UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Reconocimiento de imágenes para la identificación de la tipología de vehículos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Segura Ramos, Daniel

Tutor/a: Tormos Juan, María Pilar

Cotutor/a externo: TONDA BARBERA, JUAN RAMON

CURSO ACADÉMICO: 2022/2023

Resumen

El objetivo del trabajo es aplicar un modelo de reconocimiento de imágenes para ser capaz de identificar la tipología de vehículos captados a través de una cámara analítica. Actualmente se almacena información del tiempo de entrada/salida, matrícula, etc. de los vehículos que entran y salen de un determinado recinto de la empresa, y se pretende conseguir un método para conocer el tipo de estos: camión (y sus distintos tipos), coche, moto, etc. utilizando el aprendizaje automático, para poder extraer esta información e integrarla en un conjunto mayor de datos.

Palabras clave: aprendizaje automático, reconocimiento de imágenes, redes neuronales, python, vehículos.

Resum

L'objectiu del treball és aplicar un model de reconeixement d'imatges per a ser capaç d'identificar la tipologia de vehicles captats a través d'una càmara analítica. Actualment s'emmagatzema informació del temps d'entrada/eixida, matrícula, etc. dels vehicles que entren i ixen d'un determinat recinte de l'empresa, i es pretén aconseguir un mètode per a conéixer el tipus d'aquests: camió (i els seus diferents tipus), cotxe, moto, etc. utilitzant l'aprenentatge automátic, per a poder extraure aquesta informació i integrar-la en un conjunt major de dades.

Paraules clau: aprenentatge automàtic, reconeixement d'imatges, xarxes neuronals, python, vehicles.

Abstract

The objective of the project is to apply a model of image recognition to be able to identify the typology of vehicles captured through an analytical camera. At the moment information on entry/exit time, license plate, etc. of the vehicles that enter and leave a certain precinct of the company is stored, and it is intended to achieve a method to know the type of these: truck (and its different types), car, motorcycle, etc. using machine learning, to be able to extract this information and integrate it into a larger data set.

Keywords: machine learning, image recognition, neural networks, python, vehicles.

Agradecimientos

Gracias a mis dos tutores del TFG por haber hecho posible la realización de este trabajo. A Pilar por ayudarme siempre en todo cuanto he necesitado, y a Juan por dedicar parte de su tiempo en ser la guía que ha puesto rumbo a este proyecto.

Gracias a Víctor Giner por ofrecerme la oportunidad de formar parte del equipo de Transformación Digital, y gracias también a todos los compañeros/as que lo forman: Javi, Mabel, Eva, Paco, Nacho, Inma, Luisa… por haberme acogido como uno más del grupo desde el primer día. Gracias a Álvaro por su inmensurable ayuda.

Y, por último, gracias a mi familia por haberme brindado su apoyo ininterrumpidamente, y por haberme permitido llegar hasta donde estoy hoy.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1. Contexto y justificación del trabajo

El presente trabajo de fin de grado se enmarca en un entorno real empresarial, y más concretamente, en el ámbito de uno de los principales puertos del Mediterráneo y del mundo: el Puerto de Valencia. Por él transcurre aproximadamente un tercio de las operaciones de importación y exportación del Estado Español, generando así importantes volúmenes de tráfico de vehículos de entrada y de salida.

Cómo no puede ser ya de otro modo, el Puerto de Valencia se está adaptando a los nuevos paradigmas en la gestión y el gobierno del dato, debido a que los puertos, en general, son enormes generadores de datos, todos ellos muy heterogéneos y procedentes de muy diversos orígenes. Por ello, se hace necesaria una gestión integral del dato con el fin de encontrar el mejor modo de integrar dicha información y convertir al dato en un activo de valor empresarial.

Entre los muchos retos que se están llevando adelante en este campo, uno de ellos es el de poner en valor la información recogida por las cámaras de vídeo instaladas en el Puerto de Valencia, y es en ese marco de trabajo en el que se va a desarrollar el presente proyecto.

Uno de los campos que ha tomado importancia en los últimos años dentro del mundo de la inteligencia artificial es la visión por computador, entendida como el proceso encargado de percibir la realidad a través de la vista (vídeos e imágenes) y extraer información a partir de ella [1]. En particular, la rama de la visión por computador que nos interesa para este trabajo es la conocida como reconocimiento de imágenes.

El reconocimiento de imágenes, también llamado reconocimiento de formas/patrones/objetos, es la ciencia que se encarga de extraer características propias de un mismo conjunto para identificar y clasificar los objetos pertenecientes a este. De manera simple, el proceso de reconocimiento de imágenes se reduce a tres pasos: la adquisición de los datos, la extracción de características y, por último, la clasificación [2].

Dentro del reconocimiento de imágenes, una de las técnicas más utilizadas es el Deep Learning, y en particular, el uso de redes neuronales convolucionales (CNN por sus siglas en inglés) [3]. Esta estrategia consiste en crear un modelo basado en redes neuronales y entrenarlo, de tal manera que él solo sea capaz de extraer las características y clasificar los objetos en su conjunto correspondiente. Para ello, antes habrá que preparar los datos de entrenamiento, y una vez terminado el modelo, comprobar si efectivamente clasifica de forma correcta los datos de prueba.

Por tanto, es en el campo del reconocimiento de imágenes en el que voy a centrar el presente trabajo. Para ello, éste se va a limitar a un proyecto piloto que va a consistir en el reconocimiento de imágenes reales tomadas de la salida 2 del Acceso Sur del Puerto de Valencia, cuyo objetivo

será, a partir del desarrollo y entrenamiento de una red neuronal convolucional, detectar la tipología de los vehículos que salen por ese carril, identificarlos y almacenarlos en una base de datos con la fecha y hora de salida, la matrícula y la tipología del vehículo.

Mi objetivo personal, como futuro profesional en esta rama de la inteligencia artificial, es con el desarrollo de este proyecto adquirir la experiencia de haber trabajado en un entorno real y ver los resultados obtenidos en el marco de un entorno profesional práctico donde los problemas surgidos van más allá del marco teórico.

1.2. Objetivos del trabajo

Este proyecto surge de la necesidad por parte del Puerto de Valencia de adaptarse a los nuevos paradigmas en la gestión del dato. En esta circunstancia, adquirir información relevante sobre los vehículos que salen y entran de la empresa (como, por ejemplo, la tipología de los mismos) se ha convertido en una tarea de gran importancia, y es una parte fundamental de proyectos de mayor alcance como la trazabilidad de los vehículos dentro del puerto. Por ello, con este trabajo se pretende aportar una nueva y significativa información acerca del tráfico que circula en los recintos de la empresa, mediante el uso de las redes neuronales artificiales y el reconocimiento de imágenes.

El objetivo principal del trabajo es el siguiente:

- Desarrollar un software de reconocimiento de imágenes a partir de una Red Neuronal Convolucional que permita identificar la tipología de vehículos a partir de vídeo en tiempo real.

Para alcanzar este objetivo principal se tendrán que desarrollar los siguientes objetivos secundarios:

- Recopilar datos procedentes de las imágenes históricas grabadas.
- Etiquetar los datos recogidos anteriormente
- Crear una BBDD de imágenes.
- Desarrollar un modelo basado en redes neuronales
- Probar dicho modelo en un entorno real.
- Presentar resultados.
- Estudiar la tecnología utilizada

1.3. Estructura del trabajo

El presente trabajo está organizado en doce capítulos claramente diferenciados, más el añadido de cuatro apéndices/anexos al final del documento. De forma resumida, los cinco primeros puntos servirán para introducir el propio tema del trabajo, las cuestiones teóricas que involucra, el entorno en el que ha sido elaborado, etc. Los capítulos seis, siete, ocho y nueve se centrarán en la explicación detallada del desarrollo del software de reconocimiento de imágenes, desde sus inicios hasta su implantación y resultados. Los últimos capítulos, del diez al doce, tratarán sobre las conclusiones, los trabajos futuros y la bibliografía utilizada a lo largo de todo el documento. Por último, los apéndices contendrán información adicional como los Objetivos de Desarrollo Sostenible potenciados en este trabajo o el código completo que ha sido desarrollado durante su realización.

El primer capítulo, tras haber visto el resumen del trabajo, los agradecimientos y los diferentes índices, pretende introducir el tema del trabajo que se va a desarrollar en las siguientes páginas, así como el contexto en el que ha sido realizado y los objetivos que persigue.

El segundo punto se centra en los diversos productos o softwares que se han ido desarrollando como soportes al software principal de reconocimiento de imágenes, y que son vitales para el correcto funcionamiento del mismo.

El tercer capítulo es un estudio sobre el Estado del arte del campo del reconocimiento de imágenes, y necesariamente de la visión por computador y de la Inteligencia Artificial también. También se hablará del uso de estas tecnologías en entornos portuarios que se está produciendo en la actualidad.

El cuarto capítulo es una introducción a los conceptos previos, principalmente teóricos, cuya comprensión es necesaria si se pretende entender con solvencia el presente trabajo. Estos conceptos abarcan desde temas más generales, como el reconocimiento de imágenes, hasta tópicos más concretos como las redes neuronales convolucionales.

El punto número cinco es un análisis de la tecnología con la que ha sido desarrollado el proyecto, desde las librerías Python y los entornos de trabajo hasta las cámaras y bases de datos utilizadas.

Una vez terminados los capítulos introductorios, el sexto punto da comienzo a la exposición del desarrollo del software de reconocimiento de imágenes. En este punto se presenta el planteamiento inicial del proyecto, el cual se centra principalmente en el trabajo de campo realizado.

El séptimo capítulo comprende todo el proceso de desarrollo del software, desde la creación del banco de imágenes a partir del software de extracción de fotogramas hasta el desarrollo de la red neuronal y las diferentes pruebas con arquitecturas de redes neuronales ya conocidas, además de la creación del software para la lectura automática de matrículas.

El punto número ocho se centra en la implantación del software de reconocimiento de imágenes en un entorno real. Para ello, tal y como se explicará en dicho punto, el software es integrado en un sistema formado por diversos módulos que, en su conjunto, permiten obtener, además de la tipología, la fecha y la matrícula de los vehículos que abandonan el Puerto de Valencia por el carril 2 del Acceso Sur.

El capítulo nueve muestra los resultados de la implantación explicada anteriormente. Se presentan imágenes obtenidas de casos reales junto con su posterior tratamiento, y cómo ello permite adquirir la información ya mencionada, formada por la fecha, matrícula y tipología.

El décimo capítulo ofrece las conclusiones alcanzadas tras la realización del presente trabajo y el desarrollo del software que lo acompaña, analizando todas las tareas abordadas y los resultados que se han obtenido de ellas.

El capítulo número once expone los posibles trabajos futuros, una colección de ampliaciones que podrían añadirse al presente trabajo con el fin de mejorarlo, y que no han sido incorporadas por cuestiones temporales o económicas.

El último capítulo antes de los apéndices, el número doce, es la bibliografía utilizada, la cual muestra todas las fuentes empleadas a lo largo del trabajo.

La última parte del trabajo está formada por cuatro apéndices. El primero, el apéndice A, se centra en los Objetivos de Desarrollo Sostenible que han sido tratados en el presente trabajo. El apéndice B es un glosario de términos, en el que se encuentran explicadas expresiones de aparición frecuente. El apéndice C muestra las diferentes arquitecturas de redes neuronales que fueron estudiadas junto con sus tablas de resultados, y que permitieron diseñar la red neuronal definitiva que se utiliza a lo largo de todo el proyecto. Por último, el apéndice D proporciona el código completo de la red neuronal diseñada, de la implantación del capítulo 8 (donde ya se incluye la lectura automática de matrículas), de la posterior comprobación de los datos con una base de datos externa y de la extracción de fotogramas a partir de vídeo.

2. Metodología

2.1. Software de reconocimiento de imágenes

Con este trabajo se desea desarrollar un software de reconocimiento de imágenes capaz de identificar la tipología de vehículos a partir de las imágenes obtenidas con cámaras de vigilancia. Para que el tipo obtenido no sea un simple dato sin valor informativo, éste va a ser integrado con otros datos relacionados con el mismo vehículo, consiguiendo así información relevante acerca de los accesos de la entidad. Para ello, se juntará el dato sobre la tipología de un vehículo con otros como la matrícula del mismo o la fecha y hora en la que se produce la salida.

Por tanto, se va a implementar un modelo de entrenamiento basado en redes neuronales convolucionales que pueda clasificar los vehículos en función de su tipología en tiempo real. Este modelo recibirá imágenes en las que aparecerán los vehículos a identificar, y gracias al entrenamiento previo, será capaz de predecir el tipo de dichos vehículos.

Por otro lado, la misma imagen utilizada para obtener la tipología del vehículo será empleada para detectar y extraer la matrícula del mismo. Posteriormente, estos dos datos se unirán entre ellos y con otros datos significativos para la identificación del vehículo –como la fecha y hora de la salida–, y esta nueva información se almacenará en la base de datos seleccionada.

2.2. Software de recuperación de imágenes

Uno de los componentes fundamentales en un reconocedor de imágenes basado en el aprendizaje supervisado (del cual se hablará con más detalle en el apartado 4.4) es el conjunto de datos etiquetados correctamente que el modelo utilizará como datos de entrenamiento, y a partir de los cuales el reconocedor aprenderá a clasificar cada objeto en su correspondiente categoría, con el objetivo final de adquirir la capacidad de clasificar correctamente objetos no vistos con anterioridad.

La creación de este conjunto de datos de entrenamiento –en nuestro caso, un banco de imágenes etiquetado– supone un esfuerzo considerable en caso de hacerse totalmente de forma manual, por lo que se ha diseñado un software de recuperación de imágenes a partir de vídeos, automatizando así la parte de obtención de las imágenes.

Este programa, escrito en Python, extrae la cantidad deseada de frames de cada segundo del vídeo y los almacena en la ubicación que se le especifique, dejando así como única tarea manual el proceso de etiquetar cada ejemplo con su correspondiente categoría. Una muestra del código se este programa se puede ver en la figura 7.1.

2.3. Software para la lectura automática de matrículas

Como se ha mencionado anteriormente, aunque el objetivo principal del presente proyecto sea la identificación de la tipología a partir de imágenes, también es necesario obtener otro tipo de datos con los que integrar la información sobre el tipo, con el fin de crear unos resultados de utilidad.

En este contexto, se hace necesario encontrar una forma de conseguir la matrícula del vehículo a partir de la misma imagen de la que se obtiene la tipología. Tras contemplar varias opciones (explicadas detalladamente en los apartados 7.3 y 8.3), se ha optado por diseñar un código propio que será el encargado de, por una parte, detectar la matrícula dentro de la imagen, y por otra, aplicar el reconocimiento óptico de caracteres sobre ella para adquirir el número de matrícula. El código de este software se puede observar en la figura 7.7.

3. Estado del arte

La visión por computador (CV, por sus siglas en inglés) es una rama de gran importancia dentro del mundo de la Inteligencia Artificial (AI). Tiene como objetivo simular la visión humana, es decir, ser capaz de percibir el mundo a través de la vista y extraer información de él. Surgió en la década de los 60 como parte del proceso encargado de dotar a los ordenadores con comportamientos inteligentes, y desde entonces no ha dejado de evolucionar [4].

En los años 70 aparecieron estudios que definían lo que hoy en día son las bases de los algoritmos utilizados en visión por computador, como por ejemplo la extracción de bordes [4]. Dos décadas más tarde, en 1991, M. Turk y A. Pentland publicaron un artículo en el que se hacía uso de la visión por computador para el reconocimiento de rostros humanos (*eigenfaces*) [5]. Hoy en día las técnicas de Deep Learning son las más utilizadas en este campo de la AI, superando la precisión de técnicas previas en diversos ámbitos [6].

El *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), una competición anual creada en 2010 por *ImageNet* donde diversos softwares compiten en tareas de clasificación y detección de objetos [7], ha contribuido enormemente al desarrollo de la visión por computador y a la innovación en arquitecturas de modelos de redes neuronales, especialmente entre 2012 y 2016 [8].

La visión por computador a su vez se puede dividir en varias ramas, entre las que se pueden incluir la clasificación de imágenes, la detección de objetos o la segmentación semántica, todas ellas pertenecientes al campo del reconocimiento de imágenes (del que hablaremos más tarde). Estas subramas están basadas, generalmente, en modelos de redes neuronales convolucionales (de los cuales también hablaremos más tarde), y en función de la tarea a realizar se emplean distintos tipos de modelo [9].

Para la función de clasificación de imágenes existen gran cantidad de arquitecturas disponibles actualmente. Entre ellas, destacan ResNet (y el resto de arquitecturas de la misma familia: Res2Net, ResNeXt y ResNext), AlexNet, Inception y VGG, todas ellas ganadoras del ILSVRC.

En el caso de la detección de objetos, hoy en día se suele utilizar arquitecturas del tipo SSD (*Single Shot Detection*) como YOLO o RetinaNet por delante de arquitecturas de dos etapas, las cuales se ocupan primero de encontrar el objeto dentro de la imagen y clasificarlo después en la segunda etapa. Esta preferencia por el tipo SSD se debe a que ofrece una mejor relación resultados/tiempo, es decir, aunque las arquitecturas de dos etapas puedan producir modelos de mayor precisión es preferible el uso de las SSD por su velocidad [10].

En el campo de la segmentación semántica de imágenes, encargada de clasificar los objetos a nivel de píxeles, los modelos más comunes en la actualidad son SegNet y Mask R-CNN, ésta última también formada por dos etapas.

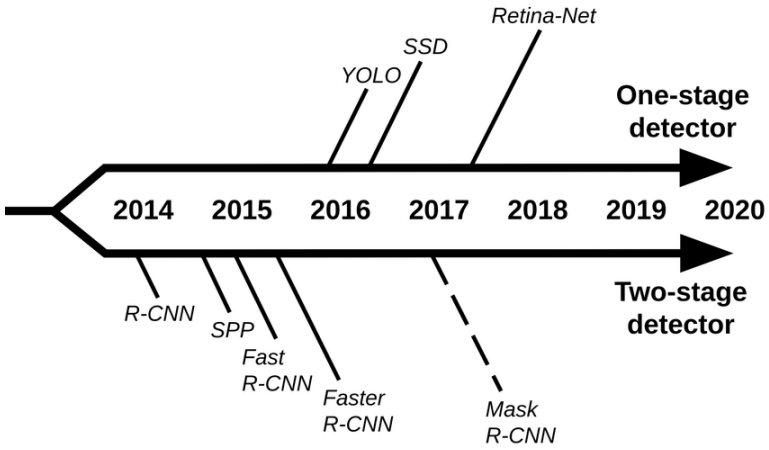

Figura 3.1: Evolución de las CNN en la detección de objetos

Además de los nuevos modelos que surgen para cada rama de la visión por computador, constantemente van apareciendo nuevas ideas dentro de este campo [11], tales como:

- Aprendizaje con ejemplos adversos: imágenes con mucho ruido pueden hacer que un modelo entrenado para la detección de objetos o el reconocimiento de imágenes produzca resultados incorrectos. Para evitar este problema, se propone incluir ejemplos adversos (imágenes con ruido) en el proceso de entrenamiento del modelo, y de esta forma conseguir que no sea engañado en el futuro [12].

- Aprendizaje contrastivo y auto-supervisado: etiquetar grandes cantidades de datos supone un gran esfuerzo humano. Una posible solución a este problema sería que los datos se auto-supervisen a sí mismo, y de esta forma poder entrenar a los modelos con grandes cantidades de datos sin etiquetar. Para ello, se utiliza la técnica *contrastive learning* (CL), que compara ejemplos para aprender características comunes en individuos de la misma clase y diferentes en individuos de distintas clases [13].

El reconocimiento de imágenes o reconocimiento de objetos es un campo de la visión por computador que, como se ha mencionado anteriormente, está formada principalmente por las ramas de clasificación de imágenes, detección de objetos y segmentación semántica, entre otras. Convencionalmente, los métodos más utilizados en el reconocimiento de imágenes han sido los pertenecientes al campo del Machine Learning (ML). Estos métodos consisten en la extracción de características y en la posterior clasificación en función de dichas características. Sin embargo, este proceso supone una limitación significativa, pues requiere de un importante trabajo manual para la extracción de características.

En la actualidad, para solucionar esta problemática, es más frecuente el uso del Deep Learning (DL), un campo dentro del ML capaz de ofrecer buenos resultados sin el costoso trabajo de

extracción de características. A diferencia del ML, el DL encuentra las características más relevantes por su propia cuenta, sin tener que estar predefinidas con anterioridad. Por estas razones, el Deep Learning, y en concreto las redes neuronales convolucionales (CNN, *Convolutional Neural Networks)*, se han convertido en métodos de gran popularidad en el ámbito de la visión por computador y el reconocimiento de imágenes [14].

Las CNN, un tipo concreto de redes neuronales artificiales, surgieron en la década de los 1980's cuando se creó una arquitectura basada en la extracción de características, las capas de *pooling* y las convoluciones dentro de una red neuronal, llamada *Neocognitron* [15]. Esta arquitectura estaba inspirada en el sistema visual, y en concreto en el modelo propuesto para esta parte del sistema nervioso por Hubel y Wiesel [16].

Sin embargo, fue en 1998 cuando se le apodó con el nombre por el que hoy conocemos a esta arquitectura, *Convolutional Neural Network*, con la publicación de un artículo científico escrito por Yann LeCun y sus compañeros en el que mostraban el diseño de la arquitectura LeNet, desarrollada para el reconocimiento de dígitos manuscritos [17]. Desde entonces, el campo de las CNN ha ido evolucionando y expandiéndose, con la aparición de nuevas arquitecturas como AlexNet, VGG (16) Net, GoogLeNet o Microsoft ResNet, y diferentes *frameworks* como YOLO (*You Only Look Once*) o SSD (*Single Shot Detector*) [18].

En los últimos años, arquitecturas mencionadas anteriormente como GoogleLeNet o AlexNet, y otras como ResNet50, han sido ampliamente utilizadas para el diseño de CNN dedicadas al reconocimiento de imágenes, llegando a la conclusión de que las redes neuronales convolucionales son la mejor técnica disponible para la clasificación de objetos del mundo real [19].

Para concluir, se ha buscado en la literatura casos en los que se ya se haya empleado el reconocimiento de imágenes en un entorno portuario, a fin de comprobar si se había resuelto la misma necesidad que presenta este proyecto con anterioridad, y en tal caso, de qué manera se había hecho y si podría mejorarse. El interés particular en el uso de esta tecnología dentro de puertos se debe a que, como se ha mencionado anteriormente, estos lugares contienen un enorme flujo de personas, vehículos y materiales circulando por ellos y, por tanto, son una fuente inmensa de datos.

No se ha obtenido ningún artículo científico o *paper* en el que se haya aplicado esta tecnología para la identificación y clasificación de vehículos por su tipología dentro de un puerto, pero sí algunos proyectos que utilizan el reconocimiento de imágenes para distintos fines. El reconocimiento de imágenes se ha utilizado dentro de puertos para la lectura del número de contenedor [20] [21], con un objetivo relativamente similar al perseguido en este trabajo, pero enfocado a la tecnología OCR. También se ha empleado en el seguimiento de buques [22] y en la detección de objetos anómalos [23], además de otros posibles usos no registrados en *papers*.

Fuera del ámbito portuario, se han encontrado artículos en los que se utiliza el reconocimiento de imágenes para extraer atributos de vehículos (tipología, fabricante y modelo), e incluso para seguir a un vehículo a través de diferentes cámaras gracias a estos atributos [24]. Sin embargo, en todos estos casos la tipología de los vehículos analizados no abarca el rango que se pretende alcanzar con este proyecto, pues sólo se centran en la tipología de vehículos ligeros (turismos,

motocicletas, etc.), mientras que en este proyecto se consideran también los diversos tipos de camiones y de más vehículos pesados. Esto supone una dificultad añadida, pues se requiere una precisión mayor para ser capaz de diferenciar los múltiples tipos de camiones que se pretende clasificar.

