave y valor es el valor de la etiqueta asignada.
b) Si solamente uno de los píxeles adyacentes posee etiqueta, éste la propaga al píxel actual.
c) Si más de un píxel adyacente puede propagar el valor de su etiqueta al píxel actual, entonces éste toma el valor de la etiqueta menor y se realiza un cambio en la tabla de equivalencias.

En la segunda fase se vuelve a realizar el recorrido de la misma forma que la fase anterior, pero con el objetivo de cambiar el valor de cada píxel por el valor de su etiqueta equivalente, mediante la consulta de la tabla de equivalencias.

Una vez finaliza el algoritmo se obtienen una serie de propiedades de la imagen, como la obtención de la ubicación y el tamaño de las áreas conexas, etiquetadas y delimitadas por un rectángulo (*Bounding Box*). Su obtención se realiza a través de la comparación de aquellas zonas que tengan una magnitud de movimiento significativo determinado por el valor umbral, previamente obtenido, entre dos *frames* de vídeo consecutivos.

Una vez obtenidas todas las regiones candidatas que son representativas del movimiento, éstas se ubican y localizan sobre la imagen original, realizando un recorte sobre ésta usando las coordenadas del *Bounding Box*. Este recorte es el que se proporciona a la Red Neuronal Convolucional como entrada, para identificar los distintos tipos de vehículos.

Otra de las propiedades de interés que también se calcula son los centroides de las áreas en las que se ha detectado un movimiento superior al valor umbral. Esto ha permitido implementar un sistema de conteo de vehículos, ya que podemos ubicar dichos centroides dentro de una franja delimitada en la imagen que se procesa.

2.3 Redes Neuronales Convolucionales

Una Red Neuronal Convolucional (CNN, *Convolutional Neural Network*) es un tipo específico de red neuronal, cuyo fundamento base son las conocidas capas de convolución, que dan pie a su nombre (Pajares y col, 2021[16]). Este tipo de red trabaja en dos fases. En la primera de ellas que sería de aprendizaje, se va entrenando dicha red mediante la utilización de un algoritmo que irá ajustando los valores de los núcleos de convolución, de forma que la clasificación de las imágenes llegue a un punto óptimo para poder identificar las regiones en movimiento automáticamente. Una vez que se entrena la red y se dispone de unos modelos ajustados a nuestras necesidades, se podrán utilizar en la segunda fase del modelo, que es la de clasificación.

Como ya se ha explicado previamente, en este trabajo se han utilizado dos modelos de CNN, concretamente el modelo AlexNet y GoogLeNet, que se describen a continuación. En cualquiera de estos modelos se realizan diferentes operaciones, con definición de una serie de parámetros, que constituyen la clave del proceso y que se describen a continuación.

2.3.1 Operaciones aplicadas en las CNN

A) Convolución

La convolución es una operación que se define como sigue para el procesamiento de imágenes, que como se sabe son estructuras bidimensionales 2-D:

$$C(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.7}$$

El resultado de la convolución es *C(i, j)* para un determinado pixel (*i,j*) cuando la entrada es una imagen *I* o bien un elemento de un tensor cuando se trata de capas más internas. En la ecuación anterior *K* hace referencia a lo que se denomina núcleo de convolución. La aplicación de esta expresión al contexto de las imágenes resulta en el esquema gráfico mostrado en la figura 2.1. El núcleo de convolución *K* con sus correspondientes valores, se posiciona sobre la cuadrícula superior izquierda para obtener el resultado *P* que se posiciona que se posiciona igualmente en la cuadrícula superior izquierda. A continuación, se desplaza el núcleo una posición hacia la derecha para obtener de forma similar el resultado *Q* y así sucesivamente se desplaza el núcleo de izquierda a derecha y de arriba hacia abajo hasta completar los valores de las celdas en la matriz resultante.

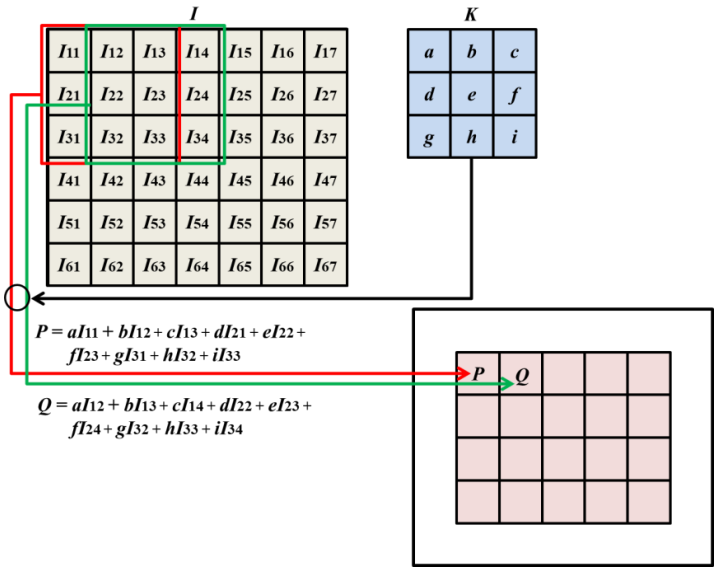


Figura 2.1 - Ejemplo de convolución 2-D

No obstante, si este núcleo en lugar de desplazarse una única celda se desplaza más de una, es necesario establecer lo que se conoce como *stride* (*s*) que puede tomar valores mayores que la unidad. Además, como el núcleo puede desbordar la imagen o tensor de entrada cuando la celda correspondiente al valor *y* del núcleo *K* se sitúa en las filas o columnas más exteriores, existen valores del núcleo que no se utilizan, en cuyo caso estas filas o columnas quedan sin procesar. Por esta razón, si se desea que el resultado tenga las mismas dimensiones de la imagen, es necesario añadir filas y columnas rellenas de ceros, esta operación se conoce como *padding* (*p*). En función de los valores *p* y *s* se obtiene la correspondiente dimensión de salida (*o)* según las siguientes relaciones. En estas expresiones *k* es la dimensión del núcleo de convolución, por ejemplo, en el caso anterior, *k*=3.

$$\textit{Relación 5}:\ o=\left[\frac{i-k}{s}\right]+1 \qquad \textit{Relación 6}:\ o=\left[\frac{i+2p-k}{s}\right]+1 \tag{2.8}$$

B) ReLU

Esta función conocida como *Unidad Lineal Rectificada* (ReLU, *Rectified Linear Unit*) se define como *f*(*x*) = *max*(0, *x*), cuya representación es la que se muestra en la figura 2.2, cuyas características y su utilidad se centran en lo siguiente:

a) Evitar la saturación del gradiente durante el proceso de optimización por el método del gradiente descendente.

b) Disminuir la complejidad computacional por el hecho de reducir valores negativos a cero.

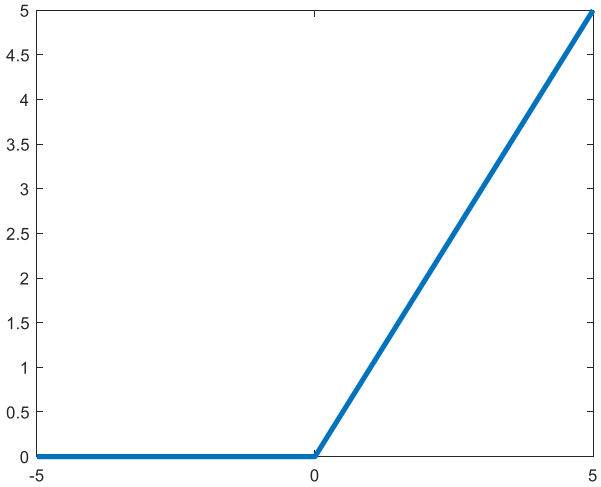


Figura 2.2- Representación de la función ReLU

C) Normalización

La operación de normalización se aplica con el fin de acelerar la convergencia durante el proceso de aprendizaje. En la expresión siguiente se define una función de normalización por lotes,

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a^i_{x,y} \right)^2 \right)^{\beta} \qquad (2.9)$$

donde *N* es el número de filtros de la capa dada, $a^i_{x,y}$ es la actividad de la neurona y *k*, α, *n*, β son hiperparametros. En Krizhevsky (2012)[14] se proponen como más apropiados los siguientes valores para los parámetros *k* = 2, *n* = 5, $\alpha = 10^{-4}$, β = 0.75.

D) Pooling

Se trata de una función cuyo objeto es agrupar valores en las capas de características, de tal forma que dada una pequeña traslación en la entrada no afecte a los valores de las salidas. En la figura 2.3 se muestra un ejemplo gráfico de agrupamiento por máximos, es decir seleccionando el valor máximo en cada una de las ventanas de agrupamiento, en este caso de dimensión 2x2.

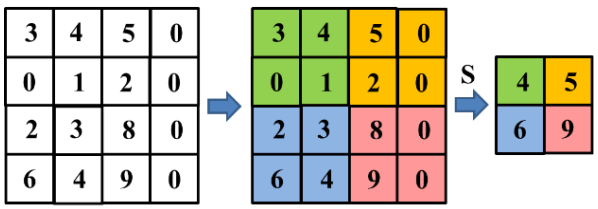


Figura 2.3 - Max pooling

E) Dropout

Es un mecanismo para evitar sobresaturaciones en la red neuronal. Consiste en anular determinado tipo de neuronas de forma aleatoria mediante un hiperparámetro que indique la probabilidad de supervivencia de cada neurona.

F) Softmax

Consistente en proyectar los valores de la salida en la capa final al rango [0,1], proporcionando así una especie de probabilidad de pertenencia a cada una de las clases para una imagen de entrada dada. Su función se define según la siguiente expresión.

$$softmax(\boldsymbol{x})_i = \frac{exp\,(x_i)}{\sum_{j=1}^{n} exp\,(x_j)} \tag{2.10}$$

G) Gradiente descendente

Es una técnica de minimización utilizada para el ajuste de los pesos involucrados en los modelos de las redes durante el proceso de retro-propagación. La función por optimizar se conoce como función *objetivo* y cuando se está minimizando, se le denomina también *loss function* o *error function*. Se trata de minimizar una función objetivo que tiene la forma,

$$J(w)=\frac{1}{n}\sum_{i=1}^{n} J_i(w) \tag{2.11}$$

donde el parámetro *w* que minimiza *J*(*w*) debe estimarse, constituyendo el objetivo principal del aprendizaje. J_i se asocia con la *i*-ésima observación en el conjunto de datos utilizados para el entrenamiento (ajuste). El gradiente descendente se usa para minimizar esta función de forma iterativa, con iteraciones *t*,

$$w(t+1)=w(t)-\varepsilon\frac{1}{n}\sum_{i=1}^{n}\nabla J_i(w) \tag{2.12}$$

De esta forma iterativa el método recorre el conjunto de entrenamiento y realiza la actualización anterior para cada muestra de entrenamiento. Se pueden realizar varios pasos sobre el conjunto de entrenamiento hasta que el algoritmo converja. Estas muestras así seleccionadas constituyen lo que se denomina *batch* como se describe más adelante a la hora de seleccionar los parámetros de entrenamiento. En este sentido, es habitual utilizar exactamente la técnica conocida como Gradiente Descendente Estocástico (SGD, *Stochastic Gradient Descent*) (Bishop, 2006[8]; Ruder, 2017[17]),

Las implementaciones típicas pueden usar una razón de aprendizaje adaptativa para que el algoritmo converja. En pseudocódigo, el método de gradiente descendente estocástico es como sigue,

1. Elegir un vector inicial de parámetros *w* (puede ser aleatoriamente) y razón de aprendizaje ε.
2. Repetir hasta que se consigue un mínimo aproximado

2.1 Seleccionar aleatoriamente ejemplos en el conjunto de entrenamiento
2.2 Para *i* = 1, 2, ...,*n*, hacer $w(t+1)=w(t)-\varepsilon\nabla J_i(w)$

2.4 AlexNet

La red AlexNet (ImageNet, 2020[12]; Russakovsky y col., 2015; Krizhevsky y col., 2012[14]; BVLC AlexNet Model, 2021[9]) como se ha mencionado previamente, es una de las utilizadas en el ámbito de las CNN.

En la figura 2.4 se muestra la estructura de la red AlexNet con las operaciones involucradas, identificando las mismas con indicación de las dimensiones de filtros en las capas convolucionales y totalmente conectadas, para las operaciones de Convolución (*conv*), ReLU (*relu*), Normalización (*norm*), *Pooling* (*pool*), *dropout* (*drop*) o *softmax*. Se incluyen las dimensiones de los filtros, el *stride* (*s*), el *padding* (*p*) y el número de filtros (*K*) en cada capa.

En la tercera columna de la tabla 2.1 se indican los parámetros (pesos) que se aprenden en este modelo, que son los pesos en las capas (primera columna) de convolución (*conv1* a *conv5*) y las totalmente conectadas (*Fully Connected*, fc6, fc7 y fc8), independientemente de que se aplique *dropout* o no en ellas. Obsérvese que como pesos a aprender se incluyen en cada capa de convolución un valor de *bias* por filtro y que es un parámetro de ajuste adicional. Por otra parte, en la tabla 2.1 también se muestran en la segunda columna las dimensiones de salida correspondientes a la capa que se indica.

Nombre de capa	Dimensión de salida	Pesos
conv1	55×55×96	*W* = 11×11×3×96 *b* = 1×1×96
conv2	27×27×256	*W* = 5×5×48×256 *b* = 1×1×256
conv3	13×13×384	*W* = 3×3×256×384 *b* = 1×1×384
conv4	13×13×384	*W* = 3×3×192×384 *b* = 1×1×384
conv5	13×13×256	*W* = 3×3×192×256 *b* = 1×1×256

fc6	1×1×4096	W = 4096×9216 b = 4096×1
fc7	1×1×4096	W = 4096×4096 b = 4096×1
fc8	1×1×4096	W = 1000×4096 b = 1000×1

Tabla 2.1 - Parámetros aprendidos en el modelo AlexNet

Además, en la estructura de la figura 2.4 se indican los números de *Relación*, que establecen precisamente la relación entre las dimensiones de salida de cada capa (o) considerando los parámetros de entrada (i, k, p, s), tal y como se explican previamente y se condensan en la ecuación (2.8), se definen las distintas relaciones posibles, junto con su número correspondiente asociado.

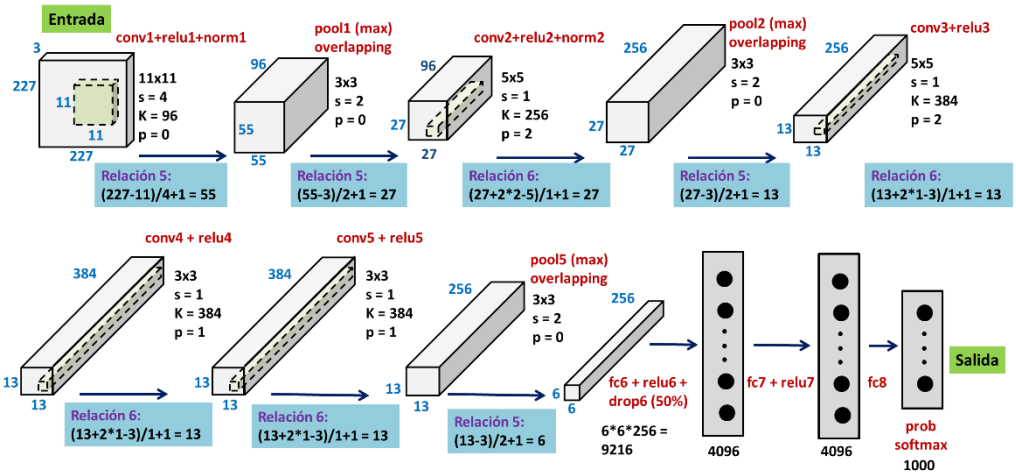


Figura 2.4 - Modelo de red AlexNet

En algunas implementaciones, Matlab (2021) [7], tras la función *relu7* se realiza una operación de *dropout* del 50% modificando las dimensiones de la salida de esta capa.

2.5 GoogLeNet

La red ganadora del concurso ILSVRC[13] 2014 fue GoogLeNet (también conocida como Inception V1) de Google (BVLC GoogLeNet Model, 2021[9]), proviene de la publicación de Szegedy y col. (2014)[18], habiendo logrado una tasa de error de 6.67%. Se trata de un resultado muy cercano al nivel humano, de modo que los organizadores del desafío necesitaron de la intervención experta. Resulta que, en realidad, la evaluación por parte

del experto era bastante difícil de realizar, requiriendo algún entrenamiento adicional para determinar la precisión de GoogLeNet.

La red se inspiró en LeNet, añadiendo un elemento novedoso denominado módulo *inception* (Inc) que consta de varias convoluciones relativamente pequeñas para reducir drásticamente el número de parámetros.

La arquitectura del modelo GoogLeNet consiste en una CNN de 22 capas de profundidad, que consigue una reducción en el número de parámetros de 60 millones (AlexNet) a 4 millones, de esas 22 capas, 9 son de tipo *inception* de distintas categorías. En total el número de capas es de 144, donde cada una de las 9 capas *inception* contiene la estructura mostrada en la figura 2.5, en la que se han incorporado las unidades ReLU presentes en el módulo. El modelo completo propuesto por Szegedy y col. (2014) [18] es el mostrado en la figura 2.6, donde aparecen las distintas capas: convolución (*Conv*) indicando las dimensiones de los filtros; *Pooling*, bien *average* (*AvgPool*) o *máximo* (*MaxPool*) con indicación en este caso también del tamaño de la ventana; Normalización (*LRN, Local Response Normalization*); capas totalmente conectadas (FC) y por supuesto los módulos *Inception* (Inc), en este caso V1 que constituyen la parte nuclear de este tipo de redes. En las capas de convolución y *pooling* se indica también el desplazamiento (*stride*), en este caso se expresa entre paréntesis con el símbolo *s* seguido del correspondiente valor de desplazamiento en el caso '*same*' y V también con el correspondiente valor de desplazamiento, si bien en este caso de tipo '*valid*'. Como puede comprobarse, este esquema presenta dos redes extra, que son exactamente sendos clasificadores auxiliares, y constan de un *pooling* promediado de dimensión 5×5 y *s* = 3 de tipo V que proporciona salidas de tamaños 4×4×512 y 4×4×528 respectivamente. Le sigue una capa de convolución 1×1 con 128 filtros para reducción de la dimensionalidad y una unidad ReLU. Continúa una unidad *dropout* con capas totalmente conectadas finalizando con unidades *softmax* (Softmax0 y Softmax1). La salida final de la red también es una unidad *softmax* (Softmax2).

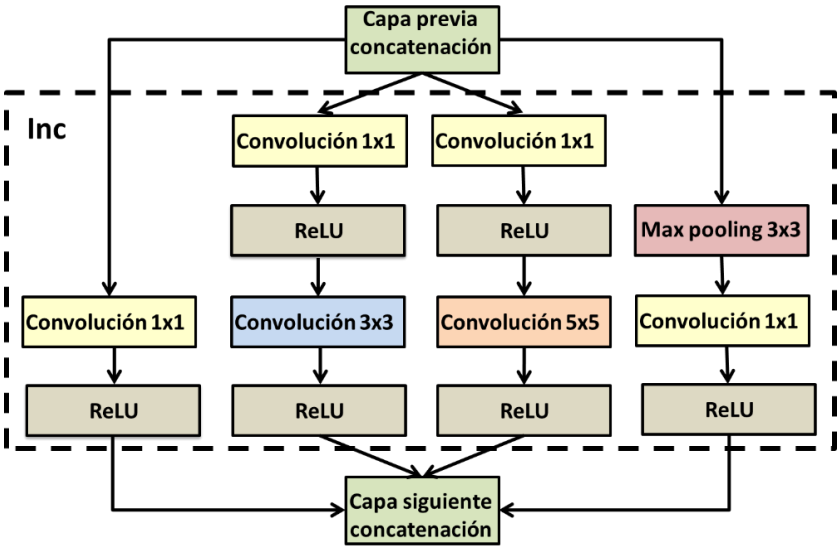


Figura 2.5 - Módulo inception con incorporación de unidades ReLU

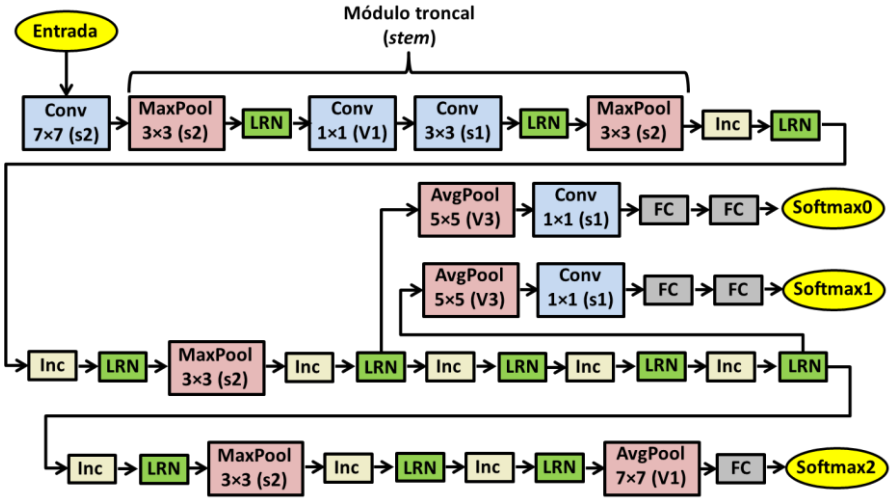


Figura 2.6 - Modelo completo GoogLeNet (inception V1)

3. Diseño y solución desarrollada

Tras la parte introductoria y un detallado análisis técnico-teórico, pasamos a describir el diseño de la aplicación, abordando en primer lugar la arquitectura del sistema bajo el modelo cliente-servidor, para continuar con los aspectos propios de la gestión del proyecto. La licencia de las imágenes utilizadas para la figura 3.1, están en el Anexo 5[A5].

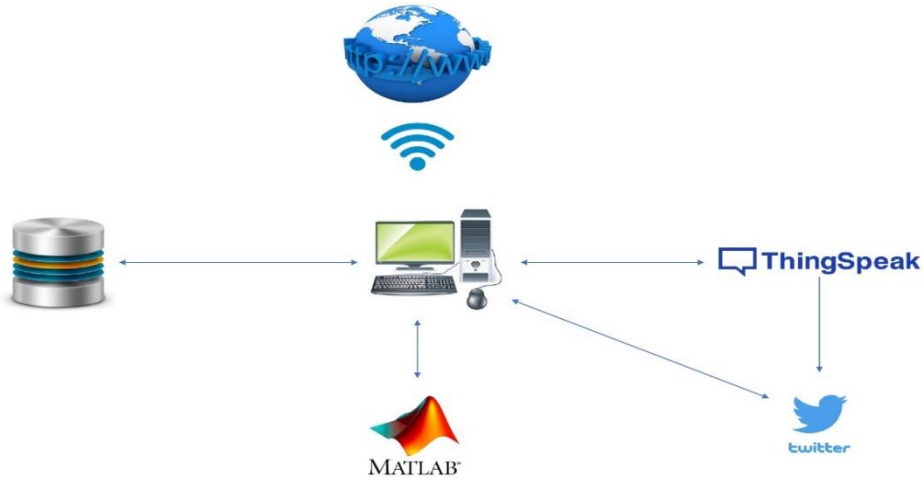


Figura 3.1 - Diagrama de capas

3.1 Arquitectura

La aplicación se basa en la arquitectura cliente-servidor mediante la cual tendremos una parte del servicio en el propio equipo del cliente y realizará una serie de peticiones a un servidor externo.

