Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Nombre del Centro

Trabajo Fin de Grado

SEGUIMIENTO DE VEHÍCULOS EN SECUENCIAS DE VÍDEO A PARTIR DE IDENTIFICACIÓN DE MATRÍCULAS

Alumno: Juan Gallego Güeto

Tutor: Prof. D. Francisco Charte Ojeda
Dpto: Informática

Junio, 2020

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Francisco Charte Ojeda, tutor del Proyecto Fin de Carrera titulado: **Seguimiento de vehículos en secuencias de vídeo a partir de identificación de matrículas**, que presenta Juan Gallego Güeto, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2020

El alumno:

GALLEGO GUETO JUAN - 77384960B

Firmado digitalmente por GALLEGO GUETO JUAN - 77384960B
Fecha: 2020.06.10 00:16:04 +02'00'

El tutor:

CHARTE OJEDA FRANCISCO - 25996290B

Firmado digitalmente por CHARTE OJEDA FRANCISCO - 25996290B
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-25996290B, givenName=FRANCISCO, sn=CHARTE OJEDA, cn=CHARTE OJEDA FRANCISCO - 25996290B
Fecha: 2020.06.09 21:36:19 +02'00'

Índice general

Índice de figuras

Capítulo 1

Introducción

En este apartado se presenta la motivación para desarrollar este Trabajo Fin de Grado. A continuación se muestran los objetivos a cumplir, la metodología seguida y la estructura de este documento.

1.1. Motivación

Las personas nos caracterizamos por buscar nuevas vías que mejoran nuestras condiciones de vida. Un claro ejemplo fue Charles Babbage [1], profesor de matemáticas de la Universidad de Cambridge, que presentó su idea revolucionaría del primer ordenador numérico del mundo. La máquina fue inicialmente descrita en el año 1835, pero estuvo refinándola toda su vida. El diseño se basó en el telar de Joseph Marie Jacquard [2], el cual usaba tarjetas perforadas para realizar diseños en el tejido. Babbage la adaptó para conseguir calcular funciones analíticas, su intento no fue fructuoso, pero desde entonces los desarrollos en este campo han tenido un auge espectacular y se ha conseguido implementar algoritmos que resuelven fácilmente multitud de problemas que antes resultaban muy engorrosos de resolver.

Los desarrollos actuales se centran en el estudio de las habilidades humanas e intentan emular características propias de las personas. Justamente esto tratan de realizar

las redes neuronales, imitar la capacidad de memorizar y asociar hechos. Las redes neuronales no son más que un modelo artificial y simplificado del cerebro humano, capaz de adquirir conocimiento a través de la experiencia y cuya unidad básica de procesamiento es la neurona.

Al desarrollar una tarea con estos avances mejoramos la calidad y aumentamos la producción. Por ejemplo, en un parking se forman muchas colas en su entrada porque un operario tiene que apuntar la matrícula y la hora de llegada de los vehículos. Se podría agilizar este proceso si se identifica automáticamente la matrícula y la hora de llegada del vehículo. La automatización de este proceso no eliminaría el puesto de trabajo del operario. Este tendría que seguir revisando que todo funcione correctamente.

Por este hecho y como motivación personal, surge la idea de usar las redes neuronales para la detección de matrículas y su posterior lectura. La resolución de este problema tiene aplicaciones en gasolineras (para verificar si el coche está en alguna lista negra), en ayuntamientos (para controlar qué vehículos entran en el carril bus/taxi, para sistemas de radar o para controlar los vehículos que se salten los semáforos en rojo), en aparcamientos (para automatizar la forma de pago), en peajes o en un largo etcétera.

1.2. Objetivos

En esta sección expondremos los objetivos necesarios para la resolución del problema. Estos han sido divididos en dos partes: el objetivo general, donde se plantea el problema, y los objetivos específicos, los cuales nos informan de la metodología seguida a lo largo del trabajo.

1.2.1. Objetivo general

El objetivo general es tratar de resolver eficientemente la detección de las matrículas en fotos y vídeos. Tras la identificación, se procederá a su lectura. Para su correcta realización surgen los siguientes problemas:

- La complejidad de su identificación se basa en la gran cantidad de tipos de matrículas que existen. La posición y tamaño de la matrícula varía dependiendo del tipo de vehículo y del país de matriculación.
- Conseguir una extensa base de datos no es una tarea sencilla debido a que las matrículas están protegidas por la Ley de protección de datos.
- La iluminación y calidad de las imágenes no suele ser la más favorable para su lectura, teniendo en cuenta que la ejecución del sistema en un entorno real dependerá de la luz natural, perspectiva o distancia.

1.2.2. Objetivos específicos

El anterior objetivo general se estructurará en los objetivos específicos que se detallan a continuación:

→ Estudio de técnicas de aprendizaje profundo, en particular las CNN (*Convolutional Neural Network* detallado en el capítulo 4) para detección de matrículas.

→ Preprocesamiento de imágenes aplicando filtros, reduciendo los canales de color o buscando los contornos de las matrículas para facilitar su lectura.

→ Implementar el sistema utilizando una técnica de aprendizaje profundo (*Deep Learning*, DL) apropiada. Para seleccionar esta técnica previamente se estudiarán las alternativas actuales.

→ Estudio de técnicas para la lectura de matrículas. Se expondrán las ventajas y desventajas de cada una.

→ Diseño de una interfaz en un lenguaje que se adecue a la técnica de aprendizaje empleada y donde se muestren las matrículas.

→ Elaboración de la memoria del Trabajo fin de grado.

1.3. Estructura del documento

La estructura de la memoria se divide en ocho capítulos. Para tener una visión general del documento se explicará a continuación brevemente el contenido de cada uno de ellos.

Capítulo 1. Introducción

Se trata del apartado actual donde se presentan la motivación, objetivos y estructura del documento.

Capítulo 2. Contextualización

En este capítulo se enuncian las fases necesarias para la identificación de objetos. A continuación se muestran las diferentes técnicas para resolver el problema y la justificación de su elección.

Capítulo 3. Estudio y elección del modelo de CNN

Presentación del campo de trabajo para facilitar su comprensión. También se expondrán las ventajas y desventajas de los modelos actuales de aprendizaje y se justificará la elección.

Capítulo 4. Creación de la base de datos

En este apartado se presenta el proceso seguido para la obtención de las imágenes y su uso para el entrenamiento de los modelos. Seguidamente se exponen las dos formas empleadas para el etiquetado de las imágenes.

Capítulo 5. Diseño e implementación del sistema de aprendizaje

En este capítulo se verá cómo llevar a cabo el entrenamiento y se explicarán los parámetros necesarios para determinar la posición de la placa de matrícula (o placas de matrículas) en una imagen. También se mostrarán las pruebas realizadas y se detallarán los resultados obtenidos.

Capítulo 6. Lectura de las matrículas

En este apartado se expondrán las técnicas actuales de lectura de las matrículas y pruebas realizadas con cada una de ellas.

Capítulo 7. Diseño de la interfaz

Selección del lenguaje de programación que más se adapte a los requisitos del problema. Además, se enumerarán las plataformas viables para el desarrollo de la interfaz gráfica de usuario (GUI). Finalmente, se presentará la aplicación finalizada y su modo de uso.

Capítulo 8. Futuras mejoras y conclusiones

En esta sección se mostrarán las potenciales mejoras en futuras implementaciones y conclusiones obtenidas tras la finalización del proyecto.

Capítulo 2

Contextualización

En este capítulo se presenta el problema en cuestión, las fases que lo componen y los desafíos que plantea. Finalmente se proponen los métodos que hay actualmente para poder resolver el problema y cuál es el más apropiado para su elección.

2.1. La Inteligencia Artificial y reconocimiento de imágenes

La Inteligencia Artificial (IA) [3] está dando lugar en los últimos tiempos a nuevas herramientas y aplicaciones. Un área de la IA es el reconocimiento de imágenes donde se están haciendo importantes avances. Algunas de las técnicas usadas en el reconocimiento de imágenes para tareas de clasificación y detección de objetos son más precisas que los propios humanos.

Algunos casos de uso que plantea el reconocimiento de imágenes son:

Identificación de usuarios basados en su rostro: esto puede ser utilizado en las industrias para autentificación del personal o seguridad.

Análisis de opiniones: a través de su expresión facial se puede llegar a detectar la experiencia de compra en tiendas físicas.

Diagnóstico de enfermedades: mediante una imagen médica se puede llegar a conocer la enfermedad de un paciente en base a comparaciones de diagnósticos previos en otros pacientes.

Detección de matrículas: puede ser utilizada en aparcamientos para controlar la entrada y salida de vehículos o por temas de seguridad. En este trabajo nos vamos a centrar en este caso de uso. Más adelante se expondrá en profundidad la detección de matrículas y sus aplicaciones.

El reconocimiento de imágenes y más concretamente la detección de objetos puede identificar la presencia y la ubicación de un objeto dentro de una imagen. Los algoritmos que implementan este tipo de inteligencia artificial, devuelven en su salida una imagen con los objetos detectados dentro de unos recuadros. Pero la detección de objetos plantea muchos desafíos. Los principales retos que presenta son:

- **Velocidad**: los algoritmos de detección de objetos deben clasificar y localizar con precisión los elementos que desean encontrar, pero además, tienen que ser rápidos prediciendo los objetos para poder ser utilizados en tiempo real.
- **Múltiples escalas espaciales**: a través de su expresión facial se puede llegar a detectar la experiencia de compra en tiendas físicas.

2.2. Fases para la identificación de objetos

La identificación de objetos [4] es capaz de proporcionar información para entender semánticamente las imágenes. Es aplicada para la clasificación de imágenes, análisis del comportamiento de los humanos [5], reconocimiento facial [6] o conducción autónoma [7] entre otras aplicaciones. Sin embargo debido a las grandes variaciones que sufren las imágenes o vídeos como los cambios de luz o los diferentes puntos de vista del objeto, hace que el proceso de identificación sea complicado. El mayor problema de la detección de objetos es determinar dónde se encuentra localizado el objeto en la imagen, para resolver esto podemos dividirlo en tres fases:

- **Selección de la región informativa:** como muchos objetos pueden aparecer en distintas posiciones en una imagen y tener tamaños diferentes, es necesario escanear la imagen entera. Con esta estrategia se pueden encontrar todas las posibles posiciones de los objetos, pero es muy exhaustiva.

- **Extracción de características:** para reconocer diferentes objetos es necesario extraer características visuales que los distingan, que permitan darle una representación robusta y sistemática. Alguna de las características mas representativas son:

  - SIFT (*Scale Invariant Feature*) [8]: encuentra puntos clave distintivos que son invariables para la ubicación, escala, rotación, y transformaciones robustas como cambios en la iluminación. Después, estos puntos se comparan con la base de datos de características obtenida de las imágenes de entrenamiento. Si los puntos coinciden, significa que puede ser el objeto que estemos buscando.

  - HOG (*Histograms of Oriented Gradients*) [9]: preprocesa la imagen para obtener de cada pixel el vector gradiente. A partir del vector gradiente obtenemos un histograma con las características de los colores de los píxeles y así construir el clasificador de objetos.

  - Haar-like [10]: considera las regiones rectangulares adyacentes, suma las intensidades de píxeles en cada región y calcula la diferencia entre estas sumas. Esta diferencia se usa para clasificar las subsecciones de una imagen. Una de las ventajas de Haar-like es su velocidad de cálculo en tiempo constante.

- **Clasificación:** un clasificador es necesario para distinguir un objeto de todas las posibles categorías. Algunos algoritmos que que realizan esta función son: **las máquinas de vectores de soporte** o SVM (*Supported Vector Machine*) [11] , dado un conjunto de entrenamiento se etiquetan las clases y se forma una frontera de decisión con pocos conocimientos de los datos y **AdaBoost** (*adaptive boosting*) [12], está basado en la idea del '`Boosting`' el cual crea una regla de predicción combinando muchas reglas que son más imprecisas. Adaboost puede ser utilizado para

mejorar otros algoritmos ya que funciona eligiendo un algoritmo base y en cada iteración se actualizan los pesos, por lo que finalmente tenemos una suma ponderada de los resultados de n algoritmos base.

2.3. Alternativas para resolución del problema

El objetivo principal del trabajo es la identificación de las matrículas de los coches y su posterior lectura. Después de un estudio de las diferentes posibilidades de resolución encontramos tres posibles alternativas que se ajustan a nuestro problema: utilizar el algoritmo SURF, Haar Cascade o aprendizaje profundo. A continuación serán detalladas estas soluciones.

2.3.1. Algoritmo SURF

El algoritmo de visión por computador SURF (*Speed Up Robust Features*) [13], procede del algoritmo SIFT [14] propuesto por Lowe en 2004, consiste en obtener una réplica de la imagen del mismo tamaño pero con el ancho reducido calculando la diferencia Gaussiana a partir del filtro Gaussiano. La diferencia Gaussiana consiste en ejecutar en la imagen dos desenfoques, con diferentes radios. De esta forma se crea el conocido `Scale-Space` que consigue un efecto de borrosidad que asegura que los puntos de interés son invariantes ante rotación, escalado, deformación, cambios de iluminación, puntos de vista y ruido. El algoritmo SURF es más eficiente computacionalmente que SIFT, aunque es menos robusto.

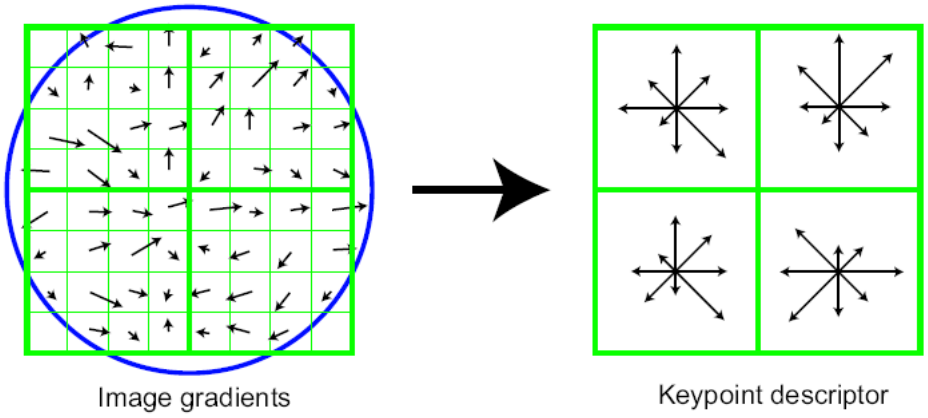

Figura 2.1: Cálculo de la Diferencia Gaussiana. (Fuente: algoritmo SIFT: fundamento teorico [15])

Para la detección de las matrículas podríamos buscar los puntos de interés de estas y compararlas con las imágenes o vídeos que le pasemos. Al utilizar el algoritmo SURF (más rápido que SIFT), podría ser posible hacerlo para vídeos en tiempo real.

2.3.2. Algoritmo Haar Cascade

Otra forma posible de realizar la detección de las matrículas sería con las características Haar-like Cascade [16], desarrollado por Paul Viola y Michael Jones. Para poner en práctica este algoritmo se necesitan imágenes positivas (donde aparezca el objeto que queremos detectar) y negativas (donde no aparezca el objeto, pero que puedan llegar a ser confusas) para así entrenar el clasificador. Después de que se entrene necesitamos extraer rasgos utilizando las características de Haar, que considera regiones rectangulares vecinas situadas en ubicaciones específicas y calcula la diferencia de intensidades luminosas entre ellas. Un ejemplo de aplicación es por ejemplo en la identificación de rostros humanos. En los rostros podemos encontrar muchos rasgos distintivos como las áreas de alrededor de los ojos que son más oscuras que las áreas de las mejillas.