4. Conceptos previos

En este apartado se pretende explicar conceptos relevantes relacionados con la visión por computador, el reconocimiento de imágenes y las redes neuronales convolucionales. El entendimiento de estos conceptos es fundamental para la comprensión del software y de las decisiones tomadas en el desarrollo del mismo.

Se profundizará en aspectos más generales, pero necesarios, como la visión por computador, el reconocimiento de imágenes o el Machine Learning; y también en temas más específicos para el cometido de este proyecto, como el Deep Learning, las redes neuronales artificiales (y en particular, las convolucionales) o las diferencias entre aprendizaje supervisado y no supervisado.

4.1. Visión por computador

La visión por computador (CV), también llamada visión artificial, es un campo de la inteligencia artificial (IA) que pretende dotar a los ordenadores de la capacidad de extraer información relevante a partir de imágenes del mundo real. Del mismo modo que los humanos obtenemos información a través de nuestros ojos y la procesamos en nuestros cerebros para comprenderla, la visión por computador busca que los ordenadores puedan adquirir esta misma habilidad que les permita ver, observar y comprender [1].

Generalmente, cuando se habla de visión por computador se distinguen dos campos: el reconocimiento de imágenes y la visión de máquina (Machine Vision). El reconocimiento de imágenes, en el cual profundizaremos más adelante, trata de reconocer los objetos presentes en una imagen, en una aproximación más científica y divulgativa a la visión artificial. Por el contrario, la Machine Vision toma un enfoque más industrial y utiliza la tecnología en visión por computador para tareas como la inspección automática o controles de procesos [25].

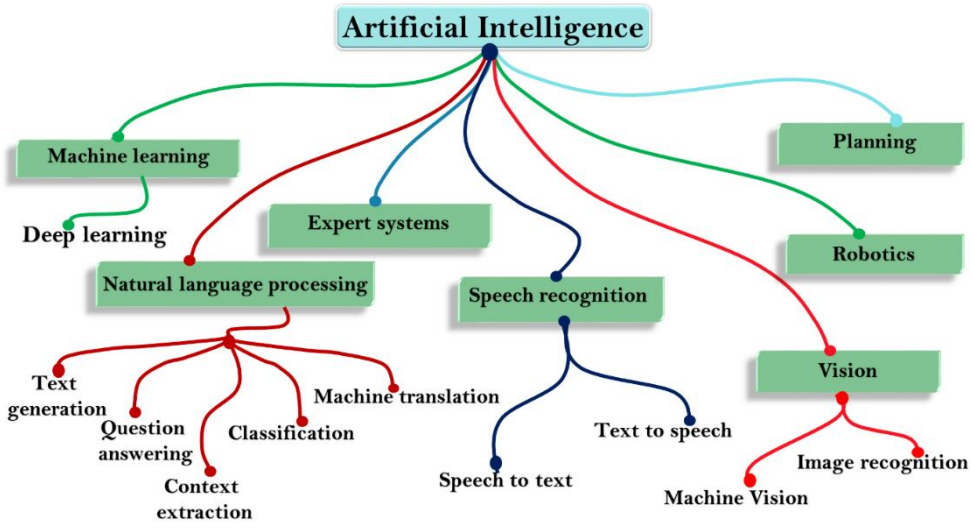

Figura 4.1: Ramas de la Inteligencia Artificial

Hoy en día, la visión por computador es utilizada en tareas como el reconocimiento óptico de caracteres (OCR), el reconocimiento facial y el reconocimiento de iris. Además, es una herramienta esencial en industrias como la videovigilancia, la medicina (InnerEye, por ejemplo, es un software de Microsoft diseñado para identificar tumores y diversas anomalías en radiografías), la conducción autónoma (como en los coches Tesla), la inspección industrial (en empresas como Amazon), las tiendas sin cajeros (Amazon Go, donde los clientes acceden a la tienda, recogen el producto, salen sin pasar por caja y reciben después el recibo, gracias a que a partir de la visión por computador se conoce qué producto llevaba cada cliente), la banca (con

sistemas como Mitek Systems para la clasificación de documentos, la extracción de información y la identificación de individuos) o las compañías de seguros (empresas como Cape Analytics y Betterview utilizan la visión por computador para obtener información sobre los factores de riesgo de una propiedad y el precio de la póliza de seguro que debería tener) [26].

4.2. Reconocimiento de imágenes

El reconocimiento de imágenes o reconocimiento de objetos es una rama dentro del campo de la visión por computador que pretende reconocer los objetos presentes en una imagen, tal y como haría un humano. A diferencia de la visión por computador, que consiste en un conjunto de técnicas que permiten la automatización de tareas a partir de imágenes o vídeos, el reconocimiento de imágenes se centra en actividades más concretas [27], como:

- Clasificación: clasificar una imagen en su clase correspondiente (sólo una clase por imagen)
- Etiquetado: clasificación de varios objetos dentro de la imagen (por tanto, una clase puede tener diversas etiquetas)
- Detección: localizar un objeto dentro de la imagen (normalmente, encuadra el objeto dentro de un rectángulo)
- Segmentación: localizar un objeto dentro de la imagen, pero con mayor precisión (localización píxel a píxel, no a través de un rectángulo)

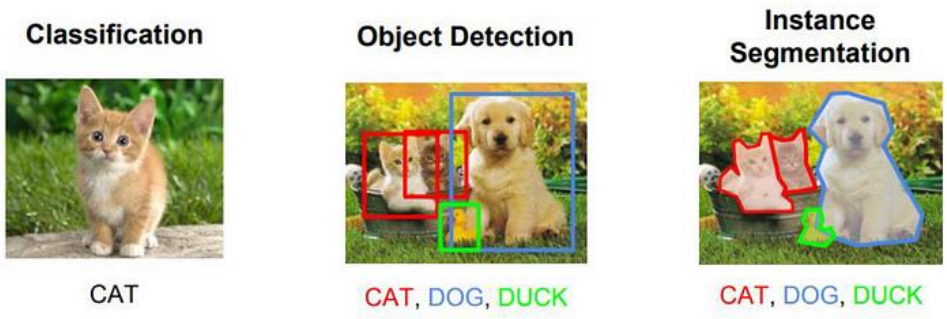

Figura 4.2: Ramas del reconocimiento de imágenes

Generalmente, el proceso de reconocimiento de imágenes se puede dividir en 3 pasos [28]:

1- Creación de un conjunto de datos etiquetados
2- Desarrollo de un modelo de entrenamiento
3- Comprobación de predicciones con imágenes que no formen parte del entrenamiento

Inicialmente se deberá crear un conjunto de datos etiquetados, es decir, datos de ejemplo previamente clasificados. Estas clases se corresponden con los diferentes tipos de objeto que el modelo deberá reconocer. A continuación, se desarrolla un modelo de entrenamiento que aprenda de los datos etiquetados, es decir, que sea capaz de extraer las características más significativas de cada grupo de objetos o clase. Por último, se comprueba que el modelo reconoce correctamente mostrándole ejemplos de objetos que no haya visto antes.

Hoy en día, el reconocimiento de imágenes se utiliza en tareas de reconocimiento facial, análisis médico, control de calidad o de búsqueda visual (Google Lens). Además, se continúa

trabajando en el desarrollo de esta tecnología y se espera que, a medida que avance, aporte soluciones cada vez mejores para industrias como la conducción autónoma, las gafas inteligentes, la realidad aumentada o la predicción de comportamiento del consumidor [28].

4.3. Machine Learning y Deep Learning

El Machine Learning (ML) o Aprendizaje Automático apareció en la década de los 80 como una nueva rama de la Inteligencia Artificial y, como su nombre indica, permite a las máquinas "aprender", es decir, mejorar gracias al uso de datos y a la experiencia [29]. Según Peter Norvig y Stuart Russell, informáticos conocidos por sus contribuciones al mundo de la IA, "*en el aprendizaje de máquinas un computador observa datos, construye un modelo basado en esos datos y utiliza ese modelo a la vez como una hipótesis acerca del mundo y una pieza de software que puede resolver problemas*" [30].

Dentro del campo del Machine Learning existen una amplia variedad de modelos, algoritmos y técnicas de clasificación. Los modelos se pueden clasificar de dos formas: por una parte, es posible diferenciar modelos de tipo geométrico, probabilístico y lógico; por otra, también se pueden clasificar como modelos de agrupamiento y de gradiente.

Respecto a los algoritmos, éstos se clasifican según su salida, y podemos encontrarnos principalmente con algoritmos de aprendizaje supervisado, no supervisado y semisupervisado, entre otros. Más adelante profundizaremos en la diferencia entre estos tipos de algoritmos.

En cuanto a las técnicas de clasificación, se pueden diferenciar varios tipos como los árboles de decisión, los algoritmos genéticos, las máquinas de vectores soporte o las redes neuronales artificiales [29].

En el año 2011 surgió una nueva rama dentro del ML, llamada Deep Learning (DL) o Aprendizaje Profundo. Este nuevo aprendizaje se diferenciaba por el uso de unos algoritmos distintos: las redes neuronales. Las redes neuronales, que explicaremos más adelante, tienen un funcionamiento similar a las conexiones neuronales biológicas de nuestro cerebro [31].

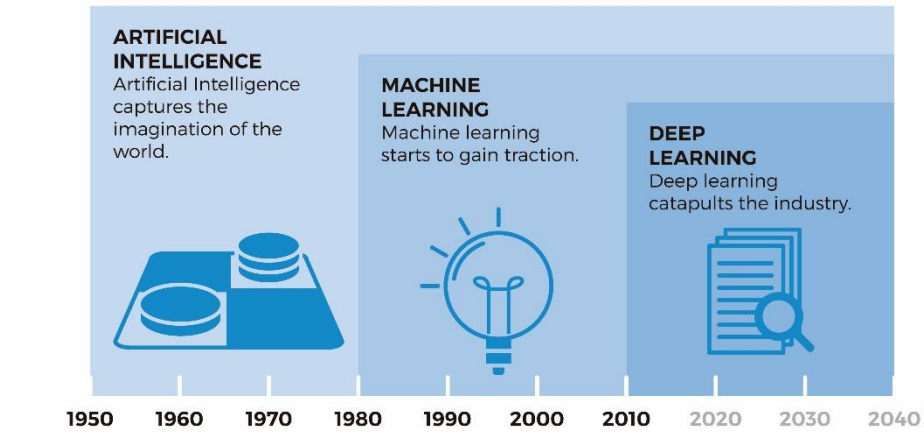

Figura 4.3: Evolución de la Inteligencia Artificial, Machine Learning y Deep Learning

A diferencia del Machine Learning, el Deep Learning hace uso de estas estructuras similares a las neuronas humanas para detectar características propias en los objetos percibidos. En el caso del DL aplicado a la visión por computador, las redes neuronales artificiales pretenden simular la zona del cerebro humano que se encarga de reconocer bordes, inclinaciones de líneas, simetrías e incluso rostros y expresiones [32].

En el caso de reconocimiento de imágenes, si utilizáramos Machine Learning seleccionaríamos manualmente las características relevantes de las imágenes, mientras que con Deep Learning entrenaremos un modelo supervisado para que las aprenda automáticamente.

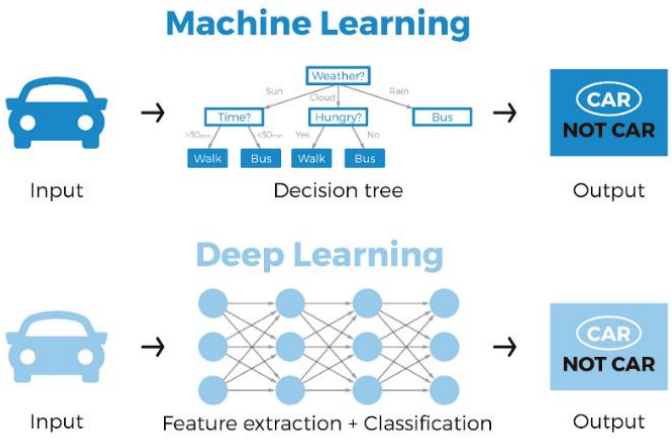

Figura 4.4: Diferencia entre ML y DL

4.4. Aprendizaje supervisado y no supervisado

Como se ha comentado en el punto anterior, dentro del aprendizaje automático existen, entre otros, dos tipos principales de algoritmos: el aprendizaje supervisado y el no supervisado. El supervisado –que será el tipo de aprendizaje que se utilizará en este proyecto– se diferencia del no supervisado por tener unos datos ya etiquetados que sirven de ejemplo para el modelo. De este modo, partiendo de unos datos de entrenamiento (pares de objetos formados por datos de entrada y su correspondiente etiqueta), el modelo debe crear una función que pueda predecir el resultado correcto para cualquier entrada. Es decir, debe ser capaz de generalizar a partir de los ejemplos para poder trabajar con entradas no vistas anteriormente [33].

En el caso del aprendizaje no supervisado no existe un conocimiento a priori, es decir, no existen datos previamente etiquetados a partir de los cuales pueda aprender el modelo. Como consecuencia, el modelo no puede predecir las etiquetas de los datos que se le pasen como entrada, pero sí puede agrupar estos datos en función de sus características, obteniendo conjuntos de datos que, según el modelo, forman parte de la misma categoría [29] [34].

En conclusión, el aprendizaje supervisado necesita datos etiquetados correctamente por humanos para poder predecir la etiqueta de nuevos datos sin ayuda humana, mientras que el no supervisado agrupa datos que pertenecen a un mismo conjunto pero necesita de la intervención humana para dar una etiqueta a estos grupos.

4.5. Perceptrón multicapa

El perceptrón multicapa es un tipo de red neuronal artificial con la capacidad de resolver problemas que no son linealmente separables (a diferencia del perceptrón simple). Puede estar totalmente o localmente conectado, y utiliza el algoritmo de propagación hacia atrás para su entrenamiento.

La propagación hacia atrás o retropropagación del error (en inglés, *backpropagation*) es un algoritmo que consiste en, después de recorrer la red en sentido normal (hacia delante) y comprobar cuánta diferencia hay entre la salida obtenida y la deseada con una función de coste (MSE, entropía cruzada, etc.), recorrer la red en sentido inverso (hacia atrás) mientras ajusta los parámetros del modelo. Por tanto, *backpropagation* pretende minimizar la función de coste mediante el reajuste de los pesos y ganancia de la red [35].

En muchas ocasiones, como en el caso del presente proyecto, se utiliza el perceptrón multicapa junto con redes neuronales convolucionales (de las que se hablará en el siguiente apartado), colocando el perceptrón (capas totalmente conectadas) a continuación de las capas convolucionales [36].

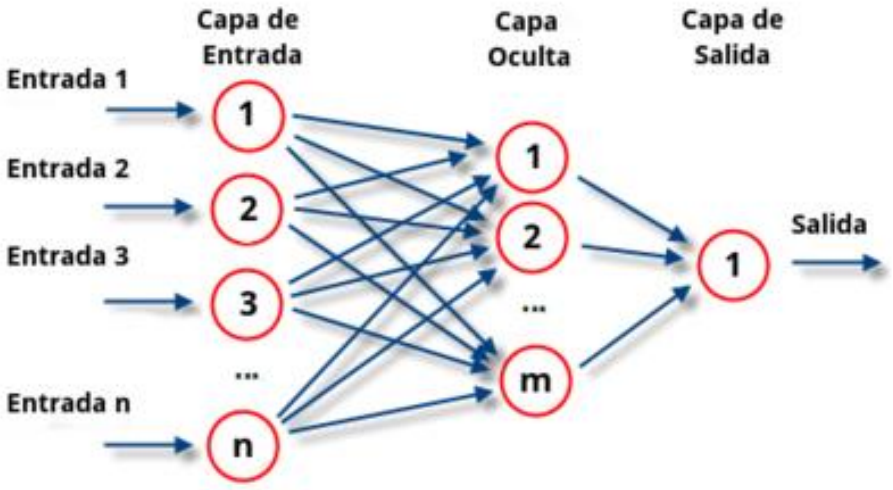

Figura 4.5: Estructura de un perceptrón multicapa

4.6. Redes Neuronales Convolucionales (CNN)

Para poder comprender las redes neuronales convolucionales, primero hay que entender qué son las redes neuronales artificiales. Las redes neuronales artificiales son un modelo computacional que pretende imitar el comportamiento de las redes de neuronas que se encuentran en los cerebros biológicos [37].

Las redes neuronales artificiales son, como su nombre indica, un conjunto de neuronas artificiales conectadas entre sí que transmiten señales. Una neurona artificial no es más que una unidad de cálculo que imita el comportamiento de una neurona natural, es decir, hace una suma ponderada de las entradas que recibe, aplica una función no lineal (también llamada función de activación) y devuelve una salida [38]. La red está formada por diversas capas: una primera capa, llamada **capa de entrada**, donde recibe los datos de entrada; una o más **capas ocultas**; y una última capa, conocida como **capa de salida**, donde se obtienen los resultados. Una neurona recibe las entradas que le envían las neuronas conectadas de la capa anterior (o los datos de entrada originales si se trata de la capa de entrada) multiplicadas por el valor del peso del enlace correspondiente, ejecuta los cálculos explicados anteriormente (sumatorio y función de activación) y propaga el resultado o señal a las neuronas adyacentes de la siguiente capa [37].

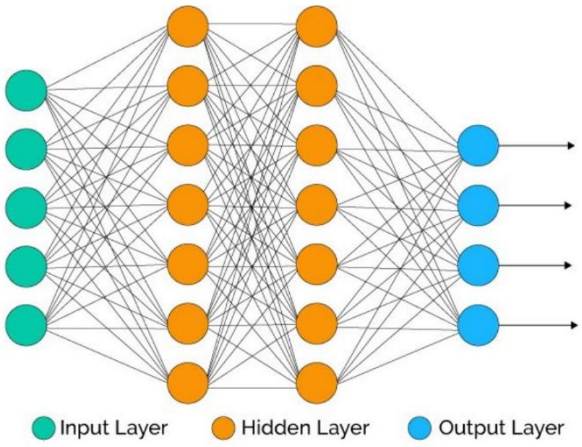

Figura 4.6: Red neuronal artificial

Las redes neuronales artificiales se utilizan en Deep Learning por la facilidad de estos modelos para aprender por su propia cuenta. De manera simple, la red neuronal irá modificando el valor de sus pesos en los enlaces hasta que los valores de salida de la última capa sean lo más adecuados posibles [39].

Las redes neuronales convolucionales o CNN por sus siglas en inglés (Convolutional Neural Network) son un tipo de redes neuronales artificiales ampliamente utilizadas en análisis de imagen, debido principalmente a que su estructura formada por capas convolucionales reduce la

alta dimensionalidad de las imágenes sin perder información. Las CNN reciben su nombre por el tipo de operación matemática que utilizan: las convoluciones.

Dados unos datos de entrada –en nuestro caso, los píxeles de una imagen en forma de matriz –, la convolución consistirá en ir aplicando un filtro (kernel) en submatrices de dicha matriz. El kernel podrá ser de mayor o menor dimensiones, y se podrá desplazar por la matriz de los datos de entrada con saltos de mayor o menor tamaño (*stride*). Para cada submatriz, se multiplicarán los valores de esta por los valores del kernel y se hará el sumatorio con los valores obtenidos. Por ejemplo, podremos tener una matriz original de 7x7 y un kernel de 3x3 que se desplace con saltos de 1 píxel, por lo que el resultado final de la convolución tendrá una dimensión de 5x5, tal y como se muestra en el ejemplo de la figura 4.7.

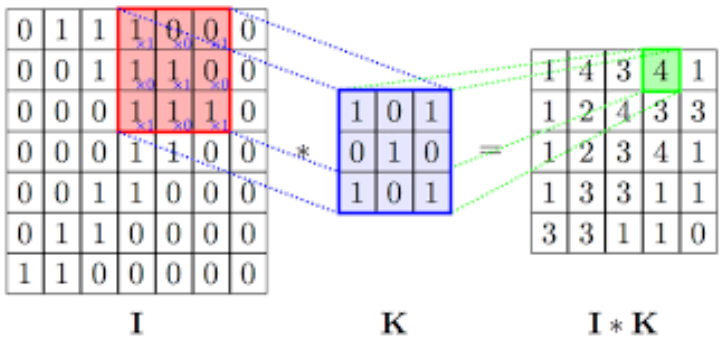

Figura 4.7: Proceso de convolución

Después de esto se aplica el *pooling*, que es una especie de agrupación sobre los elementos de la matriz resultante, de forma similar a lo que haría una convolución pero con la diferencia de que sólo puede escoger el máximo o la media para cada agrupamiento. Del mismo modo que el kernel, la matriz del pooling podrá tener diversas dimensiones y avanzar con distintos pasos. Por último, es necesario una o varias capas totalmente conectadas (*fully-connected layers*), es decir, capas que junten todas las partes que se han ido calculando en la capa precedente. Como resulta evidente, esto reduce significativamente el tamaño de la red, pues pasa de conectar cada píxel de la imagen con todas las neuronas de la capa adyacente a conectar tan solo un reducido número. Si las imágenes de entrada son de mayor dimensión y, por ende, necesitan una reducción mayor, es posible conectar varias capas convolucionales y de *pooling* antes de pasar a la capa totalmente conectada [40]. Un ejemplo de la estructura de una CNN se puede observar en la figura 4.8.

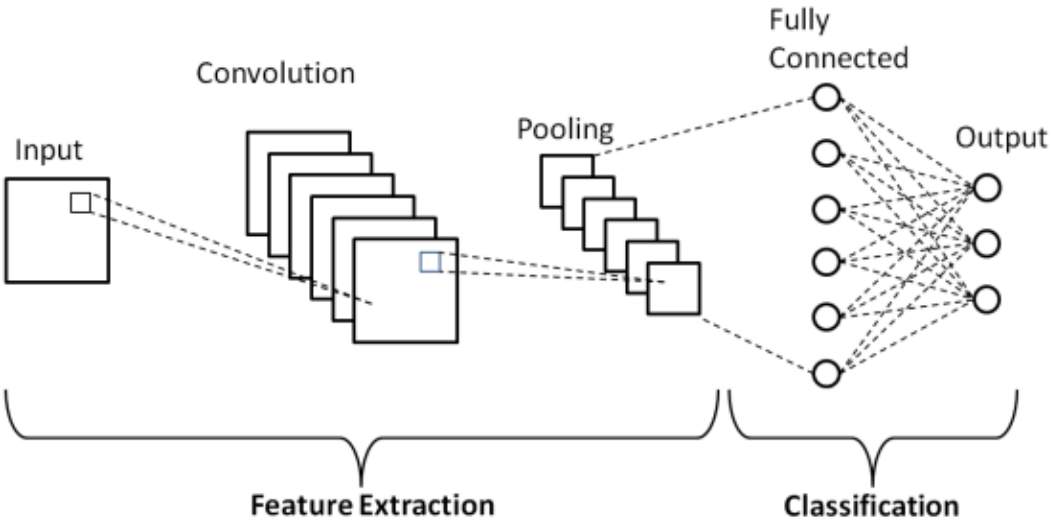

Figura 4.8: Estructura de una red neuronal convolucional

5. Tecnología

En este capítulo se describen las diferentes tecnologías que han sido utilizadas durante el desarrollo del software de reconocimiento de imágenes. Se exponen, por tanto, las librerías de Python que han sido empleadas en el desarrollo del código, con especial importancia para TensorFlow y Keras. Se habla también de los entornos de trabajo escogidos para ser utilizados durante todo el proceso de desarrollo.

Por otro lado, se profundiza en las características y detalles de la cámara a partir de la cual se obtiene el vídeo en tiempo real sobre el que trabaja el software, así como de su ubicación y protección. Además, se muestra también el proceso por el cual se adquieren los datos, desde que se obtiene el vídeo de la cámara hasta que se almacena el dato en una o más bases de datos, y las diferentes aplicaciones y sistemas que se utilizan en dicho proceso.

5.1. Librerías

5.1.1. NumPy

NumPy es una biblioteca de código abierto para Python que da soporte a operaciones con vectores y matrices multidimensionales, y ofrece una amplia gama de funciones matemáticas de alto nivel [41].