Está implementada mediante un diseño multicapa, siguiendo el patrón MVC (Modelo-Vista-Controlador), distinguiéndose las siguientes capas:

a) Capa de presentación: En esta capa se integra la interfaz mediante la cual se puede lograr la interacción entre el cliente y la propia aplicación. Trabaja por tanto la parte relacionada con la Vista.

b) Capa de negocio: En la parte de negocio se integra todo aquello relacionado con la lógica del Modelo. Se tienen en este apartado todas las operaciones relacionadas con el tratamiento de datos y las operaciones necesarias para el correcto funcionamiento de la aplicación, incluyendo las redes neuronales.

c) Capa de integración: Esta capa se encarga de la persistencia de datos. En nuestro caso no trabajamos con ninguna base de datos, sino que trataremos esta capa desde una aplicación externa como ThingSpeak[6], donde se almacenarán los datos, y en cualquier caso esto formará parte del lado del servidor. Además de ello, se trabaja con una base de datos relacional, donde de igual manera se tienen almacenados los datos y desde la cual se podrán realizar diversas consultas.

El controlador se encarga de la intermediación entre las capas de negocio y presentación. En nuestro caso se trata de un Controlador de Aplicación (*Application Controller*), siendo único para toda ella y mediante el cual, buscamos mejorar la modularidad y mantenibilidad del código, controlando cada petición de forma independiente.

3.1.1 Lado cliente

En la parte cliente se aloja toda la funcionalidad de la aplicación a excepción de la parte relacionada con ThingSpeak[6] y Twitter.

Para modelar el proyecto y para su posterior desarrollo, se han utilizado distintos patrones de diseño, descritos a continuación:

- **Application Controller:** Se encarga de tramitar y gestionar los eventos que producen las interfaces. En el proyecto ha sido desarrollado para comunicar la capa de presentación y de negocio.
- **Application Service:** Se encarga de mantener la lógica del sistema. En el proyecto se ha utilizado para ejecutar los Comandos de la aplicación de forma correcta.
- **Command:** Se encarga de solicitar una operación a un objeto, encapsulando la información que transporta para realizar dicha operación y sin conocer el receptor de los datos de salida. En este proyecto se ha utilizado para ejecutar operaciones que no implican cambios en la capa de presentación.
- **Dispatcher View:** Se encarga de la recepción de los Eventos producidos por la capa de Presentación y su correcta redirección hacia la interfaz correspondiente.
- **Singleton:** Se encarga de impedir la creación de más de un objeto de la misma clase. Además de impedirlo, proporciona un único acceso a la clase creada, asegurando así que únicamente haya una única instancia. En el presente

proyecto se ha utilizado para el controlador de la aplicación y el dispatcher de vistas.

- **Transfer:** Este patrón se encarga de transportar datos entre distintas capas, encapsulando la información que contiene y permitiendo el transporte de múltiples datos en una única vez.

Las funcionalidades tratadas desde el cliente son:

- **Entrenamiento y reentrenamiento de red (AlexNet o GoogLeNet)**
  Una de las primeras opciones dentro de la aplicación es la de reentrenar la red ya sea AlexNet o GoogLeNet. Desde la pantalla principal o el menú de navegación lateral seleccionaremos la red a utilizar y la opción Training para ello.

  En ese momento el Controller comprueba si se trata de una Vista para llamar al Dispatcher o de un comando perteneciente a la lógica de negocio, en cuyo caso llamará al CommandFactory.

  En nuestro caso, en primer lugar, se llamará a la Vista perteneciente a Training de la red seleccionada, y después, tras seleccionar la opción Start Training, a la Vista de Selección de parámetros para el entrenamiento, para ello se envía al Dispatcher el evento de dicha Vista.

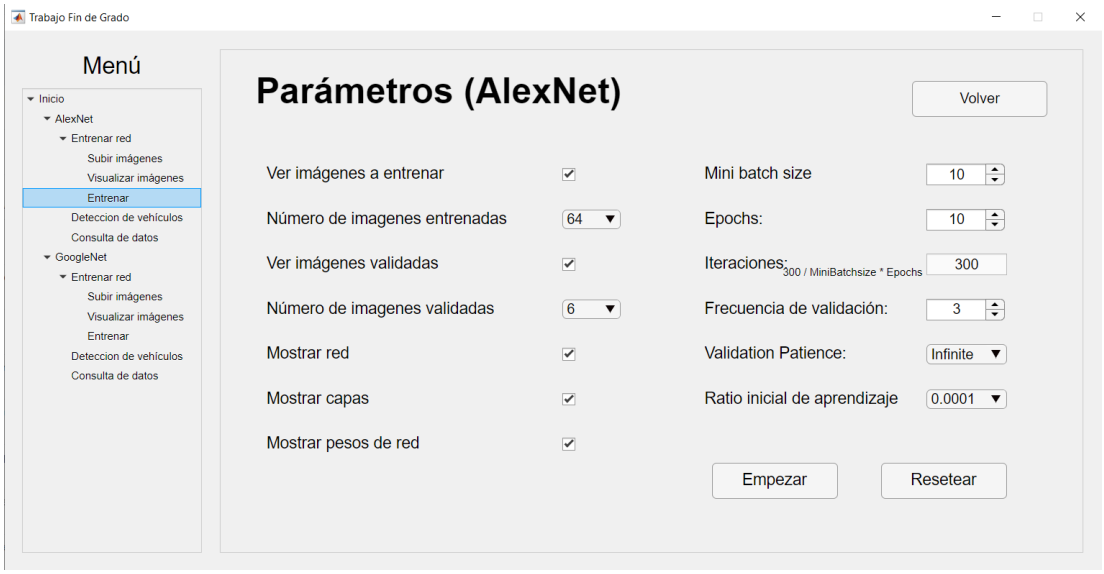


Figura 3.2 - Selección de parámetros

Estos parámetros serán almacenados en un transfer para pasarlos a la lógica de negocio del entrenamiento, llegarán junto al modo que determina si se trata de AlexNet o GoogLeNet. La función también cuenta con dos atributos que determinan la cuadrícula a construir para mostrar las imágenes utilizadas en el entrenamiento.

Una vez seleccionados los valores de dichos parámetros se vuelve al Controller, que en este caso recibe un comando para entrenar en AlexNet o en GoogLeNet, por lo que llama al CommandFactory y éste lanza el entrenamiento de la red, guardándose los resultados de este en un transfer para ser usado posteriormente en la detección de vehículos.

- **Subida de nuevas imágenes**

  Mediante esta función añadimos nuevas imágenes que ayuden al entrenamiento de la red al contar con una mayor variedad de modelos para mejorar su porcentaje de acierto.

  Al igual que en la anterior función se llamará al Controller con el evento, que en este caso pertenecerá a una Vista, figura 3.3. Se puede seleccionar la imagen a subir y el tipo de imagen según la clase a la que pertenece el vehículo.

  Una vez seleccionada, si la almacenamos en nuestro banco de imágenes, lo primero que se realiza es una redimensión de esta para que todas las imágenes a utilizar en el posterior entrenamiento sean del mismo tamaño, concretamente del requerido por el modelo de red a utilizar. Y una vez realizada dicha redimensión se añade en el directorio perteneciente al tipo de imagen seleccionada.

  A la hora de ser almacenada se tiene en cuenta el tipo de vehículo que se encuentra en la imagen y la cantidad de imágenes de este tipo de vehículo que ya existen en el directorio, para asignarle un nombre único que se va incrementando con los distintos números de imágenes que se encuentran ya en el directorio, de forma que no haya problemas con el nombrado de imágenes, y puedan ocurrir algún tipo de conflicto en el almacenamiento.

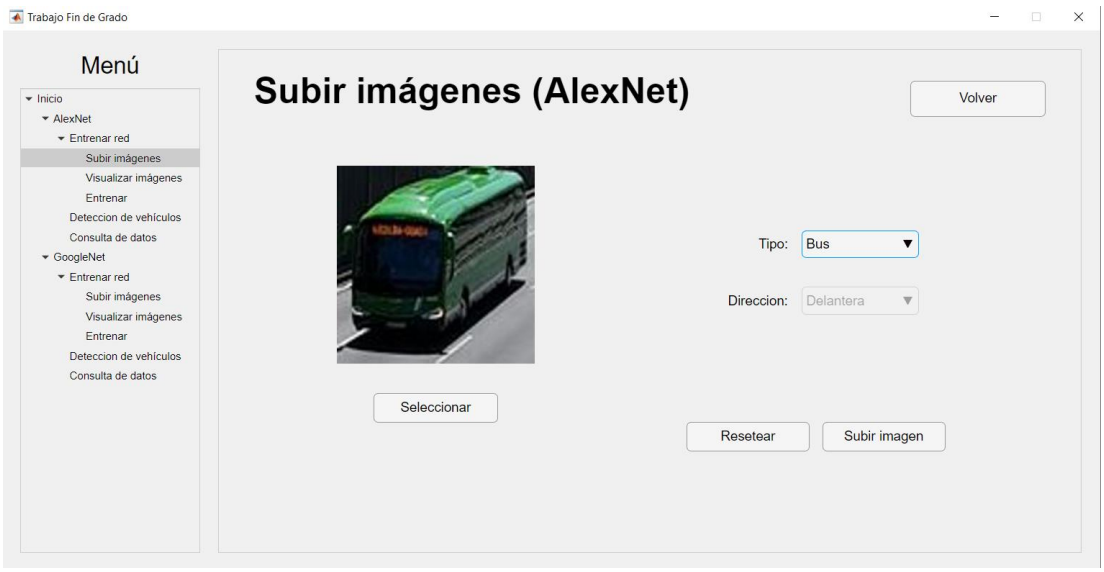


Figura 3.3 - Subida de imágenes

- **Visualización de imágenes**

  El caso de la visualización de imágenes es más sencillo, de igual manera se manda una orden al controlador que determinará que se está indicando un comando en lugar de una vista y nos redirigirá al CommandFactory. Después se recogerá el evento que dependerá de si se quieren visualizar las imágenes para AlexNet o GoogLeNet, puesto que ambas redes operan por separado, y por último abrirá el explorador con la ruta relativa para visualizarlas.

- **Detección de vehículos**

  Es el caso más amplio y la función más extensa de la aplicación desarrollada. El usuario puede realizar la detección de vehículos utilizando cualquiera de las dos redes entrenadas con las que cuenta la aplicación, una vez seleccionada por el usuario, se le mostrará una ventana similar a la que aparece en la figura 3.4:

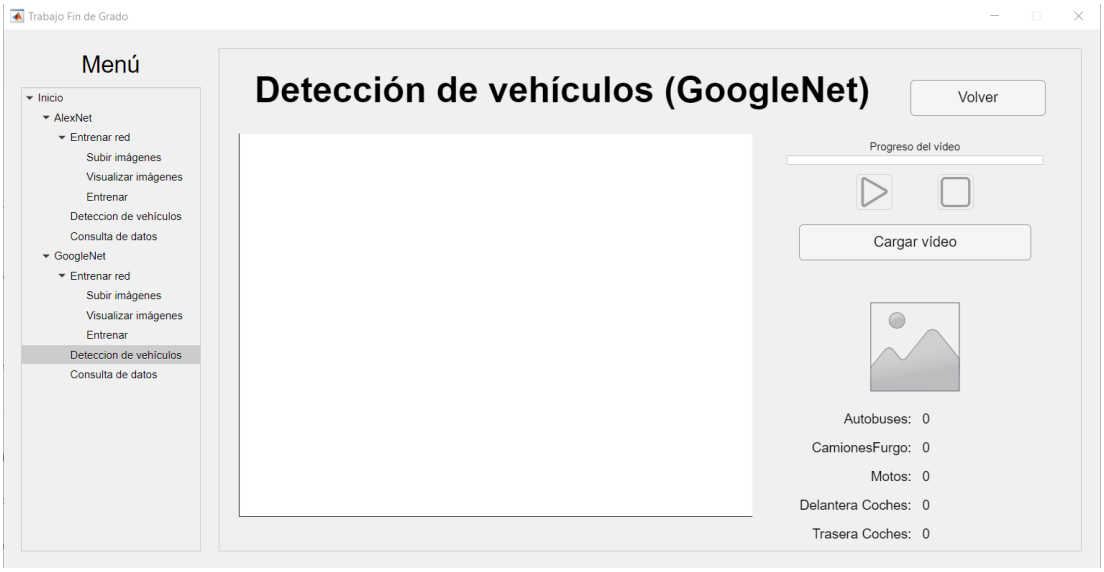


Figura 3.4 - Pantalla de detección de vehículos

El usuario podrá entonces seleccionar un video de su elección desde el botón "Subir vídeo", a continuación, se cargará el video en la ventana de previsualización del mismo y el usuario podrá controlar su reproducción a través de los botones de "*play*", "pausa" y "*stop*". Mientras se ejecuta el video mostrando fotograma a fotograma, en la parte inferior derecha se van mostrando los datos según la detección realizada en el video y una captura de la posición y vehículo detectado.

Cuando el video finaliza o el usuario pulsa el botón de "*stop*", los datos recolectados se envían a través del controlador a una nueva ventana de la aplicación para poder realizar el envío a la API implementada y guardar los datos obtenidos a la vez que publicarlos en Twitter, figura 3.5.

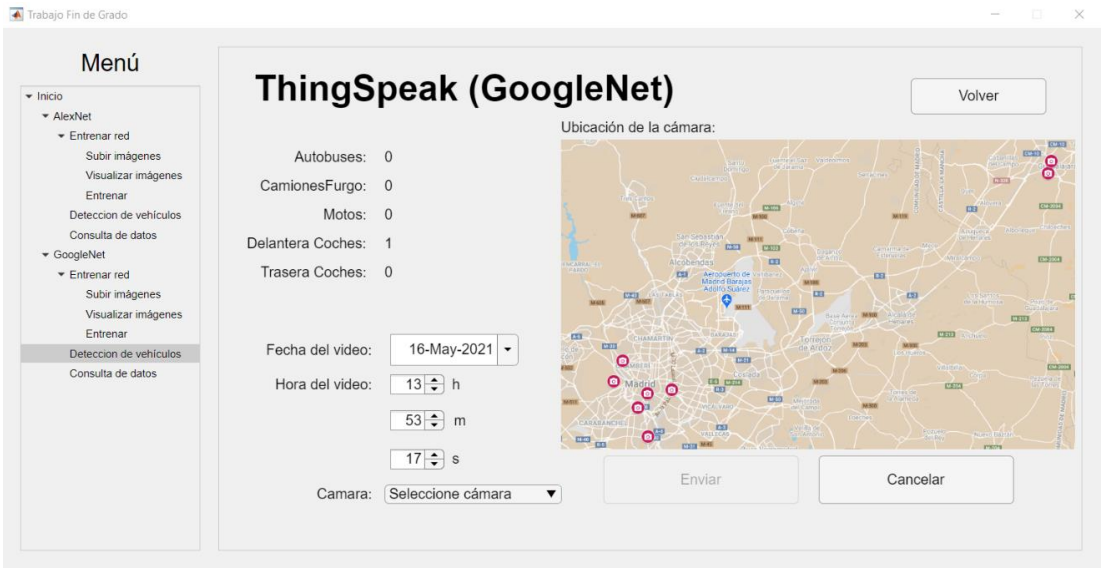


Figura 3.5 - Pantalla de envío a Twitter

A través de esta interfaz el usuario podrá seleccionar la cámara y fecha de la grabación que se ha seleccionado para realizar la detección de los vehículos y a través del botón "Enviar" podrá subir los datos obtenidos a la base de datos creada en el proyecto y publicar el *tweet* correspondiente. Si en cambio, el usuario no quiere realizar dicha subida, pulsará el botón "cancelar" o "volver" desechando así los datos obtenidos.

3.1.2 Lado servidor

Esta parte abarca lo relacionado con el tratamiento de los datos extraídos, en un primer momento se hace uso de ThingSpeak[6].

ThingSpeak[6] es una plataforma de *Internet of Things* que permite analizar, visualizar y actuar sobre los datos disponibles y previamente cargados en la plataforma. Permite la creación de canales para almacenar los datos que se recopilan de la aplicación. En cada canal se pueden añadir hasta ocho campos que pueden contener cualquier tipo de datos y otros cuatro para datos más específicos, tal como la ubicación.

Una vez creado el canal se generan automáticamente dos API *keys* que permiten escribir datos en un canal o leer datos de un canal privado, estas claves se añadirán a la aplicación de Matlab[7] para vincular a ella los canales.

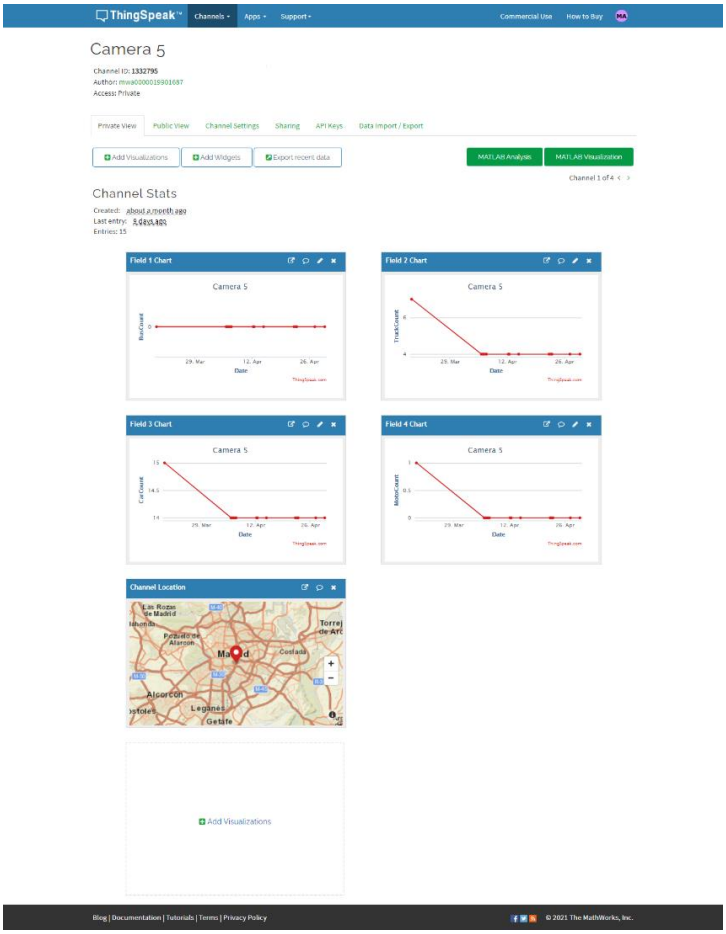


Figura 3.6 - Canal de ThingSpeak

Una de las funcionalidades planteada como objetivo del proyecto es poder compartir la información obtenida a través de Twitter, para ello ThingSpeak[6] tiene una aplicación llamada ThingTweet que vincula la cuenta de ThingSpeak[6] con la de Twitter.

No obstante, en las pruebas realizadas se comprueba que los resultados no son totalmente satisfactorios, puesto que, aunque en el caso de querer realizar una serie de posts asignando un intervalo de tiempo, el funcionamiento era correcto, no podíamos programarlos en caso de que solo se realizaran al hacer una nueva inserción de datos. Esto se debe a que React sólo permitía crear uno que cumpliera con la restricción dada y enviando un nuevo Tweet al insertar datos, lo que hace que no sea posible tener todas las hipotéticas cámaras enviando las nuevas inserciones de datos en cada momento.

Aun así, se mantiene lo creado con ThingSpeak[6], puesto que podría resultar interesante en un futuro si se decide trabajar con cámaras que estén grabando continuamente y se quiere enviar un post con una determinada frecuencia de tiempo, de modo que sea posible observar en tiempo real el tráfico rodado en una o varias ubicaciones.

Tras este problema inicial, se buscó una nueva solución que resolviera dicha dificultad. Para ello se realiza una investigación exhaustiva para buscar la forma de almacenar los datos generados por la aplicación, que se pudiera escalar sin ningún límite para posibles mejoras en el futuro, publicara *tweets* de forma automática con cada inserción de datos en el servidor y además contara con la sencillez de obtención de los datos insertados y

posibilitara la creación de seguridad para controlar de forma satisfactoria los accesos a estos datos publicados.

Por ello, se decidió realizar la programación de un pequeño servidor en lenguaje PHP para la construcción de los accesos y con SQL para el almacenamiento de los datos.

El almacenamiento en SQL se ha diseñado de la forma más simple posible. Cuenta con dos tablas: en una de ellas se guarda toda la información necesaria para cada cámara, su nombre y su ubicación en latitud y longitud; y en la otra tabla se almacenan todos los datos que va produciendo la aplicación, el número de vehículos de cada tipo que han sido detectados (Buses, Camiones, Coches y Motocicletas), la fecha y hora de la detección y la cámara que ha realizado la grabación del video.

En la figura 3.7 se puede observar el modelo de las tablas en la base de datos y su relación a través del id de Cámara.

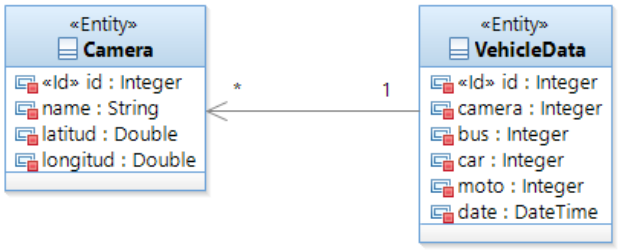


Figura 3.7 - Diagrama del modelo de dominio

- El diseño de la API se ha realizado en PHP debido a que cumple con los requerimientos exigidos, a la vez que permite implementar seguridad para la escritura de datos en la base de datos y controlar de esta forma los accesos. Para ello se contaba con un servidor que permitía la ejecución de PHP. Además, al construir una API externa nos permite escalar y construir una infraestructura para servir datos a distintas aplicaciones y webs en un futuro, de esta forma se pueden predecir posibles atascos y número de vehículos en una zona determinada cuando la aplicación opere en un entorno de ciudad inteligente.
- La API en PHP cuenta con dos *endpoints*, uno de escritura y otro de lectura, la documentación y especificación de la API está disponible a través de internet http://api.brogame.es/madridControl
- Lectura: Es accesible a través de "/thingspeak" con un método GET, para la correcta lectura de datos se deben utilizar los parámetros en formato *query*:
    - **Select:** Indica los campos que se desean obtener.
    - **Where:** Indica los filtros y reglas que deben cumplir los datos que se obtendrán.
    - **Order:** Indica el orden en el que los datos serán obtenidos de la API.