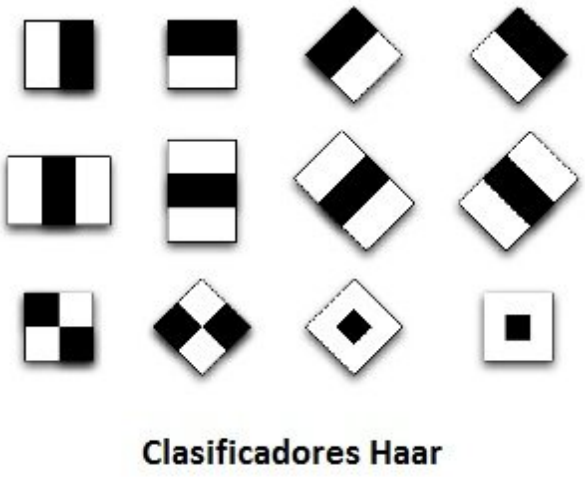

Figura 2.2: Carácterísticas Haar. (Fuente: Unipython)

Figura 2.3: Características Haar en un rostro. (Fuente: robologs.net)

2.3.3. Aprendizaje Profundo

Como última alternativa, sin que esto implique un peor rendimiento, tenemos el aprendizaje profundo [17], es una rama del aprendizaje automático que hace uso de las redes neuronales. Muchos algoritmos tradicionales de aprendizaje automático tienen una capacidad finita de aprendizaje independientemente de la cantidad de datos que adquieran. El aprendizaje profundo puede mejorar su rendimiento al poder acceder a un mayor número de datos y por lo tanto el sistema adquiere más experiencia. Las redes de aprendizaje profundo aprenden mediante la detección de estructuras complejas en los datos que reciben.

A su vez el aprendizaje profundo se subdivide en otras categorías, cada una de estas categorías ha sido desarrollada y perfeccionada para resolver diferentes problemas. Para desempeñar tareas que impliquen el uso de imágenes se han diseñado capas específicas. Un modelo que utiliza estas capas son las conocidas como CNN (*Convolutional Neural*

Networks). Las CNN procesan sus capas imitando al cortex visual del ojo humano para identificar características en las entradas. Para ello se compone de capas ocultas que están especializadas y tienen una jerarquía, es decir, en las primeras son capaces de detectar lineas, curvas y se van especializando hasta llegar a las capas más profundas que reconocen formas más complejas.

2.4. Elección final

A continuación se muestran las ventajas y desventajas de cada alternativa. Esto nos ayuda a elegir la que más se ajuste a nuestro problema.

Alternativas	**Ventajas**	**Desventajas**
Aprendizaje profundo	Buena precisión	La calidad depende de la base de datos
	Robustez ante variaciones	Tiempo de entrenamiento largo
	Sencillo de implentar con ciertas librerías	
SURF	Robusto en transformaciones	Alto esfuerzo computacional
		Presenta problemas con cambios de ángulos
Haar Cascade	Opera rápido	Sensible a cambios de iluminación
		Efectivo solo en imágenes de objetos frontales

Las ventajas y desventajas descritas en la tabla superior están centradas en nuestro problema a resolver, se han excluido aquellas que no influyen en la detección de objetos. Aunque el Algoritmo SURF, sea invariante a la posición, ángulo o escala de los objetos, es sensible al desenfoque y se obtienen resultados negativos en espacios en los que haya baja luminosidad. Por lo tanto no es una posible opción, porque vamos a tener imágenes donde la iluminación no va a ser la deseada y pueden llegar a estar desenfocadas al utilizarlo en vídeos.

Utilizar las características Haar Cascade puede ser una buena opción para detección de objetos, pero en este caso, al querer identificar matrículas de diferentes países, las características varían al no ser iguales el número de dígitos, letras o símbolos.

Figura 2.4: Matrícula Española

Figura 2.5: Matrícula de Croacia

Finalmente después del estudio de las distintas alternativas, nos decantamos por elegir las redes neuronales. El aprendizaje profundo y concretamente las redes neuronales, son la mejor opción si queremos obtener los mejores resultados. Después de entrenar la red y obtener los pesos, la identificación es muy rápida. Además la precisión es buena si la base de datos tiene suficientes imágenes con calidad e iluminación deseada.

Capítulo 3

Estudio y elección del modelo de CNN

Aunque las redes neuronales no pretenden modelar exactamente el comportamiento fisiológico de las neuronas, sí que intentan imitar el funcionamiento del cerebro humano. A continuación, en las secciones de este capítulo, se describe el modelo básico de una ANN (*Artificial Neural Networks*), se presentan diferentes arquitecturas de aplicación general y se introduce el funcionamiento de las redes más usadas para el aprendizaje a partir de imágenes: las CNN.

3.1. Modelo neuronal

El cerebro humano está formado por neuronas. Estas se componen de un núcleo y su sinapsis (axón y dendritas), que realiza la comunicación con las demás neuronas.

Cuando el cerebro humano recibe un estimulo activa neuronas que liberan un componente químico que viaja a través del axón hasta las dendritas de otras neuronas. Estas últimas se activan si superan un umbral y vuelven a repetir el proceso. Como podemos ver en la figura 3.1, las neuronas (x) envían señales de entrada al núcleo de nuestra neurona de interés (y). Los pesos sinápticos (w) situados en las dendritas de la neurona se

multiplican con los valores de entrada. Si la suma de esos valores supera un umbral la neurona se activa.

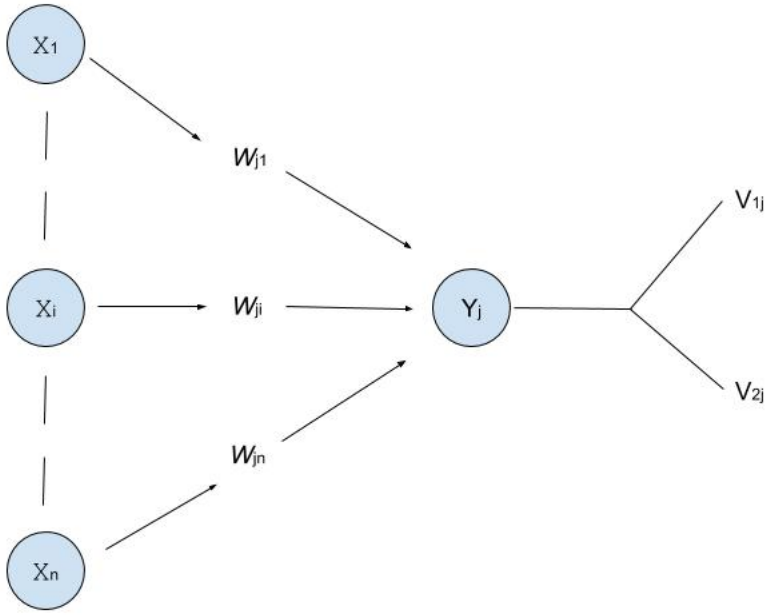

Figura 3.1: Esquema de una neurona. (Fuente: elaboración propia)

3.1.1. Clasificación de redes según el método de aprendizaje

Aunque las redes neuronales artificiales no pretenden modelar exactamente el comportamiento de las neuronas, el proceso es bastante similar. La clave de la gran flexibilidad

que nos proporcionan las redes neuronales se debe al aprendizaje o entrenamiento automático. Este se puede dividir en tres tipos: supervisado (con profesor), no supervisado (sin profesor) o aprendizaje por refuerzo.

- **Aprendizaje supervisado** [18]: es similar a como nos enseñan de pequeños. Vamos a mostrar el proceso con un ejemplo. El profesor dispone de N pares de entrenamiento compuestos por una entrada y su correspondiente respuesta correcta, por ejemplo las letras del abecedario y su pronunciación. Le enseñamos al chico la letra 'A' y se le pregunta que nos diga de qué letra se trata. Al introducir una entrada esperamos que la red responda con una salida. El chico responde que es la letra 'B', al no ser cierto esto, se le corrige diciendo que esa letra es la 'A'. La salida se compara con la respuesta correcta, si no coincide creamos una señal de error que corrige la sinapsis de la red. El aprendizaje se detiene cuando alcanza un cierto nivel de convergencia, que puede resultar de que ya no se consigue mejorar más, a pesar de hacer más iteraciones, o bien se ha alcanzado un umbral de error mínimo predeterminado. Las aplicaciones que tiene este tipo de aprendizaje son muy variadas como la clasificación de elementos (por ejemplo de vehículos), asociación de patrones (como la identificación de matrículas) o la aproximación de funciones.

- **Aprendizaje no supervisado** [19] : en este caso no se tienen referencias previas donde apoyarse, es decir que los datos no tienen que estar etiquetados. A través de las entradas que recibe, solo se describe la estructura de los datos. Posteriormente, se produce un agrupamiento, en él, se intenta buscar grupos que maximicen su similitud. Esto lo diferencia de la clasificación, en la que las clases ya están diferenciadas. Algunas aplicaciones del aprendizaje no supervisados son: la segmentación de conjuntos de datos por atributos compartidos o detección de anomalías que no encajan en ningún grupo.

- **Aprendizaje por refuerzo** [20] : dependiendo de las acciones que realiza el sistema, se proporcionan entradas en forma de recompensa o castigo para que ante una situación parecida repita la acción recompensada. Este tipo de aprendizaje se aplica

en las disciplinas de teoría de juegos, teoría de control, investigación de operaciones, teoría de la información, en estadística y en algoritmos genéticos.

3.1.2. Arquitectura

La arquitectura de las redes neuronales [21] no solo se refiere al número de capas o número de neuronas en cada capa. También hace referencia a la conexión entre las capas y las neuronas. A continuación vamos a exponer las arquitecturas más utilizadas.

3.1.2.1. Red neuronal monocapa

La red neuronal monocapa o perceptrón [22] se corresponde con la red neuronal más simple, se compone por una capa de neuronas que proyectan las entradas a las neuronas de salida. La salida se obtiene sumando los pesos de las entradas y comprobando el valor con la función de activación. El percetrón simple puede ser usado para tareas sencillas, como la clasificación binaria.

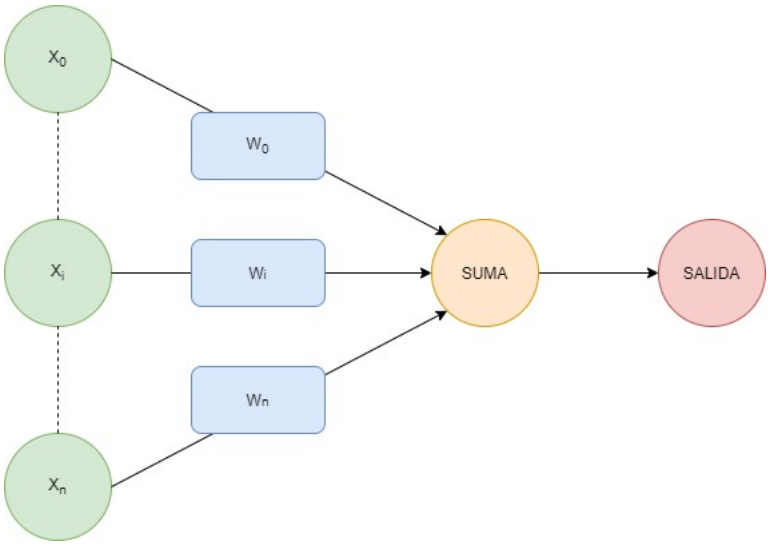

Figura 3.2: Perceptrón. Celdas de entradas (verde), pesos (azul), función de activación (naranja). (Origen: elaboración propia)

3.1.2.2. Red neuronal multicapa

La red neuronal multicapa o perceptrón multicapa [23] es una generalización de la red neuronal monocapa cuya diferencia se basa en que dispone de capas intermedias o capas ocultas entre la capa de entrada y la de salida. Esta arquitectura es capaz de resolver problemas que no son linealmente separables, siendo esa la limitación más importante de la red monocapa. Las conexiones tanto en el perceptrón simple como en el perceptrón multicapa son conexiones acíclicas dirigidas (siempre hacia adelante, sin ciclos ni conexiones entre neuronas de la misma capa), que conectan todas las unidades de una capa con todas las de la siguiente (fully connected). Los pesos sinápticos del perceptrón pueden ser fijados o adaptados iteración tras iteración.

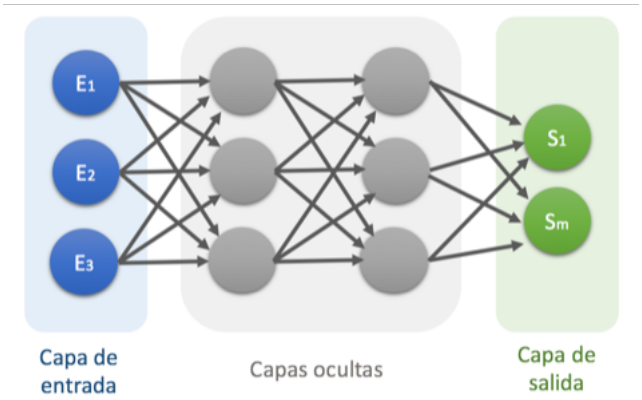

Figura 3.3: Red neuronal multicapa. (Origen: diegocalvo.es)

3.1.2.3. Red neuronal convolucional

La red neuronal convolucional (CNN) reduce el número de neuronas y la complejidad computacional necesaria para su ejecución, debido a que cada neurona no se une con todas las capas siguientes, sino que se especializa en un subgrupo. Es utilizada en análisis de imágenes.

Como esta arquitectura será la empleada en nuestro trabajo será explicada con más profundidad en los siguientes apartados.

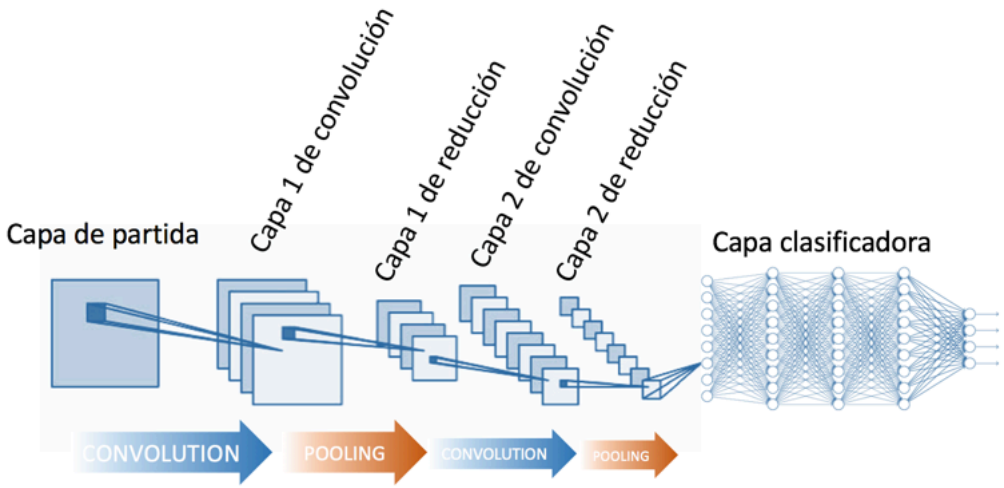

Figura 3.4: Red neuronal convolucional. (Origen: diegocalvo.es)

3.1.2.4. Red neuronal recurrente

La red neuronal recurrente (RNN) [24] elimina la estructura de capas y permite una conexión arbitraria entre neuronas formando un grafo dirigido. Esto consigue que la red tenga de cierta forma memoria y sean adecuadas para procesamiento de texto manuscrito o del habla. Esta memoria puede recordar información relevante sobre la entrada que recibieron, lo que les permite ser más precisas en la predicción de lo que vendrá después. Esto lo diferencia de las otras arquitecturas que mientras esta mantiene información de contexto, las demás no pueden recordar el pasado, excepto lo reflejado en el entrenamiento a través de sus pesos.

3.1.2.5. Red neuronal de base radial

Las redes de base radial (RBF) [25] presentan la misma arquitectura que los perceptrones multicapa, se caracterizan por estar formadas por una única capa oculta. Estos calculan la salida en función a la distancia con un centro permitiendo así que no presenten mínimos locales.