5.1.2. Matplotlib

Matplotlib es una biblioteca para la creación de representaciones gráficas estáticas, animadas e interactivas en Python. Con ella se pueden crear gráficos de calidad, figuras interactivas (zoom, panorámica, actualización), personalizar el estilo visual o exportar en archivos en distintos formatos, entre otras funciones [42].

5.1.3. TensorFlow

TensorFlow es una biblioteca de código abierto dedicada al aprendizaje automático que permite crear y entrenar redes neuronales [43]. Fue creada por Google para su uso interno y en 2015 fue publicada bajo licencia de código abierto Apache 2.0 [44].

Como su nombre indica, la biblioteca hace uso de tensores (objetos matemáticos de diferentes dimensiones que almacena valores numéricos), y es utilizada para el reconocimiento de imágenes, el análisis de sentimiento y para diagnósticos médicos, entre otros [43].

5.1.4. Keras

Keras es una biblioteca de código abierto ejecutable sobre TensorFlow y otros *frameworks*. Está diseñada para trabajar con redes neuronales y experimentar en poco tiempo haciendo uso del Deep Learning. Su popularidad se debe, además de a la posibilidad de utilizarla con TensorFlow, a ser amigable para el usuario, modular y extensible [45].

5.1.5. OpenCV

OpenCV (*Open Computer Vision*, Visión por Computador Abierta) es una biblioteca libre para la visión artificial desarrollada por Intel. Entre sus características se encuentra el estar publicada bajo licencia BSD, lo que permite su uso para propósitos comerciales y de investigación, o ser

multiplataforma, pudiéndose utilizar en GNU/Linux, Mac OS X, Windows y Android, además de arquitecturas de hardware como x86, x64 o ARM [46].

Gracias a todas estas propiedades, hoy en día es una de las bibliotecas más populares en el ámbito de la visión por computador, a más de 20 años de su lanzamiento (1999) [47].

5.1.6. Pytesseract

Pytesseract es un *wrapper* (función contenedora) del motor Tesseract-OCR de Google, que se utiliza como herramienta en Python para el reconocimiento óptico de caracteres (OCR) [48]. Esta librería permite obtener el texto presente en una imagen, y puede ser empleada, por ejemplo, en tareas como la lectura automática de matrículas.

5.1.7. ALPR

ALPR (*Automatic License Plate Recognition*, Reconocimiento Automático de Matrícula) es una biblioteca disponible en Python que permite la detección y lectura automática de la matrícula de un vehículo a partir de una imagen.

5.2. Entornos de trabajo

5.2.1. Anaconda

Se ha decidido utilizar Anaconda como entorno de trabajo para facilitar la instalación y uso de las diversas librerías ya mencionadas, puesto que en muchos casos era necesario crear diferentes entornos con diferentes versiones de cada una de ellas. Además, se ha optado por utilizar Jupyter Notebook (disponible en Anaconda) como interfaz sobre la que desarrollar el software debido a su facilidad de uso y eficiencia.

Anaconda permite crear diferentes entornos en los que instalar la colección de paquetes necesaria. Estos entornos son independientes, es decir, se puede trabajar con distintas versiones de un mismo programa en cada uno de ellos. Por ejemplo, si para un proyecto es necesaria la versión 1.0 de un determinado programa, y para otro la versión 2.0, podemos crear dos entornos e instalar en cada uno la versión correspondiente, permitiendo así el desarrollo de ambos proyectos desde un mismo ordenador.

Por otra parte, el navegador de Anaconda incluye una amplia gama de programas con los que trabajar, desde JupyterLab y el ya citado Jupyter Notebook hasta RStudio (para el desarrollo de código en el lenguaje de programación R), entre otros.

5.2.2. Jupyter Notebook

Se ha elegido Jupyter Notebook como entorno sobre el que desarrollar el código (en lenguaje Python) del proyecto. Forma parte del Proyecto Jupyter, una organización creada para desarrollar software de código abierto, estándares abiertos y para servicios para computación interactiva, y que soporta una amplia variedad de lenguajes [49].

Jupyter Notebook funciona a modo de cuaderno interactivo en el que podemos escribir y ejecutar partes específicas del código (celdas), evitando así tener que ejecutar el programa completo cada vez que se modifica una parte de éste. Esta característica resulta muy ventajosa en programas en los que se efectúa un proceso costoso en una parte concreta del código, pues permite ejecutar dicha parte una sola vez. Por ejemplo, en el caso del software de reconocimiento de imágenes, el proceso de preparación de los datos de entrenamiento requiere una mayor cantidad de tiempo que el resto del código, por lo que resulta útil poder ejecutar este proceso una única vez y posteriormente hacer pruebas en otras partes del código.

Además, Jupyter Notebook también posibilita la exposición del código de una manera más visual y explicativa, con la posibilidad de añadir textos a modo de comentarios o aclaraciones y guardando en el cuaderno los gráficos, tablas, figuras, etc. creados durante la ejecución.

5.3. Cámara

Con el fin de obtener vídeos en tiempo real, se ha instalado una cámara en el carril 2 de salida del Acceso Sur del Puerto de Valencia que proporcione las imágenes sobre las que posteriormente se aplicará el software de reconocimiento de imágenes.

Esta cámara ha sido instalada en una posición y con una perspectiva que le permite capturar imágenes donde se aprecia ampliamente el lateral de los vehículos. Esto se debe a que, para la tarea de diferenciar distintos tipos de vehículos, y en particular distintos tipos de camiones, es fundamental una buena visión de la parte lateral/trasera de éstos, pues es donde residen las mayores diferencias entre ellos.

En la figura 5.1. aparecen 3 tipos de camiones: portacontenedores (blanco), plataforma (rojo) y tolva (amarillo). Se puede observar cómo estos 3 vehículos son relativamente similares vistos desde delante (imágenes de la derecha), lo que dificultaría en gran medida diferenciarlos a partir de esta perspectiva. Sin embargo, su parte lateral (imágenes de la izquierda) es la que aporta los rasgos más representativos, haciendo más fácil la distinción para el modelo.

Figura 5.1: Vista lateral y frontal de los vehículos

5.3.1. Cámara y características

La cámara instalada es una BOSCH NBN-80052-BA[1], perteneciente a la familia de las DINION IP starlight 8000 MP. Esta cámara está diseñada para proporcionar vídeo IP de calidad las 24 horas del día, con independencia de las condiciones de iluminación, la hora del día o el movimiento de los objetos.

Figura 5.2: Cámara BOSCH NBN-80052-BA

Entre las funcionalidades de esta cámara, además de las ya citadas como el buen rendimiento con baja iluminación, se encuentran:

- Intelligent Video Analytics: comportamiento inteligente de la cámara que proporciona reducción de falsas alarmas, identificación de intervalos ampliados, gestión de multitudes y de colas, y recuento de densidad y flujo.
- Intelligent Auto Exposure: ajuste automático de la exposición de la cámara para evitar que la contraluz y las condiciones cambiantes de iluminación arruinen las imágenes.
- Intelligent Dynamic Noise Reduction: reducción de la tasa de bits al 50%, es decir, de los costes de almacenamiento y carga de red usando sólo el ancho de banda cuando es necesario.
- Resolución y relación de aspecto seleccionables: permite elegir entre 5 MP (16:9), 5.5 MP (4:3) y 1080p.
- Seguridad de los datos: ofrece un máximo nivel de seguridad para el acceso a los dispositivos y para el transporte de datos, con contraseñas de diversos niveles y cortafuegos para la protección contra ataques malintencionados.

[1] Hoja de datos: https://resources-boschsecurity-cdn.azureedge.net/public/documents/NBN_80052_Data_sheet_esES_13616032779.pdf

- Software de visualización completa: acceso a las funciones de la cámara desde el navegador web, el Bosch Video Management System, el sistema Bosch Video Client o desde software de terceros.

5.3.2. Ubicación

La cámara ha sido instalada con el fin de obtener imágenes de los vehículos que transitan el carril 2 de salida. Sin embargo, para conseguir una buena perspectiva de dichos vehículos se ha optado por instalarla en el carril 1 (sin circulación), en la zona peatonal más alejada (figura 5.3).

Se ha orientado de tal forma que permita observar a la parte trasera y lateral de los vehículos cuando estos avanzan en dirección de salida, y se ha utilizado el zoom propio de la cámara para focalizar la imagen en el vehículo a identificar, extrayendo así la mayor parte de ruido de fondo que pueda afectar negativamente al algoritmo.

Figura 5.3: Vista aérea de los carriles del Acceso Sur del Puerto de Valencia

5.3.3. Protección de la cámara

La cámara BOSCH NBN-80052-BA está diseñada para funcionar en diversas condiciones de temperatura (desde los -20 ºC a los 50 ºC) y de humedad (del 20% al 93% de humedad relativa), como indica en su hoja de datos. Sin embargo, con el fin de proteger una cámara de tanto valor (alrededor de los 1.500 €) ante posibles amenazas, medioambientales y no medioambientales, se ha optado por colocarla dentro de una carcasa tipo box.

En concreto, la carcasa seleccionada es una UHO de exterior para cámaras[2], también del fabricante BOSCH. Esta carcasa cuenta con dos calentadores y un ventilador para mantener la temperatura regulada en el interior de la carcasa y desempañar la ventana, con un parasol incluido y con tornillos a prueba de sabotajes para garantizar la seguridad antes accesos no autorizados, entre otras funcionalidades.

Figura 5.4: Carcasa UHO de exterior para cámaras

[2] Hoja de datos: https://resources-boschsecurity-cdn.azureedge.net/public/documents/UHO_x1_Data_sheet_esES_17707773579.pdf

5.4. Conectividad y base de datos

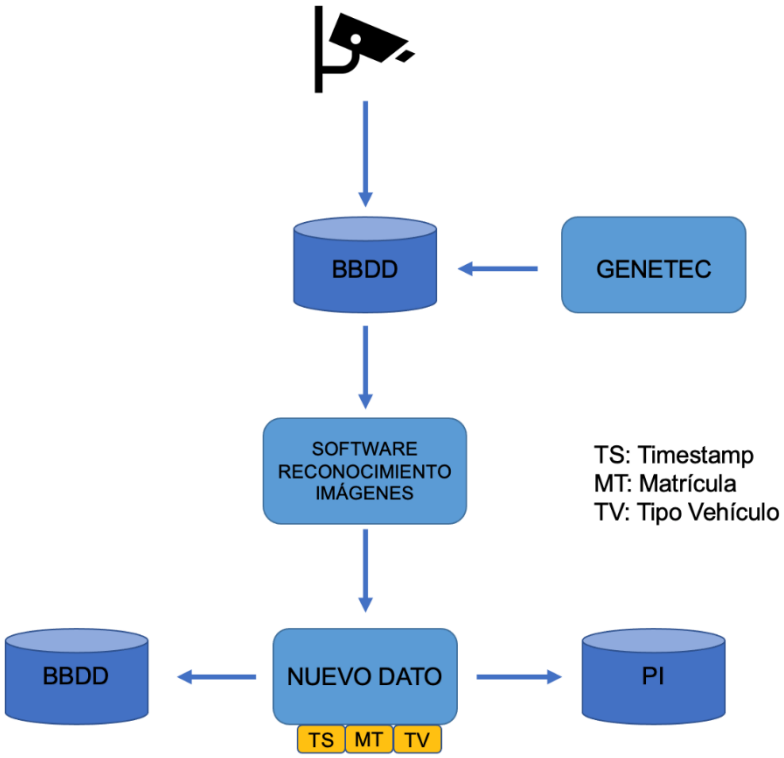

Figura 5.5: Proceso de obtención de los datos

En la figura 5.5 se observa el proceso que abarca desde la obtención del vídeo por la cámara del carril 2 hasta la generación de nuevos datos que son almacenados en bases de datos. En primer lugar, se transmite el vídeo en tiempo real que está obteniendo la cámara hasta una base de datos, a la cual se accederá a través de un programa llamado Genetec[3].

El software de reconocimiento de imágenes obtiene en tiempo real los vídeos de esta base de datos, genera nuevos datos formados por la fecha, la matrícula y el tipo del vehículo, y transmite estos nuevos datos a otra base de datos.

Adicionalmente, los datos se pueden pasar a una segunda base de datos llamada PI[4], en la que se almacenan grandes cantidades de datos relativos al Puerto de Valencia, para que puedan ser integrados y utilizados en diversas áreas.

[3] *Genetec* es un software de seguridad electrónica con gestión de vídeo, control de accesos y lectura de matrículas, entre otras funcionalidades [53].

[4] PI System es un software de Osisoft para el almacenamiento y manejo de datos.

5.4.1. Arquitectura tecnológica

El primer paso del proceso descrito anteriormente es el de transmitir el vídeo que está capturando la cámara en tiempo real a una base de datos. El vídeo que se obtiene es reproducido a través del programa Genetec, y para ello es transportado desde la cámara hasta Genetec mediante un cable de señal de tipo UTP Categoría 6[5] con cubierta para exteriores. La alimentación es a través de Ethernet (PoE), es decir, viene por el mismo cable de datos, desde un switch cuyas bocas, además de datos, son capaces de suministrar alimentación (estándar IEEE 802.3af/at).

La transmisión se realiza mediante la integración de la cámara con Genetec. Una vez integrada la cámara en el sistema (con IP, usuario y contraseña) se establece una conexión RTSP[6] segura entre la cámara y el sistema, para que después, a través de distintos medios, la imagen sea tratada y su visionado controlado. En el caso particular de la Autoridad Portuaria de Valencia, para la transmisión de las imágenes, tanto cámara como servidor están en la misma VLAN[7].

5.4.2. Base de datos

Los vídeos, recogidos por las cámaras integradas con Genetec y transportados a través de la arquitectura mencionada en el punto 5.4.1., se almacenan en una base de datos durante 30 días (debido a la Ley de Protección de Datos) [50]. En concreto, esta base de datos se encuentra en los servidores de Genetec, donde se guarda, además del propio vídeo, los metadatos que va generando la cámara: posicionamiento PTZ (Panorámica, Inclinación y Zoom), analíticas configuradas, etc.

La base de datos está basada en SQL Server de Microsoft, y por tanto, es de tipo relacional, con una arquitectura basada en una sola tabla en la base de datos con todas las categorías de datos que puede proporcionar Genetec.

5.4.3. Implantación con el software

Para la implantación final del software de reconocimiento de imágenes en un entorno real, el programa deberá recibir el vídeo en *streaming* de la cámara y a partir de él obtener la tipología de los vehículos que circulen por el carril 2 en ese momento (además de otros datos como la matrícula y la fecha).

---

[5] Cable de par trenzado estandarizado para Ethernet con capacidad para trabajar con velocidades de hasta 1 Gigabit por segundo [54].

[6] *Real Time Streaming Protocol*, protocolo de transmisión en tiempo real para flujos de datos (audio o vídeo) [55].

[7] *Virtual Local Area Network*, red de área local virtual que sirve para, dentro de una misma red física, tener varias redes lógicas independientes [56].

La manera habitual de Genetec para pasar imágenes a sistemas software es vía una conexión RTSP. Esto se hace a través del servicio propio de Genetec, llamado Media Gateway[8]. Se suele facilitar una URL (una por cada cámara que se quiera ver), usuario y contraseña mediante los cuales permite recibir las imágenes de las cámaras deseadas. Actualmente la calidad que se transmite es VGA (640x480).

[8] Los *media gateways* son servicios de traducción que convierten flujos multimedia entre tecnologías de telecomunicaciones dispares [57].

6. Planteamiento inicial

6.1. Introducción

Este trabajo surge de la necesidad de cambio dentro de la Autoridad Portuaria de Valencia en el ámbito de la gestión de los datos. Uno de los objetivos de esta reestructuración es la extracción de datos de utilidad y su correspondiente tratamiento, y para ello se pretende ser capaz de obtener información sobre los vehículos en su entrada/salida del recinto. Esta información abarca la fecha y hora de entrada/salida, el número de matrícula o la tipología del vehículo, entre otros datos.

Puesto que no sólo se busca la adquisición de datos sino también su análisis e integración en un ámbito mayor, de tal forma que datos diversos queden dotados de valor informativo, se pretende diseñar un software de reconocimiento de imágenes que pueda predecir la tipología de los vehículos a partir de las imágenes obtenidas en las cámaras de vigilancia, y tratar este dato de la manera correcta para su uso posterior.

6.2. Trabajo de campo

El primer paso para comenzar este proyecto era realizar un trabajo de campo que permitiera obtener la información necesaria sobre las cámaras de seguridad ya instaladas, como su localización, su rango de imagen, sus funcionalidades, etc. Para que el software de reconocimiento de imágenes funcione correctamente y pueda identificar a los vehículos por su tipología, es necesario que la posición, ángulo y alcance de la cámara permitan apreciar el vehículo en su totalidad o, al menos, en gran medida. Además, para poder leer la matrícula a través de la misma cámara, se necesita también una buena calidad de imagen, ya que de lo contrario la matrícula sería ilegible incluso para el ojo humano.

Por tanto, se empezó a trabajar en colaboración con el Departamento de Tecnologías de la Información y Red Industrial, y se obtuvo información acerca de los carriles de entrada y salida del Acceso Sur del Puerto de Valencia y de sus respectivas cámaras. Finalmente, se decidió instalar una nueva cámara en un carril sin una propia (carril 2 de salida), sobre el cual se implementaría el software.

La elección de este carril se debe, principalmente, a la escasez de información que se obtiene del mismo. Se trata de un carril por el que salen, además de la mayoría de turismos, motocicletas y furgonetas, los vehículos pesados una vez han terminado de descargar su mercancía. Por esta razón, en este carril se pueden observar con elevada frecuencia camiones plataforma, portacoches, cisterna, portacontenedores o tolva, por ejemplo (todos ellos vacíos en su interior).

Sin embargo, la mayor parte de los datos se recogen de los carriles colindantes al carril 2, debido a que por ellos discurren los vehículos cargados, y este último queda prácticamente sin registro. La necesidad de obtener más información acerca de los vehículos que transitan el carril 2, unida a la gran diversidad de vehículos que circulan por él, han sido las claves de la decantación por este carril para la implantación del software de reconocimiento de imágenes.

7. Desarrollo

7.1. Banco de imágenes

Una vez realizado el trabajo de campo y resueltas todas las cuestiones necesarias sobre las cámaras (localización, software, gestión de los datos, etc.), se comenzó con el desarrollo del modelo. En primer lugar, era necesario crear un banco de imágenes lo suficientemente grande para entrenar el modelo, para lo cual se utilizó la aplicación *Genetec* con el fin de recoger los vídeos que posteriormente serán utilizados en el software de recuperación de imágenes que se tratará a continuación. Dichos vídeos se almacenan en un servidor a demanda, es decir, son solicitados a los responsables de la empresa y éstos los almacenan en dicho servidor para ser utilizados por el programa de recuperación de imágenes.

El funcionamiento del programa de recuperación de imágenes se puede resumir en tres simples pasos. En primer lugar, definimos cuántos frames de cada segundo del vídeo queremos obtener. Normalmente, los vídeos tienen una cantidad de frames por segundo (FPS) que va desde los 20 fps a los 60 fps, pero en nuestro caso no nos interesa obtener una cantidad tan grande de frames por cada segundo, pues la imagen del vehículo suele ser muy similar en todos los frames de un mismo segundo. Es por esto por lo que parametrizamos en uno (1) como la cantidad deseada de frames a obtener de cada segundo.

En segundo lugar, escribimos el lugar de origen del vídeo (es decir, la ruta de la carpeta donde se encuentran almacenado el vídeo) y el lugar destino donde guardar las imágenes obtenidas, esto es, la ruta de la carpeta donde se almacenarán.

En tercer y último lugar, el código entra en un bucle en el que se va adquiriendo cada uno de los frames de cada segundo del vídeo, pero por cada segundo sólo se almacena la cantidad indicada anteriormente. Esto se consigue a partir de un contador (*FPScount*) que va desde 0 hasta el resultado de dividir el número original de frames del vídeo entre la cantidad deseada, menos uno. En el ejemplo de la figura 7.1, el vídeo original tiene 60 fps y queremos obtener 1 frame por cada segundo, por lo que el contador irá de 0 hasta 59, llegado a este valor guardará un frame y volverá a 0, o lo que es lo mismo, guardará 1 frame de cada 60.

Adicionalmente, se ha añadido la funcionalidad de recortar las imágenes antes de que sean almacenadas. Esto se debe a que, en la mayoría de las ocasiones, el vehículo se encuentra en el centro de la imagen, mientras que alrededor aparecen otros objetos (ruido) que pueden confundir al reconocedor. Para mejorar la capacidad de aprendizaje del software, y teniendo en cuenta que la posición de los vehículos en la imagen siempre es la misma, se ha implementado esta herramienta para que no sea necesario ampliar manualmente el zoom de la cámara o desarrollar un detector de objetos.

Este código está inspirado en el ejemplo disponible en https://www.geeksforgeeks.org/python-program-extract-frames-using-opencv/ . En él se obtienen todos los frames de cada segundo y se almacenan en una carpeta por defecto (la carpeta en la que esté situada el código fuente). En nuestro caso, es preferible que de cada segundo se obtenga solamente 1 fotograma, que éstos se almacenen en una ubicación específica y, además, que antes de guardar la imagen sea recortada, por lo que se han añadido estás modificaciones al código original.

El motivo de utilizar un programa a medida para recuperar las imágenes de vídeo se debe a que la ejecución manual hubiera sido un trabajo excesivamente arduo, por lo que se buscó posibles maneras de automatizarlo y se decidió utilizar dicho software que separara los vídeos de las grabaciones de las cámaras en imágenes de forma automática, dejando únicamente al programador la tarea de clasificar cada imagen en su debida clase, es decir, cada vehículo en su debido tipo.

Para completar el banco de imágenes extraído con el software comentado anteriormente, se han explorado otras fuentes posibles de imágenes, como bancos de imágenes pertenecientes a otros proyectos de la empresa o disponibles en Internet, con el fin de poder ampliar lo máximo posible el banco de imágenes.

Al término del desarrollo del software, se ha conseguido un banco con 8.428 imágenes repartidas entre los 10 tipos de vehículo que es capaz de clasificar:

- Camión portacontenedor (2806 imágenes)
- Camión plataforma (1625 imágenes)
- Camión tolva (89 imágenes)
- Camión cisterna (163 imágenes)
- Camión portacoches (193 imágenes)
- Furgoneta (350 imágenes)
- Turismo (1642 imágenes)
- Motocicleta (861 imágenes)
- Cabeza tractora (174 imágenes)
- Otros (525 imágenes)

De este conjunto de imágenes, un 23.3% son imágenes extraídas de bancos externos[9] obtenidos de Internet con el fin de ampliar y diversificar el banco de imágenes. El 76.4% restante son imágenes obtenidas de los vídeos de varias cámaras situadas dentro de la Autoridad Portuaria de Valencia, haciendo uso del software de recuperación de imágenes.

Comúnmente, cuando se trabaja con redes neuronales artificiales en clasificación de imágenes se obtienen mejores resultados si el conjunto de entrenamiento está equilibrado, esto es, si todas las clases a clasificar disponen de un número igual o similar de imágenes con las que ser entrenadas. Este no es el caso del banco de imágenes mostrado anteriormente, ya que hay

[9] https://www.kaggle.com/datasets/brsdincer/vehicle-detection-image-set?select=data
https://zenodo.org/record/7695824#.ZAmwknbMKUk
https://www.kaggle.com/datasets/prasunroy/natural-images?resource=download

vehículos que transitan con más frecuencia que otros por los carriles que se han utilizado para obtener las imágenes, y del mismo modo, hay vehículos para los que existe una mayor cantidad de imágenes en Internet que para otros.

Para solucionar la problemática de tener clases con cantidades de imágenes tan desiguales, se ha optado por utilizar la técnica de *weighted classes* (clases con peso), la cual consiste en dar mayor importancia a las clases de menor tamaño con respecto a las de mayor durante el entrenamiento (mediante la función *loss*).