Por tanto, un ejemplo de consulta hacia la API utilizando los parámetros anteriores seria:

http://api.brogame.es/thingspeak?select=bus,camera&where=bus>=0and bus<=10and camera=&order=camera

- Escritura: Es accesible a través de "/thingspeak" con un método POST, para introducir datos de forma correcta se deben utilizar los parámetros en el cuerpo de la petición:
    - **Camera:** Contiene el id de la cámara que envía los datos
    - **Bus:** Número de autobuses detectados en la grabación.
    - **Truck:** Número de camiones detectados en la grabación.
    - **Car:** Número de coches detectados en la grabación.
    - **Moto:** Número de motos detectados en la grabación.
    - **Date:** Fecha de la grabación realizada.
    - **ApiKey:** Al ser una operación de escritura, se ha implementado seguridad para controlar el acceso y que solo puedan añadir datos ciertos usuarios.

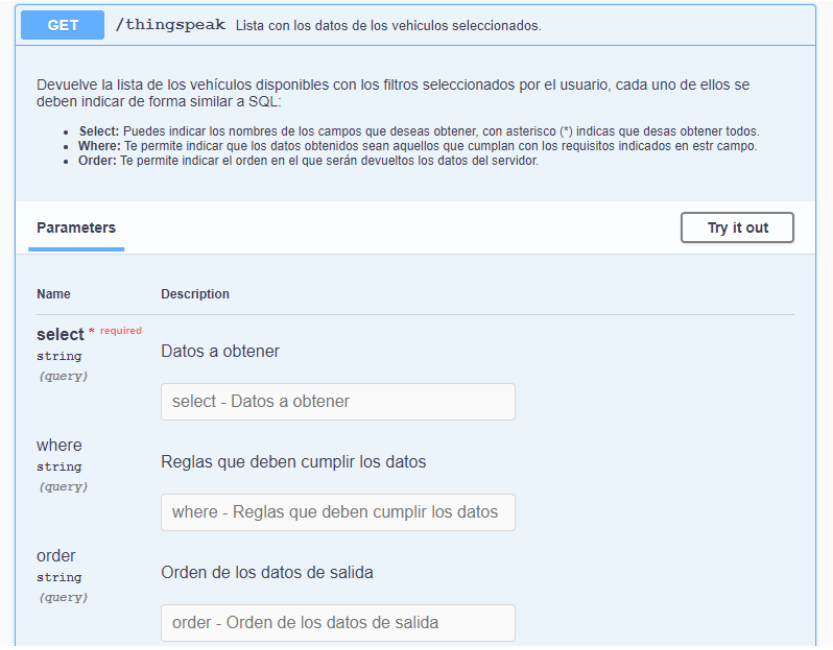


Figura 3.8 - Petición GET de lectura de la API

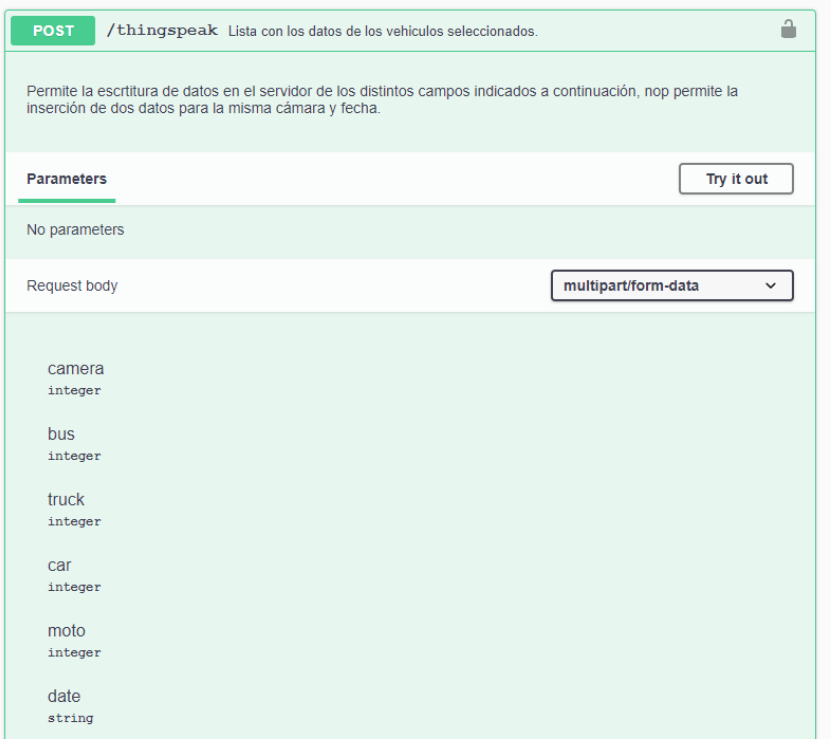


Figura 3.9 - Petición POST de escritura de la API

3.2 Gestión del proyecto

Al iniciarse el proyecto, una de las primeras cuestiones a resolver fue el tipo de metodología a utilizar en el mismo. Desde el principio se desecharon las opciones tradicionales de gestión de proyectos (ITIL, RUP...), sobre todo debido a las restricciones que generan, como son: equipos grandes, gran cantidad de roles, proceso rígido que encaja mal ante posibles cambios, entre otras.

Posteriormente, se estudiaron los diversos sistemas de metodologías ágiles, buscando aquellas características más cercanas a nuestras necesidades.

Finalmente, al tratarse de un equipo pequeño hemos gestionado el proyecto utilizando una adaptación de Scrum a las características propias de nuestro proyecto, que pasamos a explicar.

En primer lugar, denotar que los tres miembros del equipo hemos actuado al mismo nivel, no existiendo en sí misma la figura de Scrum Máster, si bien al coordinar cada integrante un área distinta del proceso, éste era el que ejercía en cierta forma dicha función con cada una de las partes del trabajo. En nuestro caso el Product Owner o cliente se asignó al director de este proyecto, quien determinaba mediante reuniones periódicas si el trabajo realizado hasta la fecha cumplía las especificaciones y el propósito buscado en la elaboración de este. Igualmente, todos los miembros del equipo realizamos trabajo de equipo de desarrollo, trabajando a partes iguales tanto en la elaboración del código referente a la aplicación como en la redacción de la memoria.

Los elementos Scrum utilizados para la evolución del trabajo son los siguientes:

- **Reunión de planificación:** Se realizan mensualmente, en ella decidimos las funcionalidades a realizar en el siguiente sprint, determinándose si el orden de las tareas en el Product Backlog sigue siendo el adecuado o bien necesita modificaciones.
- **Daily Scrum:** Realizadas semanalmente, en el fin de semana siempre debido a los horarios de trabajo dispares de los miembros del equipo. Las reuniones realizadas mediante Google Meet, siempre empiezan con una muestra de los avances realizados por cada miembro del equipo con su explicación para la comprensión por parte de los otros integrantes del equipo. Posteriormente se unifican los distintos procesos en caso de ser necesario. Tras ello se comparten los problemas encontrados durante la semana por si sucedieran en otro momento del proceso. También se aprovecha para trabajar en conjunto determinadas tareas.
- **Sprint:** Los sprint de Scrum determinan el compromiso del equipo sobre el trabajo a realizar entre las reuniones de revisión. En nuestro caso, tras cada reunión de revisión, y al término de esta, se trataba sobre el trabajo a realizar en el siguiente sprint. Como cada reunión de planificación era mensual, los sprints tenían esa misma duración, seleccionando las tareas a realizar durante el mismo.
- **Reunión de revisión:** Mensualmente tras cada sprint y también de forma telemática se realiza una reunión de revisión, en ella comprobamos el nivel de completitud de los objetivos marcados durante el sprint, se debate de forma más amplia todos los puntos problemáticos y favorables, observados durante el mismo, de forma que se pueda mejorar la gestión del siguiente sprint.

  Igualmente se realiza la integración necesaria para las labores realizadas por cada miembro del equipo y se realizan pruebas de ejecución sobre la parte del trabajo ya realizado que debe estar operativo en ese momento.
- **Reunión de retrospectiva:** De forma periódica, sin un rango de fechas concreto se realizan reuniones de retrospectiva, en las que el equipo muestra el producto al Product Owner. En dicha reunión, se muestran los avances realizados y se recibe feed-back sobre los mismos. Tras ello se deciden los cambios o mejoras a realizar y los siguientes pasos a seguir en el proceso.

Los artefactos Scrum utilizados son:

- **Product Backlog:** Determina las distintas Historias de Usuario (HU), que es necesario desarrollar. Cada una de ellas trata sobre una funcionalidad

existente en la aplicación. Se ordenan según su importancia y es utilizada por el equipo para conocer la totalidad del trabajo a realizar. En el Anexo 1[A1] se puede observar el Product Backlog completo.

- **Sprint Backlog:** Descompone las HU en pequeñas tareas, que unidas la completan. Se trabaja mediante la herramienta Trello, en la que definimos las tareas ya realizadas, aquellas que están en proceso y las que no han sido comenzadas. Gracias a ello podemos tener un seguimiento visual del trabajo realizado y la parte restante.

Decidimos no utilizar Burndown Chart y observar el progreso del equipo desde el Sprint Backlog, figura 3.10.

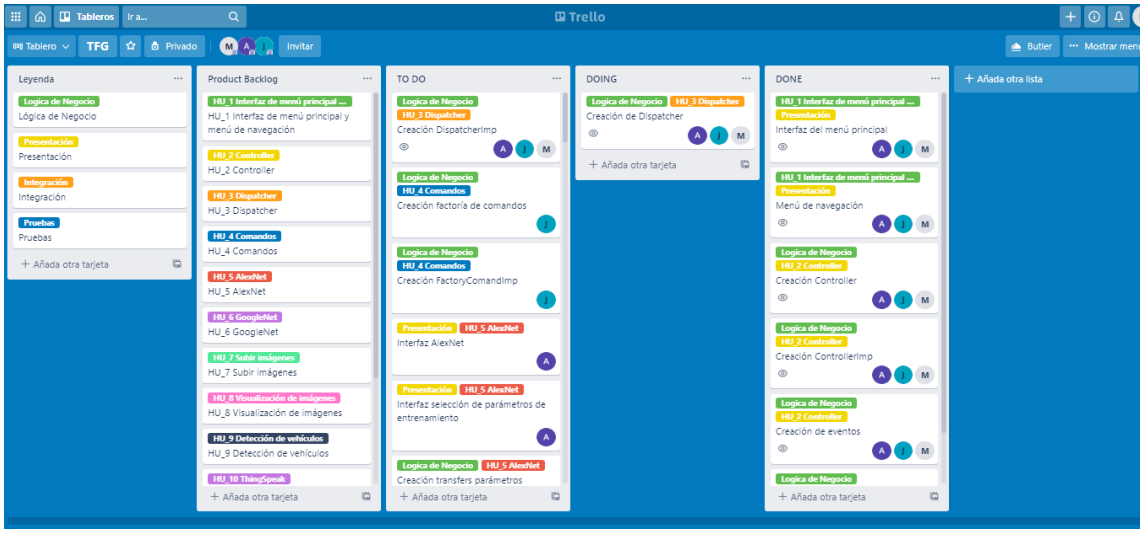


Figura 3.10 - Sprint Backlog

3.3 Herramientas utilizadas

- MATLAB R2020B
  - **ThingSpeak Support Toolbox[30]:** Complemento de Matlab[7] utilizado para crear, leer, actualizar y eliminar datos de ThingSpeak[6]. Utilizado en un principio en el proyecto para enviar los datos de los vehículos detectados a Thingspeak[6], extraerlos y realizar predicciones con ellos. Con los problemas que surgieron con esta plataforma, este complemento fue desechado en la parte final del desarrollo del proyecto.
  - **Mapping Toolbox[31]:** Complemento de Matlab[7] que proporciona algoritmos y funciones para transformar datos geográficos y crear visualizaciones de mapas. En nuestro caso hemos utilizado la

visualización de mapas para mostrar la ubicación exacta de las distintas cámaras de tráfico utilizadas en el proyecto.

- **Symbolic Math Toolbox:** Es un complemento de Matlab[7] que proporciona funciones para resolver, manipular y representar gráficamente ecuaciones de matemáticas simbólicas. En el proyecto se ha utilizado en la parte de presentación para mostrar correctamente símbolos utilizados y operaciones de una forma adecuada para el usuario de la aplicación.

- **Statistics and Machine Learning Toolbox**[31]**:** Proporciona funciones y aplicaciones para analizar y modelar datos. Sus algoritmos permiten extraer inferencias a partir de datos y crear modelos predictivos.

- **Simulink**[34]**:** es una herramienta de Matlab[7] para la simulación de sistemas dinámicos. Se construye el modelo a simular mediante la técnica drag and drop.

- **MATLAB Compiler**[35]**:** permite compartir programas Matlab[7] como aplicaciones. Utilizado en nuestro proyecto para la generación del ejecutable externo para poder abrir la aplicación creada sin necesidad del compilador de Matlab[7].

- **MATLAB Compiler SDK**[36]**:** Herramienta que amplía las funciones de Matlab Compiler permitiendo crear librerías compartidas de otros leguajes de programación. Utilizado en el proyecto para la generación de las dependencias que utilizaba la aplicación y añadirlas al ejecutable externo.

- **MATLAB Support for MinGW-w64 C/C++ Compiler**[37]**:** es un conjunto de compiladores para Windows basado en GNU. Incluye un compilador GCC y herramientas para compilar aplicaciones C y C ++ para Windows. Utilizado en nuestro proyecto para la generación del ejecutable externo para poder abrir la aplicación creada en cualquier sistema operativo Windows sin necesidad del compilador de Matlab[7].

- **Image Processing Toolbox**[38]**:** Complemento de Matlab[7] que proporciona algoritmos y apps para el procesamiento, análisis y visualización de imágenes. También se utiliza para segmentación de imágenes, mejora de imágenes, reducción de ruido, transformaciones geométricas, registro de imágenes y procesamiento de imágenes 3D.

- **Deep Learning Toolbox, Deep Learning Toolbox Model for AlexNet Network** y **Deep Learning Toolbox Model for GoogLeNet Network**[39]**:** Estos tres complementos nos proporcionan un marco para diseñar e implementar redes neuronales profundas con algoritmos, modelos previamente entrenados y apps.

En nuestro caso lo utilizamos para redes convolucionales, concretamente para AlexNet y GoogLeNet, por ello además del módulo genérico utilizamos cada uno de los módulos específicos para ambas redes.

  - **Computer Vision Toolbox:** Módulo que proporciona algoritmos, funciones y apps para diseñar y probar sistemas de procesamiento de vídeo, visión artificial y visión 3D. Permite realizar detección y seguimiento de objetos, así como detección, extracción y coincidencia de características.

- **PHPMYADMIN:** Uso de esta infraestructura de almacenamiento en lenguaje SQL, para crear y guardar los datos de detección obtenidos de la aplicación, así como los datos escalados para realizar predicciones de una forma rápida, sencilla y segura.

- **Microsoft Office:**
  - **Word:** Uso de Microsoft Word para la realización de la memoria. En concreto se utilizó la versión de OneDrive de forma que todos los miembros del equipo pudiésemos trabajar en línea sobre el mismo documento.
  - **OneDrive:** Nube de Microsoft en la que almacenamos los documentos a utilizar durante la elaboración del proyecto.

- **GIT:** Herramienta utilizada para el control de versiones y almacenamiento de código en la nube para poder realizar un desarrollo colaborativo entre nosotros, poder paralelizar el trabajo entre distintos módulos y realizar recuperaciones de código en caso de fallos críticos.

- **SQL:** Lenguaje de programación elegido para realizar la inserción, obtención, actualización y eliminación de los datos generados por la aplicación y que son almacenados en una base de datos relacional.

- **WhatsApp:** Uso de la aplicación de mensajería para realizar las comunicaciones necesarias, consultas y dudas entre nosotros de una forma rápida. También para poder realizar los distintos elementos de la gestión de proyecto como pequeñas actualizaciones del sprint que estábamos realizando y la transferencia de pequeños archivos.

- **Google Meet:** Herramienta utilizada tanto para las reuniones de equipo realizadas semanalmente, como para las reuniones con el tutor para validar los avances del proyecto y siguientes pasos a seguir. Además, utilizada para resolver dudas entre nosotros y realizar consultas para una posible mejora del proyecto.

- **Api Rest:** Web que se encarga de acceder a los datos de la aplicación y mediante distintas peticiones con distintas direcciones; crear, modificar, actualizar, obtenerlos o eliminarlos de una forma segura. En nuestro caso esta implementación se realizó cuando ThingSpeak[6] no cumplía con los requisitos que teníamos.

- **OpenAPI[40]:** Aplicación que se utiliza para crear un standard de definición para cualquier API, con código abierto y programado en Yaml y JSON y que además cuenta con un generador de código que a partir de esta definición permite crear el código de un lenguaje que elija el usuario.

- **POSTMAN:** Herramienta usada para el testing de API Rest, se realizaron tanto test simples como avanzados realizando distintas peticiones a la API Rest creada y probar así tanto el almacenamiento, obtención, actualización y eliminación de los datos como las publicaciones automáticas en Twitter a través de su portal de desarrollo.

- **IBM RSA[41]:** Herramienta utilizada para crear el diseño del modelo de dominio, diagramas de clases y casos de uso y para poder modelar la estructura del programa creado para presentarlo en el documento de la memoria.

- **Google My Maps:** Aplicación de Google perteneciente a Maps utilizada para crear un mapa privado y colaborativo[42] para el equipo con el fin de poder mostrar y guardar las ubicaciones de las distintas cámaras de tráfico utilizadas en la detección de vehículos implementada en la aplicación.

- **ThingSpeak (Descartado):**
  - **Thingspeak[6]:** Plataforma de análisis de IoT que permite agregar, visualizar y analizar flujo de datos en una nube. Fue la primera opción para la persistencia de datos, al tener múltiples opciones de tratamiento de los mismos.
  - **ThingTweet:** Aplicación nativa de Thingspeak[6] para la integración de Twitter con ésta, que permite enviar tweets automatizados con los datos elegidos por el usuario.
  - **React:** Disparador para determinados eventos en ThingSpeak[6] cuando se cumpla una regla o en un hora o minuto determinado. En nuestro caso se crean disparadores para enviar posts a Twitter cuando un dato es guardado en la base de datos de ThingSpeak[6].
  - **Matlab Analysis:** Módulo de Matlab[7] para la creación de scripts que se utilizarán para crear los mensajes a postear en Twitter y poder estructurar los datos que se publican de una forma más visual para el usuario de la web.

- **Twitter:**
    - **Web**[43]**:** Red social basada en la publicación de mensajes rápidos, pudiéndose enviar únicamente mensajes con un máximo de 280 caracteres. Utilizada en el proyecto para comprobar la correcta publicación de los tweets generados automáticamente por la aplicación y para crear la cuenta desde la que serán publicados.
    - **API REST:** API proporcionada por Twitter que permite acceder a leer y escribir datos de Twitter de forma segura proporcionando distintas claves de lectura y escritura para la cuenta que hemos creado en su plataforma.
    - **Twitter Developer Portal:** Portal para desarrolladores en Twitter en el que se explica cómo crear tweets automatizados y realizar un extenso registro de los datos que han sido publicados desde nuestra cuenta, haciendo así mucho más sencilla la identificación de cualquier fallo.
- **Trello**[44]**:** Software de administración de proyectos, permite mediante un tablón virtual obtener una vista rápida de las tareas a realizar, aquellas que están en pleno proceso y las que aún no se han iniciado.
- **Flaticon:** Aplicación web a través de la cual obtenemos los iconos de licencia libre necesarios para la capa de presentación de nuestra aplicación.
- **Cámara NIKON**[45]**:** Hardware utilizado para realizar la captura y grabación de los videos utilizados tanto en la fase de desarrollo como de pruebas para la detección de vehículos con la red con la que cuenta la aplicación. Especificaciones.
- **iMovie:** Programa de edición de video utilizado para manipular los videos grabados para la posterior realización del flujo de tráfico.
- **Ordenadores:** Se trabaja desde los ordenadores personales de los tres miembros del equipo, realizando todas las operaciones desde ellos.
- **Balsamiq:** Aplicación destinada a generar Mockups que sirvan de modelo para las interfaces de la aplicación.
- **PhpStorm**[26]**:** Programa de desarrollo de software utilizado para realizar tanto la programación de la API del proyecto, como las pruebas en local con casos límites y poder a su vez realizar un Debug del código para encontrar posibles fallos de una forma sencilla, además al contar con un compilador nos permite desarrollar de una forma correcta.

- **PHP 7.4:** Lenguaje utilizado en lado servidor utilizado para ser incrustado en HTML y programar con ello nuestra API Rest, hemos utilizado el framework Slim[46] que permite escribir APIs en PHP de una forma rápida, sencilla y segura.

- **Swagger Editor**[47]: Herramienta utilizada para documentar API Rest en formato OpenApi y ver los cambios en tiempo real en una página en formato web. En nuestro caso documenta el API realizado en PHP para la integración de los datos.

4. Resultados obtenidos

4.1 Entrenamiento de redes

Se realizaron tres tipos de entrenamiento para videos procedentes de tres cámaras distintas, de forma que se pudiera observar el nivel de eficacia alcanzado en cada entrenamiento y las diferencias existentes al modificar los parámetros involucrados en dichos entrenamientos.

A. Entrenamiento bajo. (A-1 AlexNet y A-2 GoogLeNet)

El primer tipo de entrenamiento está basado en proponer unos parámetros que optimicen la velocidad de entrenamiento en detrimento de la eficacia de aciertos. Los valores propuestos para el mismo son:

- Nº Imágenes de entrenamiento: 9
- Nº Imágenes de validación: 4
- Epochs: 8
- Iteraciones: 240/120
- Razón de aprendizaje: 0,001

Como se observa en la figura 4.1, para los parámetros de entrenamiento de nivel bajo en la red AlexNet se puede apreciar un porcentaje de precisión (*validation accuracy*) realmente bajo, concretamente finaliza con un valor promedio del 18,18%. El tiempo de entrenamiento ha sido en este caso de 7 minutos y 33 segundos, ejecutándose sobre una única CPU. Se observa también una estabilidad total en lo que respecta tanto a precisión como a la función de pérdida (*loss*), prácticamente a partir de la segunda epoch, siendo en ambos casos dos situaciones un tanto anómalas. En el caso de la función de pérdida no desciende por debajo de aproximadamente del valor 1, cuando lo ideal sería 0. Ambas consideraciones hacen que durante el proceso de detección el nivel de fallos sea elevado.