3.2. Modelos de redes convolucionales

Como hemos visto anteriormente la arquitectura de las redes convolucionales son más eficientes que otras arquitecturas. En ellas se distinguen tres capas: las convolucionales (identifican patrones gráficos), capas de pooling (tiene como objetivo la clasificación de los datos) y las capas totalmente conectadas. Las CNN son utilizadas para el análisis de imágenes, aunque estas pueden seguir siendo demasiado lentas y computacionalmente muy caras. A continuación veremos el funcionamiento de las CNN y otros métodos más eficientes que tienen como base las redes convolucionales.

3.2.1. Proceso de convolución

Las convoluciones [26] consisten en tomar grupos de píxeles cercanos de una imagen de entrada e ir operando matemáticamente con un `kernel`. Esto genera una nueva matriz de salida que será una nueva capa de neuronas ocultas. En el proceso de convolución, se aplican muchos kernels (llamados `filtros`), estos nos devuelven muchas matrices de sálida que se conocen como `mapa de características` y que nos ayudan posteriomente a distinguir un objeto de otro.

Después de aplicar la convolución se le aplica a los mapas de características una función de activación. Esta se encarga de devolver una salida a partir de una imagen de entrada. Las principales funciones de activación utilizadas en las CNN son:

- Sigmoid o Sigmoide [27]: transforma los valores introducidos a una escala (0,1). Con esta función se obtiene buen rendimiento en la última capa. Aunque tienen lenta convergencia y se puede perder información al tratar con imágenes, ya que modifican las entradas en un rango muy cerrado.
- ReLu (*Rectified Lineal Unit*) [28]: transforma los valores introducidos poniendo a 0 los valores negativos y dejando los positivos tal y como entran. Se comporta bien con las imágenes pero se pueden morir demasiadas neuronas al activarse solo si son positivos.

- Leaky ReLu [29]: esta función es similar a la anterior con la diferencia de que cambia los valores negativos multiplicándolos por un coeficiente rectificativo.

En un proceso de convolución se genera una gran cantidad de neuronas, ya que si utilizamos 32 filtros obtenemos 32 matrices de salida que por ejemplo en una imagen pequeña de 28x28x1 nos da un total de 25088 neuronas. Antes de volver a realizar una nueva convolución es necesario reducir el tamaño de las imágenes filtradas pero manteniendo las características más importantes que se detectaron en cada filtro. Este proceso es llamado `subsampling`, y el tipo más usado es el `Max-Pooling` que consiste en ir recorriendo las imágenes obtenidas de la convolución tomando más de 1 pixel y almacenando en una nueva imagen el valor más alto de esos píxeles.

3.2.2. Técnicas actuales para detección de objetos

El objetivo de la detección es localizar y clasificar objetos de cualquier imagen y delimitarlos en rectángulos.

La detección de objetos se puede llevar a cabo de dos formas [4]. La primera puede ser siguiendo el **flujo tradicional de la detección de objetos** (primero generar regiones propuestas y después clasificarlas en diferentes categorías), la otra forma considera la detección de objetos como un problema de **regresión o de clasificación**, para después llegar a un unificado resultado final.

Los principales métodos de detección de regiones son: R-CNN (*Region Convolutional Neural Network*) [30], SPP-net (*Spatial Pyramid Pooling Network*) [31], Fast R-CNN (*Fast Region Convolutional Neural Network*) [32], Faster R-CNN (*Faster Region Convolutional Neural Network*) [33], R-FCN (*Region-based Fully Convolutional Network*) [34] y Mask R-CNN [35]. La mayoría de ellos están relacionados entre sí. Los métodos que utilizan regresión o clasificación son: MultiBox [36], AttentionNet [37], G-CNN (*Grid Convolutional Neural Network*) [38], YOLO (*You Only Look Once* [39] [40], SSD (*Single Shot*

MultiBox Detector) [41], YOLOv3 [42], DSSD (*Deconvolutional Single Shot Detector*) [43] y DSOD (*Deeply Supervised Object Detector*) [44]

3.2.2.1. Algoritmos para detección de objetos a través de regiones propuestas

Los métodos propuestos se basan en examinar el escenario entero y luego se centran en regiones de interés.

R-CNN (regiones de redes neuronales convolucionales) [30]: fue propuesto por Ross Girshick en 2014 y obtuvo una media de precisión (mAP) del 53,3 % en la competición PASCAL VOC de 2012 [6] que supuso una mejora de más del 30 % con respecto a los anteriores métodos. Este método se puede dividir en 3 etapas:

- Generación de posibles regiones.
- CNN basadas en extracción profunda de características.
- Clasificación y localización.

SPP-net [31]: el modelo R-CNN tenía un problema técnico en su entrenamiento y validación. R-CNN deforma y recorta la posible región porque requiere que las imágenes tengan un tamaño definido. Sin embargo el objeto a identificar puede existir en la región que se ha recortado o puede tener una distorsión geométrica debida a la deformación realizada. Estas distorsiones pueden reducir la precisión del método. Lo que hace SPP-net para resolver este problema es considerar la teoría de la combinación de la pirámide espacial (Spatial Pyramid Matching). Esta teoría consiste en dividir la imagen en varias particiones donde se agregan características locales.

La arquitectura de SPP-net es diferente con respecto a R-CNN ya que reutiliza los mapas de características de la quinta capa para las posibles propuestas de regiones. SPP-net obtiene mejores resultados que R-CNN y además mejora la eficiencia en el periodo de entrenamiento.

Fast R-CNN [32]: aunque SPP-net consiga mejoras en exactitud y eficiencia respecto a R-CNN sigue teniendo algunos defectos. SPP-net tiene un gasto adicional en el espacio de almacenamiento.

La arquitectura de Fast R-CNN es similar a SPP-net, la imagen entera es procesada con capas de convolución para producir mapas de características. Después, un vector de características de tamaño fijo es extraído de cada posible región.

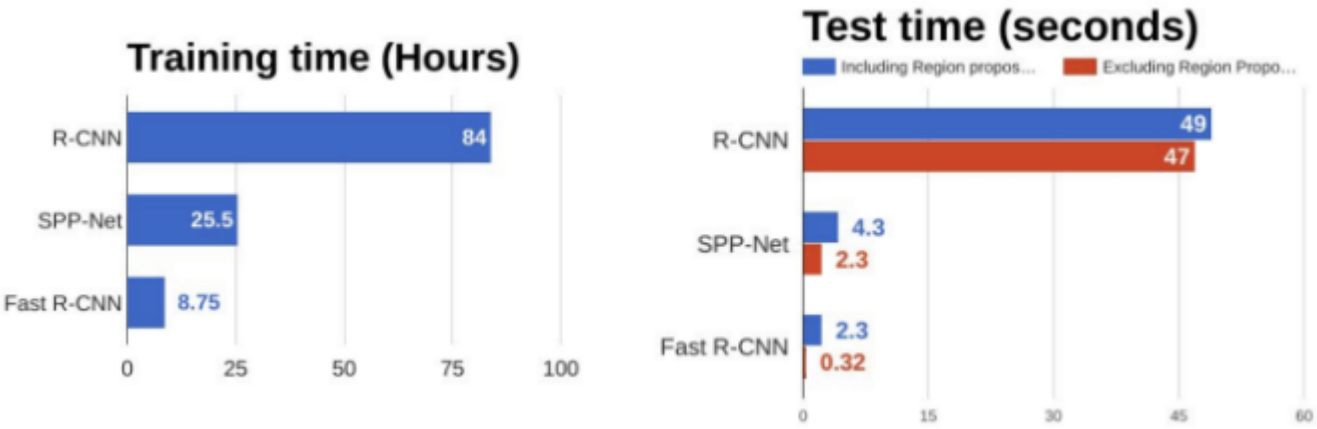

Figura 3.5: Comparación de algoritmos. (Origen: Toward Data Science)

A partir de la imagen anterior podemos ver cómo Fast R-CNN es significativamente más rápido que R-CNN, esto es debido a que la operación de convolución solo se realiza una vez por imagen. Como podemos observar la generación de propuestas de región ralentiza considerablemente el algoritmo.

Faster R-CNN [33]: los algoritmos anteriores utilizan la búsqueda selectiva para la generación de posibles regiones. Este proceso es lento y produce un cuello de botella que afecta al rendimiento de la red. Shaoquin Ren et al. introdujeron una red separada para predecir las regiones, lo que hace que el coste del cómputo sea casi nulo. En la figura 3.6 podemos observar que es mucho más rápido que sus predecesores. Por lo tanto se podría usar para la detección de objetos en tiempo real.

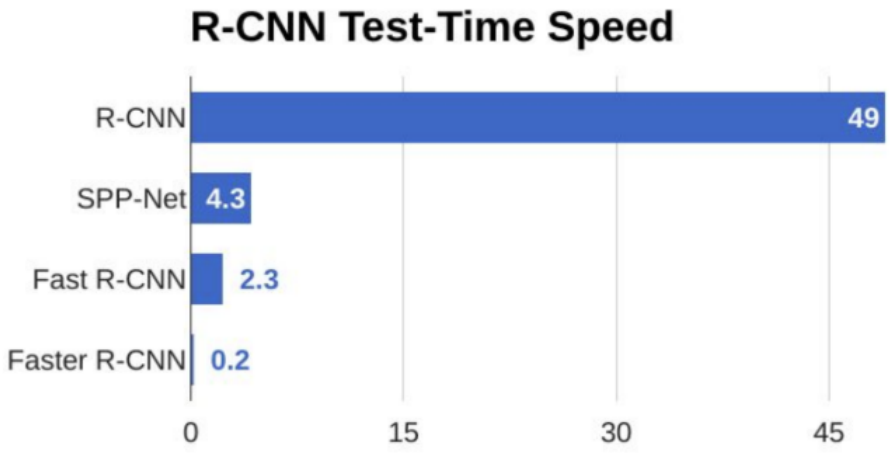

Figura 3.6: Comparación de velocidad de los algoritmos de detección de objetos. (Origen: Toward Data Science)

R-FCN [34]: Li et al. propusieron un método basado completamente en redes convolucionales (R-FCN). Con R-FCN la clasificación de redes es más poderosa, logrando la identificación de objetos con una arquitectura totalmente convolucional compartiendo todas las capas vecinas. Según los resultados obtenidos en pruebas de velocidad con datasets de PASCAL VOC y Microsoft COCO el algoritmo tarda unos 170ms por cada imagen.

Mask R-CNN [35]: la segmentación de instancias es una tarea desafiante ya que requiere la detección de objetos en una imagen y la segmentación de cada instancia (dividir una imágen en varias partes u objetos). Estas dos tareas son normalmente realizadas como dos procesos independientes. Mask R-CNN utiliza la arquitectura existente en Faster R-CNN para la clasificación y la regresión de las cajas delimitantes y añade otra rama para predecir la máscara de segmentación. Esta herramienta puede ser también una posible elección para la realización de nuestro problema, puesto que con una simple implementación promete buenos resultados de segmentación y detección de objetos.

En la figura 3.7 vemos lo que realiza cada tarea de visión. La clasificación nos dice que en esa imagen aparece un globo, la segmentación semántica nos expone todos los píxeles que pertenecen a un globo. La detección de objetos nos muestra cuántos glo-

bos aparecen y en qué lugares se encuentran y finalmente la segmentación de instancias encuentra cada globo y nos dice los píxeles que pertenecen a cada uno.

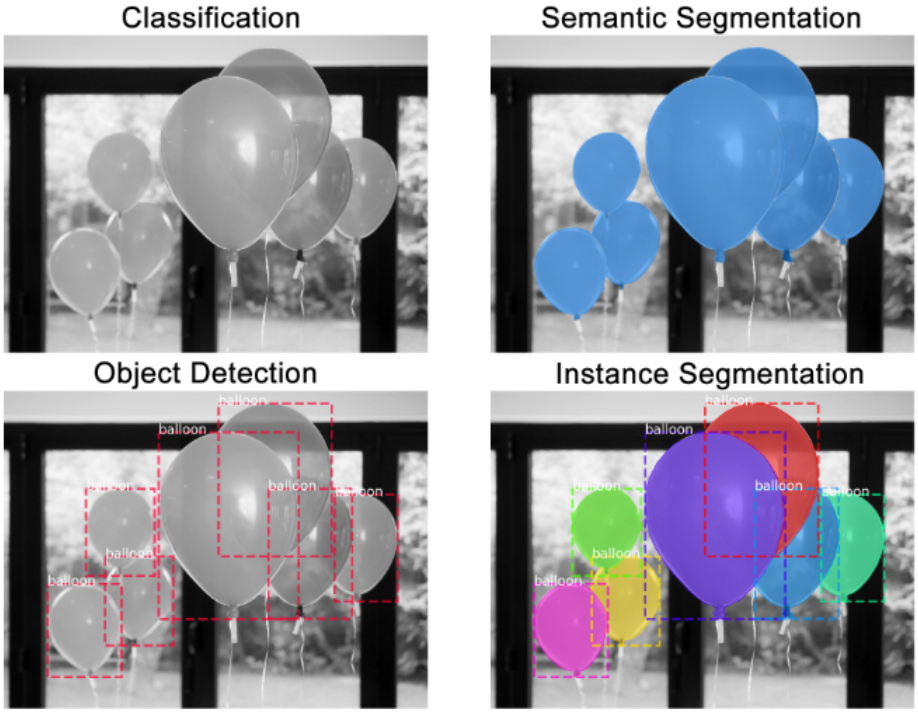

Figura 3.7: Tareas de visión. (Origen: Medium.com)

3.2.2.2. Algoritmos para detección de objetos utilizando regresión/clasificación

Como ya hemos visto los frameworks basados en regiones están compuestos de varias etapas (generación de las regiones, extracción de características, clasificación y regresión del recubrimiento) las cuáles se suelen entrenar de forma separada. Al necesitar un entrenamiento alternativo para obtener los parámetros compartidos, representan un cuello de botella en aplicaciones en tiempo real.

Los frameworks basados en la regresión y clasificación, mapean directamente de los píxeles de la imagen a las coordenadas de las cajas delimitantes y a las probabilidades de cada clase, lo que hace que se reduzca el tiempo. Vamos a estudiar dos algoritmos importantes que tienen estas características:

YOLO [40]: Redmon et al. propusieron este framework que usa todo el mapa de características para predecir múltiples categorías y cajas delimitantes[1]. YOLO divide la imagen de entrada en una cuadrícula S x S y cada celda es responsable de predecir el objeto que se encuentra en ella. Los cuadros delimitadores que tienen la probabilidad de clase por encima de un umbral se seleccionan y se utilizan para ubicar el objeto dentro de la imagen. La red puede procesar imágenes en tiempo real a 45 FPS y una versión simplificada (Fast YOLO) puede llegar a alcanzar los 155 FPS, por lo tanto proporciona mejores resultados que otros detectores en tiempo real. Pero con los objetos pequeños podría tener dificultades para detectarlos debido a las restricciones espaciales del algoritmo.

SSD [41]: dada la limitación de Yolo con los objetos pequeños, con SSD dado un mapa de características específico, Liu et al. propusieron en vez de utilizar las celdas fijas que usa YOLO, coger un conjunto de cajas por defecto con diferentes tamaños para discretizar la salida del espacio de las cajas delimitantes. Con las pruebas de velocidad realizadas, SSD es más rápido que Faster R-CNN y puede llegar a tener más precisión que YOLO aunque seguiría teniendo dificultades con los objetos pequeños.

3.3. Justificación de elección de la herramienta

Después de estudiar los posibles algoritmos para detección de objetos vemos que los que más se ajustan a nuestro problema son: Fast R-CNN, Faster R-CNN, SSD y YOLO. Para elegir el más apropiado llevamos a cabo una comparación según su precisión y según su velocidad:

[1] describe un rectángulo donde se encuentra un objeto

Figura 3.8: Comparación algoritmos precisión/velocidad. (Origen: cv-tricks.com)

La gráfica nos muestra cómo el algoritmo Faster R-CNN es el más apropiado si queremos tener la mayor precisión posible y no nos importa el tiempo de ejecución. SSD mantiene una buena precisión y además es más rápido que Faster R-CNN. En nuestro trabajo le vamos a dar más importancia a la velocidad antes que a la precisión, ya que se pretende procesar vídeo, no imágenes aisladas, a una velocidad lo más cercana posible al tiempo real. Por este motivo YOLO será el algoritmo usado para la resolución de nuestro problema.