Si, por ejemplo, tenemos la clase A con 1 muestra de entrenamiento y la clase B con 10, esta técnica equivaldría a enseñar al modelo que 1 muestra de la clase A se corresponde con 10 de la clase B. De esta forma, las clases minoritarias reciben mayor importancia que las mayoritarias, compensando así la diferencia de tamaños.

A continuación, se puede observar la figura 7.1 con el código del programa de recuperación de imágenes. Enmarcadas en un rectángulo rojo encontramos las modificaciones realizadas sobre el código original, entre las que se encuentran la obtención de un determinado número de frames por segundo (1), el recorte de la imagen previo a su almacenamiento y su depósito en la carpeta especificada.

```python
# Programa para leer un vídeo y extraer sus frames
# Basado en: https://www.geeksforgeeks.org/python-program-extract-frames-using-opencv/

import cv2
import os

# Frames por segundo del vídeo original
realFPS = 15

# Frames que queremos obtener por cada segundo
wantedFPS = 1

FPS = realFPS/wantedFPS - 1

# Función para la extracción de frames
# pathVideo: ruta del vídeo original
# pathFolder: ruta de la carpeta donde se quiere guardar los frames
def FrameCapture(pathVideo, pathFolder):

    # Ruta del vídeo
    vidObj = cv2.VideoCapture(pathVideo)

    # Contador (para indicar el número del frame)
    count = 0

    # Contador de FPS (para obtener la cantidad deseada de frames por cada segundo)
    FPScount = 0

    # comporbar si se ha extraído un frame
    success = 1

    while success:

        # Llamada a la función read() para extraer los frames
        success, image = vidObj.read()

        # Entrará en función de los frames que queramos obtener de cada segundo
        if FPScount == FPS:
            # Recorta la imagen para quedarnos con el vehículo
            h, w, channels = image.shape
            y = int(h/6)
            Y = int ((3*h)/4)
            x = int (w/4)
            X = int ((3*w)/4)
            image = image[y:Y,x:X]
            # Guarda el frame con su contador
            cv2.imwrite(os.path.join(pathFolder,"300323_13h00_frame%d.jpg" % count), image)
            count += 1
            FPScount = 0
        else:
            FPScount += 1

# Driver Code
if __name__ == '__main__':

    # Llamada a la función
    FrameCapture(r"C:\Users\dsegura\Desktop\pruebaBancoImagenes\videosFuente\300323_13h00.mp4",
                 r"C:\Users\dsegura\Desktop\pruebaBancoImagenes\imagenes")
```

Definición de frames a obtener por cada segundo

Recorte de la imagen

Almacenamiento en carpeta específica

Figura 7.1: Software de extracción de frames

7.2. Desarrollo de la red neuronal

El banco de imágenes fue incrementándose continuamente, y de forma paralela se empezó el desarrollo del código para el software de reconocimiento de imágenes. En un primer instante, se diseñó una red neuronal basada en el código del tutorial *Deep Learning basics with Python, TensorFlow and Keras*[10], disponible en Internet.

Posteriormente, y a medida que el banco de imágenes iba adoptando su tamaño final, se hicieron pruebas con dos arquitecturas ampliamente conocidas en el campo del reconocimiento de imágenes: LeNet-5 y AlexNet. A partir de los resultados obtenidos para estas dos arquitecturas, se fueron implementando sobre la arquitectura original diseñada al comienzo las características propias de LeNet y AlexNet que ofrecían un mejor rendimiento. De esta forma se obtuvo la arquitectura final con la que se ha implementado el software de reconocimiento de imágenes, y que se puede observar en el apartado 7.2.4.

7.2.1. Arquitectura original

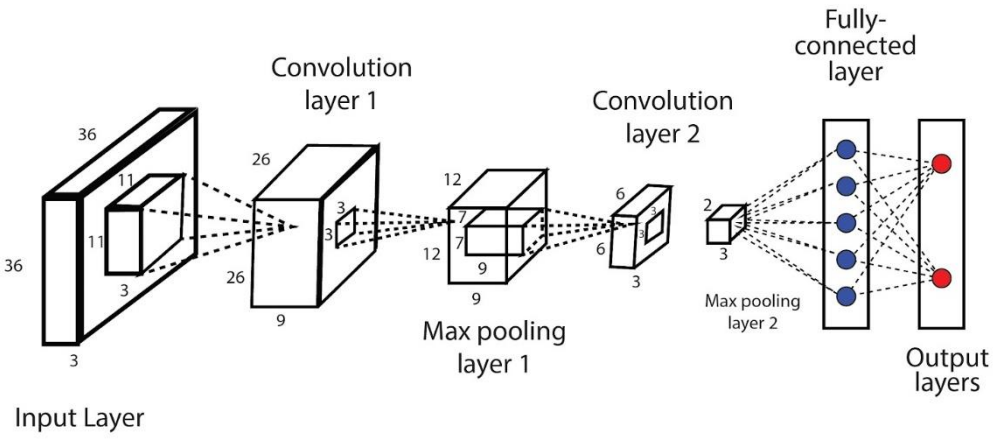

Figura 7.2: Arquitectura de la red neuronal original

Este modelo original estaba entrenado con un banco de imágenes de tamaño muy reducido, y sólo era capaz de distinguir dos tipos de vehículos. En concreto, únicamente podía diferenciar

[10] https://pythonprogramming.net/introduction-deep-learning-python-tensorflow-keras/

entre camiones portacontenedores y turismos, vehículos muy diferentes entre sí. La arquitectura de este primer modelo era muy básica, similar a la arquitectura LeNet (explicada en el siguiente apartado), formada por dos capas de convolución[11], dos capas de *pooling*[11], una capa de *flatten*[11] y dos capas finales totalmente conectadas[11] (CONV-POOL-CONV-POOL-FL-FC-FC), similar a la red de la figura 7.2.

A medida que el banco de imágenes se ampliaba, se fueron haciendo pruebas para diferenciar un mayor número de tipos con la misma arquitectura (aunque cambiando la función de activación[11] de la última capa de *binary*[11] a *softmax*[11], para clasificar más de dos clases). El mejor resultado que se obtuvo fue de 95.92% de acierto (en función del valor de *validation accuracy*[11]), aunque con un número reducido de clases. Una vez alcanzado un número considerable de imágenes de entrenamiento, se empezó a probar modelos basados en las arquitecturas de redes neuronales convolucionales más populares dentro del campo de la visión por computador y el reconocimiento de imágenes.

Número de clases	Batch Size	Learning Rate	Epochs	Validation Split	Validation Loss	Validation Accuracy
5	32	0.001	3	0.3	0.6329	0.7701
5	32	0.001	3	0.3	0.5224	0.8156
6	32	0.001	10	0.3	0.5189	0.8529
5	32	0.001	10	0.3	0.3253	0.8816
4	32	0.001	5	0.3	0.1361	0.9592

Tabla 7.1: Tabla de resultados de la red neuronal original

[11] Ver glosario de términos

7.2.2. LeNet-5

La arquitectura LeNet-5, o simplemente LeNet, fue desarrollada por Yann LeCun y su equipo en 1998, y fue una de las primeras arquitecturas que utilizó el modelo de redes neuronales convolucionales tal y como las conocemos hoy en día.

Originariamente fue diseñada y empleada para el problema de reconocimiento de dígitos manuscritos (OCR) a partir del conjunto de datos MNIST, con el que alcanzó un acierto de clasificación del 99.2% [8].

La arquitectura LeNet (figura 7.3) está formada por dos capas convolucionales, dos capas de *pooling*, una capa de *flatten*, dos capas totalmente conectadas y una última capa de salida. Como se habrá podido observar, es muy similar a la arquitectura utilizada en las primeras pruebas de este proyecto; sin embargo, las principales diferencias se encuentran en los parámetros con los que se definen las capas.

En el caso de LeNet, las dos capas de convolución tienen 6 y 16 filtros[12] respectivamente. Tanto en la primera como en la segunda capa, el tamaño del *kernel*[12] es de 5x5. Las capas de *pooling* tienen los mismos filtros de sus capas convolucionales predecesoras (6 y 16 filtros, respectivamente), y sus tamaños son de 2x2 en ambos casos. Tras la segunda y última capa de *pooling*, y una vez aplicada la capa de *flatten*, se encuentran dos capas totalmente conectadas de 120 y 84 nodos respectivamente. Por último, aparece una capa de salida con 10 nodos (debido a su uso para el reconocimiento de dígitos manuscritos, donde hay que diferenciar 10 tipos de dígitos).

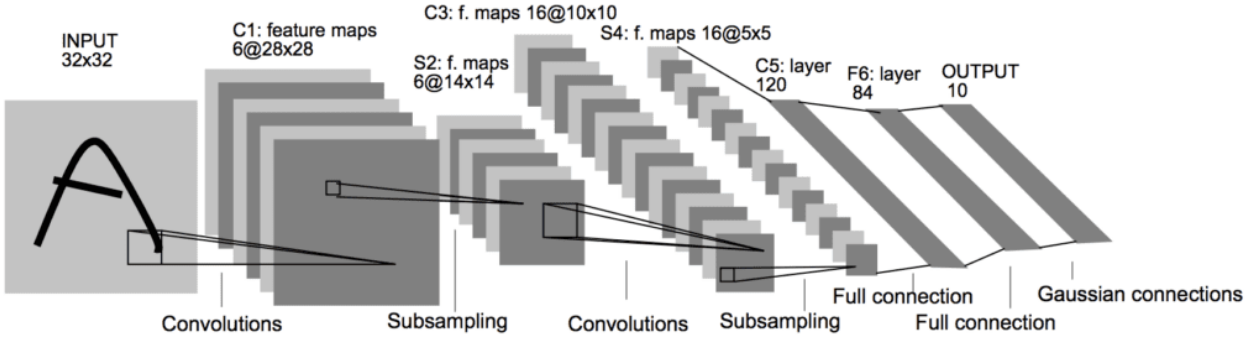

Figura 7.3: Arquitectura LeNet-5

[12] Ver glosario de términos

Esta arquitectura fue probada para la tarea de identificación de la tipología de vehículos con algunas modificaciones en su estructura, como el tamaño de la imagen de entrada (100x100 píxeles en lugar de 32x32). Se introdujeron en la arquitectura cambios en los valores de diferentes parámetros, como las *epochs* (iteraciones) o el tamaño del *batch* (*batch size*)[13], pero se respetó la estructura original de la arquitectura en cuanto al número de capas, su distribución, el número de filtros y su tamaño, las funciones de activación (*tanh*[13]) y las funciones de *pooling* (*AVERAGE-pooling*[13]).

Finalmente, se obtuvieron los resultados que se observan en la tabla 7.2. En esta ocasión, a diferencia de la red neuronal original, no fue necesario reducir el número de tipos de vehículos a clasificar, y todos los resultados que se observan en la tabla se obtuvieron mediante la clasificación de los vehículos en los 10 tipos mencionados en el apartado 7.1.

Batch Size	Learning Rate	Epochs	Validation Split	Test Split	Validation Loss	Validation Accuracy	Test Loss	Test Accuracy
32	0.005	15	0.15	0.9	2.1129	0.2335	2.0524	0.1888
16	0.001	15	0.3	0.9	0.442	0.8705	0.8187	0.8221
64	0.001	15	0.3	0.9	0.3907	0.8865	0.656	0.8335
32	0.001	9	0.4	0.9	0.4017	0.8824	0.6561	0.8411
32	0.001	15	0.15	0.9	1.5135	0.8616	0.5894	0.8691
64	0.001	15	0.15	0.8	1.1783	0.8824	0.4405	0.8843
32	0.001	18	0.15	0.8	1.0148	0.8983	0.3587	0.9129

Tabla 7.2: Tabla de resultados de la arquitectura LeNet-5

[13] Ver glosario de términos

7.2.3. AlexNet

La arquitectura AlexNet fue diseñada por Alex Krizhevsky en 2012 y publicada en el artículo "*ImageNet Classification with Deep Convolutional Neural Networks*" [51]. AlexNet fue ideado para resolver el *ImageNet Large Scale Visual Recognition Challenge 2010*, una competición donde se pretende clasificar imágenes en unas 1000 categorías diferentes. Esta tarea estaba considerada prácticamente inalcanzable hasta que apareció esta nueva arquitectura, que trajo consigo métodos nuevos o poco conocidos en aquel momento y que hoy son la base de cualquier red neuronal para la clasificación de imágenes, como la función de activación ReLU[14] o el uso de *MAX-pooling*[14] en lugar de *AVERAGE-pooling* [8].

AlexNet está formada por dos capas de convolución-*pooling* seguidas de dos capas de convolución consecutivas. Tras ello, se encuentra una nueva capa de convolución-*pooling*, para terminar con dos capas totalmente conectadas y una última de salida. La estructura de esta arquitectura y sus diferencias con la anterior expuesta (LeNet) se pueden observar en la figura 7.4.

El número de filtros en las capas convolucionales va en aumento hasta llegar a la quinta y última, con unas cantidades de 96, 256, 384, 384 y 256 filtros respectivamente. Por el contrario, el tamaño del kernel disminuye, comenzando con una dimensión de 11x11 en la primera, 5x5 en la segunda y 3x3 en las tres últimas.

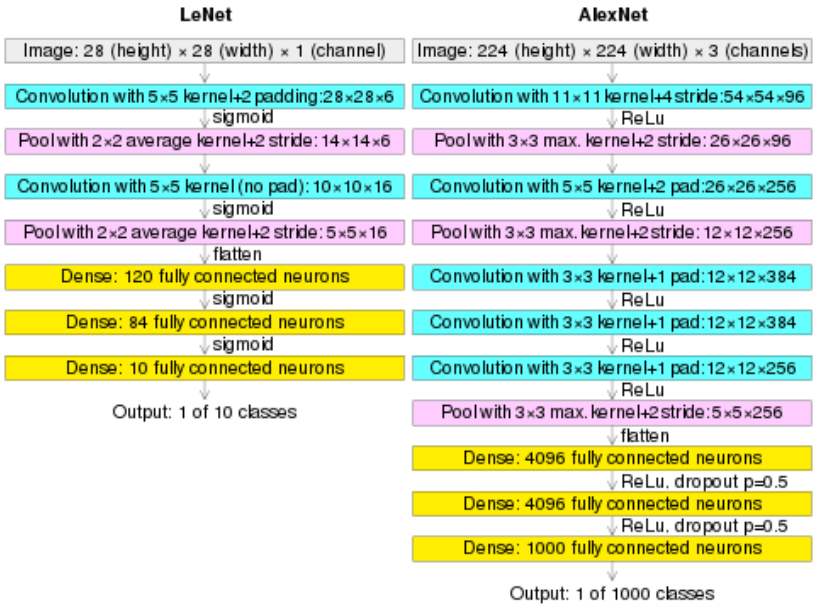

Figura 7.4: Arquitecturas LeNet y AlexNet

[14] Ver glosario de términos

Se hizo uso de esta arquitectura para el cometido de este proyecto, probando diferentes valores en parámetros como el tamaño del *batch*, la tasa de aprendizaje (*learning rate*)[15] o el porcentaje de datos de validación (*validation split*)[15], y se obtuvieron los resultados que se observan en la tabla 7.3. Al igual que con la arquitectura LeNet-5, con AlexNet también se respetó la estructura original de la arquitectura.

Batch Size	Learning Rate	Epochs	Validation Split	Test Split	Validation Loss	Validation Accuracy	Test Loss	Test Accuracy
16	0.005	20	0.3	0.9	2.3262	0.1859	2.2947	0.1918
64	0.005	20	0.3	0.9	15.67	0.279	14.39	0.2586
64	0.001	20	0.15	0.9	4.3546	0.5366	2.803	0.5447
64	0.001	20	0.3	0.8	1.9807	0.7591	0.8564	0.7723
32	0.005	19	0.2	0.9	0.5026	0.8573	0.8657	0.7916
64	0.001	20	0.3	0.9	1.1717	0.8216	0.5475	0.851

Tabla 7.3: Tabla de resultados de la arquitectura AlexNet

[15] Ver glosario de términos

7.2.4. Conclusión al desarrollo de la red neuronal

Tras la investigación realizada en los tres apartados anteriores en los que se ha trabajado en las arquitecturas seleccionadas, se ha observado cuál de ellas favorece la tarea de clasificación de vehículos por su tipología y cuáles no.

A partir de dicho estudio, se decide la creación de una nueva red neuronal que recoja las mejores características de cada una de las arquitecturas probadas. Para ello, partiendo de la red neuronal original (estudiada en el apartado 7.2.1), se ha ido añadiendo sobre todo propiedades de LeNet-5, debido a que ésta ofrecía mejores resultados que AlexNet, tal y como se puede observar en las tablas 7.1, 7.2 y 7.3, más alguna de AlexNet como se verá a continuación.

Así pues, se desarrolla una nueva red neuronal que recoge lo comentado en el párrafo anterior y cuyo código se puede ver en la figura 7.5. Dichas estructuras están compuestas por dos capas de convolución-*pooling* (con MAX-*pooling* en lugar de AVERAGE-*pooling*), una capa de *flatten*, y tres capas totalmente conectadas. Sobre la estructura de la red neuronal original, se ha incorporado una capa más totalmente conectada (pasa de tener dos capas a tres), de forma similar a la arquitectura LeNet. Además, se ha reducido el número de filtros en las capas convolucionales, dejando una cantidad intermedia entre la red neuronal original y LeNet. Por último, se han utilizado funciones de activación *relu* y MAX-*pooling*, siguiendo el esquema de la arquitectura AlexNet.

A continuación, en la siguiente figura, podemos observar el desarrollo del código de la Red Neuronal para el reconocimiento de imágenes y algunos de los resultados obtenidos con ella.

```
#------------ CREACIÓN DEL MODELO ------------
model = Sequential()

model.add(Conv2D(16, (3,3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(120))
model.add(Activation('relu'))

model.add(Dense(80))
model.add(Activation('relu'))

model.add(Dense(ntypes))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])

#tf.compat.v1.disable_eager_execution()
model.fit(Xtr,ytr,class_weight=class_weights_dict,batch_size=32,validation_split=0.15,epochs=10,shuffle=False)
```

Figura 7.5: Red Neuronal final

Batch Size	Learning Rate	Epochs	Validation Split	Test Split	Validation Loss	Validation Accuracy	Test Loss	Test Accuracy
32	0.01	10	0.15	0.9	2.3014	0.0185	2.3063	0.0166
32	0.001	10	0.4	0.7	1.8812	0.8169	0.8769	0.8148
32	0.001	10	0.15	0.7	1.2912	0.8831	0.5953	0.8641
32	0.001	10	0.15	0.8	1.2313	0.8739	0.5004	0.8706
16	0.001	10	0.15	0.9	1.3628	0.8898	0.4351	0.8996
32	0.001	10	0.15	0.9	1.1823	0.9228	0.358	0.9276

Tabla 7.4: Tabla de resultados de la Red Neuronal Final

7.2.5. Desarrollo y explicación de la red neuronal

En este apartado se va a definir con más detalle cada componente de la red neuronal final con la que ha sido implementado el software de reconocimiento de imágenes. Es importante conocer qué aporta cada apartado del conjunto que forma la red neuronal para entender correctamente el funcionamiento de un clasificador de imágenes, y cuáles son los procesos que consiguen unos resultados acertados.

Sequential():

El modelo empleado es de tipo secuencial, es decir, es un modelo formado por una secuencia lineal de capas. Aunque existen otros tipos de modelos para arquitecturas más complejas, como el modelo funcional, en nuestro caso es más adecuado utilizar el secuencial, ya que tal y como aparece en la documentación oficial de *TensorFlow*, es apropiado para conjuntos de capas donde cada una tiene exactamente un tensor de entrada y otro de salida [52].

Conv2D():

Funcionalidad de la API de *Keras*, integrada en *TensorFlow*, que permite la creación de capas convolucionales de 2 dimensiones, especialmente útiles cuando se trabaja con imágenes, como es el caso del presente trabajo.

Se le puede pasar más de una decena de argumentos, pero en la red mostrada en el apartado anterior se le pasa el número de filtros como primer argumento, el tamaño del *kernel* como segundo, y la forma de los datos de entrada (*input_shape*) en el caso de ser la primera capa del modelo.

Activation():

Mediante esta línea se define una función de activación. En el modelo de la red neuronal final mostrado en la figura 7.5. se utiliza la función de activación ReLU en todos los casos (para más información sobre este tipo de función, consultar el glosario de términos).

MaxPooling2D():

Funcionalidad a partir de la cual se pueden crean capas de *pooling*, tanto de Max como de Average. En ese te caso, se está creando una capa de MAX-*pooling*, pasándole como argumento el *pool size* o tamaño de la matriz con la que se realiza la operación.

Flatten():

La capa de *flatten* o alisado permite transformar los datos expresados de forma matricial (como en el caso de los píxeles de una imagen) a forma vectorial, con una sola dimensión. Esto resulta necesario si posteriormente se pretende pasar estos datos a capas totalmente conectadas.

Dense():

La capa *dense* permite crear capas totalmente conectadas (*fully connected layers)*, con el número de neuronas que se le especifique como argumento. En el caso sobre el que trabajamos, se crean tres capas totalmente conectadas: una de 120 neuronas, otra de 80 y una última con tantas neuronas como clases a clasificar.

Compile():

Mediante *compile* definimos la configuración del entrenamiento, especificando como parámetros el optimizador (*optimizer*), la función de pérdida (*loss*), y la métrica con la que evaluar el entrenamiento (*metrics*).

Fit():

Una vez definida la configuración del entrenamiento mediante *compile*, *fit* se ocupará de ejecutarlo durante un número determinado de iteraciones (*epochs*). Además de las iteraciones, también podemos definir como parámetros el conjunto de datos de entrenamiento con sus etiquetas correspondientes (*Xtr* e *ytr* en nuestro caso), el tamaño del *batch* (*batch_size*), el reparto para validación (*validation_split*), la proporción de pesos para cada clase cuando trabajamos con clases de diferentes tamaños (*class_weight*), o si queremos que los datos de entrenamiento sean barajados antes de cada iteración (*shuffle*).

7.3. Lectura automática de matrículas

Como ya se ha comentado, el objetivo de este trabajo no es sólo obtener la tipología de los vehículos, sino además integrar este dato en un conjunto mayor con el fin de conseguir contenido de valor informativo. Para ello es necesario que la información sobre la tipología vaya asociada a otros datos que identifiquen el vehículo, como la matrícula.

El proceso de obtención del número de matrícula a partir de las imágenes utilizadas para la identificación de la tipología del vehículo se puede abordar de diferentes maneras. En un primer lugar, se pensó en utilizar librerías Python existentes que permitieran detectar la matrícula dentro de la imagen y aplicar OCR sobre ella para obtener su número. Principalmente, se valoró el uso de la librería ALPR, la cual ofrecía unos resultados de alta fiabilidad.

Sin embargo, esta librería es una *demo*, por lo que sólo permite utilizarla un determinado número de veces al día. Esta restricción impedía su uso dentro de este proyecto, pues es necesario que la lectura de matrículas esté en funcionamiento constante. Por ello, se decidió realizar el proceso de lectura de matrículas de forma manual, utilizando código propio para la detección y lectura de una matrícula dentro de una imagen.

7.3.1. Librería ALPR

Para vincular el tipo de vehículo que aparece en una imagen con la matrícula que lleva dicho vehículo se puede utilizar la librería ALPR, disponible en Python. Esta librería es capaz de detectar la posición de la matrícula en la imagen primero, y después, mediante OCR, extraer el número de la matrícula y devolverlo en forma de cadena. Un ejemplo del funcionamiento de esta librería se puede observar en la figura 7.6.

La librería ALPR demostró una fiabilidad mayor de la esperada, pues permite obtener el número de matrícula en imágenes donde se observa el vehículo en su totalidad (es decir, el tamaño de la matrícula es muy reducido dentro de la imagen) y, además, donde la calidad de imagen no es perfecta, llegando a proporcionar la matrícula en ocasiones donde el ojo humano tiene dificultades para obtenerla. Analizando el código de la librería disponible en GitHub[16], se puede observar que ALPR es una biblioteca que funciona como intermediario entre el usuario y la página Plate Recognizer[17], la cual ofrece servicios de detección y lectura automática de matrículas.