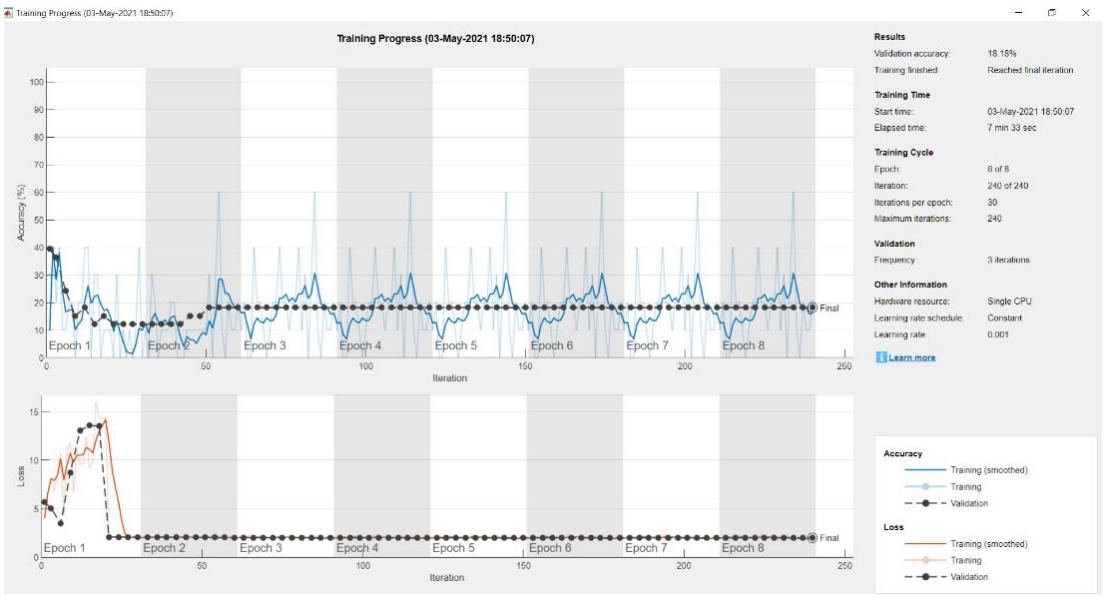


Figura 4.1 - Entrenamiento A-1 Alexnet

Por otra parte, en la figura 4.2, se puede observar la buena ratio de aciertos obtenida por GoogLeNet desde el nivel de entrenamiento más bajo, el porcentaje de aciertos, o más específicamente la precisión en la validación (*validation accuracy*) de esta red neuronal es del 90,91 %. Un porcentaje muy aceptable teniendo en cuenta los valores tan bajos de los parámetros introducidos. Por otro lado, se observa, en este caso, cómo la función de pérdida se sitúa continuamente en las proximidades del cero, llegando a alcanzar finalmente este valor, lo que, unido a lo anterior, permite afirmar que el proceso de aprendizaje ha resultado altamente satisfactorio. El proceso de entrenamiento ha tardado exactamente 34 minutos y 19 segundos, también en una única CPU.

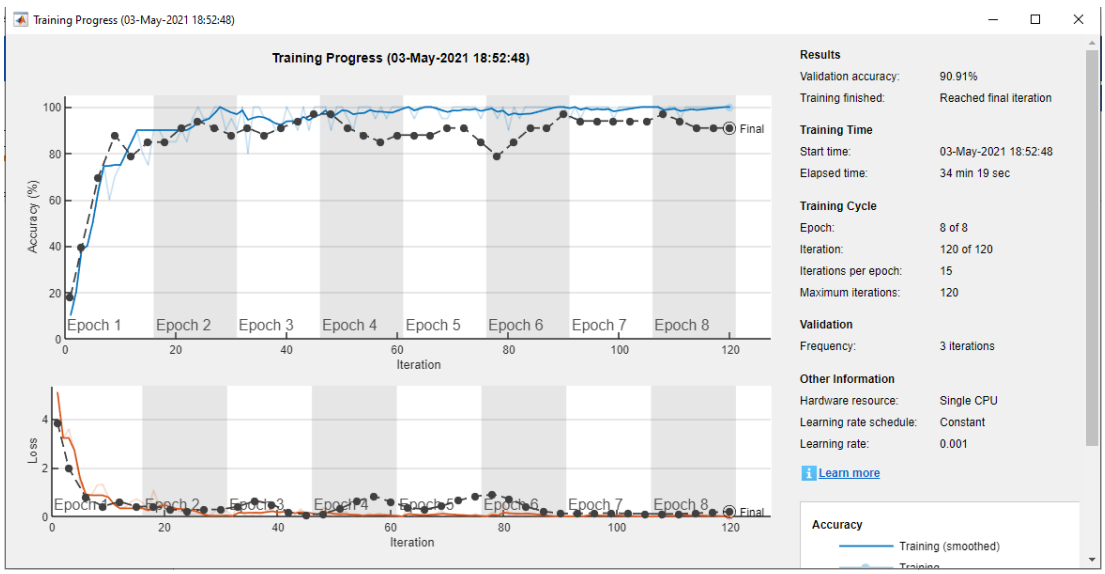


Figura 4.2 - Entrenamiento A-2 GoogLeNet

Comparando ambas figuras 4.1 y 4.2 se determina que para los mismos parámetros de entrenamiento la red GoogLeNet es considerablemente más lenta que AlexNet y ofrece una mayor precisión.

B. Entrenamiento medio. (B-1 AlexNet y B-2 GoogLeNet)

Para este entrenamiento se busca la mejor relación tiempo/aprendizaje, utilizando unos parámetros que ayuden a conseguir una mayor precisión que la obtenida anteriormente sin un coste de tiempo muy elevado. Dichos parámetros son:

- Nº Imágenes de entrenamiento: 30
- Nº Imágenes de validación: 12
- Epochs: 25
- Iteraciones: 750
- Razón de aprendizaje: 0,0001

En este nivel de entrenamiento, para la red AlexNet se aprecia un aumento significativo de la precisión (90,91%) respecto al entrenamiento anterior que se situaba en un 18,18%, como se muestra en la Figura 4.3. También se aprecia la mejora de la función de pérdida, que se sitúa durante todo el proceso con valores próximos a cero, excepto en la última epoch que experimenta un ligero ascenso, si bien con un valor de aproximadamente 0.5, que puede considerarse aceptable. En este caso, el tiempo de ejecución ha resultado ser de 26 minutos y 17 segundos, y por tanto más elevado que en el caso anterior, con ejecución sobre la misma CPU.

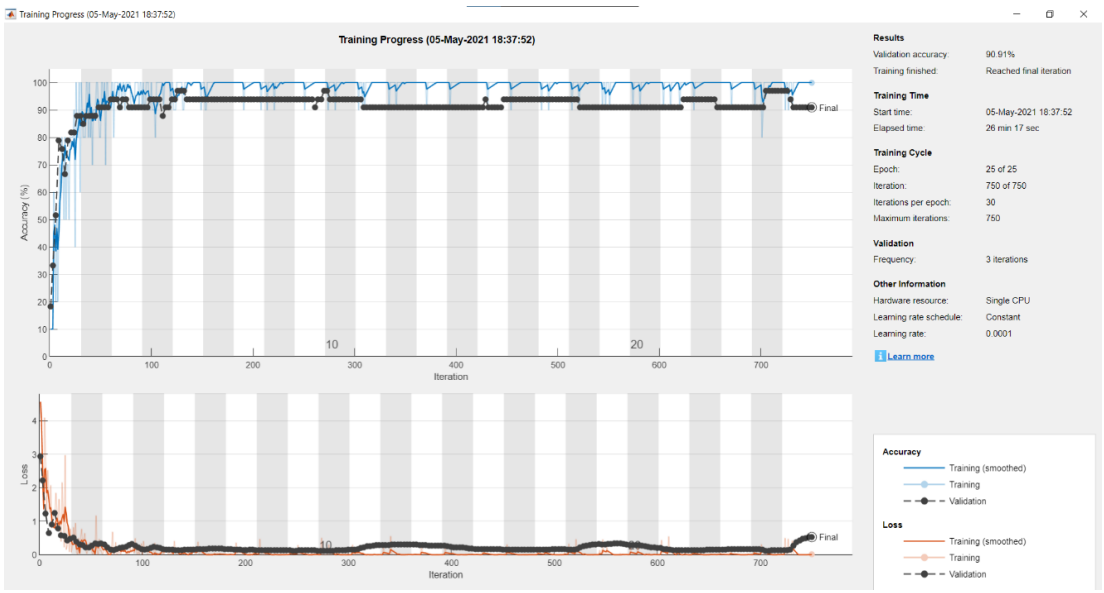


Figura 4.3 - Entrenamiento B-1 Alexnet

Se observan resultados idénticos al nivel de entrenamiento anterior en el caso de GoogLeNet, el porcentaje de acierto sigue estando en el 90,91%, por lo que el nivel de detección en los vídeos será muy similar a los resultados obtenidos anteriormente. Se ha experimentado un incremento en lo que respecta al tiempo de ejecución, que ha

resultado ser de 54 minutos y 55 segundos sobre la misma CPU. La función de pérdida se mantiene constantemente hasta el final con valores muy aceptables, situándose en las proximidades del valor óptimo cero.

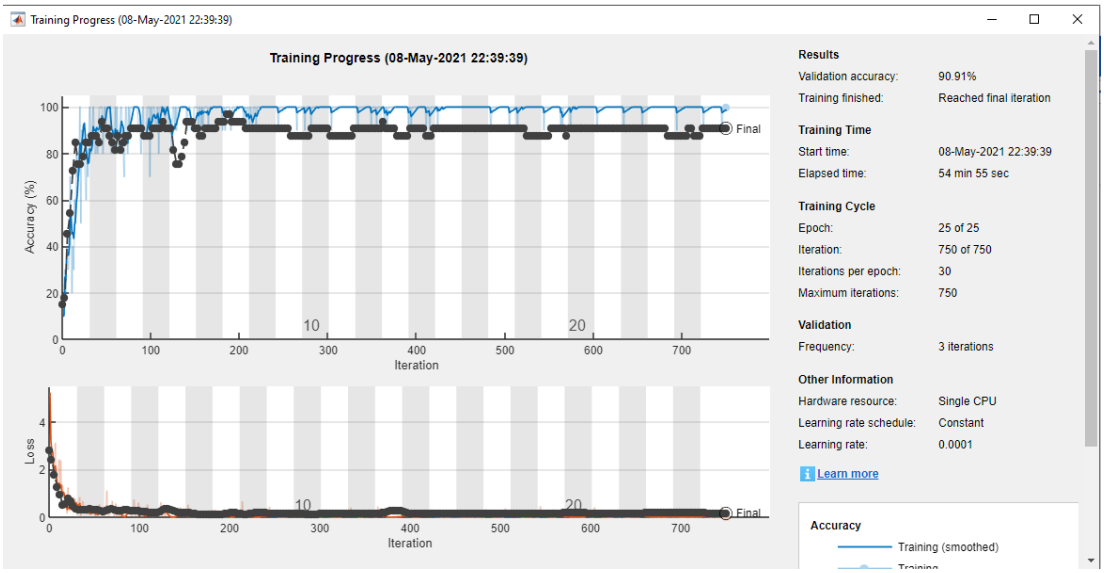


Figura 4.4 - Entrenamiento B-2 GoogLeNet

Si se contrastan los resultados de ambos entrenamientos con estos nuevos parámetros, la mejora del entrenamiento con AlexNet es ostensible, llegando al mismo nivel de acierto que GoogLeNet en este punto. Ambos entrenamientos alcanzan el 90,91 % y, por tanto, se espera que con ellos se consigan buenos resultados en la detección de vehículos.

C. Entrenamiento alto. (C-1 AlexNet y C-2 GoogLeNet)

Este último busca la mejor tasa de acierto sin importar el tiempo necesario para conseguirlo. Utilizará los valores más elevados de eficacia en cada parámetro buscando una solución de la mayor calidad posible.

- Nº Imágenes de entrenamiento: 64
- Nº Imágenes de validación: 25
- Epochs: 50
- Iteraciones: 1500
- Razón de aprendizaje: 0,00001

Para el entrenamiento más elevado, con la red Alexnet se ha dado una tasa de precisión del 93.94%, como muestra la Figura 4.5. Un ligero aumento respecto al entrenamiento B1 que se situaba en torno al 90%, aunque donde más se nota la diferencia es en el tiempo que ha tardado en realizarse el entrenamiento, cerca de 55 minutos respecto a los 26 minutos que duró el anterior. La función de pérdida se mantiene en valores muy aceptables a lo largo de todo el proceso.

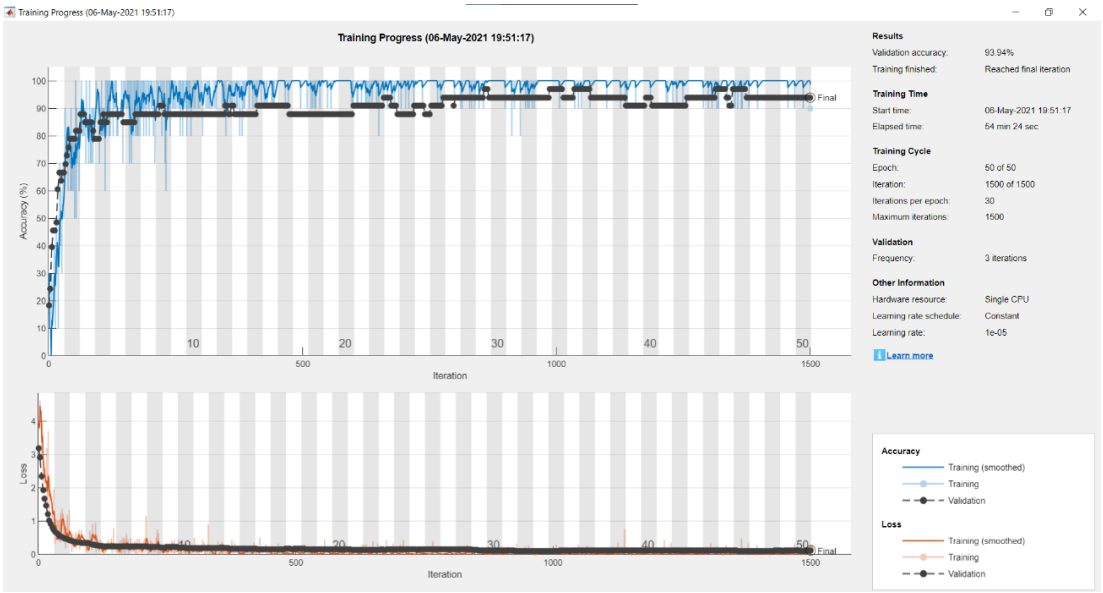


Figura 4.5 - Entrenamiento C-1 AlexNet

En este último entrenamiento con mejores valores en GoogLeNet, se ve una pequeña mejora sobre los anteriores, consiguiendo un porcentaje de acierto en la detección del 93.94 %. Este aumento de la efectividad del entrenamiento va en detrimento del coste en tiempo empleado, siendo éste un tercio mayor que en el caso anterior. De nuevo la función de pérdida consigue valores estables situándose en las proximidades del valor óptimo.

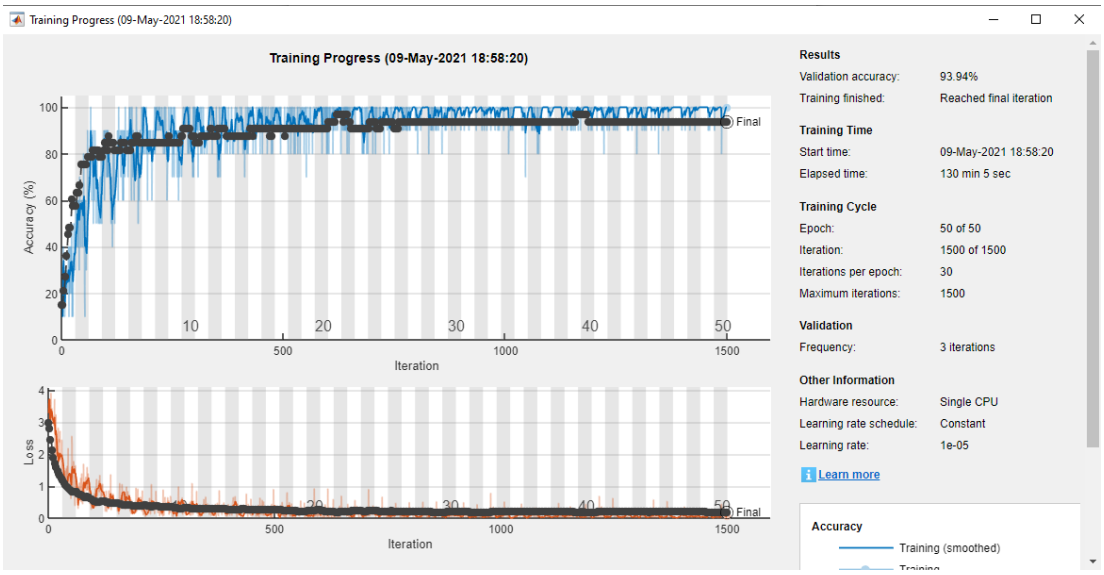


Figura 4.6 - Entrenamiento C-2 GoogLeNet

Se observa que, en el entrenamiento a más alto nivel propuesto, los valores de AlexNet y GoogLeNet, son exactamente iguales. Se aprecia un buen comportamiento de ambas redes con un 93.94 % de precisión en la validación.

De ello se deriva que AlexNet alcanza muy buenos resultados cuando se trata de un entrenamiento de cierto nivel, consiguiendo además mejores tiempos en ello que GoogLeNet. Por el contrario, realizando entrenamientos a bajo nivel, la red GoogLeNet demuestra ser más fiable. De hecho, en el primer nivel se ha constatado durante la fase de test que AlexNet no era capaz de detectar ningún tipo de vehículo, mientras que GoogLeNet ya se situaba en un porcentaje de acierto de más del 90%.

4.2 Detección de vehículos

Tal y como se especifica posteriormente, la detección de los vehículos con las redes propuestas AlexNet y GoogLeNet es más que aceptable, si bien es cierto que dependerá del nivel de entrenamiento de la propia red.

La captura de vehículos se realiza perfectamente en la mayoría de los casos, incluso con entrenamientos no demasiado costosos en tiempo, aunque como veremos posteriormente los conteos respecto del número de vehículos no sean tan exactos.

En la imagen mostrada en la figura 4.7 se puede ver una situación en la que aparecen varios vehículos y todos son detectados de forma óptima. En esta figura se muestra una captura de pantalla donde se muestran las distintas opciones del menú, así como el procesado del vídeo, con el progreso del mismo, y los resultados de la clasificación en cada imagen de la secuencia o *frame*.

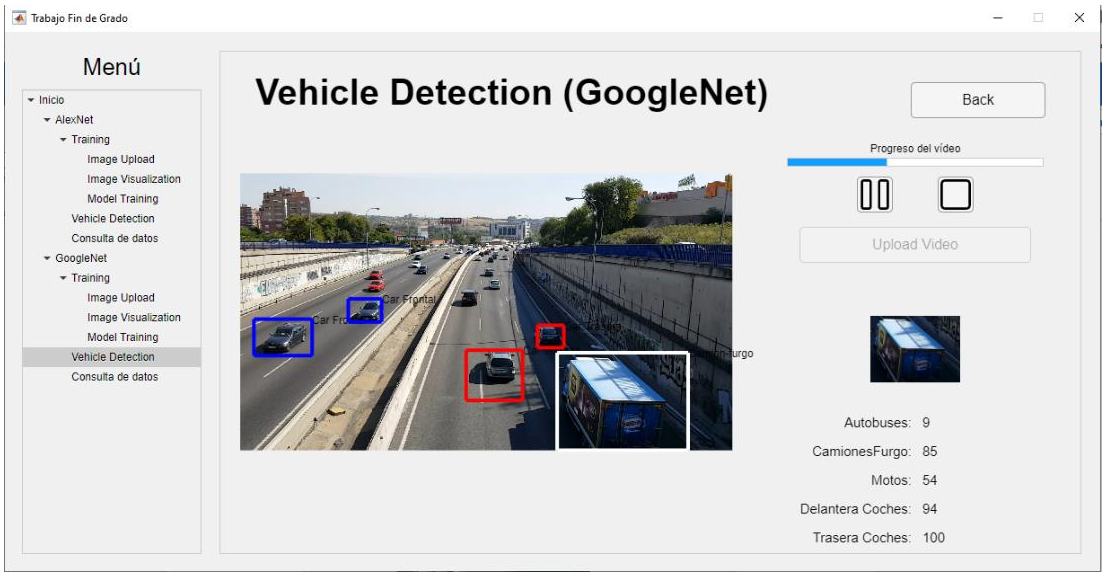


Figura 4.7 - Captura correcta de varios tipos de vehículos

4.3 Resultado de clasificación

De cara a tener una visión más precisa del conteo, se eligieron tres cámaras de muestra entre las disponibles y se procedió al conteo manual de los diferentes tipos de vehículos.

Así resultaron las tablas 1 a 3 que se muestran a continuación, en las cuales se pueden observar para cada tipo de entrenamiento realizado, los datos reales y los resultantes

de la ejecución del proceso de clasificación automática. En todos los casos se indica el tipo de entrenamiento realizado, así como el número de vehículos (coches, motocicletas, autobuses, camiones) reales existentes en cada vídeo y el número detectado.

- Cámara 1. (Moncloa)

Tipo	Coches		Motocicletas		Autobuses		Camiones	
	Real	Detectado	Real	Detectado	Real	Detectado	Real	Detectado
A1	110	0	4	0	10	0	7	0
B1		940		39		174		315
C1		905		24		137		354
A2		913		117		345		305
B2		864		61		247		240
C2		932		58		142		230

Tabla 1 – Datos de la cámara situada en Moncloa

- Cámara 2. (Ventas)

Tipo	Coches		Motocicletas		Autobuses		Camiones	
	Real	Detectado	Real	Detectado	Real	Detectado	Real	Detectado
A1	446	0	19	0	3	0	41	0
B1		1309		205		254		409
C1		1475		80		91		444
A2		1530		510		249		300
B2		1735		65		56		241
C2		1783		47		56		193

Tabla 2 – Datos de cámara situada en Ventas

- Cámara 3. (Villaverde)

Tipo	Coches		Motocicletas		Autobuses		Camiones	
	Real	Detectado	Real	Detectado	Real	Detectado	Real	Detectado
A1	192	0	0	0	0	0	19	0
B1		464		38		27		237
C1		490		23		6		198
A2		499		129		33		222
B2		495		8		17		188
C2		578		9		19		101

Tabla 3- Datos de cámara situada en Villaverde

En vista a los resultados que arrojan las tablas, se observa que en todos los casos el conteo de vehículos de cualquier tipo es muy superior al que realmente debería detectar. Esto se debe al diseño del método de conteo y diferenciación de vehículos, que se explicará en el siguiente punto.