Capítulo 4

Creación de la base de datos

En este apartado se exponen las consideraciones que hemos seguido y los problemas que han surgido para diseñar la base de datos de imágenes y su consiguiente etiquetado.

4.1. Consideraciones previas

La base de datos tiene un papel muy importante a la hora del entrenamiento de las redes neuronales. Sin una buena base de datos, nuestra red no tendrá los resultados esperados. Para la realización de nuestra base de datos de matrículas debemos de tener en cuenta que esté compuesta por:

- Imágenes con diferentes ángulos y distancias.
- Imágenes de matrículas de diferentes países.
- Imágenes de matrículas delanteras y traseras.
- Imágenes de matrículas en vehículos de distinto tipo (camiones, coches deportivos, coches antiguos, etc.).

Si conseguimos que la base de datos tenga imágenes muy variadas conseguiremos evitar el conocido como *overfitting* (sobreajuste) o *underfitting* (subajuste). Si nuestras imágenes de matrículas son solo de un país o de un tipo de vehículo en concreto, la red

será específica para las matrículas de ese país y de ese vehículo. En nuestro caso queremos que la red sea capaz de detectar cualquier matrícula.

4.2. Proceso seguido

La primera búsqueda para empezar a recopilar las imágenes se realizó en la Dirección General de Tráfico (DGT). Nuestro objetivo era encontrar alguna base de datos que incluyera imágenes o vídeos de coches donde se pudiera apreciar la matrícula. Tras ponernos en contacto con la DGT nos informaron de que no disponen de ninguna base de datos pública debido a la Ley de Protección de Datos de Carácter Personal. Las matrículas son consideradas como datos personales y por este motivo no se pueden difundir.

Seguidamente se recopilan imágenes de la web, descartando aquellas donde la matrícula no se llega a apreciar. Finalmente la mayoría de las imágenes se obtienen de dos procedencias: la primera del dataset de coches de la Universidad de Stanford [45]. Esta base de datos se compone de más de 16 mil imágenes, de las cuáles hemos descartado la mayoría porque las matrículas no eran visibles. Su nula visibilidad se debía a que estas se encontraban difuminadas para ocultarlas o porque en las imágenes aparece el vehículo, pero la matrícula está en ángulos de visión nulos.

Figura 4.1: Ejemplo de imagen desechada (Fuente: dataset de Stanford)

El segundo dataset fue extraído del proyecto Openalpr [1] del que obtuvimos unas 100 imágenes. Finalmente, nuestro dataset cuenta con aproximadamente 2500 imágenes de matrículas de distintos países con la desventaja de que la cantidad de matrículas españolas es baja.

En caso de que los resultados del entrenamiento no sean los deseados, puede ser debido a no tener suficientes imágenes. Para solucionarlo podemos usar la técnica conocida como aumento de datos o *data augmentation*. Esta técnica nos permite aumentar nuestro dataset de dos formas. La primera es introduciendo perturbaciones en las imágenes reales, como por ejemplo realizando traslación en ellas, invirtiendo los ejes o realizando un escalado. La otra forma es añadiendo filtros o un poco de ruido, pero siempre manteniendo una mayor proporción de imágenes con alta resolución. En nuestro caso, no hemos realizado está técnica, pero podría ser una futura mejora para aumentar el dataset de matrículas españolas.

Figura 4.2: Ejemplo de imagen de nuestro dataset. (Fuente: base de datos propia)

[1] `https://github.com/openalpr/benchmarks/tree/master/endtoend`

Aunque en la figura 4.2 las imágenes aparecen en color, se decide transformarlas a blanco y negro. Esta decisión es tomada porque casi en toda la base de datos, el fondo de las placas es blanco y por lo tanto la red solo entrenaría un pequeño número de matrículas de color. Esto podría tener como consecuencia que no identificará las matrículas de los vehículos públicos, vehículos de importación o los demás que poseen su placa de color. En cambio, si en líneas futuras se quieren distinguir estos vehículos por el color de su matrícula, las imágenes deberían de ser a color.

Después de la recolección, se etiquetan las matrículas que aparecen en cada imagen. El proceso de etiquetado consiste en indicar la posición de las matrículas y su categoría. Se han probado dos formas de etiquetar la base de datos:

1. Para que el proceso fuera más rápido, se realizó un script que identificaba la posición de cada matrícula o matrículas visibles en cada imagen con un haar-like cascade [10] optimizado para su detección. Al comprobar los resultados, desechamos esta opción porque solo detectaba un tipo de matrícula que tuviera un ángulo e iluminación determinado.

2. Finalmente el etiquetado se realizó manualmente con el programa LabelImg[2]. Su funcionamiento es sencillo, se recuadran la/las matrículas que aparecen en cada imagen y con el formato YOLO seleccionado.

[2] `https://github.com/tzutalin/labelImg`

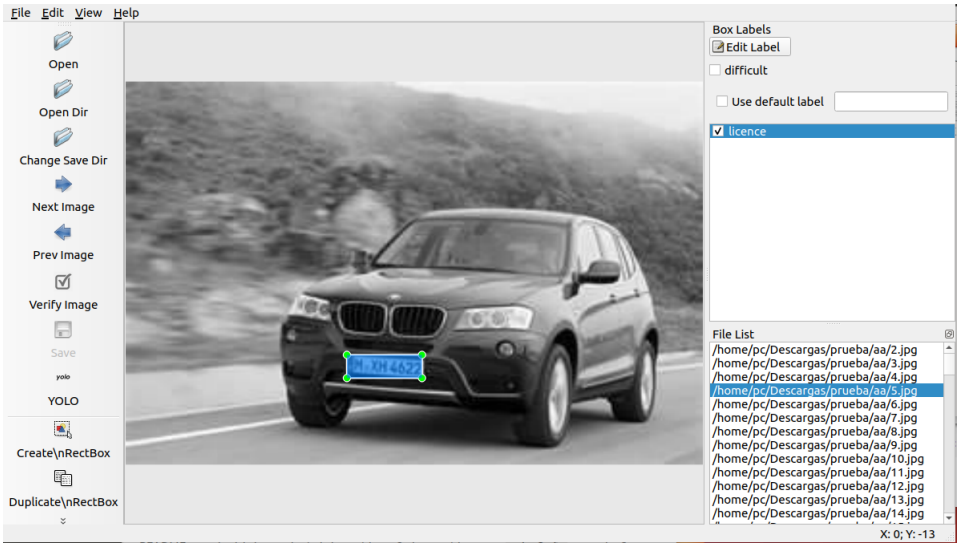

Figura 4.3: Etiquetado con LabelImg (Fuente: elaboración propia)

Automáticamente se guarda un archivo de texto con el mismo nombre que la imagen y con el formato que utiliza YOLO (número de categoría, posición en X, posición en Y, anchura y altura).

```
0 0.104688 0.650000 0.134375 0.091667
```

Figura 4.4: Ejemplo de notación YOLO (Fuente: elaboración propia)

En nuestro caso solo teníamos la categoría matrícula, pero para posibles mejoras del trabajo podría ser interesante tener una clase para cada país y así identificar la matrícula y su procedencia. En caso de utilizar otro programa para el etiquetado, posteriormente se debe hacer una conversión de la notación a formato YOLO.

Capítulo 5

Diseño e implementación del sistema de aprendizaje

En este apartado vamos a explicar cómo funciona YOLO (la primera versión de este algoritmo) y se mostrarán las mejoras realizadas en las siguientes versiones hasta llegar a su última versión YOLOv3. Además, se expondrá cómo se realiza el entrenamiento de la red y los parámetros que hay que configurar en YOLOv3 para conseguir los mejores resultados en nuestro problema.

5.1. Arquitectura de la red YOLO

YOLO está formada por 24 capas convolucionales, seguidas por dos capas completamente conectadas. Como se aprecia en la figura 5.1 algunas capas convolucionales son de 1x1 para así reducir la profundidad.

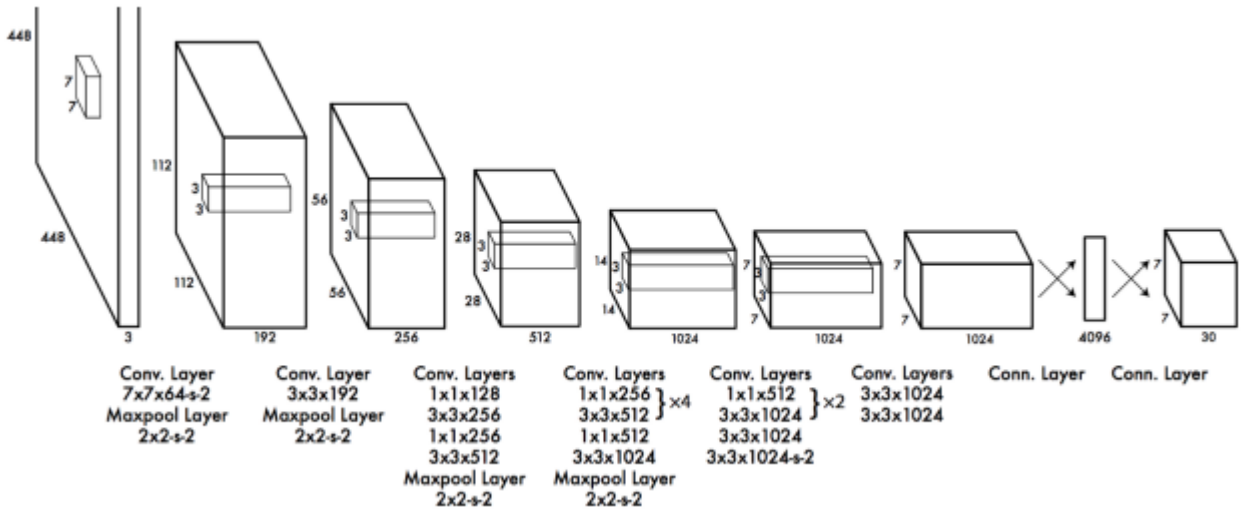

Figura 5.1: Arquitectura YOLO. (Fuente: [46])

La figura 5.1 está pensada para el dataset Pascal VOC, que contiene 20 clases. A continuación vamos exponer el proceso llevado a cabo para obtener en la salida una dimensión de (7,7,30):

- Primero, la imagen se divide en celdas de siete píxeles de ancho y de largo. De cada celda obtenemos dos cajas delimitantes por lo tanto (2x5+20), donde cinco son los parámetros que se necesitan para identificar una caja delimitante (pc,bx,by,bh,bw). Esto significa que cada celda nos da 30 valores y la probabilidad de que el objeto pertenezca a cada una de las clases.

- Después, de esas dos cajas delimitantes se coge la que mayor probabilidad tenga con respecto a la clase. Esta arquitectura accede a toda la imagen para realizar predicciones, cosa que no se cumple con los algoritmos basados en regiones que están limitados a la región seleccionada. Pero esta arquitectura también conlleva una desventaja, si dos objetos se encuentran muy próximos, YOLO será incapaz de detectarlos. Este problema no ocurre con la detección de las matrículas puesto que están suficientemente separadas una de otra.

Para conocer la precisión de la red, YOLO utiliza la función de pérdida SSE (*Sum-Squared Error*) [47]. Mide la media de los errores al cuadrado, es decir, mide la diferencia entre el estimador y lo que se estima. Esta diferencia se produce porque se aspira a un valor esperado y debido a falta de información o de aleatoriedad no se origina ese valor. Esta función es usada en YOLO para contabilizar la pérdida de localización, de confianza y de clasificación.

5.2. Versiones de YOLO

Hemos destacado que YOLO es más rápido que la mayoría de los algoritmos de detección, debido a que hace predicciones con una única evaluación de red, pero también cuenta con ciertas limitaciones. Estas se han resumido en tres:

- YOLO tiene ciertos errores buscando objetos cercanos, sobre todo si son pequeños.
- YOLO puede detectar cómo máximo en una imagen 49 objetos.
- El error de localización es relativamente alto.

En la segunda versión se centraron en mejorar esta última limitación y que la red mantuvierá la precisión en la clasificación. Para ello se normalizaron los lotes [48], esto consiste en reducir los reajustes en la capa actual producidos por los cambios en las capas anteriores. Otra mejora fue una mayor resolución en el clasificador, ya que en la primera versión se entrenaba con clasificadores con 224x224 de resolución y en YOLOv2 se aumenta a 448x448, lo que mejora un 4 % la precisión. Finalmente, también mejoraron los cuadros delimitantes, tratando de encontrar las formas que más se ajusten a los objetos para que así la red aprenda más fácilmente.

Otra versión de YOLO fue llamada YOLO9000 [49]. Este modelo aumenta el número de clases que puede detectar YOLO, de anteriormente 20 a más de 9000 categorías, de ahí el nombre de la versión. Para conseguir esto se optimizó de forma conjunta el entrenamiento de la clasificación y de la detección. Por lo tanto se utiliza el mismo conjunto de

datos para la detección y clasificación, donde las diferentes categorías están organizadas de forma jerárquica (Animal >Gato >Gato Persa). El conjunto creado para YOLO9000 se creó combinando el conjunto de datos de detección de COCO [50] y las 9000 clases principales de ImageNet [51].

La última versión es YOLOv3 [51] y es la utilizada en nuestro problema. En algunas ocasiones un objeto puede tener etiquetas superpuestas, como por ejemplo, mujer y persona. Para poder etiquetar el objeto con las dos clases YOLOv3 utiliza clasificadores independientes para cualquier clase. Los objetos pequeños son un problema en YOLO, pero con esta nueva versión mejora su rendimiento debido al uso de conexiones de atajo que nos permite obtener información más detallada del mapa de características. Aunque esta versión sea más lenta que YOLO9000, es más robusta, debido a que realiza predicciones en tres escalas diferentes.

A continuación se muestra una comparación de las versiones de YOLO según su velocidad (FPS), y precisión media promedio (*mean Average Precision*, mAP) [52]. Para calcular la precisión media promedio tenemos que utilizar la intersección sobre unión (*Intersection over Union*, IoU) para definir un verdadero positivo. Esta consiste en la intersección entre una caja delimitante predicha del objeto y su verdadera caja delimitante. Tras conocer los verdaderos positivos, los falsos positivos y los falsos negativos podemos calcular la mAP, calculada con la curva de precisión (la relación entre TP/(TP+FP)) y recuperación (TP/(TP+FN)).

Modelo	**FPS**	**mAP**	**Dataset**
YOLOv1	45	63.4	VOC
YOLOv2	40	48.1	COCO
YOLOv3	20	57.9	COCO

Tabla 5.1: Rendimiento de las versiones de YOLO (Fuente del libro: Information Science and Applications: ICISA 2019)

La tabla anterior nos muestra cómo YOLOv1 obtiene la precisión más alta con el conjunto de datos VOC. Esta versión no puede ser comparada con las otras dos por utilizar un dataset distinto. Comparando YOLOv2 y YOLOv3 encontramos como hemos dicho anteriormente que la versión 3 pierde velocidad a cambio de robustez.

5.3. Fase de pre-entrenamiento

Antes de llevar a cabo el entrenamiento debemos de configurar los parámetros y archivos necesarios según las características del problema. En este caso vamos a usar un modelo pre-entrenado llamado darknet53 [53] utilizado para las capas convolucionales. Este modelo nos va a reducir el tiempo de entrenamiento. También se puede entrenar desde cero pero le costará más tiempo encontrar la configuración óptima de pesos.