La problemática que presenta esta librería es que se trata de una versión *demo*, es decir, sólo permite un número determinado de usos. La librería trabaja con los servicios gratuitos que ofrece Plate Recognizer, y si se quiere trabajar con el resto de servicios, se debe pagar la licencia. Debido a que el objetivo de este proyecto es que el software esté obteniendo información sobre los

[16] https://github.com/ismail424/ALPR
[17] https://platerecognizer.com/

vehículos de forma continua y prolongada, se han buscado otras alternativas a la librería ALPR. Sin embargo, si se decidiera abonar el coste de la licencia (cuyos precios se encuentran en el apartado *pricing* de la página web de Plate Recognizer[18]), se podría utilizar esta librería con un número mayor de usos y aprovechar la gran fiabilidad que proporciona.

```
from ALPR import license_recognition as lr
import os
```

```
path = r"C:\Users\dsegura\Desktop\pruebaMat"

for mat in os.listdir(path):
    pathImage = os.path.join(path,mat)
    license_plates = lr.GetLicensePlateDemo(pathImage)
    number = license_plates.get_license_img(pathImage)
    print("matrícula: ", number)
```

```
Token generated: zxOoZFQKXHikari1RjP3hHp4dbuC0IhQpSOgrAyP98pDKdELSNsjcQExF98g9XdF
6094JYY
License image generated from C:\Users\dsegura\Desktop\pruebaMat\1.png : ['6094JYY']
6094JYY
matrícula:  ['6094JYY']
Token generated: OWCAigtaEnSMou15EPMrgCAgCxspp3OEatlO1pNfj7Pd43ge7fWj0nKdyegPUCCh
5763KYX
License image generated from C:\Users\dsegura\Desktop\pruebaMat\2.png : ['5763KYX']
5763KYX
matrícula:  ['5763KYX']
```

Figura 7.6: Código para la lectura de matrículas mediante ALPR

7.3.2. Código propio para la lectura de matrículas

Con el objetivo de tener un método gratuito para la lectura automática de matrículas, y no tener que pagar licencias a ninguna empresa externa, se desarrolló un código propio para obtener unos resultados iguales o similares. Este código está basado en fuentes de Internet donde se explica cómo extraer y leer la matrícula de una imagen mediante las librerías OpenCV y Pytesseract[19]. El código implementado se puede observar en la figura 7.7.

En primer lugar, se necesita obtener una imagen donde sólo aparezca la matrícula. Para ello, a partir de la imagen original se buscan los objetos de forma rectangular. Para facilitar esta tarea,

[18] https://platerecognizer.com/pricing/

[19] https://www.section.io/engineering-education/license-plate-detection-and-recognition-using-opencv-and-pytesseract/
https://www.geeksforgeeks.org/detect-and-recognize-car-license-plate-from-a-video-in-real-time/

se utilizará una imagen previamente recortada (explicado con más detalle en el apartado 8.3) donde, en lugar de contemplarse el vehículo entero, se observe únicamente la sección del mismo en la que se encuentra la matrícula.

Una vez se han encontrado en la imagen los objetos con cuatro costados y han sido almacenados en imágenes aparte, se aplica sobre ellas OCR (tras haberlas sometido a un preprocesado previo) hasta que en una de ellas se obtenga una cadena de caracteres con una estructura similar a la de una matrícula (con una longitud de entre cuatro y diez caracteres).

```
import pytesseract
import cv2
import os
import imutils
from scipy import ndimage
```

```
def checkIfLicense(screencnt,file,img):
    croppedLP = cv2.imread(file)
    gray = cv2.cvtColor(croppedLP, cv2.COLOR_BGR2GRAY)
    croppedLP = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11, 2)
    predicted_result = pytesseract.image_to_string(croppedLP, lang ='eng',
    config ='--oem 3 --psm 6 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
    if predicted_result == "":
        return False
    elif (len(predicted_result)-1 > 4) & (len(predicted_result)-1 < 10):
        print("predicted result: " + predicted_result)
        return True
    else:
        return False
```

```
def readLicense(path):
    image = cv2.imread(path)

    # --- PREPROCESAMIENTO DE LA IMAGEN ---
    image = cv2.resize(image,(600,400))
    imgBlurred =  cv2.bilateralFilter(image,11,17,17)
    gray = cv2.cvtColor(imgBlurred, cv2.COLOR_BGR2GRAY)
    sobelx = cv2.Sobel(gray, cv2.CV_8U, 1, 0, ksize = 3)
    ret2, threshold_img = cv2.threshold(sobelx, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    element = element_structure
    morph_n_thresholded_img = threshold_img.copy()
    cv2.morphologyEx(src = threshold_img,op = cv2.MORPH_CLOSE,kernel = element,dst = morph_n_thresholded_img)

    # --- EXTRACCIÓN DE CONTORNOS ---
    cnts,new = cv2.findContours(morph_n_thresholded_img,mode = cv2.RETR_EXTERNAL,method = cv2.CHAIN_APPROX_NONE)
    cnts = sorted(cnts,key = cv2.contourArea, reverse = True)
    screenCnt = None
    check = None
    for c in cnts:
        perimeter = cv2.arcLength(c,True)
        approx = cv2.approxPolyDP(c,0.018*perimeter,True)
        if len(approx) == 4:
            screenCnt = approx
            x,y,w,h = cv2.boundingRect(c)
            new_img = image[y:y+h,x:x+w]
            fileName = "./matriculasTesseract/mat.jpg"
            cv2.imwrite(fileName,new_img)
            check = checkIfLicense(screenCnt,fileName,image)
            if check is not None:
                break
    return check
```

```
if __name__ == '__main__':
    pytesseract.pytesseract.tesseract_cmd = r"C:\Users\dsegura\AppData\Local\Programs\Tesseract-OCR\tesseract.exe"
    element_structure = cv2.getStructuringElement(shape = cv2.MORPH_RECT, ksize =(22, 5))
    readLicense(r"C:\Users\dsegura\Desktop\proyecto\matriculas\2023-06-08-09_35_02.856157.jpg")
```

Figura 7.7: Código propio para la lectura automática de matrículas

7.3.3. Comparación con cámara externa

Originariamente, tanto la biblioteca ALPR como el código propio se iban a utilizar exclusivamente para obtener la matrícula del vehículo además de su tipología, y así vincular ambos datos posteriormente. Sin embargo, en el transcurso del desarrollo del proyecto se instaló otra cámara en el carril 2, por parte de otros departamentos, para la lectura de matrículas. Con el fin de proporcionar una información nueva y de utilidad para la empresa, se decidió comparar los resultados de lectura de matrículas obtenidos por la última cámara instalada con los que se obtenían para este proyecto mediante la librería ALPR y el código desarrollado, y comprobar así cual era la fiabilidad tanto de la nueva cámara como de la biblioteca y el código.

El código para comparar los resultados obtenidos por la nueva cámara y la librería ALPR se puede observar en la figura 7.8. En primer lugar, se obtienen las matrículas registradas por la nueva cámara a partir del banco de datos donde son almacenadas, en el cual también se almacena la imagen de donde ha sido obtenida. Sobre esta imagen se utilizará la librería ALPR, para posteriormente comparar los resultados que obtiene con los que se encuentran en la base de datos. De forma similar se puede comparar los resultados de la cámara nueva con los obtenidos por el código propio, trasladándole al mismo la imagen de la cámara, en lugar de a la librería ALPR.

```
from ALPR import license_recognition as lr
import os
import openpyxl
import pandas as pd

contDif = 0
```

```
def checkLP(path, value):
    global contDif
    license_plates = lr.GetLicensePlateDemo(path)
    value2 = license_plates.get_license_img(path)
    if not value2:
        contDif = contDif + 1
        f = open("results.txt", "w")
        f.write("Matrícula cámara:" + value + "\n")
        f.write("Matrícula librería: / \n")
        f.write('----------------------------')
        f.close()
    elif value != value2[0]:
        contDif = contDif + 1
        f = open("results.txt", "w")
        f.write("Matrícula cámara:" + value + "\n")
        f.write("Matrícula librería:" + value2[0] + "\n")
        f.write('----------------------------')
        f.close()
```

```
pathxlsx = r"C:\Users\dsegura\Desktop\LPR1718.xlsx"
wb = openpyxl.load_workbook(pathxlsx)
ws = wb.active

for i in range(6, ws.max_row):
    cellPath = 'I'+ str(i)
    valuePath = ws[cellPath].value
    valuePath = valuePath.replace('\\10.10.30.232\copias\LPR_SMART\Datos','Z:')
    valuePath = valuePath[1:]
    cellFile = 'J' + str(i)
    valueFile = ws[cellFile].value
    valueFile = valueFile.replace('.xml','.jpg')
    cellLP = 'F'+ str(i)
    valueLP = ws[cellLP].value
    valueLP = valueLP.replace(" ","")
    valueLP = valueLP.replace("-","")
    imagePath = os.path.join(valuePath,valueFile)
    checkLP(imagePath,valueLP)

perc = (contDif/ws.max_row)*100
print("Porcentaje diferentes: " + perc + "%")
```

Figura 7.8: Código para la comparación de matrículas

8. Implantación

El uso en un entorno real se consigue a través de la implantación de los diferentes módulos que forman el software de reconocimiento de imágenes siguiendo el esquema que se observa en la figura 8.1.

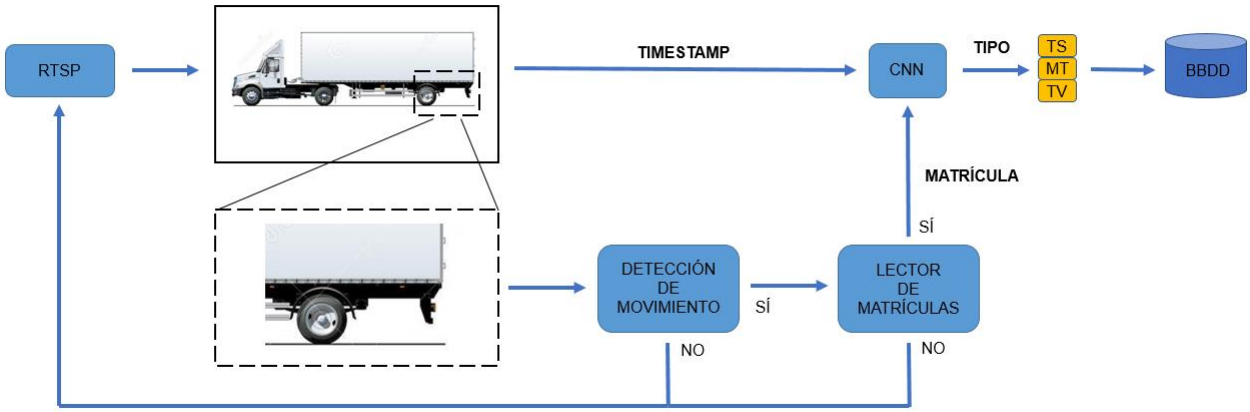

Figura 8.1: Sistema de reconocimiento de imágenes

Así pues, partiendo del vídeo en *streaming* que proporciona la cámara BOSCH NBN-80052-BA situada en el carril 2, la información llega a módulos tales como el lector de matrículas o el modelo entrenado mediante redes neuronales, con el objetivo de obtener nuevos datos formados por la tipología (TV), la matrícula (MT) y la fecha y hora de salida del vehículo (TS).

En los próximos apartados se desarrollará con más detalle cada módulo de este esquema, explicando cuál es su funcionamiento, mostrando su implementación en código y su integración con el resto de módulos.

8.1. Obtención de vídeo en tiempo real (RTSP)

El primer paso del proceso que se observa en la figura 8.1 es la obtención del vídeo en tiempo real (*streaming*). Como ya se ha comentado anteriormente en el apartado 5.4.3, la manera adecuada para conseguir este vídeo en *streaming* es mediante el protocolo RTSP[20].

Para obtener un vídeo transmitido por RTSP a través de Python se ha utilizado la librería OpenCV, ya empleada con anterioridad. Esta librería permite adquirir el vídeo a partir de la url del *streaming*, declarado en el siguiente formato:

rtsp://username:password@<address>:<port>/Streaming/Channels/<id>(?parm1=value1…)

El vídeo en tiempo real se obtiene fotograma a fotograma, y a cada fotograma se le asocia el *timestamp* en el que ha sido recogido. Por tanto, el vídeo en *streaming* que está reproduciendo la cámara se recibe a través del protocolo RTSP, y se almacena cada *frame* con su respectiva fecha y hora.

Estos fotogramas serán posteriormente tratados, tal y como se advierte en la figura 8.1, por varios módulos, entre los que se encuentran el detector de movimiento, el lector de matrículas o la red neuronal para la identificación del tipo.

```
if __name__ == '__main__':
    np.set_printoptions(precision=3,suppress=False)
    pytesseract.pytesseract.tesseract_cmd = r"C:\Users\dsegura\AppData\Local\Programs\Tesseract-OCR\tesseract.exe"
    element_structure = cv2.getStructuringElement(shape = cv2.MORPH_RECT, ksize =(22, 5))
    classes = ["CabezaTractora","CamionCisterna","CamionTolva","Furgoneta","Motocicleta",
               "Otros","Plataforma","Portacoches","Portacontenedor","Turismo"]
    model = tf.keras.models.load_model("CNNfinal10.model")
    cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")
    bbdd = r"C:\Users\dsegura\Desktop\BBDD.xlsx"
    stream(cap)
```

Figura 8.2: Código para la obtención de vídeo en tiempo real (I)

[20] *Real Time Streaming Protocol*

```
def stream(cap):
    background = None

    wb = openpyxl.load_workbook(bbdd)
    ws = wb['Hoja1']

    while True:
        try:
            if cap.isOpened():
                time = datetime.now()
                ret, frame = cap.read()
                frame = imutils.resize(frame,width=1900)
#                 cv2.imshow('carril2',frame)

                # --- Pixeles donde se encuentra el vehiculo ---
                h, w, channels = frame.shape
                y = int(h*0.04)
                Y = int (h*0.877)
                x = int (w*0.123)
                X = int (w*0.877)
                frameT = frame[y:Y,x:X]

                # --- Pixeles donde se encuentra la matricula ---
                y = int (h*0.43)
                Y = int (h*0.72)
                x = int (w*0.4)
                X = int (w*0.66)
                frameMat = frame[y:Y,x:X]
#                 cv2.imshow('carril2Mat',frameMat)

                gray = cv2.cvtColor(frameMat,cv2.COLOR_BGR2GRAY)
                gray = cv2.GaussianBlur(gray,(21,21),0)
                if background is None:
                    background = gray
                diff = mse(background,gray)
                if diff > 5:
                    timestr = time.strftime("%Y-%m-%d-%H_%M_%S.%f")
                    mov = "C:/Users/dsegura/Desktop/proyecto/movimiento/" + timestr + ".jpg"
                    cv2.imwrite(mov,frameMat)
                    predictedMat = readLicense(frameMat)
                    if predictedMat is not None:
                        mat = "C:/Users/dsegura/Desktop/proyecto/matriculas/" + timestr + ".jpg"
                        cv2.imwrite(mat,frameMat)
                        fileName = "C:/Users/dsegura/Desktop/proyecto/vehiculos/" + timestr + ".jpg"
                        cv2.imwrite(fileName,frameT)
                        predictedType = getType(fileName)
                        print("TS: " + timestr)
                        print("MAT: " + predictedMat)
                        print("TIPO: " + predictedType)
                        print("---------" + "\n")
                        data = [time,predictedMat,predictedType]
                        newRow = ws.max_row + 1
                        for i,j in zip(range(1,4),range(3)):
                            ws.cell(column=i,row=newRow,value=data[j])
                        wb.save(filename=bbdd)

                background = gray

#                 if cv2.waitKey(20) & 0xFF == 32:
#                     break

        except Exception as e:
            print("ERROR")
            print(e)
            cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")

    cap.release()
    cv2.destroyAllWindows()
    wb.close()
```

Figura 8.3: Código para la obtención de vídeo en tiempo real (II)

8.2. Detección de movimiento

El proceso para tratar de obtener una matrícula y tipología a partir de una imagen debería empezarse únicamente cuando se detecte que existe un vehículo circulando por el carril 2 en ese preciso instante. La alternativa, que fue contemplada al inicio de la implantación, sería realizar el proceso de forma constante; es decir, intentar en todo momento obtener una matrícula y una tipología a partir de la imagen que se esté obteniendo en la cámara en ese momento, haya o no un vehículo presente. Obviamente, esta opción resulta mucho más costosa e ineficiente.

Por tanto, se ha implementado un módulo para la detección de movimiento, con el objetivo de que envíe una señal que permita empezar todo el proceso mencionado en el párrafo anterior cada vez que un vehículo pase por el carril 2. El código perteneciente a este módulo ha sido desarrollado a partir de código disponible en fuentes recopiladas de Internet[21].

Debido a que la cámara dispone de un amplio ángulo de visión, y con el riesgo que esto supone por el hecho de que pueda detectar un movimiento no deseado y empezar a recoger vídeo sin requerirlo, se ha optado por aprovechar el recorte de la imagen original que se hace en el módulo que realiza la lectura de matrículas y que, por tanto, se centra únicamente en una parte de la imagen que es la que visualiza la matrícula de los vehículos que transitan por el carril. Al utilizar este factor de recorte, nos aseguramos en la detección del movimiento, que la recogida del vídeo únicamente se activará cuando transite un vehículo por el carril.

La detección de movimiento se hará comparando las imágenes (recortadas y centradas en una parte concreta, como se ha mencionado anteriormente) con la imagen recogida previamente en el fotograma anterior, y almacenada con el nombre de *background*. Esta imagen es, por tanto, igual que las que obtiene la cámara en los siguientes fotogramas (con la misma perspectiva y orientación), pero obtenida momentos antes, por lo que si existe una marcada diferencia entre la anterior (*background*) y la actual, se debe a que se está produciendo movimiento.

El nuevo fotograma que se recibe de la cámara será comparado con el fotograma *background* mediante la técnica MSE[22], que proporcionará un valor numérico que represente la diferencia entre la imagen actual y la anterior. Si dicho valor supera un umbral determinado, se considera que se está produciendo movimiento y, por tanto, debe comenzar el proceso de lectura de matrícula.

[21] https://www.geeksforgeeks.org/webcam-motion-detector-python/
https://www.tutorialspoint.com/how-to-compare-two-images-in-opencv-python#

[22] *Mean Squared Error*, Error Cuadrático Medio

```
def stream(cap):
    background = None

    wb = openpyxl.load_workbook(bbdd)
    ws = wb['Hoja1']

    while True:
        try:
            if cap.isOpened():
                time = datetime.now()
                ret, frame = cap.read()
                frame = imutils.resize(frame,width=1900)
#                 cv2.imshow('carril2',frame)

                # --- Pixeles donde se encuentra el vehiculo ---
                h, w, channels = frame.shape
                y = int(h*0.04)
                Y = int (h*0.877)
                x = int (w*0.123)
                X = int (w*0.877)
                frameT = frame[y:Y,x:X]

                # --- Pixeles donde se encuentra la matricula ---
                y = int (h*0.43)
                Y = int (h*0.72)
                x = int (w*0.4)
                X = int (w*0.66)
                frameMat = frame[y:Y,x:X]
#                 cv2.imshow('carril2Mat',frameMat)

                gray = cv2.cvtColor(frameMat,cv2.COLOR_BGR2GRAY)
                gray = cv2.GaussianBlur(gray,(21,21),0)
                if background is None:
                    background = gray
                diff = mse(background,gray)
                if diff > 5:
                    timestr = time.strftime("%Y-%m-%d-%H_%M_%S.%f")
                    mov = "C:/Users/dsegura/Desktop/proyecto/movimiento/" + timestr + ".jpg"
                    cv2.imwrite(mov,frameMat)
                    predictedMat = readLicense(frameMat)
                    if predictedMat is not None:
                        mat = "C:/Users/dsegura/Desktop/proyecto/matriculas/" + timestr + ".jpg"
                        cv2.imwrite(mat,frameMat)
                        fileName = "C:/Users/dsegura/Desktop/proyecto/vehiculos/" + timestr + ".jpg"
                        cv2.imwrite(fileName,frameT)
                        predictedType = getType(fileName)
                        print("TS: " + timestr)
                        print("MAT: " + predictedMat)
                        print("TIPO: " + predictedType)
                        print("---------" + "\n")
                        data = [time,predictedMat,predictedType]
                        newRow = ws.max_row + 1
                        for i,j in zip(range(1,4),range(3)):
                            ws.cell(column=i,row=newRow,value=data[j])
                        wb.save(filename=bbdd)

                background = gray

#                 if cv2.waitKey(20) & 0xFF == 32:
#                     break

        except Exception as e:
            print("ERROR")
            print(e)
            cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")

    cap.release()
    cv2.destroyAllWindows()
    wb.close()
```

Figura 8.4: Código para la detección de movimiento (I)

```
def mse(img1,img2):
    h,w = img1.shape
    diff = cv2.subtract(img1,img2)
    err = np.sum(diff**2)
    mse = err/(float(h*w))
    return mse
```

Figura 8.5: Código para la detección de movimiento (II)

8.3. Lectura de matrícula

Cuando, mediante el módulo de detección de movimiento, se revela que un vehículo se encuentra circulando por el carril 2, se procede a intentar obtener su matrícula. Explicado brevemente, el módulo para la lectura de matrícula buscará una matrícula dentro de una imagen, detectará su posición y aplicará OCR sobre ésta.

En primer lugar, el mismo fotograma que hemos utilizado para detectar el movimiento se le pasa a un método (*readLicense*) que busca los contornos propios de una matrícula. Es decir, tratará de encontrar los contornos de cuatro costados que se encuentren en una imagen dada.

Para cada contorno de la forma mencionada en el párrafo anterior que detecte, se recortará la imagen para quedarse únicamente con la parte marcada por el contorno, y se le pasará esta nueva imagen al método *checkIfLicense*. Este método se encargará de aplicar OCR (mediante la librería Pytesseract) con el objetivo de leer el número de la matrícula.

En el mejor de los casos, el método *readLicense* nos devolverá una imagen donde se vea exclusivamente la matrícula, sin nada a su alrededor. Al aplicar OCR sobre esta imagen, en el método *checkIfLicense*, es muy probable que nos devuelva el número de la matrícula con un alto grado de acierto.