En primer lugar, denotar que con el entrenamiento a más bajo nivel de AlexNet (A-1), no es capaz de detectar ningún tipo de vehículo, esto se puede deber a varios factores entre los que se podría destacar la falta de más imágenes de entrenamiento para que la red consiga una mayor capacidad de detección. Esta consideración está en línea con lo esperado, a la vista de los resultados del entrenamiento.

En relación al resto de los resultados, el problema reside en la captura de un mismo vehículo como distintos tipos a lo largo de su movimiento en el vídeo, una vez que este, durante algún *frame* es captado como un tipo de vehículo que no corresponde, aunque luego lo detecte correctamente ya sumará dos por lo que se dobla el número de elementos del vídeo y así sucesivamente.

En algunos casos esto se debe a la captura de vehículos en zonas muy alejadas de cada imagen con respecto a la zona más próxima a la cámara (parte inferior), lo que hace que el sistema todavía no pueda detectar correctamente su tipo, también existe el problema de que en ocasiones las secuencias de video contienen capturas de los vehículos de una forma distinta a las imágenes de entrenamiento utilizadas. Por ejemplo, se puede captar el vídeo tomando las imágenes de los vehículos de forma que se muestre totalmente su frente o su parte trasera, o en algunos casos, los vehículos aparecerán parcialmente con una vista lateral por lo que esto dificulta su diferenciación durante el proceso de clasificación.

Es importante también denotar la mejora de ambos algoritmos al mejorar el proceso de entrenamiento, sobre todo en la detección de motos, camiones y autobuses. Se puede observar que en la progresión del entrenamiento hacia un nivel superior los valores de estos tipos de vehículos se acercan cada vez más a los reales. Caso distinto es el de los coches donde en todos los entrenamientos se obtienen valores muy alejados de los reales, la dificultad para el algoritmo de detectar partes delanteras de coches y partes traseras es uno de los motivos que hace aumentar considerablemente su número, debido al problema explicado anteriormente, cuando un mismo vehículo es captado como una parte u otra a lo largo de su evolución en el vídeo.

En el siguiente punto veremos los errores más frecuentes observados por el equipo y que ayudan a comprender también la discrepancia de los valores mostrados en las mencionadas tablas.

4.4 Algoritmo de conteo y diferenciación de vehículos

El diseño de detección de vehículos implementado cuenta y suma los vehículos que se van detectando *frame* a *frame* del video en ejecución, lo que produce un nuevo fallo debido a la lógica del diseño, ya que el algoritmo suma el vehículo detectado en un *frame*

y el vehículo detectado en el *frame* siguiente, y como se observa en la figura 4.8 donde se muestran dos *frames* consecutivos, el procedimiento puede realizar la suma del mismo vehículo varias veces haciendo así que se obtengan datos de conteo incorrectos.

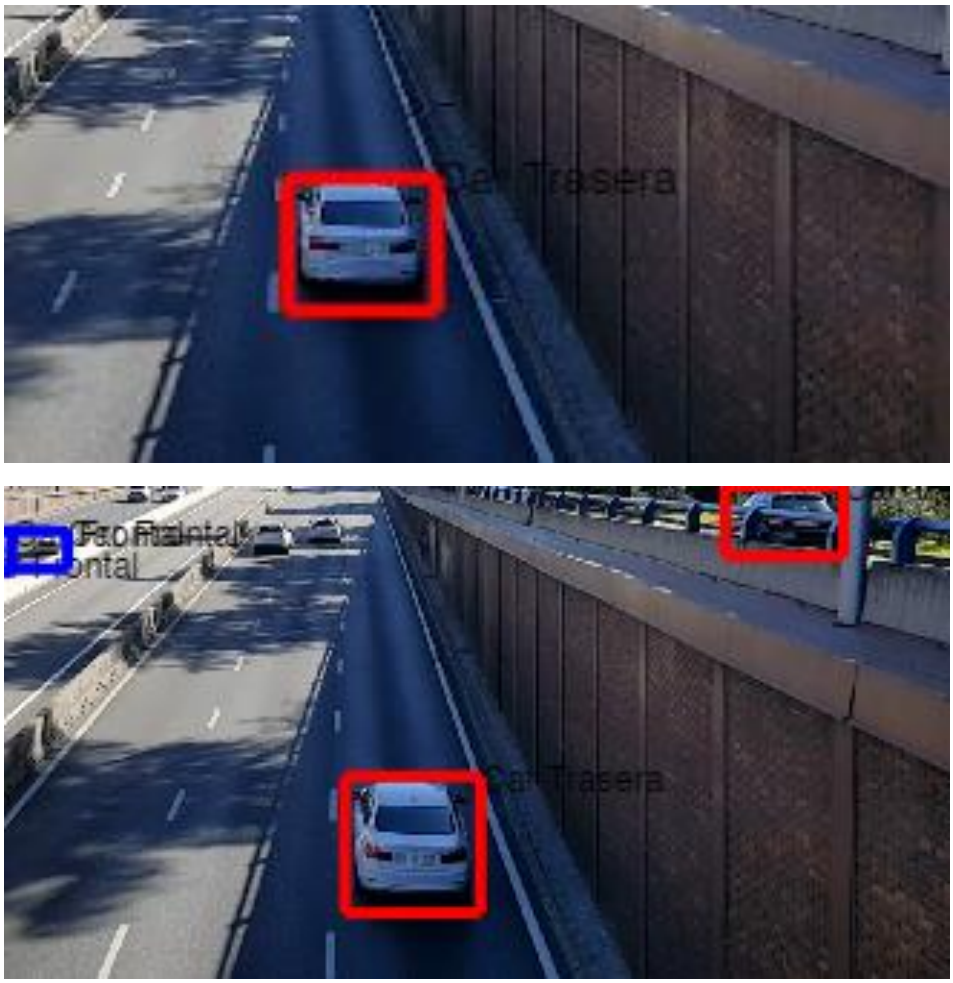


Figura 4.8 – La primera imagen muestra un frame del video en el que se detecta un vehículo, y la segunda imagen muestra el frame siguiente detectando el mismo vehículo y realizando de forma incorrecta el conteo.

Para solucionar este problema, se realiza un nuevo diseño que se encarga de detectar si los vehículos detectados en distintos *frames* del video, corresponden al mismo, para poder así realizar una corrección en el conteo de los vehículos y mantener la consistencia de la aplicación.

En el desarrollo del algoritmo se tienen en cuenta las coordenadas de los cuatro vértices del rectángulo que indica la posición del vehículo en cada *frame* del video que la red entrenada ha detectado. La figura 4.9 muestra un ejemplo ilustrativo del planteamiento propuesto.

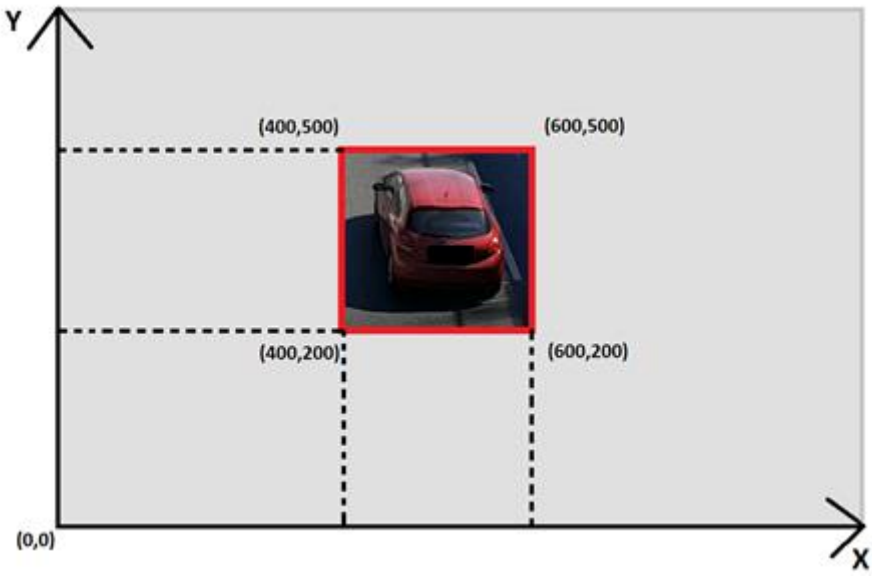


Figura 4.9 – Mapeo de un frame determinado del video donde se muestran las coordenadas del rectángulo del vehículo detectado.

En un principio se decidió seguir la estrategia general que se suele utilizar para detectar si dos cuadrados con las coordenadas de cada uno de sus vértices y el *frame* en el que ha aparecido pertenecen al mismo vehículo. Para realizar esto, se considera una línea horizontal imaginaria a lo largo de la proyección de la imagen del video que sirve para dictaminar que, si el rectángulo perteneciente a un vehículo es atravesado por esta línea horizontal, será considerado como un único vehículo. Hasta que este, en el vídeo, acabe de atravesar por completo la línea horizontal declarada. Para que se vea de forma más clara, en la figura 4.10 se muestra un ejemplo ilustrativo sobre cómo se detectan los vehículos en relación a la mencionada línea.

Figura 4.10 – Se muestra la línea de detección que determina si un rectángulo en distintos frames pertenece al mismo vehículo.

Pero esta forma de detectar vehículos ocasionó un nuevo problema tal y como se muestra en la figura 4.11. Cuando una red no está entrenada de forma correcta al 100% puede haber algún *frame* del video que no detecte el vehículo que sí ha sido detectado

en el *frame* anterior, provocando así que el algoritmo detecte este vehículo de forma duplicada. Si este error se produce varias veces mientras cruza la línea horizontal se puede multiplicar el número de detecciones del mismo vehículo.

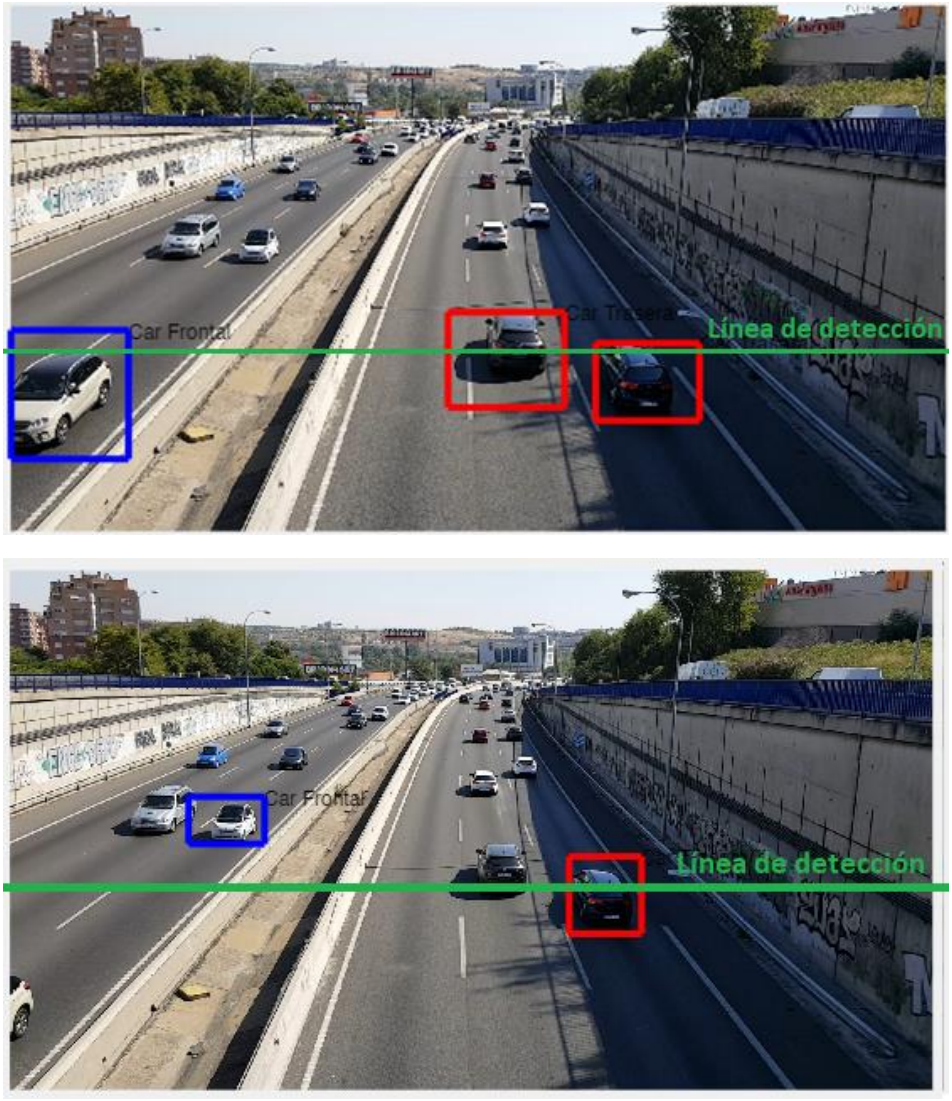


Figura 4.11 – En la primera imagen se puede observar cómo sí detecta el coche del centro, pero en la segunda no, por lo tanto, ese vehículo se contaría de forma duplicada.

Por ello se decidió que el algoritmo tomara únicamente las coordenadas de los rectángulos y gracias a ellas, calcular si un rectángulo se solapa con otro o no. Se dice que un rectángulo se solapa con otro si alguno de los vértices del primero se sitúa dentro de la región delimitada por el segundo o viceversa. En la figura 4.12 se muestran ocho ejemplos ilustrativos relativos al concepto de solapamiento con indicación en cada caso de si existe o no, donde el aspa roja indica que no y la marca verde lo contrario.

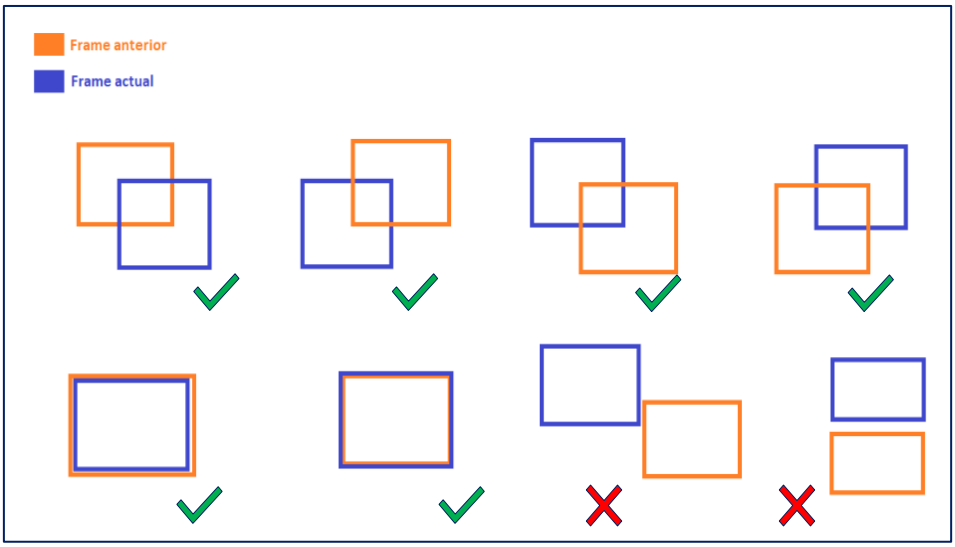


Figura 4.12 – Varios ejemplos de solapamiento positivos y negativos

Para corregir además el error de que en un *frame* del video no se detecte el vehículo y por tanto produzca fallo, se realiza un guardado de las coordenadas y del *frame* en el que se ha detectado, por tanto, si al detectar un nuevo vehículo, este se solapa con otro y la diferencia entre ambos *frames* es de 7, es decir existen 6 *frames* intermedios, o inferior, se determina que ambos rectángulos pertenecen al mismo vehículo. En las figuras 4.13, 4.14 y 4.15 se muestra un ejemplo de detección de vehículo con un error en un *frame* intermedio y cómo el procedimiento es capaz de detectarlo a pesar de este fallo.

Figura 4.13 – Muestra del frame número 15 extraído de un video de ejemplo en el que se observa cómo se ha detectado la parte trasera de un vehículo.

Figura 4.14 – Muestra el frame 17, dos frames posteriores de la imagen anterior donde debido a un fallo de la red, esta no es capaz de detectar el mismo vehículo.

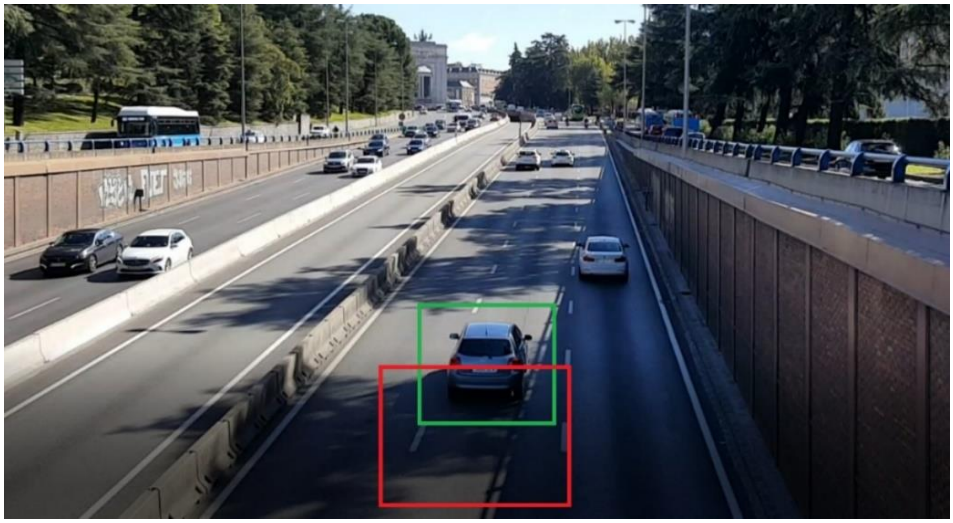


Figura 4.15 – Se muestra el frame 20 del vídeo donde en color verde se ha vuelto a detectar de forma correcta el vehículo, al estar este rectángulo (verde) contenido en el rectángulo del frame 15 (rojo) el vehículo es detectado como único.

Por último, el procedimiento se encarga de actualizar y borrar la pila de detecciones, ya que cuando el *frame* actual del video supera en 8 o más el *frame* guardado para un vehículo, este es borrado de la pila y añadido a la variable de número de vehículos detectados correspondiente. Explicando así que las variables que almacenan el número de vehículos detectados ya están confirmadas y no volverán a ser actualizados, en cambio los vehículos que se encuentran en la pila también guardan las coordenadas y el último *frame* en el que ha sido detectado para realizar una posible actualización.

4.5 Errores más frecuentes

- Captura de motos como muro, líneas o personas

Este error se produce cuando en el video que está en ejecución aparecen viandantes por las aceras, es común que se confundan con una moto al ser objetos de pequeño tamaño en movimiento. La figura 4.16 muestra el momento en el que se ha detectado a una persona como una moto.

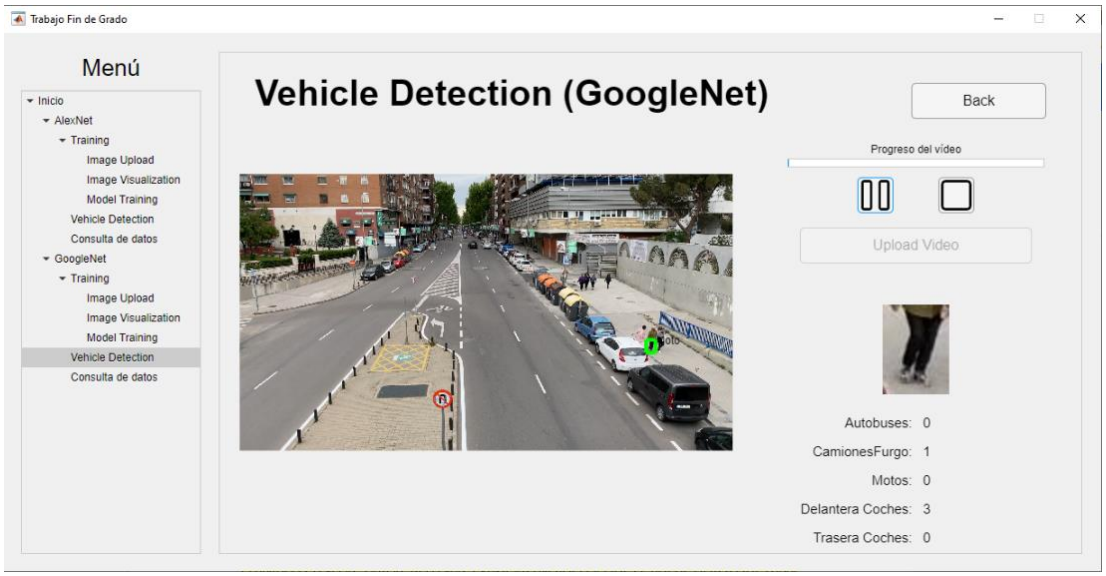


Figura 4.16 - Error por captura de motos como muro, líneas o personas

- Captura de dos o más vehículos como uno solo

En ocasiones, se detectan varios vehículos como uno solo en un *frame* concreto. Esto se debe a un fallo debido al entrenamiento de las redes.

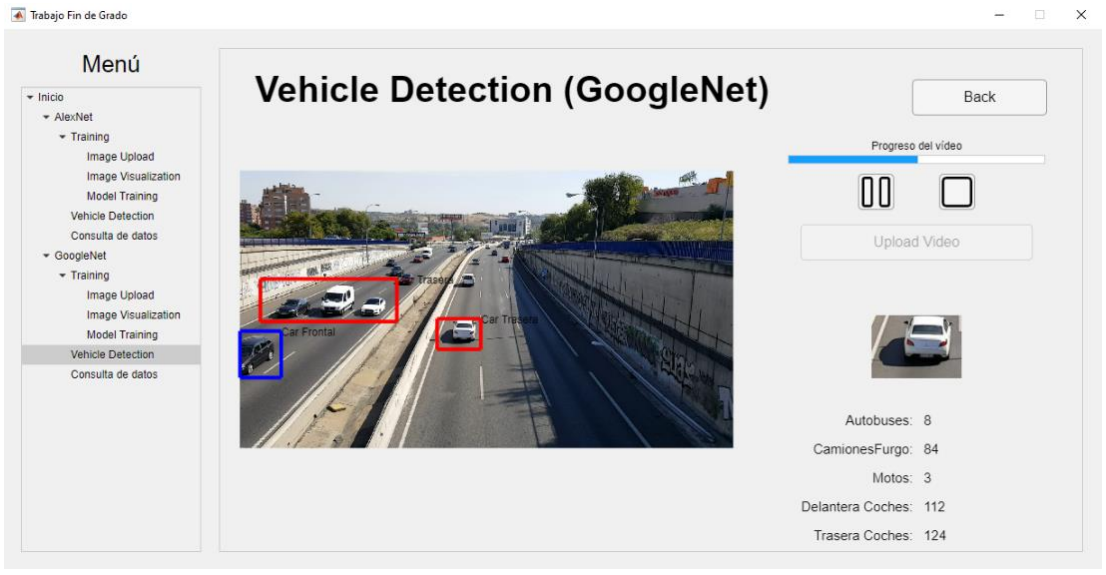


Figura 4.17 - Error por captura de varios vehículos como uno solo

- Captura de varios vehículos como un autobús o un camión

Se capturan varios vehículos y el sistema entiende que se trata uno de mayor volumen como un autobús o un camión. También afectará al conteo final en un aumento de los vehículos detectados y además en una detección de tipo de vehículo errónea. En la Figura 4.18 se detecta una furgoneta y un coche como si de un camión se tratara.