Model	Top-1	Top-5	Ops	GPU	CPU	Cfg	Weights
AlexNet	57.0	80.3	2.27 Bn	3.1 ms	0.29 s	cfg	238 MB
Darknet Reference	61.1	83.0	**0.96 Bn**	**2.9 ms**	**0.14 s**	cfg	28 MB
VGG-16	70.5	90.0	30.94 Bn	9.4 ms	4.36 s	cfg	528 MB
Extraction	72.5	90.8	8.52 Bn	4.8 ms	0.97 s	cfg	90 MB
Darknet19	72.9	91.2	7.29 Bn	6.2 ms	0.87 s	cfg	80 MB
Darknet19 448x448	76.4	93.5	22.33 Bn	11.0 ms	2.96 s	cfg	80 MB
Resnet 18	70.7	89.9	4.69 Bn	4.6 ms	0.57 s	cfg	44 MB
Resnet 34	72.4	91.1	9.52 Bn	7.1 ms	1.11 s	cfg	83 MB
Resnet 50	75.8	92.9	9.74 Bn	11.4 ms	1.13 s	cfg	87 MB
Resnet 101	77.1	93.7	19.70 Bn	20.0 ms	2.23 s	cfg	160 MB
Resnet 152	77.6	93.8	29.39 Bn	28.6 ms	3.31 s	cfg	220 MB
ResNeXt 50	77.8	94.2	10.11 Bn	24.2 ms	1.20 s	cfg	220 MB
ResNeXt 101 (32x4d)	77.7	94.1	18.92 Bn	58.7 ms	2.24 s	cfg	159 MB
ResNeXt 152 (32x4d)	77.6	94.1	28.20 Bn	73.8 ms	3.31 s	cfg	217 MB
Densenet 201	77.0	93.7	10.85 Bn	32.6 ms	1.38 s	cfg	66 MB
Darknet53	77.2	93.8	18.57 Bn	13.7 ms	2.11 s	cfg	159 MB
Darknet53 448x448	**78.5**	**94.7**	56.87 Bn	26.3 ms	7.21 s	cfg	159 MB

Figura 5.2: Modelos previamente entrenados para YOLOv3 (Fuente: pjreddie.com)

5.3.1. Parámetros

Para el entrenamiento en YOLOv3 [54] es necesario ajustar parámetros que se encuentran en el archivo con extensión '.cfg'. Los parámetros que se van modificar son:

- **batch**: este parámetro es el conjunto de imágenes que modifican en cada iteración los pesos de la red. Cuanto mayor sea, más rápido se entrena la red, pero esto conlleva un aumento del uso de la memoria principal o de la memoria de la GPU (dependiendo de la configuración). El valor ideal de este parámetro es 64 pero depende de la memoria de la máquina donde se realice el entrenamiento. El valor que se ha usado para nuestro entrenamiento es 32.

- **subdivisions**: corresponde a las divisiones de `batch` y se realiza para acelerar el entrenamiento y mejorar la generalización. El parámetro ideal es 64 pero en nuestro caso lo hemos dejado en 16, ya que debe ser menor o igual al `batch`.

- **classes**: es el número de clases que se van a identificar o clasificar. Para nuestro entrenamiento el valor será 0, esto quiere decir que tenemos una clase, las matrículas.

- **filters**: los filtros son los kernel que aplicaremos. Por lo tanto si tenemos 32 filtros quiere decir que tendremos 32 matrices de salida, y cada una de estas salidas tendrá su número de neuronas dependiendo del tamaño de las imágenes. Para una imagen cuadrada de apenas 28 píxeles tendríamos 25088 neuronas (28*28*1*32). Los filtros dependen del número de clases que tengamos, para calcularlos seguiremos esta relación: filtros = (clases+5)*3 (donde 5 son los parámetros que se necesitan para identificar una caja delimitante (`pc,bx,by,bh,bw`) y 3 es por el número de cajas delimitantes que se encuentran por celda). Por lo tanto en nuestro caso se utilizan 18 filtros (1+5)*3.

5.3.2. Ficheros

Se crean los ficheros `test.txt` y `train.txt` donde se incluyen las rutas de las imágenes. Estos archivos serán utilizados para que la red encuentre las imágenes durante el test o

el entrenamiento. Para que este proceso sea automático se ha realizado un script donde únicamente le facilitamos la dirección de la carpeta contenedora de las imágenes y crea automáticamente el archivo de rutas.

En el archivo `obj.names` se introducen los nombres de las clases. Nuestro fichero contiene solamente la clase 'matricula'. Si quisiéramos distinguir las matrículas de cada país en un proyecto futuro, se tendría que incluir en este archivo los países como clases.

El último fichero que se debe crear es un archivo con extensión `.data`. En él se incluye el número de clases (1), y las rutas de los ficheros anteriormente creados (`test.txt`, `train.txt` y `obj.names`). Este fichero será llamado a la hora de realizar el entrenamiento o validación de la red.

5.4. Entorno de ejecución

Como ya sabemos, para realizar cálculos los ordenadores utilizan dos unidades de procesamiento: la unidad de procesamiento central (CPU) y por otro lado, las unidades de procesamiento gráfico (GPU). La principal diferencia reside en su arquitectura, el número de núcleos en las GPU es mayor que en las CPU. Por este motivo, la GPU realiza las operaciones en paralelo mucho más rápido que la CPU, en cambio, en las tareas secuenciales la GPU no es capaz de llevarlas a cabo y se encarga la CPU debido a que sus núcleos son más potentes. Como la mayoría de las operaciones que se realizan en el entrenamiento de una red neuronal son matriciales, es muy recomendable llevarlas a cabo con una buena GPU.

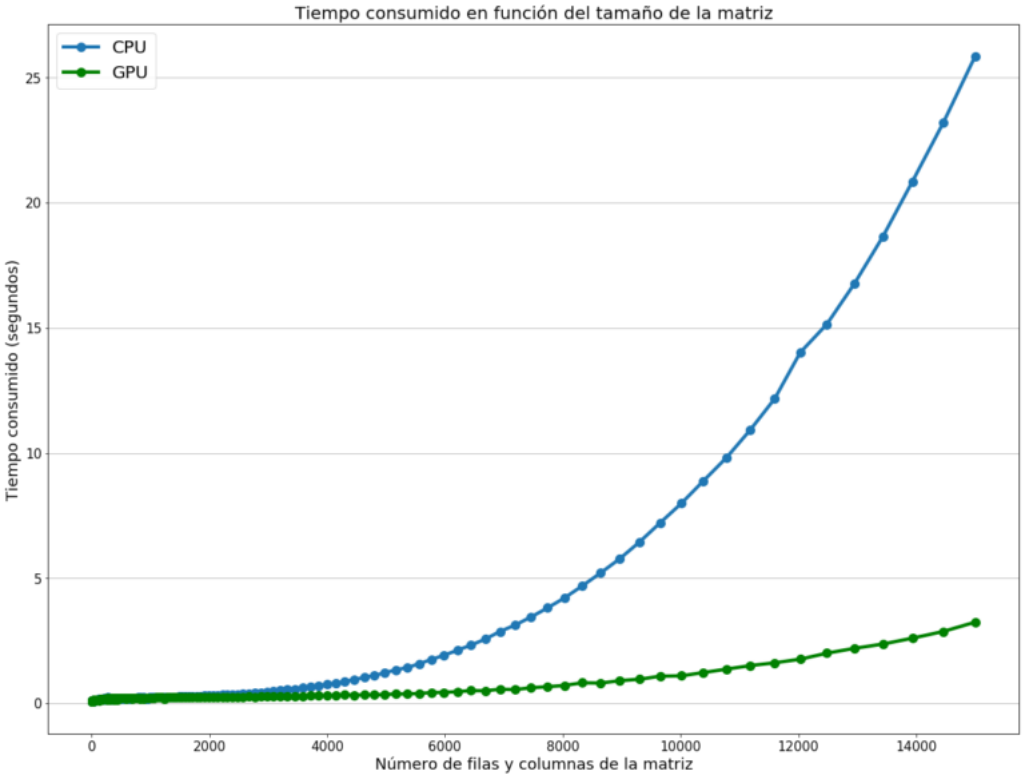

Figura 5.3: Tiempo consumido conforme el tamaño de las matrices aumenta (Fuente: machinelearningparatodos.com)

El entrenamiento se ha realizado con Google Colab desarrollado en el anexo C ya que tiene pre-instaladas algunas librerías como CUDA[1] y podemos enlazar la base de datos si la tenemos guardada en Google Drive. Solo es necesario instalar cuDNN[2] y cambiar el acelerador por hardware a GPU en vez de CPU.

5.5. Ensayos realizados y resultados obtenidos

En este apartado vamos a exponer las pruebas realizadas y veremos cuál ha dado mejores resultados.

[1] `https://developer.nvidia.com/cuda-downloads?`
[2] `https://developer.nvidia.com/rdp/cudnn-download`

1. Ensayo elaborado con 500 imágenes.
2. Ensayo elaborado con 2530 imágenes.
    - División de datos 50-50 (50 % entrenamiento y 50 % test).
    - División de datos 75-25 (75 % entrenamiento y 25 % test).
    - División de datos 90-10 (90 % entrenamiento y 10 % test).

5.5.1. Ensayo elaborado con 500 imágenes

Para realizar una buena validación de una red se debe descomponer la base de datos en 3 partes: entrenamiento, validación y test. El conjunto de entrenamiento, son los datos con los que se va a construir el modelo, el conjunto de validación se usa para validar el modelo y prevenir que este se sobre-ajuste o se sub-ajuste y el conjunto de test es el subconjunto que se mantiene al margen del entrenamiento y que se utiliza para evaluar el modelo. En YOLO, el proceso de validación se realiza junto con las imágenes test.

Al principio, se hicieron algunas pruebas con una base de datos de 500 matrículas, cuyas imágenes eran en color y de un solo país (Croacia). Además, el etiquetado del primer entrenamiento se hizo con haar-like cascade. La validación se hizo también con matrículas de Croacia y la precisión media promedio (mAP) fue del 96 %, pero a la hora de probarlo con matrículas de otro país o incluso que fueran de Croacia pero que tuvieran algún ángulo, no detectaba ninguna matrícula.

5.5.2. Ensayo elaborado con 2530 imágenes

Los resultados con el entrenamiento de esas 500 imágenes y 4000 iteraciones no eran los deseados debido a que la red se había sobre-entrenado. Por lo tanto se decide buscar más imágenes de matrículas y pasarlas a blanco y negro (el color no es un dato relevante).

Finalmente entrenamos la red con 2530 imágenes y comparamos los resultados con el primer entrenamiento.

```
 calculation mAP (mean average precision)...
20
 detections_count = 1, unique_truth_count = 18
class_id = 0, name = licence, ap = 0.00%         (TP = 0, FP = 1)


 calculation mAP (mean average precision)...
20
 detections_count = 54, unique_truth_count = 18
class_id = 0, name = licence, ap = 86.97%        (TP = 16, FP = 2)
```

Figura 5.4: Validación de los entrenamientos. (Fuente: elaboración propia)

Como podemos ver, la primera imagen es el resultado de realizar el test al primer entrenamiento. Este test solo contaba con 18 imágenes, de esas 18, solo detecta una matrícula que resulta ser un falso positivo. La segunda imagen corresponde con el test realizado de la red entrenada con 2530 matrículas. En este caso, obtenemos como resultado 16 matrículas encontradas correctamente y 2 erróneas, con un 87 % mAP aproximadamente.

Para saber qué porcentaje debemos dejar para el conjunto de entrenamiento y el conjunto de test hemos realizado 3 pruebas: 50 % entrenamiento y 50 % test, 75 % entrenamiento y 25 % test y 90 % entrenamiento y 10 % test.

5.5.2.1. División de datos 50-50

En esta primera prueba dividimos en dos la base de datos. Contamos con 1265 imágenes usadas para el entrenamiento y 1265 imágenes para el test.

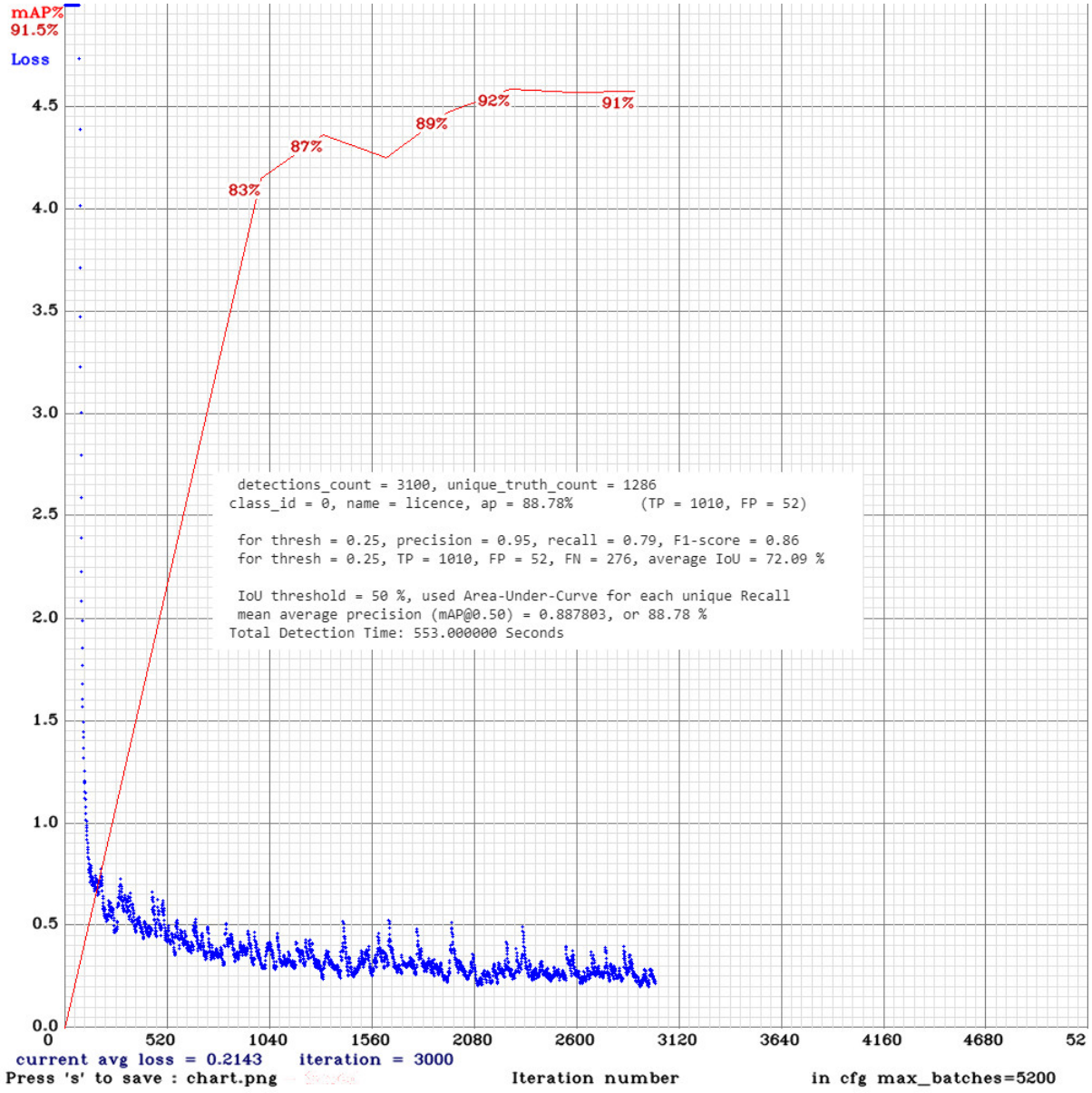

Figura 5.5: Estudio de la división de datos 50-50 (Fuente: elaboración propia))

Contemplando la figura 5.5 se observa cómo la precisión es alta. Al evaluar el entrenamiento obtenemos un 88.78 % de acierto, es menor que el 91 % mAP logrado en el entrenamiento, por lo tanto llegamos a la conclusión de que la red puede estar infra-ajustada.

5.5.2.2. División de datos 90-10

En la siguiente prueba dejamos la mayoría de las imágenes para el entrenamiento y tan solo un 10 % para el test. Por lo tanto contamos con 2278 imágenes para el entrenamiento y 253 imágenes para el test.