En otros casos, donde haya detectado un objeto de forma similar a una matrícula por error y sea enviado a *checkIfLicense*, el resultado que retornará será nulo. En muchos casos no encontrará caracteres a leer y no podrá aplicar OCR; en otros, puede que aparezcan algunos caracteres dentro de la imagen y trate de leerlos como si se tratara de una matrícula. En ambos casos, los resultados devueltos tras ejecutar OCR no se corresponderán con el formato de una matrícula, por lo que serán descartados.

```
if __name__ == '__main__':
    np.set_printoptions(precision=3,suppress=False)
    pytesseract.pytesseract.tesseract_cmd = r"C:\Users\dsegura\AppData\Local\Programs\Tesseract-OCR\tesseract.exe"
    element_structure = cv2.getStructuringElement(shape = cv2.MORPH_RECT, ksize =(22, 5))
    classes = ["CabezaTractora","CamionCisterna","CamionTolva","Furgoneta","Motocicleta",
               "Otros","Plataforma","Portacoches","Portacontenedor","Turismo"]
    model = tf.keras.models.load_model("CNNfinal10.model")
    cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")
    bbdd = r"C:\Users\dsegura\Desktop\BBDD.xlsx"
    stream(cap)
```

Figura 8.6: Código para la lectura de matrículas (I)

```
def stream(cap):
    background = None

    wb = openpyxl.load_workbook(bbdd)
    ws = wb['Hoja1']

    while True:
        try:
            if cap.isOpened():
                time = datetime.now()
                ret, frame = cap.read()
                frame = imutils.resize(frame,width=1900)
#                 cv2.imshow('carril2',frame)

                # --- Pixeles donde se encuentra el vehiculo ---
                h, w, channels = frame.shape
                y = int(h*0.04)
                Y = int (h*0.877)
                x = int (w*0.123)
                X = int (w*0.877)
                frameT = frame[y:Y,x:X]

                # --- Pixeles donde se encuentra la matricula ---
                y = int (h*0.43)
                Y = int (h*0.72)
                x = int (w*0.4)
                X = int (w*0.66)
                frameMat = frame[y:Y,x:X]
#                 cv2.imshow('carril2Mat',frameMat)

                gray = cv2.cvtColor(frameMat,cv2.COLOR_BGR2GRAY)
                gray = cv2.GaussianBlur(gray,(21,21),0)
                if background is None:
                    background = gray
                diff = mse(background,gray)
                if diff > 5:
                    timestr = time.strftime("%Y-%m-%d-%H_%M_%S.%f")
                    mov = "C:/Users/dsegura/Desktop/proyecto/movimiento/" + timestr + ".jpg"
                    cv2.imwrite(mov,frameMat)
                    predictedMat = readLicense(frameMat)
                    if predictedMat is not None:
                        mat = "C:/Users/dsegura/Desktop/proyecto/matriculas/" + timestr + ".jpg"
                        cv2.imwrite(mat,frameMat)
                        fileName = "C:/Users/dsegura/Desktop/proyecto/vehiculos/" + timestr + ".jpg"
                        cv2.imwrite(fileName,frameT)
                        predictedType = getType(fileName)
                        print("TS: " + timestr)
                        print("MAT: " + predictedMat)
                        print("TIPO: " + predictedType)
                        print("---------" + "\n")
                        data = [time,predictedMat,predictedType]
                        newRow = ws.max_row + 1
                        for i,j in zip(range(1,4),range(3)):
                            ws.cell(column=i,row=newRow,value=data[j])
                        wb.save(filename=bbdd)

                background = gray

#                 if cv2.waitKey(20) & 0xFF == 32:
#                     break

        except Exception as e:
            print("ERROR")
            print(e)
            cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")

    cap.release()
    cv2.destroyAllWindows()
    wb.close()
```

Figura 8.7: Código para la lectura de matrículas (II)

```python
def readLicense(frameMat):

    # --- PREPROCESAMIENTO DE LA IMAGEN ---
    image = cv2.resize(frameMat,(600,400))
    imgBlurred =  cv2.bilateralFilter(image,11,17,17)
    gray = cv2.cvtColor(imgBlurred, cv2.COLOR_BGR2GRAY)
    sobelx = cv2.Sobel(gray, cv2.CV_8U, 1, 0, ksize = 3)
    ret2, threshold_img = cv2.threshold(sobelx, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    element = element_structure
    morph_n_thresholded_img = threshold_img.copy()
    cv2.morphologyEx(src = threshold_img,op = cv2.MORPH_CLOSE,kernel = element,dst = morph_n_thresholded_img)

    # --- EXTRACCIÓN DE CONTORNOS
    cnts,new = cv2.findContours(morph_n_thresholded_img,mode = cv2.RETR_EXTERNAL,method = cv2.CHAIN_APPROX_NONE)
    cnts = sorted(cnts,key = cv2.contourArea, reverse = True)
    screenCnt = None
    check = None
    for c in cnts:
        perimeter = cv2.arcLength(c,True)
        approx = cv2.approxPolyDP(c,0.018*perimeter,True)
        if len(approx) == 4:
            screenCnt = approx
            x,y,w,h = cv2.boundingRect(c)
            new_img = image[y:y+h,x:x+w]
            fileName = "./matriculasTesseract/mat.jpg"
            cv2.imwrite(fileName,new_img)
            check = checkIfLicense(screenCnt,fileName,image)
            if check is not None:
                break
    return check
```

Figura 8.8: Código para la lectura de matrículas (III)

```python
def checkIfLicense(screencnt,file,img):
    image = cv2.imread(file)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    croppedLP = cv2.adaptiveThreshold(gray,
                                      255,
                                      cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      cv2.THRESH_BINARY,
                                      11, 2)
    predicted_result = pytesseract.image_to_string(croppedLP, lang ='eng',
    config ='--oem 3 --psm 6 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
    if (len(predicted_result)-1 > 4) & (len(predicted_result)-1 < 10):
        return predicted_result
    else:
        return None
```

Figura 8.9: Código para la lectura de matrículas (IV)

8.4. Obtención de la tipología del vehículo

Una vez se ha detectado movimiento en el carril 2, y del vehículo que ha producido dicho movimiento se ha obtenido su matrícula, el siguiente paso es el de obtener su tipología. Cuando su tipo sea determinado, ya tendremos toda la información necesaria para la creación de un nuevo dato (*timestamp*, matrícula y tipología), y podrá ser almacenada en una base de datos e ingestada la información allá donde se requiera.

Para conseguir la tipología de un vehículo, del que previamente hemos detectado su matrícula, se utilizará el modelo de redes neuronales que hemos entrenado. A este modelo se le pasara la imagen original (en la que se ve el vehículo completo) a partir de la cual se ha obtenido la matrícula (tras aplicarle a la imagen original las transformaciones mencionadas en los apartados anteriores).

El proceso para predecir cuál es la tipología de un vehículo presente en una imagen se realiza en el método *getType*. En él se preprocesará la imagen que se le pasa con argumento, para que tenga las mismas características y transformaciones que las imágenes con las que ha sido entrenado el modelo. Posteriormente, el modelo hará una predicción sobre la tipología del vehículo presente en la imagen y la retornará para ser unida a la fecha, hora y matrícula.

```
if __name__ == '__main__':
    np.set_printoptions(precision=3,suppress=False)
    pytesseract.pytesseract.tesseract_cmd = r"C:\Users\dsegura\AppData\Local\Programs\Tesseract-OCR\tesseract.exe"
    element_structure = cv2.getStructuringElement(shape = cv2.MORPH_RECT, ksize =(22, 5))
    classes = ["CabezaTractora","CamionCisterna","CamionTolva","Furgoneta","Motocicleta",
               "Otros","Plataforma","Portacoches","Portacontenedor","Turismo"]
    model = tf.keras.models.load_model("CNNfinal10.model")
    cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")
    bbdd = r"C:\Users\dsegura\Desktop\BBDD.xlsx"
    stream(cap)
```

Figura 8.10: Código para la obtención de la tipología del vehículo (I)

```
def stream(cap):
    background = None

    wb = openpyxl.load_workbook(bbdd)
    ws = wb['Hoja1']

    while True:
        try:
            if cap.isOpened():
                time = datetime.now()
                ret, frame = cap.read()
                frame = imutils.resize(frame,width=1900)
#                 cv2.imshow('carril2',frame)

                # --- Pixeles donde se encuentra el vehiculo ---
                h, w, channels = frame.shape
                y = int(h*0.04)
                Y = int (h*0.877)
                x = int (w*0.123)
                X = int (w*0.877)
                frameT = frame[y:Y,x:X]

                # --- Pixeles donde se encuentra la matricula ---
                y = int (h*0.43)
                Y = int (h*0.72)
                x = int (w*0.4)
                X = int (w*0.66)
                frameMat = frame[y:Y,x:X]
#                 cv2.imshow('carril2Mat',frameMat)

                gray = cv2.cvtColor(frameMat,cv2.COLOR_BGR2GRAY)
                gray = cv2.GaussianBlur(gray,(21,21),0)
                if background is None:
                    background = gray
                diff = mse(background,gray)
                if diff > 5:
                    timestr = time.strftime("%Y-%m-%d-%H_%M_%S.%f")
                    mov = "C:/Users/dsegura/Desktop/proyecto/movimiento/" + timestr + ".jpg"
                    cv2.imwrite(mov,frameMat)
                    predictedMat = readLicense(frameMat)
                    if predictedMat is not None:
                        mat = "C:/Users/dsegura/Desktop/proyecto/matriculas/" + timestr + ".jpg"
                        cv2.imwrite(mat,frameMat)
                        fileName = "C:/Users/dsegura/Desktop/proyecto/vehiculos/" + timestr + ".jpg"
                        cv2.imwrite(fileName,frameT)
                        predictedType = getType(fileName)
                        print("TS: " + timestr)
                        print("MAT: " + predictedMat)
                        print("TIPO: " + predictedType)
                        print("---------" + "\n")
                        data = [time,predictedMat,predictedType]
                        newRow = ws.max_row + 1
                        for i,j in zip(range(1,4),range(3)):
                            ws.cell(column=i,row=newRow,value=data[j])
                        wb.save(filename=bbdd)

                background = gray

#                 if cv2.waitKey(20) & 0xFF == 32:
#                     break

        except Exception as e:
            print("ERROR")
            print(e)
            cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")

    cap.release()
    cv2.destroyAllWindows()
    wb.close()
```

Figura 8.11: Código para la obtención de la tipología del vehículo (II)

```
def getType(path):
    imgArray = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
    imgArray = cv2.resize(imgArray,(100,100))
    imgArray = imgArray.reshape(-1,100,100,1)
    prediction = model.predict([imgArray])
    index = np.argmax(prediction)
    return classes[index]
```

Figura 8.12: Código para la obtención de la tipología del vehículo (III)

8.5. Almacenamiento de los datos

Los nuevos datos que se creen, formados por el *timestamp*, la matrícula y la tipología, se almacenarán en una base de datos propia, con la posibilidad de ser integrados posteriormente en bases de datos de la empresa, tales como PI (de la que se habló en el apartado 5.4).

En un primer instante, todos los datos que se generen serán guardados en la base de datos, sin ejercer comprobaciones como si la matrícula ya se encuentra almacenada. Más tarde, la base de datos será procesada y se eliminarán de ella los datos inválidos y los repetidos.

Para la comprobación de datos inválidos se hará uso de la cámara para la lectura de matrículas ya instalada por parte de la empresa, mencionada en el apartado 7.3.3. Cuando se obtenga un nuevo dato, se verificará que su matrícula es idéntica o similar a alguna de las matrículas detectadas por la cámara ajena en la misma fecha y hora (en concreto, se comprobarán las matrículas que se encuentren en un intervalo de tres minutos, en el caso de que el *timestamp* de ambas cámaras no esté perfectamente sincronizado).

Cuando se haya cotejado con la cámara ajena la matrícula del nuevo dato, y exista una matrícula similar a la misma hora en la base de datos de dicha cámara, la matrícula del dato será sustituida por la que se encuentra en esta base de datos externa, pues ofrece una mayor fiabilidad. Este nuevo dato, con la modificación de la matrícula realizada, será introducido en la base de datos definitiva siempre y cuando no se haya introducido un dato con la misma matrícula recientemente, con el fin de evitar los datos repetidos.

```
if __name__ == '__main__':
    np.set_printoptions(precision=3,suppress=False)
    pytesseract.pytesseract.tesseract_cmd = r"C:\Users\dsegura\AppData\Local\Programs\Tesseract-OCR\tesseract.exe"
    element_structure = cv2.getStructuringElement(shape = cv2.MORPH_RECT, ksize =(22, 5))
    classes = ["CabezaTractora","CamionCisterna","CamionTolva","Furgoneta","Motocicleta",
               "Otros","Plataforma","Portacoches","Portacontenedor","Turismo"]
    model = tf.keras.models.load_model("CNNfinal10.model")
    cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")
    bbdd = r"C:\Users\dsegura\Desktop\BBDD.xlsx"
    stream(cap)
```

Figura 8.13: Código para el almacenamiento de datos (I)

```
def stream(cap):
    background = None

    wb = openpyxl.load_workbook(bbdd)
    ws = wb['Hoja1']

    while True:
        try:
            if cap.isOpened():
                time = datetime.now()
                ret, frame = cap.read()
                frame = imutils.resize(frame,width=1900)
#                 cv2.imshow('carril2',frame)

                # --- Pixeles donde se encuentra el vehiculo ---
                h, w, channels = frame.shape
                y = int(h*0.04)
                Y = int (h*0.877)
                x = int (w*0.123)
                X = int (w*0.877)
                frameT = frame[y:Y,x:X]

                # --- Pixeles donde se encuentra la matricula ---
                y = int (h*0.43)
                Y = int (h*0.72)
                x = int (w*0.4)
                X = int (w*0.66)
                frameMat = frame[y:Y,x:X]
#                 cv2.imshow('carril2Mat',frameMat)

                gray = cv2.cvtColor(frameMat,cv2.COLOR_BGR2GRAY)
                gray = cv2.GaussianBlur(gray,(21,21),0)
                if background is None:
                    background = gray
                diff = mse(background,gray)
                if diff > 5:
                    timestr = time.strftime("%Y-%m-%d-%H_%M_%S.%f")
                    mov = "C:/Users/dsegura/Desktop/proyecto/movimiento/" + timestr + ".jpg"
                    cv2.imwrite(mov,frameMat)
                    predictedMat = readLicense(frameMat)
                    if predictedMat is not None:
                        mat = "C:/Users/dsegura/Desktop/proyecto/matriculas/" + timestr + ".jpg"
                        cv2.imwrite(mat,frameMat)
                        fileName = "C:/Users/dsegura/Desktop/proyecto/vehiculos/" + timestr + ".jpg"
                        cv2.imwrite(fileName,frameT)
                        predictedType = getType(fileName)
                        print("TS: " + timestr)
                        print("MAT: " + predictedMat)
                        print("TIPO: " + predictedType)
                        print("----------" + "\n")
                        data = [time,predictedMat,predictedType]
                        newRow = ws.max_row + 1
                        for i,j in zip(range(1,4),range(3)):
                            ws.cell(column=i,row=newRow,value=data[j])
                        wb.save(filename=bbdd)

                background = gray

#                 if cv2.waitKey(20) & 0xFF == 32:
#                     break

        except Exception as e:
            print("ERROR")
            print(e)
            cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")

    cap.release()
    cv2.destroyAllWindows()
    wb.close()
```

Figura 8.14: Código para el almacenamiento de datos (II)

```
wbbd = openpyxl.load_workbook(r"C:\Users\dsegura\Desktop\BBDD.xlsx")
wbcit = openpyxl.load_workbook(r"C:\Users\dsegura\Desktop\InformeLprAccesos_13_06_2023.xlsx")
wsbd = wbbd['Hoja1']
wscit = wbcit['Informe LPR Accesos']
wsres = wbbd['Res']


while True:
    row = wsbd[2]
    if not all(cell.value for cell in row):
        break

    f = wsbd.cell(row=2,column=1).value
    m = wsbd.cell(row=2, column=2).value.replace(" ","")
    t = wsbd.cell(row=2,column=3).value
    for j in range(6,wscit.max_row):
        if wscit.cell(row=j,column=4).value == "11S2":
            fecha = wscit.cell(row=j,column=5).value
            dif = abs((f-fecha).total_seconds()/60.0)
            if dif < 4:
                matricula = wscit.cell(row=j,column=6).value.replace(" ","")
                if SequenceMatcher(None,matricula,m).ratio() >= 0.65:
                    matAnterior = wsres.cell(row=wsres.max_row,column=2).value
                    if matAnterior != matricula:
                        newRow = wsres.max_row+1
                        wsres.cell(row=newRow,column=1,value=f)
                        wsres.cell(row=newRow,column=2,value=matricula)
                        wsres.cell(row=newRow,column=3,value=t)

    wsbd.delete_rows(2)



wbbd.save(filename=r"C:\Users\dsegura\Desktop\BBDD.xlsx")
wbbd.close()
wbcit.save(filename=r"C:\Users\dsegura\Desktop\InformeLprAccesos_13_06_2023.xlsx")
wbcit.close()
```

Figura 8.15: Código para la comprobación de matrículas

8.6. Gestión de los datos

Al término del proceso descrito en la figura 8.1 se obtiene una base de datos propia formada por la fecha, la matrícula y la tipología de los vehículos que salen por el carril 2 del Acceso Sur del Puerto de Valencia. Esta nueva información que se obtiene a partir del software desarrollado forma parte de un proyecto de mayor dimensión dentro de la propia empresa, y por ello es fundamental su correcta integración con el resto de los datos.

El proyecto del que forma parte el presente trabajo persigue la trazabilidad de los vehículos durante su estancia en el Puerto. A partir de fuentes como las cámaras de vigilancia, de las que se obtienen datos como el tiempo en el que ha sido visto el vehículo, su matrícula o, en el caso concreto de la cámara de este trabajo, la tipología del vehículo avistado, se pretende conseguir información fiable acerca de cada vehículo relativa a su posición, su tiempo de estancia en determinadas zonas del recinto, o la mercancía que portaba tanto en la entrada como en la salida.

Por tanto, una parte del proyecto encargado de seguir la trazabilidad de los vehículos sería el presente software de reconocimiento de imágenes, que por una parte proporcionará información sobre la matrícula y fecha de los vehículos, disponible en otras zonas y carriles, pero no en el carril 2 de salida; y por otra, aportará una nueva información como la tipología del vehículo que se dispone a abandonar el recinto.

Los datos obtenidos por este software serán integrados con otros datos del mismo vehículo obtenidos en otras áreas, formando así la información completa que pretende proporcionar la trazabilidad del vehículo. Sin embargo, cabe mencionar que este ambicioso proyecto de la empresa todavía no está en funcionamiento, por lo que todavía no se puede probar adecuadamente la correcta integración de los nuevos datos en un sistema mayor.

9. Pruebas

Para analizar el comportamiento del software en un entorno real se realizó un gran número de pruebas. A partir de los resultados de estas pruebas se han ido ajustando factores determinantes para el correcto rendimiento del software, como qué fracción de la imagen utilizar para buscar y detectar movimiento o qué configuración de la librería Pytesseract utilizar para el proceso de OCR en matrículas, entre otros.

Para realizar estas pruebas, tal y como se ha explicado en el apartado de la Implantación (8), el vídeo procedente de la cámara del carril 2 se obtiene en tiempo real mediante el protocolo RTSP. Este vídeo se obtiene fotograma a fotograma, recibiendo imágenes como la que se muestra en la figura 9.1 en cada uno de ellos. Para cada una de estas imágenes se almacenará, además, su *timestamp*, es decir, la fecha y hora en la que ha sido obtenida[23].

Figura 9.1: Fotograma obtenido por la cámara

[23] Las matrículas han sido pixeladas en todas las imágenes debido a la Ley de Protección de Datos.

Cada fotograma obtenido se recorta para extraer de él la fracción de la imagen en la que se encontrará la matrícula del vehículo. Como se ha explicado en el apartado 8.2, esta nueva imagen será utilizada también para el proceso de detección de movimiento.

En la figura 9.2 se observa el fragmento recortado de la imagen original para el caso en el que no circula ningún vehículo, es decir, no existe movimiento. Esta imagen se almacenará como *background*, y será utilizada posteriormente para la comparación con la imagen del siguiente fotograma procedente de la cámara, con el fin de determinar si existe o no movimiento en función de la similitud entre ambas imágenes.

Figura 9.2: Imagen en la que no circula ningún vehículo

En la figura 9.3 se encuentra una imagen de la misma orientación y ángulo que la imagen de la figura 9.2, pero tomada en el momento en el que sí circulaba un vehículo por el carril 2. Esta imagen, por tanto, detectará movimiento (pues su fotograma anterior, almacenado en *background*, será distinto) y comenzará el proceso de búsqueda de matrícula. Sin embargo, en la imagen no existe ninguna matrícula visible, por lo que descartará la imagen y se repetirá el proceso completo (desde su obtención por RTSP) con el siguiente fotograma.

Figura 9.3: Imagen con movimiento y sin matrícula

El caso de detectar movimiento y además encontrar una matrícula visible se puede observar en la figura 9.4. A partir de esta imagen será extraída la matrícula, que tras pasar por un proceso de comparaciones para asegurar que es correcta, pasará a formar parte del nuevo dato fecha-matrícula-tipología.

Figura 9.4: Imagen con movimiento y matrícula

Una vez obtenida la matrícula, y con el *timestamp* almacenado, comienza el proceso para la identificación de la tipología, tal y como indica la figura 8.1. Este procedimiento, que se ejecutará a partir de la imagen original levemente recortada para eliminar ruido externo y focalizar la imagen en el vehículo, dará como resultado la tipología del vehículo que circula por el carril 2.

Unidos los tres datos se obtiene el nuevo dato (*timestamp*-matrícula-tipología), que será almacenado en una base de datos provisional, tal y como aparece en la figura 9.5. En esta base de datos, construida en Excel, se acumularán todos los datos que se vayan obteniendo.

Timestamp	Matrícula	Tipología
11_12_53.693186		Plataforma
11_12_54.913059		Plataforma
11_12_55.575665		Plataforma
11_12_56.223869		Plataforma
11_12_56.888789		Plataforma
11_14_26.427492		Plataforma
11_17_40.083707		CamionCisterna
11_17_40.688032		CamionCisterna
11_17_41.039330		CamionCisterna
11_17_50.666409		Plataforma
11_17_52.095806		Plataforma
11_17_52.433253		Plataforma
11_17_53.118608		Plataforma
11_17_53.451872		Plataforma
11_17_53.877474		Plataforma
11_17_54.542814		Plataforma
11_20_00.436143		Turismo
11_21_12.133459		CamionCisterna
11_21_30.890949		Plataforma
11_41_35.174835		Portacontenedor
11_41_43.865642		Plataforma

Figura 9.5: Muestra de la Base de Datos provisional

Los datos almacenados en la base de datos provisional serán sometidos a un proceso en el que se confirmará que la matrícula leída se corresponde con las que lee la cámara externa destinada exclusivamente a la lectura de matrículas, y se eliminarán elementos repetidos (con la misma matrícula en un período de tiempo muy cercano). Tras ello, se obtendrá la base de datos definitiva, formada por la información de los vehículos que salen por el carril 2 del Acceso Sur del Puerto de Valencia.

Timestamp	Matrícula	Tipología
2023-05-15-11_57_34		Plataforma
2023-05-15-11_57_56		Portacontenedor
2023-05-15-11_58_44		Turismo
2023-05-15-12_00_35		Portacontenedor
2023-05-15-12_01_18		Portacontenedor
2023-05-15-12_04_29		Plataforma
2023-05-15-12_05_08		Furgoneta
2023-05-15-12_11_17		Plataforma
2023-05-15-12_17_26		Turismo
2023-05-15-12_20_41		Turismo
2023-05-15-12_21_02		Turismo

Figura 9.6: Muestra de la Base de Datos definitiva

10. Conclusiones

El trabajo de fin de grado realizado ha buscado no sólo alcanzar el objetivo propuesto, que es el de desarrollar un software que permita la identificación de los vehículos que salen del puerto a partir de una lista personalizada de tipologías de vehículos, sino que además se ha incidido en una línea de investigación que ha permitido poder elegir aquella herramienta de trabajo más adecuada al objetivo marcado, así como la de construir un banco de imágenes lo suficientemente amplio como para entrenar un modelo bajo las condiciones requeridas.

Tal y como se comenta en el Estado del Arte (apartado 3), no se ha encontrado ningún artículo de investigación en el que se haya aplicado una tecnología de este tipo para identificar la tipología del vehículo en ningún puerto. Si bien es cierto que existe software que permite identificar si el vehículo que entra es un camión o turismo, aquí lo que se pretende es atender una necesidad que procede de un departamento vinculado a la gestión del dato, por lo que el reconocimiento de los vehículos por tipología debe ser aquél con el suficiente detalle tipológico que permita encargarse de dicha demanda, y esto no lo ofrece ningún software de reconocimiento de imágenes. Por lo tanto, uno de los retos del trabajo ha sido el de construir un banco de 8.428 imágenes repartidas entre 10 tipos de vehículos a partir tanto de imágenes extraídas de bancos externos como obtenidas a partir de videos de las cámaras del puerto con el fin de que sirvan para el entrenamiento del modelo.