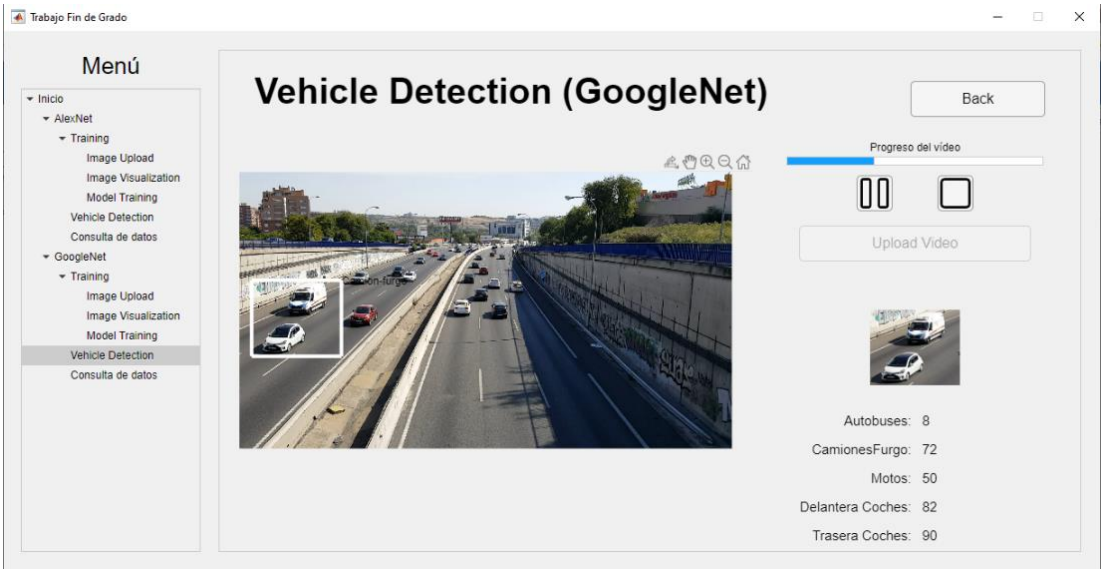


Figura 4.18 - Error por captura de varios vehículos como un autobús o un camión

- Cambio en la captura de un coche como parte frontal a trasera o viceversa

Es el que más desvirtúa los resultados reales, puesto que es uno de los más comunes. Se trata de la detección de un vehículo en su parte trasera que en otro punto del vídeo se recoge cómo un vehículo en su parte delantera. En las Figuras 4.19 y 4.20 se puede observar cómo el vehículo rojo de la parte superior izquierda es detectado como una parte trasera en principio, pero en un *frame* posterior ese mismo vehículo se detecta como una parte delantera, que sería la imagen real.

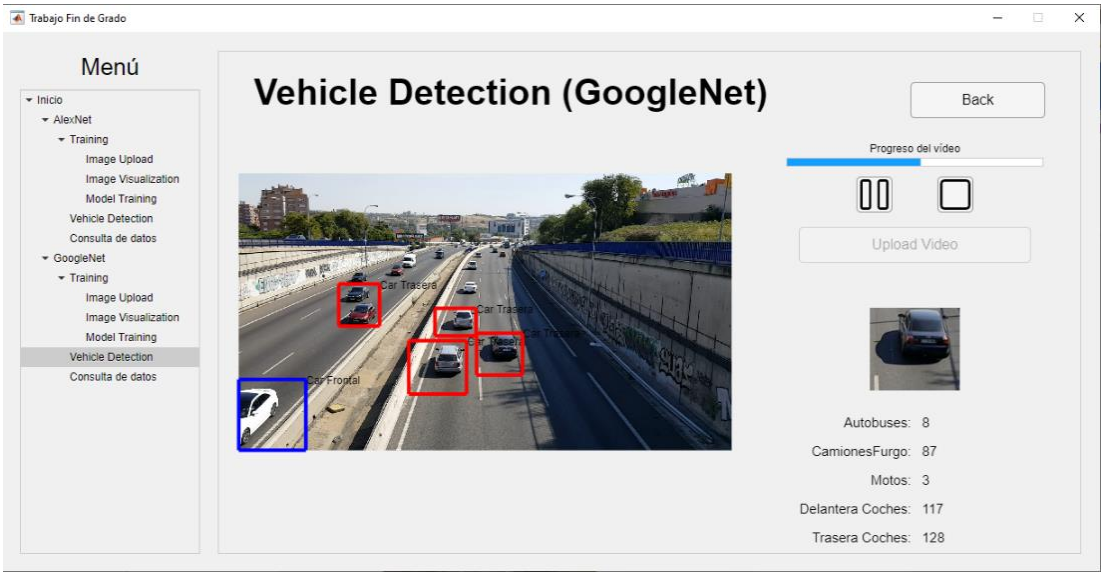


Figura 4.19 - Error por captura de coche como frontal y posteriormente como trasera o viceversa (1)

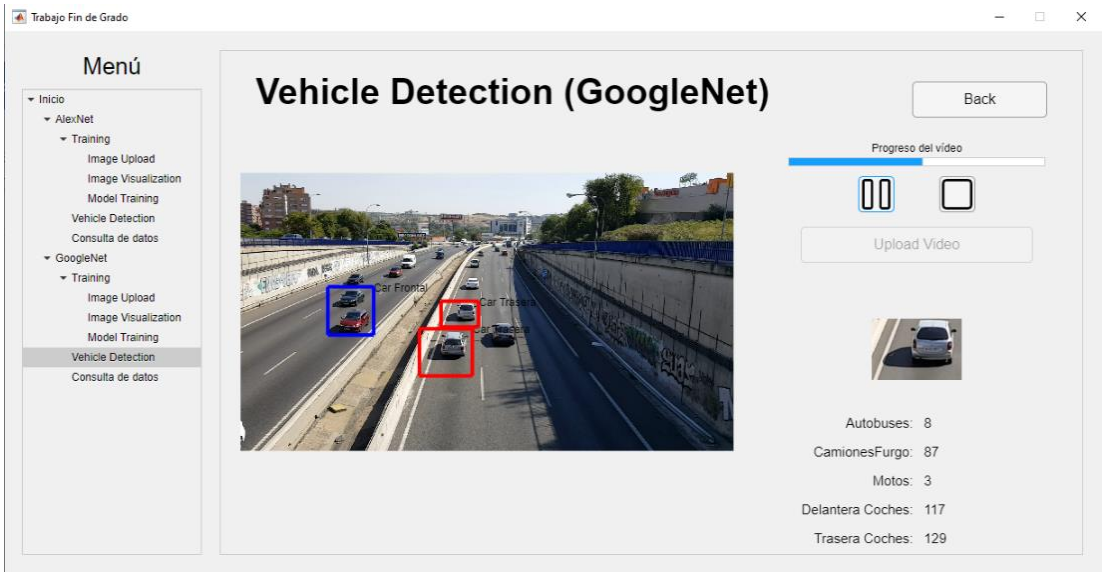


Figura 4.20 - Error por captura de coche como frontal y posteriormente como trasera o viceversa (2)

- Captura de elementos no pertenecientes a la vía principal

Debido a una toma de vídeo que no se ajusta a los carriles de circulación con los que se quieren trabajar, en ocasiones se recogen elementos que no pertenecen a la escena objeto de detección, esto es, los vehículos en movimiento. Existen casos en los que aparecen elementos en movimiento por las aceras como pueden ser personas y otros en los que se recoge movimiento en carriles contiguos, como pueden ser incorporaciones u otro tipo de vías. En la figura 4.21, se muestran elementos que pertenecen a una vía superior que tiene además una valla por lo que no se pueden detectar correctamente los vehículos dando lugar a diversos errores.

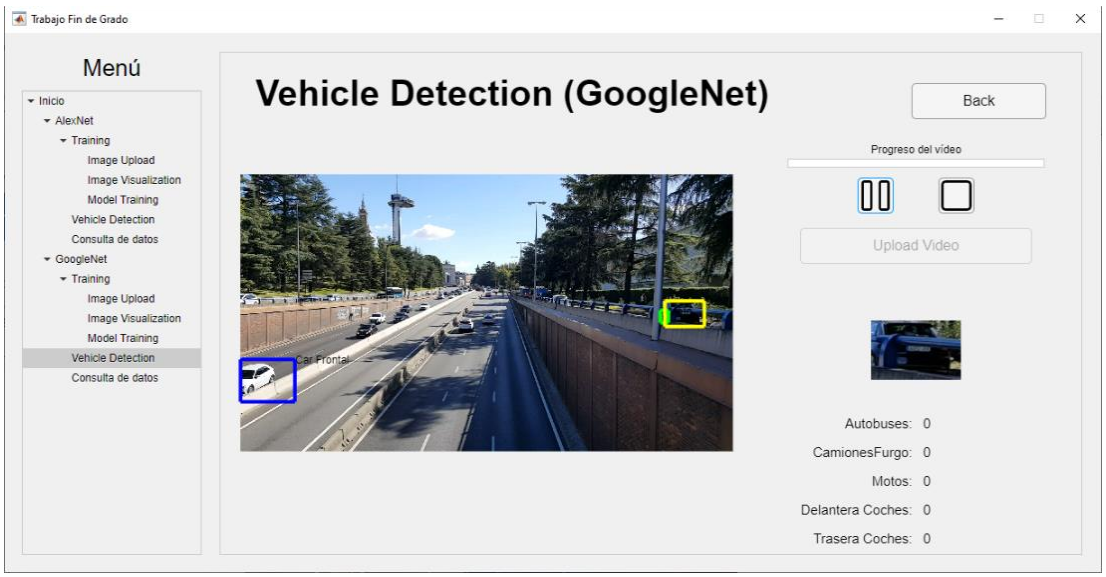


Figura 4.21 - Error por captura de elementos no pertenecientes a la vía principal

- Vehículos cortados por bordes en el vídeo

Al captar un elemento cortado por los bordes del video, el sistema en ocasiones detecta elementos que no se corresponden con los reales, siendo el caso más común el de capturar una moto cuando apenas aparece una pequeña parte de cualquier tipo de vehículo, como sucede en la Figura 4.22, de forma que en la esquina inferior izquierda se detecta una moto cuando en realidad es una parte de un coche.

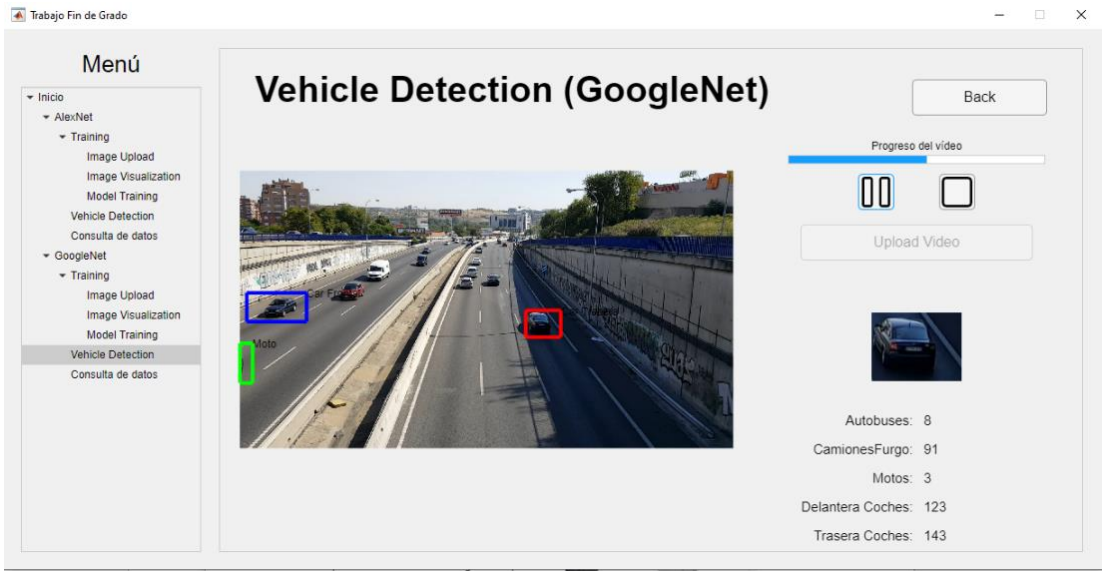


Figura 4.22 - Vehículos cortados por el borde del video

- Captura de sombra como un vehículo

La sombra de los vehículos es otro elemento que puede confundir al algoritmo, ya que también se encuentra en movimiento solidario con el vehículo que la genera y por ello es detectada. En el caso de la Figura 4.23 una sombra es identificada erróneamente. Sucede que lo detecta como cualquier tipo de vehículo, o incluso hace que el conjunto detectado pueda parecer mayor y cambiar el tipo de vehículo detectado.

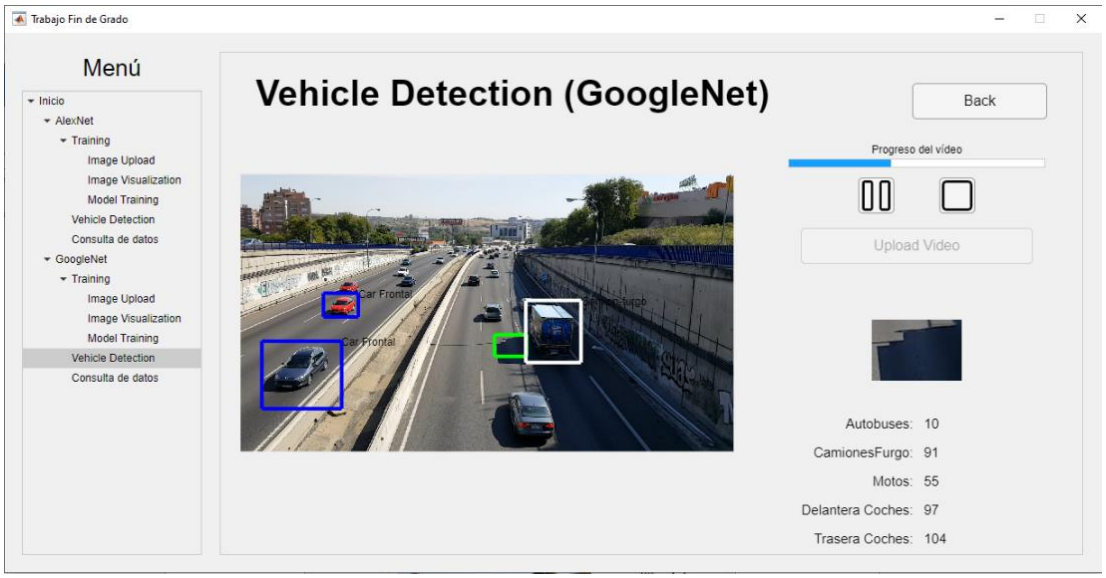


Figura 4.23 – Captura de sombra como vehículo

- Detección errónea de elementos

Puntualmente, se detectan elementos que no tienen nada que ver con el real, como se puede observar en la Figura 4.24. Se detecta un coche como camión, y un muro como coche. Este tipo de error sucede de forma más habitual, como es lógico, en los entrenamientos de nivel más bajo.

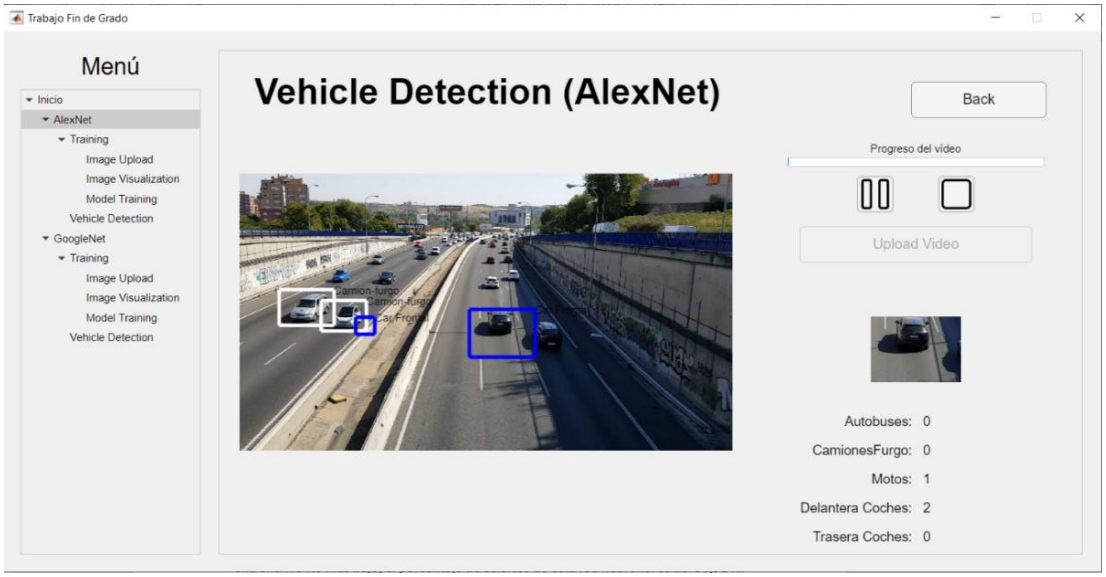


Figura 4.24 - Detección errónea de elementos

- Duplicación de vehículos debido al algoritmo de conteo.

Existen ciertos casos en los que el algoritmo de detección no es capaz de reconocer que dos rectángulos detectados en distintos puntos del video, pertenecen al mismo vehículo. Esto provoca que se dupliquen coches y en ciertos puntos del video, el numero de vehículos detectados sea superior al número de vehículos reales. Como se muestra en la figura 4.25 que se ve que el número de vehículos detectados en la parte inferior derecha es demasiado grande para el progreso del video actual.

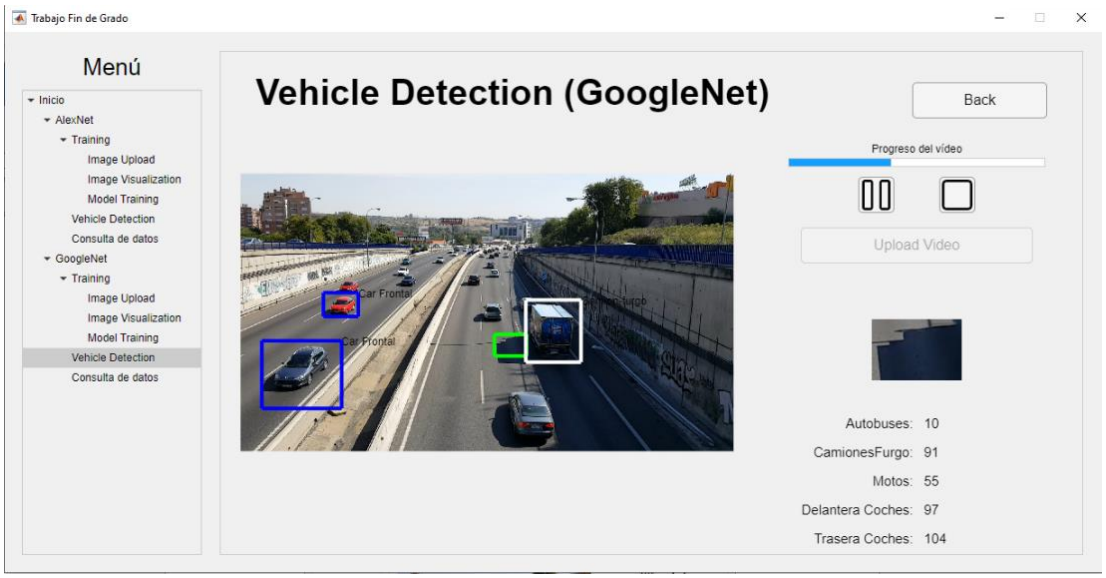


Figura 4.25 - Detección errónea de elementos

4.6 Integración de los datos

Una vez realizada la detección se redirige al usuario a una interfaz, como se muestra en la figura 4.26, en la que es posible seleccionar a que cámara pertenecen los datos, así como la hora y el día en que se produjo la detección. Una vez seleccionados los parámetros más convenientes, si selecciona el botón "Enviar", los datos se mandarán a la plataforma de ThingSpeak[6], procediendo a la publicación de un tweet en la cuenta @controlMadrid, como se muestra en la figura 4.27.

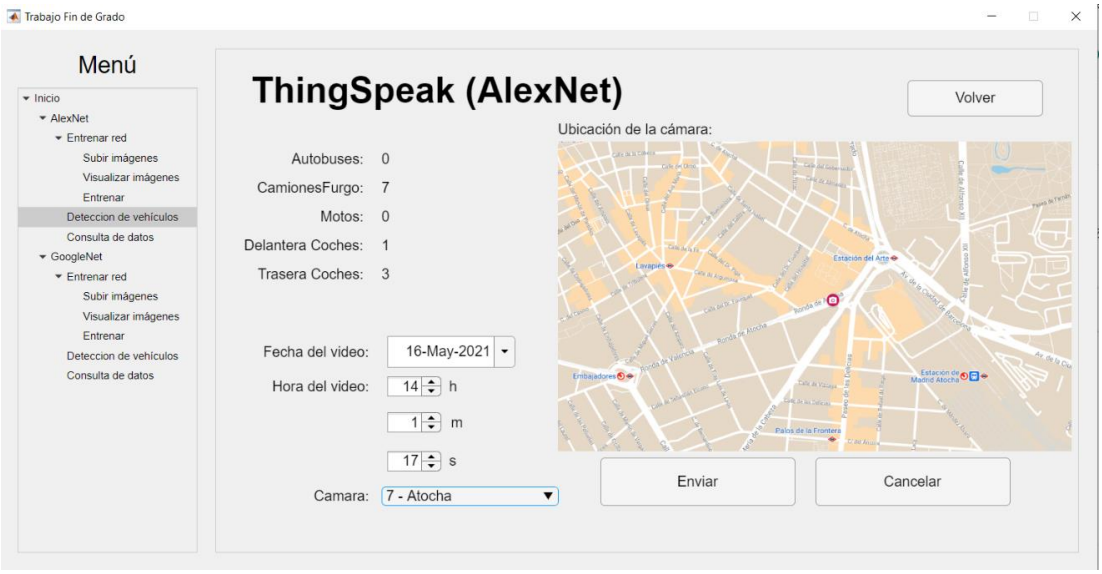


Figura 4.26 – Selección de parámetros para envío

Figura 4.27 – Tweet generado con los datos de la detección

Si se desea probar la aplicación, en el Anexo 4[A4] se muestra una guía de instalación.