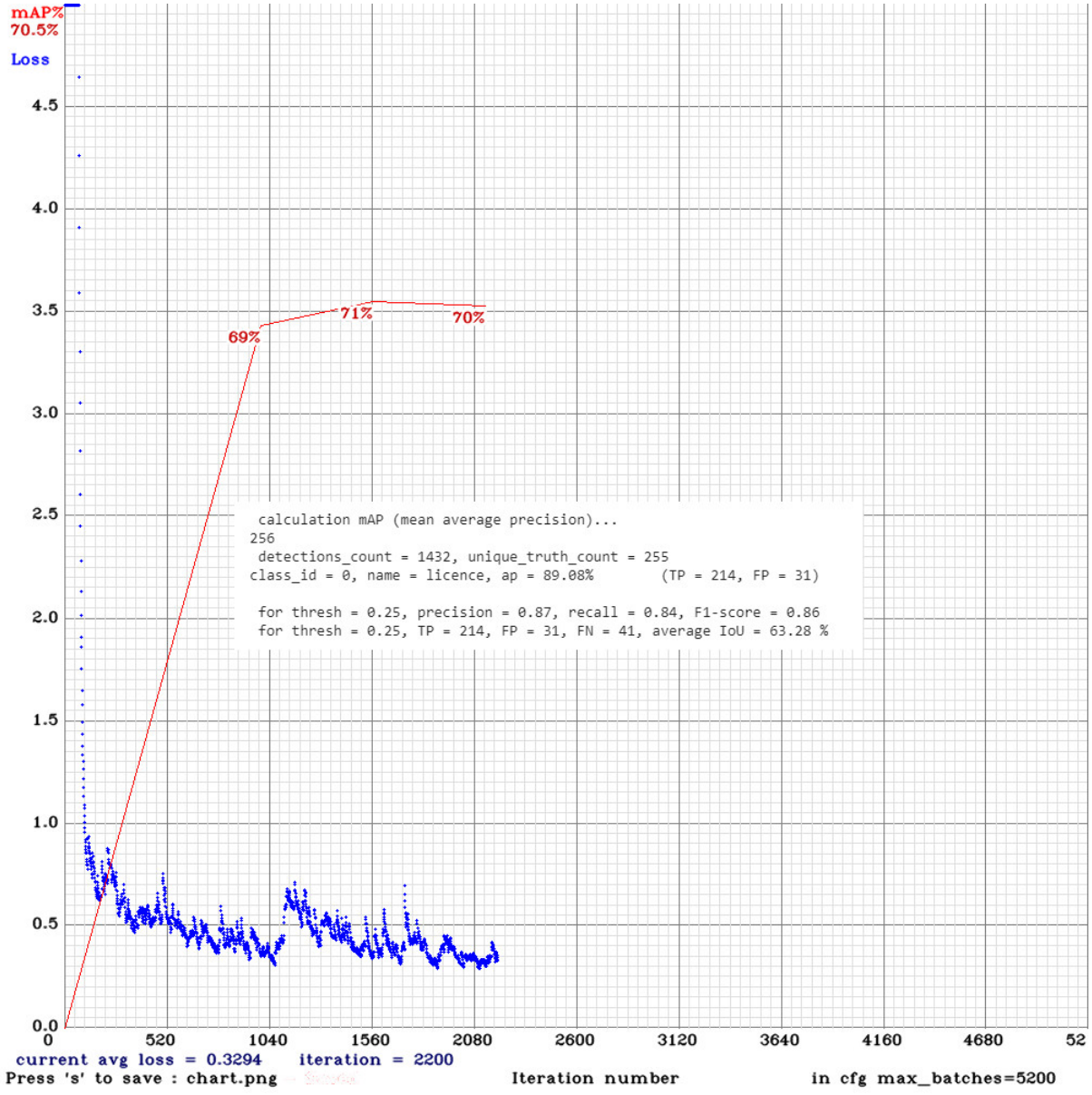

Figura 5.6: Estudio de la división de datos 90-10 (Fuente: elaboración propia))

Los resultados en este estudio nos muestran que la precisión en el entrenamiento es

mayor en la prueba anterior, pero en la evaluación se obtiene mayor precisión que con la división anterior.

5.5.2.3. División de datos 75-25

Viendo los resultados obtenidos en las anteriores, se decide probar con una división que sea el término medio de los anteriores. Contamos con 1899 imágenes para el entrenamiento y 632 imágenes para el test.

Con esta división obtenemos la mayor precisión tanto en el entrenamiento como en la evaluación. Esto quiere decir que la red esta bien ajustada, de modo que esta división será usada para la red definitiva.

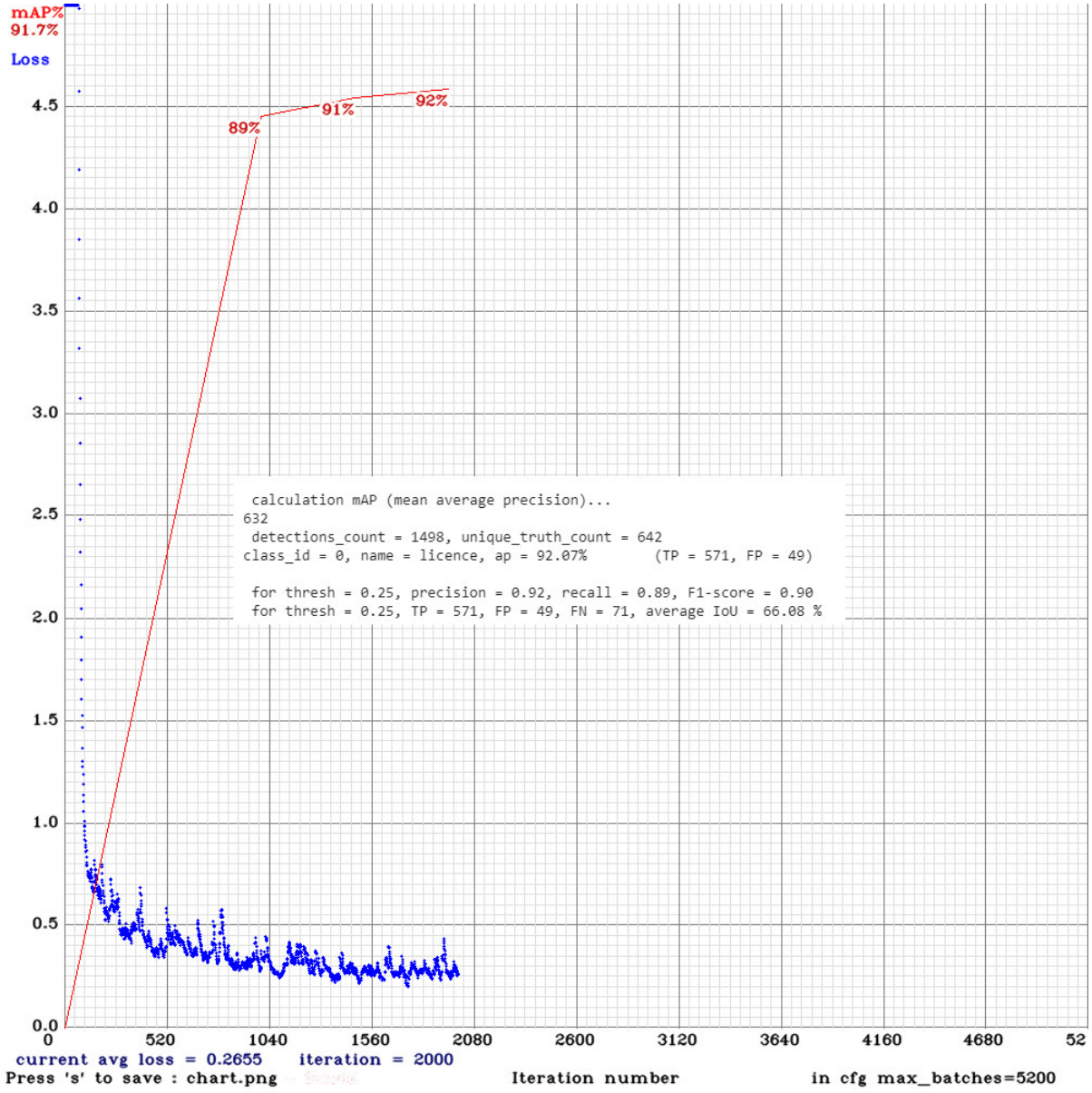

Figura 5.7: Estudio de la división de datos 75-25 (Fuente: elaboración propia))

Capítulo 6

Lectura de matrículas

Tras realizar el entrenamiento y obtener las matrículas en imágenes y vídeos, procedemos a su lectura. Nos proporciona mucha información que puede ser utilizada en control de frontera, parkings, radares, etc. Las principales técnicas usadas son: utilizando OCR (*Optical Character Recognition*) [55] o realizando aprendizaje profundo.

6.1. Lectura utilizando OCR

Utilizando OCR es posible identificar automáticamente dígitos o caracteres y almacenarlos de forma digital. Este proceso es bastante usado actualmente para la digitalización de libros o documentos, con la ventaja de que posteriormente pueden ser modificados en un editor de texto. Para obtener buenos resultados en nuestro problema es importante que las condiciones de iluminación y perspectiva de la matrícula sean favorables.

6.1.1. Algoritmo OCR

Para obtener una mayor precisión en la lectura es necesario realizar un pre-procesamiento de la imagen. Las etapas seguidas han sido las siguientes:

- **Convertir a blanco y negro**: como el color no es necesario para su lectura transformamos la imagen a blanco y negro. Esto es aconsejable porque procesar la imagen en un solo canal es más sencillo que en tres.

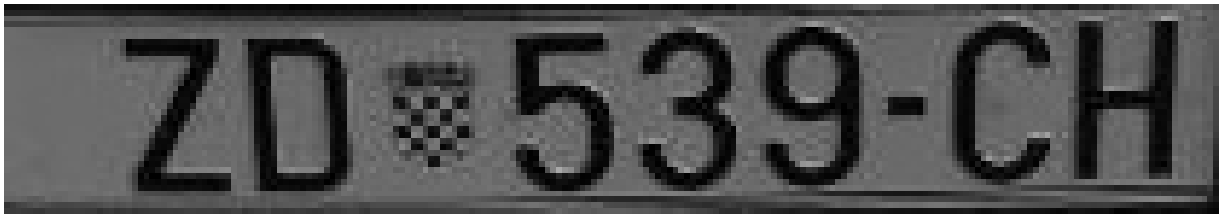

Figura 6.1: Matrícula en Blanco y Negro (Fuente: elaboración propia)

- **Umbralización**: consiste en obtener píxeles que superen un umbral determinado y pasarlos a negros y los que queden por debajo de ese valor pasarlos a blancos. Con este proceso conseguimos eliminar ruido y podemos aumentar la nitidez de los caracteres y dígitos. Para elegir el umbral se ha utilizado el método Otsu porque nos ofrecía los mejores resultados. El método Otsu [56] emplea técnicas estadísticas como la varianza para resolver este problema. Calcula el umbral buscando el coeficiente máximo entre las varianzas de cada segmento.

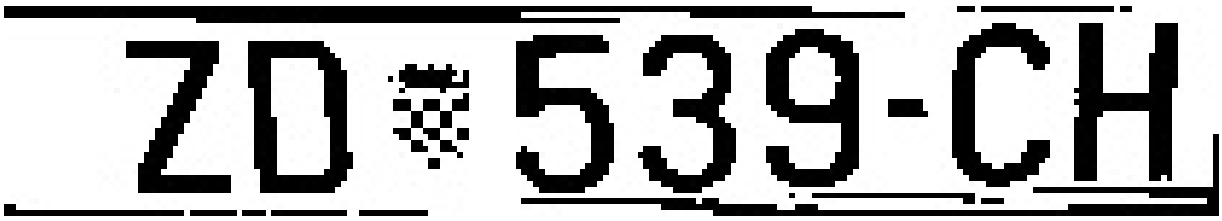

Figura 6.2: Matrícula umbralizada (Fuente: elaboración propia)

- **Aplicar un filtro**: el objetivo de usar un filtro es suavizar la imagen y resaltar la información de las matrículas. Para saber qué filtro se ajusta más al problema, realizamos pruebas entre : **filtro medio** (hace una media entre los valores del pixel que se encuentran por debajo del filtro), **filtro de la mediana** (se obtiene tomando el número central entre los píxeles ordenados por tamaño) y **filtro gaussiano** (usa el valor central de la función de Gauss que representa el valor con mayor peso)

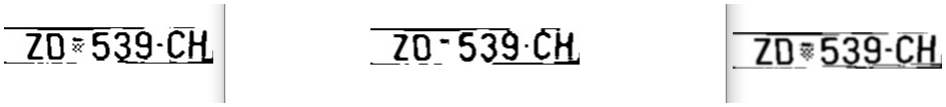

Figura 6.3: Filtros aplicados de izquierda a derecha (filtro medio, filtro de la mediana y filtro gaussiano). (Fuente: elaboración propia)

El filtro medio nos proporciona buenos resultados para esta matrícula, puesto que nos elimina símbolos innecesarios como el escudo de Croacia. Aunque al utilizar el filtro para las matrículas que se encuentran más alejadas hace que la imagen se quede emborronada y nos perjudica a la hora de la lectura.

Figura 6.4: Lectura realizada sin filtro y con filtro mediana. (Fuente: elaboración propia)

Tras experimentar si el filtro con mediana nos proporciona mejores resultados que sin filtro, llegamos a la conclusión que aunque elimine los símbolos, nos inserta errores en las demás letras, que en cambio la lectura sin filtros no incluye. Por este motivo y porque en nuestra base de datos solo tienen símbolos las matrículas de Croacía, se ha decidido no utilizar los filtros en nuestra aplicación final.

6.1.2. Resultados obtenidos con OCR

Después del pre-procesamiento, se utilizará el motor de reconocimiento Tesseract [57], desarrollado por Google y considerado uno de los más precisos. Tesseract consta de tres variables (idioma, modo del motor OCR y modo de segmentación) que podemos configurar según el texto que se quiera leer. En nuestro caso, el idioma no nos interesa, ya que no va a leer palabras con significado sino simplemente va a reconocer números o letras.

Tras probar algunas configuraciones que contiene Tesseract como la lectura línea por línea o carácter a carácter, se llega a la conclusión de que el mejor resultado lo proporciona reconocer las imágenes línea por línea. Para evitar que el algoritmo reconozca símbolos en vez de alguna letra se añade una lista blanca (solo reconoce los símbolos que se incluyan en ella) con el alfabeto y con los dígitos del 0 al 9.

6.2. Lectura aplicando entrenamiento profundo

Aunque el OCR es sencillo y rápido de implementar, surgen ciertos errores. Una posible mejora es la lectura aplicando entrenamiento profundo. Actualmente es la técnica más usada para el reconocimiento de matrículas y hay sistemas que ya lo tienen implementado como OpenALPR[1], capaz de realizar lecturas de matrículas de muchos países.

Los pasos a seguir para realizar el aprendizaje profundo para la lectura de las matrículas son:

- Obtener imágenes de los dígitos y letras: se necesitan fotos de cada dígito y carácter para entrenar la red. Las imágenes para realizar las pruebas han sido obtenidas del proyecto 'License-Plate-Recognition-Nigerian-Vehicles'. [2]

[1] `https://www.openalpr.com/`
[2] `https://github.com/femioladeji/License-Plate-Recognition-Nigerian-vehicles`

Figura 6.5: Ejemplo de la base de datos. (Fuente: Github)

- Entrenamiento de la red: se entrena con las imágenes de la base de datos. En este caso hemos utilizado la red entrenada en el proyecto descrito anteriormente. Esta red está pensada para matrículas nigerianas por lo que sería interesante mejorarlo en un futuro para matrículas españolas.

- Dividir la matrícula: obtenemos cada carácter de la matrícula para su posterior predicción. Para dividir la matricula buscamos los contornos, este proceso no divide bien la imagen cuando tiene algún tipo de ángulo. Una futura mejora sería realizar un prepocesamiento de la matrícula y corregir el ángulo para obtener correctamente los contornos.