La arquitectura de la red neuronal se decidió finalmente a partir de una investigación realizada sobre dos arquitecturas ampliamente conocidas en el campo del reconocimiento de imágenes como son la LeNet-5 y la AlexNet. Cabe mencionar que los primeros intentos de identificación de tipologías, en los que se clasificaban un número reducido de éstas (generalmente dos o tres), se ejecutaron sobre una arquitectura original simple, basada en un tutorial encontrado en Internet[24]. Una vez efectuadas un número determinado de pruebas sobre las arquitecturas LeNet-5 y AlexNet (clasificando en las diez tipologías que se pretende identificar), se llegó a la conclusión de que la red neuronal que se utilizaría para el reconocimiento de imágenes debería recoger las mejores características de cada una de las arquitecturas probadas, y especialmente, las procedentes de la arquitectura LeNet-5, al aportar ésta los mejores resultados.

Otro tema muy interesante es el de la lectura de la matrícula, ya que lo que se pretende es asociar la matrícula del vehículo a la tipología, así como la fecha y hora de paso por la salida. Con este propósito, se realizó también una investigación para averiguar qué herramienta podría ser la mejor para la obtención de la matrícula. Se utilizó en un primer momento la librería ALPR de Python, la cual ofreció resultados de alta fiabilidad. Sin embargo, al ser una demo y requerir del pago de una licencia para su uso completo, se tuvo que pensar en otra opción y se desarrolló un código propio mediante las librerías OpenCV y Pytesseract. En primer lugar, este código obtiene la matrícula del vehículo a partir de la imagen, y posteriormente, compara los datos

[24] https://pythonprogramming.net/introduction-deep-learning-python-tensorflow-keras/

extraídos con los datos procedentes de una cámara auxiliar existente en el acceso y verifica que los datos recogidos tengan un alto porcentaje de fiabilidad.

Finalmente, en el proceso de implantación se integran los diferentes módulos siguiendo el esquema siguiente, el cual se explica con detalle en el apartado ocho del presente trabajo.

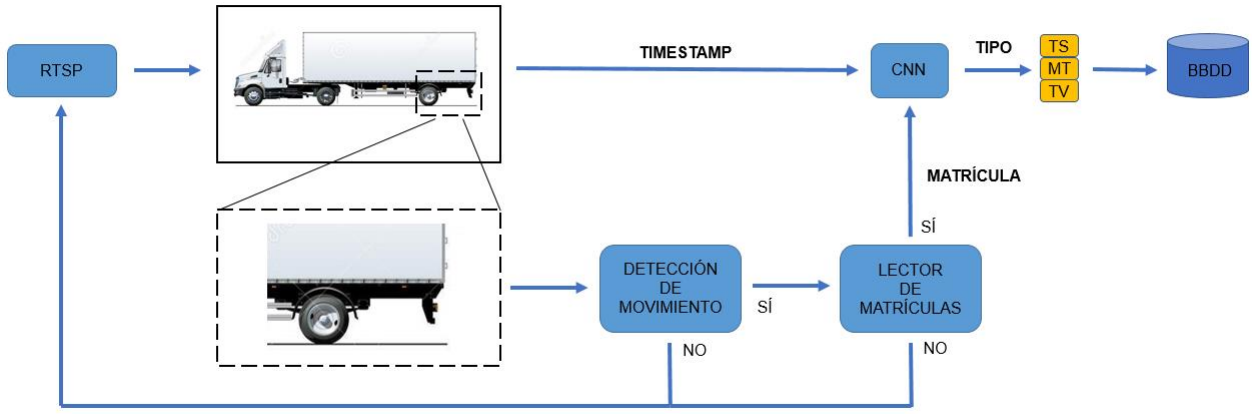

Figura 10.1: Sistema de reconocimiento de imágenes

El software desarrollado para el presente trabajo se enmarca en un proyecto mucho más amplio que pretende recoger la trazabilidad de cualquier vehículo que entre y salga del recinto portuario, aportando el software desarrollado en este trabajo información sobre la parte última de dicha trazabilidad al desencadenar el evento final con la salida del vehículo por una salida manual como es la salida número dos del puerto (carril 2 del Acceso Sur). La información recogida en este proyecto es trasladada a una base de datos con el fin de ser integrada con información procedente de otros eventos.

11. Trabajos futuros

Respecto a posibles trabajos futuros, existen diferentes aspectos mejorables que se han ido comentando a lo largo del trabajo. Estas mejoras, de las cuales se hablará con más detalle a continuación, no han sido implementadas por cuestiones temporales o económicas, en función de cada caso particular. Sin embargo, se dejan propuestas en este apartado con el fin de que sean de utilidad para una posible versión mejorada a desarrollar en el futuro.

En primer lugar, sería posible **mejorar el banco de imágenes** para el entrenamiento del modelo. La creación de dicho banco fue una de las tareas que más tiempo requirió del presente proyecto, por lo que, una vez conseguido un banco con el tamaño necesario para el correcto funcionamiento del software, se pasó a trabajar en otras tareas del mismo. No obstante, se podría continuar con la ampliación del banco, aportando nuevas imágenes para cada clase de vehículo (en especial, para las que menos imágenes tengan), lo que mejoraría la precisión del software.

Por otro lado, para la **lectura automática de matrículas** (apartado 7.3) se utiliza un código propio desarrollado a partir de fuentes encontradas en Internet. Sin embargo, y como ya se comentó en apartados anteriores, existe una librería Python llamada ALPR, la cual es capaz de realizar la misma tarea con mayor fiabilidad y acierto. En este trabajo no se hace uso de esta librería debido a que se necesita pagar una licencia para ello, pero teniendo en cuenta el alto rendimiento que proporciona, sería interesante abonar su precio y realizar el proceso de lectura de matrículas utilizando la librería ALPR.

Por último, el software de reconocimiento de imágenes ha sido implantado únicamente en el carril 2 de salida del Acceso Sur, tal y como se ha mencionado en apartados anteriores. Sin embargo, una posible ampliación a realizar en el futuro sería la **implantación del software en otros carriles de circulación** dentro del Puerto de Valencia, con el fin de obtener una cantidad mayor de información y, además, comprobar el funcionamiento del reconocedor de imágenes en múltiples entornos.

12. Bibliografía

[1] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial . [Último acceso: 23 Marzo 2023].

[2] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Reconocimiento_de_patrones. [Último acceso: 24 Marzo 2023].

[3] «MathWorks,» [En línea]. Available: https://es.mathworks.com/discovery/image-recognition-matlab.html. [Último acceso: 24 Marzo 2023].

[4] R. Szeliski, «Computer Vision: Algorithms and Applications,» *Springer Science & Business Media,* pp. 10-16, 2010.

[5] M. Turk y A. Pentland, «Face recognition using eigenfaces,» de *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1991.

[6] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Computer_vision . [Último acceso: 24 Marzo 2023].

[7] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/ImageNet . [Último acceso: 28 Marzo 20223].

[8] «Machine Learning Mastery,» [En línea]. Available: https://machinelearningmastery.com/review-of-architectural-innovations-for-convolutional-neural-networks-for-image-classification/. [Último acceso: 28 Marzo 2023].

[9] «Deci,» [En línea]. Available: https://deci.ai/blog/sota-dnns-overview/. [Último acceso: 24 Marzo 2023].

[10] «ClearML,» [En línea]. Available: https://clear.ml/blog/the-battle-of-speed-accuracy-single-shot-vs-two-shot-detection/#:~:text=While%20two%2Dshot%20detection%20models,than%20a%20two%2Dshot%20detector. [Último acceso: 24 Marzo 2023].

[11] «Medium,» [En línea]. Available: https://medium.com/georgian-impact-blog/state-of-computer-vision-cvpr-2021-7c02b60e70e2. [Último acceso: 24 Marzo 2023].

[12] X. Chen, C. Xie, M. Tan, L. Zhang, C.-J. Hsieh y B. Gong, «Robust and Accurate Object Detection via Adversarial Learning,» 2021.

[13] S. Becker y G. Hinton, «Self-organizing neural network that discovers surfaces in random-dot stereograms,» *Nature* , pp. 161-163, 1992.

[14] X. Gao, J. Zhang y Z. Wei, «Deep learning for sequence pattern recognition,» de *IEEE 15th International Conference on Networking, Sensing and Control*, 2018.

[15] K. Fukushima y S. Miyake, «Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition,» de *Competition and cooperation in neural nets*, 1982, pp. 264-285.

[16] D. Hubel y T. Wiesel, «Receptive fields, binocular interaction and functional architecture in the cat's visual cortex,» *The Journal of Physiology*, pp. 106-154, 1962.

[17] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE*, pp. 2278-2324, 1998.

[18] A. Ajit, K. Acharya y A. Samanta, «A review of Convolutional Neural Networks,» de *Internacional Conference on Emerging Trend in Information Technology and Engineering* , 2020.

[19] N. Sharma, V. Jain y A. Mishra, «An analysis of Convolutional Neural Networks for image classification,» de *International Conference on Computational Intelligence and Data Science*, 2018.

[20] X. Feng, Z. Wang y T. Liu, «Port Container Number Recognition System Based on Improved YOLO and CRNN Algorithm,» de *International Conference on Artificial Intelligence and Electromechanical Automation*, 2020.

[21] C. Mi, L. Cao, Z. Zhang, Y. Feng, L. Yao y Y. Wu, «A Port Container Code Recognition Algorithm under Natural Conditions,» *Journal of Coastal Research*, 2020.

[22] D. Li y Y. Zhou, «Recognition and detection of ports and ships based on image applications and non local feature enhancement,» de *IEEE 10th Joint International Information Technology and Artificial Intelligence Conference* , 2022.

[23] L. Jiang, G. Peng, B. Xu, Y. Lu y W. Wang, «Foreign object recognition technology for port transportation channel based on automatic image recognition,» *EURASIP Journal on Image and Video Processing*, 2018.

[24] X. Ni y H. Huttunen, «Vehicle attribute recognition by appearance: computer vision methods for vehicle type, make and model classification,» *Journal of Signal Processing Systems*, 2021.

[25] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Machine_vision . [Último acceso: 29 Marzo 2023].

[26] «Dynatec,» [En línea]. Available: https://dynatec.es/2021/07/24/computer-vision-la-vision-artificial-y-sus-aplicaciones/#:~:text=%C2%BFQu%C3%A9%20es%20la%20visi%C3%B3n%20artificial,esa%20informaci%C3%B3n%20para%20tomar%20decisiones.. [Último acceso: 10 Abril 2023].

[27] «deepomatic,» [En línea]. Available: https://deepomatic.com/what-is-image-recognition. [Último acceso: 28 Marzo 2023].

[28] «TechTarget,» [En línea]. Available: https://www.techtarget.com/searchenterpriseai/definition/image-recognition. [Último acceso: 29 Marzo 2023].

[29] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico. [Último acceso: 30 Marzo 2023].

[30] P. Norvig y S. Russell, de *Artificial Intellligence: A Modern Approach*, Pearson, 2021, p. 651.

[31] «bismart,» [En línea]. Available: https://blog.bismart.com/diferencia-machine-learning-deep-learning. [Último acceso: 30 Marzo 2023].

[32] «Xataka,» [En línea]. Available: https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial. [Último acceso: 30 Marzo 2023].

[33] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_supervisado. [Último acceso: 30 Marzo 2023].

[34] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_no_supervisado. [Último acceso: 30 Marzo 2023].

[35] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa. [Último acceso: 22 Mayo 2023].

[36] «Machine Learning Mastery,» [En línea]. Available: https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/. [Último acceso: 4 Mayo 2023].

[37] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Red_neuronal_artificial. [Último acceso: 23 Mayo 2023].

[38] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Neurona_de_McCulloch-Pitts. [Último acceso: 23 Mayo 2023].

[39] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network. [Último acceso: 23 Mayo 2023].

[40] X. Fan y P. Truong, «An Introduction to Convolutional Neural Network (CNN),» *Medium*, 2022.

[41] «NumPy,» [En línea]. Available: https://es.wikipedia.org/wiki/NumPy. [Último acceso: 24 Marzo 2023].

[42] «Matplotlib,» [En línea]. Available: https://matplotlib.org/. [Último acceso: 24 Marzo 2023].

[43] «Incentro,» [En línea]. Available: https://www.incentro.com/es-ES/blog/que-es-tensorflow. [Último acceso: 24 Marzo 2023].

[44] «TensorFlow,» [En línea]. Available: https://www.incentro.com/es-ES/blog/que-es-tensorflow. [Último acceso: 24 Marzo 2023].

[45] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Keras . [Último acceso: 24 Marzo 2023].

[46] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/OpenCV . [Último acceso: 24 Marzo 2023].

[47] «Svitla,» [En línea]. Available: https://svitla.com/blog/overview-of-modern-computer-vision-tools . [Último acceso: 24 Marzo 2023].

[48] «Pypi,» [En línea]. Available: https://pypi.org/project/pytesseract/. [Último acceso: 23 Mayo 2023].

[49] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Proyecto_Jupyter. [Último acceso: 24 Marzo 2023].

[50] «Boletín Oficial del Estado,» [En línea]. Available: https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673&p=20230221&tn=1#a3-4. [Último acceso: 06 Abril 2023].

[51] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks,» de *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, Curran Associates, 2013, p. 3328.

[52] «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/guide/keras/sequential_model. [Último acceso: 8 Mayo 2023].

[53] «Genetec,» [En línea]. Available: https://www.genetec.com/es. [Último acceso: 4 Abril 2023].

[54] «Black Box,» [En línea]. Available: https://www.blackbox.com.mx/mx-mx/page/46780/Recursos/Technical/black-box-explica/Copper-Cable/Categorias-5e-y-6#:~:text=CAT6%20es%20un%20cable%20de,10%20Gigabits%20a%20distancias%20limitadas.. [Último acceso: 4 Abril 2023].

[55] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Protocolo_de_transmisi%C3%B3n_en_tiempo_real. [Último acceso: 4 Abril 2023].

[56] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/VLAN#:~:text=Una%20VLAN%20(virtual%20local%20area,en%20una%20%C3%BAnica%20red%20f%C3%ADsica.. [Último acceso: 4 Abril 2023].

[57] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Media_gateway. [Último acceso: 4 Abril 2023].

Apéndice A. ODS

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.		X		
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Los ODS u Objetivos de Desarrollo Sostenible están formados por 17 objetivos relacionados con la mejora del mundo y de la vida de todos sus habitantes. Estos objetivos aparecieron en 2015 con la aprobación por parte de la ONU de la Agenda 2030 sobre el Desarrollo Sostenible, y se ocupan de cuestiones como la pobreza, el hambre en el mundo o la igualdad de la mujer, entre otros.

En el presente trabajo se abordan, en mayor o menor medida, 3 de los 17 Objetivos de Desarrollo Sostenible: Trabajo decente y crecimiento económico (ODS 8), Industria, innovación e infraestructuras (ODS 9), y Ciudades y comunidades sostenibles (ODS11).

En primer lugar, el **ODS 8: Trabajo decente y crecimiento económico** se ocupa de mejorar los estándares de vida de los ciudadanos mediante la creación de empleos decentes, lo que a su vez proporciona un crecimiento económico inclusivo y sostenido. En este sentido, el software de reconocimiento de imágenes desarrollado es capaz de generar datos que, tratados de la forma correcta, producen información útil que puede ser utilizada para la gestión del tráfico, de recursos, de personal, etc. dentro de una empresa, y en este caso particular, del Puerto de Valencia.

De esta forma, se alcanza un beneficio económico impulsado por la utilización sostenible y responsable de recursos, la reducción de causas contaminantes provocadas por el exceso de tráfico, y la creación de puestos de trabajos relacionados con el análisis de la nueva información generada y su aplicación.

El **ODS 9: Industria, innovación e infraestructuras** hace referencia al fomento de la industrialización sostenible, de la innovación y de la construcción de infraestructuras resilientes, con el fin último de desarrollar nuevas tecnologías que posibiliten el uso eficiente de los recursos. En este contexto, el presente trabajo se centra en el estudio de una tecnología novedosa y con gran proyección de futuro como es el reconocimiento de imágenes mediante redes neuronales convolucionales, aplicada además en un entorno empresarial donde puede proporcionar beneficios económicos y relativos a la gestión de recursos.

Por último, el **ODS 11: Ciudades y comunidades sostenibles** pretende transformar las ciudades para que sean más sustentables y menos contaminantes. Como ya se ha comentado anteriormente, el software desarrollado en el presente trabajo proporciona información práctica para la gestión eficiente del tráfico y los recursos. Por tanto, mediante el uso de este software se puede conseguir que un determinado lugar, el Puerto de Valencia en este caso, sea menos contaminante con el medio ambiente y utilice mejor los recursos de los que dispone, siendo esto extrapolable a las ciudades y países de todo el mundo.

Apéndice B. Glosario de términos

(Ordenados alfabéticamente).

***AVERAGE-pooling*:** proceso de *pooling* donde se obtiene el valor resultante a partir de la media de los valores de entrada.

***Batch size*:** tamaño del *batch*, es decir, cantidad de imágenes pertenecientes a los datos de entrenamiento que se le pasan a la red neuronal. Si, por ejemplo, el tamaño del *batch* es de 32, la red neuronal recibirá los datos de 32 en 32, hasta llegar al total de datos de entrenamiento.

Capa totalmente conectada: una capa totalmente conectada (*fully connected layer*) es una capa neuronal donde cada una de sus neuronas están conectadas, por un lado, con todas las neuronas de la capa anterior, y por otro, con todas las de la capa siguiente.

Convolución: en redes neuronales, la convolución es el proceso de aplicar un filtro (*kernel*) en forma de matriz sobre otra matriz (la matriz de entrada) para obtener una nueva matriz. La aplicación del filtro consiste en el producto de los valores de su matriz por determinados valores de la matriz de entrada, y la posterior suma de los valores obtenidos.

Filtro: los filtros o *kernels* son las matrices que se aplican sobre la capa convolucional para obtener una nueva matriz. Se pueden aplicar diferentes filtros sobre una misma matriz, obteniendo diversos resultados.

Flatten: *flatten* (alisado) es el proceso de transformación de una matriz (más de una dimensión) en un vector columna (una sola dimensión).

Función de activación: nodo que indica si una neurona se activa o no (es decir, si manda una señal a la capa siguiente o no), dependiendo de los valores de entrada y la fórmula de la función de activación.

Función de activación *binary*: tipo de función de activación utilizada para casos de clasificación donde solamente existen dos clases a clasificar.

Función de activación *ReLU*: función que devuelve 0 para todos los valores de entradas negativos, mientras que para los valores iguales o mayores que 0 devuelve el mismo valor de entrada.

Función de activación *sigmoid*: empleada principalmente en casos de regresión, es una función que transforma los valores de entrada a una escala (0,1), con los valores bajos próximos a 0 y los altos a 1.

Función de activación *softmax*: tipo de función de activación utilizada en clasificación en más de dos clases, donde cada objeto pertenece exclusivamente a una clase. *Softmax* devuelve un

vector de probabilidades (desde 0 a 1), de tal forma que cada valor representa la probabilidad de que el objeto pertenezca a dicha clase.

Función de activación *tanh*: función similar a *sigmoid* pero en una escala (-1,1), donde los valores de entrada negativos devolverán un valor negativo (del 0 al -1), los positivos un valor positivo (del 0 al 1) y el valor de entrada 0 devolverá 0.

***Kernel*:** véase *Filtro.*

***Learning rate*:** tasa de aprendizaje, es decir, parámetro que define la velocidad de aprendizaje del modelo (la frecuencia con la que la red neuronal se actualiza con la nueva información aprendida).

***MAX-pooling*:** proceso de *pooling* donde el valor resultante es el máximo de los valores de entrada.

***Padding*:** en el caso de *same padding*, adición de píxeles alrededor de la matriz de entrada para mantener el tamaño en la matriz de salida tras los procesos de convolución o *pooling*. *Valid padding* supone no añadir ningún píxel y obtener una matriz resultante de menor dimensión.

***Pooling*:** el proceso de *pooling*, similar al de convolución, consiste en aplicar un filtro en forma de matriz sobre una matriz de entrada, con el objetivo de obtener una nueva matriz resultante. Su principal diferencia con la convolución es que los resultados alcanzados se consiguen aplicando la media (*AVERAGE-pooling*) o el máximo (*MAX-pooling*) de los valores originales seleccionados por el filtro.

***Stride*:** desplazamiento del *kernel* o la matriz de *pooling* dentro de la matriz de entrada. Si *stride* es 1, se moverá píxel a píxel recorriendo la matriz; si es 2, efectuará saltos de 2 píxeles; y así sucesivamente.

Testing accuracy: información proporcionada por Tensorflow que indica el porcentaje de imágenes bien clasificadas del conjunto de *testing*.

***Testing split*:** porcentaje de los datos de entrenamiento que serán utilizados en el proceso de *testing*. Un *testing split* de 0.9 indica que el 90% de los datos serán para entrenamiento, mientras el resto se dedicará al proceso de *testing*.

Validation accuracy: información proporcionada por Tensorflow que indica el porcentaje de imágenes bien clasificadas del conjunto de validación.

***Validation split*:** porcentaje de los datos de entrenamiento (una vez ya han sido separados de los de *testing*) que serán utilizados en el proceso de validación.

Apéndice C. Resultados completos de los experimentos con redes neuronales

A continuación, se muestran los resultados completos obtenidos en los diversos experimentos realizados con las diferentes arquitecturas (una red neuronal original, una basada en LeNet, otra en AlexNet y, por último, la red neuronal final). De cada arquitectura se puede encontrar el código con el que ha sido implementada y una tabla con los resultados de los experimentos en función de parámetros como el tamaño del *batch*, la tasa de aprendizaje o el número de *epochs*.

Para la correcta interpretación de las tablas es necesario comprender cada parámetro valorado, por lo que se va a explicar brevemente el contenido de cada columna:

- Img/cl.: Número de imágenes de cada clase con las que se ha entrenado el modelo.
- Nº cl.: Número de clases con las que se ha entrenado el modelo.
- Batch: Tamaño del *batch*.
- Opt.: Optimizador utilizado.
- Learning_rate: Tasa de aprendizaje.
- Epochs: Número de *epochs* (iteraciones).
- V_split: *Validation split*, es decir, porcentaje de los datos de entrenamiento (una vez separados de los de *testing*) utilizados para validación.
- T_split: *Testing split*, es decir, porcentaje de los datos de entrenamiento utilizados para *testing*. Un T_split de 0.9 indica que el 90% de los datos serán para entrenamiento, mientras el resto se utilizarán para *testing*.
- Loss: Medida que calcula la distancia entre el resultado predicho y el real durante el entrenamiento.
- Acc: Medida que calcula el porcentaje de acierto entre los resultados predichos y los reales durante el entrenamiento.
- V_loss: Medida que calcula la distancia entre el resultado predicho y el real durante la validación.
- V_acc: Medida que calcula el porcentaje de acierto entre los resultados predichos y los reales durante la validación.
- T_loss: Medida que calcula la distancia entre el resultado predicho y el real durante el *testing*.

- T_acc: Medida que calcula el porcentaje de acierto entre los resultados predichos y los reales durante el *testing*.

La aparición de asteriscos (*) en los resultados se debe a pruebas en las que no midió determinadas métricas (como el acierto y pérdida durante el *testing*) o no se tuvieron en cuenta factores (como el *testing split*). La (W) en el número de clases indica que se utilizó la técnica de pesado de clases (weighted clases), explicado en el apartado 7.1.