5. Conclusiones y trabajo futuro

5.1 Conclusiones

En el presente trabajo se ha desarrollado una aplicación para la detección de distintos tipos de vehículos a partir de videos previamente grabados, utilizando técnicas de

Aprendizaje Profundo mediante el entrenamiento y posterior ejecución de Redes Neuronales Convolucionales.

En primer lugar, se busca, mediante la realización de Mockups, disponibles en el Anexo 2[A2], diseñar una interfaz amigable para el usuario de forma que cualquier persona que no haya tenido excesivo contacto con el mundo de las nuevas tecnologías pueda interactuar con la aplicación. Se busca también un menú que permita un acceso rápido a cualquier funcionalidad de la misma desde cualquier punto, por ello se utiliza un menú lateral que facilita este tránsito.

El programa trabaja el concepto de aprendizaje profundo, utilizando para ello dos redes neuronales convolucionales como son AlexNet y GoogLeNet, ambas redes son entrenadas bajo un conjunto de imágenes y una serie de parámetros seleccionados convenientemente, de forma que se busca la mayor tasa de acierto posible. Se ha observado, como ya se ha expuesto a lo largo del trabajo, que la red AlexNet tiene un tiempo de entrenamiento menor que el caso de GoogLeNet, funcionando mejor con unos parámetros de entrenamiento más exigentes, mientras que GoogLeNet, pese a conllevar un mayor coste en tiempo ofrece mejores resultados, ello si se trata de entrenamientos con unos parámetros de entrenamiento menores.

El entrenamiento de las redes se basa en el aprendizaje por parte de estas de una serie de vehículos seleccionados, que en el caso de este trabajo serán camiones, motos, autobuses, coches en su parte delantera y coches en su parte trasera. Aunque estos últimos a la hora de almacenar los datos derivados de su detección se almacenen como coches en general, sumando ambos valores.

La detección y diferenciación de vehículos se basan en la aplicación de métodos basados en el flujo óptico y la segmentación de regiones, esto nos proporciona la opción de captar las imágenes en movimiento dentro de los vídeos de tráfico utilizados para tal propósito. Tras la detección de dichos vehículos se ha diseñado un método de diferenciación de estos para identificar el número de los distintos tipos de vehículos que transcurren por dicha vía.

Para la realización de pruebas en la aplicación, se han capturado vídeos en distintos puntos de la Comunidad de Madrid. De esta forma, se cuenta con elementos suficientes para observar el nivel de acierto tanto en la detección de vehículos como en el algoritmo de conteo. Se observa que la detección de vehículos es suficientemente aceptable, por regla general, para ambas redes; sin embargo, la diferenciación entre vehículos y el conteo de estos han generado problemas que se deberán mejorar en futuras ampliaciones.

Una vez se obtienen los datos relativos a cada tipo de vehículo se pueden almacenar en una base de datos relacional de forma que se pueda acceder a los resultados mediante la realización de consultas. Para ello se ofrece la posibilidad al usuario de elegir el tipo de vehículos, la cantidad de estos, la fecha y cámara donde fueron tomados,

mostrándose en una tabla los resultados obtenidos de la misma. Se valoró de igual forma el tratamiento de los datos bajo el paradigma IoT, teniendo la posibilidad de almacenarlos en ThingSpeak[6] y exponerlos mediante Twitter. Si bien se consiguió su correcto funcionamiento, restricciones internas del propio ThingSpeak[6], hacen que se descartara dicha opción, dejándola implementada para el caso futuro en que se recogieran datos de cámaras de forma ininterrumpida.
El usuario también tiene acceso en todo momento al conjunto de imágenes que forman el dataSet, además de poder incluir en él nuevas imágenes.

Por último, se realizó un completo banco de pruebas tanto para la aplicación completa como para cada uno de sus módulos de forma que se valida su correcto funcionamiento.

5.2 Trabajo futuro

De cara a la continuación del trabajo en el futuro, se han identificado los siguientes aspectos.

Mejoras de la aplicación:

- Utilización de un *dataSet* con más imágenes para la mejora de los entrenamientos. Actualmente la aplicación cuenta con más de 300 imágenes disponibles para el entrenamiento de las CNN, ampliando dicho número se podría mejorar la eficacia de dichos entrenamientos.
- Ampliación del número de entrenamientos realizados de cara a encontrar la mejor ratio precisión/tiempo.
- Ubicación de cámaras fijas conectadas a la aplicación para el control del tráfico. En lugar de utilizar videos pregrabados, tener acceso a cámaras que muestren el tráfico en directo.
- Posibilidad de incorporar más elementos a las consultas a la base de datos ya existentes, buscando proporcionar la mayor cantidad de información posible al usuario.
- Mejora del algoritmo de conteo para que no duplique vehículos, y realizar una mejora en el código para que el algoritmo corrija este fallo detectado durante el desarrollo. Además es necesario mejorar el rendimiento para que el procesamiento de los vídeos de deteccion se realice mas rápido.

Nuevas funcionalidades:

- Captura de vídeos al mismo nivel de posicionamiento de la cámara para homogeneizar las capturas

  Una forma de mejorar la detección de vehículos será definiendo los valores óptimos para la captura de los vídeos de tráfico. De esta forma se puede indicar el nivel de

altura sobre la carretera que deben tener estos vídeos, así como su orientación de forma que los vehículos sean detectados siempre de frente.

Se ha detectado, que, debido a la orientación de la cámara, en ocasiones se recogen vehículos parcialmente de perfil y no totalmente de frente, lo que complica la clasificación del vehículo detectado. De la misma forma, vídeos tomados desde una altura diferente pueden tomar imágenes distintas del mismo vehículo en dos cámaras, por lo que los datos del mismo pueden verse afectados.

- Control de velocidad de vehículos

  Gracias al tratamiento *frame* a *frame* del vídeo en el que se sabe exactamente el tiempo que transcurre entre ellos y a la distancia en la que cada vehículo avanza en ese intervalo de tiempo, la aplicación puede ser capaz de detectar la velocidad a la que circula dicho vehículo en ese instante de tiempo.

  También se dispone de la localización de las carreteras o calles en las que están instaladas las cámaras, por lo que si a esa ubicación se le añadiera un atributo que determinara la velocidad máxima de circulación en la misma, se podrían realizar capturas de los vehículos que la sobrepasen y tratar dicha imagen para obtener los datos del vehículo.

- Detección de infracciones en stop y semáforos

  Al igual que se realizan entrenamientos para la detección de vehículos, líneas de la calzada y muros; se podrían realizar para detectar semáforos o señales de tráfico.

  Una vez incluidos dichos elementos en la CNN, ésta sería capaz de detectar si un vehículo en un punto de stop o semáforo no ha efectuado la parada obligatoria.

- Detección de atascos

  Utilizando de la misma forma el control de velocidad de vehículos, se puede detectar cuando los vehículos de una vía están circulando a una velocidad anormalmente lenta, lo que se puede deducir como un atasco en dicha vía.

- Control de accidentes de tráfico o averías de vehículos

  Al detectar un solo vehículo que no avanza en una carretera se pueden detectar posibles accidentes de tráfico o averías de vehículos en circulación.

  Además, al tener detección de las líneas de la carretera también se pueden detectar posibles accidentes de vehículos por salidas de la vía.

- Aplicación móvil

  Gracias a la implementación en forma de API para la obtención de los datos obtenidos por la aplicación se puede desarrollar una pequeña aplicación que realizando consulta a dicha API permita realizar predicciones de tráfico y servir esta información a los usuarios de la aplicación desarrollada.

- Página web de consulta de datos

  Al igual que con el desarrollo de la aplicación móvil, se puede crear una página web que permita consultar todos los datos obtenidos, que permita crear gráficos para representar de forma visual y que realice predicciones de forma precisa con los datos.

6. Conclusions and future work

6.1 Conclusions

In the present work an application has been developed for the detection of different types of vehicles from previously recorded videos, using Deep Learning techniques through the training and subsequent execution of Convolutional Neural Networks.

First, using the Mockups, available in Anexo 2[A2], we seek to design a user-friendly interface so that any person who has not had much contact with the world of new technologies can interact with the application. A menu is also attempted to allow quick access to any functionality of the application from any point, so a lateral menu is used to facilitate this transit.

The program works with the concept of Deep Learning, using two convolutional neural networks such as AlexNet and GoogLeNet, both networks are trained under a set of images and a series of conveniently selected parameters, so that the highest possible hit rate is sought. It has been observed, as already exposed throughout the work, that the AlexNet network has a shorter training time than GoogLeNet, working better with more demanding training parameters, while GoogLeNet, despite entailing a higher cost in time, offers better results, this if it is training with lower training parameters.

The training of the networks is based on the learning by these networks on a series of selected vehicles, which in the case of this work will be trucks, motorcycles, buses and front/back cars. Although the latter, when storing the data derived from their detection, are stored as cars in general, adding up both values.

The detection and differentiation of vehicles are based on the application of methods based on optical flow and region segmentation; this gives us the option of capturing moving images within the traffic videos used for this purpose. After the detection of these vehicles, a method of differentiation of these has been designed to identify the number of different types of vehicles passing through the road.

To test the application, different videos have been captured in different points of the Community of Madrid. In this way, there are enough elements to observe the level of accuracy both in the detection of vehicles and in the counting method. It is observed that the detection of vehicles is sufficiently acceptable, as a general rule, for both networks; however, the differentiation between vehicles and the counting of these have generated problems that should be improved in future extensions.

Once the data for each type of vehicle is obtained, it can be stored in a relational database so that the results can be accessed by performing queries. For this purpose, the user can choose the type of vehicles, the number of vehicles, the date and camera where they were taken, and the results obtained are displayed in a table. The treatment of the data under the IoT paradigm was also evaluated, with the possibility of storing them in ThingSpeak[6] and exposing them through Twitter. Although its correct operation was achieved, internal restrictions of ThingSpeak[6] itself, made this option to be discarded, leaving it implemented for the future case in which camera data would be collected uninterruptedly.

The user also always has access to the set of images that make up the dataSet, in addition to being able to include new images in it.

Finally, a complete test bench was carried out both for the complete application and for each of its modules to validate its correct operation.

6.2 Future work

Application improvements:

- Use of dataSet with more images to improve training. Currently the application contains more than 300 images available for the CNN trainings, expanding this number, the system could improve the effectiveness of these trainings.
- Modification of the vehicle differentiation algorithm to improve vehicle counting, adding the update of the counter only when the vehicle is in a specific horizontal strip, avoiding multiple counts.
- Expansion of the number of trainings performed to find the best accuracy/time ratio.
- Location of fixed cameras connected to the application for traffic control. Instead of using pre-recorded videos, have access to cameras capturing live traffic.
- Incorporate more elements to the existing queries, seeking to provide the user with as much information as possible.
- Improvement of the counting algorithm so that it does not duplicate vehicles, and make an improvement in the code so that the algorithm corrects this fault detected during development. In addition to improving performance so that the processing of detection videos is done faster.

New functionalities:

- Capture of videos at the same level in order to homogenize captures at identical levels.

  One way to improve vehicle detection is to define the optimal values for capturing traffic videos. In this way, it is possible to indicate the level of height above the road

that these videos should have, as well as their orientation so that vehicles are always detected from the front.

It has been detected that, due to the orientation of the camera, sometimes vehicles are captured partially in profile and not completely from the front, which complicates the classification of the detected vehicle. In the same way, videos taken from a different height can take different images of the same vehicle in two cameras, so the vehicle data can be affected.

- Vehicle speed control

  Thanks to the *frame*-by-*frame* processing of the video in which the exact time elapsed between them and the distance in which each vehicle advances in that time interval is known, the application can be able to detect the speed at which the vehicle is traveling at that instant of time.

  The location of the roads or streets where the cameras are installed is also available, so that if an attribute is added to this location that determines the maximum speed of circulation, it would be possible to capture the vehicles that exceed it and process this image to obtain the vehicle data.

- Detection of stop sign and traffic light violations.

  In the same way that CNNs are trained to detect vehicles, roadway lines and walls, they could be trained to detect traffic lights or traffic signs.

  Once these elements are included, it would be able to detect if a vehicle at a stop sign or traffic light has not made the mandatory stop.

- Traffic jam detection

  Using vehicle speed control in the same way, it is possible to detect when vehicles on a road are traveling at an unusually slow speed, which can be deduced as a traffic jam on that road.

- Monitoring of traffic accidents or vehicle breakdowns

  By detecting a single vehicle that is not moving on a road, it is possible to detect possible traffic accidents or breakdowns of vehicles in circulation.

  In addition, by detecting the lines of the road, it is also possible to detect possible accidents of vehicles leaving the road.

Bibliografía

1. Mirame.net (2021): Inteligence Solutions. Disponible on-line: https://www.mirame.net (accedido abril 2021).
2. Mall Parking Manager: propiedad de Mirame.net. Disponible on-line: https://www.mirame.net/mall-parking-manager/
3. Bosonit.com (2021): Disponible on-line: https://www.bosonit.com (Accedido Abril 2021)
4. Madrid Mobility 360 (2021). Disponible on-line: https://www.mobility360.app/ (accedido abril 2020).
5. Corral-Descargue, A.L., Delgado-Yepes, G., Goicoechea-Enrique, V., Guerrero-Moñús, M. (2019). Smart cities: Aprendizaje profundo con imágenes. Trabajo Fin de Grado. Facultad de Informática. Universidad Complutense Madrid. Disponible on-line: https://eprints.ucm.es/id/eprint/61779/ (Accedido abril 2021).
6. ThingSpeak (2021). ThingSpeak for IoT Projects. Disponible on-line: https://thingspeak.com/ (accedido abril 2021).
7. Matlab (2021). Matlab para Inteligencia Artificial. Disponible on-line: https://es.mathworks.com/ (accedido abril 2021).
8. Bishop, C.M. (2006). Pattern Recognition and Machine Learning, Springer, NY, USA.
9. BVLC GoogLeNet Model (2021). Disponible on-line: https://github.com/BVLC/caffe/tree/master/models/bvlc_GoogLeNet (Accedido marzo 2021).
10. Farneback, G. (2003). Two-Frame Motion Estimation Based on Polynomial Expansion. In Proceedings of the 13th Scandinavian Conference on Image Analysis, 363 - 370. Halmstad, Sweden: SCIA.
11. Haralick, R.M., Shapiro, L.G. (1992). Computer and Robot Vision. Addison-Wesley, Boston.
12. Imagenet (2021). Disponible on-line: http://www.image-net.org (Accedido marzo 2021).
13. ImageNet LSVRC (2012). Large Scale Visual Recognition Challenge 2012 (ILSVRC2012). Disponible on-line: http://www.image-net.org/challenges/LSVRC/2012/ (Accedido marzo 2021).
14. Krizhevsky, A., Sutskever, I., Hinton, G.E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Proc. 25th Int. Conf. on Neural Information Processing Systems (NIPS'12), vol. 1, pp. 1097-1105.
15. Pajares, G., Cruz, J.M. (2007). Visión por Computador: Imágenes digitales y aplicaciones. RA-MA, Madrid.

16. Pajares, G., Herrera, P.J, Besada, E. (2021). Aprendizaje Profundo. RC-Libros, Madrid.

17. Ruder, S. (2017). An overview of gradient descent optimization algorithms. arXiv:1609.04747v2 [cs.LG].

18. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2014). Going Deeper with Convolutions. Computing Research Repository. arXiv:1409.4842 [cs.CV].

19. Agudelo, N., Tano, G., Vargas, C.A. () Historia de la automatización. Universidad ECCI.

20. Tavizon-Salazar, A., Guajardo, T., Laines, C. (2016) IOT, el internet de las cosas y la innovación de sus aplicaciones. Universidad Autónoma de Nuevo León.

21. Going deeper with convolutions, Google (2004) https://arxiv.org/pdf/1409.4842v1.pdf

22. Deep Learning: GoogLeNet Explained (Dec 2020) Richmond Alake, Toward Data Science: https://towardsdatascience.com/deep-learning-GoogLeNet-explained-de8861c82765

23. A Review of Popular Deep Learning Architectures: AlexNet, VGG16, and GoogLeNet (2020) https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-GoogLeNet/

24. Understanding GoogLeNet Model – CNN Architecture (03 May, 2020) https://www.geeksforgeeks.org/understanding-GoogLeNet-model-cnn-architecture/

25. Plan de alojamiento para páginas web y almacenamiento de APIs con sistema de almacenamiento relacional https://www.dondominio.com/products/hosting/

26. El IDE rápido e inteligente de PHP desarrollado por JetBrains e IntelliJ Idea https://www.jetbrains.com/es-es/phpstorm/

27. The Collaboration Platform for API Development https://www.postman.com/

28. ¿Qué son los patrones de diseño de software? https://profile.es/blog/patrones-de-diseno-de-software/

29. Benefits of patterns, Design Patterns and Classification and Catalog of patterns https://refactoring.guru/design-patterns

30. ThingSpeak Support Toolbox file Exchange functions for connectivity from MATLAB to ThingSpeak (2019) https://es.mathworks.com/matlabcentral/fileexchange/52244-thingspeak-support-toolbox

31. Mapping Toolbox. (2019) (https://es.mathworks.com/products/mapping.html)

32. Statistic and Machine Learning Toolbox (2019) (https://es.mathworks.com/products/statistics.html)
33. Symbolic Math Toolbox https://es.mathworks.com/products/symbolic.html
34. Simulink, MATLAB product https://es.mathworks.com/products/simulink.html
35. MATLAB Compiler, ejecutables independientes de MATLAB https://es.mathworks.com/products/compiler.html
36. MATLAB Compiler SDK, componentes software para MATLAB https://es.mathworks.com/products/matlab-compiler-sdk.html
37. MATLAB Support for MinGW-w64 C/C++ Compiler (Marzo - 2021) https://es.mathworks.com/matlabcentral/fileexchange/52848-matlab-support-for-mingw-w64-c-c-compiler
38. Image Processing Toolbox. Procesamiento, visualización y análisis de imágenes https://es.mathworks.com/products/image.html
39. Deep Learning Toolbox. Diseño, entrenamiento y análisis de redes de deep learning https://es.mathworks.com/products/deep-learning.html
40. OpenAPI Specification, Version 3.1.0 (Feb-2021) https://spec.openapis.org/oas/v3.1.0
41. Fundamentos de modelado con Rational Software Architect Designer, Version 9.5 https://www.ibm.com/docs/es/rational-soft-arch/9.5?topic=overview-essentials-modeling-rational-software-architect-designer-self-paced-training
42. Mapa creado con las ubicaciones de las cámaras que utiliza la aplicación https://www.google.com/maps/d/edit?mid=1Hq0C79PgFeWxoeky9BY-jtYSV0kqnt1G&usp=sharing
43. Cuenta de Twitter desde la que se publican los tweets con los datos recopilados. https://twitter.com/ControlMadrid
44. Base de datos de iconos gratuitos disponibles en formato PNG, SVG, EPS, PSD y BASE 64. https://www.flaticon.es/
45. Especificaciones Técnicas NIKON D3500 https://www.nikon.com.mx/nikon-products/product/dslr-cameras/d3500.html#tab-ProductDetail-ProductTabs-TechSpecs
46. Slim, a micro framework for PHP to write simple yet powerful web applications and APIs. https://www.slimframework.com/
47. Editor online para describir especificaciones de API REST https://editor.swagger.io/

Anexos

Anexo 1 – Product Backlog

- HU_1 Creación de interfaz principal y menú.
  - Interfaz menú principal.
  - Menú de navegación.
- HU_2 Controller.
  - Creación Controller.
  - Creación ControllerImp.
  - Creación de eventos.
  - Creación de main.
- HU_3 Dispatcher
  - Creación de Dispatcher.
  - Creación DispatcherImp.
- HU_4 Comandos.
  - Creación factoría de comandos.
  - Creación FactoryComandImp.
- HU_5 AlexNet.
  - Interfaz AlexNet.
  - Interfaz selección de parámetros de entrenamiento.
  - Creación transfers parámetros.
  - Lógica de negocio del entrenamiento
  - Integración de red AlexNet.
  - Pruebas y control de errores.
- HU_6 GoogLeNet.
  - Interfaz GoogLeNet.
  - Interfaz selección de parámetros de entrenamiento.
  - Crear evento subir imágenes para AlexNet y GoogLeNet.
  - Lógica de negocio del entrenamiento
  - Creación transfers parámetros.
  - Integración de red GoogLeNet.
  - Pruebas y control de errores.
- HU_7 Subir imágenes.
  - Interfaz subir imágenes.
  - Crear evento subir imágenes para AlexNet y GoogLeNet.
  - Integración vista en dispatcher.
  - Redimensión de imágenes.
  - Lógica de negocio para subir imágenes.
  - Pruebas y control de errores.
- HU_8 Visualización de imágenes.
  - Interfaz de visualizacion.
  - Crear evento visualizar para Alex y Google.
  - Integrar en dispatcher.

- Lógica de visualización.
- Pruebas y control de errores.

- HU_9 Detección de vehículos.
  - Interfaz de detección.
  - Control de reproducción del vídeo.
  - Crear TransferObjetoIdentificado.
  - Crear TransferIdentificado.
  - Lógica de tratamiento de video.
  - Control de conteo y diferenciación de objetos.
  - Pruebas y control de errores.
- HU_10 ThingSpeak[6]
  - Interfaz para subir datos a ThingSpeak[6].
  - Creación de canales en ThingSpeak[6].
  - Lógica en Matlab[7] para subir datos a ThingSpeak[6].
  - Creación cuenta Twitter.
  - Creación de Reacts.
  - Creación ThingTweet.
  - Creación Matlab Analysis.
  - Pruebas y control de errores.
- HU_11 Creación Base de datos.
  - Creación Tablas.
  - Creación relaciones.
  - Creación datos cámaras
  - Pruebas y control de errores.
- HU_12 API PHP para Twitter.
  - Lógica en PHP para postear en Twitter.
  - Implementación seguridad API.
  - Integración con Matlab[7].
  - Twitter Developer Portal.
  - Pruebas y control de errores.
- HU_13 Queries
  - Integrar menú queries en interfaz.
  - Crear interfaz para queries.
  - Lógica de negocio de queries.
  - Pruebas y control de errores.