- Realizar la predicción: se realiza la predicción para cada carácter y nos quedamos con el que mayor probabilidad nos de en cada caso.

Figura 6.6: Segmentación de caracteres. (Fuente: elaboración propia)

En la figura 6.6 podemos observar como en el primer vehículo se realiza bien la segmentación de los caracteres y obtenemos como predicción de lectura la matrícula `J S430 L`. Aunque la predicción no es del todo correcta (se confunde el '5' por la letra 'S'), se aproxima bastante. Pero cuando la matrícula tiene cierto ángulo surgen problemas con la segmentación y no es capaz de segmentar todos los caracteres o como en el último vehículo, que no es capaz de segmentar ningún carácter. En cambio, con Tesseract OCR la lectura es la correcta, excluyendo la placa del vehículo de la imagen central, la cuál no es capaz de leer correctamente.

Por lo tanto, llegamos a la conclusión de que con esta técnica obtenemos peores resultados que con OCR, debido a que la red estaba entrenada para matrículas nigerianas y además, la segmentación de los caracteres no es correcta en todos los casos. Por estos motivos se opta por usar OCR Tesseract.

Capítulo 7

Diseño de la interfaz

En este capítulo se expondrá la justificación de la plataforma utilizada para el desarrollo de la aplicación. Esta tiene que ser capaz de incorporar nuestra red neuronal entrenada para la detección de las matrículas.

7.1. Consideraciones previas

En el momento de buscar una solución software, llegamos al paradigma de elegir desarrollar una aplicación de escritorio o Web. Para seleccionar la mejor herramienta, vemos a continuación las ventajas y desventajas de cada uno:

Comparación	**Web**	**Aplicación escritorio**	**Aplicación móvil**
Tiempo de respuesta más rápido	✗	✓	✗
El cliente no necesita instalar nada	✓	✗	✗
Rápido de implementar	✗	✓	✗
Portabilidad	✓	✗	✓

Tras analizar las ventajas y desventajas de cada opción, nos decantamos por realizar una aplicación de escritorio. Con esta elección se busca proteger más la privacidad de datos

personales, como lo son las matrículas. El lenguaje de programación elegido es Python, ya que la implementación de YOLO es más sencilla que en otros lenguajes como Java.

Por último, elegimos una biblioteca gráfica que utilice Python. Las principales bibliotecas que existen para Python son: Tkinter, WxPython, PyQT, PyGTK. Para nuestra aplicación utilizamos Tkinter, aunque la apariencia es un poco anticuada y los elementos gráficos son limitados pero viene preinstalada con Python.

7.2. Funcionalidad de la aplicación

A continuación se explicará cómo desplegar la aplicación con unos simples pasos y se expondrá el funcionamiento básico para el reconocimiento y lectura de las matrículas en las fotos y en los vídeos.

7.2.1. Instalación

La aplicación estará disponible en el repositorio de Github indicado en este pie de página[1] y tras descargar el proyecto, accedemos a la carpeta `aplicacion`. Para poder ejecutar la aplicación son necesarios los paquetes `opencv-python` o `pytesseract` entre otros, por lo tanto, para automatizar su instalación simplemente desde la consola ejecutamos la orden:

```
pip install -r requirements.txt
```

Finalmente, después de que la instalación de los paquetes haya terminado, ejecutamos el archivo `seguimientoVehiculos.py` con el siguiente comando desde la terminal:

```
python seguimientoVehiculos.py
```

7.2.2. Manual de uso de la aplicación

La aplicación puede ser usada tanto para fotos como para vídeos, por este motivo al inicio de la aplicación nos aparecerán dos botones donde podremos elegir una de las dos

[1] Enlace a GitHub

opciones.

7.2.2.1. Procesamiento de imágenes

El proceso seguido para que realice una búsqueda en una foto es el siguiente:

- Insertar la imagen: para ello pulsamos el botón de insertar y seleccionamos una imagen donde queramos buscar las matrículas. Tras insertar la foto, esta aparece en nuestra aplicación.
- Buscar matrículas: después de insertar la imagen, se activará el botón de buscar matrículas, al pulsar este botón ejecutará la red neuronal y si encuentra alguna matrícula se recuadrará y mostrará su lectura en `Matrículas Encontradas`.

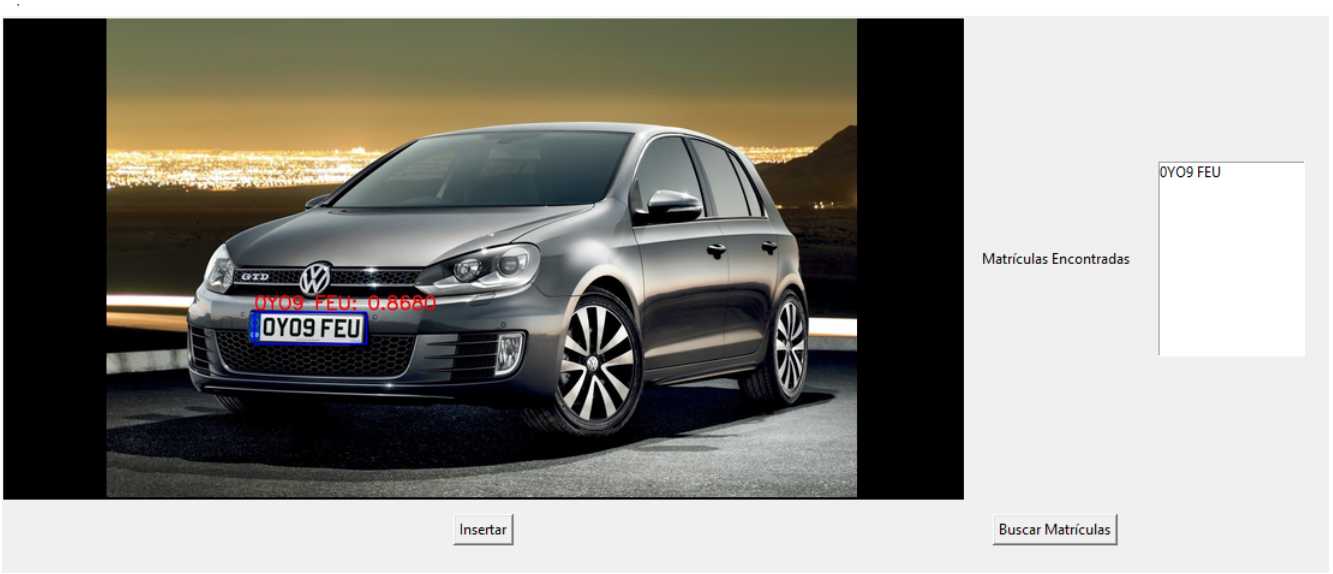

Figura 7.1: Resultado obtenido en la aplicación con una imagen. (Fuente: elaboración propia)

7.2.2.2. Procesamiento de vídeo

En el procesamiento de vídeos efectuamos la búsqueda de las matrículas igual que con una foto pero con muchas más imágenes. Cada imagen se denomina `frame` y el `frame rate` nos dice las imágenes que obtenemos de un segundo de vídeo. Por este motivo el procesamiento de vídeo es muy costoso computacionalmente puesto que se necesita ejecutar la red neuronal en una gran cantidad de ocasiones. Por lo tanto, para el procesamiento de vídeo hemos seguido los siguientes pasos:

1. Se recorre el vídeo `frame` a `frame`.
2. Después, se ejecuta la red buscando las posibles matrículas dentro de ese `frame`.
3. Se leen las matrículas que tengan una probabilidad más alta a un umbral.
4. Finalmente, se recuadran las matrículas y se guarda el `frame` en un nuevo vídeo. Se crea un nuevo vídeo porque si guardamos todos los `frames` en memoria dinámica, esta se llena rápidamente y se colapsa el sistema.

En el apartado para vídeos nos aparecen tres botones. El primero `Insertar Vídeo`, buscamos nuestro archivo de vídeo y le damos a añadir. El segundo botón, `Buscar Matrículas`, lo pulsamos para que reconozca las matrículas. Cuando este proceso termina, nos aparecerán las matrículas en pantalla y se activará el botón `Ver Vídeo`. Tras pulsarlo nos aparecerá el vídeo con las matrículas recuadradas encontradas, además este vídeo se guardará en la raíz con el nombre `video_procesado.avi`

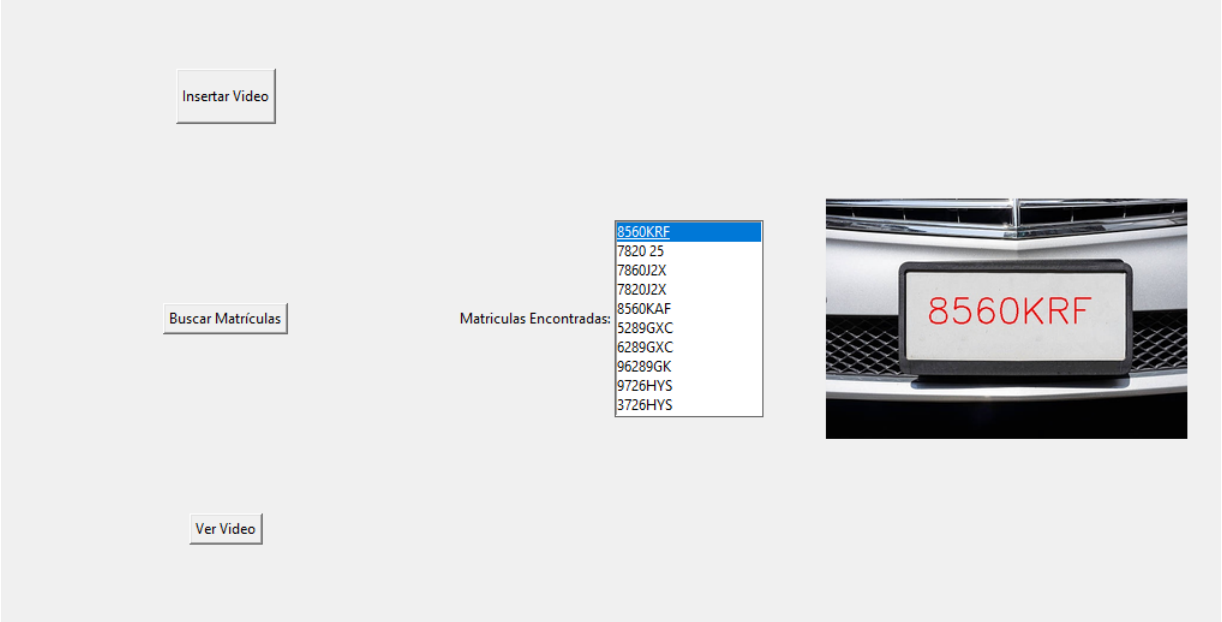

Figura 7.2: Resultado obtenido en la aplicación tras procesar un vídeo. (Fuente: elaboración propia)

Capítulo 8

Futuras mejoras y conclusiones

En este último capítulo se presentan las conclusiones extraídas del trabajo y se proponen las posibles mejoras que se pueden realizar en él.

En el proceso de entrenamiento nos hemos percatado de que tener una amplia y variada base de datos es crucial para conseguir buenos resultados. Cuando al inicio nuestra red tan solo tenía 200 fotos de matrículas nos encontramos con una red infra-ajustada que no era capaz de reconocer matrículas. Posteriormente realizando pruebas con 500 imágenes procedentes de un solo país, la red mejoraba sus resultados, pero aún así seguían siendo pobres. Solo detectaba coches de ese país y en ocasiones tampoco era capaz de ello porque la red estaba sobre-ajustada. Con esto llegamos a la conclusión de que cuanto mayor sea la cantidad y la diversidad de la base de datos, mejores serán los resultados.

Una posible mejora del sistema, por lo tanto, sería obtener más imágenes para nuestra base de datos. Al obtener más matrículas, la precisión de la red aumentaría. Si la cantidad de matrículas de cada país es suficiente, sería posible su detección categorizada por países. Esta sería capaz de encontrar las matrículas y además decirnos de qué país proceden. El sistema sería útil para tomar datos estadísticos en lugares con tránsito de personas de diferentes países de procedencia como por ejemplo en controles fronterizos.

También hemos sido conscientes de la dificultad que conlleva la lectura de las matrículas. Los resultados con el reconocimiento óptico de caracteres no son del todo malos pero algunos símbolos dan falsos caracteres. Por lo tanto, otra posible mejora del trabajo sería realizar aprendizaje profundo de los caracteres de las matrículas españolas.

Bibliografía

[1] Nathan L Ensmenger. *The computer boys take over: Computers, programmers, and the politics of technical expertise*. Mit Press, 2012.

[2] Wikipedia. *Joseph Marie Jacquard — Wikipedia, La enciclopedia libre*. [Internet; descargado 10-mayo-2020]. 2020. URL: `https://es.wikipedia.org/w/index.php?title=Joseph_Marie_Jacquard&oldid=123519491`.

[3] Manuel Zaforas. *Inteligencia Artificial como servicio: reconocimiento de imágenes*. URL: `https://www.paradigmadigital.com/techbiz/inteligencia-artificial-servicio-reconocimiento-imagenes/`.

[4] Zhong-Qiu Zhao y col. "Object detection with deep learning: A review". En: *IEEE transactions on neural networks and learning systems* (2019).

[5] Zhe Cao y col. "Realtime multi-person 2d pose estimation using part affinity fields". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, págs. 7291-7299.

[6] M Everingham y col. "The pascal visual object classes challenge 2012 (voc2012) results (2012)". En: *URL http://www. pascal-network. org/challenges/VOC/voc2011/workshop/inde html*. 2011.

[7] Yutong Ye, Liming Fu y Bijun Li. "Object detection and tracking using multi-layer laser for autonomous urban driving". En: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2016, págs. 259-264.

[8] Tony Lindeberg. "Scale invariant feature transform". En: (2012).

[9] Lilian Weng. *Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS*. URL: https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html.

[10] Rainer Lienhart y Jochen Maydt. "An extended set of haar-like features for rapid object detection". En: *Proceedings. international conference on image processing*. Vol. 1. IEEE. 2002, págs. I-I.

[11] Gustavo A Betancourt. "Las máquinas de soporte vectorial (SVMs)". En: *Scientia et technica* 1.27 (2005).

[12] Raúl E. Lopez Briega. *Boosting en Machine Learning con Python*. 2017. URL: https://relopezbriega.github.io/blog/2017/06/10/boosting-en-machine-learning-con-python/.

[13] PM Panchal, SR Panchal y SK Shah. "A comparison of SIFT and SURF". En: *International Journal of Innovative Research in Computer and Communication Engineering* 1.2 (2013), págs. 323-327.

[14] Wikipedia. *SURF — Wikipedia, La enciclopedia libre*. [Internet; descargado 4-marzo-2020]. 2020. URL: https://es.wikipedia.org/w/index.php?title=SURF&oldid=122738833.

[15] Juan Braun Pablo Flores. *Algoritmo SIFT: fundamento teorico*. URL: http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2011/keypoints/FundamentoSIFT.pdf.

[16] Sander Soo. "Object detection using Haar-cascade Classifier". En: *Institute of Computer Science, University of Tartu* (2014), págs. 1-12.

[17] Antonio Arista-Jalife y col. "Clasificación de Imágenes Urbanas Aéreas: Comparación entre Descriptores de Bajo Nivel y Aprendizaje Profundo". En: *Información tecnológica* 28.3 (2017), págs. 209-224.

[18] Damián Jorge Matich. "Redes Neuronales: Conceptos básicos y aplicaciones". En: *Universidad Tecnológica Nacional, México* (2001).

[19] Fernando Izaurieta y Carlos Saavedra. "Redes neuronales artificiales". En: *Departamento de Física, Universidad de Concepción Chile* (2000).

[20] Fernando Sancho Caparrini. *Aprendizaje por refuerzo: algoritmo Q Learning*. 2019. URL: http://www.cs.us.es/~fsancho/?e=109.

[21] Diego Calvo. *Clasificación de redes neuronales artificiales*. URL: http://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/.