Red neuronal original

Arquitectura:

```
#------------ CREACIÓN DEL MODELO ------------
model = Sequential()

model.add(Conv2D(256, (3,3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(256,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(64))

model.add(Dense(4))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X,y,batch_size=32,validation_split=0.3,epochs=5)
```

Tabla de resultados:

img/cl.	Nº cl.	batch	opt.	learning_rat	epochs	v_split	t_split	loss	acc	v_loss	v_acc	t_loss	t_acc
113	4	32	Adam	0,001	3	0,3	*	*	*	0,9065	0,6544	*	*
116	5	32	Adam	0,001	3	0,3	*	0,5892	0,8025	0,6329	0,7701	*	*
223	4	32	Adam	0,001	3	0,3	*	0,2741	0,9038	0,3306	0,8619	*	*
188	5	32	Adam	0,001	3	0,3	*	0,6062	0,7869	0,5224	0,8156	*	*
238	4	32	Adam	0,001	3	0,3	*	0,3018	0,9174	0,284	0,9301	*	*
503	3	32	Adam	0,001	3	0,3	*	0,2567	0,9005	0,3994	0,819	*	*
113	6	32	Adam	0,001	10	0,3	*	0,1457	0,9515	0,5189	0,8529	*	*
253	5	32	Adam	0,001	10	0,3	*	0,0796	0,9717	0,3253	0,8816	*	*
838	4	32	Adam	0,001	5	0,3	*	0,062	0,9817	0,1361	0,9592	*	*

Arquitectura LeNet-5

Arquitectura:

```
#------------ CREACIÓN DEL MODELO ------------
model = Sequential()

model.add(Conv2D(6, (5,5), input_shape=X.shape[1:]))
model.add(Activation('tanh'))
model.add(AveragePooling2D(pool_size=(2,2)))

model.add(Conv2D(16,(5,5)))
model.add(Activation('tanh'))
model.add(AveragePooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(120))
model.add(Activation('tanh'))

model.add(Dense(84))
model.add(Activation('tanh'))

model.add(Dense(ntypes))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])

#tf.compat.v1.disable_eager_execution()

model.fit(Xtr,ytr,class_weight=class_weights_dict,batch_size=32,validation_split=0.15,epochs=18,shuffle=False)
```

Tabla de resultados:

Img/cl.	Nº cl.	batch	opt.	learning_rat	epochs	v_split	t_split	loss	acc	v_loss	v_acc	t_loss	t_acc	Modelo	Bancos
838	4	32	Adam	0,001	5	0,3	***	0,128	0,9561	0,2405	0,9185	***	***	1	C2+2
113	6	64	Adam	0,001	12	0,3	***	0,2516	0,9114	0,2438	0,8186	***	***	1	C2+2
100	10	64	Adam	0,001	16	0,3	0,9	0,0739	0,9964	0,5957	0,8208	0,3542	0,8652	1	Final
100	10	64	Adam	0,005	25	0,3	0,9	2,2097	0,0877	2,2083	0,0792	2,1965	0,146	1	Final
100	10	64	Adam	0,0005	20	0,3	0,9	0,0826	0,9964	0,5599	0,8375	0,5463	0,8315	1	Final
100	10	64	Adam	0,001	19	0,3	0,9	0,0372	1	0,7003	0,7865	0,6927	0,7676	1	Final
100	10	64	Adam	0,005	6	0,3	0,9	2,2229	0,1752	2,2162	0,1723	2,286	0,0808	1	Final
100	10	64	Adam	0,0005	18	0,3	0,9	0,1559	0,9887	0,6399	0,794	0,7071	0,7778	1	Final
Todas	10	64	Adam	0,001	15	0,3	0,9	0,1336	0,9602	0,3907	0,8865	0,656	0,8335	1	Final
Todas	10	32	Adam	0,001	15	0,3	0,9	0,0488	0,9869	0,441	0,8822	0,7219	0,8487	1	Final
Todas	10	16	Adam	0,001	15	0,3	0,9	0,1424	0,9554	0,442	0,8705	0,8187	0,8221	1	Final
Todas	10	32	Adam	0,001	6	0,15	0,9	0,1177	0,9657	0,3652	0,8861	0,6405	0,8615	1	Final
Todas	10	32	Adam	0,001	9	0,4	0,9	0,1444	0,9581	0,4017	0,8824	0,6561	0,8411	1	Final
Todas (W)	10	32	Adam	0,001	15	0,15	0,9	0,0217	0,9919	1,5135	0,8616	0,5894	0,8691	1	Final
Todas (W)	10	64	Adam	0,001	15	0,15	0,9	0,0364	0,9885	0,925	0,8823	0,5069	0,8564	1	Final
Todas (W)	10	16	Adam	0,001	15	0,15	0,9	0,0524	0,9766	0,9853	0,8766	0,5451	0,8577	1	Final
Todas (W)	10	32	Adam	0,005	15	0,15	0,9	2,3033	0,0615	2,1129	0,2335	2,0524	0,1888	1	Final
Todas (W)	10	32	Adam	0,0005	15	0,15	0,9	0,0392	0,9885	1,0392	0,8578	0,6243	0,8424	1	Final
Todas (W)	10	32	Adam	0,001	15	0,4	0,9	0,0571	0,9788	1,1432	0,8569	0,5534	0,8488	1	Final
Todas (W)	10	32	Adam	0,001	15	0,15	0,8	0,08	0,9611	1,2409	0,8644	0,604	0,8614	2	Final
Todas (W)	10	64	Adam	0,001	15	0,15	0,8	0,0803	0,9581	1,1783	0,8824	0,4405	0,8843	2	Final
Todas (W)	10	16	Adam	0,001	15	0,15	0,8	0,0338	0,9845	1,6086	0,8898	0,5571	0,9021	2	Final
Todas (W)	10	16	Adam	0,001	18	0,15	0,8	0,0108	0,9974	1,6345	0,8951	0,4111	0,9141	2	Final
Todas (W)	10	32	Adam	0,001	18	0,15	0,8	0,0552	0,9759	1,0148	0,8983	0,3587	0,9129	2	Final

Arquitectura AlexNet

Arquitectura:

```
#------------ CREACIÓN DEL MODELO ------------
model = Sequential()

model.add(Conv2D(96, (11,11), strides=(4,4), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3)))

model.add(Conv2D(256, (5,5), strides=(1,1),padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3)))

model.add(Conv2D(384, (3,3), strides=(1,1),padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())

model.add(Conv2D(384, (3,3), strides=(1,1),padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())

model.add(Conv2D(256, (3,3), strides=(1,1),padding="same"))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3,3)))

model.add(Flatten())

model.add(Dense(4096))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(4096))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(ntypes))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])

#tf.compat.v1.disable_eager_execution()
model.fit(Xtr,ytr,class_weight=class_weights_dict,batch_size=64,validation_split=0.3,epochs=20,shuffle=False)
```

Tabla de resultados:

Img/cl.	Nº cl.	batch	opt.	learning_rat	epochs	v_split	t_split	loss	acc	v_loss	v_acc	t_loss	t_acc
838	4	32	Adam	0,001	8	0,3	***	0,1145	0,9676	0,1815	0,9453	***	***
419	4	32	Adam	0,001	14	0,2	***	0,0606	0,9806	0,3998	0,878	***	***
100	10	32	Adam	0,001	20	0,2	0,9	0,1884	0,9484	8,0661	0,2562	8,8976	0,1685
100	10	64	Adam	0,001	7	0,3	0,9	0,5878	0,7871	2,9673	0,1167	12,537	0,1123
100	10	32	Adam	0,005	13	0,2	0,9	1,069	0,5915	1,7605	0,5875	3,5194	0,4831
100	10	32	Adam	0,007	15	0,2	0,9	1,4544	0,4554	1,455	0,4187	22,615	0,2696
Todas	10	32	Adam	0,005	19	0,2	0,9	0,4254	0,8673	0,5026	0,8573	0,8657	0,7916
Todas	10	64	Adam	0,005	17	0,2	0,9	0,4492	0,8578	0,5968	0,8319	1,2097	0,7318
Todas	10	16	Adam	0,005	10	0,2	0,9	0,6709	0,8125	1,0347	0,7429	1,2493	0,7268
Todas	10	32	Adam	0,005	8	0,4	0,9	0,7049	0,8005	1,1591	0,6584	1,1453	0,6455
Todas	10	32	Adam	0,005	20	0,3	0,9	0,302	0,9031	0,6581	0,8464	0,6232	0,85
Todas (W)	10	32	Adam	0,005	20	0,3	0,9	2,17	0,1298	1,8521	0,1837	1,8646	0,1817
Todas(W)	10	64	Adam	0,005	20	0,3	0,9	1,9787	0,2336	15,6706	0,279	14,39	0,2586
Todas(W)	10	16	Adam	0,005	20	0,3	0,9	2,3019	0,1035	2,3262	0,1859	2,2947	0,1918
Todas(W)	10	32	Adam	0,001	20	0,3	0,9	0,2863	0,8915	1,2276	0,8079	0,6089	0,8271
Todas(W)	10	64	Adam	0,001	20	0,3	0,9	0,2574	0,8991	1,1717	0,8216	0,5475	0,851
Todas(W)	10	64	Adam	0,001	20	0,15	0,9	0,3022	0,8968	4,3546	0,5366	2,803	0,5447
Todas(W)	10	64	Adam	0,001	20	0,4	0,9	0,2319	0,9053	3,1885	0,6437	2,0356	0,6317
Todas(W)	10	64	Adam	0,001	20	0,3	0,8	0,1775	0,9272	1,9807	0,7591	0,8564	0,7723
Todas(W)	10	64	Adam	0,001	20	0,3	0,7	0,2929	0,8983	1,1934	0,745	0,9834	0,7346

Red neuronal final

Arquitectura:

```
#------------ CREACIÓN DEL MODELO ------------
model = Sequential()

model.add(Conv2D(16, (3,3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(120))
model.add(Activation('relu'))

model.add(Dense(80))
model.add(Activation('relu'))

model.add(Dense(ntypes))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])

#tf.compat.v1.disable_eager_execution()
model.fit(Xtr,ytr,class_weight=class_weights_dict,batch_size=32,validation_split=0.15,epochs=10,shuffle=False)
```

Tabla de resultados:

Img/cl.	Nº cl.	batch	opt.	learning_rat	epochs	v_split	t_split	loss	acc	v_loss	v_acc	t_loss	t_acc
Todas (W)	10	32	Adam	0,001	10	0,15	0,9	0,0179	0,9933	1,1823	0,9228	0,358	0,9276
Todas (W)	10	16	Adam	0,001	10	0,15	0,9	0,0312	0,987	1,3628	0,8898	0,4351	0,8996
Todas (W)	10	64	Adam	0,001	10	0,15	0,9	0,0633	0,9682	1,1771	0,8632	0,6322	0,8555
Todas (W)	10	32	Adam	0,005	10	0,15	0,9	0,213	0,9037	1,7155	0,782	0,7837	0,8188
Todas (W)	10	32	Adam	0,01	10	0,15	0,9	2,3065	0,0315	2,3014	0,0185	2,3063	0,0166
Todas (W)	10	32	Adam	0,001	10	0,3	0,9	0,0652	0,9703	1,5454	0,8649	0,582	0,8772
Todas (W)	10	32	Adam	0,001	10	0,4	0,9	0,1169	0,9602	1,4295	0,848	0,6172	0,8295
Todas (W)	10	32	Adam	0,001	10	0,15	0,8	0,0576	0,9742	1,2313	0,8739	0,5004	0,8706
Todas (W)	10	16	Adam	0,001	10	0,15	0,8	0,1071	0,9586	1,2516	0,86	0,4724	0,8599
Todas (W)	10	64	Adam	0,001	10	0,15	0,8	0,098	0,9563	1,3528	0,8818	0,4641	0,8665
Todas (W)	10	32	Adam	0,001	10	0,3	0,8	0,0716	0,9634	1,3822	0,8698	0,5345	0,8742
Todas (W)	10	32	Adam	0,001	10	0,4	0,8	0,065	0,9697	1,7258	0,8598	0,5946	0,8617
Todas (W)	10	32	Adam	0,001	10	0,15	0,7	0,0444	0,9788	1,2912	0,8831	0,5953	0,8641
Todas (W)	10	16	Adam	0,001	10	0,15	0,7	0,0396	0,9836	1,3742	0,8663	0,7022	0,8462
Todas (W)	10	64	Adam	0,001	10	0,15	0,7	0,1358	0,9413	1,0482	0,8661	0,6314	0,8327
Todas (W)	10	32	Adam	0,001	10	0,3	0,7	0,0632	0,9715	1,374	0,8575	0,682	0,8331
Todas (W)	10	32	Adam	0,001	10	0,4	0,7	0,0599	0,9722	1,8812	0,8169	0,8769	0,8148

Apéndice D. Código completo

Red Neuronal Final

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
import random
import pickle
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard
import time
import pandas as pd
from sklearn.utils import class_weight
```

```
#------------ CREACIÓN DE DATOS DE ENTRENAMIENTO ------------
databasePath = r"C:\Users\dsegura\Desktop\BBDDfinal"
classes = ["CabezaTractora","CamionCisterna","CamionTolva","Furgoneta","Motocicleta",
           "Otros","Plataforma","Portacoches","Portacontenedor","Turismo"]
ntypes = len(classes)
maxImg = 3000
imageSize = 100

trainingData = []
for c in classes:
    path = os.path.join(databasePath,c)
    num = classes.index(c)
    contImg = 0

    for image in tqdm(os.listdir(path)):
        try:
            if contImg < maxImg:
                imageArray = cv2.imread(os.path.join(path,image),cv2.IMREAD_GRAYSCALE)
                imageArray = cv2.resize(imageArray,(imageSize,imageSize))
                trainingData.append([imageArray,num])
                contImg += 1
            else:
                break
        except Exception as e:
            pass
random.shuffle(trainingData)
dataSize = len(trainingData)
print(len(trainingData))

X = []
y = []

for image, c in trainingData:
    X.append(image)
    y.append(c)

X = np.array(X).reshape(-1,imageSize,imageSize,1)
X = X/255.0
y = np.array(y)

# ----- Prepararación de los datos de entrenamiento -----
tr = int (dataSize*0.9) # Porcentaje de los datos decicados a entrenamiento
Xtr = X[:tr]
ytr = y[:tr]
print("Tamaño datos de entrenamiento = ", len(Xtr))

# ----- Prepararación de los datos de testing -----
Xte = X[tr:]
yte = y[tr:]
print("Tamaño datos de testing = ", len(Xte))

# ----- Técnica de pesado de clases -----
class_weights = class_weight.compute_class_weight(class_weight='balanced',
                                                  classes=np.unique(ytr),
                                                  y=ytr)
class_weights_dict = dict(enumerate(class_weights))
```

```
#------------ CREACIÓN DEL MODELO ------------
model = Sequential()

model.add(Conv2D(16, (3,3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(120))
model.add(Activation('relu'))

model.add(Dense(80))
model.add(Activation('relu'))

model.add(Dense(ntypes))
model.add(Activation('softmax'))

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics=['accuracy'])

#tf.compat.v1.disable_eager_execution()
model.fit(Xtr,ytr,class_weight=class_weights_dict,batch_size=32,validation_split=0.15,epochs=12,shuffle=False)
```

```
results = model.evaluate(Xte,yte)
print("test loss: ", results[0])
print("test accuracy: ", results[1])
```

```
model.save('CNNfinal.model')
```


Implantación

```
import cv2
from datetime import datetime
import os
import imutils
import pytesseract
import numpy as np
import tensorflow as tf
import openpyxl
```

```
def getType(path):
    imgArray = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
    imgArray = cv2.resize(imgArray,(100,100))
    imgArray = imgArray.reshape(-1,100,100,1)
    prediction = model.predict([imgArray])
    index = np.argmax(prediction)
    return classes[index]
```

```
def checkIfLicense(screencnt,file,img):
    image = cv2.imread(file)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    croppedLP = cv2.adaptiveThreshold(gray,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11, 2)
    predicted_result = pytesseract.image_to_string(croppedLP, lang ='eng',
    config ='--oem 3 --psm 6 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789')
    if (len(predicted_result)-1 > 4) & (len(predicted_result)-1 < 10):
        return predicted_result
    else:
        return None
```

```
def readLicense(frameMat):

    # --- PREPROCESAMIENTO DE LA IMAGEN ---
    image = cv2.resize(frameMat,(600,400))
    imgBlurred =  cv2.bilateralFilter(image,11,17,17)
    gray = cv2.cvtColor(imgBlurred, cv2.COLOR_BGR2GRAY)
    sobelx = cv2.Sobel(gray, cv2.CV_8U, 1, 0, ksize = 3)
    ret2, threshold_img = cv2.threshold(sobelx, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    element = element_structure
    morph_n_thresholded_img = threshold_img.copy()
    cv2.morphologyEx(src = threshold_img,op = cv2.MORPH_CLOSE,kernel = element,dst = morph_n_thresholded_img)

    # --- EXTRACCIÓN DE CONTORNOS ---
    cnts,new = cv2.findContours(morph_n_thresholded_img,mode = cv2.RETR_EXTERNAL,method = cv2.CHAIN_APPROX_NONE)
    cnts = sorted(cnts,key = cv2.contourArea, reverse = True)
    screenCnt = None
    check = None
    for c in cnts:
        perimeter = cv2.arcLength(c,True)
        approx = cv2.approxPolyDP(c,0.018*perimeter,True)
        if len(approx) == 4:
            screenCnt = approx
            x,y,w,h = cv2.boundingRect(c)
            new_img = image[y:y+h,x:x+w]
            fileName = "./matriculasTesseract/mat.jpg"
            cv2.imwrite(fileName,new_img)
            check = checkIfLicense(screenCnt,fileName,image)
            if check is not None:
                break
    return check
```

```
def mse(img1,img2):
    h,w = img1.shape
    diff = cv2.subtract(img1,img2)
    err = np.sum(diff**2)
    mse = err/(float(h*w))
    return mse
```

```
def stream(cap):
    background = None

    wb = openpyxl.load_workbook(bbdd)
    ws = wb['Hoja1']

    while True:
        try:
            if cap.isOpened():
                time = datetime.now()
                ret, frame = cap.read()
                frame = imutils.resize(frame,width=1900)
#                 cv2.imshow('carril2',frame)

                # --- Píxeles donde se encuentra el vehículo ---
                h, w, channels = frame.shape
                y = int(h*0.04)
                Y = int (h*0.877)
                x = int (w*0.123)
                X = int (w*0.877)
                frameT = frame[y:Y,x:X]

                # --- Píxeles donde se encuentra la matrícula ---
                y = int (h*0.43)
                Y = int (h*0.72)
                x = int (w*0.4)
                X = int (w*0.66)
                frameMat = frame[y:Y,x:X]
#                 cv2.imshow('carril2Mat',frameMat)

                gray = cv2.cvtColor(frameMat,cv2.COLOR_BGR2GRAY)
                gray = cv2.GaussianBlur(gray,(21,21),0)
                if background is None:
                    background = gray
                diff = mse(background,gray)
                if diff > 5:
                    timestr = time.strftime("%Y-%m-%d-%H_%M_%S.%f")
                    mov = "C:/Users/dsegura/Desktop/proyecto/movimiento/" + timestr + ".jpg"
                    cv2.imwrite(mov,frameMat)
                    predictedMat = readLicense(frameMat)
                    if predictedMat is not None:
                        mat = "C:/Users/dsegura/Desktop/proyecto/matriculas/" + timestr + ".jpg"
                        cv2.imwrite(mat,frameMat)
                        fileName = "C:/Users/dsegura/Desktop/proyecto/vehiculos/" + timestr + ".jpg"
                        cv2.imwrite(fileName,frameT)
                        predictedType = getType(fileName)
                        print("TS: " + timestr)
                        print("MAT: " + predictedMat)
                        print("TIPO: " + predictedType)
                        print("---------" + "\n")
                        data = [time,predictedMat,predictedType]
                        newRow = ws.max_row + 1
                        for i,j in zip(range(1,4),range(3)):
                            ws.cell(column=i,row=newRow,value=data[j])
                        wb.save(filename=bbdd)

                background = gray

#                 if cv2.waitKey(20) & 0xFF == 32:
#                     break

        except Exception as e:
            print("ERROR")
            print(e)
            cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")

    cap.release()
    cv2.destroyAllWindows()
    wb.close()
```

```
if __name__ == '__main__':
    np.set_printoptions(precision=3,suppress=False)
    pytesseract.pytesseract.tesseract_cmd = r"C:\Users\dsegura\AppData\Local\Programs\Tesseract-OCR\tesseract.exe"
    element_structure = cv2.getStructuringElement(shape = cv2.MORPH_RECT, ksize =(22, 5))
    classes = ["CabezaTractora","CamionCisterna","CamionTolva","Furgoneta","Motocicleta",
               "Otros","Plataforma","Portacoches","Portacontenedor","Turismo"]
    model = tf.keras.models.load_model("CNNfinal.model")
    cap = cv2.VideoCapture("rtsp://user:password@xxx.xx.x.xxx:xxx/xxx/xxx")
    bbdd = r"C:\Users\dsegura\Desktop\BBDD.xlsx"
    stream(cap)
```

Comprobación de matrículas

```
import openpyxl
from difflib import SequenceMatcher
```

```
wbbd = openpyxl.load_workbook(r"C:\Users\dsegura\Desktop\BBDD.xlsx")
wbcit = openpyxl.load_workbook(r"C:\Users\dsegura\Desktop\InformeLprAccesos_13_06_2023.xlsx")
wsbd = wbbd['Hoja1']
wscit = wbcit['Informe LPR Accesos']
wsres = wbbd['Res']


while True:
    row = wsbd[2]
    if not all(cell.value for cell in row):
        break

    f = wsbd.cell(row=2,column=1).value
    m = wsbd.cell(row=2, column=2).value.replace(" ","")
    t = wsbd.cell(row=2,column=3).value
    for j in range(6,wscit.max_row):
        if wscit.cell(row=j,column=4).value == "11S2":
            fecha = wscit.cell(row=j,column=5).value
            dif = abs((f-fecha).total_seconds()/60.0)
            if dif < 4:
                matricula = wscit.cell(row=j,column=6).value.replace(" ","")
                if SequenceMatcher(None,matricula,m).ratio() >= 0.5:
                    matAnterior = wsres.cell(row=wsres.max_row,column=2).value
                    if matAnterior != matricula:
                        newRow = wsres.max_row+1
                        wsres.cell(row=newRow,column=1,value=f)
                        wsres.cell(row=newRow,column=2,value=matricula)
                        wsres.cell(row=newRow,column=3,value=t)

    wsbd.delete_rows(2)


wbbd.save(filename=r"C:\Users\dsegura\Desktop\BBDD.xlsx")
wbbd.close()
wbcit.save(filename=r"C:\Users\dsegura\Desktop\InformeLprAccesos_13_06_2023.xlsx")
wbcit.close()
```

Extracción de fotogramas

```
import cv2
import os

# Frames por segundo del vídeo original
realFPS = 15

# Frames que queremos obtener por cada segundo
wantedFPS = 1

FPS = realFPS/wantedFPS - 1

# Función para la extracción de frames
# pathVideo: ruta del vídeo original
# pathFolder: ruta de la carpeta donde se quiere guardar los frames
def FrameCapture(pathVideo, pathFolder):

    # Ruta del vídeo
    vidObj = cv2.VideoCapture(pathVideo)

    # Contador (para indicar el número del frame)
    count = 0

    # Contador de FPS (para obtener la cantidad deseada de frames por cada segundo)
    FPScount = 0

    # comporbar si se ha extraído un frame
    success = 1

    while success:

        # Llamada a la función read() para extraer los frames
        success, image = vidObj.read()

        # Entrará en función de los frames que queramos obtener de cada segundo
        if FPScount == FPS:
            # Recorta la imagen para quedarnos con el vehículo
            h, w, channels = image.shape
            y = int(h/6)
            Y = int ((3*h)/4)
            x = int (w/4)
            X = int ((3*w)/4)
            image = image[y:Y,x:X]
            # Guarda el frame con su contador
            cv2.imwrite(os.path.join(pathFolder,"300323_13h00_frame%d.jpg" % count), image)
            count += 1
            FPScount = 0
        else:
            FPScount += 1

# Driver Code
if __name__ == '__main__':

    # Llamada a la función
    FrameCapture(r"C:\Users\dsegura\Desktop\pruebaBancoImagenes\videosFuente\300323_13h00.mp4",
                 r"C:\Users\dsegura\Desktop\pruebaBancoImagenes\imagenes")
```