Anexo 2 – Mockups

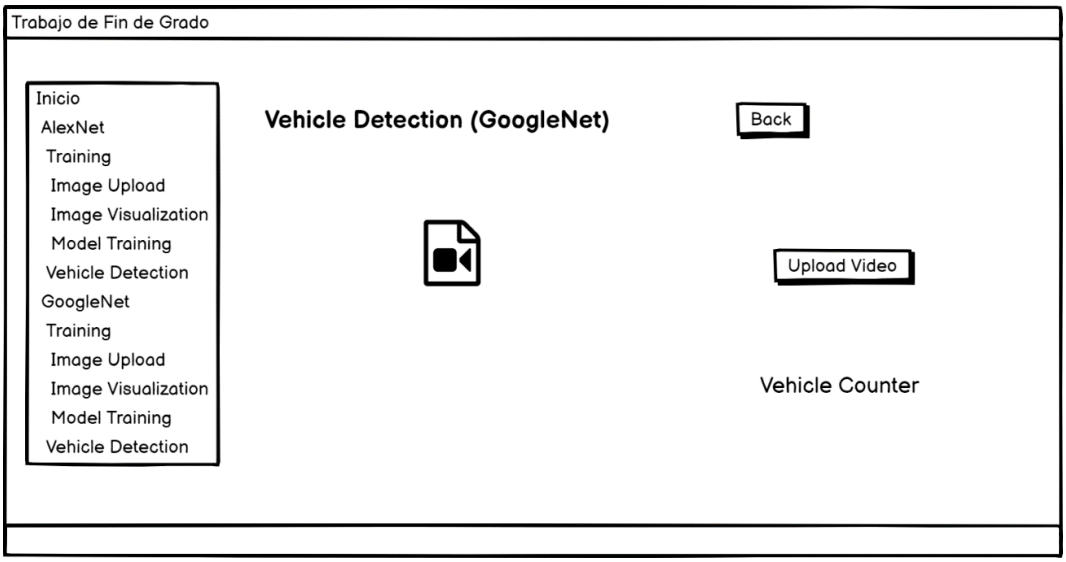


Figura A2.1 – Vehicle Detection GoogLeNet

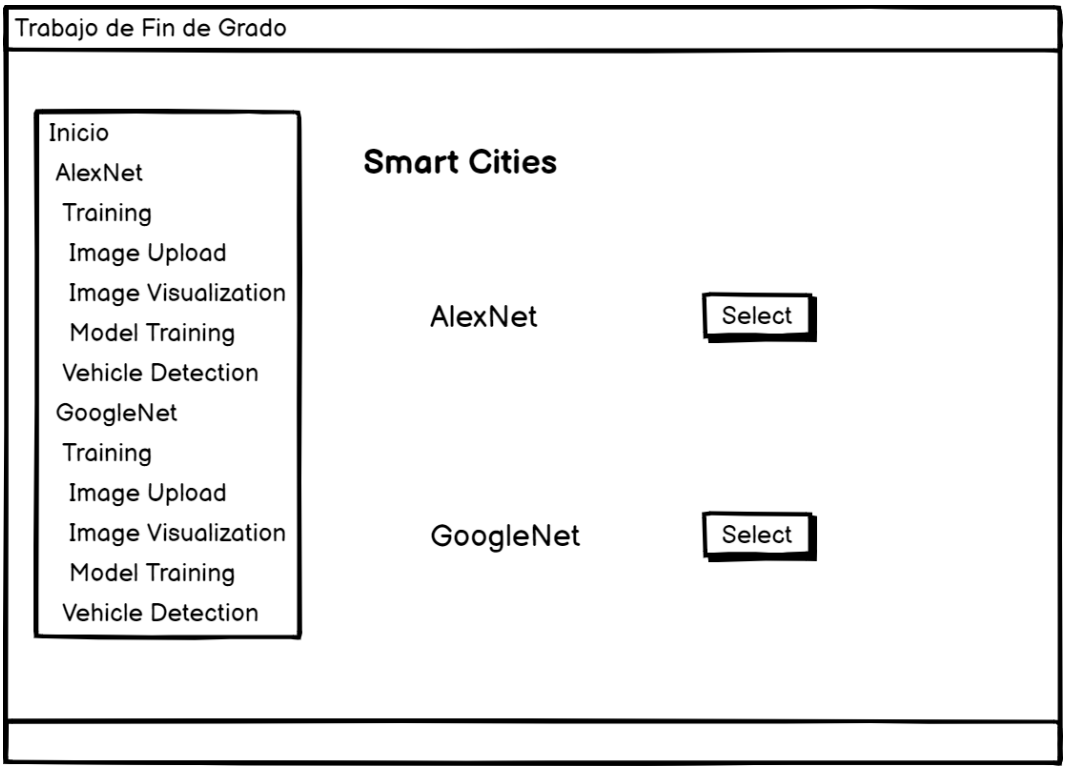


Figura A2.2 – Principal

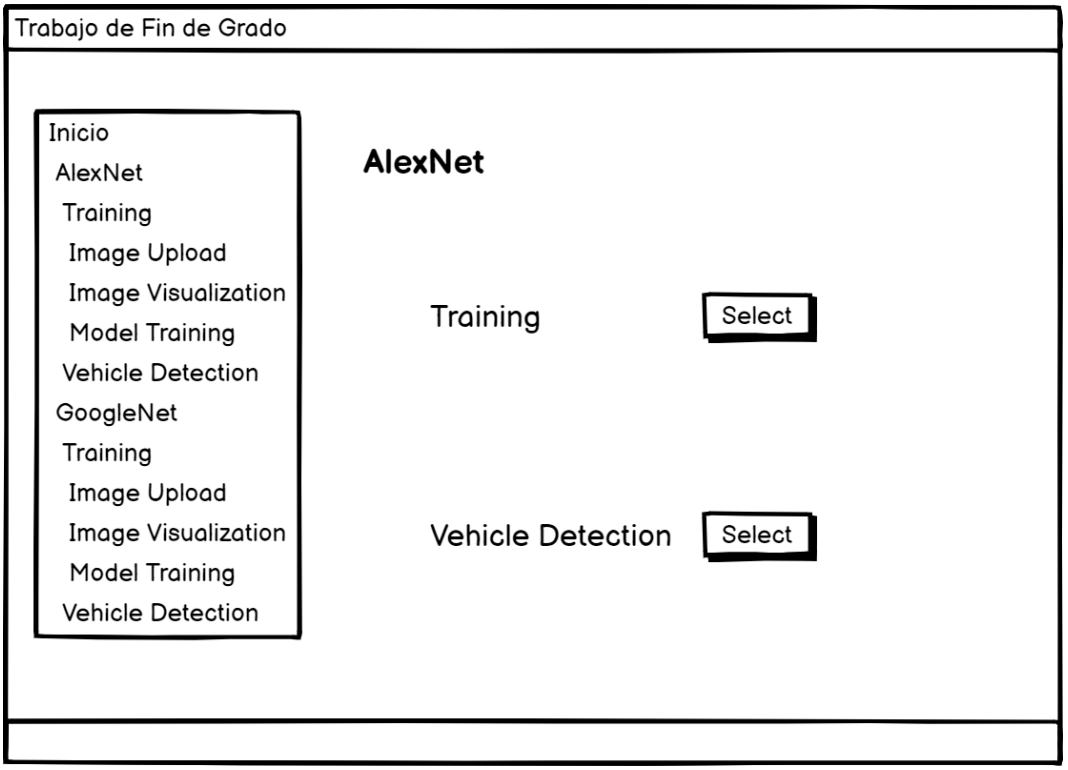


Figura A2.3 – Principal AlexNet

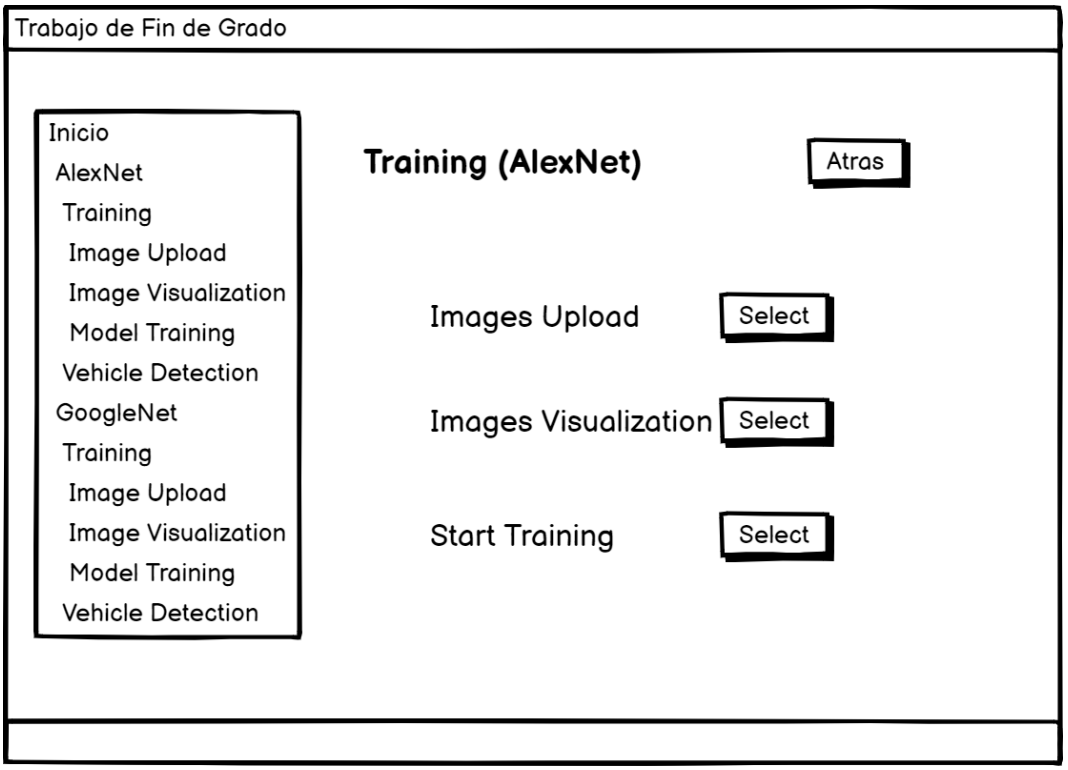


Figura A2.4 – Entrenamiento

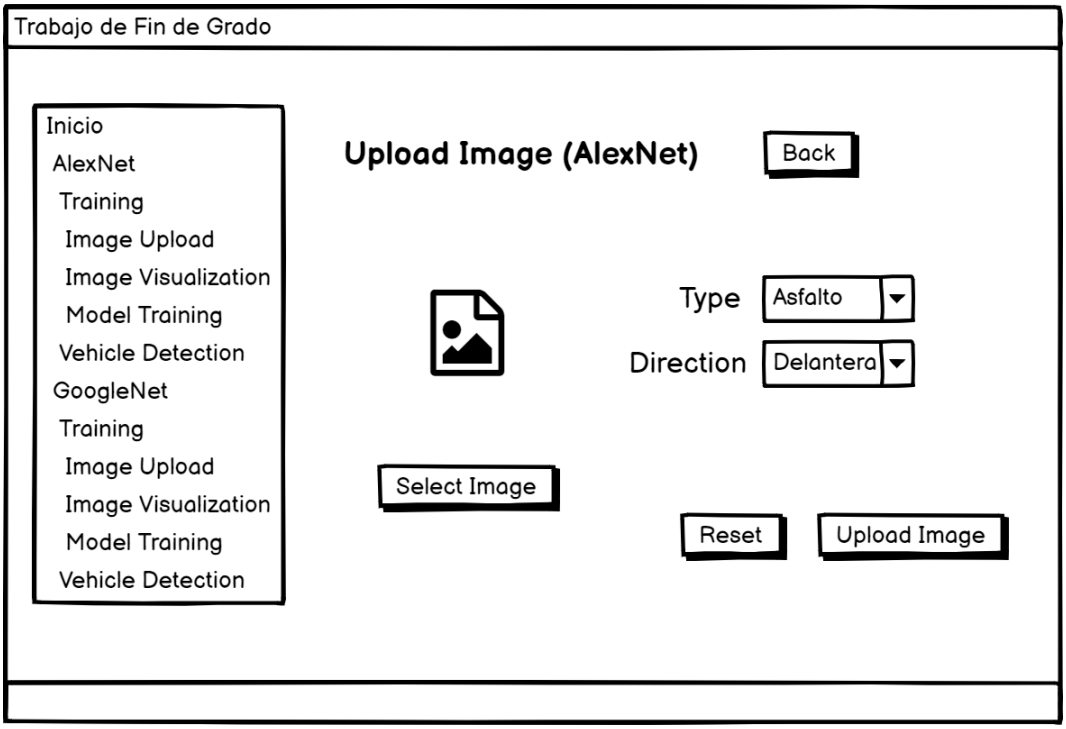


Figura A2.5 – Subida de imágenes

Anexo 3 – Diagramas de clases

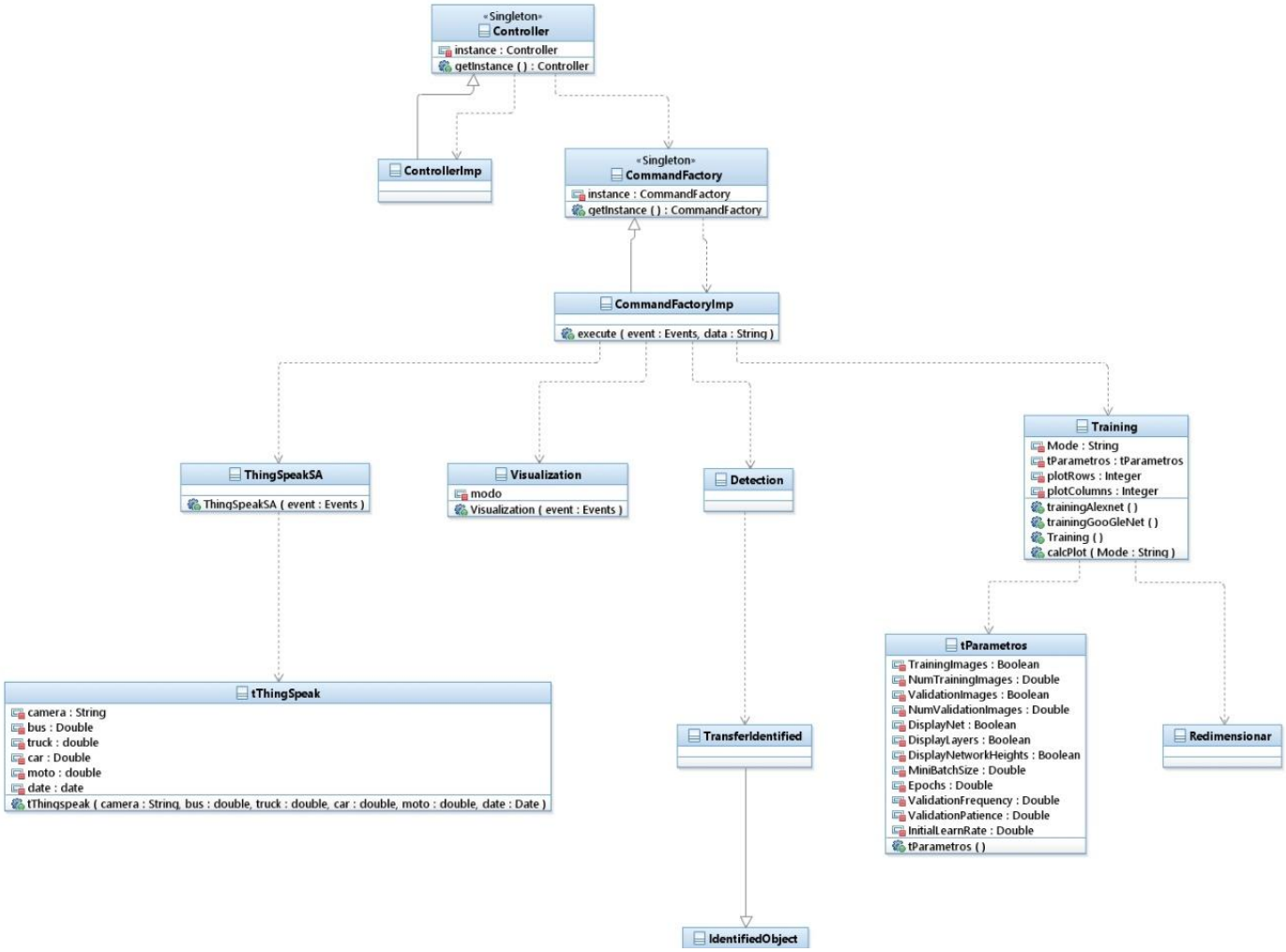


Figura A3.1 – Diagrama de clase de Negocio

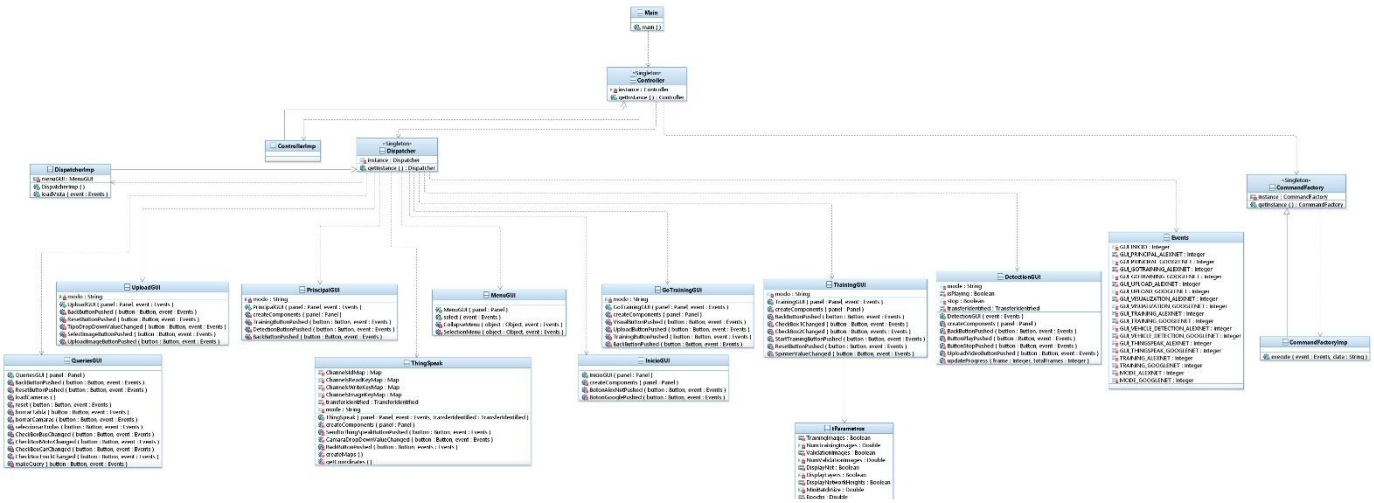


Figura A3.2 – Diagrama de clase de Presentación

Anexo 4 – Instalación y ejecución de la aplicación

1. Descargar el código del repositorio de Github: https://github.com/JesusGranizo/Codigo

2. Instalar MATLAB R2020b así como las siguientes extensiones:

    - ThingSpeak Support Toolbox
    - Mapping Toolbox
    - Symbolic Math Toolbox
    - Statistics and Machine Learning Toolbox
    - Simulink
    - MATLAB Compiler
    - MATLAB Compiler SDK
    - MATLAB Support for MinGW-w64 C/C++ Compiler
    - Image Processing Toolbox
    - Deep Learning Toolbox, Deep Learning Toolbox Model for AlexNet Network y Deep Learning Toolbox Model for GoogLeNet Network
    - Computer Vision Toolbox

3. A partir de aquí existen dos opciones:
    - Abrir con MATLAB la carpeta clonada del proyecto llamada código y ejecutar la clase Main.m desde esta misma carpeta (la ruta que aparece en la parte superior de MATLAB debe ser la de la carpeta /código), se debe hacer de esta forma porque la aplicación utiliza rutas relativas que pueden fallar.
    - La otra opción es descargarse el fichero ejecutable .exe generado a través de MATLAB y disponible en este enlace, después situarlo en la capeta /código y ejecutarlo.

4. Para realizar la detección de vehículos se disponen de videos grabados desde distintos puntos de Madrid y se encuentran disponibles en este enlace. Estos videos deberán encontrarse en una carpeta concreta localmente para poder ser ejecutados desde la aplicación.

Anexo 5 – Licencias de imágenes de la memoria e iconos de la aplicación

1. Imágenes de la memoria.

   Las siguientes imágenes pertenecen al diagrama de capas de la figura 3.1:

   - Imagen World Wide Web.
     https://www.muycomputer.com/wp-content/uploads/2014/03/WWW.jpg

   - Imagen Señal de Wifi.
     https://www.itpedia.nl/wp-content/uploads/2018/07/wifi.png

   - Imagen ThingSpeak.
     https://brands.home-assistant.io/_/thingspeak/logo.png

   - Imagen Matlab:
     https://logos-marcas.com/wp-content/uploads/2020/12/MATLAB-Emblema.png

   - Imagen BBDD.
     https://encrypted-tbn0.gstatic.com/images?q=tbn:usqp=CAU

   - Imagen Twitter.
     https://www.designbust.com/download/1022/png/transparent_twitter_icon256.png

   - Imagen Ordenador.
     https://cerebriti.b-cdn.net/uploads/37a9016871776830.jpg

2. Iconos de la aplicación.

   Las siguientes imágenes se utilizan en la capa de presentación de la aplicación:

   - Imagen Carpeta. 'Folder.svg': Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Good Ware. Disponible desde este enlace.

   - Imagen Google. 'Google.svg': Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Freepik. Disponible desde este enlace.

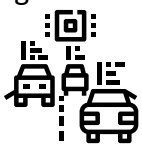
     

- Imagen Pause. ‘Pause.svg’: Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Pixel Perfect. Disponible desde este enlace.

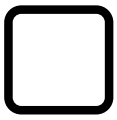
- Imagen Play. ‘Play.svg’: Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Freepik. Disponible desde este enlace.

- Imagen Presentación. ‘Presentation.svg’: Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Freepik. Disponible desde este enlace.

- Imagen Radar. ‘TrafficRadar.svg’: Esta imagen se ha diseñado con recursos de Flaticon.com y creado por PongsakornRed. Disponible desde este enlace.



- Imagen Parar. ‘Stop.svg’: Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Freepik. Disponible desde este enlace.



- Imagen Pensar. ‘Thinking.svg’: Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Smashicons. Disponible desde este enlace.

- Imagen Subida. ‘Upload.svg’: Esta imagen se ha diseñado con recursos de Flaticon.com y creado por Pixel Perfect. Disponible desde este enlace.