[22] Universidad de Sevilla. *El perceptrón*. URL: http://bibing.us.es/proyectos/abreproy/11084/fichero/Memoria+por+cap%C3%ADtulos+%252FCap%C3%ADtulo+4.pdf+.

[23] Matt W Gardner y SR Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". En: *Atmospheric environment* 32.14-15 (1998), págs. 2627-2636.

[24] Jordi Torres. *Redes Neuronales Recurrentes*. URL: https://torres.ai/redes-neuronales-recurrentes/.

[25] Juan Cevallos Ampuero. "Redes Neuronales de Base Radial aplicadas a la mejora de la calidad". En: *Industrial data* 11.2 (2008), págs. 63-72.

[26] Na8. *¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador*. 2018. URL: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[27] Wikipedia. *Función sigmoide — Wikipedia, La enciclopedia libre*. [Internet; descargado 30-mayo-2020]. 2020. URL: https://es.wikipedia.org/w/index.php?title=Funci%C3%B3n_sigmoide&oldid=125950720.

[28] Sitio Big Data. *ReLU: Funciones de activación*. 2019. URL: https://sitiobigdata.com/2019/06/22/relu-funciones-activacion/.

[29] Diego Calvo. *Función de activación – Redes neuronales*. 2018. URL: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/.

[30] Ross Girshick y col. "Rich feature hierarchies for accurate object detection and semantic segmentation". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, págs. 580-587.

[31] Kaiming He y col. "Spatial pyramid pooling in deep convolutional networks for visual recognition". En: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), págs. 1904-1916.

[32] Ross Girshick. "Fast r-cnn". En: *Proceedings of the IEEE international conference on computer vision*. 2015, págs. 1440-1448.

[33] Joseph Redmon y col. "You only look once: Unified, real-time object detection". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788.

[34] Jifeng Dai y col. "R-fcn: Object detection via region-based fully convolutional networks". En: *Advances in neural information processing systems*. 2016, págs. 379-387.

[35] Tiba Razmi. *Mask R-CNN*. URL: `https://medium.com/@tibastar/mask-r-cnn-d69aa596761f`.

[36] Dumitru Erhan y col. "Scalable object detection using deep neural networks". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, págs. 2147-2154.

[37] Donggeun Yoo y col. "Attentionnet: Aggregating weak directions for accurate object detection". En: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, págs. 2659-2667.

[38] Mahyar Najibi, Mohammad Rastegari y Larry S Davis. "G-cnn: an iterative grid based object detector". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 2369-2377.

[39] Shaoqing Ren y col. "Faster r-cnn: Towards real-time object detection with region proposal networks". En: *Advances in neural information processing systems*. 2015, págs. 91-99.

[40] Jonathan Hui. *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3.* URL: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e208.

[41] Wei Liu y col. "Ssd: Single shot multibox detector". En: *European conference on computer vision*. Springer. 2016, págs. 21-37.

[42] Joseph Redmon y Ali Farhadi. "Yolov3: An incremental improvement". En: *arXiv preprint arXiv:1804.02767* (2018).

[43] Cheng-Yang Fu y col. "Dssd: Deconvolutional single shot detector". En: *arXiv preprint arXiv:1701.06659* (2017).

[44] Zhiqiang Shen y col. "Dsod: Learning deeply supervised object detectors from scratch". En: *Proceedings of the IEEE international conference on computer vision*. 2017, págs. 1919-1927.

[45] Jonathan Krause y col. "3D Object Representations for Fine-Grained Categorization". En: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia, 2013.

[46] Christian Gutiérrez Lancho. "Detección de armas en vídeos mediante técnicas de Deep Learning". En: (2019).

[47] Wikipedia. *Error cuadrático medio — Wikipedia, La enciclopedia libre*. [Internet; descargado 27-mayo-2020]. 2020. URL: https://es.wikipedia.org/w/index.php?title=Error_cuadr%C3%A1tico_medio&oldid=125821363.

[48] Sergey Ioffe y Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". En: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[49] Joseph Redmon y Ali Farhadi. "YOLO9000: Better, Faster, Stronger". En: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[50] Tsung-Yi Lin y col. "Microsoft COCO: Common Objects in Context". En: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[51] *ImageNet — Wikipedia , La enciclopedia libre*. [En línea; consultado el 28 de mayo de 2020]. URL: https://en.wikipedia.org/w/index.php?title=ImageNet&oldid=946020209.

[52] Ren Jie Tan. *Desglose de la precisión media promedio (mAP)*. 2019. URL: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52.

[53] Joseph Chet Redmon. *Classifying With Pre-Trained Models*. URL: https://pjreddie.com/darknet/imagenet/#darknet53.

[54] Ángela Casado García. *Guiando la creación de modelos de detección de objetos basados en deep learning*. URL: https://biblioteca.unirioja.es/tfe_e/TFE004595.pdf.

[55] Wikipedia. *Reconocimiento óptico de caracteres — Wikipedia, La enciclopedia libre*. [Internet; descargado 20-marzo-2020]. 2020. URL: https://es.wikipedia.org/w/index.php?title=Reconocimiento_%C3%B3ptico_de_caracteres&oldid=123367680.

[56] Wikipedia. *Método del valor umbral — Wikipedia, La enciclopedia libre*. [Internet; descargado 20-marzo-2020]. 2019. URL: https://es.wikipedia.org/w/index.php?title=M%C3%A9todo_del_valor_umbral&oldid=120799493.

[57] Wikipedia. *Tesseract OCR — Wikipedia, La enciclopedia libre*. [Internet; descargado 20-marzo-2020]. 2019. URL: https://es.wikipedia.org/w/index.php?title=Tesseract_OCR&oldid=122240378.

Apéndice A

Anexo I. Pliego de condiciones

En este anexo vamos a indicar el material utilizado para llevar a cabo el desarrollo del trabajo.

Hardware utilizado:

El ordenador con el que se han realizado las pruebas ha sido:

- Intel Core i5 7400
- NVIDIA GTX 1050 Ti
- 8 GB de RAM
- 2 TB HDD de almacenamiento

Software Utilizado:

- CUDA Toolkit 10.1 Update 2
- CuDNN 7.6.4
- Python 3.7.4
- Latex, para el informe

- Keras 1.1.0
- Tensorflow 2.2.0
- Tesseract 0.3.4
- Scipy 1.4.1
- OpenCV 4.2.0.34

Apéndice B

Anexo II. Instalación del entorno local

En este anexo presentamos los programas e instrucciones necesarios para que el usuario pueda recrear el entorno de trabajo.

Los elementos que vamos a abordar son los siguientes:

1. Actualización de los drivers
2. Descarga de CUDA

Si disponemos de una GPU que sea superior a la capacidad de cómputo de nivel 3 procedemos a la descarga de CUDA y cuDNN, ya que nos permite la aceleración de los procesos al usar las redes neuronales.

La capacidad de cómputo de nuestra gráfica la podemos ver en la página de NVIDIA, concretamente en clasificación de GPUs[1].

Si el equipo de ejecución no dispone de GPU, no hay problema, no es necesario descargar CUDA y cuDNN. El inconveniente es que la CPU se va a encargar de realizar todos los procesos necesarios y por lo tanto el tiempo de ejecución será mayor.

[1] `https://developer.nvidia.com/cuda-gpus`

1. Actualizar los drivers

Desde NVIDIA elegimos la gráfica que tiene nuestro ordenador y nos descargamos e instalamos la última versión.

Figura B.1: Actualizar controladores

2. Descargamos CUDA

Descargamos la versión CUDA Toolkit 10.1 desde la página oficial de Nvidia[2] y seguimos las instrucciones que nos aparecen debajo.

[2] `https://developer.nvidia.com/cuda-downloads?`

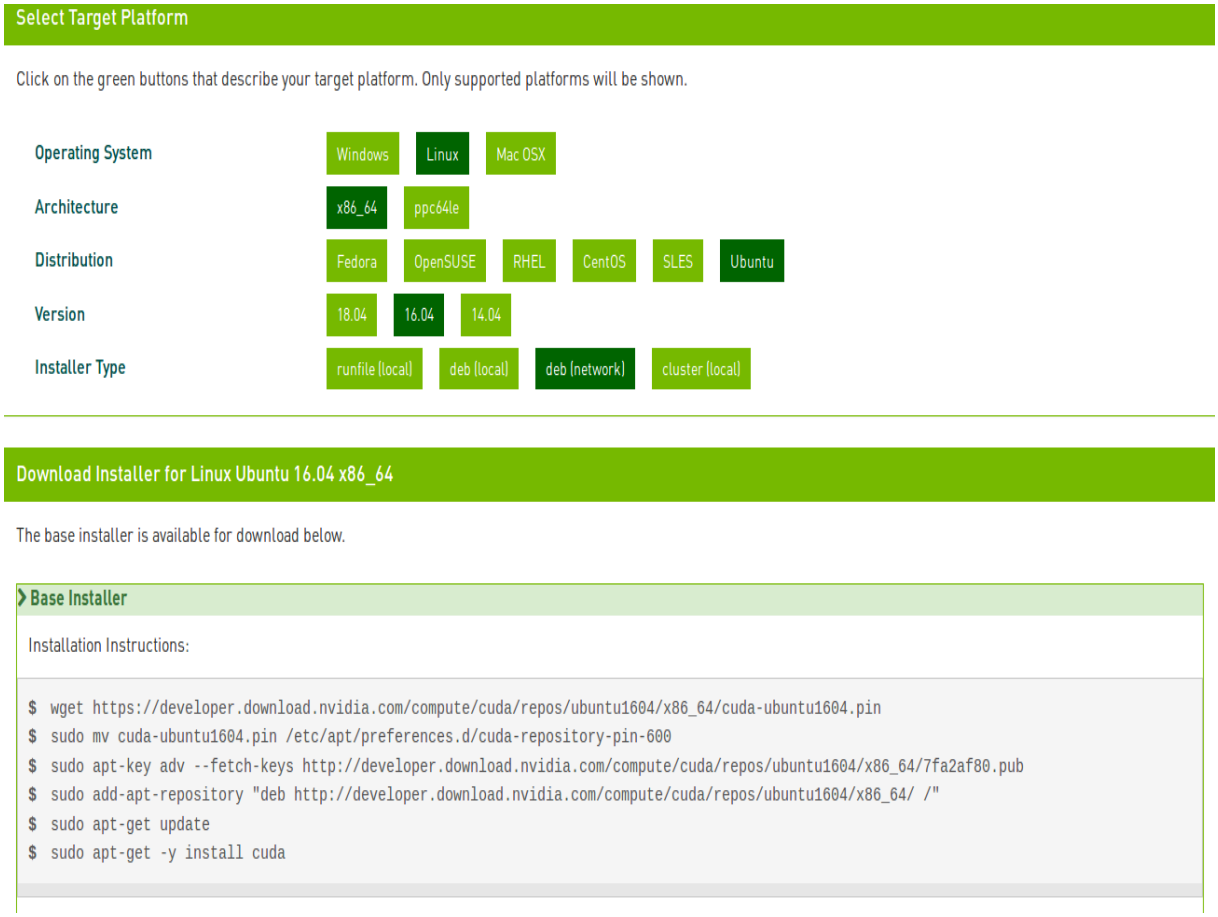

Figura B.2: Descarga CUDA

Para descargarnos cuDNN lo podemos realizar desde la página oficial de Nvidia[3]. Tenemos que acceder o crear una cuenta developer. A continuación respondemos a la encuesta y procedemos a descargar el archivo para nuestro sistema operativo y según la versión de CUDA que tengamos. En nuestro caso al tener la versión 10.1, descargamos la versión v7.6.4 de cuDNN.

Por último para terminar la instalación de cuDNN tenemos que meter los archivos que se nos han descargado en el archivo comprimido en las correspondientes carpetas y concederle permisos.

[3]`https://developer.nvidia.com/rdp/cudnn-download`

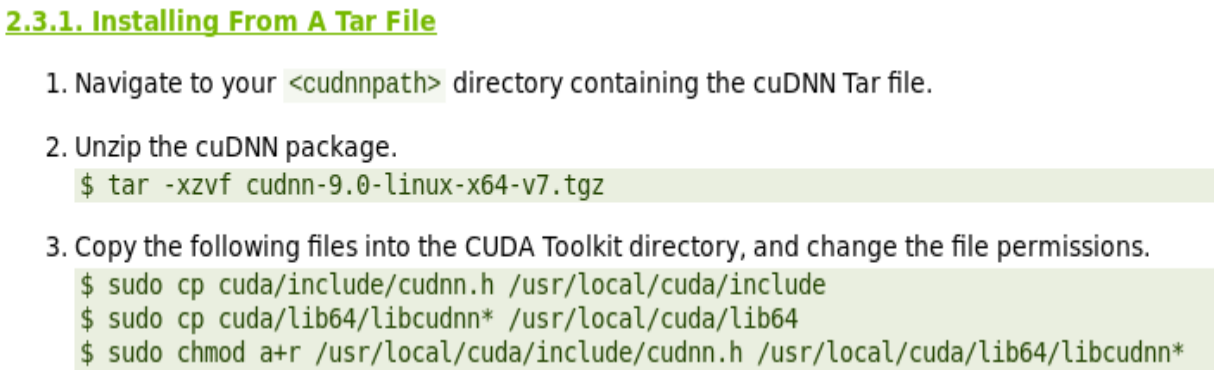

Figura B.3: Instalación de cuDNN

Para instalar Keras y TensorFlow ejecutamos las órdenes `pip install keras` y `pip install tensorflow` respectivamente.

Apéndice C

Anexo III. Entrenamiento de forma remota

En este anexo se presenta la herramienta Google Colab, que permite la ejecución en la nube de código Python, así como instrucciones para ejecutar en ella el entrenamiento de la red neuronal.

Para poder utilizar Google Colab se necesita una cuenta de Google Drive. Dentro de la nube nos vamos a la carpeta donde queramos crear nuestro notebook. A continuación pulsamos en **Nuevo >Más >Google Colaboratory**.

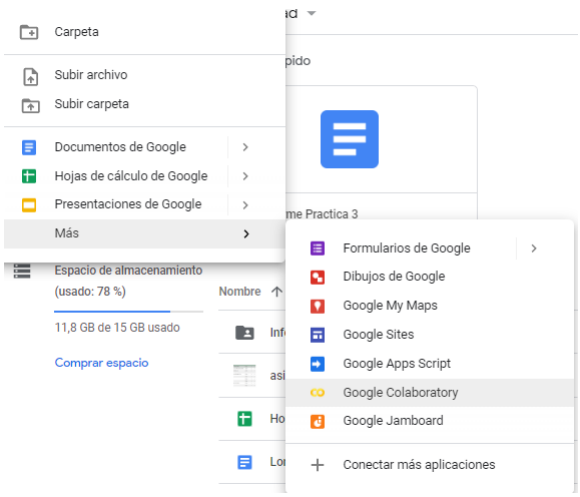

Figura C.1: Crear archivo Google Colaboratory. (Origen: Google Drive)

Podemos activar la aceleración de la GPU o CPU para los módulos que lo permitan. Para ello, simplemente pulsamos en **Entorno de ejecución >Cambiar tipo de entorno de ejecución**. En nuestro caso queremos que la aceleración sea de la GPU.

Posteriormente cargamos nuestra base de datos, guardada en Google Drive. Primero, autentificamos que queremos dar el permiso para acceder con esta orden:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Luego navegamos por las carpetas con la orden 'cd' hasta que lleguemos a la carpeta deseada. Después se instalan las bibliotecas o paquetes que necesitamos. CUDA ya viene pre-instalado por lo tanto no sería necesario volver a instalarlo. Finalmente le damos permisos de ejecución a la carpeta deseada de esta manera:

```
!chmod +x ./darknet
```

Y compilamos con la orden: '!make'. Tras realizar estos pasos ya podríamos entrenar de forma remota nuestra red.

```
!./darknet detector train "/obj.data" "/yolov3.cfg" "/darknet53.conv
   .74" -dont_show
```